8085 Microprocessor Laboratory

Third Year

Dr. Hussein Ali Abdulnabi

Dr. Ammar Ali Sohrab

Lec. Gregor Alexander Armice

Experiment 1

An Introduction to (8085) Microprocessor Principles with Simple Programs

Object:

To introduce the basic principles of microprocessor systems and to explain the fundamental structure of a typical microcomputer system and to be familiar with the MAT385 monitor facilities for loading & executing a program.

Theory:

The word Microprocessor (μP) in general means that part of the microcomputer which is responsible for the main processes such as Arithmetic processes like (Adding, Subtracting, ….etc), logical processes like (ANDing, ORing, …etc) and others processes such as storing temporarily data.

The 8085 is an 8-bit microprocessor, it means that it deals with 8-bits bytes (i.e. every data byte contains 8-bits).

The address which contains 16-bit is used to identify the data source or the destination or both for the transmission of data.

Basic Computer Architecture:

A digital computer executes a list of basic binary machine instructions (a program) which have been selected and ordered by the user to solve his particular task; the program is stored within the computer. A basic digital computer comprises a memory which is primarily used to hold or store the program, a processor (often referred to as the central processor unit or CPU) which executes the individual machine instructions, and some input and output (I/O) ports. These ports form the interface between the computer and the source of the input data and the subsequent output data when the processor is a single integrated circuit it is called a microprocessor. The complete combination of microprocessor, memory and input and output ports is referred to as microcomputer.

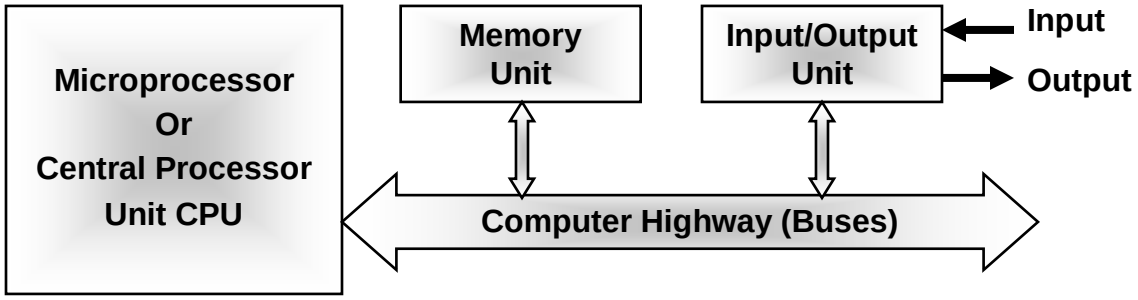


Figure 1-1 Microcomputer Block Diagram

The memory consists of a number of locations each individually identified by a number called it's address. Each location contains a binary pattern. The binary pattern stored at an address is referred to as the contents of that address.

In a microprocessor system the memory usually comprises two types, Random Access Memory (RAM) which is a volatile memory and Read Only Memory (ROM) which is a non volatile memory. Basically RAM has the capability of having information both written into and read out of each location and is often

used for storing intermediate results (data) during a computation. ROM has information fixed into it during its manufacture and consequently can only be operated a read only mode.

The computer highway consists of three separate buses. This is shown in the figure below.

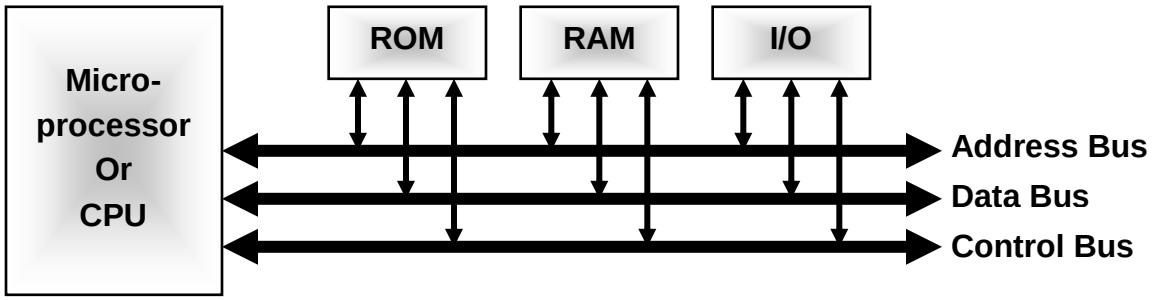


Figure 1-2 Microcomputer Highway Buses

The data bus is used to carry the data associated with a memory or input/output transfer. The data bus in many computers is bidirectional. Hence data can be transferred from the processor to a device or from a device to the processor over a single set of data lines.

The address bus is used to specify the memory location or input/output port involved in a transfer. The address bus consists of 16 lines on which a binary coded address can be presented to a memory or input/output ports. The range of possible memory addresses are from 0000_H to $FFFF_H$.

The control bus is made up of lines carrying various control signals generated by the microprocessor and other system components to synchronize transfer. The memory map for the MAT385 are listed on it that the student should be noted, and write in a paper the range of memory that could use for the main or sub program to help him through the course lab experiments progress.

Addressing mode

The way of specifying data to be operated by an instruction is called addressing mode. In 8085 microprocessor there are 5 types of addressing modes

1. **Immediate addressing mode**

In this mode, the 8/16-bit data is specified in the instruction itself as one of its operand. **For example:** MVI K, 20F: means 20F is copied into register K.

2. **Register addressing mode**

In this mode, the data is copied from one register to another. **For example:**MOV K, B: means data in register B is copied to register K.

3. **Direct addressing mode**

In this mode, the data is directly copied from the given address to the register. **For example:** LDB 5000K: means the data at address 5000K is

copied to register B.

4. Indirect addressing mode

In this mode, the data is transferred from one register to another by using the address pointed by the register. **For example:** MOV K, B: means data is transferred from the memory address pointed by the register to the register K.

5. Implied addressing mode

This mode doesn't require any operand; the data is specified by the opcode itself. **For example:** CMP.

The 8085 instruction set:

The 8085 instruction set includes five different types of instructions.

1. Data Transfer Group:
Move data between registers or between memory and registers.
2. Arithmetic Group:
Add, subtract, increment, or decrement data in registers or in memory.
3. Logical Group:
AND, OR, EXCLUSIVE-OR, Compare, Rotate or complement data in registers or in memory.
4. Branch Group:
Conditional and Unconditional jump instructions subroutine call instructions and return instructions.
5. Stacks I/O and Machine Control Group:
This group includes I/O instructions, as well as instructions for maintaining the stack memory and internal control flags.

The 8085 can operate either on the internal CPU registers (A, B, C, D, E, F, H & L) or on the system memory (RAM or ROM).

1. Data Transfer Instructions:

The basic machine instructions are those from the data transfer group these are generally referred to as the move or load instructions and the main instructions are as follows:

Instruction	Description
MOV B,A	Move the data between the internal processor reg., is expressed (B) ← (A)
MVI A,8-bit data	Move immediate 8-bit data value to transfer to accumulator (A) ← 8-bit
XCHG	Exchange the contents of reg. pair HL & DE with (DE) ↔ (HL)

LXI	reg. pair, 16-bit (BC, DE or HL)	The reg. pair being loaded with immediate 16-bit data or address.
LDA	16-bits memory location (address)	This results in the A reg. being loaded with the contents of memory location. e.g. LDA 28EA (A) ← (28EA)
STA	16-bits memory location (address)	Similar the contents of the A reg. may be stored in a specified memory location. e.g. STA 28F2 (28F2) ← (A)
LHLD	16-bit memory address	Load the reg. pair H&L a 16-bit memory address using an direct addressing, e.g. LHLD $28A2_H$, i.e. (L) ← (28A2) & (H) ← (28A3)
SHLD	16-bit memory Address	Similar, store the reg. Pair H&L a 16-bit memory address using an direct addressing, e.g. SHLD $28A2_H$, i.e. (28A2) ← (L) & (28A3) ← (H)
SPHL		The content of registers pair HL copied into stack pointer SP

Procedure:

1- write a program, which exchange of the contents of (B reg.) to the contents of the accumulator (A reg.)

2- Implement the following programs

a-
```
LDA   2850H      : Get a byte from monitor ROM
MOV   D, A       : Store it in register D
STA   2880H      : AND 2880H in RAM
HLT              : Stop
```

b-
```
LHLD  2850H      : Load H and L from monitor ROM
SHLD  2870H      : StoreH and L in RAM
HLT              : Stop
```

c-
```
LXI   H, 1234H   : Put 12H into register H, & 34H into register L
LXI   D, 5678H   : Put 56H into register D, & 78H into register E
XCHG             : Swap DE with HL
HLT              : Stop
MOV   M,A        : Store the MS sum byte
HLT              : Stop
```

Note: LS=Least significant & MS=Most Significant

3- write 3 different program segments to store $(05)_H$ in memory location $(2850)_{Hri}$

Discussion:

1. What is the difference between (ROM) and (RAM) memory types?
2. How do the data retention characteristics of RAM and ROM storage devices differ?
3. What is the purpose and use of the Program Counter (PC), Stack Pointer (SP), and Memory Address Register (MAR) registers?
4. How many bytes of RAM are available to the MAT-8085 user? What range of addresses is occupied by this RAM?
5. Write an 8085 program to store $4E_H$ in memory location 2840_H and store $3E_H$ in memory location 2841_H, exchange them and store the result in memory locations 2842_H and 2843_H respectively. Solve this problem using three different methods.
6. Wrie a program to exchange the content of memory locations [2801][2800] with [2803][2802]

Experiment 2
Binary Arithmetic

Object

The instruction set of the 8085 microprocessor includes instructions to add and subtract 8-bits binary quantities. Arithmetic operations are performed on multibyte quantities by cascading these simple 8-bits operations.

. Add or Subtract 8-bits binary quantities.
. Add or Subtract multibyte binary quantities.
. Inspect and interpret the contents of the flag register.

Theory

The majority of machine instructions available with a microprocessor operates on or affects the state of various internal registers which make up the μP. The main registers for the Intel 8085 are as follows:

A	F
B	C
D	E
H	L
F	IM
Stack Pointer SP	
Program Counter PC	

The letters have the following meanings:

Register	Meaning
A	: 8-bit accumulator.
B, C, D, E, H, L	: Four 8-bit general purpose registers.
F	: 8-bit flags register (modified by ALU operations).
IM	: 8-bit interrupt control register.
HL	: Two 8-bit registers which are normally used to form a 16-bit memory pointer.
SP	: Stack pointer a 16-bit memory address which always points to the top of a system stack.
PC	: Program counter a 16-bit memory address which always points to the next sequential instruction to be executed.

8085 Arithmetic Instructions:

The main 8085 arithmetic instructions are as follows:

Instruction		Description
ADD	B	Add contents of reg. B to accumulator. (All flags affected)
ACI	8-bit data	Add immediate with carry 8-bit data to accumulator. (All flags affected)
ADI	8-bit data	Add immediate 8-bit data value to accumulator. (All flags affected)
ADC	B	Add contents of reg. B with carry to accumulator. (All flags affected)
ADD	M	Add contents of the memory location whose address is contained in the H&L regs. to the current contents of A reg. (All flags affected)

ADC	M	Add contents of the memory location whose address is contained in the H&L regs. With carry to the current contents of A reg. (All flags affected
DAD	B	Add BC reg. pair contents to reg. pair HL. (carry only affected)
SUB	B	Subtract contents of reg. B from accumulator. (All flags affected) Add contents of the memory location whose address is contained in the H&L regs. to the current contents of A reg. (All flags affected)
SBI	8-bit data	Subtract immediate with borrow 8-bit data from accumulator. (All flags affected)
SUI	8-bit data	Subtract immediate 8-bit data from accumulator. (All flags affected)
SUB	M	Subtract contents of the memory location whose address is contained in the H&L regs. to the current contents of A reg (All flags affected)
SBB	M	Subtract contents of the memory location whose address is contained in the H&L regs. with carry to the current contents of A reg.
INR	reg,	Increment the contents of reg. by unity. All flags except the carry flag are affected.
INX	H or D	Increment the combined contents of a pair of registers (i.e. increment regs. Pair H&L by unity). (H)(L) ← (H)(L)+1 (Nol flags affected)
INR	M	Increment the memory location whose address is contained in H&L regs. [(H)(L)] ← [(H)(L)+1] All flags except the carry flag are affected.
DCR	reg.	These instructions are identical to the increment but at it decrement by unity. All flags except the carry flag are affected.
DCX	H or D	Decrement the combined contents of a pair of registers (i.e. increment regs. Pair H&L by unity). (H)(L) ← (H)(L)-1 (Nol flags affected)
DCR	M	decrement the memory location whose address is contained in H&L regs. [(H)(L)] ← [(H)(L)-1] All flags except the carry flag are affected.

Each of these instructions affects one or more flags in the processor status reg. The carry (CY) and zero (Z) flags are of particular interest in arithmetic processing. Figure (2-1) illustrates the position of each flags in the F reg. of the MAT385 using the Examine Register function.

Multibyte Addition and Subtraction:

Addition of multibyte quantities must be performed one byte at a time using the 8-bit addition instructions, as an example, the following program adds the 16-bits contents of register pairs BC and DE and places the sum in register pair HL.

```
MOV  A,C    : Get least significant byte of first operand.
ADD  E      : Add it to LS byte of second operand.
MOV  L,A    : Then store the LS sum byte.
MOV  A,B    : Get MS byte of first operand.
ADC  D      : Add it with CY to MS byte of second operand.
MOV  H,A    : Then store the MS sum byte.
HLT         : Stop.
```

This example program could be expanded to handle operands containing three or more bytes. When doing multibyte addition with operands larger than two bytes, add the two least significant bytes first with the ADD instruction. Add all more significant pairs of bytes using ADC instructions.

The individual flag bits are grouped together to form the flag register “F” and for the Intel 8085 it is made up as follows:

MS LS

S	Z	x	AC	x	P	x	CY
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

Figure (2-1) Position of Each Flag in the Flag Register.

Flag Bit	**Description**
S = Sign	This flag is intended for use when signed numbers are being used, it is set when the result of an arithmetic operation is negative (i.e. the MS bit of the A reg. is 1, otherwise it is cleared.
Z = Zero	This flag is set if the result of an arithmetic operation in the A reg. is zero, otherwise it is cleared.
AC = Auxiliary Carry	This flag is intended for use when BCD number representation is being used, it is set when the result of an arithmetic operation produces a carry out from the fourth bit of the A reg., and cleared otherwise.
P = Parity	This is used with the logical operations. The flag is set if the result of a logical operation (AND, OR, XOR) produces an even number of 1's and cleared otherwise.
CY = Carry	This flag is the carry out from the MS bit of the A reg.

Program Debugging:

The MAT385 provides a debugging feature that allows the user to execute a program one instruction at a time. This technique, known as single stepping, it is accomplished on the MAT 385 using this procedure:-

1. Press the SINGLE STEP key on the MAT385.
2. Enter the 4-digits address of the first program instruction.
3. Press the NEXT key to execute the first instruction. The MAT385 display shows the program counter contents and the data contained at that address. You may press the NEXT key repeatedly to a SINGLE-STEP through several instructions.
4. Press EXEC key to terminate the SINGLE-STEP mode. You may now examine or modify the contents of memory and registers.
5. You may resume single-stepping the program by pressing the SINGLE-STEP key again, followed by the next key. You do not need to enter an address before resuming; the PC is still pointing to the next instruction in the program.
6. Instead of resuming program execution in the SINGLE-STEP mode, you may choose to EXEC the remaining instructions at full speed. Do this by pressing the GO key followed by the EXEC key.

Procedure:

1. Write an 8085 program to perform the following tasks:

(B) ← 87_H
(A) ← (2B)
(C) ← (3D)
(D) ← $2F_H$
(E) ← (D)
(H) ← $3E_H$
(L) ← (H)

Code your program then list it on a form and load it into memory starting at address 2800_H. Use the "SUBSTITUTE MEMORY" command and finally SINGLE-STEP through it to verify correct operation of each instruction.

2-Write a program to find (6F2E3D +2B1A6C). Store the result in BDE registers

3-Write a program to find (6F2E3D - 2B1A6C). Store the result in BDE registers

4- write a program to add 3 bytes number stored in memory locations [2802][2801][2800] to the 3 bytes number stored in memory locations [2805][2804][2803] store the results in [2808][2807][2806]

5- write a program to subtract 3 bytes number stored in memory locations [2802][2801][2800] from 3 bytes number stored in memory locations [2805][2804][2803] store the results in [2808][2807][2806]

6. Write an 8085 program to subtract the contents of memory location 2850_H from the contents of memory location 2851_H, and store the difference in location 2852_H. Try your program for each of the following cases; record the difference and the status of the CY and Z flags for each case.

a. $(2850)_H=47_H$, $(2851)_H=8E_H$

b. $(2850)_H=8E_H$, $(2851)_H=47_H$

c. $(2850)_H=F7_H$, $(2851)_H=F7_H$

Code your program, then list it on a coding form, then examine the contents of the appropriate processor register memory location to verify correct execution of each program instruction.

6. Write an 8085 program to perform the following multi byte addition operation: Store the result in memory [2802][2801][2800]

$$\begin{array}{rr} & 91986F_H \\ + & 2E6794_H \\ \hline \end{array}$$

Test the program by manually placing the two 3-byte numbers into consecutive memory locations. Design your program to add the values together and store the sum in three other memory locations. Record the sum and the CY flag and Z flag values after the program adds each pair of bytes verify these observations with manual computations.

Discussion:

1. The following program is supposed to add the contents of memory location 2050_H to the Accumulator. The program works correctly only part of the time. Show that it is wrong to execute the program and how to correct it.

```
MOV   B,A
LDA   2050H
ADC   B
HLT
```

2. In a multibyte addition program, when you must use the ADD instruction? When you must use the ADC instruction?

3. Assume that regs. pair HL contains the value 1234_H, what does it contain after executing the instruction DAD H?

4. Write a program to add the following 32-bit numbers. Use the DAD, LHLD and SHLD instructions.(first store the first number in [2800]-[2803] and the second number [2804]-[2807] store the result in memory [2808] –[280B]

$$\begin{array}{rr} & 1F2CD43A_H \\ + & B724BD6C_H \\ \hline \end{array}$$

6. Write an 8085 program to perform the following subtraction. Store the result in memory [2802][2801][2800]

$$\begin{array}{rr} & 910B6F_H \\ - & 2E3714_H \\ \hline \end{array}$$

Experiment 3
Bit Manipulation

Object:

Bit manipulation techniques are used in a variety of microprocessor applications. Bit manipulation tasks include setting, resetting and complementing individual bits of information and also shifting and rotating groups of bits, you will be able to:

. Use the 8085 logical instructions to set, clear, or complement individual bits.
. Use the 8085 rotate instructions to manipulate groups of bits.
. Multiply two binary values using a shift-and-add technique.

Theory:

The data within the microcomputer is always representing a numerical value. In many applications, however, the data may simply be indicating the state of, say, a controlled state of a controlled system. For example, a single binary bit may indicate the state of a control value: 0 = value open, 1= value closed. Thus the 8-bit binary value 01100111 may mean control values 1,2,3,6 and 7 are closed whilst control values 4, 5 and 8 are open. In addition to the arithmetic instructions already available a microprocessor has a number of data manipulation instructions which are primarily included to manipulate non-numeric data of this kind.

8085 Bit Manipulation Instructions:

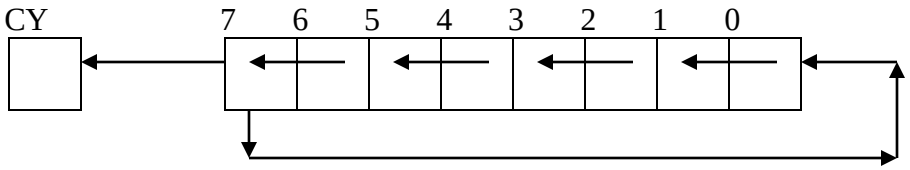
The logical instruction that can be used for bit manipulation:

<table>
<tr><td colspan="2">Instruction</td><td>Description</td></tr>
<tr><td>ANA</td><td>r</td><td>And contents of register r with accumulator</td></tr>
<tr><td>ANA</td><td>M</td><td>And contents of memory with accumulator</td></tr>
<tr><td>ANI</td><td>8-bit data</td><td>And immediate data with accumulator</td></tr>
<tr><td>ORA</td><td>r</td><td>OR contents of reg. r with accumulator</td></tr>
<tr><td>ORA</td><td>M</td><td>OR contents of memory with accumulator</td></tr>
<tr><td>ORI</td><td>8-bit data</td><td>OR immediate data with accumulator</td></tr>
<tr><td>XRA</td><td>r</td><td>Exclusive-OR contents of reg. r with accumulator</td></tr>
<tr><td>XRA</td><td>M</td><td>Exclusive-OR contents of memory with accumulator</td></tr>
<tr><td>XRI</td><td>8-bit data</td><td>Exclusive-OR immediate data with accumulator</td></tr>
<tr><td>CMA</td><td></td><td>Complement accumulator contents</td></tr>
<tr><td>CMC</td><td></td><td>Complement carry</td></tr>
<tr><td>STC</td><td></td><td>Set carry</td></tr>
<tr><td>CMP</td><td>r</td><td>Compare contents of reg. r with accumulator the flags are
<table>
<tr><td></td><td>CY</td><td>Z</td><td>S</td></tr>
<tr><td>1. A > r</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>2. A=r</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>3. A< r</td><td>1</td><td>0</td><td>1</td></tr>
</table>
affected.</td></tr>
<tr><td>CMP</td><td>M</td><td>Compare contents of memory location with accumulator,
<table>
<tr><td></td><td>CY</td><td>Z</td><td>S</td></tr>
<tr><td>1. A > M</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>2. A=M</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>3. A< M</td><td>1</td><td>0</td><td>1</td></tr>
</table></td></tr>
</table>

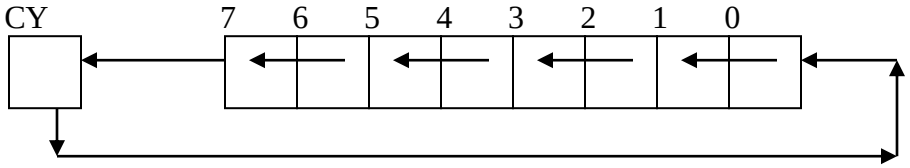
CPI	8-bit data	Compare immediate with accumulator:
		If (A) < data , CY=1, Z=0
		If (A) = data , CY=0, Z=1
		If (A) > data , CY=0, Z=0

The rotate instructions are often used to test the status of an individual bit. This is useful, for example, when performing binary multiplication and division a left shift (rotate) is a *2 operation (multiplication) and a right shift is a /2 operation (division). This is done by rotating the bit into the CY flag, then using the JC or JNC instruction to jump, for example:-

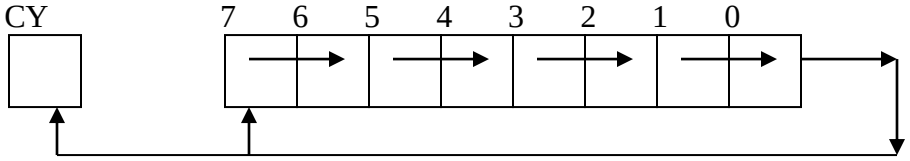
RLC Rotate Accumulator left with carry



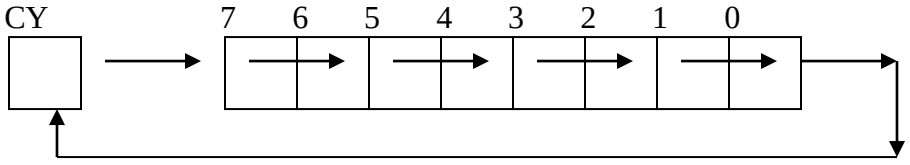
RAL Rotate All Accumulator left through carry



RRC Rotate All Accumulator right with carry



RAR Rotate All Accumulator right through carry



Procedure:

1. Write a program to compare between content of registers A and B, show the content flags CY, Z, S for the following cases
   a. A=05, B=04
   b. A=05 ,B= 05
   c. A=05, B= 06

2- Write an 8085 program to compare the contents of two regs. A and B (load immediate data into two regs.), and then rotate the contents of reg. A, the new contents of A are then AND-ed with a constant and finally the resulting contents of A are OR-ed with the contents of B. Code your program then list it on a coding form and load it into memory starting at address 2800. Check the contents of the A reg. to verify its correct operation.

Note: the data is F0, 0F

3- write an 8085 program to divide the contents of memory location 2840 into two 4-bit sections and store them in memory location 2841 and 2842; place the four most significant bits of memory location 2840 in the four least significant bits positions of memory location 2841; place the four least significant bits of memory location 2840 in the four least significant bits position of memory location 2842.

Note: 2840=3F, Result=2841=03; 2842=0F

4- Write an 8085 program to place the four most significant bits of the contents of memory location 2850 into memory location 2851, clear the four least significant bits of memory location 2851.

Note: 2850=C4, Result=2851=C0

5- Write an 8085 program that will manipulate the bits in the accumulator as follows: if bit 7 is logic 1, complement the remaining seven bits. If bit 7 is low, clear bits 0and 1, set bit 2, and complement bits 3-6. test your program for each of these values:

80 h, 00 h, 55 h and AA h

Discussion:

1- The 8085 instruction set does not include a clear accumulator instruction. Which single-byte logical instruction can perform this task?

2- The 8085 instruction set does not include a clear carry instruction. Which single-byte logical instruction can clear the carry without modifying the contents of the accumulator?

3- Write a program to set b0,b1, reset b2,b3 and complement b6,b7 for the content of memory location [2800], store the result in [2801]. Note ([2800]= (6C)]

4- What value will remain in the accumulator after executing each of the following program segments? What will be the status of the CY and Z flags?

a-

```
MVI  A,0F
MVI  B,55
XRA  B
```

b-

```
MVI  A,AA
ANI  0F
ORI  B0
```

c-

```
MVI  A,96
XRI  FF
```

d-

```
MVI  B,12
MVI  C,34
MOV  A,C
ORA  A
RLC
MOV  C,A
MOV  A,B
RAL
MOV  B,A
```

Experiment 4
Branching & Decision Masking

Object:

To see how the normal sequential execution of the machine instructions can be broken and to further investigate the conditional and unconditional branch instructions.

Theory:

The microprocessor can read and interpret thousands of pieces of information every second. This high-speed decision masking capability is the key element in many microprocessor applications.

The program loop is the basic structures which force the CPU to repeat a sequence of instructions.

Loop has four sections:

1- The initialization section which establishes the storing values of counters, address, registers (pointers) and other-variables.
2- The processing section where the actual data manipulation occurs. This is the section which dose the work.
3- The loop control section which updates counters and pointers for the next iteration.
4- The concluding section which analyzes and store the results.

Note that the computer performs sections 1 and 4 only once, while it may perform sections 2 and 3 many times. Thus the execution time of the loop will be mainly dependent on the execution time of sections 2 and 3.

A typical program loop can be flowchart as shown in figure 4-1 or the positions of the processing and loop control sections may be reversed as shown in figure 4-2.

The processing section in figure 4-1 is always executed at least once, while the processing section in figure 4-2 may not be executed at all.

Figure 4-1 seems more natural, but figure 4-2 is more efficient and avoids the problem of what to do when there is no data.

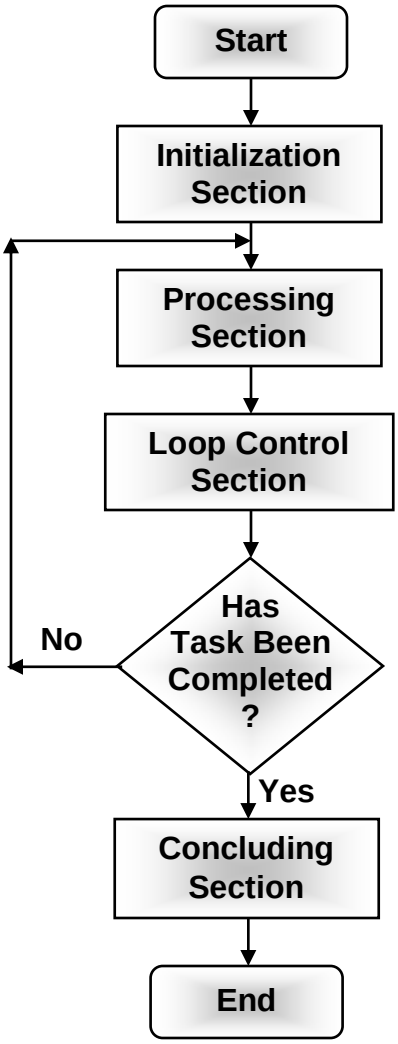


Figure 4-1 Flowchart of a Program Loop

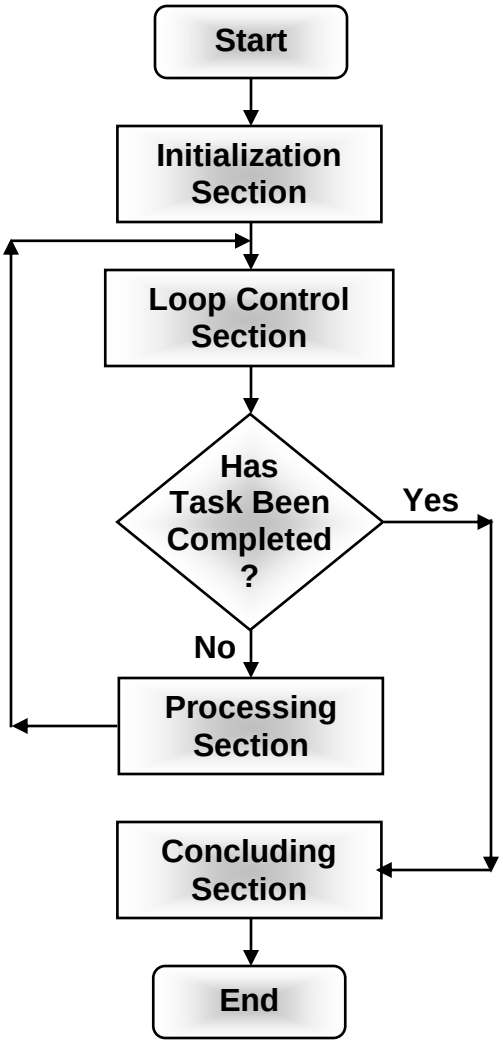


Figure 4-2 Flowchart of a Program Loop which Allows Zero Iterations

Simple Looping Structures:

Program loops are designed to repeat a group of instructions. In addition to the body of the loop (the instructions that are repeated), each loop includes a loop counter to control the number of repetitions and a flag to indicate when to stop the looping process.

The following program implements a simple repeated-addition multiplication technique to multiply two 8-bit numbers. The flowchart for this program is given in figure 4-3. In this algorithm, the multiplier (register B) serves as the loop counter, the multiplicand resides in register C, and the product is contained in register pair HL.

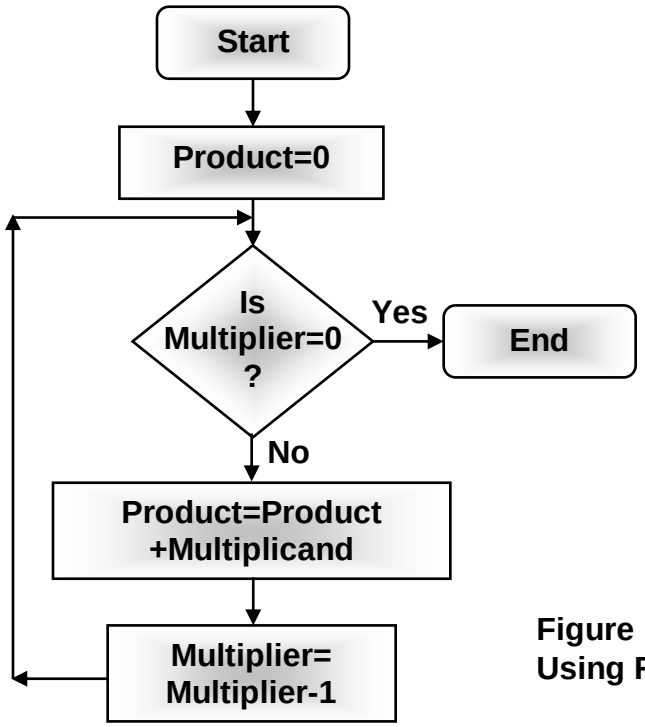


Figure 4-3 Flowchart for Multiplication Using Repeated Addition

8085 Decision Instruction:

Instruction		Description	
OP-Code	Address	Condition	Flag Status
JMP	16-bit address	Jump to specified address.	
JC	16-bit address	Jump on carry.	CY=1
JNC	16-bit address	Jump on no carry.	CY=0
JZ	16-bit address	Jump on zero.	Z=1
JNZ	16-bit address	Jump on no zero.	Z=0
JPO	16-bit address	Jump on Odd Parity.	P=0
JPE	16-bit address	Jump on Even Parity.	P=1
JP	16-bit address	Jump on Plus.	S=0
JM	16-bit address	Jump on Minus.	S=1
NOP		No operation instruction used to provide some time delay depends on its T-state value with clock.	
RST	1	This instruction to stooping the main program & returning control to the monitor program of machine and the monitor program would write “-8085”.	

All conditional jump instructions result in the status of a processor flag being examined to see if the branch is to be executed.

Bit Masking:

Bit masking involves isolating one or more bits in a binary quantity while hiding, or masking, the unwanted bits. Masking is usually done with the logical AND instructions. In this program, if bit 5 of location $(2050)_H$ is 1 the program jumps to location $(2020)_H$, if bit 5 is 0, the program jumps to location $(2040)_H$.

```
        LDA   2050H      : Retrieve byte to be tested.
        ANI   20H        : Mask all bits, except bit 5.
        JNZ   2020H      : Jump to location (2020)H if bit 5 is 1.
        JMP   2040H      : Else, go to location (2040)H.
        LXI   H,0000H    : Clear product register pair.
LOOP:   MOV   A,B        : Get multiplier.
        CPI   00H        : Compare multiplier with zero.
        JNZ   OVER       : If not zero, continue.
        RST   1          : Else, quit (return to monitor).
OVER:   MOV   A,L        : Get LS (least significant) byte of product.
        ADD   C          : Add multiplicand.
        MOV   L,A        : Save the LS product byte.
        MOV   A,H        : Add carry AND MS (most significant) product byte.
        ACI   00H
        MOV   H,A        : Save MS product byte.
        DCR   B          : Decrement multiplier.
        JMP   LOOP       : Repeat.
```

Procedure:

1- Write an 8085 program that will perform the tasks described by the flowchart in figure 4-4. Document your program by including appropriate comments in the program listing. Coding your program and execute it.

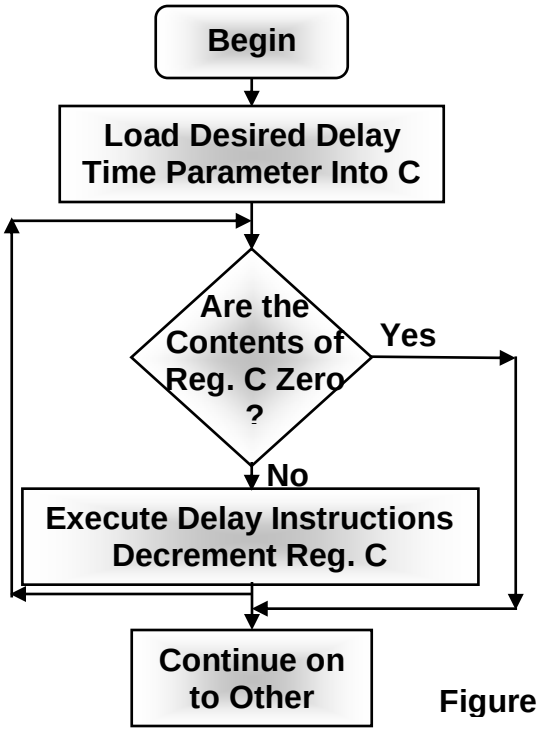


Figure 4-4

2- Write an 8085 program to calculate the sum of a series of numbers (16-bits data). The length of the series is in memory location $(2842)_H$, and the series itself begins in memory location $(2843)_H$. Store the sum in memory locations $(2840)_H$ and $(2841)_H$ (eight most significant bits in memory location $(2841)_H$).

Sample Problem:

2842=03 2844=FA
2843=C8 2845=96

Result: C8+FA+96 = 0258 ⇒ 2840=58 & 2841=02

3- Write an 8085 program to find the largest element in a block of data. The length of the block is in memory location $(2821)_H$ and the block itself begins in memory location $(2822)_H$. Store the maximum in memory location $(2820)_H$. Assume that the numbers in the block are all 8-bit unsigned binary numbers.

Sample Problem:

2821=05 2824=15
2822=67 2825=E3
2823=79 2826=72

Result: 2820=E3

5- Write a program to multiply memory locations [2800] x [2801]. Store the result in DE registers
[2800] =3C , [2801] = 32
The result (DE = 0BB8)

Discussion:

1- Calculate the check sum of a series of numbers, the length of the series is in memory location $(2841)_H$ and the series itself begins in memory location $(2842)_H$. Store the check sum in memory location $(2840)_H$. The check sum is formed by Exclusive-ORing all the numbers in the series into the accumulator.

Sample Problem:

2841=03
2842=28
2843=55
2844=26

	0	0	1	0	1	0	0	0	=28
+	0	1	0	1	0	1	0	1	=55
	0	1	1	1	1	1	0	1	
+	0	0	1	0	0	1	1	0	=26
	0	1	0	1	1	0	1	1	=5B

Result:

2840=(2842)+(2843)+(2844)=28+55+26=5B

2- Write an 8085 program to find the smallest element in a block of data. The length of the block is in memory location $(2841)_H$ and the block itself begins in memory location $(2842)_H$. Store the minimum in memory location $(2840)_H$. Assume the numbers in the block are 8-bit unsigned binary numbers.

Sample Problem:

2841=05 2844=15

2842=67 2845=E3
2843=79 2846=72

Results: 2840=15

3- Write an 8085 program to determine the number of negative elements (most significant bit is 1) in a block of data. The length of the block is in memory location $(2851)_H$ and the block itself begins in memory location $(2852)_H$. Store the number of negative elements in memory location $(2850)_H$.

Sample Problem:

2851=06
2852=68 2855=30
2853=F2 2856=59
2854=87 2857=2A

Results: 2850=02 ,Since 2853 & 2854 are negative numbers

4- Write an 8085 program to shift the contents of memory location $(2830)_H$ left until the most significant bit of the number is 1. Store the result in memory location $(2831)_H$ and the number of left shifts required in memory location $(2832)_H$. If the contents of memory location $(2830)_H$ are zero, clear both $(2831)_H$ and $(2832)_H$.

Sample Problem:

a- 2830=22
Result: 2831=88 & 2832=02

b- 2830=01
Result: 2831=80 & 2832=07

c- 2830=CB
Result: 2831=CB & 2832=00

d- 2830=00
Result: 2831=00 & 2832=00

5- What value remains in the accumulator after executing each of the following program segments? What are the states of the CY and Z flags?

a- MVI A,57
ANI 08

b- MVI A,FF
INR A

c- MVI A,FF
ADI 01

d- MVI A,80
CPI F0

e- MVI A,F0
CPI 80

6- Write an 8085 program to produce a delay time of 0.5 sec.

7- Write a program to divide the content of memory [2800] by the content of memory [2801] store the result in register D and the remainder in register E

[2800]= 09 , [2801] = 02. The result must be (D=04, E=01)

Experiment 5
Character Coded Data

Object:

To obtain the ASCII (American Standard Code for International Interchange) code for the keyboard keys function.

Theory:

Microprocessor often handle character-coded data not only do keyboards, telltale writers, communications devices, displays, and computer terminals expect or provide character-coded data (in binary form) that represented by ASCII. Many instruments and control systems do also. For example, character 10 represents the LF (Line Feed) function that causes a printer to advance its paper, and character 8 represents BS (Back Space). We will assume all of our character-coded data to be 7-bit ASCII with the most significant bit zero (see table (6-1).

	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	′	P
1	SOH	DC1	!	1	A	Q	a	Q
2	STX	DC2	“	2	B	R	b	R
3	ETX	DC3	#	3	C	S	c	S
4	EOT	DC4	$	4	D	T	d	T
5	ENQ	NAK	%	5	E	U	e	U
6	ACK	SYN	&	6	F	V	f	V
7	BEL	ETB	‘	7	G	W	g	W
8	BS	CAN	(	8	H	X	h	X
9	HT	EM	)	9	I	Y	i	Y
A	LF	SUB	*	:	J	Z	j	Z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

Table (6-1) Hex-ASCII Table of keyboard.

Length of a String of Characters:

To determine the length of a string of ASCII characters (7-bits binary with the most significant bit is zero). The string starts in memory location $(2841)_H$; the end of string is marked by a carriage return character (“CR”, hex=0D). Place the length of the string (excluding the carriage return) in memory location $(2840)_H$, then use the following data in executing the program for each case.

a- 2841=0D CR

b-

2841=43	C	2846=54	T
2842=4F	O	2847=45	E
2843=4D	M	2848=52	R

2844=50	P		2849=0D	'CR'
2845=55	U			

c- Enter the string=your name.

```
        LXI H, 2841H     : Pointer=start of the string
        MVI B, 0H        : Length=0
        MVI A, 0DH       : get ASCII 'CR' to compare
CHK CR: CMP (HL)         : Is character 'CR'
        JZ DONE          : Yes, end of string
        INR B            : No, add 1 to length
        INX H            : Increment the memory address
        JP CHK CR        : Examine next character
DONE:   MOV A, B         : Get string length
        MOV M, A         : Save string length
        HLT
```

The compare instruction sets the flags but leaves the carriage return character in the accumulator for later comparison. The zero flag is set as follows:
Z=1 if the character in the string is a carriage return.
Z=0 if it is not a carriage return.
The instruction INR B adds 1 to the string length counter in register B. MVI $B,0_H$ initializes this counter to zero before the loop. Remember to initialize variables before using them in a loop.

Procedure:

1- Write an 8085 program to search a string of ASCII characters (7-bits with the most significant bit is zero) for a non-blank character, the string starts in memory location $(2842)_H$. Place the address of the first non-blank character in memory location (2840, $2641)_H$ most significant four bits in memory location $(2841)_H$. A blank character is hex 20 in ASCII.

Note:

Use the following data: in execution and write the result in each case.

a-	2842=37	(ASCII 7)
b-	2842=20	SP
	2843=20	SP
	2844=20	SP
	2845=46	F

2- Write an 8085 program to replace all leading zeros with blanks in edit as string of ASCII character. The string starts in memory location $(2851)_H$, assume that it consists entirely of ASCII-coded decimal digits. The length of the string is in the location $(2850)_H$.

Note:

Use the following data in execution and write the result in each case:

a- 2850=02 STX
2851=36 ASCII 6
2852=30 ASCII 0

b- 2850=03 ETX
2851=30 ASCII 0
2852=30 ASCII 0
2853=38 ASCII 8

3- Write an 8085 program to add even parity to a string of 7-bit ASCII characters. The length of the string is in memory location $(2860)_H$ and the string itself begins in memory location $(2861)_H$. Place even parity in the most significant bit of each character by setting the most significant bit to 1 if that makes the total number of 1 bits in the word an even number.

Note:

Use the following data in execution and write the result:

2860=06
2861=31
2862=32
2863=33
2864=34
2865=35
2866=36

4- Write an 8085 program to determine the length of an ASCII message (length of A Teletype writer message). All characters are 7-bits ASCII with MSB 0. The string of characters in which the message is embedded starts in memory location $(2841)_H$. The message itself starts with an ASCII STX character $(02)_H$ and ends with ETX $(03)_H$. Place the length of the message (the number of character between the STX and the ETX but including neither) in memory location $(2840)_H$.

Sample problem:

2841=42
2842=02
2843=47
2844=4F
2845=03

Result: 2840=02 , since there are two characters between the STX and ETX.

Discussion:

1- Write an 8085 program to compare two strings of ASCII characters to see if they are the same. The length of the string is in memory location $(2861)_H$; one string starts in memory location $(2862)_H$; while the other starts at $(2882)_H$. If both strings are equal set $(2860)_H$ to the value of"00_H" otherwise set $(2860)_H$ to "FF_H".

Note:

Use the following data in execution and write the result in location $(2860)_H$:

a-			b-		
	2861=03			2861=03	
	2862=43	C		2862=52	R
	2863=50	P		2863=41	A
	2864=55	U		2864=4D	M
	2882=43	C		2882=52	R
	2883=50	P		2883=4F	O
	2884=55	U		2884=4D	M

2- Write an 8085 program to compare two strings of ASCII characters to see which is larger (i.e. which follows the other in 'alphabetical' ordering). The length of the strings is in memory location $(2841)_H$; one string starts from memory location $(2842)_H$ and the other starts at memory location $(2862)_H$. If the string in location $(2842)_H$ is larger than or equal to the other string clear memory location $(2840)_H$; otherwise, set location $(2840)_H$ to "FF_H" (all ones).

2841=03	
2842=43	C
2843=41	A
2844=54	T
2862=42	B
2863=41	A
2864=54	T

Results: 2840=00 since CAT is "larger" than BAT.

3- <u>Check Even Parity in ASCII Character:</u>

Check even parity in a string of ACII characters. The length of the string is in memory location $(2841)_H$ and the string itself begins in memory location $(2842)_H$. if the parity of all characters in the string is correct, clear memory location $(2840)_H$; otherwise, set the contents of memory location $(2840)_H$ to FF_H (all ones).

2841=03
2842=B1
2843=B2
2844=33

4- Truncate Decimal String to Integer Form:

Edit a string of ACII decimal characters so as replace all digits to the right of the decimal point with ASCII blanks $(20)_H$; the string starts in memory location $(2801)_H$ and is assumed to consist entirely of ASCII-coded decimal digits and a possible decimal point $(2E)_H$. The length of the string is in memory location $(2800)_H$. if no decimal point appears in the string, assume that all digits are whole numbers with the decimal point (implicit) at the far right.

Sample Problem:

2800=04
2801=37 7
2802=2E .
2803=38 8
2804=31 1

Result:

2801=37 7
2802=2E .
2803=20 'SP'
2804=20 'SP'

Ministry of Higher Education
& Scientific Research
Al-Mustansiriyah University
College of Engineering

Electrical Eng. Department
Microprocessor Lab
Third Year Class
Hussein A. Abdulnabi , Ammar A. Sohrab & Gregor A. Armice

Experiment -6
Subroutines and the Stack

Object: to examine the machine instructions associated with subroutines can be use in a simple application example.

Theory: 8085 subroutines are called from user program to perform a variety of task. In program it may be necessary to perform a sub task many times over. Subroutines provide an efficient means of repeating a series of instructions in a program. Subroutines are linked with the main program through the use of the stack. Subroutines not only safe program memory because repetition of instructors is avoided; they also provide the opportunity to construct a program in convenient sections which can be written and tested independently, and then collected together to perform the overall task.

The Stack: Stack is a special memory area that is used to "remember" where to return after a subroutine is executed. The stack is also used for temporary data storage. A stack is simply a last – in – first – out queue. It is implemented as a set of successive read/write memory locations, together with a register called a (Stack Pointer) which holds the address of the entry at the top of the stack, and this use the stack pointer the register SP. SP may be initially set at any arbitrary value, although usually on the MAT385 it is set to 2oc2 by the instruction initialize stack pointer.

A byte is entered into the stack by first decrementing the stack pointer, so that it contains the next address beyond the head of the stack, and then placing the byte in the location at the new address. This operation is some times referred to as "pushing" the byte onto the stack. When a byte is no longer required to remain on the stack the converse process, "popping" the byte from the stack, must first transfer the byte to whenever it is next required, and then increment the stack pointer so that it still points to the head of the reduced stack.

8085 stack control & subroutine instructions:

Here are some of the 8085 instructions used in stack and subroutine applications:

Instruction	Description
LXI sp,16-bit data	Load stack pointer with 16-bit value
CALL	16-bit address call subroutine at address specified
CC	16-bit address call subroutine if cy=1
CNC	16-bit address call subroutine if cy=0
CZ	16-bit address call subroutine if z=1
CNZ	16-bit address call subroutine if z=0

RET	Return to calling program
RC	Return if cy=1
RNC	Return if cy=0
RZ	Return if z=1
RNZ	Return if z=0
PUSH rp	Push register pair onto stack
POP rp	Pop register pair from stack
XTHL	Exchange top of stack with HL contents
SPHL	Move HL contents to stack pointer
DAD sp	Add stack pointer contents of HL

Instructions in the CALL and RETURN instruction group automatically use the stack to create the linkage between a main program and a subroutine. Is a subroutine modifies the contents of certain registers, and if you must use these values later in the program, then place the contents of the registers in a safe place before calling the subroutine. Either copy the contents of the registers to memory using data transfer instructions, or place the values temporarily on the stack. Store data on the stack using the PUSH instructions; later, retrieve the data using POP instructions.

Operation of Subroutine Instructions:

			Stack	
Address	Contents	Instruction	Address	Contents
2810	CD	Call 2848	20C0	xx
2811	48		20C1	xx
2812	28		20C2	xx
2813		2813 pc		
		20C2 sp		

i) **Instruction CALL just started to be executed PC updated**

Address	Contents	Instructions	Address	Contents
2810	CD			
2811	48		20C0	13
2812	28	2813 pc	20C1	28
2813		20C0 sp	20C2	xx

ii) PC contents saved on stack and SP decremented to point to last entry

2810	CD			
2811	48		20C0	13
2812	28	2848 pc	20C1	28

		20C0 sp	20C0	xx

iii) **Control transferred to subroutine start address 2848**.

Whenever a RETURN instruction is executed, the process illustrated is reversed. The program counter is loaded (least significant byte first) with the contents at the top of the stack, the stack pointer is incremented by two and control returns to the instruction following the subroutine call. Subroutine call and return instructions are available that are conditional in just the same way and for the same range of conditional jump instructions.

Example (1)

The following program segment saves H and L on the stack

```
LXI     Sp,20C2H        ;initialize stack pointer
PUSH    H               ;Save HL on stack
Call    SUB1            ;Call subroutine SUB1
POP     H               ;Restore HL to original
                        value
```

When saving several register pairs on the stack, the order of pushing and popping is very important, consider the following example, which demonstrates the last-in-first-out nature of the stack:

```
PUSH    H               ;save HL
PUSH    D               ;save DE
CALL    SUB1            ;CALL subroutine sub1
POP     D               ;Restore DE
POP     H               ;Restore HL
```

If the registers are restored in the wrong sequence problems may arise:

```
PUSH    H               ;save HL
PUSH    D               ;save DE
CALL    SUB1            ;CALL subroutine sub1
POP     H               ;Restore HL to old DE value
POP     D               ;Restore DE to old HL value
```

In this program, the contents of register pairs D and H are exchanged, which may not be intended result. The instruction PUSH PSW is used to place the processor status word, or PSW, on the stack. The PSW consists of the Accumulator register in the most significant byte and the flag register in the least significant byte. This instruction provides a useful way to access the contents of the flag register. For example, the program segment

```
PUSH    PSW
POP     H
```

Place the flag register contents into register H and moves the accumulator contents to register L.
The DAD SP instruction provides the only way to transfer the stack pointer contents to another register. For examination, the following program segment demonstrates this application:

```
LXI   H, 0000 H     ; clear HL
DAD   SP            ; ADD stack pointer contents to HL
```

Register pair HL now contains a copy of the stack pointer contents.

Monitor Subroutines

The monitor ROM contains several subroutines that can be used to simplify information processing. The UPDAD and UPDDT subroutines are used to disply information on the LED digits.

Procedure

1. The following program manipulates stack data. Study the program and predict what will happen to memory locations 2050H-205F H.

```
        LXI     SP,205F(H)      ;initialize stack pointer
        XRA     A               ;Clear Accumulator
        LXI     H, 21050  H     ;point to start of array
        MVI     B, 10(H)        ;Loop counter
Loop:   MOV     M, A            ;clear M location
        INX     H
        DCR     B
        JNZ     Loop            ;Repeat until done
        LXI     H, 1234(H)
        LXI     D, 5678(H)
        LXI     SP, 20C2
        CALL    SUB1
                                ;put SP here before quieting so the
                                monitor program will not bother it
        RST     1               ;quit
SUB1:   PUSH    H               ;subroutine starts here
        PUSH    D
        LXI     H, 0000(H)      ;do this to capture SP value
        DAD     SP
        SHLD    2050 H          ;save it in memory
        POP     D
        POP     H
        RET
```

Assemble and execute the program – examine memory locations 2050H – 205H to see if those values were manipulated as you predicted. Observe the contents of DE and HL. Note: (locations 2050H – 205F H were cleared before doing any stack operations normally the stack is not cleared before used.

2. Write an 8085 subroutine to generate a precise time delay where the delay time parameter is passed to subroutine in register B. Use this calling program to test your subroutine

```
LXI     SP, 20C2  (H)
MVI     B, XX (H)
CALL    DELAY
RST     1
```

Test your subroutine by producing delay times of 1, 10, 30 and 60 sec. verify the result using a stopwatch.

3. Write an 8085 subroutine program to determine the length of a string of ASCII characters. The starting address of the string is in register pair HL. The end of the string is marked by a carriage return character (CR: hex 0D). Place the length of the string (excluding the carriage return) into the Accumulator.

Sample program

a- (HL) = 2843
(2843) = 0D

Result: (A) =00

b-

(HL)	=2843	
(2843)	=52	'R'
(2844)	=41	'A'
(2845)	=54	'T'
(2846)	=48	'H'
(2847)	=45	'E'
(2848)	=52	'R'
(2849)	=0D	CR

Result: (A) =60

4. Write an 8085 subroutine program to add even parity to a string of 7-bit ASCII characters. The length of the string is in the accumulator and the starting address of the

string is in register pair HL, place even parity in bit 7 of each character (i.e. set bit 7 to 1 if that makes the total number of 1 bits in the byte even).

Sample problem:

		Result	
(A)	=06	(2841)	=B1
(HL)	=2841	(2842)	=B2
(2841)	=31	(2843)	=33
(2842)	=32	(2844)	=B4
(2843)	=33	(2845)	=35
(2844)	=34	(2846)	=36
(2845)	=35		
(2846)	=36		

5. Write an 8085 subroutine program to add two multiple-byte binary numbers. The length of the numbers in bytes is in the accumulator. The starting addresses of the numbers are in register pair DE and HL. All the numbers begin with the least significant bits.

Sample problem:

		Result	
(A)	=04		
(DE)	=2851	2871	=7B
(HL)	=2861	2872	=DD
(2851)	=C3	2873	=3A
(2852)	=A7	2874	=44
(2853)	=5B		
(2854)	=2F		
(2861)	=B8		
(2862)	=35		
(2863)	=DF		
(2864)	=14		

i.e. 2F5BA7C3
14DF35B8 +

443ADD7B

Discussion:

Each of the following program segments uses the stack. For each program segment specify the resulting contents of any affected registers. Draw a memory map showing the stack locations used and their final contents.

a)

```
LXI     SP, 6000 (H)
LXI     H, 1234 (H)
PUSH    H
```

b)

```
LXI     SP, 2050 (H)
LXI     H, 1234 (H)
SPHL
PUSH    H
```

c)

```
LXI     SP, 3040 (H)
LXI     D, 1234 (H)
PUSH    D
LXI     H, 5678 (H)
XTHL
```

d)

```
ORG     2000 (H)
LXI     SP, 3000 (H)
CALL    SUB1
```

Ministry of Higher Education
& Scientific Research
Al-Mustansiriyah University
College of Engineering

Electrical Eng. Department
Microprocessor Lab
Third Year Class
Hussein A. Abdulnabi , Ammar A. Sohrab & Gregor A. Armice

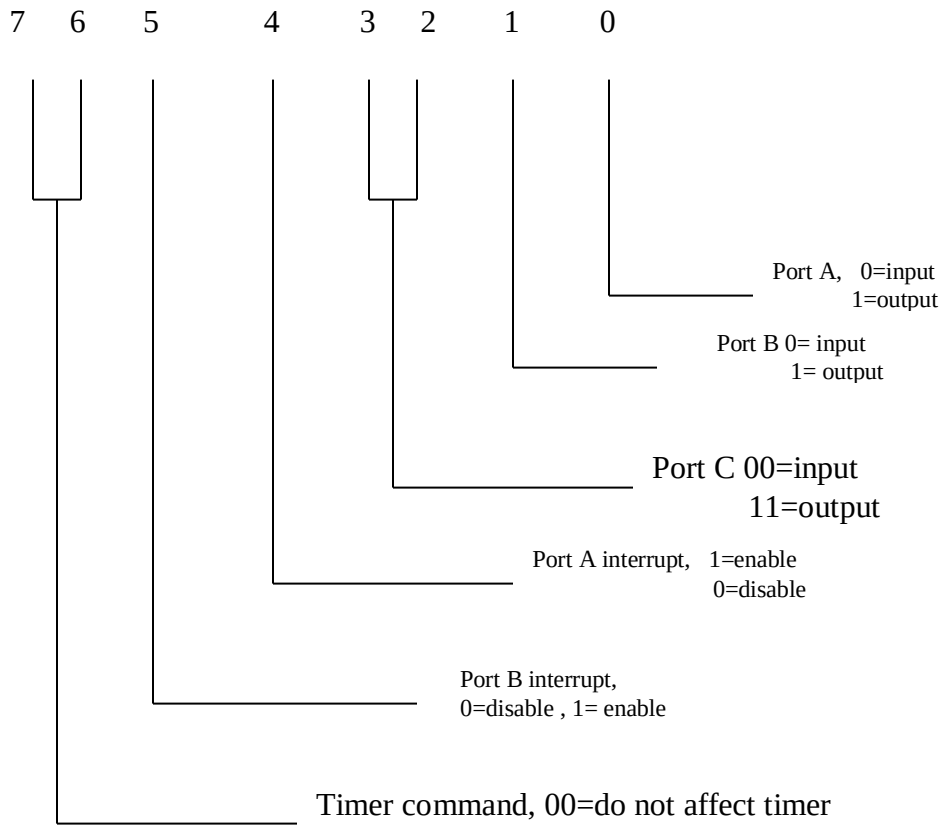
Experiment -7
Input / Output Ports

Object:
To explore the nature of a programmable input-output device and its use for simple input-output data transfers.

Or

To study the operation of the input-output (i/o) ports of the MAT 385 trainer with the aid of the install 8085 i/p-o/p instruction.

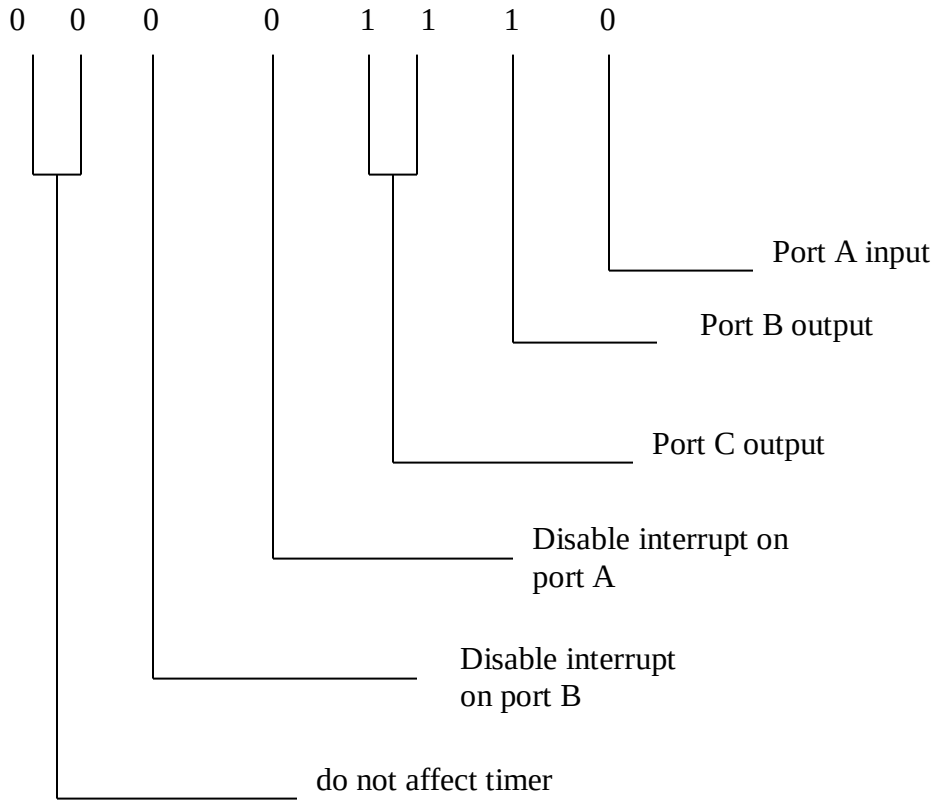
Theory:
Computer can monitor the outside world using input ports; they can control it using output ports. An 8085 microprocessor performs input/output (I/O) tasks using special I/O instructions and some external hardware.
The MAT 385 contains a set of 8 switches can be used as logical inputs and a set of 8 LEDs which can be used to display logical outputs. A logical 1 corresponds to an LED being ON and a logical 0 to an LED being OFF.
Input/output operation of the 8085 MPU is achieved through the use of 8-bit programmable ports. Since this input/output device may be programmed on either input or output, hence it must be programmed into the required configuration for the programmable I/O port used (8155), the command word is:



In order to write the command byte to the programmable interface, the proper command byte should be set to command port (in the MAT 385 it is at address 20h). for example if it is required to use:

Port A as input
Port B as output
Port C as output

Then we use:



The 8085 instructions relating the use of the I/O ports are:

OUT (port)

This instruction is used to send the contents of the accumulator to the required port. Ports are address on eight bits and hence only 256 I/O locations are addressed by this location only.

IN (port)

This instruction is used to read the contents of the required port into the accumulator.

The locating used to address the I/O ports are:

Address (hex)	Use
20	Command register
21	Port A (data)
22	Port B (data)
23	Port C (control)

On the MAT 385, the eight switch input are connected to port A (21) and the eight LED, output indicators are connected to port 22, port 23 must be set to 00 to enable the switch inputs.

The single step command provided by the monitor program uses the timer within the same device to determine when a single instruction has been executed. The monitor program, therefore; modifies the command byte within the device as each step operation is performed. It is necessary, when performing input-output operation using the single step facility, to write in the command register. The monitor program then automatically transfers the information in this memory location to the command register as each single step operation is executed.

Example: Transferring data to the LEDs

In the following example port 22 is configured to be an output port and port 21 to be an input port. An arbitrary data byte is transferred to the LED indicators on port 22.

```
MVI    A, oE        ; command byte
STA    20FF         ; save command byte to permit single step operation
OUT    20           ; transfer command to 8055
MVI    A, 00        ; load control port data
OUT    23           ; set control port
MVI    A, 55        ; load LED data
OUT    22           ; transfer data to LEDs
```

Procedure:

1. Code the example 8-1 and load it into memory starting at address 2800. Use the "single step" command. Charge the LED data and repeat. Replace the OUT 22 instructions with an in 21 instruction the accumulator should contain the switch input data; confirm this using the "exam register" command.
2. Write a program to output the number (AA H) to the LEDs on the panel of the trainer (the LEDs are connected to port 22 of the MAT trainer).
3. modify the above program to read the states of the switches and then display them using the LEDs on the MAT (the switches are connected to input port 21 of the MAT).

4. write a program to continuously flash the LEDs of the MAT panel (flash the LEDs by continuously writing (55H) and (AA H) to the LEDs lamp set).
   01010101=55(H)
   10101010=AA (H)

Discussion:

1. Draw a flowchart illustrating the programs in procedure step 3 and 4.
2. Show how you can modify the flashing ration of the program in step 4. Illustrating using a flowchart (or a program) how can you modify the above flashing ration using the states of the switches in the MAT panel.

Switch state		value
0000	0000	1
0000	0001	1/2
0000	0010	1/4
0000	0011	1/8

Note:

The value of the accumulator can be displayed on the LED display of the MAT 385 as two hex digits, by calling the subroutine at address 036E (display).

Ministry of Higher Education
& Scientific Research
Al-Mustansiriyah University
College of Engineering

Electrical Eng. Department
Microprocessor Lab
Third Year Class
Hussein A. Abdulnabi , Ammar A. Sohrab & Gregor A. Armice

Experiment -8
Practice 1 "Traffic Lights"

Object:
Getting use of the traffic light control model MIC957 microprocessor application board.

Theory:
The crossroads with a pedestrian crossings are drawn on the board. Two sides of the crossroads are indicated on panel together with their traffic lamps (three lamps for north – south and three for east – west road, with two lamps for the pedestrian crossings). That means there are eight lamps to be controlled by the microprocessor, so we need only one eight bit output port to control all the eight lamps (one bit for each lamp, logic 1 for ON and 0 for OFF).
It is recommended to use port B (2A) of (PIO) 2 as an output port to control the eight lamps as follows:

North road	East road	Pedestrian
Bit 0 RED	Bit 3 RED	Bit 6 Red
Bit 1 Amber	Bit 4 Amber	
Bit 2 Green	Bit 5 Green	Bit 7 Green

Each time a pushbutton is pressed the computer will receive a logic 0 on the corresponding bit position, yet on normal conditions the computer will read logic 1 on this bit.

Practice (1)
The sequence used for traffic lights is: red; red and amber together; green; amber; red again; etc. the controlled read junction is shown in fig. (9.1). it comprises the four roads labeled North, South, East, West and a set of traffic lights at each corner (RAG).
The two sets of lights seen by traffic approaching the junction from North and South change in the same sequence, similarly, the two sets seen by traffic approaching from East and West also change in the same sequence. There are two principal items

The first is the matter of getting the input and output ports working and the second is to establish the sequence of bit-patterns to be associated with each stage of the eventual sequence.
The example therefore has the following stages:

a. Calculate the number which must be sent to the output port for each stage of the sequence tabulated in table 9-1.
b. Complete program which enable each of these numbers to be entered by you at the keyboard and sent to the output port.

(See program 1)

Table (9-1)

south	west	Pedestrian
Red	Red	Red
Red	Red	Green
Red	Red/Amber	Red
Red	Green	Red
Red	Amber	Red
Red/Amber	Red	Red
Green	Red	Red
Amber	Red	Red

The sequencer can be implemented by sorting this state table in memory and stepping through it. The appropriate states of the lights are then output, writing the appropriate delay time between steps. This is shown in the flowchart of fig (9-2).

This flowchart assumes that the most significant three bits of port A drive the north/south lights and the next most significant three bits drive the east/west lights.

Note:
There are two types of delays: long delay between over all direction chanches (2 min.) short delay between transitional light setting (3sec.)

Example 2: sequence altered by pedestrian request

It is required (in the absence of pedestrians) to allow vehicles to travel alternately along the north-south and the east-west roads. If however a pedestrian presses the request button by the crossing, the normal sequence should proceed until the south road lights return to red, and then the pedestrian crossing should change to green and back to red before the normal sequence as in those for traffic on each of the two roads, for this reason table (9-2) shows “dummy” entries in the sequence at lines 0 and 2, in order to make up the regular pattern of four lines per sub-sequence.

Line	South	West	Pedestrian	Time	Sub-sequence
0	Red	Red	Red	0	pedestrian
1	Red	Red	Green	6	Pedestrian
2	Red	Red	Red	0	Pedestrian
3	Red	Red	Red	1	Pedestrian
4	Red	Red/amber	Red	2	West
5	Red	Green	Red	6	West
6	Red	Amber	Red	3	West
7	Red	Red	Red	1	West
8	Red/amber	Red	Red	1	South
9	Green	Red	Red	6	South
10	Amber	Red	Red	3	South
11	Red	Red	Red	1	South

Time is an arbitrary unit. It will be convenient to make all times a multiple of a single variable, whose value can then be changed in order to adjust the time scale simply. Each of the lines of table (9-2) will have associated with it two pieces of data. One is the bit pattern the other is the required delay time. It is not necessary to make the times very accurate. It must be recommended however that a pedestrian may press the button to request a green light on the crossing at any time, he will then expect the system to remember the request until it has been fulfilled (see program 1).

Program for example 1 and example 2:

Start	LXI	Sp,20C2(H)	Set stack pointer
	MVI	A,02(H)	Port command byte
	OUT	28(H)	To command REGISTER
	MVI	A,Q	
	OUT	2A(H)	Extinguish lamp3
	LXI	D,Q	Initialize the index in D,E
	CALL	OUTP	Change the lights
	CALL	GPLAY	Get delay duration from data table
	CALL	Wait	Wait, strong any pedestrian requests which arrive meanwhile
	INX	D	Increment the index
	MOV	A,E	Get it (8-bits) into A
	CPI		
	JC		
	LDA		
	ANI		
	MVI		
	JZ		
	MVI		

	SUB		
	STA		
	NOP		
	JMP		
	DS		
	LXI		
	DAD		
	MOV		
	RET		
	SUB		
	ORA		
	RZ		
	PUSH		
	PUSH		
	MVI		
	CALL		
	LXI		
	CALL		
	DCX		
	MOV		
	ORA		
	JNZ		
	POP		
	POP		
	RET		
	DS		
GREQ	PUSH		
	LXI		
	IN		
	CMA		