Microprocessor Fundamentals (246)

Handout # 1

1. Introduction

- The evolution of the computer Marketplace over the last 30 years has taken us from large mainframe computers to smaller minicomputers and now to even smaller microcomputers.

- In other words, we have now three classes of computers:
  1. Mainframe
  2. Minicomputer
  3. Microcomputer

- The mainframe was used in an environment where it serviced a large number of users (example: University)

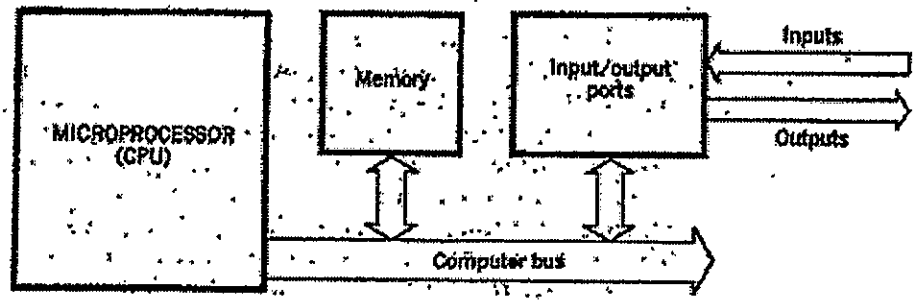
- Minicomputer is the primary computer solution for the small, multi-user business environment. In this environment several to hundred users connect to the system with terminals, all sharing the same computer system, with many of these users actively working ob the computer at the same time. The main disadvantage of this system is if the minicomputer is not working, all users are down and cannot work at their terminals.

- Microcomputer is what we use it now everywhere (home, school, work...etc) which is called the personal computer (in short form PC). Each PC will have its own application installed on it.

- Each of these classes does not cancel or affect on the other. Each one has its own environment and use.

1.2 Microcomputer Structure

- Microcomputer is a general purpose electronic data processing system intended for use in a wide variety of applications.



- The above diagram represents the microcomputer system.
- The microcomputer system consists of a number of subsystems interconnected by paths that transfer data between these subsystems
- It consists of:
  1. Central processing unit (CPU) (Microprocessor)
  2. Input/Output
  3. Memory

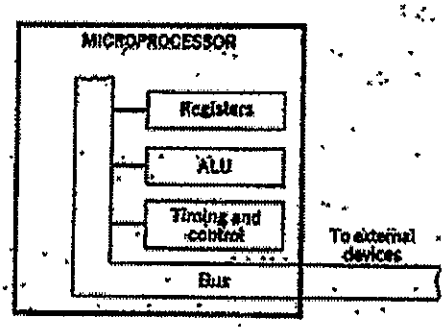
  All are connected together by the computer Bus
- The heart of the microcomputer is its central processing unit CPU.
- The microprocessor is implemented with a VLSI devices.
- This microprocessor is a general purpose processing unit built into a single integrated circuit (IC)
- The Input/Output subsystems are used to connect the microcomputer to the external word
- The Memory subsystem will be talked about it in the later sections

1.3 Microprocessor Structure

- the microprocessor is the central component in a ny microcomputer system.

- This microprocessor:
  1. controls the functions performed by the other subsystem devices and provides the system's arithmetic and logic capability
  2. it fetches instructions from memory and decodes and executes them
  3. it references memory and I/O devices for data responds to control signals from external devices

- A microprocessor system is comprised of a very large and highly structured collection of digital storage locations with a central control device.



- The above diagram represents the microprocessor system.

- The microprocessor performs the central control and has its own structured collection of storage locations, mostly 8-bits long.

- The storage locations within the microprocessor are called registers.

- A register is a temporary data storage location

- We can say that the microprocessor consists of:
  1. Registers
  2. ALU (Arithmetic Login Unit)
  3. Timing and Control

- The bus connects the microprocessor to the memory and I/O devices, which are used to communicate with the outside world.

1.3.1 Registers

- The microprocessor's registers are critical to its operation, because they are resident on the microprocessor, they can be manipulated rapidly without selecting external chips.
- The resisters are divided to several types, some of them are:

1. Program Counter (PC):
   It is 16-bit number contains the address of the next instruction to be fetched. Its size determines the range of memory in which program instructions can be located.

2. Accumulator (AC):
   It is used extensively for data manipulations. The arithmetic and login operations commonly use the AC for data to initiate the operations and as the destination for the result.

3. Flag Register:
   It contains various bits of status information (example: zero flag, carry flag ...etc). It is critical for computer decision making.

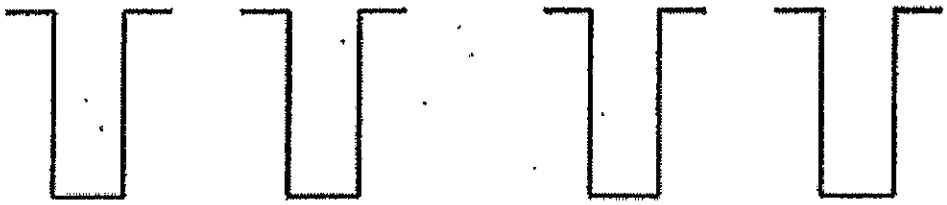
4. Index Register:
   Used for counting and certain kinds of memory operations.

5. Instruction Register (IR):
   Which is the destination for an instruction during the fetch cycle.

6. Stack Pointer (SP):
   Contains the address of a free memory location. Some set of memory locations are arranged and connected in such a way that is called Stack. The stack has two operations PUSH and POP. The PUSH operation will increment the SP while the POP decrement the SP. This kind of stack also called Last in First out. When we make a read operation then we are doing POP, on the other hand, the write operation is PUSH. The advantage of having stack that we will have many available memory location.

But the disadvantage that it takes longer time because data transfers occur external to CPU.

Example:



1.3.2 ALU

- The arithmetic logic unit (ALU) and Timing and Control blocks in the microprocessor system form the central control features of the computer system.
- The ALU generally do the arithmetic operations (addition, subtraction...etc) and the logic operations (AND, OR, exclusive OR....etc).
- The ALU produces flags , such as the carry flag
- Example of ALU operation: add R
  AC+R → AC
  The ALU will add the content of AC to the value R and replace it in the AC.

1.3.3 Timing and Control

- It refers to complex set of logic functions that synchronize the machine and direct various operations.

- A number of critical timing conditions must be taken if there is any data transmission.

- The timing and control block decodes instructions and directs the execute cycle.

- Example: add D with carry.
  The timing and control sequence will be as follow:
  1. command to ALU to add AC with D and the carry flag
  2. then, ALU store the result into AC
  3. set or reset the flags.
  ⇒ Timing and control block coordinate this activity.

- Microprocessors also allows interrupts.

- An interrupt suspends the current operation of the processor in response to an external input so that it can attend to a critical request.

1.4 Bus Structure

- As we said, the microprocessor system performs many data transfers into and out of the CPU. These transfers use a bus structure.

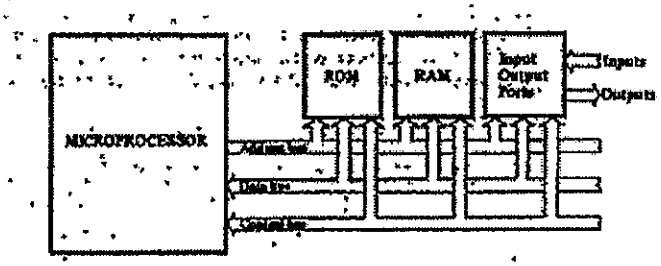
- A bus is a set of conductors over which data, address, control or other information can be transmitted.

- The microprocessor system include the following busses:
  1. Data Bus (8 bits)
  2. Address Bus (16 bits)
  3. Control Bus (6-to-8 bits)

- The data bus is used to transfer all instructions and data.

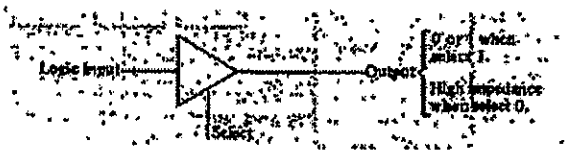
- The address bus identifies any memory location to be activated.

- The control bus carries all timing and control signals for coordinating data transfers.



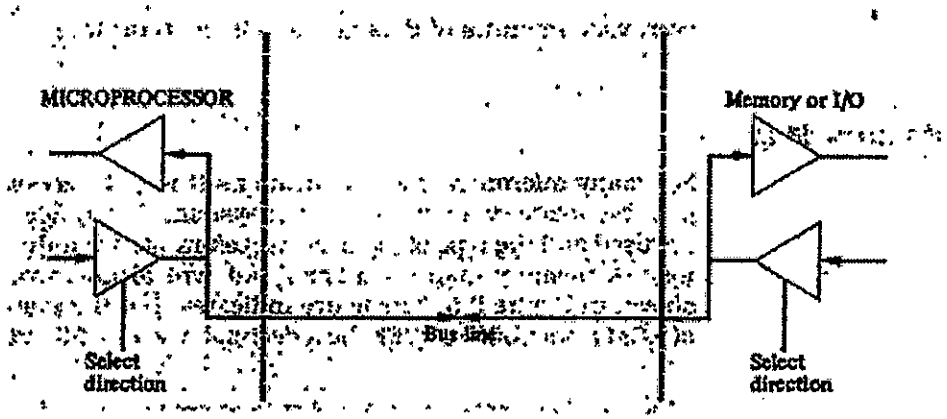
- While reading or writing, both the CPU and the memory circuit must be able to:
  1. drive a data bus line to a one
  2. drive a data bus line to a zero
  3. allow a data bus line to be driven by other devices.

- This operation is implemented by the three-state driver as in the following figure.



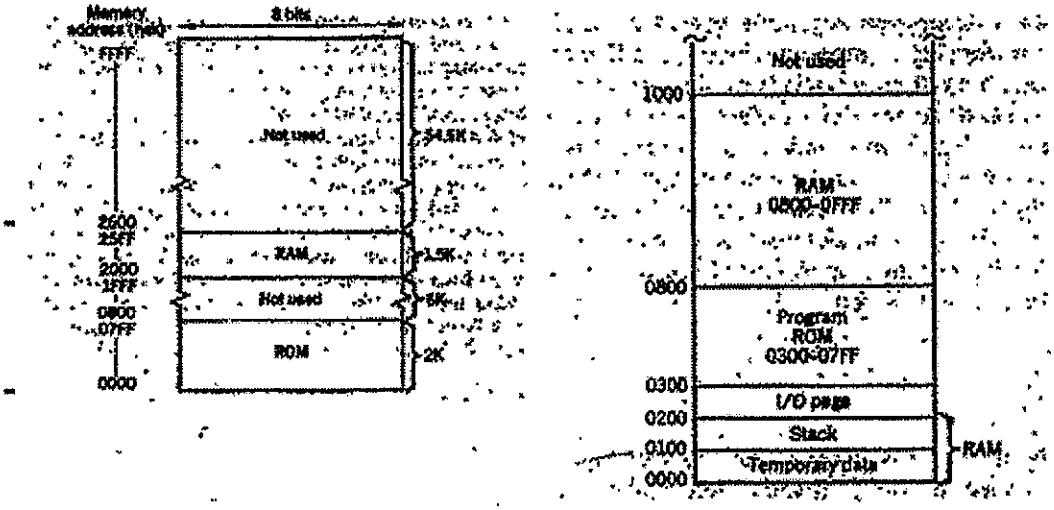
- The three-state driver is a digital circuit that exhibits three states. Two if these states are signals equivalent to logic 1 or 0 as in the input and the third state is a high impedance state which behaves as an open circuit.

- To construct a bidirectional bus we use a bidirectional interface as shown in the following figure

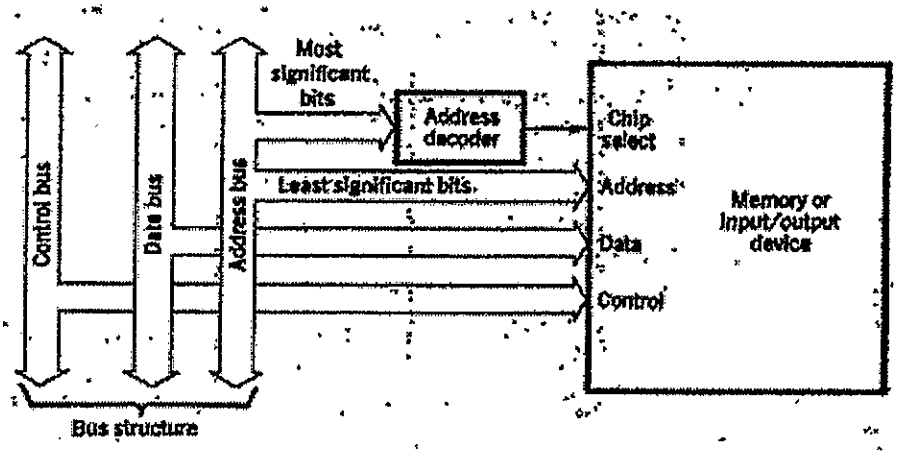


1.5 Memory and I/O

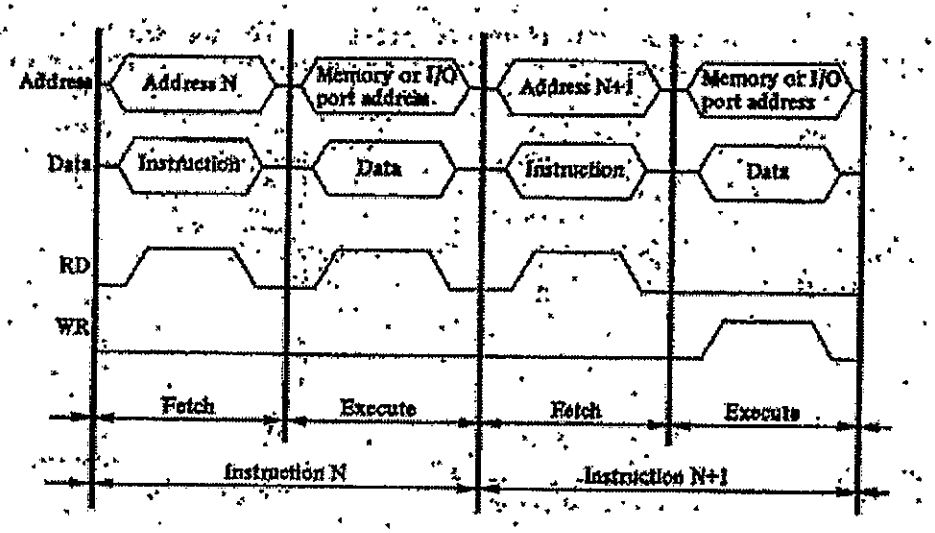
- In a microcomputer system the memory can be divided into addressable (primary) and non-addressable (secondary) memory.
- The addressable memory is considered as an internal part of the microcomputer.
- We have two types of the addressable Memory:
  1. Random access Memory (RAM)
  2. Read only Memory (ROM)
- Both types are accessed randomly, but the RAM can be written to or read from while the ROM only read from.
- ROM is nonvolatile while RAM is volatile.
- The non-addressable is thought of as memory external to the microcomputer (example: floppy disk)
- The range of addresses that are used and the type of memory in each range are indicated by a memory map.
- The following figure indicates a typical memory map:



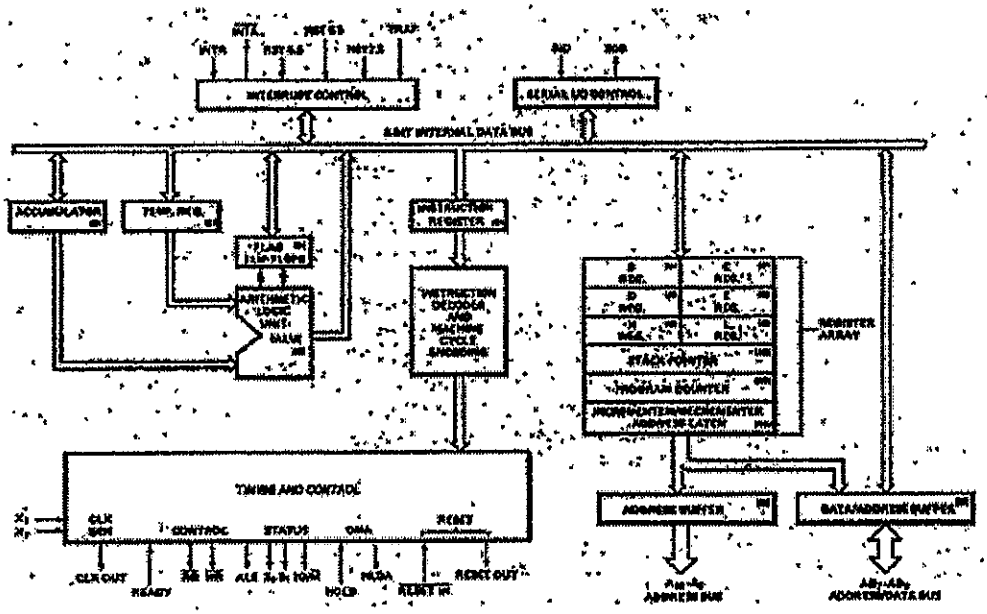
- Memory space is equal to 2^n where n is the address size (number of bits)
- To determine which device to be accessed, an address decoding should be done to enable the required device.
- This is illustrated as follows



- During the execution time, the timing and control signals are generated to do the read/write operations.

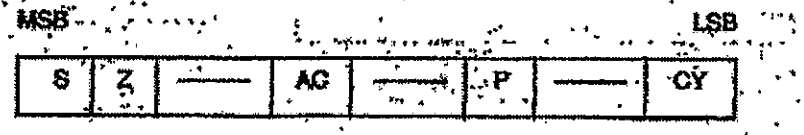


1.6 The 8085 Microprocessor



- The 8085 microprocessor has the same structure as we discussed before for the basic microprocessor.

- 8085 Flag Register



- The flag register contains:
  1. S: sign flag (sign number)
  2. Z: zero flag (result zero)
  3. AC: Auxiliary flag (carryout of bit 3)
  4. P: parity flag (set if even numbers of ones and 0 if odd)
  5. CY: carry flag (used for addition (carry) and subtraction (borrow))

Microprocessor Fundamentals (246)

Handout # 2

Introduction to programming

- A microprocessor can be considered as a device that reads binary-coded information from its input, manipulates this information according to a program stored within its memory, and subsequently produces information at its output. This process is shown in Figure 1.

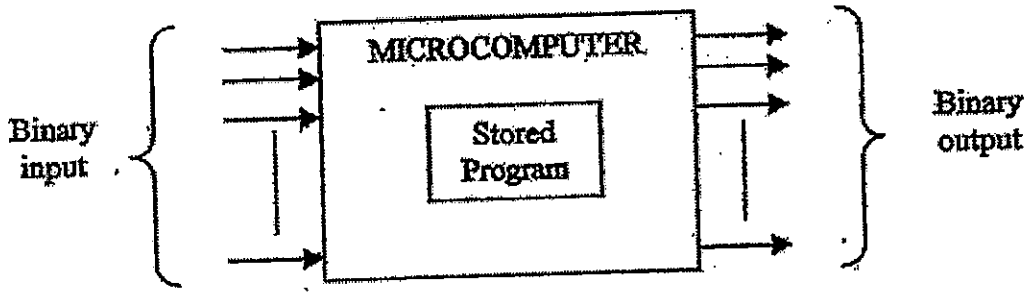
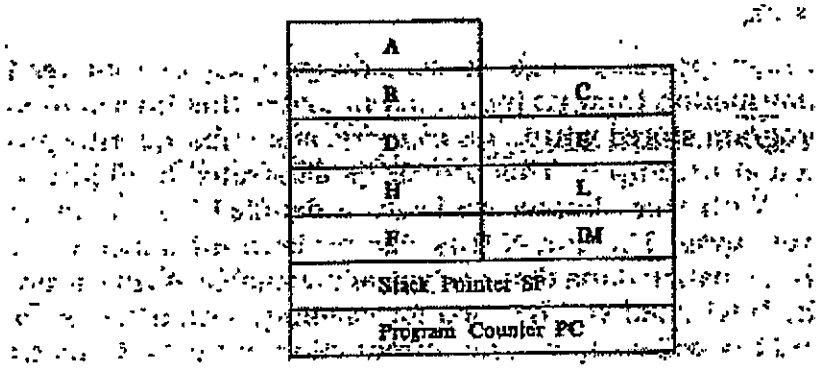


Figure 1

- The program instructions/manipulation process could affect the internal registers and/or memory locations.
- Some of the registers are accessible by the machine instructions. These are called visible registers.
- The non-visible register is the one where the processor can not access it directly.

Microprocessor Registers:

- As mentioned before, the microprocessor has its own internal register.
- The 8085 registers are listed down:



A Is an 8-bit arithmetic register (accumulator).

B, C, D, E are four 8-bit general purpose registers.

F is an 8-bit flags register (modified by ALU operations).

IM is an 8-bit interrupt control register.

HL are two 8-bit registers which are normally used to form a 16-bit memory pointer.

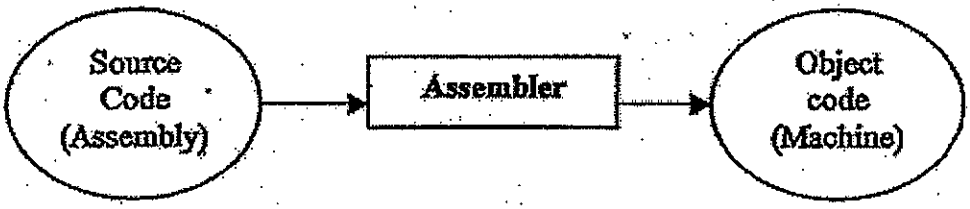
SP is the stack pointer register – this contains a 16-bit memory address which displays (points to) the top of a system stack.

PC is the program counter register which contains a 16-bit memory address which points to the next instruction to be executed.

- These registered will be used in the programming.
- Note that the B and C registers are shown as a pair although they can be used separately. Also, this is true for DE and HL registers.

Assembly Language:

- To let the microprocessor do its job in a good manner we should program it as we want so, it will do this functions.
- The program is a set of instructions
- The microprocessor understands the binary representation which means the form of zeros and ones.
- The programmer writes his program in a convenient representation of machine instruction, such that each instruction has its own representation in words.
- This kind of representation is called assembly language
- The assembly language is far more readable and understandable
- So, the programmer programs the processor using this assembly language then, it is converted to the binary representation to be understandable by the machine.
- The program written in assembly code will be the source code and the translated will be the object code (machine code).
- The translator that converts the assembly language to machine code is called assembler.
- The conversion between the assembly code and the machine code is a straightforward.



- Each instruction in the program is called mnemonic.
- Then, the complete 8085 assembly language instruction format is:

  Label operation mnemonic operands ;comments

- The label and comments are optional.
- The operation mnemonic is a simple word that reflects the action involved.
- The operands are the registers or memory locations that are used to perform this operation.
- Some mnemonics are called directives, which provide the computer with important data.
- To understand how to write a program, we should know first the instruction types that can be used in the processor.

Instruction Types

- In microprocessor we have a large number of machine instructions. These instructions can be generally classified to five groups as follows:

1. Data Transfer
2. Data Manipulation
3. Transfer of control
4. Input/output
5. Machine control

- Data Transfer: moves the data between registers or between register and memory location.

MOV	Move
MVI	Move Immediate
LDA	Load Accumulator Directly from Memory
STA	Store Accumulator Directly in Memory
LHLD	Load H & L Registers Directly from Memory
SHLD	Store H & L Registers Directly in Memory

An 'X' in the name of a data transfer instruction implies that it deals with a register pair (16-bits);

LXI	Load Register Pair with Immediate data
LDAX	Load Accumulator from Address in Register Pair
STAX	Store Accumulator in Address in Register Pair
XCHG	Exchange H & L with D & E

Example:

MOV A,B

This instruction will move the content of B-register to A-register. The move here means copy.

- Data Manipulation: performs the arithmetic or logical operations on data stored in a register or memory location.

Example:

ADD B

This instruction modifies the content of the A-register (accumulator) to the sum of previous contents and the contents of B-register.

- Transfer of control: provides the capability and ability to transfer from one sequence of operations to another, based on variety of conditions.

Example:

JMP 0FE3H

This instruction is unconditional branch to the memory location 0FE3H

- Input/Output: moves the data between the registers and I/O ports

Example:

OUT 5H

The contents of A-register will be moved to output port #5

- Machine control: affects the state or mode of operation of the processor itself.

Example:

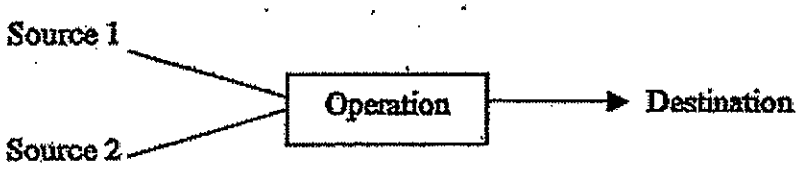
NOP

This causes the machine to wait through an instruction.

- Also, each of this group is classified depending on the types of the source and destination address.
- These sub-classifications called addressing mode.

Operand Addressing Mode

- Data manipulation instruction implies access as many as three locations:
  - Two for values to be manipulated (i.e. sources)
  - One to specify the destination for the result



- Most microprocessor including 8085 only requires a maximum of two addresses to be specified.
- The type of the source or destination is determined by the instruction type which is called the addressing mode.
- Each microprocessor could provide several addressing modes.
- The main four types are:

1. Register addressing 2. Immediate addressing	Used for data transfer and manipulation that use only internal registers
3. Direct addressing 4. Indirect addressing	Used for data transfer and manipulation that uses the system memory

- Other addressing modes used in some microprocessors like index addressing; this type is the same as the third and fourth type in addition of a shift used.

- When several addressing modes are used more instruction will be available to program which let the programming faster and easier.

Data Transfer Instructions

- Data Transfer instructions move data from one location to another.
- The heart of each of the data transfer instruction is the definition of the locations where the source and destination bytes are to be found.

1. Register Addressing

- This kind of addressing moves data between internal registers.
- The order of the operands defines which register is the source and which is the destination.

a) 8-bit Data Transfer:

MOV R_1, R_2

$(R_1) \leftarrow (R_2)$

Operation: MOV
Operands: R_2 (source)
R_1 (Destination)
Function: moves the contents of R_2 to R_1

Example:

MOV C, B
(C) ← (B)

moves the contents of B-register to C-register

b) 16-bit Data Transfer:

XCHG

(HL) ↔ (DE)

Operation: XCHG
Operands: No operands
Function: moves 16-bit address length transfer between two register pairs (DE, HL).

(H) ↔ (D)
(L) ↔ (E)

2. Immediate Addressing

- Allows the programmer to move a specific data contained within the instruction into a location/register.
- If a specific number is required in an instruction, it must start with a number symbol rather than a letter symbol. The hexadecimal number A0 which is 160 in decimal must be expressed as 0A0H. The H is added to indicate a hexadecimal number.

a) 8-bit immediate transfer:

MVI R, data

(R) ← data

Operation: MVI
Operands: data (source)
R (Destination)
Function: moves the data to R register

Example:

MVI A, 0FEH

(A) ← FEH or (A) ← 11111110
Sets the value of A-register to FEH

b) 16-bit immediate transfer:

LXI R_P, data

$(R_P) \leftarrow$ data

Operation: LXI
Operands: data (source)
R_P (Destination)
Function: moves the 16-bit data to 16-bit register pair R_P (BC, DE, HL)

R_p (BC): B
R_p (DE): D
R_p (HL): H

Example 1:

LXI H, 802DH
(H) ← 80H
(L) ← 2DH

Example 2:

LXI D, 0E62H
(D) ← 0EH
(E) ← 62H

Exercise1:

Write an assembly code to set A, B, C with the value FEH and HL with 71D3H and DE with A071H then exchange HL with DE

Solution:

```
MVI     A, 0FEH
MOV     B, A
MOV     C, B
LXI     H, 71D3H
LXI     D, 0A071H
XCHG
```

(A) ← FE
(B) ← (A)
(C) ← (B)
(HL) ← 71D3 ⇔ (H) ← 71 and (L) ← D3
(DE) ← A071 ⇔ (D) ← A0 and (E) ← 71
(DE) ↔ (HL)

A	FE			
B	FE		FE	C
D	~~A0~~ 71		~~71~~ D3	E
H	~~71~~ A0		~~D3~~ 71	L

Note: using register addressing is better than immediate addressing because it is faster and use less memory

3. Direct Addressing

- This addressing mode differs from register and immediate mode because it can read from and write to a memory location. Register and immediate only use locations within the CPU itself.
- The address of the memory location is specified in the instruction itself.
- Sometimes, it is called extended addressing because the memory address requires two bytes.

a) 8-bit Transfer

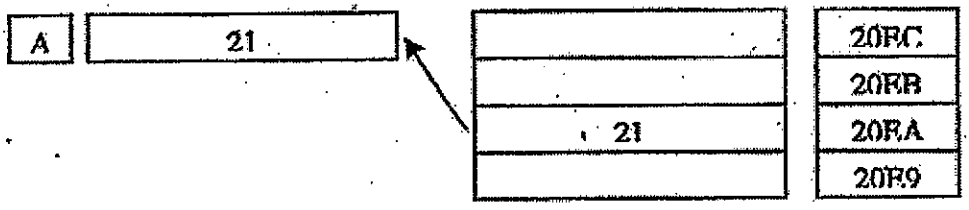
LDA address

(A) ← (address)

Operation: LDA
Operands: address (16-bits)
Function: loads A-register with the contents of the memory location specified by the address

Example:

LDA 20EAH
(A) ← (20EA)
(A) ← 21



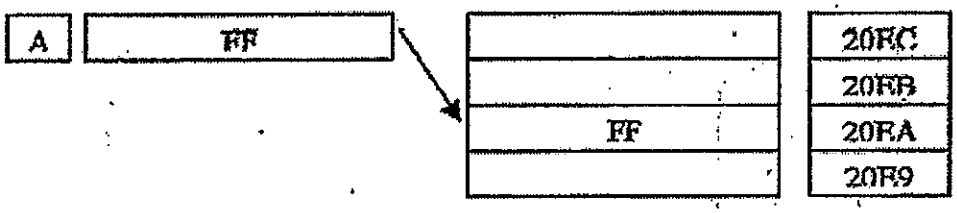
STA address

(address) ← (A)

Operation: STA
Operands: address (16-bit)
Function: stores the contents of A-register in the memory location specified by the address

Example:

STA 20EAH
(20EA) ← (A)
(20EA) ← FF



b) 16-bit Transfer

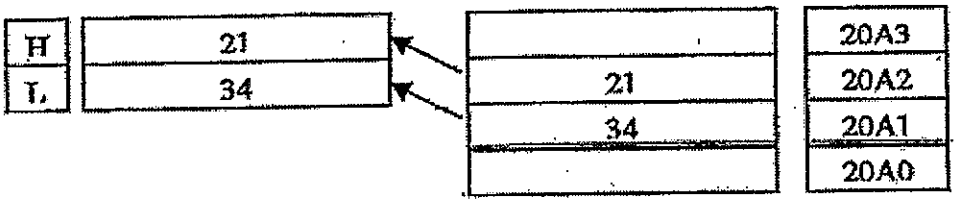
LHLD address

(L) ← (address)
(H) ← (address+1)

Operation: LHLD
Operands: address (16-bit)
Function: this instruction is used to load HL with the content of the memory locations (address, address+1), such that L is loaded with the content of memory location specified by the address and H is loaded with the contents of the next consecutive location (address+1).

Example:

LHLD 20A1H
(L) ← (20A1)
(H) ← (20A2)



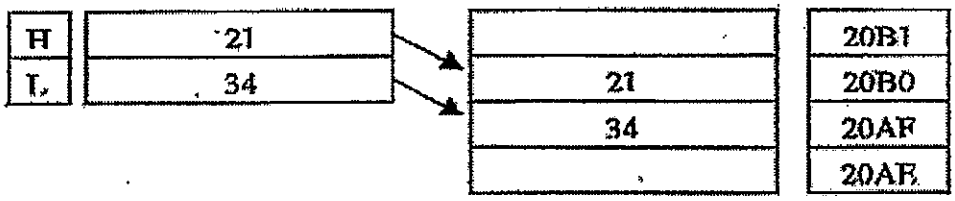
SHLD address

(address) ← (L)
(address+1) ← (H)

Operation: SHLD
Operands: address (16-bit)
Function: this instruction will store the low byte (L) in the first location (address) followed by the high byte (H) in the next location (address+1)

Example:

SHLD 20AFH
(20AF) ← (L)
(20B0) ← (H)



4. Indirect Addressing

- With direct addressing, only A or HL-pair can be used to store or load a value to or from memory using 8085 microprocessor. But using indirect mode, data may be transferred between any of the processor registers and the system memory.
- Indirect addressing uses the contents of a storage location to point to the required address.
- The HL-pair is referenced using the single term "M" for memory.
- The contents of the HL-pair are used to address a memory location
- When an internal register is used for indirect addressing, the addressing mode is termed Register indirect addressing.
- The 8085 allows immediate data to be moved into system memory locations. This operation is done directly through the HL-pair "M" and it is termed immediate register indirect memory addressing.

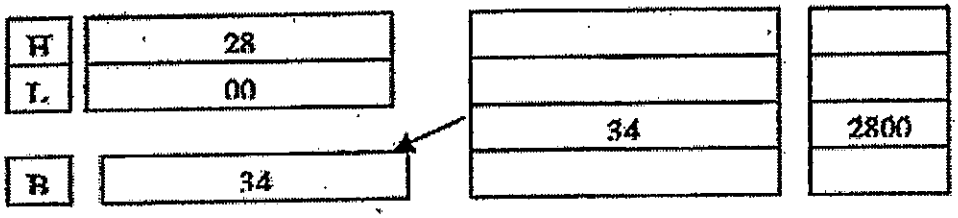
Register Indirect Addressing

MOV R, M

$$(R) \leftarrow ((HL))$$

Operation: MOV
Operands: (M) (source)
R (Destination)
Function: moves the contents of memory location specified by the HL-pair to R

Example: MOV B, M

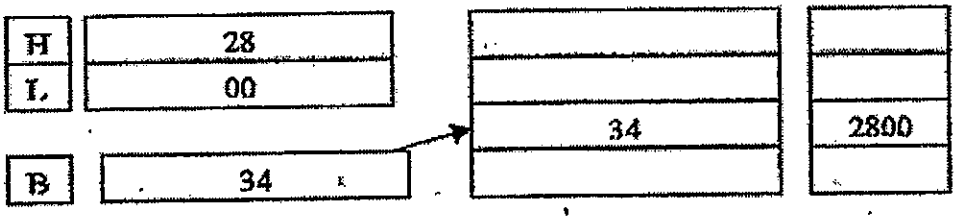


MOV M, R

$$((HL)) \leftarrow (R)$$

Operation: MOV
Operands: R (source)
(M) (Destination)
Function: moves the contents of register R to the memory location specified by the HL-pair

Example: MOV M, B



Immediate Register indirect addressing

MVI M, data

((HL)) ← data

Operation: MVI
Operands: data (source)
(M) (Destination)
Function: moves the data to the memory location specified by the address found in the HL-pair

Example:

MVI M, 0FEH
((HL)) ← FEH

LDAX R_p

(A) ← ((R_p))

Operation: LDAX
Operands: R_p
Function: this instruction loads the accumulator indirectly from a location pointed by the R_p (DE or BC only)

Example:

LDAX D
(A) ← ((DE))

STAX R_p

((R_p)) ← (A)

Operation: STAX
Operands: R_p
Function: this instruction stores the content of A into the memory location specified by the address stored in the R_p (BC or DE only)

Example:

STAX B
((BC)) ← (A)

- The last two indirect addressing LDAX and STAX use only the register pairs BC and DE

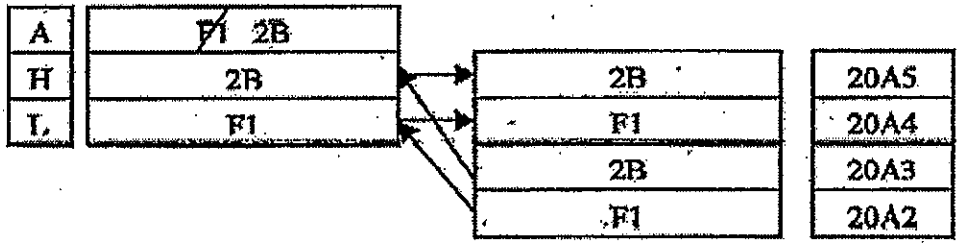
Exercise2:

Find the contents of the internal registers from the following code

```
MVI     A, 0F1H
STA     20A2H
MVI     A,02BH
STA     20A3H
LHLD    20A2H
SHLD    20A4H
```

Solution:

```
(A)     ← F1
(20A2)  ← (A)
(A)     ← 2B
(20A3)  ← (A)
(L)     ← (20A2)
(H)     ← (20A3)
(20A4)  ← (L)
(20A5)  ← (H)
```



The Assembly process

- The instructions that a microprocessor executes are stored in its memory as binary-coded numbers.
- The instructions must first be converted from assembly language to object code before any programs can be executed.
- Various programs are available to perform the generation of the object code. These programs called assemblers
- Each assembly instruction has between 1 to 3 bytes in hexadecimal form.
- The instructions that include registers only need one byte to represent it in hexadecimal form.
- Depending on the instruction, number of bytes will be needed.

Example:

MOV A,B	single byte	78
MVI A,0FEH	two bytes	3E, FE
STA 20F2H	three bytes	32, F2, 20

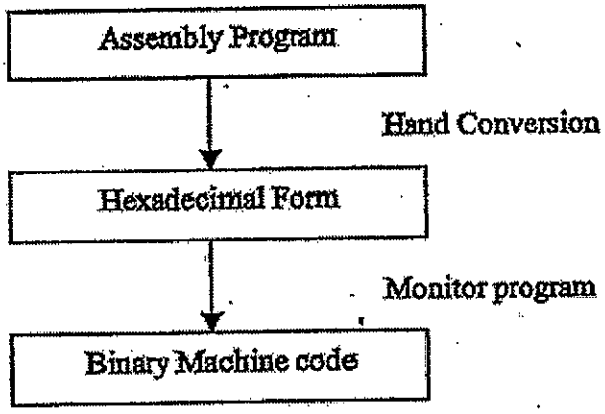
- The address data are loaded in low order byte, high order byte sequence.
- A manual assembly conversion could be done by hand code.
- Each instruction representation in hexadecimal is known in the appendix attached at the end of the note.

- For each instruction we can convert it to the hexadecimal code representation by the hand assembly coding form.

- The hand assembly coding form is represented as the following table

Line #	Memory		Label	Assembly			Comments
	ADDRESS	CONTENT		MNEMONIC	OP1	OP2	

- We noticed that each line or instruction in the assembly language could have from 1 to 3 bytes of addresses to store the content of the representation of that instruction.



Single Board System

- Many manufactures offer microprocessor trainers and dependent systems.
- A trainer is intended primary for educational use.
- A development system is normally intended for use as engineering tool during the process of designing a new product that uses a microprocessor.
- The heart of most development systems is some kind of single board computer. It also can serve as trainer.
- The advantage of using it, that the programmer feels free from the tasks required to build the actual microcomputer system.

Exercise3:

List the preceding program fragments in Exercise 2 in the hand assembly coding form

Solution:

Line #	Memory		Label	Assembly			Comment
	ADDRESS	CONTENT		MNEMONIC	OP1	OP2	
1	2800	3E		MVI	A	0FF	
	2801	FF					
2	2802	32		STA	20A2		
	2803	A2					
	2804	20					
3	2805	3E		MVI	A	0EE	
	2806	EE					
4	2807	32		STA	20A3		
	2808	A3					
	2809	20					
5	280A	2A		LHLD	20A2		
	280B	A2					
	280C	20					
6	280D	22		SHLD	20A4		
	280E	A4					
	280F	20					

محمد یاقوت
اساسیات

Microprocessor Fundamentals (246)

Handout # 3

Introduction

The programmer must be able to represent and manipulate microprocessor data in a number of different ways.

Data Representation

- There are 256 (00000000 to 11111111) unique possibilities for each word in a memory or in a one byte register.
- Data representation methods involve the specific coding used to give each word meaning.
- Data representation Methods:
    1. Binary Number
        - Unsigned Binary Number
        - Signed Binary Number
            - Sign/Magnitude
            - 2's complement
            - 1's complement
    2. Binary Coded Decimal (BCD)
    3. Other Forms
        - ASCII
        - Bit Mapped

Binary Number

Unsigned Binary Number

- All numbers are assumed to be a positive number
- A byte is simply the direct, 8-bit binary form of the number
- Each bit is usually given the name associated with its weighting

$$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0 \quad = \quad \textbf{weighting}$$

- Bit 0, in the most right is the Least significant bit (LSB)
- Bit 7, in the most left is the Most significant bit (MSB)
- The range of numbers is 256 (0 to 255)
- Provide very compact storage in memory since all possible states of a byte are used
- They are easy to manipulate and provide direct interface with a number of I/O devices.

Example 1:

$+16 = 00010000$

$+34 = 00100010$

Signed Binary Number

- Represents both positive and negative
- With signed form, one bit, the MSB, is used to indicate the sign of the number
- Three representation could be done:
  1. Sign/Magnitude
  2. 2's complement
  3. 1's complement

1. Sign/Magnitude Method

- MSB indicates the sign (0 for positives, and 1 for negative)
- Other seven bits indicate the magnitude

Example 2:

+26 = **0** 0011010
-100 = **1** 1100100

- 255 possibilities number ranging -127 to 127 (**11111111** to **01111111**)
- 00000000 or 10000000 are the same number zero
- Disadvantage: it is not possible to perform simple arithmetic operations on numbers and automatically produce correct signed numbers

Example 3:

+26 + (-100) = -74

00011010
+11100100

=11111110

Note: 11111110 equals -126 which is not equal to -74

2. 2's Complement

- MSB called S (if 0 means positives, 1 is negative)
- For positive number, the simple binary code is used to represent
- For positive numbers from zero to 127; the 2's complement and straight binary forms are the same
- For negative numbers, we get the positive number then we complement it (each bit is inverted) and the result is incremented by one

Example 4:

+15 = 00001111

(00001111)' = 11110000
11110000 + 1 = 11110001
⇒ -15 = 11110001

Example 5:

+88 = 01011000
(01011000)' + 1 = 10101000
⇒ -88 = 10101000

- We can do it in easy way, we retain all trailing zeros and the least significant 1 bit and complement all the rest of the bits

Example 6:

+76 = 01001 100

Complement Retain

⇒ -76 = 10110 100

- Total of 256 possible numbers -128 to 127 including zero
- We can now do simple signed arithmetic operation using 2's complement

Example 7:

-88	=	10101000
+15	=	00001111
-73	=	10110111

Example 8:

-73	=	10110111
+76	=	01001100
+3	=	00000011

- The carry will have no meaning for single byte manipulation, but it is important not to exceed the range -128 to +127

- If the result exceeds the range we should add more byte which represents the sign of the number.

Example 9:

+68	=	10101000
+76	=	00001111
+144	=	00000000 10010000

Example 10:

-68	=	10110100
-76	=	10111100
-144	=	11111111 01110000

3. 1's complement

- It is simply 2's complement without increment

Example 11:

+15 = 00001111

(00001111)' = 11110000

⇒ -15 = 11110000

Binary Coded Decimal (BCD)

- Four bits are required for each digit
- 8-bits may be used to store two BCD digits

Example 12:

1000 0110 = 86

Example 13:

0101 0001 = 51

- Allows only 100 possible numbers (0 to 99)
- Less efficient coding format
- No signed numbered in standard BCD representation

Other Forms of Data Representation

ASCII

- ASCII refer to the American Standard Code for Information Interchange
- Relates a single alphanumeric character to a byte
- Common for certain I/O devices, and most long distance transmission links (e.g.: phone connection)
- The standard ASCII character set consists of 128 decimal numbers ranging from zero through 127 assigned to letters, numbers, punctuation marks, and the most common special characters.

Example 14:

A = 65 = 01000001

a = 97 = 01100001

Bit Mapped

- The Byte is broken into individual bits
- Each bit in the byte would signify the on/off status

Microprocessor Fundamentals (246)

Handout # 4

Arithmetic Instruction

- The arithmetic instructions include add, subtract, Increment, decrement, and shift.

- The add and subtract instructions involve the A-register (the accumulator) and either another processor register or a memory location.

- The increment and decrement operations can generally be performed on various registers and locations through the use of different addressing modes.

- The microprocessor contains number of status bits called flags.

    - S Set if result is negative (MSB of A is 1)
    - Z Set if result is zero (A is all 0s)
    - AC Set if carry is generated from bit 3 (for BCD use)
    - CY Set if carry is from bit 7 (carry-out of MSB)
    - P Set if result is even parity

- Flags are either set or reset, depending on a particular instruction.

- The programmer is able to use these flags as an aid in data manipulation.

The arithmetic instructions add, subtract, increment, or decrement data in registers or memory.

ADD	Add to Accumulator
ADI	Add Immediate Data to Accumulator
ADC	Add to Accumulator Using Carry Flag
ACI	Add immediate data to Accumulator Using Carry
SUB	Subtract from Accumulator
SUI	Subtract Immediate Data from Accumulator
SBB	Subtract from Accumulator Using Borrow (Carry) Flag
SBI	Subtract Immediate from Accumulator Using Borrow (Carry) Flag
INR	Increment Specified Byte by One
DCR	Decrement Specified Byte by One
INX	Increment Register Pair by One
DCX	Decrement Register Pair by One
DAD	Double Register Add; Add Content of Register

Add Instruction

- The sum can be only 0 or 1.
- A carry will often be generated
- The carry from the previous pair is termed the carry-in bit.
- The datum produced as a carry to the next higher order is called the carry-out bit.
- Addition of Two bits

Bit 1	Bit 0	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- Addition of Two bits and a Carry-in

Bit 1	Bit 0	Carry-in	Sum	Carry-out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Register addressing

ADD R

(A) ← (A) + (R)

Operation: ADD
Operands: R
Function: the contents of R- register is added to the contents of A-register and the result is placed in the A-register.
Affected Flags: all flags

Example:

ADD B
(A) ← (A) + (B)

Immediate Addressing

ADI data

(A) ← (A) + data

Operation: ADI
Operands: 8-bits data
Function: the contents of A- register is added to immediate value 8-bits data and the result is placed in the A-register.
Affected Flags: all flags

Example:

ADI 0FH
(A) ← (A) + 0F

Register Indirect Addressing

ADD M

$$(A) \leftarrow (A) + ((HL))$$

Operation: ADD
Operands: M
Function: the contents of the memory location addressed by the HL-pair register is added to the contents of A-register and the result is placed in the A-register.
Affected Flags: all flags

Example:

ADD M

$$(A) \leftarrow (A) + ((HL))$$

Subtract Instruction

- Subtraction of Two bits and a Barrow-in

Bit 1	Bit 0	Borrow-in	Difference	Borrow-out
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Register addressing

SUB R

$$(A) \leftarrow (A) - (R)$$

Operation: SUB
Operands: R
Function: the contents of R- register is subtracted from the contents of A-register and the result is placed in the A-register.
Affected Flags: all flags (the carry flag is used now for borrow)

Example:

SUB B

$$(A) \leftarrow (A) - (B)$$

Immediate Addressing

SUI data

(A) ← (A) - data

Operation: SUI
Operands: 8-bits data
Function: the immediate value 8-bits data is subtracted from the contents of A-register and the result is placed in the A-register.
Affected Flags: all flags (the carry flag is used now for borrow)

Example:

SUI 0FH
(A) ← (A) - 0F

Register Indirect Addressing

SUB M

(A) ← (A) - ((HL))

Operation: SUB
Operands: M
Function: the contents of the memory location addressed by the HL-pair register is subtracted from the contents of A-register and the result is placed in the A-register.
Affected Flags: all flags (the carry flag is used now for borrow)

Example:

SUB M
(A) ← (A) - ((HL))

Increment Instruction

- The increment instruction is adding one to the contents of a location, which is either a register or memory location.

Register addressing

- 8-bit increment

INR R

$$(R) \leftarrow (R) + 1$$

Operation: INR
Operands: R
Function: the contents of R- register is incremented by one.
Affected Flags: all except the carry flag

Example:

INR A

$$(A) \leftarrow (A) + 1$$

- 16-bit increment

INX R_P

$$(R_P) \leftarrow (R_P) + 1$$

Operation: INX
Operands: R_P
Function: the contents of R_P-register (16-bit) is incremented by one.
Affected Flags: no flags

Example:

INX D

$$(DE) \leftarrow (DE) + 1$$

Register Indirect Addressing

INR M

$$((HL)) \leftarrow ((HL)) + 1$$

Operation: INR
Operands: M
Function: the contents of the memory location addressed by the HL-pair register is incremented by one.
Affected Flags: all except the carry flag

Example:

INR M
$((HL)) \leftarrow ((HL)) + 1$

Decrement Instruction

- The decrement instruction is subtracting one from the contents of a location, which is either a register or memory location.

Register addressing

- 8-bit decrement

DCR R

(R) ← (R) - 1

Operation: DCR
Operands: R
Function: the contents of R- register is decremented by one.
Affected Flags: all except the carry flag

Example:

DCR A
(A) ← (A) - 1

- 16-bit decrement

DCX R_P

(R_P) ← (R_P) - 1

Operation: DCX
Operands: R_P
Function: the contents of R_P-register (16-bit) is decremented by one.
Affected Flags: no flags

Example:

DCX D
(DE) ← (DE) - 1

Register Indirect Addressing

DCR M

((HL)) ← ((HL)) - 1

Operation: DCR
Operands: M
Function: the contents of the memory location addressed by the HL-pair register is decremented by one.
Affected Flags: all except the carry flag

Example:

DCR M
((HL)) ← ((HL)) - 1

Exercise1: (Unsigned arithmetic)
Find the contents of the processor registers after the following program is run.

```
MVI     A, 91H
MVI     B, 3AH
ADD     B
SBI     48H
DCR     A
```

Solution:

(A) ← 91
(B) ← 3A
(A) ← (A) + (B)

```
    91
+   3A
----
    CB
```

(A) ← (A) – 48

```
    CB
-   48
----
    83
```

(A) ← (A) - 01

```
    83
-   01
----
    82
```

A	~~91~~ ~~CB~~ ~~83~~ 82
B	3A

Exercise2: (Signed arithmetic)
Find the contents of the processor registers after the following program is run.

```
MVI     A, 23H
MVI     B, 0B8H
ADD     B
SBI     0DBH
DCR     A
```

Solution:

(A) ← 23
(B) ← B8
(A) ← (A) + (B)

```
    23
+   B8
----
    DB
```

(A) ← (A) – DB

```
    DB
-   DB
----
    0 0
```

(A) ← (A) - 01

```
    00
-   01
----
    FF
```

A	~~23~~ ~~DB~~ ~~00~~ FF
B	B8

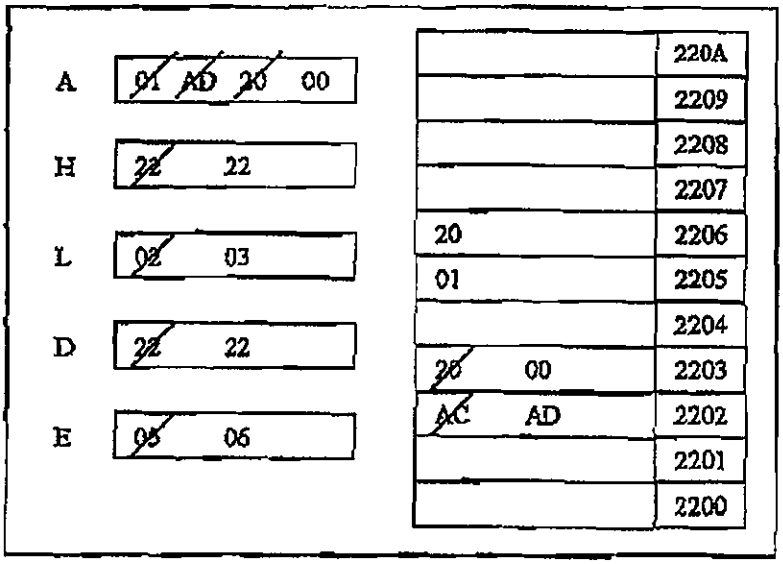
Exercise3:
Write an assembly program to do the following addition and subtraction operations.

$$(2202H) \leftarrow (2202H) + (2205H)$$
$$(2203H) \leftarrow (2203H) - (2206H)$$

Assume that the memory locations have the following data:
(2202H) = ACH (2203H) = 20H (2205H) = 01H (2206H) = 20H

Solution 1:

```
LXI     H, 2202H
LXI     D, 2205H
LDAX    D
ADD     M
MOV     M, A
INX     H
INX     D
LDAX    D
SUB     M
MOV     M, A
```

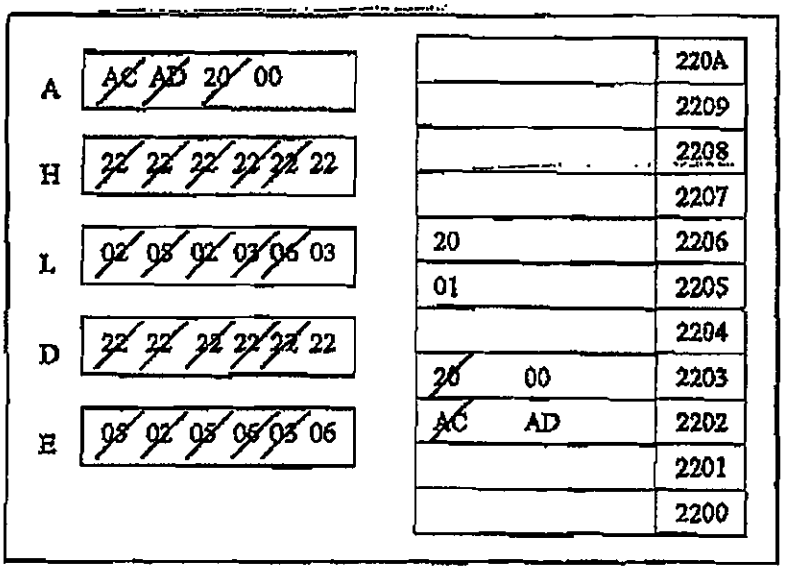


Exercise3:

Assume that the memory locations have the following data:
(2202H) = ACH (2203H) = 20H (2205H) = 01H (2206H) = 20H

Solution 2:

```
LXI     H, 2202H
LXI     D, 2205H
MOV     A, M
XCHG
ADD     M
XCHG
MOV     M, A
INX     H
INX     D
MOV     A, M
XCHG
SUB     M
XCHG
MOV     M, A
```



Multiprecision Arithmetic Instruction

- Some Application requires more than 8 bits for data representation.
- Most 8-bit microprocessors, including the 8085, provide a number of arithmetic instructions for manipulating numbers of more than 8-bits.

16-bit Arithmetic

DAD R_p

$$(HL) \leftarrow (HL) + (R_p)$$

Operation: DAD
Operands: R_p
Function: the contents of the Register-pair is added to the contents of HL-register pair and the result is placed in the register pair HL.
Affected Flags: only the carry flag

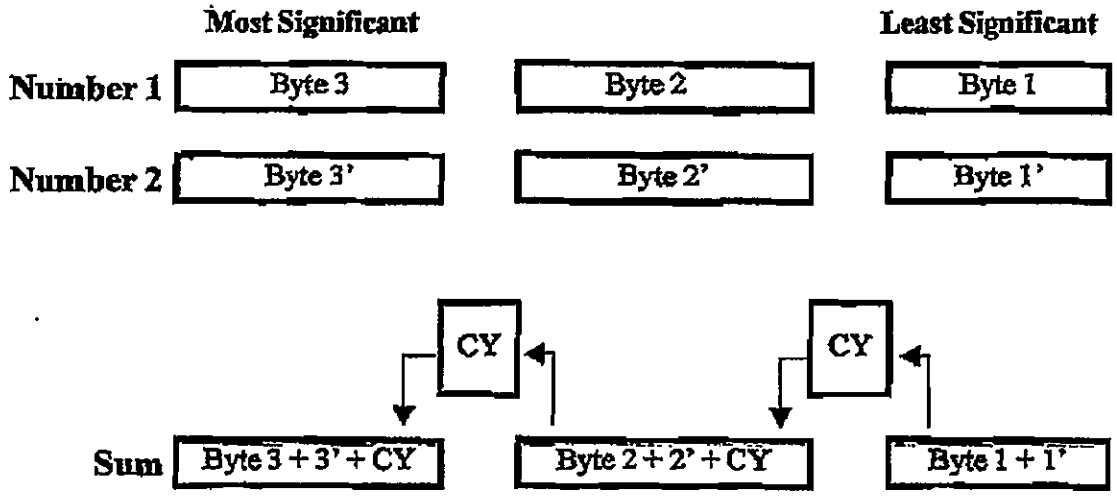
Example:

DAD B

$$(HL) \leftarrow (HL) + (BC)$$

Multiprecision addition/subtraction – the Carry Flag

- If more than 16-bit addition is required, a number of instructions must be used to fulfill this operation.
- For these multiprecision addition operations it is necessary to add the carry bit (CY) when adding together the second and third pairs of bytes.
- The 8085 provides additional add, subtract, instructions that use the carry, or borrow, bit.
- The Figure down illustrate the Multiprecision arithmetic where we add two 24-bit numbers



Multiprecision Addition :

- Register Addressing

ADC R

(A) ← (A) + (R) + (CY)

Operation: ADC
Operands: R
Function: the contents of R- register is added to the contents of A-register, in addition the carry bit is added to the LSB of this sum. The result is placed in the A-register.
Affected Flags: all flags

Example:

ADC B
(A) ← (A) + (B) + (CY)

- Register Indirect Addressing

ADC M

(A) ← (A) + ((HL)) + (CY)

Operation: ADC
Operands: M
Function: the contents of the memory location addressed by the HL-pair is added to the contents of A-register, in addition the carry bit is added to the LSB of this sum. The result is placed in the A-register.
Affected Flags: all flags

Example:

ADC M
(A) ← (A) + ((HL)) + (CY)

- Immediate Addressing

ACI data

(A) ← (A) + data + (CY)

Operation: ACI
Operands: 8-bits data
Function: the contents of A- register is added to immediate value 8-bits data, in addition the carry bit is added to the LSB of this sum. The result is placed in the A-register.
Affected Flags: all flags

Example:

ACI 0AEH
(A) ← (A) + AE + (CY)

Multiprecision Subtraction

- Register addressing

SBB R

(A) ← (A) - (R) – (CY)

Operation: SBB
Operands: R
Function: the contents of R- register is subtracted from the contents of A-register along with the borrow bit (CY) and the result is placed in the A-register.
Affected Flags: all flags (the carry flag is used now for borrow)

Example:

SBB B
(A) ← (A) - (B) – (CY)

- Register Indirect Addressing

SBB M

(A) ← (A) - ((HL)) – (CY)

Operation: SBB
Operands: M
Function: the contents of the memory location addressed by the HL-pair register is subtracted from the contents of A-register along with the borrow bit (CY) and the result is placed in the A-register.
Affected Flags: all flags (the carry flag is used now for borrow)

Example:

SBB M
(A) ← (A) - ((HL)) – (CY)

- **Immediate Addressing**

SBI data

(A) ← (A) – data – (CY)

Operation: SBI
Operands: 8-bits data
Function: the immediate value 8-bits data is subtracted from the contents of A-register along with the borrow bit (CY) and the result is placed in the A-register.
Affected Flags: all flags (the carry flag is used now for borrow)

Example:

SBI 0FH
(A) ← (A) - 0F – (CY)

Exercise4:
Assume we have two 24-bits numbers to be added as in the figure Page# 14. these numbers are stored in the memory locations 20C0 H and 20D0H. assume that the Least significant are stored the lowest byte of each address. The result should be placed in the address 20C0h. The assembly language code for the this process..

```
LXI     H, 20C0H
LXI     D, 20D0H
LDAX    D
ADD     M           ; LOW-BYTE ADDITION
MOV     M, A
INX     H
INX     D
LDAX    D
ADC     M           ; 2ND BYTE ADDITION
MOV     M, A
INX     H
INX     D
LDAX    D
ADC     M           ; HIGH-BYTE ADDITION
MOV     M, A
```

BCD Arithmetic

- Most microprocessors provide instructions that allow BCD data to be manipulated.
- The 8085 uses a two-stage process for BCD arithmetic:
  - First, a straight binary operation (i.e. addition, subtraction...)
  - Second, Adjust the result number into its proper BCD format.
- We will use the AC flag and the accumulator to make the required adjustment.
- As we knew that the BCD numbers are between 0 to 99 for an 8-bit data.
- So, We may have cases when we do the normal arithmetic operations:
  - The binary sum and BCD sum are the same. So no correction is needed. The AC flag is set if there is a carry-out of bit-3.

    Example 1:

    Number 1 = 0110 0010 (BCD) = 62 (decimal)
    + Number 2 = 0010 0101 (BCD) = 25 (decimal)
    Binary Sum = 1000 0111 (CY = 0, AC = 0)
    BCD Sum = 1000 0111 (BCD) = 87 (decimal)

  - The sum for the low 4-bit is greater than 9 and no carry-out of bit-3. This needs adjustment.

    Example 2:

    Number 1 = 0111 1001 (BCD) = 79 (decimal)
    + Number 2 = 0001 0110 (BCD) = 16 (decimal)
    Binary Sum = 1000 1111 (CY = 0, AC = 0)
    BCD Sum = 1001 0101 (BCD) = 95 (decimal)

- The sum for the low 4-bit is less than 9 and there is carry-out of bit-3. This needs adjustment.

  Example 3:

  Number 1 = 0011 1001 (BCD) = 39 (decimal)
  + Number 2 = 0100 1000 (BCD) = 48 (decimal)
  Binary Sum = 1000 0001 (CY = 0, AC = 1)
  BCD Sum = 1000 0111 (BCD) = 87 (decimal)

- The BCD sum operation should be as follows:

  - For each 4-bit group produce an answer that is less than 9 and no AC is generated, then the result is correct as BCD number.

  - If the result is greater than 9 or an AC is generated, an adjustment is needed which is adding six to the normal binary sum.

  ⇒ the adjustment for Example 2:

  Binary Sum = 1000 1111 (CY = 0, AC = 0)
  Add + 6 = 0000 0110
  Adjusted Sum = 1001 0101 (BCD) = 95 (decimal)

  ⇒ the adjustment for Example 3:

  Binary Sum = 1000 0001 (CY = 0, AC = 1)
  Add + 6 = 0000 0110
  Adjusted Sum = 1000 0111 (BCD) = 87 (decimal)

- The 8085 provides an adjustment instruction that acts to adjust both the low- and the high-order digits of the BCD number.

DAA

Operation: DAA
Operands: no operands
Function:
1. if the value of the least significant four bits of the A-register is greater than 9 or if the AC flag is set, six is added to the A-register.
2. if the value of the most significant four bits of the A-register is now greater than 9 or if the CY-flag is set, six is added to the most significant for bits of the A register.

Affected Flags: all flags

Logic Instruction

Logical AND

Bit 1	Bit 2	Bit 1 AND Bit 2
0	0	0
0	1	0
1	0	0
1	1	1

- The logical AND instructions perform the bit-by-bit AND operation between the contents of the A-register and:
  - Immediate data
  - The contents of another processor register
  - Memory location depending on the addressing mode.

Immediate Addressing

ANI data

(A) ← (A) AND data.

Operation: ANI
Operands: 8-bits data
Function: the immediate value 8-bits data masks the contents of the A-register. The result is placed in the A-register.
Affected Flags: all flags (with CY and AC equal zero always)

Example:

ANI 40H
(A) ← (A) AND 40

Register Addressing

ANA R

(A) ← (A) AND (R)

Operation: ANA
Operands: R
Function: the contents of register R masks the contents of the A-register. The result is placed in the A-register.
Affected Flags: all flags (with CY and AC equal zero always)

Example:

ANA B
(A) ← (A) AND (B)

Register Indirect Addressing

ANA M

(A) ← (A) AND ((HL))

Operation: ANA
Operands: M
Function: the content of the memory location addressed by the HL-pair register masks the contents of the A-register. The result is placed in the A-register.
Affected Flags: all flags (with CY and AC equal zero always)

Example:

ANA M
(A) ← (A) AND ((HL))

Logical OR

Bit 1	Bit 2	Bit 1 OR Bit 2
0	0	0
0	1	1
1	0	1
1	1	1

Immediate Addressing

ORI data

(A) ← (A) OR data

Operation: ORI
Operands: 8-bits data
Function: the immediate value 8-bits data sets the selected bits of the A-register without disturbing the others bits. The result is placed in the A-register.
Affected Flags: all flags (with CY and AC equal zero always)

Example:

ORI 40H
(A) ← (A) OR 40

Register Addressing

ORA R

(A) ← (A) OR (R)

Operation: ORA
Operands: R
Function: the contents of register R sets the selected bits of the A-register without disturbing the others bits. The result is placed in the A-register.
Affected Flags: all flags (with CY and AC equal zero always)

Example:

ORA B
(A) ← (A) OR (B)

Register Indirect Addressing

ORA M

(A) ← (A) OR ((HL))

Operation: ORA
Operands: M
Function: the content of the memory location addressed by the HL-pair register sets the selected bits of the A-register without disturbing the others bits. The result is placed in the A-register.
Affected Flags: all flags (with CY and AC equal zero always)

Example:

ORA M
(A) ← (A) OR ((HL))

Exclusive-OR Operation

Bit 1	Bit 2	Bit 1 XOR Bit 2
0	0	0
0	1	1
1	0	1
1	1	0

Immediate Addressing

XRI data

(A) ← (A) XOR data

Operation: XRI
Operands: 8-bits data
Function: Exclusive-OR operation as shown in the table above between the immediate 8-bit data and the A-register. The result is placed in the A-register.
Affected Flags: all flags (with CY and AC equal zero always)

Example:

XRI 40H
(A) ← (A) XOR 40

Register Addressing

XRA R

(A) ← (A) XOR (R)

Operation: XRA
Operands: R
Function: Exclusive-OR operation as shown in the table above between the content of the R-register and the A-register. The result is placed in the A-register.
Affected Flags: all flags (with CY and AC equal zero always)

Example:

XRA B
(A) ← (A) XOR (B)

Register Indirect Addressing

XRA M

(A) ← (A) XOR ((HL))

Operation: XRA
Operands: M
Function: Exclusive-OR operation as shown in the table above between the content of the memory location addressed by the HL-pair register and the A-register. The result is placed in the A-register.
Affected Flags: all flags (with CY and AC equal zero always)

Example:

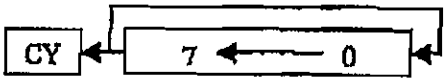
XRA M
(A) ← (A) XOR ((HL))

Rotate/Shift

- The logical group contains instructions to move the bits within a byte.
- The primary forms of this operation are the rotate and shift functions.
- The 8085 does not provide the shift operation; however, it does provide two forms of rotate in two direction.

Rotate

RLC

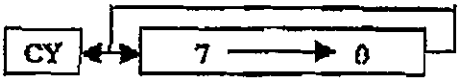


Operation: RLC
Operands: No operand
Function: Rotate left the accumulator A-register operation causes each bit in the accumulator to move to the next higher-order location. The MSB is rotated back to the now-vacant LSB position; it is also moved into CY flag bit.
Affected Flags: only CY

Example

Before RLC	0	10110100
After RLC	1	01101001

RRC



Operation: RRC
Operands: No operand
Function: Rotate right the accumulator A-register operation causes each bit in the accumulator to move to the next lower-order location. The LSB is rotated to the now-vacant MSB position; it is also moved into CY flag bit.
Affected Flags: only CY

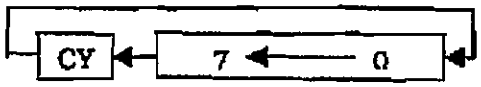
Example

Before RRC	0	10110100
After RRC	0	01011010

Rotate through carry

- There are instruction where the rotate operation includes the CY flag in the rotation path like RAL and RAR instructions.

RAL

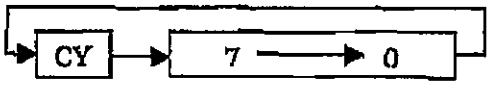


Operation: RAL
Operands: No operand
Function: Rotate left the accumulator A-register operation through carry causes the rotate operation to include the CY-flag in the rotation path.
Affected Flags: only CY

Example

Before RAL	0	10110100
After RAL	1	01101000

RAR



Operation: RAR
Operands: No operand
Function: Rotate right the accumulator A-register operation through carry causes the rotate operation to include the CY-flag in the rotation path.
Affected Flags: only CY

Example

Before RAR	0	10110100
After RAR	0	01011010

Shift

- If the upper or lower bit is not rotated back to opposite end of the byte, we have the general form of a sift instruction.
- A shift left causes a zero to be entered into the LSB, with MSB pushed into the CY flag, note that the old value of CY flag is lost.
- The shift operation is a multiplication of 2 (shift left) or division by two (shift right).
- The shift right operation causes a division by two with any reminder appearing in the carry bit.

Exercise5:

Number	Rotate Left (RLC)	Rotate Right (RRC)	Rotate Left through Carry (RAL)	Rotate Right through Carry (RAR)	Shift Left	Shift Right
11001010 and CY = 0	10010101	01100101	10010100	01100101	10010100	01100101
	CY = 1	CY = 0	CY = 1	CY = 0	CY = 1	CY = 0
01110001 and CY = 0	11100010	10111000	11100010	00111000	11100010	00111000
	CY = 0	CY = 1	CY = 0	CY = 1	CY = 0	CY = 1

Compare Instruction

- Compare instructions provide the keystone of computer decision-making operations.
- Most decision instructions cause one or another result based on the status of the various flags.
- To select a particular option based on a byte reaching a given value, we might subtract the byte from the value and make the decision based on the status of the zero flag:
  - If byte and the value are the same, the zero flag will be set.
  - If not, the zero flag will be a zero.
- The subtraction operation causes one of the words to be lost.
- The compare instructions set the various flags as though a subtraction and occurred without actually completing the subtraction.
- The Z-flag is set to one if the contents are the same
- The CY-flag is set to one if the contents of A are less than the other value.

Register addressing

CMP R

(A) - (R)

Operation: CMP
Operands: R
Function: The contents of the R-register are compared with the contents of the A-register.
Affected Flags: all flags

Register Indirect addressing

CMP M

(A) - ((HL))

Operation: CMP
Operands: M
Function: The contents of the memory location addressed by the HL-register pair are compared with the contents of the A-register.
Affected Flags: all flags

Immediate addressing

CPI data

(A) - data

Operation: CPI
Operands: 8-bit data
Function: The immediate 8-bit value is compared with the contents of the A-register.
Affected Flags: all flags

Other Operations

- The 8085 instruction set includes three other operations that may or may not be available with other processors.
- None of the three affects any flags other than the carry or requires no operand.

Complement A-register

CMA

(A) ← (A)'

Operation: CMA
Operands: no operand
Function: This causes a bit-by-bit complement of the accumulator A-register.
Affected Flags: no flags

Complement the carry flag

CMC

(CY-flag) ← (CY-flag)'

Operation: CMC
Operands: no operand
Function: complements the carry flag

Set the carry flag

STC

(CY-flag) ← 1

Operation: STC
Operands: no operand
Function: This sets the CY-flag (CY) = 1

Exercise6:
Write an assembly routine that help the programmer to do the shift left operation.

Solution 1:

```
Step1    RLC          ; Rotate left
Step2    ANI 0FEH     ; Reset the LSB (LSB = 0)
```

Example:

Assume (A) = D5 H

After Step1: (A) = AB H , (CY) = 1
After Step2: (A) = (A) AND FE
(A) = 1010 1011 AND 1111 1110
(A) = 1010 1010
(A) = AA H

Solution 2:

```
Step1    STC          ; Set CY-Flag
Step2    CMC          ; Complement CY-flag
Step3    RAL          ; Rotate Left through the carry
```

Example:

Assume (A) = D5 H

After Step1: (CY) = 1
After Step2: (CY) = 0
After Step3: (A) = AA H

Microprocessor Fundamentals (246)

Handout # 5

Transfer of control

- One of the great powers of modern computers is their ability to adapt their operation based on input data and computed operation.
- The transfer control function is implemented by instructions that shift execution to alternate routines.

Jump operations/Labels

Unconditional Jump

- A jump instruction is used to break normal sequential program execution.
- The program counter (PC) usually steps one by one through instructions in memory.
- When a jump instruction is encountered, execution is moved to another part of the program.
- This is done by replacing the two bytes in the PC with the address of the start of the new routine.
- The processor is thus forced to fetch the contents of this new location for the next instruction.
- The PC then is incremented.

JMP address

(PC) ← address

Operation: JMP
Operands: 16-bit address
Function: The immediate 16-bit address is placed with the contents of the PC-register.

JMP Label

(PC) ← (Label)

Operation: JMP
Operands: Label
Function: The address of the label is placed with the contents of the PC-register.

Conditional Jump

- Conditional Jumps cause the processor to branch to a new routine only if a specific condition is satisfied; otherwise, the PC is not changed and executed continues with no branch occurring.

Op code	Condition	Flag Status
JNZ	Not zero	Z=0
JZ	Zero	Z=1
JNC	No carry	C=0
JC	Carry	C=1
JPO	Parity odd	P=0
JPE	Parity even	P=1
JP	Plus	S=0
JM	Minus	S=1

Exercise 1:

Write a program that compares the elements of two arrays X(i), and y(i). Each array contains 5 8-bit numbers. The comparison is to be done by comparing the corresponding elements of the two arrays until either two elements are found to be unequal or all elements of the arrays have been compared and found to be equal. Assume that the arrays start at A000H and B000H respectively. If the two arrays are found to be unequal, save the address of the first unequal element of X(i) in memory location C000H ; otherwise, write all zeros.

```
                MVI     C, 05H
                LXI     H, 0A000H
                LXI     D, 0B000H
GO_ON           LDAX    D
                CMP     M
                JNZ     MIS_MATCH
                INX     H
                INX     D
                DCR     C
                JNZ     GO_ON
                MVI     A, 00H
                STA     0C000H
                STA     0C001
                JMP     DONE
MIS_MATCH       SHLD    0C000H
DONE
```

Exercise 2:

Given an array X(i) of 10 8-bit signed number stored in memory starting at A000H, write a program to generate two arrays from the given array such that P(j) consists of all positive numbers and the other N(k) consists of all the negative numbers. Store the arrays of positive numbers in memory starting at B000H and the array of negative numbers starting at C000H.

```
                LXI     SP, 000AH
                LXI     H, 0A000H
                LXI     D, 0B000H
                LXI     B, 0C000H
GO_ON           MOV     A, M
                CPI     00H
                JS      NEGATIVE
POSITIVE        STAX    D
                INX     D
                INX     H
                DCX     SP
                JNZ     GO_ON
                JMP     DONE
NEGATIVE        STAX    B
                INX     B
                INX     H
                DCX     SP
                JNZ     GO_ON
                JMP     DONE
DONE
```

Subroutines

- A subroutine is a section of code designed to perform a subtask and then return control to main instruction sequence.

- A subroutine execution, called a subroutine call, must allow two critical functions:
  - A transfer of control to subroutine
  - A way to return back to the original routine, called calling routine.

- The advantages of subroutines:
  - Save program memory
  - Provide the opportunity to construct a program.

- The two primary instructions provided by the 8085 to call a subroutine and return from it are:

  - **CALL ADDR/LABEL**
  - **RET**

Example: Short Delay

```
        CALL    SDLY

SDLY    DCR     C
        JNZ     SDLY
        RET
```

Input/Output

- The 8085 and a number of other microprocessors provide specific locations for I/O data.
- This I/O-mapped I/O gives a group of locations, 256 locations for 8085, and numbered zero to 255 that can provide the CPU with data or can accept output data from CPU.
- **Input Instruction**

IN location

(A) ← (location)

Operation: IN
Operands: location
Function: the byte at the input location is moved to the accumulator

- **Output Instruction**

OUT location

(location) ← (A)

Operation: OUT
Operands: Location
Function: the byte in the accumulator is moved to the output location

Exercise 3:

Write an assembly language program to sum together the even numbers from zero to twenty.

```
        MVI     A, 00H
        MVI     B, 02H
        MVI     C, 0AH
LOOP    ADD     B
        INR     B
        INR     B
        DCR     C
        JNZ     LOOP
```