Chapter-2

Programming with 8085 microprocessor

Internal Architecture of 8 bit microprocessor and its registers:

The Intel 8085 A is a complete 8 bit parallel central processing unit. The main components of 8085A are array of registers, the arithmetic logic unit, the encoder/decoder, and timing and control circuits linked by an internal data bus. The block diagram is shown below:

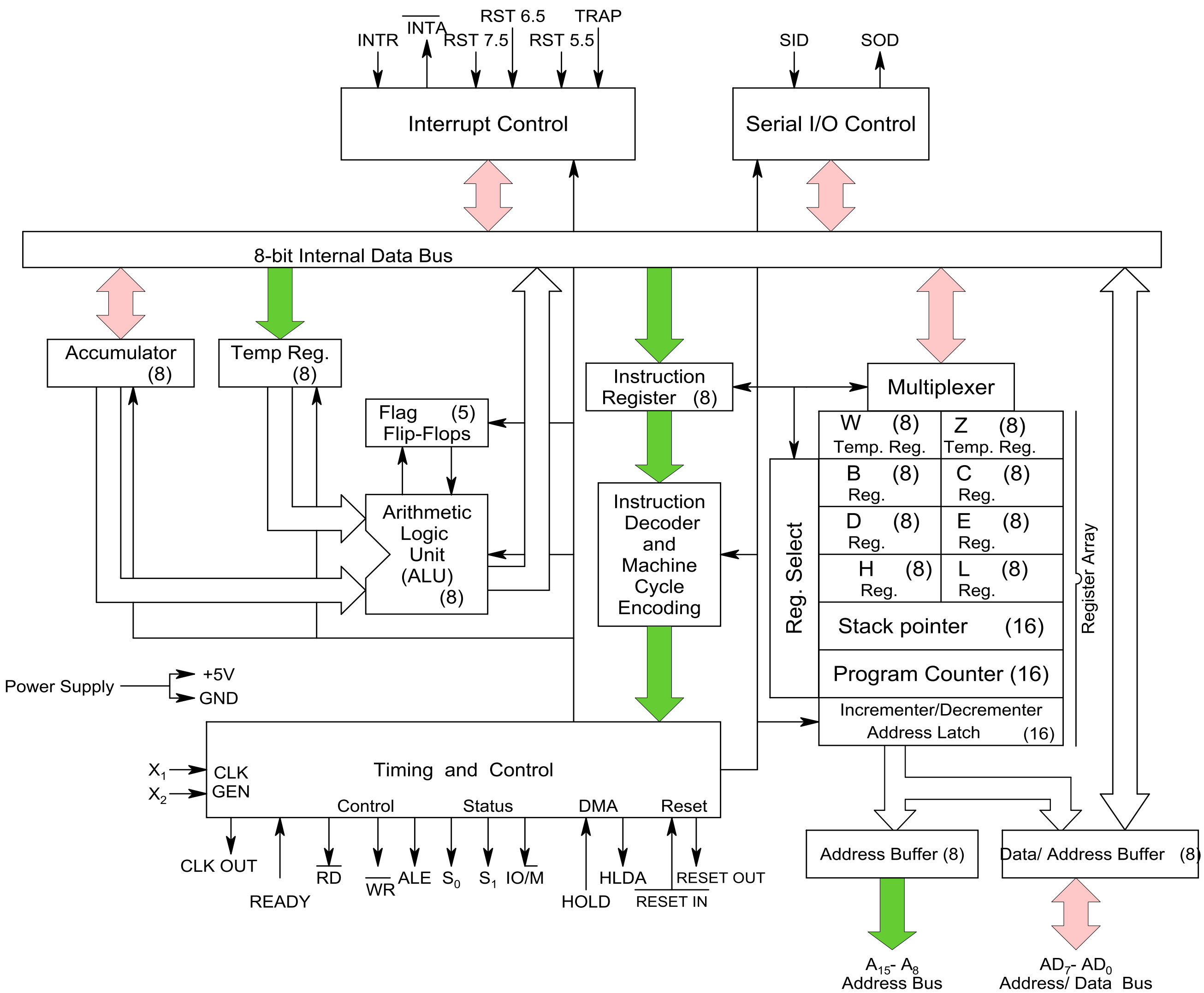


Fig: The 8085A microprocessor Functional Block Diagram

Source: Intel Corporation. *Embedded Microprocessors (Santa* Clara. Calif: Author.1994) pp 1-11

1: ALU:- The arithmetic logic unit performs the computing functions, it includes the accumulator, the temporary register, the arithmetic and logic circuits and five flags. The temporary register is used to hold data during an arithmetic/logic operation. The result is stored in the accumulator; the flags (flip-flops) are set or reset according to the result of the operation.

2. Accumulator (register A): It is an 8 bit register that is the part of ALU. This register is used to store the 8-bit data and to perform arithmetic and logic operations and 8085 microprocessor is called accumulator based microprocessor. When data is read from input port, it first moved to accumulator and when data is sent to output port, it must be first placed in accumulator.

3. Temporary registers(W & Z): They are 8 bit registers not accessible to the programmer. During program execution, 8085A places the data into it for a brief period.

4. Instruction register(IR): It is a 8 bit register not accessible to the programmer. It receives the operation codes of instruction from internal data bus and passes to the instruction decoder which decodes so that microprocessor knows which type of operation is to be performed.

5. Register Array: (Scratch pad registers B, C, D, E): It is a 8 bit register accessible to the programmers. Data can be stored upon it during program execution. These can be used individually as 8-bit registers or in pair BC, DE as 16 bit registers. The data can be directly added or transferred from one to another. Their contents may be incremented or decremented and combined logically with the content of the accumulator.

Register H & L: - They are 8 bit registers that can be used in same manner as scratch pad registers.

Stack Pointer (SP): - It is a 16 bit register used as a memory pointer. It points to a memory location in R/W memory, called the stack. The beginning of the stack is defined by loading a 16-bit address in the stack pointer.

Program Counter (PC): - Microprocessor uses the PC register to sequence the execution of the instructions. The function of PC is to point to the memory address from which the next byte is to be fetched. When a byte is being fetched, the PC is incremented by one to point to the next memory location.

6. Flags:

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

Register consists of five flip flops, each holding the status of different states separately is known as flag register and each flip flop are called flags. 8085A can set or reset one or more of the flags and are sign(S), Zero (Z), Auxiliary Carry (AC) and Parity (P) and Carry (CY). The state of flags indicates the result of arithmetic and logical operations, which in turn can be used for decision making processes. The different flags are described as:

- **Carry**: - If the last operation generates a carry its status will 1 otherwise 0. It can handle the carry or borrow from one word to another.
- **Zero:** - If the result of last operation is zero, its status will be 1 otherwise o. It is often used in loop control and in searching for particular data value.
- **Sign: -** If the most significant bit (MSB) of the result of the last operation is 1 (negative), then its status will be 1 otherwise 0.
- **Parity:** - If the result of the last operation has even number of 1's (even parity), its status will be 1 otherwise 0.
- **Auxiliary carry: -** If the last operation generates a carry from the lower half word (lower nibble), its status will be 1 otherwise 0. Used for performing BCD arithmetic.

7. Timing and Control Unit:

This unit synchronizes all the microprocessor operations with the clock and generates the control signals necessary for communication between the microprocessor and peripherals. The control signals are similar to the sync pulse in an oscilloscope. The $\overline{RD}$ and $\overline{WR}$ signals are sync pulses indicating the availability of data on the data bus.

8. Interrupt controls:

The various interrupt controls signals (INTR, RST 5.5, RST 6.5, RST 7.5 and TRAP) are used to interrupt a microprocessor.

9. Serial I/O controls: Two serial I/O control signals (SID and SOD) are used to implement the serial data transmission.

Characteristics (features) of 8085A microprocessor and its signals

The 8085A (commonly known as 8085) is a 8-bit general purpose microprocessor capable of addressing 64K of memory. The device has 40 pins, require a +5V single power supply and can operate with a 3-MHZ, single phase clock.

The all the signals associated with 8085 can be classified into 6 groups:

1: Address bus: The 8085 has 16 signal lines that are used as the address bus; however, these lines are split into two segments A_{15}-A_8 and AD_7- AD_0. The eight signals A_{15}-A_8 are unidirectional and used as high order bus.

2. Data bus: The signal lines AD_7- AD_0 are bidirectional, they serve a dual purpose. They are used the low order address bus as well as data bus.

3. Control and status signals: This group of signals includes two control signals ($\overline{RD}$ and $\overline{WR}$), three status signals (IO/$\overline{M}$, S_1 and S_0) to identify the nature of the operation, and one special signals (ALE) to indicate the beginning of the operation.

- ALE- Address Latch Enable: This is a positive going pulse generated every time the 8085 begins an operation (machine cycle): it indicates that the bits AD_7-AD_0 are address bits. This signal is used primarily to latch the low-order address from the multiplexed bus and generate a separate set of eight address lines A_7 –A_0 .
- $\overline{RD}$- Read: this is a read control signal(active low). This signal indicates that the selected I/O or memory device is to be read and data are available on the data bus.
- $\overline{WR}$- Write: This is a write control signal (active low) . This signal indicates that the data on the data bus are to be written into a selected memory or I/O location.
- $IO/\overline{M}$: This is a status signal used to differentiate between I/O and memory operations. When it is high , it indicates an I/O operation; When it is low indicates a memory operation. This signal is combined with $\overline{RD}$ (Read) and $\overline{WR}$ (Write) to generate I/O and memory signals.
- S_1 and S_0 : These status signals, similar to $IO/\overline{M}$, can identify various operations, but they are rarely used in small systems.

4. Power Supply and Clock frequency:

- VCC: +5V power supply
- VSS: Ground reference
- X1 and X2: A crystal (RC or LC network) is connected at these two pins for frequency.

- CLK OUT: It can be used as the system clock for other devices.

5. Externally Initiated signals:

-**INTR (**input**):** interrupt request, used as a general purpose interrupt.

- $\overline{INTA}$ (Output): This is used to acknowledge an Interrupt.

-RST 7.5, 6.5, 5.5 (inputs): These are vectored interrupts that transfer the program control to specific memory locations. They have higher priorities than INTR interrupt. Among these three, the priority order is 7.5, 6.5, and 5.5.

- TRAP (input): This is a non-maskable interrupt with highest priority.

-HOLD (input): This signal indicates that a peripheral such as a DMA(Direct Memory Access) controller is requesting use of Address and data bus.

-HLDA (output): Hold Acknowledge: This signal acknowledges the HOLD request

- READY (Input) : This signal is used to delay the microprocessor Read or Write cycles until a slow- responding peripheral is ready to send or accept data. When this signal goes low, the microprocessor waits for an integral number of clock cycles until it goes high.

-$\overline{\text{RESET IN}}$: When the signal on this pin goes low, the program counter is set to zero, the buses are tri-stated, and MPU is reset.

-RESET OUT: This signal indicates that the MPU is being reset. The signal can be used to reset other devices.

Serial I/O ports: The 8085 has two signals to implement the serial transmission: SID (Serial Input Data) and SOD (Serial Output Data). In serial transmission, data bits are sent over a single line, one bit at a time, such as the transmission over telephone lines.

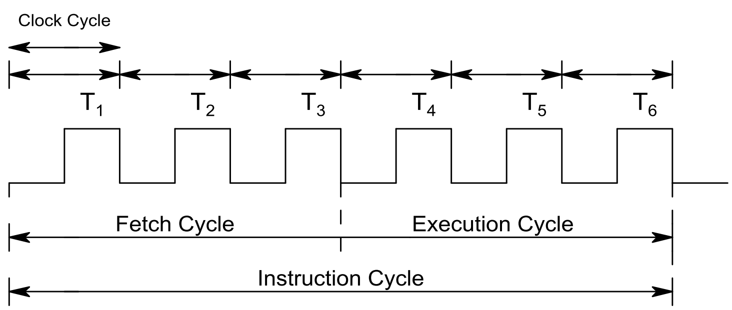
<u>Instruction description and Format:</u>

The computer can be used to perform a specific task, only by specifying the necessary steps to complete the task. The collection of such ordered steps forms a ‘program’ of a computer. These ordered steps are the instructions. Computer instructions are stored in central memory locations and are executed sequentially one at a time. The control reads an instruction from a specific address in memory and executes it. It then continues by reading the next instruction in sequence and executes it until the completion of the program.

Instruction cycle:

Instruction contains in the program and is pointed by the program counter. It is first moved to the instruction register and is decoded in binary form and stored as an instruction in the memory. The computer takes a certain period to complete this task i.e., instruction fetching, decoding and executing on the basis of clock speed. Such a time period is called 'Instruction cycle' and consists two cycles namely fetch and decode and Execute cycle.

In the fetch cycle the central processing unit obtains the instruction code the memory for its execution. Once the instruction code is fetched from memory, it is then executed. The execution cycle consists the calculating the address of the operands, fetching them, performing operations on them and finally outputting the result to a specified location.



Instruction description and format:

An instruction manipulates the data and a sequence of instructions constitutes a program. Generally each instruction has two parts: one is the task to be performed, called the **operation code** (Op-Code) field, and the second is the data to be operated on, called the **operand** or address field**.** The operand (or data) can be specified in various ways. It may include 8-bit (or 16-bit) data, an internal register, a memory location, or an 8-bit (or 16-bit) address. The Op-Code field specifies how data is to be manipulated and address field indicates the address of a data item. For example:

ADD R_1, R_0

Op-code address

Here R_0 is the source register and R_1 is the destination register. The instruction adds the contents of R_0 with the content of R_1 and stores result in R_1.

8085 A can handle at the maximum of 256 instructions (2^8)(246 instructions are used) . The sheet which contains all these instructions with their hex code, mnemonics, descriptions and function is called an instruction sheet. Depending on the number of address specified in instruction sheet, the instruction format can be classified into the categories.

- **One address format** (1 byte instruction)**:** Here 1 byte will be Op-code and operand will be default. E.g. ADD B, MOV A,B
- **Two address format** (2 byte instruction) :Here first byte will be Op-code and second byte will be the operand/data.
  E.g. IN 40H, MVI A, 8-bit Data

- **Three address format** (3 byte instruction): Here first byte will be Op-code, second and third byte will be operands/data. That is

  2nd byte- lower order data.

  3rd byte – higher order data

  E.g. LXI B, 4050 H

  Micro operation specifies the transfer of data into or out of a register.

Classification of an instruction

An instruction is a binary pattern designed inside a microprocessor to perform a specific function (task). The entire group of instructions called the instruction set. The 8085 instruction set can be classified in to 5- different groups.

- Data transfer group: The instructions which are used to transfer data from one register to another register or register to memory.
- Arithmetic group: The instructions which perform arithmetic operations such as addition, subtraction, increment, decrement etc.
- Logical group: The instructions which perform logical operations such as AND, OR, XOR, COMPARE etc.
- Branching group: The instructions which are used for looping and branching are called branching instructions like jump, call etc.
- Miscellaneous group: The instructions relating to stack operation, controlling purposes such as interrupt operations are fall under miscellaneous group including machine control like HLT, NOP.

Data transfer group instructions:

It is the longest group of instructions in 8085. This group of instruction copy data from a source location to destination location without modifying the contents of the source. The transfer of data may be between the registers or between register and memory or between an I/O device and accumulator. None of these instructions changes the flag. The instructions of this group are:

1) **MOV R_d, R_s** $R_d \leftarrow R_s$ (move register instruction)
   - 1 byte instruction
   - Copies data from source register to destination register.
   - Rd & Rs may be A, B, C, D, E, H &L
   - E.g. MOV A, B
2) **MVI R, 8 bit data** (move immediate instruction)
   - 2 byte instruction
   - Loads the second byte (8 bit immediate data) into the register specified.
   - R may be A, B, C, D, E, H & L
   - E.g. MVI C, 53H $C \leftarrow$53H
3) **MOV M, R** (Move to memory from register)
   - Copy the contents of the specified register to memory. Here memory is the location specified by contents of the HL register pair.
   - E.g. MOV M, B
4) **MOV R, M** (move to register from memory)
   - Copy the contents of memory location specified by HL pair to specified register.
   - E. g. MOV B, M

 # Write a program to load memory locations 7090 H and 7080 H with data 40H and 50H and then swap these data.

Sol^n :

```
MVI H, 70H
MVI L, 90H
MVI A, 40H
MOV M, A
MOV C, M
MVI L, 80H
MVI B, 50H
MOV M, B
MOV D, M
```

```
MOV M, C
MVI L, 90H
MOV M, D
HLT
```

5) LXI R_P, 2 bytes data (load register pair)
- 3-byte instruction
- Load immediate data to register pair
- Register pair may be BC, DE, HL & SP(Stack pointer)
- 1[st] byte- Op-code
- 2[nd] byte – lower order data
- 3[rd] byte- higher order data
- E.g. L X I B, 4532H; B ← 45, C ← 32H

6) MVI M, data (load memory immediate)
- 2 byte instruction.
- Loads the 8-bit data to the memory location whose address is specified by the contents of HL pair. E.g. MVI M , 35H; [HL] ←35H

7) LDA 4035H (Load accumulator direct)
- 3-byte instruction
- Loads the accumulator with the contents of memory location whose address is specified by 16 bit address.
- A ← [4035H]

8) LDAX R_P (Load accumulator indirect)
- 1 byte instruction.
- Loads the contents of memory location pointed by the contents of register pair to accumulator.
- E. g. LDAX B [A] ← [[BC]]

  LXI B, 9000H → B= 90, C= 00

  LDAX B → A= [9000]

9) STA 16-bit address (store accumulator contents direct)
- 3-byte instruction.
- Stores the contents of accumulator to specified address
- E.g. STA FA00H [FA00] ← [A]

10) STAX RP [RP] ← A
- Store s the contents of accumulator to memory location specified by the contents of register pair. 1 byte instruction

- E. g. STAX B

```
LXI B, 9500H          output
LXI D, 9501H          [9500] = 32
MVI   A, 32H          [9501] = 7A
STAX B
MVI A, 7AH
STAX D       [DE] ←A
```

11) IN 8-bit address

- 2-byte instruction
- Read data from the input port address specified in the second byte and loads data into the accumulator i. e. input port content to accumulator:
- E. g. IN 40H A ←[40H]

12) OUT 8-bit address

- 2-byte instruction
- Copies the contents of the accumulator to the output port address specified in the 2nd byte. That means accumulator to output port: P ← A
- E. g. OUT 40H [40] ← A

13) LHLD 16-bit address (Load HL directly)

- 3-byte instruction.
- Loads the contents of specified memory location to L –register and contents of next higher location to H-register.

E.g. LXI H, 9500H

MVI M, 32H

MVI L, 01H

MVI m, 7AH

LHLD 9500H H=7A, L=32

32	9500
7A	9501

14) SHLD 16-bit address (store HL directly)

- Opposite to LHLD.
- Stores the contents of L-register to specified memory location and contents of H-register to next higher memory location.
- E .g. LXI H, 9500H [8500]=00
  SHLD 8500H [8501]=95

15) XCHG (Exchange)

- Exchanges DE pair with HL pair.

- E. g. LXI H, 7500H H= 75, L=00
  LXI D, 9532H D=95. E=32
  XCHG H=95, L=32
  D=75 E=00

Addressing modes:

Instructions are command to perform a certain task in microprocessor. The instruction consists of op-code and data called operand. The operand may be the source only, destination only or both of them. In these instructions, the source can be a register, a memory or an input port. Similarly, destination can be a register, a memory location, or an output port. The various format (way) of specifying the operands are called addressing mode. So addressing mode specifies where the operands are located rather than their nature. The 8085 has 5 addressing mode:

1) **Direct addressing mode:**
   The instruction using this mode specifies the effective address as part of instruction. The instruction size either 2-bytes or 3-bytes with first byte op-code followed by 1 or 2 bytes of address of data.

   E. g. LDA 9500H A ←[9500]
   IN 80H A ←[80]

   This type of addressing is called absolute addressing.
2) **Register Direct addressing mode:**
   This mode specifies the register or register pair that contains the data.
   E g. MOV A, B
   Here register B contains data rather than address of the data. Other examples are: ADD, XCHG etc.
3) **Register Indirect addressing mode:**
   In this mode the address part of the instruction specifies the memory whose contents are the address of the operand. So in this type of addressing mode, it is the address of the address rather than address itself. (One operand is register)
   e. g. MOV R, M MOV M, R STAX, LDAX etc.
   STAX B B= 95 C =00
   [9500] ←A
4) **Immediate addressing mode:**
   In this mode, the operand position is the immediate data. For 8-bit data, instruction size is 2 bytes and for 16 bit data, instruction size is 3 bytes.

   E.g. MVI A, 32H

LXI B, 4567H

5) **Implied or Inherent addressing mode:**

The instructions of this mode donot have operands. E.g.
NOP: No operation
HLT: Halt
EI: Enable interrupt
DI: Disable interrupt

Q) What do you understand by addressing modes in microprocessor? Explain all the addressing modes of 8085 up with suitable example for each.

Arithmetic group Instructions:

The 8085 microprocessor performs various arithmetic operations such as addition, subtraction, increment and decrement. These arithmetic operations have the following mnemonics.

1) **ADD R/M**
   - 1 byte add instruction.
   - Adds the contents of register/memory to the contents of the accumulator and stores the result in accumulator.
   - E. g. Add B; A ←[A] + [B]
2) **ADI 8 bit data**
   - 2 byte add immediate instruction.
   - Adds the 8 bit data with the contents of accumulator and stores result in accumulator.
   - E g. ADI 9BH ; A←A+9BH
3) **SUB R/M**
   - 1 byte subtract instruction.
   - Subtracts the contents of specified register / m with the contents of accumulator and stores the result in accumulator.
   - E. g. SUB D ; A ←A-D
4) **SUI 8 bit data**
   - 2 byte subtract immediate instruction.
   - Subtracts the 8 bit data from the contents of accumulator stores result in accumulator.
   - E. g. SUI D3H; A ←A-D3H
5) **INR R/M, DCR R/M**
   - 1 byte increment and decrement instructions.

- Increase and decrease the contents of R(register) or M(memory) by 1 respectively.
- E. g. DCR B ; B=B-1
  DCR M ; [HL] = [HL]-1
  INR A ; A=A+1
  INR M ; [HL] +1
  For these, all flags are affected except carry.

6) INX Rp, DCX R_P

- Increase and decrease the register pair by 1.
- Acts as 16 bit counter made from the contents of 2 registers (1 byte instruction)
- E.g. INX B ;BC=BC+1
  DCX D ;DE=DE+1
- No flags affected

7) ADC R/M and ACI 8-bit data (addition with carry (1 byte))

- ACI 8-bit data= immediate (2 byte).
- Adds the contents of register or 8 bit data whatever used suitably with the Previous carry.
- E.g. ADC B ; A ←A+B+**CY**
  ACI 70H ; A ←A + 70+**CY**

8) SBB B/M

- 1 byte instruction.
- Subtracts the contents of register or memory from the contents of accumulator and stores the result in accumulator.
- e. g. SBB D ; A ←A-D-**Borrow**

SBI 8 bit data

- 2 byte instruction.
- Subtracts the 8-bit immediate data from the content of the accumulator and stores the result in accumulator.
- E.g. SBI 70H ; A ←A-70-**Borrow**

9) DAD Rp(double addition)

- 1 byte instruction.
- Adds register pair with HL pair and store the 16 bit result in HL pair.
- E. g. LXI H, 7320H
  LXI B, 4220H

  DAD B; HL=HL+BC

7320+4220=B540

10) DAA (Decimal adjustment accumulator)

- Used only after addition.
- 1 byte instruction.
- The content of accumulator is changed from binary to two 4-bit BCD digits.
- E. g

```
MVI A, 78H    ; A=78
MVI B, 42H    ; B=42
ADD B         ; A=A+B = BA
DAA           ; A=20, CY=1
```

The arithmetic operation add and subtract are performed in relation to the contents of accumulator. The features of these instructions are

1) They assume implicitly that the accumulator is one of the operands.
2) They modify all the flags according to the data conditions of the result.
3) They place the result in the accumulator.
4) They do not affect the contents of operand register or memory.

But the INR and DCR operations can be performed in any register or memory. These instructions

1) Affect the contents of specified register or memory.
2) Affect the flag except carry flag.

<u>Addition operation in 8085</u>:

8085 performs addition with 8-bit binary numbers and stores the result in accumulator. If the sum is greater than 8-bits (FFH), it sets the carry flag.

E.g.	MVI A, 93H	1 0 1 1 0 1 1 1	B7
	MVI C, B7H	+1 0 0 1 0 0 1 1	+ 93
	ADD C	**1 0 1 0 0 1 0 1 0**	1 4A
		CY	**CY**

<u>Subtraction operation in 8085:</u>

8085 performs subtraction operation by using 2's complement and the steps used are:

1) Converts the subtrahend (the number to be subtracted) into its 1's complement.
2) Adds 1 to 1's complement to obtain 2's complement of the subtrahend.
3) Adds 2's complement to the minuend (the contents of the accumulator).
4) Complements the carry flag.

```
E.g.    MVI A, 97H                              65H: 0  1  1  0  0  1  0  1
        MVI B, 65H    1's complement of 65H    : 1  0  0  1  1  0  1  0
        SUB B                                                          +1
                      ------------------------------------------------------
                      2's Complement of 65H:     1  0  0  1  1  0  1  1
                                          97H:  +1  0  0  1  0  1  1  1
                                                1  0  0  1  1  0  0  1  0
                                                CY |________32_________|
```

Complement carry =0 ∴ **A=32 CY=0**

B=97H, A=65H

```
97H:    1  0  0  1  0  1  1  1
        MVI B, 65H    1's complement of 97H    : 0  1  1  0  1  0  0  0
        SUB B                                                          +1
                      ------------------------------------------------------
                      2's Complement of 97H:     0  1  1  0  1  0  0  1
                                          65H:  +0  1  1  0  0  1  0  1
                                                0  1  1  0  0  1  1  1  0
                                                CY |_(Result in 2's complement form)
```

```
CY=1,      A=   CE:     1  1  0  0  1  1  1  1  0
   1's complement:      0  0  1  1  0  0  0  0  1
   2's complement:      0  0  1  1  0  0  0  1  0
                        |__________32___________|
```

1. **The memory location 2050H holds the data byte F7H. Write instructions to transfer the data byte to accumulator using different op-codes: MOV, LDAX and LDA.**

LXI H, 2050H	LXI B, 2050H	LDA 2050H
MOV A, M	LDAX B	

2. **Register B contains 32H, Use MOV and STAX to copy the contents of register B in memory location 8000H.**

LXI H, 8000H	LXI D, 8000H
MOV M, B	MOV A, B

3. **The accumulator contains F2H, Copy A into memory 8000H. Also copy F2H directly into 8000H.**

```
STA 8000H          LXI H, 8000H
                   MVI M, F2H
```

4. **The data 20H and 30H are stored in 2050H and 2051H. WAP to transfer the data to 3000H and 3001H using LHLD and SHLD instructions.**

```
MVI A, 20H
STA 2050H          LHLD 2050H
MVI A, 30H         SHLD 3000H
STA 2051H          HLT
```

5. **Pair B contains 1122H and pair D contains 3344H. WAP to exchange the contents of B and D pair using XCHG instruction.**

```
LXI B, 1122H       B=11, C=22
LXI D, 3344H       D=33, E=44
MOV H, B
MOV L, C
XCHG               (Exchange DE pair with HL pair)
MOV B, H
MOV C, L
HLT
```

6. **WAP to add two 4 digit BCD numbers equals 7342 and 1989 and store result in BC register.**

```
LXI H, 7342H
LXI B, 1989H
MOV A, L
ADD C
DAA
MOV C, A
MOV A, H
ADC B
DAA
MOV B, A
```

7. **Register BC contain 2793H and register DE contain 3182H. Write instruction to add these two 16 bit numbers and place the sum in memory locations 2050H and 2051H. What is DAA instruction? Explain its purpose with an example**.(Back Paper 2062)

MOV A, C		93H:		1 0 0 1 0 0 1 1
ADD E		+82H:		1 0 0 0 0 0 1 0
MOV L, A	L=15	15H	1	0 0 1 0 1 0 1 15H
MOV A, B		27H:		0 0 1 0 0 1 1 1

```
ADC D                +31H:          0 0 1 1 0 0 0 1
MOV H, A    H=59H                   0 1 0 1 1 0 0 1    59H
SHLD 2050H            ; [2050]← 15H  ,  [2051]← 59H
```

Note: SHLD stores the contents of L in specified location and contents of H in next higher location.

8. **Register BC contains 8538H and register DE contain 62A5H. Write instructions to subtract the contents of DE from the contents of BC and Place the result in BC.**

```
MOV A, C                          38
SUB E                           - A5
MOV C, A         ; C=93         1 93
MOV A, B                          85
SBB D                             62
MOV B, A         ; B=22           -1
                                  22
```

BCD Addition:

In many applications data are presented in decimal number. In such applications, it may be convenient to perform arithmetic operations directly in BCD numbers.

The microprocessor cannot recognize BCD numbers; it adds any two numbers in binary. In BCD addition, any number larger than 9 (from A to F) is invalid and needs to be adjusted by adding 6 in binary.

```
E.g.        A: 0000   1010
             + 0000   0110
               0001   0000  →  10BCD
```

A special instruction called DAA performs the function of adjusting a BCD sum in 8085. It uses the AC flag to sense that the value of the least four bits is larger than 9 and adjusts the bits to BCD value. Similarly, it uses CY flag to adjust the most significant four bits.

E.g. Add BCD 77 and 48

```
 77=   0111   0111
+48=   0100   1000
125    1011   1111
             +0110
           1 1 0101
   +0110      ......
CY= 1 0010    0101→ 25BCD
```

Logical Group Instructions:

A microprocessor is basically a programmable logic chip. It can perform all the logic functions of the hardwired logic through its instruction set. The 8085 instruction set includes such logic functions as AND, OR, XOR and NOT (Complement):

The following features hold true for all logic instructions:

1) The instructions implicitly assume that the accumulator is one of the operands.
2) All instructions reset (clear) carry flag except for complement where flag remain unchanged.
3) They modify Z, P & S flags according to the data conditions of the result.
4) Place the result in the accumulator.
5) They do not affect the contents of the operand register.

The logical operations have the following instructions.

1) **ANA R/M** (the contents of register/memory)
   - Logically AND the contents of register/memory with the contents of accumulator.
   - 1 byte instruction.
   - CY flag is reset and AC is set.
2) **ANI 8 bit data**
   - Logically AND 8 bit immediate data with the contents of accumulator.
   - 2 byte instruction.
   - CY flag is reset and AC is set. Others as per result
3) **ORA R/M**
   - Logically OR the contents of register/memory with the contents of accumulator.
   - 1 byte instruction.
   - CY and AC is reset and other as per result.
4) **ORI 8 bit data**
   - Logically OR 8 bit immediate data with the contents of the accumulator.
   - 2 byte instruction.
   - CY and AC is reset and other as per result.
5) **XRA R/M**
   - Logically exclusive OR the contents of register memory with the contents of accumulator.
   - 1 byte instruction.
   - CY and AC is reset and other as per result.

6) XRI 8 bit data

- Logically Exclusive OR 8 bit data immediate with the content of accumulator.
- 2 byte instruction.
- CY and AC is reset and other as per result.

7) CMA (Complement accumulator)

- 1 byte instruction.
- Complements the contents of the accumulator.
- No flags are affected.

Instruction	**CY**	**AC**
ANA/ANI	0	1
ORA/ORI	0	0
XRA/XRI	0	0

Logically Compare instructions

CMP R/M (1 byte instruction)

CPI 8 bit data (2 byte instruction)

- Compare the contents of register/ memory and 8 bit data with the contents of accumulator.
- Status is shown by flags & all flags are modified.

Case	**CY**	**Z**	
[A]<[R/M] or 8 bit	1	0	A-R<0
[A]=[R/M] or 8 bit	0	1	A-R=0
[A]>[R/M] or 8 bit	0	0	A-R>0

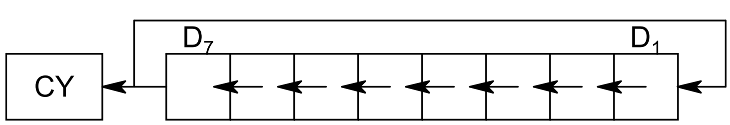
Used to indicate end of data.

Logical Rotate instructions

This group has four instructions, two are for rotating left and two are for rotating right. The instructions are:

1) RLC: Rotate accumulator left

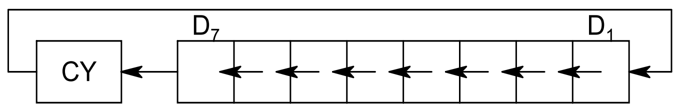
- Each bit is shifted to the adjacent left position. Bit D_7 becomes D_0.
- The carry flag is modified according to D_7.



$CY= D_7, D_7= D_6, D_6=D_5, \ldots\ldots, D_1=D_0, D_0=D_7$

2) RAL: Rotate accumulator left through carry

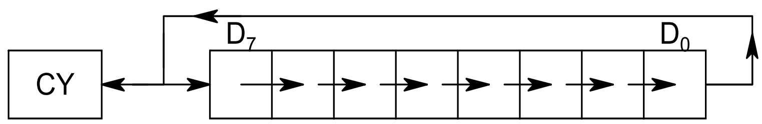
- Each bit is shifted to the adjacent left position. Bit D7 becomes the carry bit and the carry bit is shifted into D0.
- The carry flag is modified according to D7.



$CY= D_7, D_7= D_6, D_6=D_5,......,D_1=D_0, D_0=CY$

3) RRC: rotate accumulator right

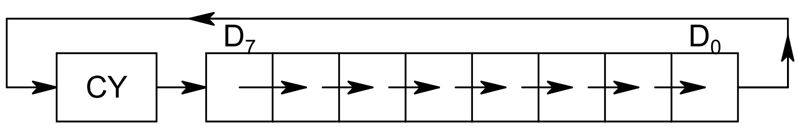
- Each bit is shifted right to the adjacent position. Bit D_0 becomes D_7.
- The carry flag is modified according to D_0.
- The carry flag is modified according to D_0.



$CY= D_0, D_7= D_0,......,D_0=D_1$

4) RAR: Rotate accumulator right through carry

- Each bit is shifted right to the adjacent position. Bit D_0 becomes the carry bit and the carry bit is shifted into D_7.



$CY= D_0$, $D_0= D_1$, ...D_7 = CY
Others:
-CMC – Complement carry
STC- set carry flag

9. The rotate instructions are primarily used in arithmetic multiply and divide operations and for serial data transfer.

   For e.g. If A=0000 1000=08H

   - By rotating 08H right: A=0000 0100=04.
   - This is equivalent to division by 2.
   - By rotating 08H left: A= 0001 0000 =10H
   - This is equivalent to multiplication by 2 (10H = 16_{10})

However these procedures are invalid when logic 1 is rotated from D_7 to D_0 or vice versa.

- E. g 80H rotates left = 01H
- 01 H rotate right = 80H

Q) Explain the instructions that fall in data transfer, arithmetic and logical groups with example: Show how the flags are affected by each instruction : - (10) [2061 Ashwin (2)]

Branching Group Instructions:

The microprocessor is a sequential machine; it executes machine codes from one memory location to the next. The branching instructions instruct the microprocessor to go to a different memory location and the microprocessor continues executing machine codes from that new location.

The branching instructions are the most powerful instructions because they allow the microprocessor to change the sequence of a program, either unconditionally or under certain test conditions. The branching instruction code categorized in following three groups:

- Jump instructions
- Call and return instruction
- Restart instruction

Jump Instructions:

The jump instructions specify the memory location explicitly. They are 3 byte instructions, one byte for the operation code followed by a 16 bit (2 byte) memory address. Jump instructions can be categorized into unconditional and conditional jump.

Unconditional Jump

8085 includes unconditional jump instruction to enable the programmer to set up continuous loops without depending only type of conditions. E.g. JMP 16 bit address: loads the program counter by 16 bit address and jumps to specified memory location.

E.g. JMP 4000H

Here, 40H is higher order address and 00H is lower order address. The lower order byte enters first and then higher order.

- The jump location can also be specified using a label or name.

E.g.

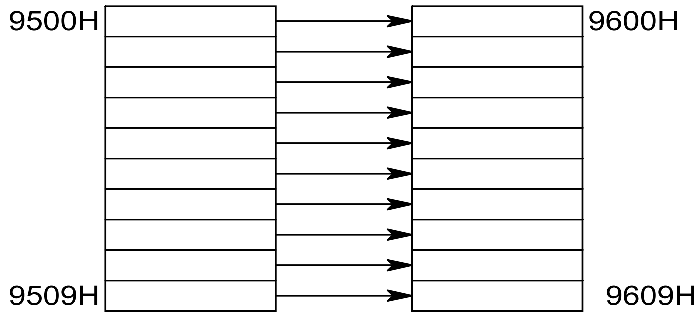
```
        MVI A, 80H          START: IN 00H
        OUT 43H                    OUT 01H
        MVI A, 00H                 JMP START
L1:     OUT 40H
        INR A
        JMP L1
        HLT
```

Conditional Jump

The conditional jump instructions allow the microprocessor to make decisions based on certain conditions indicated by the flags. After logic and arithmetic operations, flags are ser or reset to reflect the condition of data. These instructions check the flag conditions and make decisions to change or not to change the sequence of program. The four flags namely carry, zero, sign and parity used by the jump instruction.

Mnemonics	Description
JC 16 bit	Jump on carry (if CY=1)
JNC 16 bit	Jump on if no carry (if CY=0)
JZ 16bit	Jump on zero (if Z=1)
JNZ 16bit	jump on if no zero (if Z=0)
JP 16bit	jump on positive (if S=0)
JM 16bit	jump on negative (if S=1)
JPE 16bit	Jump on parity even (if P=1)
JPO 16bit	Jump on parity odd (if P=0)

E.g. WAP to move 10 bytes of data from starting address 9500 H to 9600H



2000 MVI B, 0AH
2002 LXI H, 9500H
2005 LXI D, 9600H

```
2008    MOV A, M
2009    STAX D          ; Store the contents of accumulator to register pair.
200A    INX H           ; Increment the register pair by 1.
200B    INX D
200C    DCR B
200D    JNZ 2008
2010    HLT
```

Q .Write to transfer 30 data starting from 8500 to 9500H if data is odd else store 00H.

```
        MVI B, 1EH
        LXI H, 8500H
        LXI D, 9500H
    L2: MOV A, M
        ANI 01H
        JNZ L1      ; If data is odd then go to L1.
        MVI A, 00H
        JMP L3
    L1: MOV A, M
    L3: STAX D
        INX D
        INX H
        DCR B
        JNZ L2
        HLT
```

<u>Call and return instructions: (Subroutine)</u>

Call and return instructions are associated with subroutine technique. A subroutine is a group of instructions that perform a subtask. A subroutine is written as a separate unit apart from the main program and the microprocessor transfers the program execution sequence from main program to subroutine whenever it is called to perform a task. After the completion of subroutine task microprocessor returns to main program. The subroutine technique eliminates the need to write a subtask repeatedly, thus it uses memory efficiently. Before implementing the subroutine, the stack must be defined; the stack is used to store the memory address of the instruction in the main program that follows the subroutines call.

To implement subroutine there are two instructions CALL and RET.

1. CALL 16 bit memory

- Call subroutine unconditionally.
- 3 byte instruction.
- Saves the contents of program counter on the stack pointer. Loads the PC by jump address (16 bit memory) and executes the subroutine.

2. RET

- Returns from the subroutine unconditionally.
- 1 byte instruction
- Inserts the contents of stack pointer to program counter.

3. CC, CNC, CZ, CNZ, CP, CM, CPE, CPO

- Call subroutine conditionally.
- Same as CALL except that it executes on the basis of flag conditions.

4. RC, RNC, RZ, RNZ, RP, RM, RPE, RPO

- Return subroutine conditionally.
- Same as RET except that if executes on the basis of flag conditions.

E.g. **Write an ALP to add two numbers using subroutines.**

2000	MVI B, 4AH	3000	MOV A, B
2002	MVI C, A0H	3001	ADD C
2004	CALL 3000H	3002	RET
2007	MOV B, A		
2008	HLT	PC = 2007 (i. e. sp)	

SP = 2007 (i.e. PC)

PC = 3000 (i.e. 16bit)

Q. What is the purpose of branching instruction? List out all the branching in 8085 and explain each with example. (2+8) (2046 shrawan)

<u>Restart Instruction:</u>

8085 instruction set includes 8 restart instructions (RST). These are 1 byte instructions and transfer the program execution to a specific location.

Restart instruction	Hex Code	Call location in hex
RST 0	C7	0000H
RST 1	CF	0008H
RST 2	D7	0010H
RST 3	DF	0018H
RST 4	E7	0020H
RST 5	EF	0028H
RST 6	F7	0030H
RST 7	FF	0038H

When RST instruction is executed, the 8085 stores the contents of PC on SP and transfers the program to the restart location. Actually these restart instructions are inserted through additional hardware. These instructions are part of interrupt process.

Miscellaneous Group Instructions:

STACK

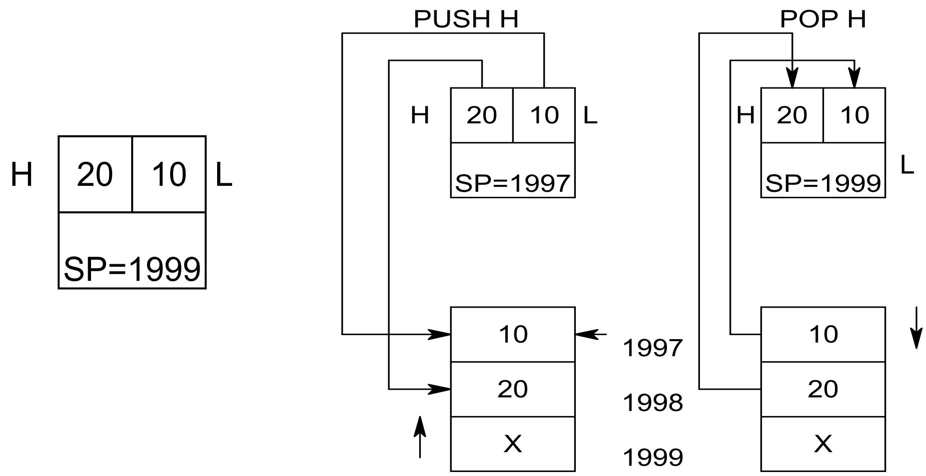
The stack is defined as a set of memory location in R/W memory, specified by a programmer in a main memory. These memory locations are used to store binary information temporarily during the execution of a program.

The beginning of the stack is defined in the program by using the instruction LXI SP, 16 bit address. Once the stack location is defined, it loads 16 bit address in the stack pointer register. Storing of data bytes for this operation takes place at the memory location that is one less than the address e.g. LXI SP, 2099H

Here the storing of data bytes begins at 2098H and continuous in reverse order i.e 2097H. Therefore, the stack is initialized at the highest available memory location to prevent the program from being destroyed by the stack information. The stack instructions are:

1. **PUSH Rp/PSW** (Store register pair on stack)
   - 1 byte instruction.
   - Copies the contents of specified register pair or program status word (accumulator and flag) on the stack.
   - Stack pointer is decremented and content of high order register is copied. Then it is again decremented and content of low order register is copied.
2. **POP Rp/PSW** (retrieve register pair from stack)
   - 1 byte instruction.
   - Copies the contents of the top two memory locations of the stack into specified register pair or program status word.

– A content of memory location indicated by SP is copied into low order register and SP is incremented by 1. Then the content of next memory location is copied into high order register and SP is incremented by 1.



3. XTHL – exchanges top of stack (TOS) with HL
4. SPHL – move HL to SP
5. PCHL – move HL to PC

Some instructions related to interrupt
DI – disable interrupt
EI – Enable interrupt
SIM – set interrupt mask
RIM – read interrupt mask

E.g.

```
LXI SP, 1FFFH
LXI H, 9320H
LXI B, 4732H
LXI D, ABCDH
MVI A, 34H
PUSH H
PUSH B
PUSH D
PUSH PSW
POP H
POP B
POP D
POP PSW
HLT
```

BEORE EXECUTION

H= 93	L= 20
B= 47	C=32
D= AB	E=CD
A= 34	F= 10

AFTER EXECUTION

H= 34	L=10
B=AB	C=CD
D= 47	E=32
A= 93	F=20

Note: **STACK Works in LIFO (Last In First Out) manner.**

Question: What do you mean by stack and subroutine? What is the purpose of stack in subroutines call? Explain the concept of subroutines call and usage along with the changes in program execution sequence with a suitable example for 8085 microprocessor. (3+2+5) [2063 kartik]

6. WAP to sort in ascending order for 10 bytes from 1120H.

```
START: LXI H, 1120H
       MVI D, 00H
       MVI C, 09H
L2:    MOV A, M
       INX H
       CMP M
       JC L1    ; if A<M
       MOV B, M
       MOV  M, A
       DCX H
       MOV M, B
       INX H
       MVI D, 01H
L1:    DCR C
       JNZ L2
       MOV    A, D
       RRC
       JC START
       HLT
```

Time Delay and Counter

Counter:

It is designed simply by loading an appropriate number into one of the registers and using the INR or DCR instructions. A loop is established to update a count, and each count is checked to determine whether it has reached the final number, if not the loop is repeated.

Time Delay

When we use loop by counter, the loop causes the delay. Depending upon the clock period of the system, the time delay occurred during looping. The instructions within the loop use their own T-states so they need certain time to execute resulting delay.

Suppose we have an 8085 micro processor with 2MHZ clock frequency. Let us use the instruction MVI which takes 7 T-states. Clock frequency of system (f) = 2 mhz

Clock period (T) = 1/f= ½ *10^{-6} = 0.5Microproceesor

Time to execute MVI = 7 T-states * 0.5

= 305Micro processor

```
Eg.MVI C, FFH         7
LOOP:  DCR C          4
       JNZ LOOP       10/7
```

Here register C is loaded with count FFH (255_{10}) by using MVI which takes 7 T-states. Nest 2 instructions DCR and JNZ form a loop with a total of 14 (4+10) T-states. The loop is repeated 255 times until C=0. The time delay in loop (TC) with 2 mhz frequency is

Tl= (T * loop T- states * count)

Where, Tl = time delay in loop

T= system clock period

Count = decimal value for counter

Tl= 0.5 * 10^{-6} * 14 * 255

= 1785 ms

But JNZ takes only 7 T-states when exited from loop i. e. last count = 0.50 adjusted loop delay

Tla= Tl-(3 T-states – clock period)

= 1785 ms – 1.5 ms = 1783.5 ms

Total delay loop of program

TD= Time to execute outside code + TLA inside Loop

= 7 * 0.5MS + 1783.5 ms

= 1787 ms ≈ 1.8 ms

To increase the time delay beyond 1-8 ms for 2MHZ microprocessor, we need to use counter for register pair or loop within a loop.

		T-States	Clocks
	MVI B, 40H; 64	7	7*1
L2:	MVI C, 80H; 128	7	7*64
L1:	DCR C	4	4*128*64
	JNZ L1	10/7	(10*127+7*1) *64
	DCR B	4	4*64
	JNZ L2	10/7	10*63+7*1
	RET	10	10*1
			115854

For 2 MHZ Microprocessor
Total time taken to execute above
Subroutine = 1158 * $0.5*10^{-6}$ S
= 57.927 ms

BCD to binary conversion:

In most microprocessor-based products. Data are entered and displayed in decimal numbers. For example, in an instrumentation laboratory, readings such as voltage and current are maintained in decimal numbers, and data are entered through decimal keyboard. The system monitor program of the instrument converts each key into an equivalent 4-bit binary number and stores two BCD numbers in an 8-bit register or a memory location. These numbers are called packed BCD.

Conversion of BCD number into binary number employs the principle of positional weighting in a given number.

E.g. $72_{10} = 7 \times 10 + 2$

converting an 8-bit BCD number into its binary equivalent requires.

- Separate an 8- bit packed BCD number into two 4 bit unpacked BCD digits i.e. BCD_1 and BCD_2.
- Convert each digit into its binary value according to its position.
- Add both binary numbers to obtain the binary equivalent of the BCD number.

E.g.

72_{10} = 0111 0010BCD

Step 1: 0111 0010 → 0000 0010 Unpacked BCD_1

→ 0000 0111 Unpacked BCD_2

Step 2: Multiply BCD_2 by 10 (7×10)

Step 3: Add BCD_1 to answer of step 2

Program: 2-digit BCD- to- Binary Conversion

A BCD number between 0 and 99 is stored in an R/W memory location called the **Input Buffer** (INBUF). Write a main program and a conversion subroutine (BCDBIN) to convert the BCD number into its equivalent binary number. Store the result in a memory location defined as the **Output Buffer** (OUTBUF).

```
    LXI H, 2020H
    MVI E, 0A H
    MOV A, M      ; 0111 0010
    ANI F0H       ; 0111 0000
    RRC
    RRC
    RRC
    RRC
    MOV B,A
    XRA A
L1: ADD B; 7× 10+2
    DCR E
    JNZ L1
    MOV C, A
    MOV A, M
    ANI 0FH
    ADD C
    STA 2030H
    HLT
```

Binary to BCD conversion

$1111\ 1111_2$ (FFH) = 255_{10} = $0010\ 0101\ 0101_{BCD}$

Step 1: If < 100 goto step 2

Else subtract 100 repeatatively.

Quotient is BCD_1 (Divide by 100)

Step 2: If < 10 goto step 3

Else subtract 10 repeatatively.

Quotient is BCD_2 (Divide by 10)

Step 3: Remainder is BCD_3

Program to convert 8-bit binary to BCD

```
                LXI SP, 1999H
                LXI H, 2020H;   Source
                MOV A, M
                CALL PWRTEN
                HLT
PWRTEN:         LXI H, 2030H;   Destination
                MVI B, 64H
                CALL BINBCD
                MVI B, 0AH
                CALL BINBCD
                MOV M, A
                RET
BINBCD:         MVI M, FFH
NEXT:           INR M
                SUB B
                JNC NEXT
                ADD B
                INX H
                RET
```

Binary to ASCII conversion

Alphanumeric codes:

A computer is a binary machine, to communicate with the computer in alphanumeric letters and decimal numbers, translation codes are necessary. The commonly used code is Known as ASCII (American standard codes for information interchange). It is a 7-bit code with 128 combinations and each combination from 01H to 7FH is organized to a letter, decimal number, symbol or machine command. For eg. 30H to 39H represents 0 to 9, 41H to 5AH represents A to Z, 21H to 2FH represents various symbols and 61H to 7AH represents a to z.

General Letters / Numbers	ASCII (Hex)	ASCII (Decimal)
0 – 9	30 – 39	48 – 57
A – Z	41 – 5A	65 – 90
a – z	61 – 7A	97 – 122

Program to Convert 8 – bit binary number to ASCII
Step 1: If number < 10, then add 30H
Else add 37H (30H + 07H)
For example: A = A + 30H + 07H = 41H

```
        LXI SP, 1999H
        LXI H, 1120H;  Source
        LXI D, 1160H;  Destination
        MOV A, M
        ANI F0H
        RRC
        RRC
        RRC
        RRC
        CALL ASCII
        STAX D
        INX H
        MOV A, M
        ANI 0FH
        CALL ASCII
        STAX D
        HLT
ASCII:  CPI 0AH
        JC BELOW
        ADI 07H
BELOW:  ADI 30H
        RET
```

ASCII to Binary Conversion
Step 1: Subtract 30H
Step 2: If < 0AH, then binary as it is
Else subtract 07H
For example: If ASCII is 41H, then 41H – 30H = 11H; 11H – 07H = 0AH

Program to convert ASCII number to Binary:

```
        LXI SP, 1999H
        LXI H, 1040H;  Source
        LXI D, 1050H;  Destination
        MOV A, M
        CALL ASCBIN
        STAX D
        HLT
ASCBIN: SUI 30H
```

```
            CPI 0AH
            RC
            SUI 07H
            RET
```

BCD to 7 – Segment LED code Conversion:

- Table lookup technique (TLT) is used.
- In TLT, the codes of digits to be displayed are stored sequentially in memory.

BCD Number	7-Segment Code
0	3FH
1	06H
2	5BH
3	4FH
4	66H
5	6DH
6	7DH
7	07H
8	7FH
9	6FH
Invalid	00H

Program to convert 3 packed BCD to 7-Segment LED code

```
            LXI SP, 1999H
            LXI H, 1150H
            MVI D, 03H
            LXI B, 1190H
NEXT:       MOV A, M
            ANI F0H
            RRC
            RRC
            RRC
            RRC
            CALL CODE
            INX B
            MOV A, M
            ANI 0FH
            CALL CODE
            INX B
            INX H
            DCR D
            JNZ NEXT
            HLT
```

```
CODE:       PUSH H
            LXI H, 1170H
            ADD L
            MOV L, A
            MOV A, M
            STAX B
            POP H
            RET
```

Program to multiply two 8 bit binary numbers.

```
            LXI SP, 1999H
            LHLD 2050H;   Two numbers
            XCHG;  Multiplier on D, Multiplicand on E
            CALL MULT
            SHLD 2090H;   Store the product
            HLT
MULT:       MOV A, D
            MVI D, 00H
            LXI H, 0000H
            MVI B, 08H;    Counter
NEXT:       RAR;    Check multiplier bit is 1 or 0
            JNC BELOW
            DAD D;Partial result in HL
BELOW:      XCHG
            DAD H;Shift left multiplicand
            XCHG;  Retrieve shifted multiplicand
            DCR B
            JNZ NEXT
            RET
```

Note: Try Division yourself.