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:

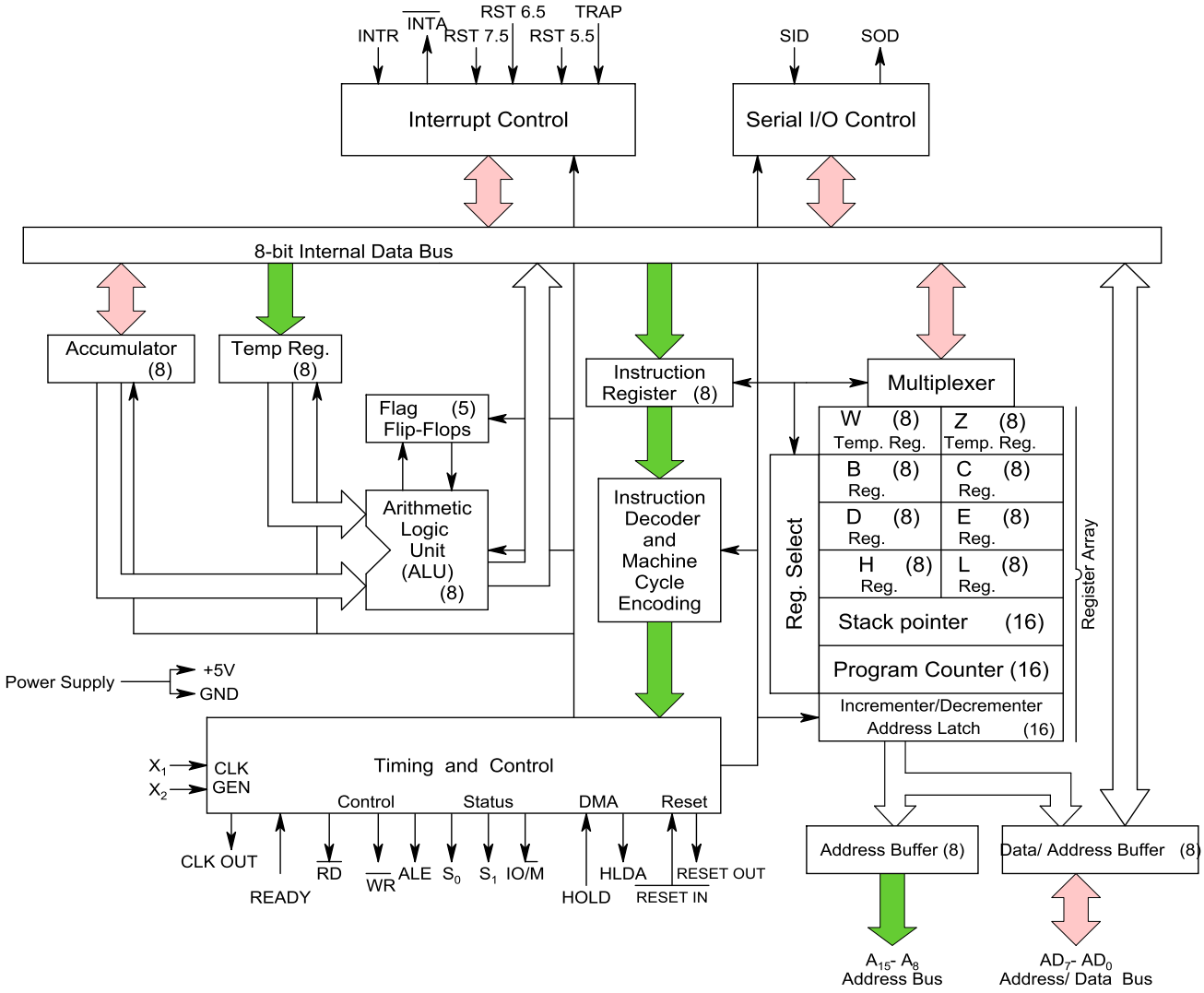


Fig. 2.1: The 8085A microprocessor Functional Block Diagram

The functioning of different components of 8085 microprocessor are listed below:

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 an 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. Instruction Decoder and Machine Cycle Encoding: The instruction decoder receives the bit pattern from instruction register, decodes the instruction and gives the decoded information to control logic. Machine cycle encoder establishes the sequences of events to follow and it gives the information of which machine cycle is currently being executed.

6. Register Array:

- **Scratch pad registers (B, C, D, & E):** It is an 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.
- **Memory Register (H & L)**: They are 8 bit registers that can be used in same manner as scratch pad registers. They are also used in pointing the memory location.
- **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.

7. 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

Fig. 2.2: 8085 Flag Register

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.

8. 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.

9. 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.

10. 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 an 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. It requires external hardware to separate data lines from address line, this achieved by the signal address latch enable (ALE). When ALE is set, it works as address lines; when ALE is reset, it works as data lines.

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 statuses 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.
- $\overline{\textbf{RESET IN}}$ **(input):** 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 (output):** This signal indicates that the MPU is being reset. The signal can be used to reset other devices.

6. **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.

Instruction

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. This time period is called 'Instruction cycle' and consists two cycles namely fetch and decode & 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.

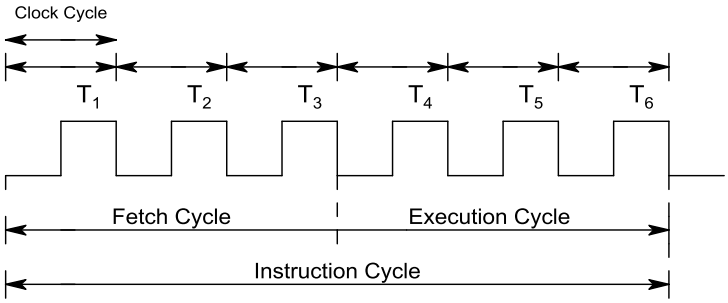


Fig. 2.3: Instruction cycle

Instruction format and data 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: if the instruction is ADD R_1, R_0; here ADD is Op-code and R_1, R_0 are addresses where data resides for processing.

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.

8085A can handle at the maximum of 256 instructions (2^8=256) among which 246 instructions are being 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 instruction format** (1 byte instruction): Here 1 byte will be Op-code and operand will be default. E.g. ADD B, MOV A, B etc.
- **Two address instruction 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, 4AH etc
- **Three address instruction format** (3 byte instruction): Here first byte will be Op-code, second and third byte will be operands/data. That is
  2^{nd} byte- lower order data
  3^{rd} byte – higher order data
  E.g. LXI B, 4050H; LDA 4050H etc

Data Format

The 8085 is an 8-bit microprocessor, and it processes only binary numbers. However, the real world operates in decimal numbers and languages of alphabets and characters. In 8-bit processor system, commonly used codes and data format are ASCII, BCD, signed integers and unsigned integers.

ASCII Code: This is a 7-bit alphanumeric code that represents decimal numbers, English alphabets, and nonprintable characters such as carriage return. Extended ASCII is an 8-bit code.

BCD Code: The term BCD stands for Binary Coded Decimal; it is used for decimal numbers. The decimal numbering system has ten digits 0 to 9. Therefore we need only four bits to represent ten digits from 0000 to 1001. An 8-bit register of 8085 can store two BCD numbers and known ad packed BCD number.

Signed Integer: A signed integer is either positive number or a negative number. In an 8-bit processor, the most significant digit, D_7, is used for the sign; 0 represents the positive sign and 1 represents the negative sign. The remaining 7 bits, D_6-D_0 represents the magnitude of an integer.

Unsigned Integer: An integer without a sign can be represented by all 8-bits. In an 8-bit processor, the range of numbers can be processed at one time is 00H to FFH.

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 five 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 etc.
- Branching group: The instructions which are used for looping and branching are called branching instructions like JMP, 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 largest 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** (move register instruction)
   - 1 byte instruction
   - Copies data from source register (R_s) to destination register (R_d).
   - Rd & Rs may be A, B, C, D, E, H & L
   - E.g. MOV A, B; A ← B

2) **MVI R, 8 bit data** (move immediate instruction)
   - 2 byte instruction
   - Loads the second byte (8 bit immediate data) into the specified register.
   - R may be A, B, C, D, E, H & L
   - E.g. MVI C, 53H; C ← 53H

3) **MOV M, R** (Move to memory from register)
   - Copy the content of the specified register to memory. Here memory is the location specified by contents of the HL register pair.
   - E.g. MOV M, B; [HL] ← 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; B ← [HL]

5) **LXI R_P, 16-bit data** (load register pair immediate)
   - 3-byte instruction
   - Load immediate 16-bit data to register pair
   - Register pair (R_p) may be BC, DE, HL & SP (Stack pointer)
     During loading: 1st byte- Op-code, 2nd byte – lower order data, 3rd byte- higher order data
   - E.g. LXI B, 4532H; C ← 32H, B ← 45H

6) **MVI M, 8-bit 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 16-bit address** (Load accumulator directly)
   - 3-byte instruction
   - Loads the accumulator with the contents of memory location whose address is specified by 16 bit address.
   - E.g. LDA 4035H; A ← [4035H]

8) **LDAX R_P** (Load accumulator indirectly)
   - 1 byte instruction.
   - Loads the contents of memory location pointed by the contents of register pair (R_p) to accumulator.
   - R_p may be B and D only
   - E. g. LXI B, 9000H; B ← 90H, C ← 00H
     LDAX B; A ← [9000H]

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

10) **STAX R_P** (store accumulator indirectly)
   - 1 byte instruction
   - Store s the contents of accumulator to memory location specified by the contents of register pair (R_p).
   - R_p may be B and D only
   - E.g. STAX B; A ← [BC]

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

11) **IN 8-bit address** (input from port address)
   - 2-byte instruction
   - Read data from the input port address specified in the second byte and loads data into the accumulator i.e. copy content of input port to accumulator:
   - E.g. IN 40H; A ← [40H]

12) OUT 8-bit address (output to port address)

- 2-byte instruction
- Copies the contents of the accumulator to the output port address specified in the 2nd byte. That means copy data from accumulator to output port.
- E.g. OUT 40H; [40H] ← 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 ← 7AH, L ← 32H

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

- 3-byte instruction
- Stores the contents of L-register to specified memory location and contents of H-register to next higher memory location.
- E .g. LXI H, 9450H ;
  SHLD 8500H; [8500H] ← 50H, [8501H] ← 94H

15) XCHG (Exchange)

- 1-byte instruction
- Exchanges DE pair with HL pair.
- E. g. LXI H, 7500H ; H ← 75H, L ← 00H
  LXI D, 9532H; D ← 95H, E ← 32H
  XCHG; H ← 95H, L ← 32H, D ← 75H, E ← 00H

Some Examples:

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            HLT
HLT               HLT
```

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
HLT               HLT
```

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

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

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

5. **Register B contains 5AH and register E contains 45H. Write instructions to swap the register contents B and E.**

```
MVI B, 5AH
MVI E, 45H
MOV A, B
MOV B, E
MOV E, A
HLT
```

6. **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 ← 11H, C ← 22H
LXI D, 3344H ;    D ← 33H, E ← 44H
MOV H, B
MOV L, C
XCHG ;            HL ←→ DE
MOV B, H
MOV C, L
HLT
```

Arithmetic group Instructions

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

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.

Instructions under Arithmetic Group:

1) ADD R/M (addition)

- 1 byte instruction
- Adds the contents of register/memory with the content of the accumulator and stores the result in accumulator.
- E.g. ADD B; A ← A + B
  ADD M; [HL] ← A + [HL]

2) ADI 8-bit data (addition immediate)

- 2 byte instruction
- Adds the 8-bit data with the content of accumulator and stores result in accumulator.
- E.g. ADI 9BH; A ← A + 9BH

3) SUB R/M (subtraction)

- 1 byte instruction
- Subtracts the contents of specified register/memory from the content of accumulator and stores the result in accumulator.
- E.g. SUB D; A ← A – D

4) SUI 8-bit data (subtraction immediate)

- 2 byte instruction.
- Subtracts the 8-bit data from the content of accumulator and stores result in accumulator.
- E.g. SUI D3H; A ← A - D3H

5) INR R/M (increment)
DCR R/M (decrement)

- 1 byte instructions
- Increments or decrements the contents of register (R) or memory (M) by 1 respectively.
- E.g. DCR B; B ← B - 1
  DCR M; [HL] ← [HL] - 1
  INR A; A ← A + 1
  INR M; [HL] ← [HL] +1

  For INR and DCR, all flags are affected except carry.

6) INX R_p (increment register pair)
DCX R_P (decrement register pair)

- 1 byte instructions
- Increments or decrements the content of register pair by 1.

- Acts as 16 bit counter made from the contents of 2 registers
- E.g. INX B; BC ← BC + 1
  DCX D; DE ← DE + 1
  For INX and DCX, no flags are affected

7) ADC R/M (addition with carry)
ACI 8-bit data (addition with carry immediate)
- Adds the content of register/memory or 8-bit data with the content of accumulator and stores result into accumulator with the Previous carry.
- E.g. ADC B; A ← A + B + CY
  ACI 70H; A ← A + 70H + CY

8) SBB R/M (subtraction with borrow)
SBI 8-bit data (subtraction with borrow immediate)
- Subtracts the content of register/memory or 8-bit data from the content of accumulator and stores the result in accumulator.
- E.g. SBB D; A ← A – D – Borrow (CY)
  SBI 70H; A ← A – 70H – Borrow (CY)

9) DAD Rp (double addition)
- 1 byte instruction
- Adds the content of specified register pair (Rp) with the content of HL pair and store the 16-bit result in HL pair.
- Rp may be B, D and H
- E.g. LXI H, 7320H
  LXI B, 4220H
  DAD B; HL ← HL + BC [7320H + 4220H = B540H]

10) DAA (Decimal adjustment accumulator)
- 1 byte instruction
- Used only after addition.
- The content of accumulator is changed from binary to two 4-bit BCD digits.
- E.g MVI A, 75H; A ← 78H
  MVI B, 42H; B ← 42H
  ADD B; A ← A + B = B7H
  DAA; A ← 17, CY=1

Addition operation in 8085
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;
MVI C, B7H;
ADD C;

```
   1 0 1 1 0 1 1 1        B7
+  1 0 0 1 0 0 1 1     + 93
---------------------  -----
  1 0 1 0 0 1 0 1 0      1 4A
  ↑                      ↑
  CY                     CY
```

Subtraction operation in 8085:

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

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 of subtrahend to the minuend (the contents of the accumulator).
4) If carry is 0, result is same from step 3 and positive; else the result is 2's complement of step 3 and negative.

E.g.
```
MVI A, 97H
MVI B, 65H
SUB B
```

```
                    65H:    0  1  1  0  0  1  0  1
1's complement of 65H:      1  0  0  1  1  0  1  0
                                                +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
                         ↑ └-------------------┘
                                  32H
```

Since, CY = 1, A = 32H [Result = 32H]

E.g.
```
MVI A, 65H
MVI B, 97H
SUB B
```

```
                    97H:    1  0  0  1  0  1  1  1
1's complement of 97H:      0  1  1  0  1  0  0  0
                                                +1
                           ------------------------
2's Complement of 97H:      0  1  1  0  1  0  0  1
                    65H: + 0  1  1  0  0  1  0  1
                           ------------------------
                         1  1  0  0  1  1  1  0
                         └-------------------┘
                                  CEH
```

Since, CY = 0, A = 2's Complement of CEH [Result = -0011 0010 = -32H]

Some examples:

7. **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
HLT
```

8. **Register BC contains 2793H and register DE contains 3182H. Write instruction to add these two 16 bit numbers and place the sum in memory locations 2050H and 2051H.**

```
MOV A, C
ADD E;          A ← 15H
MOV L, A
MOV A, B
ADC D;          A ← 59H
MOV H, A
SHLD 2050H;          [2050H] ← 15H, [2051H] ← 59H
HLT
```

9. **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                    85H     38H
SUB E ;         A ← 93H     62H    -A5H
MOV C, A                    -1     1 93H
MOV A, B                    22H
SBB D ;         A ← 22H
MOV B, A
HLT
```

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 06H. For example, if the number is A,

```
  0000   1010   (0AH)
+0000    0110   (06H)
 0001    0000 → 10BCD
```

A special instruction called DAA performs the function of adjusting a BCD sum in 8085. It uses the Auxiliary Carry flag (AC) to sense that the value of the least four bits is larger than 9 and adjusts the bits to BCD value. Similarly, it uses Carry flag (CY) 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   0101
      +0110   ……
    1 0010    0101→ 125BCD
```

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 (AND)
- 1 byte instruction
- Logically AND the content of register/memory with the content of accumulator and stores the result into accumulator.
- CY flag is reset and AC is set. Others as per the result.
  E.g. ANA D; A ← A && D

2) ANI 8-bit data (AND Immediate)
- 2 byte instruction
- Logically AND 8-bit immediate data with the content of accumulator and stores the result into accumulator.
- CY flag is reset and AC is set. Others as per the result.
  E.g. ANI 85H; A ← A + 85H

3) ORA R/M (OR)
- 1 byte instruction
- Logically OR the content of register/memory with the content of accumulator and stores the result into accumulator.
- CY and AC flag are reset and other as per the result.
  E.g. ORA M; A ← A || [HL]

4) ORI 8 bit data (OR Immediate)
- 2 byte instruction
- Logically OR 8-bit immediate data with the content of the accumulator and stores the result into accumulator.
- CY and AC are reset and other as per the result.
  E.g. ORI 7AH; A ← A || 7AH

5) XRA R/M (XOR)
- 1 byte instruction
- Logically exclusive OR the content of register/memory with the content of accumulator and stores the result into accumulator.
- CY and AC are reset and other as per the result.

E.g. XRA L; A ← A ⊕ L

6) **XRI 8 bit data** (XOR Immediate)
   - 2 byte instruction
   - Logically Exclusive OR 8 bit data immediate with the content of accumulator and stores the result into accumulator.
   - CY and AC are reset and other as per the result.

   E.g. XRI 87H; A ← A ⊕ 87H

Instructions	CY	AC
ANA/ANI	0	1
ORA/ORI	0	0
XRA/XRI	0	0

7) **CMA** (Complement accumulator)
   - 1 byte instruction
   - Complements the contents of the accumulator.
   - No flags are affected.

   E.g. MVI A, 85H; A ← 85H (1000 0101)
   CMA; A ← 7AH (0111 1010)

8) **CMC** (Complement Carry)
   - 1 byte instruction
   - Complements the carry flag.

   E.g. CMC

9) **STC** (Set Carry)
   - 1 byte instruction
   - Sets the carry flag.

   E.g. STC

Logical Compare instructions

10) **CMP R/M** (1 byte – compare instruction)
11) **CPI 8 bit data** (2 byte – compare immediate instruction)
   - Compares the content of register/memory/8-bit data with the content of accumulator.
   - Status is shown by flags & all flags are modified.
   - Implicitly uses the subtraction operation without modifying the content of accumulator.
   - Can be used to indicate the end of data by comparing the target stage.

   E.g. CMP D
   CPI 85H

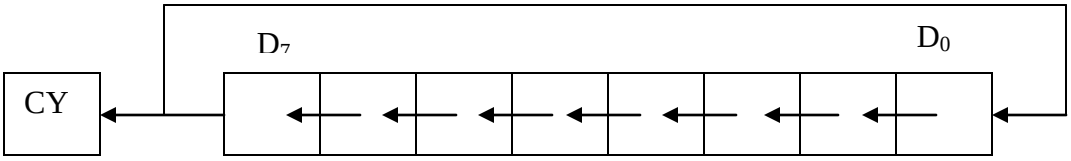
Case	CY	Z	Operation
[A] < [R/M] or 8-bit data	1	0	A – [R/M] or 8-bit data < 0
[A] = [R/M] or 8-bit data	0	1	A – [R/M] or 8-bit data = 0
[A] > [R/M] or 8-bit data	0	0	A – [R/M] or 8-bit data > 0

Logical Rotate instructions

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

12) RLC: Rotate accumulator left

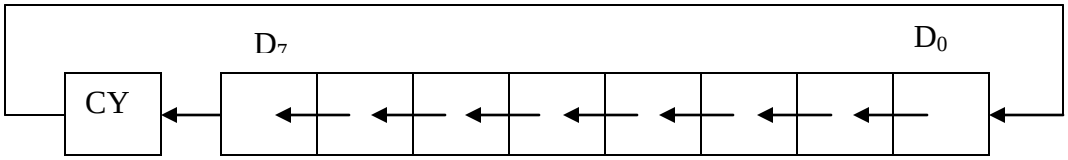
- Each bit is shifted to the adjacent left position. Bit D_7 becomes D_0.
- The carry flag is modified according to D_7.



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

13) RAL: Rotate accumulator left through carry

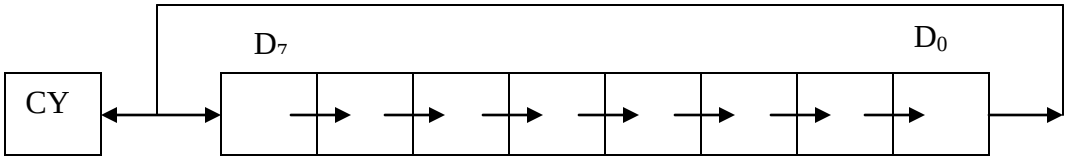
- Each bit is shifted to the adjacent left position. Bit D7 becomes the carry bit and the carry bit is shifted into D0.
- The carry flag is modified according to D7.



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

14) RRC: rotate accumulator right

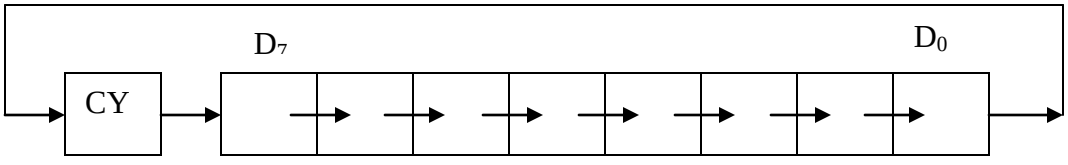
- Each bit is shifted right to the adjacent position. Bit D_0 becomes D_7.
- The carry flag is modified according to D_0.
- The carry flag is modified according to D_0.



$CY = D_0$, $D_7 = D_0$, $D_6 = D_7$, …, $D_1 = D_2$, $D_0 = D_1$

15) 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.



$CY = D_0$, $D_7 = CY$, $D_6 = D_7$, …,, $D_1 = D_2$, $D_0 = D_1$

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 = 04H. 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

Some Examples:

10. **Write instructions to AND the content of register B with the content of memory location C050H and store it into C051H.**

```
MOV A, B
LXI H, C050H
ANA M
STA C051H
HLT
```

11. **Write instructions to calculate the 2's complement of the content of register D.**

```
MOV A, D
CMA; or XRI FFH; 1's Complement
ADI A, 01H; 2's Complement
MOV D, A
HLT
```

12. **Write instructions to exchange the lower nibble and upper nibble of the data stored in memory location 2050H and store it into 2051H.**

```
LDA 2050H
RLC
RLC
RLC
RLC
STA 2051H
HLT
```

13. **Write instructions to set the bit D_4 and reset the bit D_6 of the data stored in memory location 2050H.**

```
LDA 2050H
ORI 10H; sets D4 (0001 0000)
ANI BFH; resets D6 (1011 1111)
STA 2050H
HLT
```

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 on the type of conditions. E.g. JMP 16-bit address: loads the program counter by 16 bit address and jumps to specified memory location.

Consider the instruction JMP 4000H;

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

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

```
C000  MVI A, 00H          MVI A, 00H
C002  OUT 40H       NEXT: OUT 40H
C004  INR A               INR A
C005  JMP C002H           JMP START
C008  HLT                 HLT
```

Conditional Jump

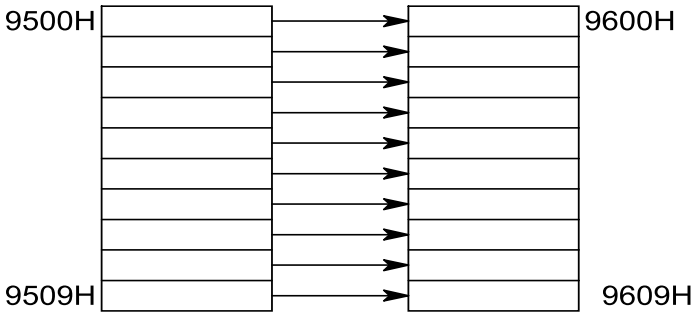
The conditional jump instructions allow the microprocessor to make decisions based on certain test conditions indicated by the flags. After logic and arithmetic operations, flags are ser or reset to reflect the condition of data. These instructions check the flag conditions and make decisions to change 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 Address/Label	Jump on carry (if CY=1)
JNC 16-bit Address/Label	Jump on if no carry (if CY=0)
JZ 16-bit Address/Label	Jump on zero (if Z=1)
JNZ 16-bit Address/Label	jump on if no zero (if Z=0)
JP 16-bit Address/Label	jump on positive (if S=0)

JM 16-bit Address/Label	jump on negative (if S=1)
JPE 16-bit Address/Label	Jump on parity even (if P=1)
JPO 16-bit Address/Label	Jump on parity odd (if P=0)

Some Examples:

14. WAP to transfer 10 bytes of data from starting address 9500 H to 9600H.




```
2000   MVI C, 0AH; Counter 10
2002   LXI H, 9500H; Initial Address of Source table
2005   LXI D, 9600H; Initial Address of Destination table
2008   MOV A, M
2009   STAX D        ; [DE] ← A
200A   INX H         ; HL ← HL + 1
200B   INX D
200C   DCR C
200D   JNZ 2008
2010   HLT
```

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

```
      MVI B, 1EH; 30D
      LXI H, 8500H
      LXI D, 9500H
L2:   MOV A, M
      RRC
      JC L1  ; If CY ← D0 = 1, i.e. 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
```

Call and return instructions: (Subroutine)

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 completing the subroutine task, microprocessor returns to main program.

Benefits of Subroutine:

- Reduces the amount of code because a common subroutine can be called from any number of places in the main program.
- The subroutine technique eliminates the need to write a subtask repeatedly, thus it uses memory efficiently.
- Encourage better program organization.
- Facilitates debugging of a program because defects can be more clearly isolated.
- Helps in the ongoing maintenance of programs because subroutines are readily identified for modification.

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 types of subroutines namely Unconditional and Conditional.

Unconditional Subroutine

CALL 16-bit memory address / label

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

  E.g. CALL 2050H
  CALL Label

RET

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

  E.g. RET

Conditional Subroutine

CC/ CNC/ CZ/ CNZ/ CP/ CM/ CPE/ CPO 16-bit address / label

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

RC/ RNC/ RZ/ RNZ/ RP/ RM/ RPE/ RPO

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

Some Examples:

16. **Write an ALP to add two numbers using subroutine.**

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

SP = 2007H (i.e. PC)
PC = 3000H (i.e. 16-bit jump address)

Restart Instruction

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
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 defined 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.

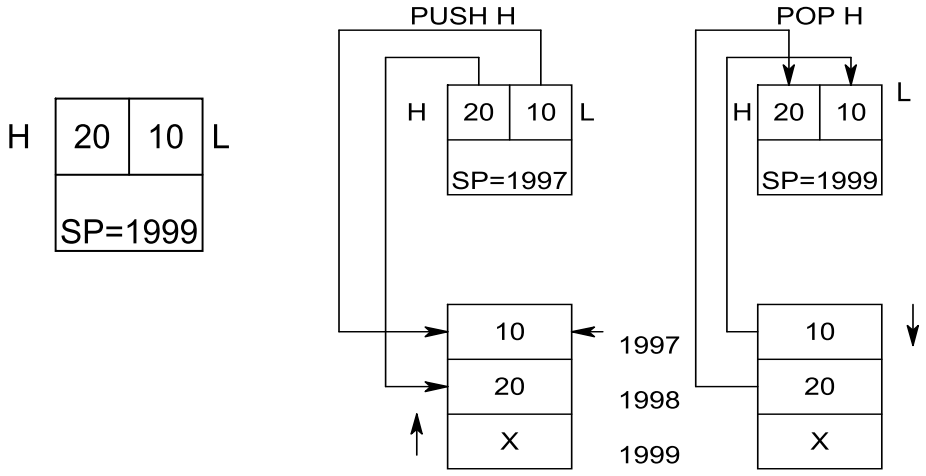
The stack instructions are:

PUSH R_P / PSW (Store register pair on stack)

- 1 byte instruction.
- Copies the contents of specified register pair or program status word - PSW (pair of 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.

POP R_P / 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 (PSW).
- 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.



Other Miscellaneous Instructions

XTHL – exchanges top of stack (TOS) with HL
SPHL – move HL to SP
PCHL – move HL to PC
DI – disable interrupt
EI – Enable interrupt
SIM – set interrupt mask
RIM – read interrupt mask
HLT
NOP

Some Examples

17. **Here are instructions to demonstrate the stack operation. Examine the content of related memory and registers during the stack operations. Consider the flag content is 10H.**
    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
```

During PUSH

H = 93H L = 20H
B = 47H C = 32H
D = ABH E = CDH
A = 34H F = 10H
SP = 1FF7H

Address	Content
1FFF	X
1FFE	93H
1FFD	20H
1FFC	47H
1FFB	32H
1FFA	ABH
1FF9	CDH
1FF8	34H
1FF7	10H

During POP

H = 34H L = 10H
B = ABH C = CDH
D = 47H E = 32H
A = 93H F = 20H
SP = 1FFFH

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

18. WAP to sort in ascending order for 10 bytes of data stored in a table starting 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
```

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 modes:

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 ← [9500H]
   IN 40H; A ← [40H]

   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
   ADD H
   XOR C 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.

   E.g. MOV A, M
   STAX B etc.

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
SBI 76H etc.

5) **Implied or Inherent addressing mode:**
The instructions of this mode do not have operands.
E.g. NOP: No operation
HLT: Halt
EI: Enable interrupt
DI: Disable interrupt

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.
Example:

Instructions	T-States
MVI C, FFH	7
LOOP: DCR C	4
JNZ LOOP	10/7

Suppose we have an 8085 micro processor with 2MHz clock frequency.
Clock frequency of system (f) = 2 MHz
Clock period (T) = 1/f= ½ *10^{-6} = 0.5 μs

Time delay outside the loop:
The instruction MVI takes 7 T-states.
Time to execute MVI (Outside the loop) – T_1 = 7 T-states * 0.5 μs
= 3.5 μs
Time delay inside the loop:
Here register C is loaded with count FFH (255_{10}). Next 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 (T_L) is
T_L= (Loop T-states * count * T)
Where, T_L = time delay in loop
T = system clock period
Count = decimal value of counter

T_L = 14 * 255 * 0.5 μs
= 1785 μs

But JNZ takes only 7 T-states when exited from loop i.e. when count = 0. So the adjusted loop delay (T_{LA})

$T_{LA} = T_L$ - (3 T-states * clock period)
= 1785 μs – 1.5 μs = 1783.5 μs

Total delay loop of program

T_D= Time to execute outside code (T_1) + Time to execute the loop (T_{LA})
= 3.5 μs + 1783.5 μs
= 1787 μs ≈ 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.

	Instructions	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
		Total Clocks:	**115854**

For 2 MHz Microprocessor, total time taken to execute above Subroutine
T_D = 115854 * 0.5 μs
= 57927 μs ≈ 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× 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. $72_{10} = 0111\ 0010_{BCD}$

Step 1: 0111 0010 → 0000 0111 Unpacked BCD_1
→ 0000 0010 Unpacked BCD_2

Step 2: Multiply BCD_1 by 10 (7×10)

Step 3: Add BCD_2 to answer of step 2

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).

19. WAP to read BCD number (Suppose 70_{10}: 0111 0000_{BCD}) stored at memory location 2020H and converts it into binary equivalent and finally stores that binary pattern into memory location 2030H.

```
    LXI H, 2020H ; Input Buffer Location
    MVI E, 0A H
    MOV A, M     ; A ← 0111 0010
    ANI F0H      ; A ← 0111 0000
    RRC
    RRC
    RRC
    RRC;  A ← 0000 0111
    MOV B, A
    XRA A; Clear Accumulator
L1: ADD B; 7 × 10
    DCR E
    JNZ L1
    MOV C, A; C ← 70D = 46H
    MOV A, M; A ← 0111 0010
    ANI 0FH; A ← 0000 0010
    ADD C; A ← 02H + 46H = 48H
    STA 2030H; Output Buffer Location
    HLT
```

Binary to BCD conversion

Suppose binary number is FFH → $1111\ 1111_2 = 255_{10} = 0010\ 0101\ 0101_{BCD}$

Step 1: If < 100 go to step 2
Else subtract 100 repetitively.
Quotient is BCD_1 (Divide by 100)

Step 2: If < 10 go to step 3
Else subtract 10 repetitively.
Quotient is BCD_2 (Divide by 10)

Step 3: Remainder is BCD_3

20. A binary number (Suppose FFH: 1111 1111_2) is stored in memory location 2020H. Convert the number into BCD and store each BCD as two unpacked BCD digits in memory location from 2030H.

```
        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

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, symbols or machine command. For e.g. 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

Method: If number < 10, then add 30H
Else add 37H (30H + 07H)
For example: A = A + 30H + 07H = 41H

21. An 8 bit binary number is stored in memory location 1120H. WAP to store the ASCII codes of the binary digits in location 1160H and 1161H.

```
        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 is as it is
Else subtract 07H

For example: If ASCII is 41H, then 41H – 30H = 11H; 11H – 07H = 0AH

22. **WAP to convert ASCII code stored at memory location 1040H to binary equivalent and store the result at location 1050H.**

```
                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

23. A set of three packed BCD numbers are stored in memory locations starting at 1150H. The seven segment codes of digits 0 to 9 for a common cathode LED are stored in memory locations starting at 1170H and the output buffer memory is reserved at 1190H. WAP to unpack the BCD number and select an appropriate seven segment code for each digit. The codes should be stored in output buffer memory.

```
        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
```

24. A multiplicand is stored in memory location 1150H and a multiplier is stored in location 1151H. WAP to multiply these numbers and store result from 1160H. Using shifting method.

```
        LXI SP, 1999H
        LHLD 1150H; Two numbers
        XCHG;       Multiplier on D, Multiplicand on E
        CALL MULT
        SHLD 1160H;         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
```

25. Write an assembly language program for dividing a 16-bit number by an 8-bit number. Suppose the dividend store in memory location 2050H and divisor at 2052H. Store the quotient and remainder into memory locations 2053H and 2054H respectively. Using shifting method.

```
        LHLD 2050H;     HL ← Dividend
        LDA 2052H;      A ← Divisor
        MOB B, A
        MVI C, 08H; Count
LOOP:   DAD H;      Shift Dividend by 1 bit to left
        MOV A, H;   A ← MSB of dividend
        SUB B;      Subtract Divisor
        JC NEXT;    is most significant part of dividend > divisor? if no, go to NEXT.
        MOV H, A;   H ← MSB of dividend
        INR L;      Increment Quotient by 1
NEXT:   DCR C;      Decrement count by 1
        JNZ LOOP
        SHLD 2053H; Store quotient and remainder
        HLT
```