3. Study of an 8-bit Microprocessor

3.1 Introduction

8-bit microprocessors use an 8-bit data bus and a 16-bit address bus, meaning their address space is limited to 64KB. The first widely adopted 8-bit microprocessor was Intel's 8080, which was used in many home computers in the late 1970s and early 1980s. The Zilog Z80 (compatible with the 8080) and the Motorola 6800 were also used in similar computers, including the MOSTEK 6502 inspired by the 6800 and Z80. When these early microprocessors reached the limit of their performance, manufacturers produced the new, more powerful generation of 8-bit processors, such as Motorola's 6809 and Intel's 8085.

3.2 8-bit Microprocessor

The table 3.5 summarizes the characteristics of 8-bit microprocessors from various manufacturers.

The internal architecture of these microprocessors is directly based on the Von Neumann structure.

3.3 Study of the 8085 Microprocessor

The 8085 microprocessor comes in a 40-pin Dual In-line Package (DIP) package and has the following characteristics:

- ✓ Requires only a +5V power supply;
- ✓ Can operate with a clock speed of $3 \rightarrow 6.144 MHz$ and performs at 1.5 MIPS;
- ✓ Capable of addressing 64 KB of memory;
- ✓ Requires 4 cycles per instruction;

Table 3.1: 8-bit Microprocessors and Their Manufacturers

Manufacturer Reference	**Intel 8008**	**Intel 8080**	**Intel 8085**	**Motorola 6800**	**Zilog Z80**	**Mostek 6502**	**Rockwell PPS8**	**National SC/MP**
Number of instructions	48	69	71	71	69	71	90	50
Memory space	16K	64K	64K	64K	64K	64K	32K	64K in 4K pages
General-pur -pose register	7	7	7	3	17	3	3	6
Number of transistors	3300	4000	6200	-	-	-	-	-
Clock in MHz	0.3	2-2.67	3.6 or 6	11.5 or 2	-	-	-	-
Year	1972	1974	1976	1974	1976	1975	-	1976

3.3.1 External Architecture of the 8085

Figure 3.1 shows the logic pinout of the 8085 microprocessor. All the signals are classified into six groups: (1) address bus, (2) data bus, (3) control and status signals, (4) power supply and frequency signals, (5) externally initiated signals, and (6) serial I/O ports.

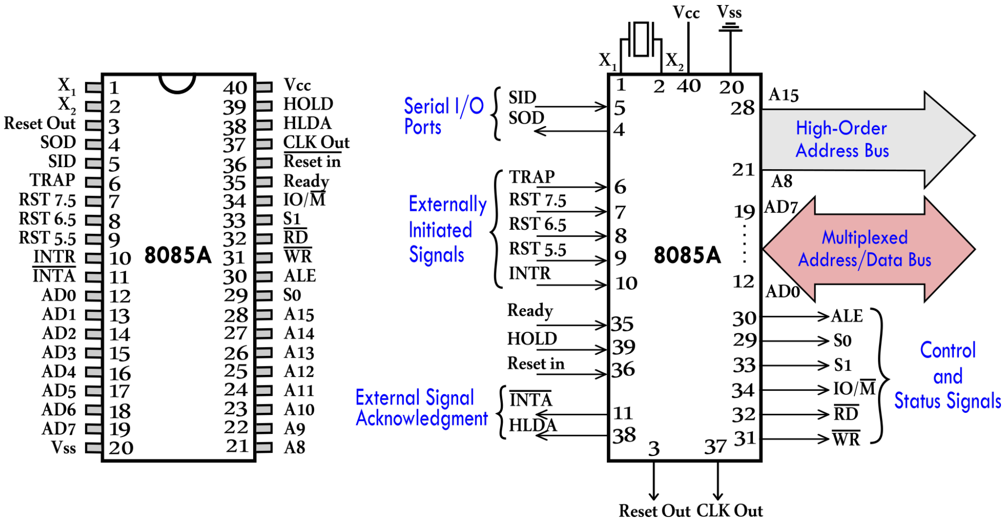


Figure 3.1: The Pinout and Signals of the 8085 Microprocessor

Les broches du 8085 peuvent être classées en cinq groupes:

a. Address Bus

The 8085 has 16 signal lines (pins) 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 signal lines, $A_{15} - A_8$, are unidirectional and used for the most significant bits, called the high-order address, of a 16-bit address. The signal lines $AD_7 - AD_0$ are used for a dual purpose, as explained in the next section.

b. Multiplexed Address/Data Bus

The signal lines $AD_7 - AD_0$ are bidirectional: they serve a dual purpose. They are used as the low-order address bus as well as the data bus. In executing an instruction, during the earlier part of the cycle, these lines are used as the low-order address bus. During the later part of the cycle, these lines are used as the data bus (This is also known as multiplexing the bus). However, the low-order address bus can be separated from these signals by using a latch as will be explained later.

c. Control and Status Signals

ALE-Address Latch Enable: This signal indicates that the bits on $AD_7 - AD_0$ are address bits and used to latch the low-order address $A_7 - A_0$.

$\overline{\textbf{RD}}$-Read: This is a Read control signal (active low). It indicates that the selected I/O or memory device is to be read and data are available on the data bus.

$\overline{\textbf{WR}}$-Write: This is a Write control signal (active low). It indicates that the data on the data bus are to be written into a selected memory or I/O location.

$\mathbf{IO/\overline{M} - Input - Output/Memory}$: This status signal used to indicate whether the 8085 is addressing memory ($IO/\overline{M} = 0$) or input/output ($IO/\overline{M} = 1$)."

S0 et **S1:** Status signals indicating the type of operation in progress on the bus. Rarely used in small systems.

d. Power Supply and Clock Frequency

Vcc: +5 V power supply.

Vss: 0 V power supply.

X1 et **X2:** A crystal (or RC, LC network) is connected at these two pins to generate a periodic square wave signal. To operate at 3 MHz, the crystal should have a frequency of 6 MHz.

CLK OUT-Clock Output: This signal can be used as the system clock for other devices.

e. Interrupt Signals

INTR (Interrupt Request): This is used as a general-purpose interrupt; it is sent by an interface indicating a request for interruption.

$\overline{\textbf{INTA}}$ (Interrupt Aknowlege): The 8085 responds to INTR by sending "0" on the $\overline{\text{INTA}}$ signal.

RST 7.5, **RST 6.5** and **RST 5.5** (Restart Interrupts): These are vectored interrupts that transfer the program control to specific memory locations.

TRAP: This is a non-maskable interrupt and has the highest priority.

f. Externally Initiated Signals

HOLD : This signal informs the 8085 that an external device, such as a DMA (Direct Memory Access) controller, is requesting the use of the address and data buses.

HLDA (Hold Acknowledge) : This signal acknowledges the HOLD signal.

READY: This signal is used to delay the microprocessor Read or Write cycles until a slow-responding peripheral is ready to send or accept data.

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

RESET OUT: This signal indicates that the CPU is being reset and will reset the devices connected to it.

g. Serial I/O Ports

SID (Serial Input Data) / **SOD** (Serial Output Data) : In serial transmission, data bits are sent over single line, one bit at a time.

3.3.2 Internal Architecture of the 8085

The internal structure of the 8085, as shown in Figure 3.2, essentially comprises of:

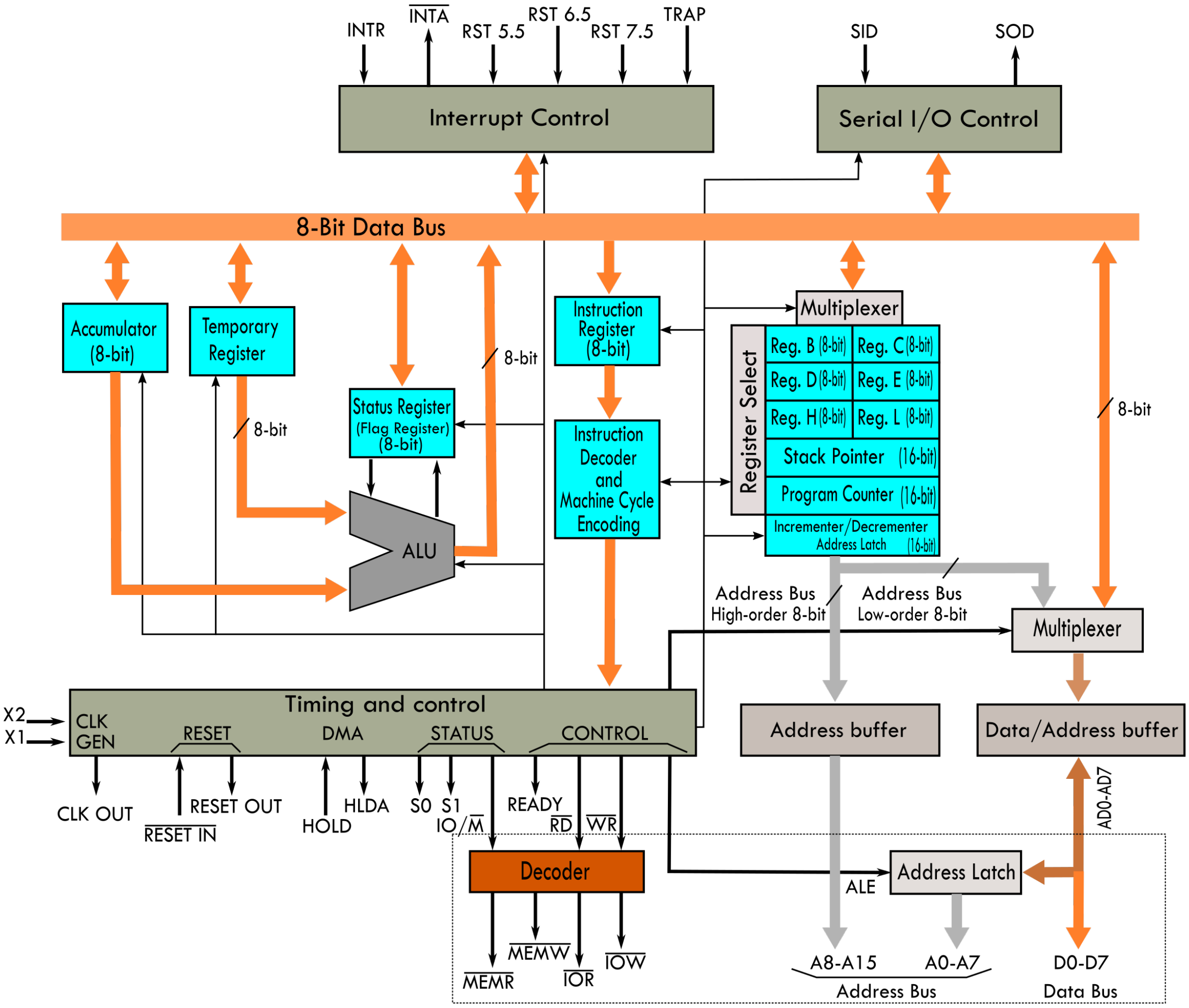


Figure 3.2: Functional Block Diagram of the Microprocessor 8085

1. Arithmetic and Logic Unit (ALU)

The Arithmetic and Logic Unit (ALU) performs various computing functions, including addition (+), subtraction (-), logical operations (AND, OR, NOT, XOR), increment/decrement (INR/DCR), comparison, left/right shifting, and more. It consists of the accumulator, the temporary register, the arithmetic and logic circuits, and five flags. The temporary register holds data during an arithmetic/logic operation, and the result is stored in the accumulator. The flags (flip-flops) are set or reset based on the outcome of the operation.

2. 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. It also controls fetching and decoding of instructions and generates appropriate control signals for their execution.

3. Instruction Register and Decoder

The instruction register and the decoder are part of the ALU. When an instruction is fetched from memory, it is loaded in the instruction register. The decoder decodes the instruction and establishes the sequence of events to follow. The instruction register is not programmable and cannot be accessed through any instruction.

4. Serial I/O Control Unit

Used to control serial data communication with external devices via SID/SOD lines.

5. Interrupt Control Unit

Used to handle various interrupts that occur via 5 interrupt pins: TRAP, RST7.5, RST6.5, RST5.5, and INTR.

6. Incrementer/Decrementer Address Latch

It increments/decrements the contents of the Program Counter and Stack Pointer when instructions related to them are executed.

7. Address and Data Buffers

They are used to temporarily hold Address/Data bits while they are placed on the Address/Data bus. They act as an interface between the internal and external buses.

8. Registers

The registers of the 8085 can be classified into three categories:

a. General-Purpose Registers

- **B, C, D, E, H,** and **L**: These are 8-bit registers used for general data storage and manipulation. They can be paired to form register pairs (16-bit), such as BC, DE, and HL, for certain instructions.

b. Special-Purpose Registers:

- **Program Counter (PC):** It is a 16-bit register that holds the address of the next instruction to be executed. It gets automatically incremented (+1) after fetching each instruction.

- **Stack Pointer (SP):** It is a 16-bit register which points to the ‘stack’. The stack is an area in the R/W memory where temporary data or return addresses (in cases of subroutine CALL) are stored. Stack is a auto-decrement facility provided in the system. The stack top is initialized by the SP by using the instruction LXI SP.
  Figure 3.3 illustrates the auto-increment and auto-decrement functionality for the PC and SP registers, respectively.

- **Instruction Register (IR):** It is a non-programmable register that temporarily holds the opcode of the instruction being fetched from memory. The opcode is a part of the instruction that specifies the operation to be performed by the microprocessor.

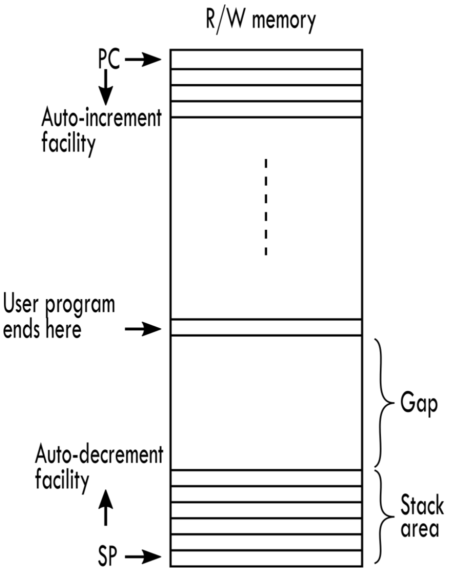


Figure 3.3: Auto-increment and Auto-decrement Functionality for the PC and SP registers

- **Flag Register :** It is an 8-bit register in which five bits represent various condition flags. These flags provide information about the result of the last executed arithmetic or logic instruction.

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

Figure 3.4: Five Flags of the Flag Register

The descriptions and conditions of the five flags are as follows:

✓ **Sgin (S) Flag :** If the most significant bit (MSB) of the result of an operation is 1, this flag is set to 1; otherwise, it is reset to 0.

✓ **Zero (Z) Flag:** If the result of an instruction is zero, this flag is set to 1; otherwise, it is reset to 0.

✓ **Auxillary Carry (AC) Flag):** If there is a carry out of bit 3 and into bit 4 resulting from the execution of an arithmetic operation, it is set to 1; otherwise, it is reset to 0.

✓ **(Parity (P) Flag):** This flag is set to 1 when the result of an operation contains an even number of 1's and is reset otherwise.

✓ **(Carry (CY) Flag):** If an instruction results in a carry (for addition operation) or borrow (for subtraction or comparison) out of bit D_7, then this flag is set; otherwise, it is reset.

3.3.3 Demultiplexing the Bus $AD_7 - AD_0$

The demultiplexing of the lower-order address bus $A_7 - A_0$ from the multiplexed address-data bus $AD_7 - AD_0$ is performed by storing the address found on the bus using an external latch 74LS373 (a set of D flip-flops). The control of this latch is triggered by the ALE signal (Address Latch Enable) when it transitions from high to low at the end of T_1 of each machine cycle, as shown in Figure 3.5.

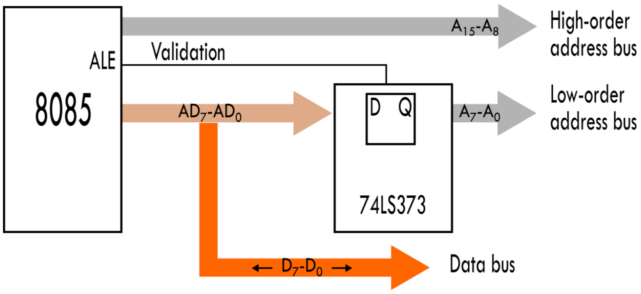


Figure 3.5: Demultiplexing the Bus $AD_7 - AD_0$

Example:

This example illustrates the addressing of a 1KB memory (Figure 3.6) while using a latch to demultiplex the lower-order address bus $A_7 - A_0$.

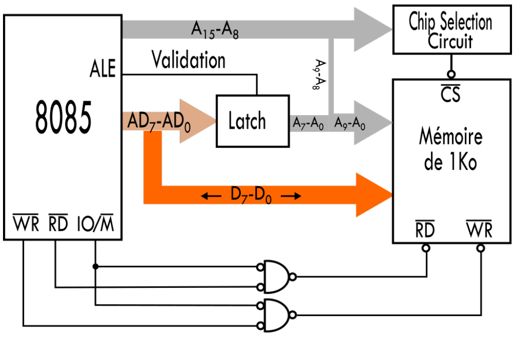


Figure 3.6: Addressing a 1 KB Memory while using Demultiplexing

3.3.4 Instruction Format

An instruction is formed by one, two, or three bytes divided into two fields:

- The first field is the task to be performed, called **opcode** (Operation Code), which is always encoded in 8 bits.
- The second field is the data to be operated on, called the **operand**, which can be an 8-bit (or 16-bit) data, an internal register, or a memory location.

Example:

LDA **1300H**

↗ ↖

Operation to be performed *Operand (address of 16-bit)*

3.3.5 Instructions Word Size

The instruction set of the 8085 is divided into three groups:

1. Instructions of size **1 byte** (no operand; opcode only).

   CMA: Complement the A register (opcode is 2FH, Mnemonic: CMA).

2. Instructions of size **2 bytes** (opcode + 1 operand of 8 bits)

   MVI A, 28H: Write the value 28H into register A.

3. Instructions of size **3 bytes** (opcode + 16-bit operand).

   LDA 1001H: Load the content of address 1001H into accumulator A. The opcode for LDA is 3AH, and the operand is a 16-bit address (2 bytes).

The previous three instructions will be written in memory, starting from address 1000H, as follows:

1000H	2FH	⟶ **CMA**	1003H	3AH	⟶ **LDA**
1001H	3EH	⟶ **MVI A**	1004H	01H	
1002H	28 H		1005H	10H	

3.3.6 Addressing Mode

The addressing mode specifies how the memory address of an operand is calculated or determined during the execution of an instruction. The 8085 microprocessor has five addressing modes.

a/ Register Addressing

This addressing mode involves any transfer or operation between two 8-bit registers.

Example:

MOV A, B : *This is the transfer of the content from B to A* (A←B); *it does not involve any memory address.*

b/ Immediate Addressing

In this addressing mode, the operand is data, and there is no operand address.

Example:

MVI A, 50H : *The value 50H will be immediately stored in A* (A←50H).

c/ Direct Addressing

In this mode, the address of the data in memory is directly specified in the operand.

Example:

LDA 2000H : *Load accumulator A with the content of address 2000H* (A←[2000H]).

d/ Indirect Addressing

In this mode, accessing data in memory is done through a register that accommodate the 16-bit address of the operand.

Example:

MOV A, M : *Load accumulator A with the content of the memory location whose address is the content of the HL register pair.*

e/ Implicit Addressing

It refers to a mode where the operand is fully absent but implicitly specified by the instruction itself or understood based on the opcode.

Example:

CMA , NOP , RAR : *These instructions have no operand and mean respectively: Complement Accumulator A, No Operation, Rotate Accumulator Right through Carry.*

3.4 Instruction set of the 8085

As the 8085 microprocessor is 8-bit, it can have up to $2^8 = 256$ instructions. However, the 8085 utilizes only 246 combinations, representing 74 instructions. These instructions can be classified into five functional categories:

- Data transfert (copy) operations.
- Arithmetic operations.
- Logic operations.
- Branching operations.
- Machine-control operations.

3.4.1 Data Transfer (Copy) Operations

This group of instructions copies data from a location called a source (register, memory, or I/O interface) to another location, called a destination (register, memory, or I/O interface), without modifying the contents of the source.

1. MOV: MOVe . *Transfer data between two registers or between a register and a memory location.*

Syntax: MOV Rd, Rs *or* MOV M, Rs *or* MOV Rd, M

Where: Rs: source register
Rd: destination register
M: address specified by HL

Example:

MOV C, D [C]←[D] *or* **MOV M, B** $[M_{HL}]$ ←[B]

2. MVI: Mo**V**e **I**mmediate. *Load an immediate 8-bit data into a specific register or memory location*

Syntax: MVI Rd, 8-bit data

Example:

MVI A, 14H [A]←14H

3. OUT: OUTput to port. *Send the contents of the accumulator (A) to an output port*

Syntax: Out address of the output port

Example:

OUT 08H

4. IN: **IN**put from port. *Read data from an input port into the accumulator (A)*

Syntax: IN address of the input port

Example:

IN 09H

5. LXI: **L**oad e**X**tended **I**mmediate. *Load a 16-bit immediate value into a register pair*

Syntax: LXI Rp, 16-bit data

Where: Rp: register pair

Example:

LXI B, 1400H [B]=14H and [C]=00H

B	C
14	00

6. LDA: **L**oa**D** **A**ccumulator. *Load an 8-bit data from a memory location into the accumulator (A)*

Syntax: LDA 16-bit address

Example:

LDA 2000H $[A] \leftarrow [M_{2000}]$

7. LDAX: **L**oa**d** **A**ccumulator e**X**tended. *Load the accumulator (A) with the 8-bit data stored at the memory location specified by the BC or DE register pair. It does not accept the HL*

Syntax: LDAX Rp

Example:

LDAX B

8. STA: **ST**ore **A**ccumulator. *Store the contents of the accumulator (A) into a specific memory location*

Syntax: STA 16-bit address

Example:

STA 3000H $[M_{3000}] \leftarrow [A]$

9. STAX: **ST**ore **A**ccumulator indirect (by e**X**tended register). *Store the contents of the accumulator (A) into the memory location specified by the BC or DE register pair.*

Syntax: STAX Rp

Example:

STAX D $[M_{DE}] \leftarrow [A]$

10. LHLD: Load **HL** register **D**irect. *Load the HL register pair with the contents of a specific memory address and the next consecutive memory address.*

Syntax: LHLD 16-bit address

Example:
LHLD 2000H

11. SHLD: Store **HL** register **D**irect. *Store the contents of the HL register pair into a specific memory address and the next consecutive memory address.*

Syntax: SHLD 16-bit address

Example:
SHLD 3000H

12. XCHG: e**XCH**an**G**e. *Exchange the contents of the DE and HL register pairs.*

Syntax: XCHG

Example:

	H	L	D	E
Before **XCHG**	A9	C2	67	89
After **XCHG**	67	89	A9	C2

13. XTHL: e**X**change **T**op stack with **HL**. *Exchange the content of H and that of L with the top of the stack.*

Syntax: XTHL

14. SPHL: Copy **HL** registers into the **S**tack **P**ointer. *Load the stack pointer (SP) with the contents of the HL register pair.*

syntax: SPHL

15. PCHL: Copy **HL** registers into the **P**rogram **C**ounter. *Load the program counter (PC) with the contents of the HL register pair.*

Syntax: PCHL

3.4.2 Arithmetic Operations

In arithmetic operations, including addition (ADD or ADI), subtraction (SUB or SUI), incrementation (8-bit INR or 16-bit INX), and decrementation (8-bit DCR or 16-bit DCX), the microprocessor assumes that the accumulator (A) is by default one of the two operands, and the result of the operations will be stored in it. The flags of the status register will also be affected by the obtained results.

1. ADD: **ADD**ition. *Add the contents of accumulator A to the contents of the register or memory location.*

Syntax: ADD R or ADD M

Where: R: 8-bit register , M: Address of a memory location

Example:

```
MVI  A, 04H
MVI  B, 03H
ADD  B       [A]←—[A] + [B]
             07H= 04H + 03H
       Flags: S=0, Z=0, CY=0
```

2. ADI: **AD**dition **I**mmediate. *Add an immediate 8-bit data to the accumulator (A).*
Syntax: ADI 8-bit data

Example:

```
MVI  A, FFH
ADI  01H       [A]←—[A] + 01H
               00H= FFH + 01H
         Flags: S=0, Z=1, CY=1
```

3. SUB: **SUB**straction. *Subtract the contents of a register or memory location from the accumulator (A).*

Syntax: SUB R or SUB M

Example:

```
MVI  A, FFH
  MVI  B, 03H
  SUB  B       [A]←—[B] - [A]
               FCH= 03H - FFH
         Flags: S=1, Z=0, CY=0
```

4. SUI: **SU**bstraction **I**mmediate. *Subtract an immediate 8-bit data from the accumulator (A).*
Syntax: SUI 8-bit data

Example:

```
MVI  A, 05H
SUI  02H       [A]←—05H - [A]
               03H= 02H - 05H
         Flags: S=0, Z=0, CY=0
```

5. INR: **IN**c**R**ementation . *Increment the contents of a specified 8-bit register or memory location (+1).*
Syntax: INR R or INR M

Example:

```
MVI  A, 05H
INR  A       [A]←—1 + [A]
```

06H= 01H + 05H
Flags: S=0, Z=0

6. INX: **IN**crement e**X**tended register . *Increment the contents of a register pair or the stack pointer (SP).*
Syntax: INX Rp

Example:

LXI SP, 2000H
INX SP SP⟵1 + SP
2001H= 1H + 2000H
Flags: S=0, Z=0

7. DCR: **D**e**CR**ementation . *Decrement the contents of a specified 8-bit register or memory location (-1).*
Syntax: DCR R or DCR M

Example:

MVI A, 01H
DCR A [A]⟵1 - [A]
0H= 1H - 01H
Flags: S=0, Z=1

8. DCX: **D**e**C**rement e**X**tended register . *Decrement the contents of a register pair or the stack pointer (SP).*
Syntax: DCX Rp

Example:

LXI B, 3000H
DCX B BC⟵1 - BC
2FFFH= 1H - 3000H
Flags: S=0, Z=0

9. ADC: **AD**d with **C**arry . *Add the contents of a specified 8-bit register or memory location along with CY to the accumulator (A).*
Syntax: ADC R or ADC M

Example:

MVI A, 30H
MVI B, F0H
ADD B [A]⟵[A] + [B]
CY=1 / 20H= 30H + F0H
ADC B [A]⟵CY + [A] + [B]
11H= 1H + 20H + F0H

10. ACI: **A**dd with **C**arry **I**mmediate . *Add an immediate 8-bit data along with CY to the accumulator (A).*

syntax: ACI 8-bit data

Example:

```
MVI  A, 30H
MVI  B, F0H
ADD  B        [A]⟵[A] + [B]
                 20H= 30H + F0H
ACI  34H       [A]⟵CY + 34H + [A]
                 55H= 1H + 34H + 20H
```

11. SBB: **S**u**B**tract with **B**orrow . *Subtract an immediate 8-bit data and the contents of CY from the accumulator (A).*

syntax: SBB R or SBB M

Example:

```
MVI  A, 30H
MVI  B, 20H
ADI  FFH       [A]⟵FFH + [A]
            CY=1 / 2FH= FFH + 30H
SBB  B        [A]⟵CY - [B] - [A]
                0EH= 1 - 20H - 2FH
```

12. SBI: **S**u**B**tract **I**mmediate with borrow . *Subtract an immediate 8-bit data and CY from the contents of the accumulator (A).*

Syntax: SBI 8-bit data

Example:

```
MVI  A, 30H
ADI  FF        [A]⟵FF + [A]
        CY=1 / 2FH= FFH + 30H
SBI  20H       [A]⟵CY - 20H - [A]
                 0EH= 1H - 20H - 2FH
```

13. DAA: **D**ecimal **A**just **A**ccumulator . *Converts the content of the accumulator into two BCD values.*

Syntax: DAA

Example:

```
MVI  A, 38H       [A]⟵38H
MVI  B, 45H       [B]⟵45H
ADD  B        [A]⟵[B] + [A]
            7DH= 38H + 45H
DAA        [A]⟵83H
```

$$\begin{array}{l l l l l}
|\ \ 38 & \underline{BCD} & 0011\ 1000 & & 0111\ 1101 \\
|\ +45 & \underline{BCD} & 0100\ 0101 & + & 0110 \\
|\ \overline{=83} & & \overline{01111101} & & \overline{10000011} \\
| & & \quad \bar{7}\ \ \bar{D} & & \quad \bar{8}\ \ \bar{3}
\end{array}$$

3.4.3 Logic Operations

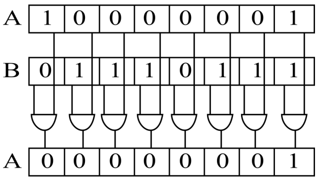
These instructions perform logical operations on the contents of the accumulator. These operations will result in clearing the CY (Carry) flag.

1. ANA: ANd with **A**ccumulator . *Performs a logical AND operation between the accumulator (A) and the specified register or memory location.*

Syntax: ANA R or ANA M

Example:

```
MVI  A, 81H
MVI  B,77H
ANA  B    [A]←—[B] AND [A]
              01H= 77HH AND 81H
```



2. ANI: ANd **I**mmediate. *Performs a logical AND operation between the accumulator (A) and an immediate 8-bit data.*

Syntax: ANI 8-bit data

Example:

```
MVI  A, 55H
ANI  01H       [A]←—01H AND [A]
                01H= 01H AND 55H
```

3. ORA: OR with **A**ccumulator . *Performs a logical OR operation between the accumulator (A) and the specified register or memory location.*

Syntax: ORA R or ORA M

Example:

```
MVI  A, 81H
MVI  B, 7EH
ORA  B       [A]←—[B] OR [A]
            FFH= 7EH OR 81H
```

4. ORI: OR Immediate. *Performs a logical OR operation between the accumulator (A) and an immediate 8-bit data.*

Syntax: ORI 8-bit data

Example:

```
MVI  A, 55H
ORI  02H       [A]←—02H OR [A]
                57H= 02H OR 55H
```

5. XRA: Xo**R** with **A**ccumulator . *Performs a logical XOR operation between the accumulator (A) and the specified register or memory location..*

Syntax: XRA R or XRA M

Example:

```
MVI  A, 80H
MVI  B, 7EH
XRA  B        [A]⟵[B]  XOR  [A]
             FEH=  7EH  XOR  80H
```

6. XRI: **X**o**R** **I**mmediate. *Performs a logical XOR operation between the accumulator (A) and an immediate 8-bit data.*

Syntax: XRI 8-bit data

Example:

```
MVI  A, 55H
XRI  02H        [A]⟵02H  XOR  [A]
                 57H=  02H  XOR  55H
```

7. CMA: **C**o**M**plement the **A**ccumulator. *Complements all the bits of the accumulator (A) (using 1's complement).*

Syntax: CMA

Example:

```
MVI  A, 55H
CMA              1010 1010  ⟵  0101 0101
                   A    A          5    5
```

3.4.4 Additional Logic Operations

A. Rotate Instructions

These are operations that rotate the bits of the accumulator to the left or right, with or without CY (RLC, RAL, RRC, RAR)

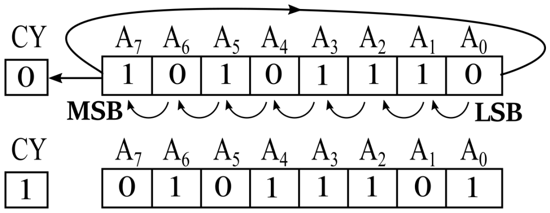
1. RLC: **R**otate accumulator **L**eft . *Rotates all the bits of the accumulator to the left. The MSB (Most Significant Bit)* (A_7) *is shifted into the Carry flag CY and into the LSB (Least Significant Bit)* (A_0) *of the accumulator.*

Note: *The left rotation of A results in multiplying the value of A by 2 (provided that the* A_7 *bit is 0).*

Example:

Before **RLC** [A]=AEH, CY=0

After **RLC** [A]=5DH, CY=1

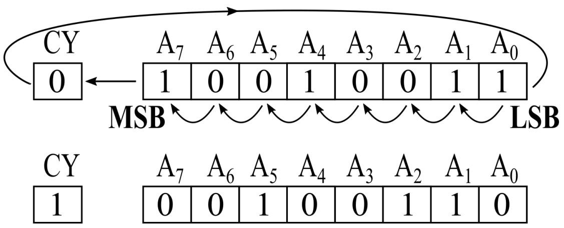

CY
A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0
0
1 0 1 0 1 1 1 0
MSB
LSB
CY
A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0
1
0 1 0 1 1 1 0 1

2. RAL: **R**otate **A**ccumulator **L**eft through carry . *Rotate accumulator A left through carry with the previous content of the carry flag (CY). Each bit of A is shifted to the left by one position. The MSB* (A_7) *is shifted into the Carry flag, and the previous content of CY is shifted into the LSB* (A_0) *of the accumulator.*

Example:

Before **RAL** [A]=(93)H, CY=1

After **RAL** [A]=26H, CY=0


CY
A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0
0
1 0 0 1 0 0 1 1
MSB
LSB
CY
A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0
1
0 0 1 0 0 1 1 0

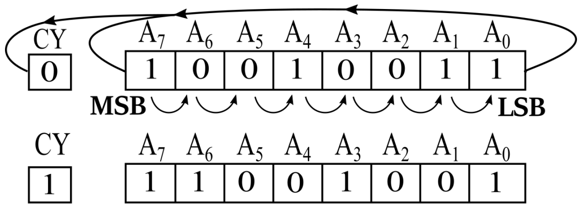
3. RRC: **R**otate accumulator **R**ight . *Rotate accumulator A right. Each bit of A is shifted to the right by one position. The LSB (*A_0*) becomes the MSB (*A_7*). The shifted-out LSB is also copied into the CY (Carry) flag.*

Note: *Rotating A to the right results in dividing the value of A by 2 (provided that bit* A_0 *is 0).*

Example:

Before **RRC** [A]=93H, CY=0

After **RRC** [A]=C9H, CY=1

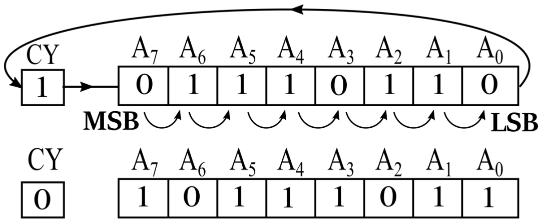

CY
A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0
0
1 0 0 1 0 0 1 1
MSB
LSB
CY
A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0
1
1 1 0 0 1 0 0 1

4. RAR: **R**otate **A**ccumulator **R**ight through carry . *Right rotation of accumulator A through carry. Each bit of A is shifted to the right by one position. The content of CY becomes the MSB (*A_7*), and the LSB (*A_0*) is stored in CY.*

Example:

Before **RAR** [A]=76H, CY=1

After **RAR** [A]=BBH, CY=0


CY
A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0
1
0 1 1 1 0 1 1 0
MSB
LSB
CY
A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0
0
1 0 1 1 1 0 1 1

B. Compare Instructions

The 8085 has two comparison instructions, namely CMP and CPI.

1. CMP: **C**o**MP**are register . *Compare the contents of the accumulator (A) with a specified register or memory location. It performs a subtraction internally but does not store the result.*

syntax: CMP R or CMP M
If [A]=[R/M] → Z=1 otherwise Z=0
If [A]<[R/M] → CY=1 otherwise CY=0

Example:

```
MVI   B, 00H
MVI   A, 09H
```

CMP B [A]>[B]$\longrightarrow$CY=0, Z=0

2. CPI: Com**P**are **I**mmediat. *Compare the accumulator (A) with an immediate 8-bit data.*

syntax: CPI 8-bit data
If [A]=8-bit data → Z=1 otherwise Z=0
If [A]<8-bit data → CY=1 otherwise CY=0

Example:

MVI A, 09H
CPI 10H [A]<10H$\longrightarrow$CY=1, Z=0

3.4.5 Branching Operations

The branching instructions change the flow of program execution. They are divided into two types:

a. Unconditional branch (jump) instructions

Unconditional jump instructions alter the flow of program execution without any conditions to a specified memory address.

1. JMP: **J**u**MP** . *Jump to a specified address.*
syntax: JMP 16-bit address

Example:

```
         MVI   B, 68H
         MVI   C, 5AH
         INR   B
         JMP   FINISH
         DCR   C
FINISH:  MOV   A, B
         ADD   C
```

2. CALL: *Calls the addressed subroutine program.*

• *Subroutine is a group of instructions written separately from the main program to perform a task that will be used repeatedly in different locations of the main program.*

A subroutine is a group of instructions the program

syntax: CALL 16-bit address of the subroutine

The execution of this instruction allows:

- Save the content of the PC in the stack and copy the address of the subroutine into the PC. $[SP-1] \longleftarrow [PCH]$
$[SP-2] \longleftarrow [PCL]$

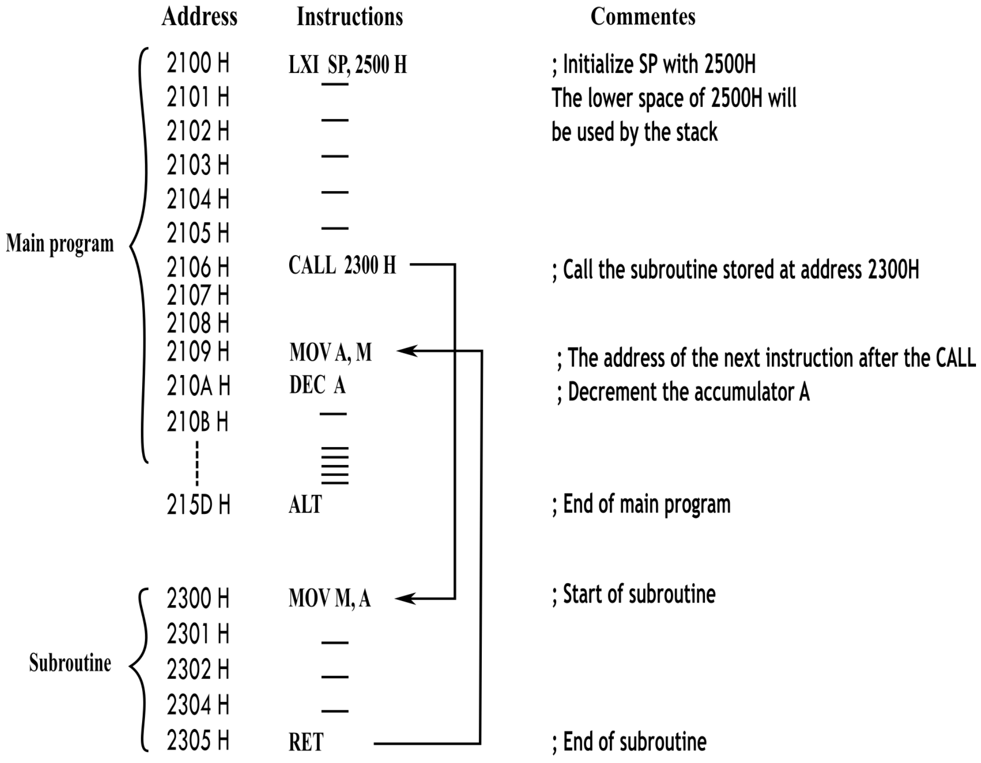
- Decrement SP by 2. $[PC] \longleftarrow$Adresse 16-bit

2. RET: **RET**urn. *Unconditional return to the main program.*
syntax: RET

The execution of this instruction allows:

- Retrieve the initial value of the PC from the stack. $[PCL] \longleftarrow [M_{SP}]$ $[PCH] \longleftarrow [M_{SP+1}]$
- Increment SP by 2.

Example:



Address	Instructions	Commentes
2100 H	LXI SP, 2500 H	; Initialize SP with 2500H The lower space of 2500H will be used by the stack
2101 H	—	
2102 H	—	
2103 H	—	
2104 H	—	
2105 H	—	
2106 H	CALL 2300 H	; Call the subroutine stored at address 2300H
2107 H		
2108 H		
2109 H	MOV A, M	; The address of the next instruction after the CALL
210A H	DEC A	; Decrement the accumulator A
210B H	—	
215D H	ALT	; End of main program
2300 H	MOV M, A	; Start of subroutine
2301 H	—	
2302 H	—	
2304 H	—	
2305 H	RET	; End of subroutine

b. Conditional Branch Instructions

Conditional branch instructions alter the flow of program execution based on certain conditions. These conditions are typically determined by the states of the status register flags.
The branching instructions are summarized as follows:

- **JZ Adresse** (**J**ump on **Z**ero)
  - Jump to the specified address if the result is zero (Z=1)
- **JNZ Adresse** (**J**ump on **N**ot **Z**ero)
  - Jump to the specified address if Z=0.
- **JC Adresse** (**J**ump on **C**array)
  - Jump to the specified address if CY=1.
- **JNC Adresse** (**J**ump on **N**o **C**array)
  - Jump to the specified address if CY=0.

- **JP Adresse** (**J**ump on **P**lus)
  - Jump to the specified address if S=0.

- **JM Adresse** (**J**ump on **M**inus)
  - Jump to the specified address if S=1.

- **JPE Adresse** (**J**ump if **P**arity is **E**ven)
  - Jump to the specified address if P=1.

- **JPO Adresse** (**J**ump if **P**arity is **O**dd)
  - Jump to the specified address if P=0.

Note

There are two instructions for manipulating the CY flag, namely:

1. **STC** (**S**e**T** **C**arry) **:** Setting the Carry Flag to 1 CY $\longleftarrow$ 1
2. **CMC** (**C**o**M**plement the **C**arry flag) **:** Complementing CY to 1 de CY $\longleftarrow \overline{\text{CY}}$

Subroutine Instructions

In addition to the call instruction and the unconditional subroutine return instruction, there are other conditional call and return instructions.

a. Conditional Subroutine Calls

- ▶ **CZ Adresse** (**C**all if **Z**ero) ; Call subroutine if Z=1.
- ▶ **CNZ Adresse** (**C**all if **N**ot **Z**ero) ; Call subroutine if Z=0.
- ▶ **CC Adresse** (**C**all if **C**arry) ; Call subroutine if CY=1.
- ▶ **CNC Adresse** (**C**all if **N**ot **C**arry) ; Call subroutine if CY=0.
- ▶ **CP Adresse** (**C**all if **P**ositive) ; Call subroutine if S=0.
- ▶ **CM Adresse** (**C**all if **M**inus) ; Call subroutine if S=1.
- ▶ **CPE Adresse** (**C**all if **P**arity is **E**ven) ; Call subroutine if P=1.
- ▶ **CPO Adresse** (**C**all if **P**arity is **O**dd) ; Call subroutine if P=0

b. Conditional Subroutine Returns

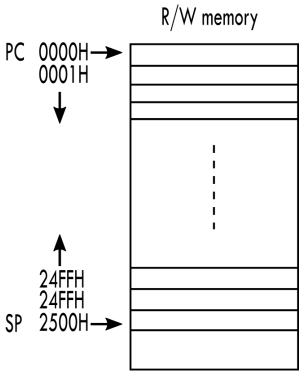
- ▷ **RZ Adresse** (**R**eturn if **Z**ero) ; Return from subroutine if Z=1.
- ▷ **RNZ Adresse** (**R**eturn if **N**o **Z**ero) ; Return from subroutine if Z=0.
- ▷ **RC Adresse** (**R**eturn if **C**arry) ; Return from subroutine if CY=1.
- ▷ **RNC Adresse** (**R**eturn if **N**o **C**arry) ; Return from subroutine if CY=0.
- ▷ **RP Adresse** (**R**eturn if **P**ositive) ; Return from subroutine if S=0.
- ▷ **RM Adresse** (**R**eturn if **M**inus) ; Return from subroutine if S=1.
- ▷ **RPE Adresse** (**R**eturn if **P**arity is **E**ven) ; Return from subroutine if P=1.
- ▷ **RPO Adresse** (**R**eturn if **P**arity is **O**dd) ; Return from subroutine if P=0

3.4.6 Stack Instructions

The stack is a section of the RAM memory designated by the stack pointer (SP), typically located towards the end of memory, used to temporarily store bytes during the execution of a program. The SP points to the top of the stack, and data is stored at addresses lower than the address pointed to by SP.

Example:

LXI SP, 2500H



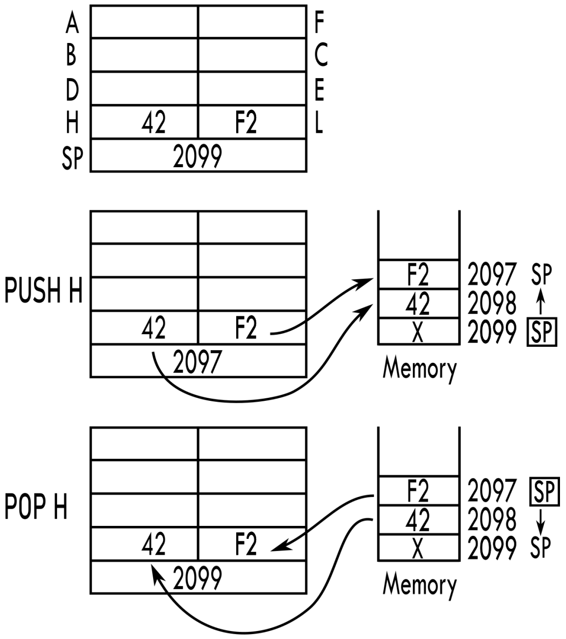
Besides storing the return address of the main program on the stack after the execution of the CALL instruction, the contents of pair registers can also be stored/restored on/from the stack by the PUSH/POP instruction.

syntax: PUSH Rp and POP Rp

As Rp: represents one of the registers pair B, D, or H or the PSW register (Program Status Word), which is the combination of the accumulator A and the flag register.

Example:

Address	**Instruction**
2000H	**LXI SP, 2099H**
2003H	**LXI H, 42F2H**
2006H	**PUSH H**
2007H	**POP H**



3.4.7 I/O and Machine-control Operations

The machine-control instructions are summarized in the following table.

3.5 Op-code and mnemonic of 8085 instructions

The table 3.5 below gathers the op-codes and mnemonics of all 8085 instructions..

Mnemonic	Operand	Operation	Explanation
NOP	No operand	No operation	No operation is performed
HLT	No operand	Halt and enter wait state	The microprocessor completes the execution of the current instruction and stops execution
DI	No operand	**D**isable **I**nterrupts	All interrupts are disabled except TRAP.
EI	No operand	**E**nable **I**nterrupts	All interrupts are enabled
RIM	No operand	**R**ead **I**nterrupt **M**ask	It is used to check the status of interrupts 7.5, 6.5, 5.5 and to read the serial input bit
SIM	No operand	**S**et **I**nterrupt **M**ask	It is used to implement interrupts 7.5, 6.5, 5.5, and serial data output

3.6 The 8085 Machine Cycles

The 8085 microprocessor is designed to execute 74 different instruction types. Instructions can have opcode only or opcode and operand. To execute an instruction, the 8085 needs to perform one or more machine cycles, and each machine cycle is divided into T-states. Basically the microprocessor external communications functions can be divided into three categories:

- Memory Read and Write
- I/O Read and Write
- Interrupt Request Acknowledge

In this section, we will focus on Opcode Fetch, Memory Read and Memory Write.

3.6.1 Opcode Fetch Machine Cycle

To fetch the opcode byte, the CPU needs four T-states to perform the following steps:

Step 1: During the first clock period T_1, the microprocessor places the high-order memory address on the address lines $A_{15} - A_8$, the low-order memory address on the bus $AD_7 - AD_0$, increments the program counter and the ALE signal goes high. To differentiate an opcode from a data byte, the status signals will be $(IO/\overline{M} = 0,\ S_1 = 1,\ S_0 = 1)$.

Step 2: During the second clock period T_2, the control unit sends the control signal $\overline{RD}$ to enable the memory chip (it is active during two clock periods).

Step 3: During the third T-state T_3, the opcode is sent via the data bus to the instruction register (IR).

Step 4: During T_4, the opcode is placed in the instruction decoder to be decoded, and then executed.

3.6.2 Memory Read Machine Cycle

The memory read machine cycle is needed to execute a 2-byte or a 3-byte instruction. In this section, a case of a 2-byte instruction is examined. During T_4 (previous Steps 4), the 8085 determines that a second byte needs to be read, which will be performed in three T-states.

Hex	Mnémonique		Hex	Mnémonique		Hex	Mnémonique		Hex	Mnémonique	
CE	ACI	8-bit	2B	DCX	H	52	MOV	D,D	E5	PUSH	H
8F	ADC	A	3B	DCX	SP	53	MOV	D,E	F5	PUSH	PSW
88	ADC	B	F3	DI		54	MOV	D,H	17	RAL	
89	ADC	C	FB	EI		55	MOV	D,L	1F	RAR	
8A	ADC	D	76	HLT		56	MOV	D,M	D8	RC	
8B	ADC	E	BD	IN	8-Bit	5F	MOV	E,A	C9	RET	
8C	ADC	H	3C	INR	A	58	MOV	E,B	20	RIM	
8D	ADC	L	04	INR	B	59	MOV	E,C	07	RLC	
8E	ADC	M	0C	INR	C	5A	MOV	E,D	F8	RM	
87	ADD	A	14	INR	D	5B	MOV	E,E	D0	RNC	
80	ADD	B	1C	INR	E	5C	MOV	E,H	C0	RNZ	
81	ADD	C	24	INR	H	5D	MOV	E,L	F0	RP	
82	ADD	D	2C	INR	L	5E	MOV	E,M	E8	RPE	
83	ADD	E	34	INR	M	67	MOV	H,A	E0	RPO	
84	ADD	H	03	INX	B	60	MOV	H,B	0F	RRC	
85	ADD	L	13	INX	D	61	MOV	H,C	C7	RST	0
86	ADD	M	23	INX	H	62	MOV	H,D	CF	RST	1
C6	ADI	8-Bit	33	INX	SP	63	MOV	H,E	D7	RST	2
A7	ANA	A	DA	JC	16-Bit	64	MOV	H,H	DF	RST	3
A0	ANA	B	FA	JM	16-Bit	65	MOV	H,L	E7	RST	4
A1	ANA	C	C3	JMP	16-Bit	66	MOV	H,M	EF	RST	5
A2	ANA	D	D2	JNC	16-Bit	6F	MOV	L,A	F7	RST	6
A3	ANA	E	C2	JNZ	16-Bit	68	MOV	L,B	FF	RST	7
A4	ANA	H	F2	JP	16-Bit	69	MOV	L,C	C8	RZ	
A5	ANA	L	EA	JPE	16-Bit	6A	MOV	L,D	9F	SBB	A
A6	ANA	M	E2	JPO	16-Bit	6B	MOV	L,E	98	SBB	B
E6	ANI	8-Bit	CA	JZ	16-Bit	6C	MOV	L,H	99	SBB	C
CD	CALL	16-Bit	3A	LDA	16-Bit	6D	MOV	L,L	9A	SBB	D
DC	CC	16-Bit	0A	LDAX	B	6E	MOV	L,M	9B	SBB	E
FC	CM	16-Bit	1A	LDAX	D	77	MOV	M,A	9C	SBB	H
2F	CMA		2A	LHLD	16-Bit	70	MOV	M,B	9D	SBB	L
3F	CMC		01	LXI	B,16-Bit	71	MOV	M,C	9E	SBB	M
BF	CMP	A	11	LXI	D,16-Bit	72	MOV	M,D	DE	SBI	8-Bit
B8	CMP	B	21	LXI	H,16-Bit	73	MOV	M,E	22	SHLD	16-Bit
B9	CMP	C	31	LXI	SP,16-Bit	74	MOV	M,H	30	SIM	
BA	CMP	D	7F	MOV	A,A	75	MOV	M,L	F9	SPHL	
BB	CMP	E	78	MOV	A,B	3E	MVI	A,8-Bit	32	STA	16-Bit
BC	CMP	H	79	MOV	A,C	06	MVI	B,8-Bit	02	STAX	B
BD	CMP	L	7A	MOV	A,D	0E	MVI	C,8-Bit	12	STAX	D
BE	CMP	M	7B	MOV	A,E	16	MVI	D,8-Bit	37	STC	
D4	CNC	16-Bit	7C	MOV	A,H	1E	MVI	E,8-Bit	97	SUB	A
C4	CNZ	16-Bit	7D	MOV	A,L	26	MVI	H,8-Bit	90	SUB	B
F4	CP	16-Bit	7E	MOV	A,M	2E	MVI	L,8-Bit	91	SUB	C
EC	CPE	16-Bit	47	MOV	B,A	36	MVI	M,8-Bit	92	SUB	D
FE	CPI	8-Bit	40	MOV	B,B	00	NOP		93	SUB	E
E4	CPO	16-Bit	41	MOV	B,C	B7	ORA	A	94	SUB	H
CC	CZ	16-Bit	42	MOV	B,D	B0	ORA	B	95	SUB	L
27	DAA		43	MOV	B,E	B1	ORA	C	96	SUB	M
09	DAD	B	44	MOV	B,H	B2	ORA	D	D6	SUI	8-Bit
19	DAD	D	45	MOV	B,L	B3	ORA	E	EB	XCHG	
29	DAD	H	46	MOV	B,M	B4	ORA	H	AF	XRA	A
39	DAD	SP	4F	MOV	C,A	B5	ORA	L	A8	XRA	B
3D	DCR	A	48	MOV	C,B	B6	ORA	M	A9	XRA	C
05	DCR	B	49	MOV	C,C	F6	ORI	8-Bit	AA	XRA	D
0D	DCR	C	4A	MOV	C,D	D3	OUT	8-Bit	AB	XRA	E
15	DCR	D	4B	MOV	C,E	E9	PCHL		AC	XRA	H
1D	DCR	E	4C	MOV	C,H	C1	POP	B	AD	XRA	L
25	DCR	H	4D	MOV	C,L	E1	POP	D	AE	XRA	M
2D	DCR	L	4E	MOV	C,M	E1	POP	H	EE	XRI	8-Bit
35	DCR	M	57	MOV	D,A	F1	POP	PSW	E3	XTHL	
0B	DCX	B	50	MOV	D,B	C5	PUSH	B			
1B	DCX	D	51	MOV	D,C	D5	PUSH	D			

Step 1: During T_1, the new address is placed on the address bus, and the program counter is incremented. Status signals will become ($IO/\overline{M} = 0,\ \ S_1 = 1,\ \ S_0 = 0$), and the ALE goes high.
Step 2: During T_2, the control signal $\overline{RD}$ becomes active to enable the memory chip, and the memory places the data byte on the data bus.
Step 3: During T_3, the 8085 reads the byte and performs the operation instructed by the instruction.

Note *I/O Read and Write machine cycles are almost similar to Memory Read and Write machine cycles respectively. The difference here is in the $IO/\overline{M}$ signal status which remains 1 indicating that these machine cycles are related to I/O operations. These machine cycles take 3T states.*

Example

The following 2-byte instruction is considered:

Address	**Machine code**	**Instruction**
2000H	$\boxed{0\,0\,1\,1\,1\,1\,1\,0} \rightarrow$ 3EH	$\underbrace{MVI\ \ A}_{\substack{\text{Opcode}\\ \text{M.C.}\\ \text{4 C.C.}}},\underbrace{32H}_{\substack{\text{Operand}\\ \text{M.C.}\\ \text{3 C.C.}}}$; Load 32H in A
2001H	$\boxed{0\,0\,1\,1\,0\,0\,1\,0} \rightarrow$ 32H	

As : M.C. stands for **M**achine **C**ycle
C.C. stands for **C**lock **C**ycle

Figure 3.7 illustrates the timing diagram for execution of the instruction (MVI A, 32H)."

3.7 Execution Time of an Instruction

The execution time of an instruction is related to the number of its machine cycles (1, 2, or 3 machine cycles). Similarly, each machine cycle is related to the number of its clock cycles; consequently, the execution time of an instruction (Tins) is calculated according to the following equation:

$$Tins = \frac{\sum(Clock\ \ Cycle)}{Clock\ \ Frequncey} \tag{3.1}$$

If the instruction from the previous example (MVI A, 32H) is taken and the clock frequency is assumed to be 2MHz, it is found that:

□ T-state = clock period(1/f) $T = 1/f = 1/2MHz = 0.5\mu s$
□ Execution time for Opcode Fetech: $4CC = 4x0.5\mu s = 2\mu s$
□ Execution time for Memory Read: = $3CC = 3x0.5\mu s = 1.5\mu s$
□ Execution time for Instruction Tins $= 7CC = 7x0.5\mu s = 3.5\mu s$

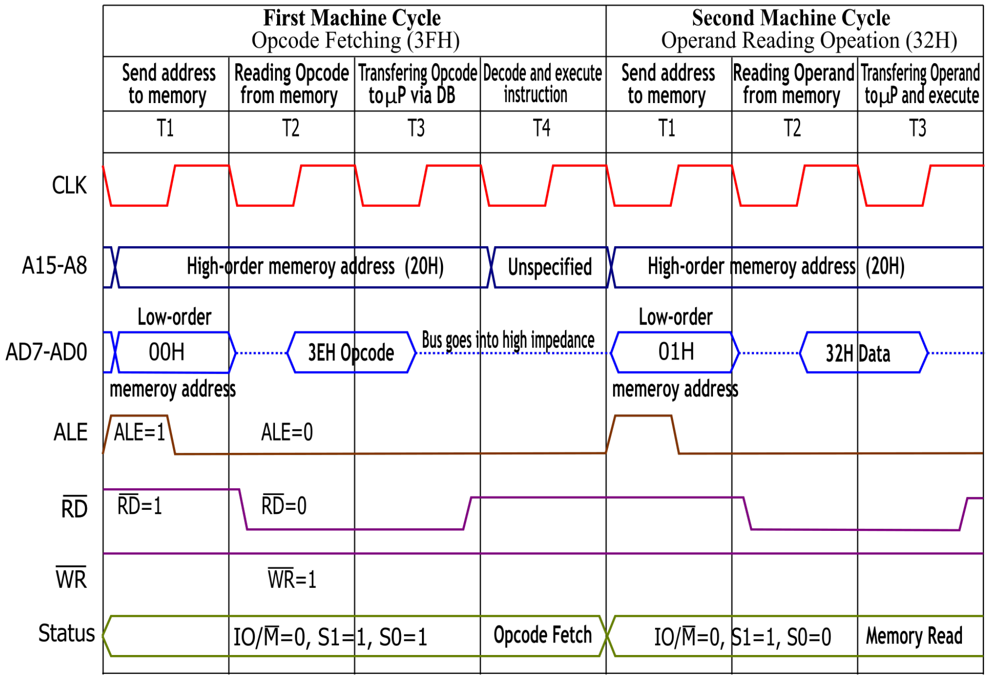


Figure 3.7: Timing Diagram for Execution of the Instruction: MVI A, 32H

3.8 Loops and Counters in Assembly Language

The programming technique used to instruct the microprocessor to repeat tasks is called looping. A loop is set up by instructing the microprocessor to change the sequence of execution and perform the task again. This process is accomplished by using Jump instructions. In addition, techniques such as counting and indexing are used in setting up a loop. Loops can be classified into two groups:

□ Continuous loop
□ Conditional loop

3.8.1 Continuous Loop

A continuous loop repeats a task continuously and is set up using an unconditional Jump instruction, as shown in the flowchart in Figure 3.8. Typical examples of such a program include a continuous counter or a continuous monitor system.

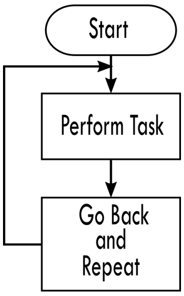


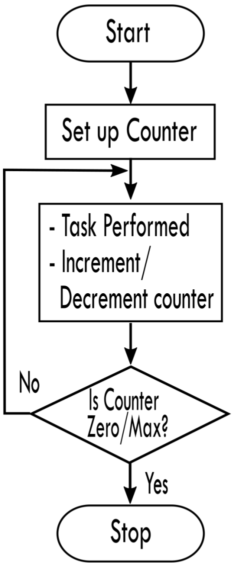
Figure 3.8: Flowchart of Continuous Loop

3.8.2 Conditional Loop

A conditional loop is set up by the conditional Jump instructions. These instructions check flags (S, Z, CY, P) and repeat the specified tasks if the conditions are satisfied. These loops usually include counting and indexing.

3.8.3 Conditional Loop and Counter

A counter is a typical application of the conditional loop, and this can be accomplished by setting up a counter. The counter is configured by loading a general-purpose register (B, C, D, E, H, L) or a pair register (for a value > 255) with a specific value. Then, the DCR/DCX or INR/INX instructions are used to decrement/increment the content of this register. Conditional branching is established until the required value (either '0' or the maximum value) is reached.



Example:

- By using a general-purpose register :

```
       MVI   C, 15H
loop:  DCR   C
       JNZ   loop
```

- By using a pair register :

```
       LXI   B, 1000H   ; Load BC with 1000H
loop:  DCX   B
       MOV   A, C
       ORA   B          ; OR between B and A (C)
       JNZ   loop
```

Note: *The two instructions MOV and ORA have been added to test the final value of the counter since DCX and INX work with pair registers and do not affect the flags.*

3.8.4 Generation of delays

The generation of delays and pauses is in great demand and proves to be very beneficial in programming. Delays are used in various applications, including data display, process control, etc.

The delay time is determined by calculating the number of machine cycles required for each instruction inside the loop, and each instruction has a certain number of clock cycles (CC). The sum of all instructions' clock cycles within the loop will give the total number of clock cycles required to execute the loop once. Then, this number will be multiplied by how many times this loop is repeated to obtain the global number of cycles. This final number of CC will be divided by the clock frequency to obtain the delay time, as given in the following formula:

$$Delay = \frac{Number\ \ of\ \ CC}{Clock\ \ Frequency} \tag{3.2}$$

⋆ *A loop can also be used to create a delay time in a program.*

Example:

In this example, an 8-bit register is used

Label	Opcode	Operand	Comment	Number of cycles
	MVI C,	**FFH**	*;Load C with FFH*	**7**
loop:	**DCR**	**C**	*;Decrement C*	**4**
	JNZ	**loop**	*;If* $Z \neq 0$*, jump to loop*	**10/7**

The number of cycles generating the delay for this loop is calculated by the following relationship:

$$T_{delay} = T_0 + T_L \tag{3.3}$$

As: T_0 is the number of cycles outside the loop
T_L is the number of cycles within the loop.

We have: $T_0 = 7$ and $T_L = (4 \times 255) + (10 \times 254) + 7$

So: $T_{delay} = 7 + 3567 = 3574\, cycles$

and $Delay = 3574/2MHz = 3574 \times 0.5\mu s$

$Delay = 1787\mu s = 1.787ms$

⋆ *To achieve a longer delay time, it is necessary to switch to using a register pair.*

Example:

In this example, a register pair is used.

Label	Opcode	Operand	Comment	Number of cycles
	LXI B,	**1000H**		**10**
loop:	**DCX B**			**4**
	MVI C,	**A**	*;Load the content of C*	**4**
	ORA B		*; OR between A and B to set Z flag*	**4**
	JNZ	**loop**		**10/7**

As: $T_0 = 10$

and $T_L = (6 \times 4096) + (4 \times 4096) + (4 \times 4096) + (10 \times 4095) + 7 = 98301$

So: $T_{delay} = 10 + 98301 = 98311\, cycles$

and $Delay = 98311/2MHz = 491555.5\mu s$

$Delay \approx 0.5s$

3.8.5 Generating Delay Using Nested Loops

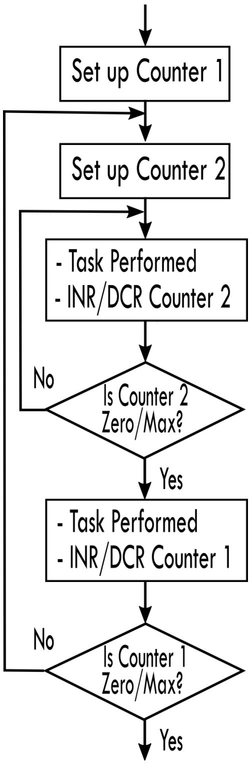
Nested loops can be created using two registers, one for the inner loop and one for the outer loop.

Example:

Program

```
            MVI   B, 10H          7T
 loop_2:    MVI   C, FFH          7T
 loop_1:    DCR   C               4T
            JNZ   loop_1          10/7T
 loop_1 :    DCR   B              4T
            JNZ   loop_2          10/7T
```

Flowchart



Generated delay

$T_{01} = 7T$

$T_{B1} = (4 \times 255) + (10 \times 254) + 7 = 3567T$

$T_{B2} = (7 \times 16) + (3567 \times 16) + (4 \times 16) + (10 \times 15) + 7 = 57405T$

$T_{delay} = 7 + 57405 = 57412T$

$Delay = 57412 \times 0.5\mu s = 28706\mu s = 28,706ms$

3.9 Interrupts

3.9.1 Introduction

A microprocessor-based system can communicate with an external device or a peripheral (mouse, keyboard, ADC, ...) in two ways:

- **Periodic Polling**: The microprocessor cyclically checks or reads the status or data from the input/output ports of the peripherals.
- **Interrupt**: The microprocessor temporarily suspends its normal operation when a device or peripheral requires immediate attention.

So, an interrupt is an event that temporarily halts the normal execution of a program to respond promptly to an external event triggered by a peripheral device (hardware interrupt) that requires immediate attention, such as having new data to be processed, as the interrupt can be initiated by a signal generated internal to the processor (software interrupt). It directs the microprocessor to a specific subroutine called the 'interrupt service routine'.

Figure 3.9 illustrates the principle of communication between a microprocessor and a peripheral interface through interrupt.

Interrupt Process and Interrupt Handling by the 8085

If the ability to handle interrupts by the 8085 was enabled by writing the EI (Enable Interrupt) instruction in the main program, the interrupt process can be described by the following steps:

1. In the second T-state of the last machine cycle of every instruction, the 8085 processor checks the interrupt signal from the peripheral to know whether an interrupt request is made or not.