# Unit 2: Intel 8085 (8 Hrs.)

BIM-3rd Semester

Prime College

Rolisha Sthapit

## CONTENTS

- Functional Block Diagram and Pin Configuration; Timing and control Unit; Registers; Data and Address Bus; Intel 8085 Instructions; Operation Code and Operands; Addressing Modes; Interrupts; Flags; Institutions and Data Flow inside 8085; Basic Assembly Language Programming Using 8085 Instruction Sets.

## 8085 Microprocessor & its Operation

- 8085 is pronounced as "eighty-eighty-five" microprocessor.
- It is an 8-bit microprocessor designed by Intel in 1977 using NMOS technology (**N-type metal-oxide-semiconductor logic** )
- It is a complete 8 bit parallel central processing unit.
- The main components of 8085 are array of registers, the ALU, encoder/decoder, and timing and control circuits linked by an internal data bus.

It has the following configuration –

- 8-bit data bus

- 16-bit address bus, which can address up to 64KB

- A 16-bit program counter

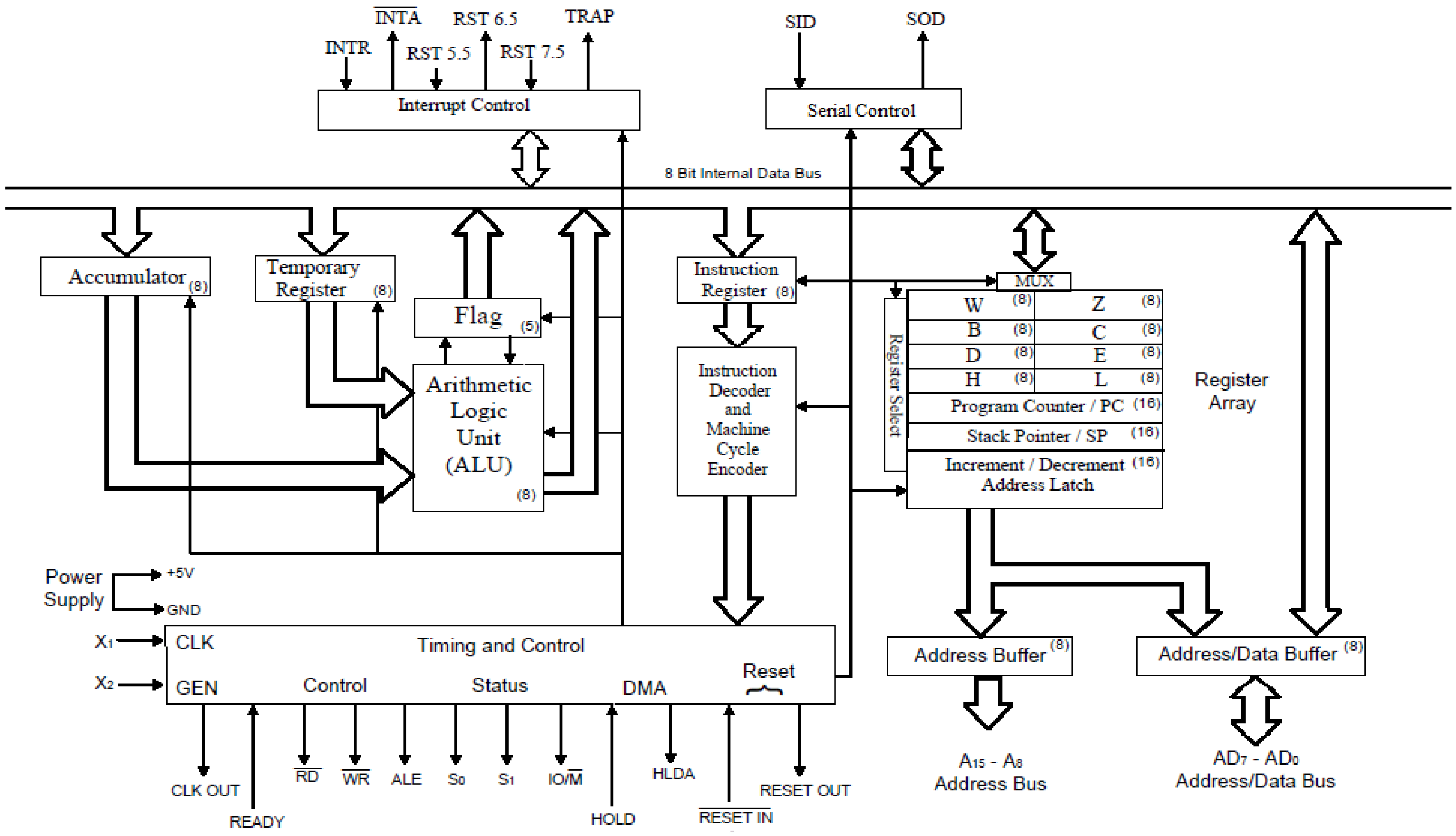
- A 16-bit stack pointer

- Six 8-bit registers arranged in pairs: BC, DE, HL

- Requires +5V supply to operate at 3.2 MHZ single phase clock

- It is used in washing machines, microwave ovens, mobile phones, etc

# 8085 Microprocessor Architecture & Functional Units



8085 consists of the following functional units −

**1) Accumulator**

It is an 8-bit register used to perform arithmetic, logical, I/O & LOAD/STORE operations. It is connected to internal data bus & ALU.

**2) Arithmetic and logic unit**

As the name suggests, it performs arithmetic and logical operations like Addition, Subtraction, AND, OR, etc. on 8-bit data.

**3) General purpose register**

There are 6 general purpose registers in 8085 processor, i.e. B, C, D, E, H & L. Each register can hold 8-bit data. These registers can work in pair to hold 16-bit data and their pairing combination is like B-C, D-E & H-L.

**4) Program counter**

It is a 16-bit register used to store the memory address location of the next instruction to be executed. Microprocessor increments the program whenever an instruction is being executed, so that the program counter points to the memory address of the next instruction that is going to be executed.

**5) Stack pointer**

It is also a 16-bit register works like stack, which is always incremented/decremented by 2 during push & pop operations.

The stack is a LIFO (last in, first out) data structure implemented in the RAM area and is used to store addresses and data when the microprocessor branches to a subroutine. Then the return address used to get pushed on this stack. Also to swap values of two registers and register pairs we use the stack as well.

The Stack Pointer register will hold the address of the top location of the stack.

In case of PUSH operation, the SP register gets decreased by 2 and new data item used to insert on to the top of the stack. On the other hand, in case of POP operation, the data item will have to be deleted from the top of the stack and the SP register will get increased by the value of 2.

**6) Temporary register**

It is an 8-bit register, which holds the temporary data of arithmetic and logical operations.

Wand Z registers are temporary registers. These registers are used to hold 8-bit data during the execution of some instructions. These registers are not available for the programmer since 8085Microprocessor Architecture uses them internally.

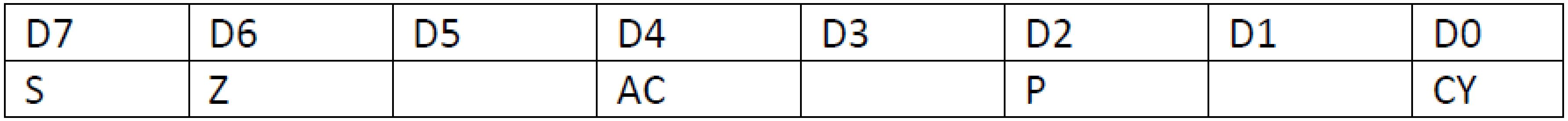
**7) Flag register**

It is an 8-bit register having five 1-bit flip-flops, which holds either 0 or 1 depending upon the result stored in the accumulator.

These are the set of 5 flip-flops –

- Sign (S)
- Zero (Z)
- Auxiliary Carry (AC)
- Parity (P)
- Carry (C)

Its bit position is shown in the following table –



| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| S | Z |  | AC |  | P |  | CY |

**a) Sign Flag (S)** – After any operation if result is negative sign flag becomes set, i.e. If result is positive sign flag becomes reset i.e. 0.

Example:

MVI A 30 (load 30H in register A)

MVI B 40 (load 40H in register B)

SUB B (A = A – B)

These set of instructions will set the sign flag to 1 as 30 – 40 is a negative number.

MVI A 40 (load 40H in register A)

MVI B 30 (load 30H in register B)

SUB B (A = A – B)

These set of instructions will reset the sign flag to 0 as 40 – 30 is a positive number.

**b) Zero Flag (Z)** – After any arithmetical or logical operation if the result is 0 (00)H, the zero flag becomes set i.e. 1, otherwise it becomes reset i.e. 0.

Example:

MVI A 10 (load 10H in register A)

SUB A (A = A – A)

These set of instructions will set the zero flag to 1 as 10H – 10H is 00H

**c) Auxiliary Carry Flag (AC)** – If intermediate carry is generated this flag is set to 1, otherwise it is reset to 0.

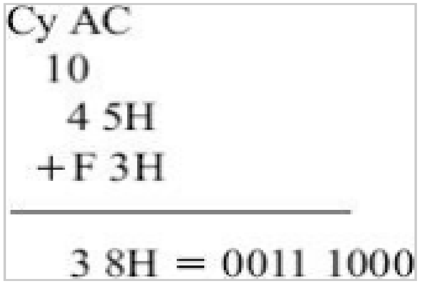
•Example:

MOV A 2B (load 2BH in register A)

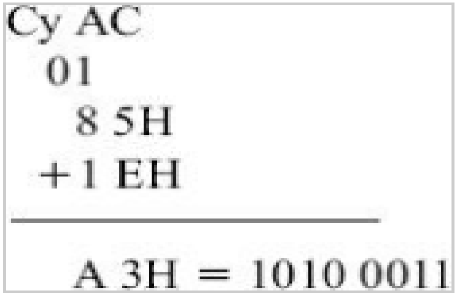
MOV B 39 (load 39H in register B)

ADD B (A = A + B)

These set of instructions will set the auxiliary carry flag to 1, as on adding 2B and 39, addition of lower order nibbles B and 9 will generate a carry.



```
Cy AC
 10
  4 5H
 +F 3H
 ________
  3 8H = 0011 1000
```



```
Cy AC
 01
  8 5H
 +1 EH
 ________
  A 3H = 1010 0011
```

d) **Parity Flag (P) –** If after any arithmetic or logical operation the result has even parity, an even number of 1 bits, the parity register becomes set i.e. 1, otherwise it becomes reset.

1-accumulator has even number of 1 bits

0-accumulator has odd parity

e) **Carry Flag (CY) –** Carry is generated when performing n bit operations and the result is more than n bits, then this flag becomes set i.e. 1, otherwise it becomes reset i.e. 0.

During subtraction (A-B), if A>B it becomes reset and if (A<B) it becomes set.

Carry flag is also called borrow flag.

**8) Instruction register and decoder**

It is an 8-bit register. When an instruction is fetched from memory then it is stored in the Instruction register. Instruction decoder decodes the information present in the Instruction register.

**9) Timing and control unit**

It provides timing and control signal to the microprocessor to perform operations. Following are the timing and control signals-

- Control Signals: READY, RD’, WR’
- Status Signals: S0, S1, IO/M’
- DMA Signals: HOLD, HLDA
- RESET Signals: RESET IN, RESET OUT

[S0 and S1 are used to observe the state, HOLD is used by peripheral device to request permission for accessing the system bus, HLDA is used to grant or deny access to its system bus, IO/M’ is to distinguish between I/O and memory port.

**10) Interrupt control**

As the name suggests it controls the interrupts during a process. When a microprocessor is executing a main program and whenever an interrupt occurs, the microprocessor shifts the control from the main program to process the incoming request. After the request is completed, the control goes back to the main program.

There are 5 interrupt signals in 8085 microprocessor: INTR, RST 7.5, RST 6.5, RST 5.5, TRAP. When microprocessor receives interrupt signals, it sends an acknowledgement (INTA) to the peripheral which is requesting for its service.

**Maskable and Non-Maskable Interrupts**

-  **Maskable Interrupts** are those which can be disabled or ignored by the microprocessor. INTR, RST 7.5, RST 6.5, RST 5.5 are maskable interrupts in 8085 microprocessor.
-  **Non-Maskable Interrupts** are those which cannot be disabled or ignored by microprocessor. TRAP is a non-maskable interrupt.

**11) Serial Input/output control**

It controls the serial data communication by using these two instructions: SID (Serial input data) and SOD (Serial output data).

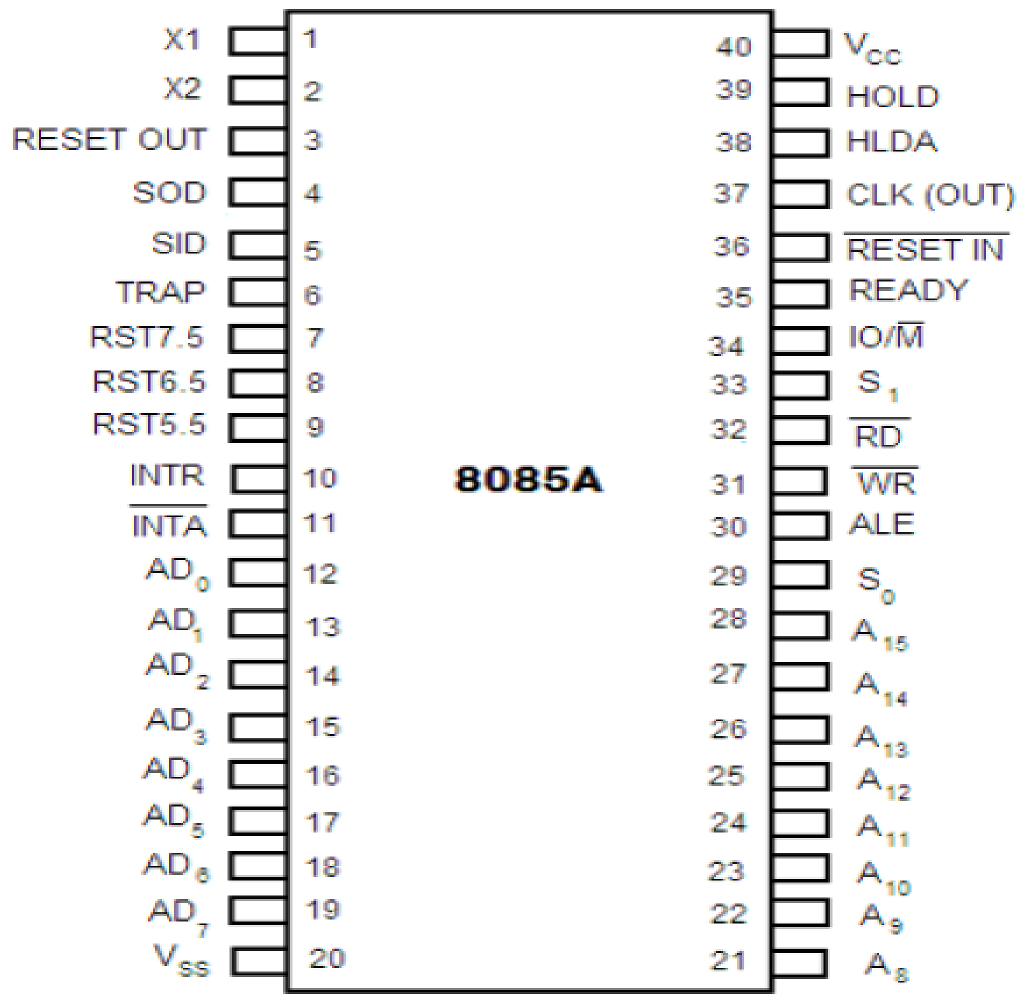
**12) Address buffer and address-data buffer**

The content stored in the stack pointer and program counter is loaded into the address buffer and address-data buffer to communicate with the CPU. The memory and I/O chips are connected to these buses; the CPU can exchange the desired data with the memory and I/O chips.

**13) Address bus and data bus**

Data bus carries the data to be stored. It is bidirectional, whereas address bus carries the location to where it should be stored and it is unidirectional. It is used to transfer the data & Address I/O devices.

# 8085 Pin Configuration



The pins of a 8085 microprocessor can be classified into seven groups:

**1) Address bus**

A15-A8, it carries the most significant 8-bits of memory/IO address.

**2) Data bus**

AD7-AD0, it carries the least significant 8-bit address and data bus.

**3) Control and status signals**

These signals are used to identify the nature of operation. There are 3 control signal and 3 status signals.

Three control signals are RD, WR & ALE.

**a)** **RD** − This signal indicates that the selected IO or memory device is to be read and is ready for accepting data available on the data bus.

**b) WR** − This signal indicates that the data on the data bus is to be written into a selected memory or IO location.

**c) ALE (Address Latch Enable)** − It is a positive going pulse generated when a new operation is started by the microprocessor. When the pulse goes high, it indicates address. When the pulse goes down it indicates data.

Three status signals are IO/M, S0 & S1.

**IO/M**

This signal is used to differentiate between IO and Memory operations, i.e. when it is high indicates IO operation and when it is low then it indicates memory operation.

**S1 & S0**

These signals are used to identify the type of current operation.

| IO/M (Active Low) | S1 | S2 | Data Bus Status (Output) |
|---|---|---|---|
| 0 | 0 | 0 | Halt |
| 0 | 0 | 1 | Memory WRITE |
| 0 | 1 | 0 | Memory READ |
| 1 | 0 | 1 | IO WRITE |
| 1 | 1 | 0 | IO READ |
| 0 | 1 | 1 | Opcode fetch |
| 1 | 1 | 1 | Interrupt acknowledge |

**4) Power supply**

There are 2 power supply signals: VCC & VSS. VCC indicates +5v power supply and VSS indicates ground signal.

**5) Clock signals**

There are 3 clock signals, i.e. X1, X2, CLK OUT.

- **X1, X2** − A crystal (RC, LC N/W) is connected at these two pins and is used to set frequency of the internal clock generator. This frequency is internally divided by 2.
- **CLK OUT** − This signal is used as the system clock for devices connected with the microprocessor.

**6) Interrupts & externally initiated signals**

- Interrupts are the signals generated by external devices to request the microprocessor to perform a task. There are 5 interrupt signals, i.e. TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR.
- **INTA** − It is an interrupt acknowledgment signal.
- **RESET IN** − This signal is used to reset the microprocessor by setting the program counter to zero.
- **RESET OUT** − This signal is used to reset all the connected devices when the microprocessor is reset.

- **READY** − This signal indicates that the device is ready to send or receive data. If READY is low, then the CPU has to wait for READY to go high.
- **HOLD** − This signal indicates that another master is requesting the use of the address and data buses.
- **HLDA (HOLD Acknowledge)** − It indicates that the CPU has received the HOLD request and it will relinquish the bus in the next clock cycle. HLDA is set to low after the HOLD signal is removed

**7) Serial I/O signals**

There are 2 serial signals, i.e. SID (Serial input data line) and SOD (Serial output data line) and these signals are used for serial communication.

## Addressing Modes in 8085

To perform any operation, we have to give the corresponding instructions to the microprocessor. In each instruction, programmer has to specify 3 things:

1. Operation to be performed.
2. Address of source of data.
3. Address of destination of result.

The method by which the address of source of data or the address of destination of result is given in the instruction is called Addressing Modes. The term addressing mode refers to the way in which the operand of the instruction is specified.

Intel 8085 uses the following addressing modes:

1) Direct Addressing Mode

2) Register Addressing Mode

3) Register Indirect Addressing Mode

4) Immediate Addressing Mode

5) Implicit Addressing Mode

## Direct Addressing Mode

In direct addressing mode, the data to be operated is available inside a memory location and that memory location is directly specified as an operand. The operand is directly available in the instruction itself.

Example:

LDA 2050 (load the contents of memory location into accumulator A)

**Register Addressing Mode**

In register addressing mode, the data to be operated is available inside the register(s) and register(s) is(are) operands. Therefore, the operation is performed within various registers of the microprocessor.

Examples:

MOV A, B (move the contents of register B to register A)

ADD B (add contents of registers A and B and store the result in register A)

INR A (increment the contents of register A by one)

**Register Indirect Addressing Mode**

In register indirect addressing mode, the data to be operated is available inside a memory location and that memory location is indirectly specified by a register pair.

**Example:**

MOV A, M (move the contents of the memory location pointed by the H-L pair to the accumulator)

## Immediate Addressing Mode

In immediate addressing mode the source operand is always data. If the data is 8-bit, then the instruction will be of 2 bytes, if the data is of 16-bit then the instruction will be of 3 bytes.

**Example:**

MVI B,45 (move the data 45H immediately to register B)

**Implied/Implicit Addressing Mode**

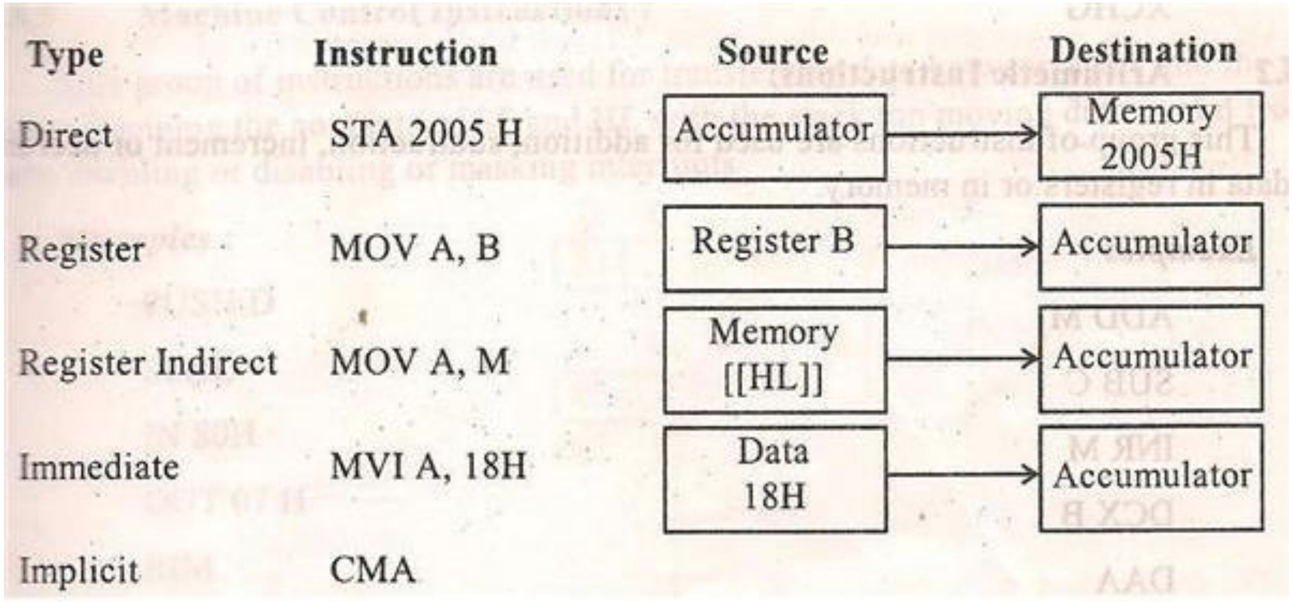
In implied/implicit addressing mode the operand is hidden and the data to be operated is available in the instruction itself.

**Examples:**

RRC (rotate accumulator A right by one bit)

RLC (rotate accumulator A left by one bit)

# Addressing Modes of 8085



| Type | Instruction | Source | | Destination |
|---|---|---|---|---|
| Direct | STA 2005 H | Accumulator | → | Memory 2005H |
| Register | MOV A, B | Register B | → | Accumulator |
| Register Indirect | MOV A, M | Memory [[HL]] | → | Accumulator |
| Immediate | MVI A, 18H | Data 18H | → | Accumulator |
| Implicit | CMA | | | |

## Instruction cycle, Machine Cycle & T State in 8085 microprocessor

Time required to execute and fetch an entire instruction is **called instruction cycle.**

It consists:

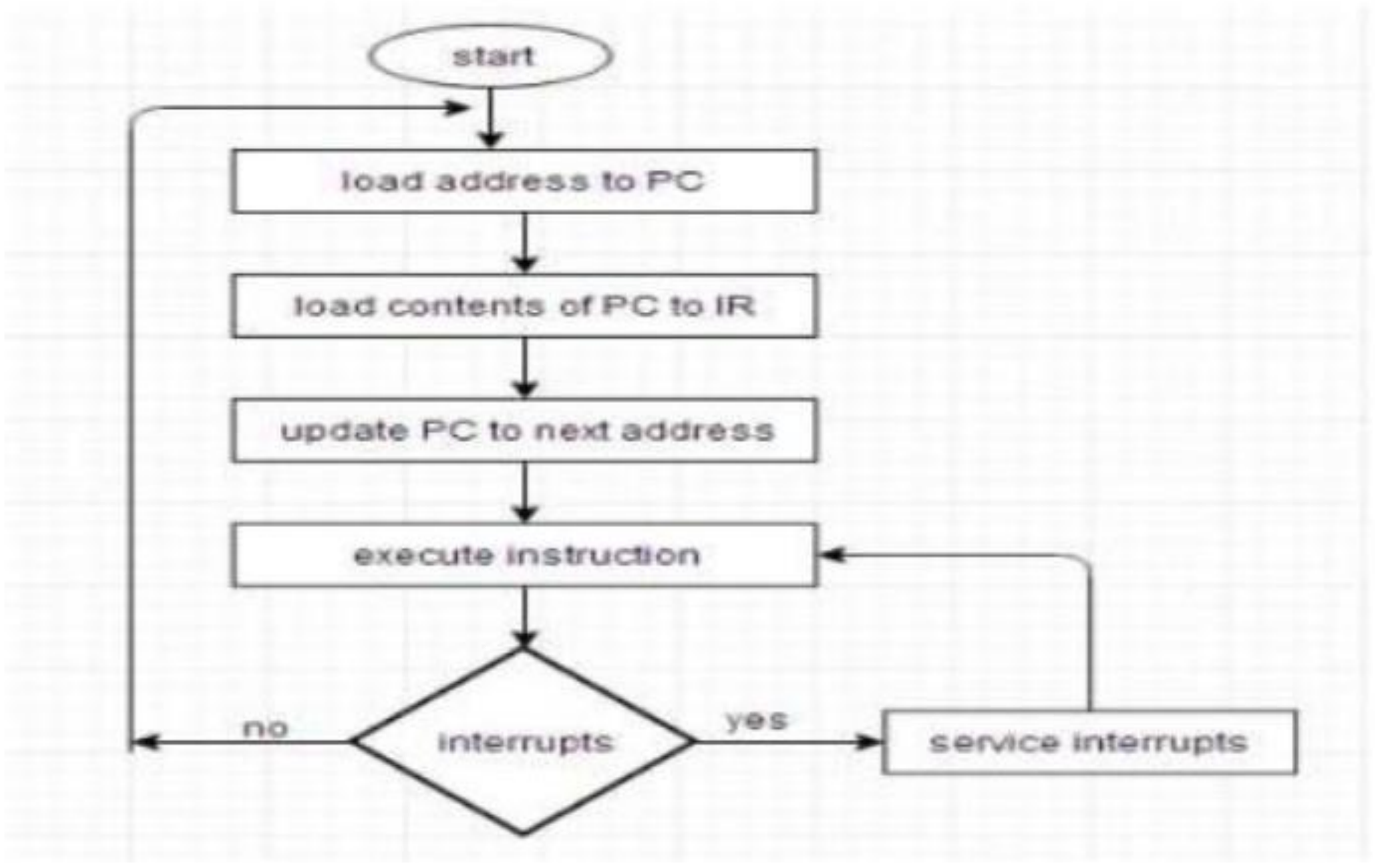
a) **Fetch cycle** – The next instruction is fetched by the address stored in program counter (PC) and then stored in the instruction register.

b) **Decode instruction** – Decoder interprets the encoded instruction from instruction register.

c) **Reading effective address** – The address given in instruction is read from main memory and required data is fetched. The effective address depends on direct addressing mode or indirect addressing mode.

d) **Execution cycle** – consists memory read (MR), memory write (MW), input output read (IOR) and input output write (IOW).

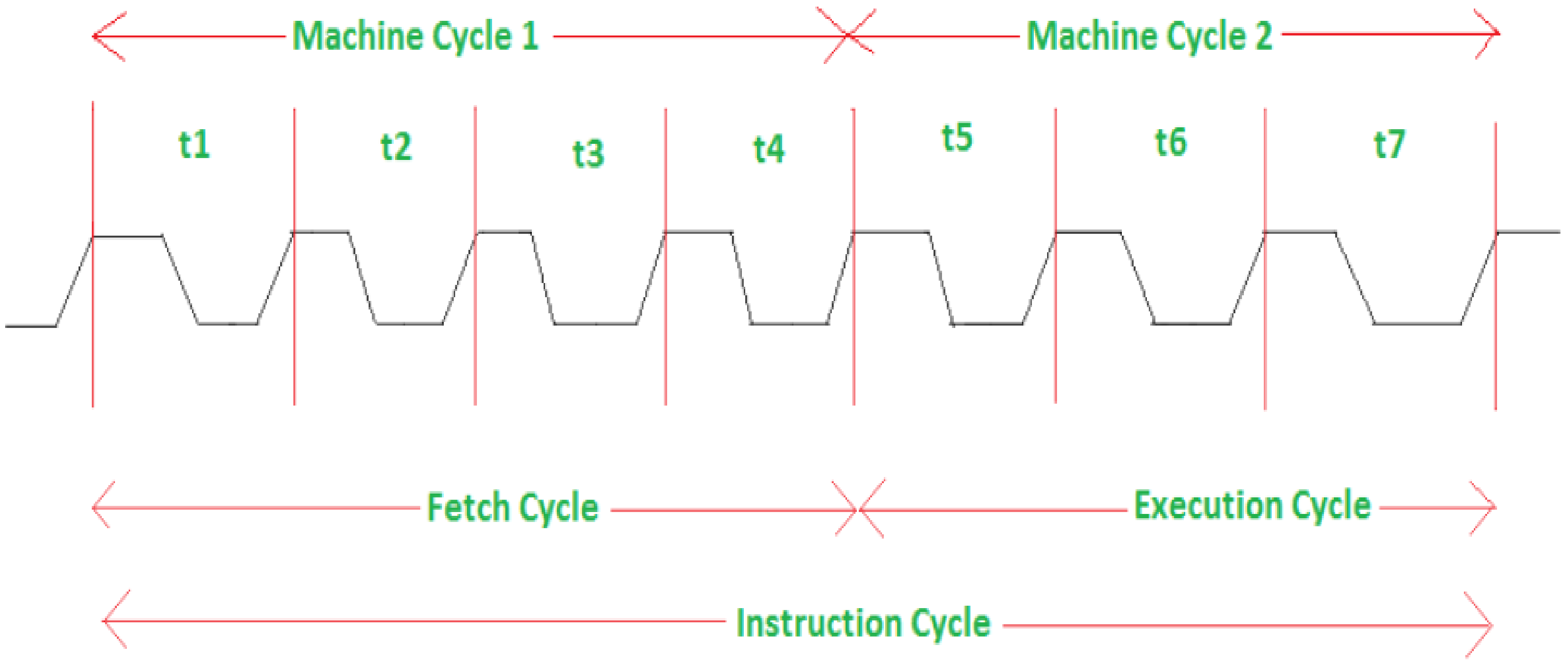
## General Flowchart of Instruction Cycle



Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction. This process continues indefinitely unless a HALT instruction is encountered.

The time required by the microprocessor to complete an operation of accessing memory or input/output devices is called machine cycle. Following are the different types of machine cycle:

- Opcode fetch, which takes 4 t-states
- Memory Read, which takes 3 t-states
- Memory Write, which takes 3 t-states
- I/O Read, which takes 3 t-states
- I/O Write, which takes 3 t-states


Machine Cycle 1
Machine Cycle 2
t1
t2
t3
t4
t5
t6
t7
Fetch Cycle
Execution Cycle
Instruction Cycle

- The time required by the microprocessor to complete an operation of accessing memory or input/output devices is called **machine cycle**.
- One time period of frequency of microprocessor is called **t-state.** A t-state is measured from the falling edge of one clock pulse to the falling edge of the next clock pulse.
- Fetch cycle takes four t-states and execution cycle takes three t-states.

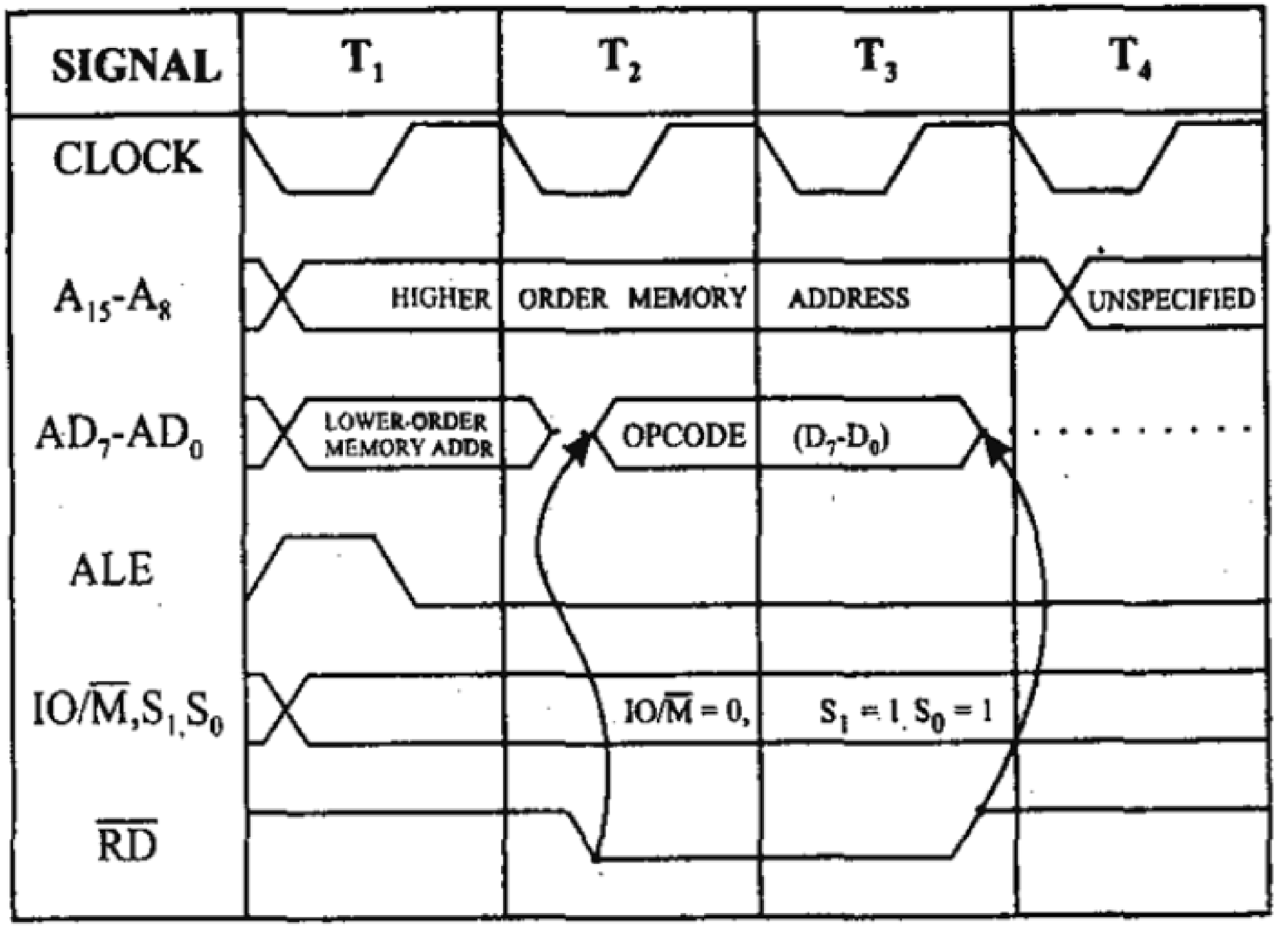
## Fetch & Execute Operation: Timing Diagram

- The graphical representation of status of various signals involved during a machine cycle with respect to time is called timing diagram.
- This gives basic idea of what is happening in the system when the instruction is getting fetched and executed, at what instant which signal is getting activated.
- The signals involved during machine cycle are CLK, A15 – A8, AD7 – AD0, IO/M(bar), RD(bar), WR(bar) and S1,S0.

$S_0$.

| IO/M(bar) | $S_1$ | $S_0$ | Operation |
|---|---|---|---|
| 0 | 0 | 0 | Halt |
| 0 | 0 | 1 | Memory write |
| 0 | 1 | 0 | Memory read |
| 0 | 1 | 1 | Op-code fetch |
| 1 | 0 | 1 | IO write |
| 1 | 1 | 0 | IO read |
| 1 | 1 | 1 | Interrupt acknowledge |

## Timing diagram for op-code fetch cycle



The op-code fetch timing diagram can be explained as below:

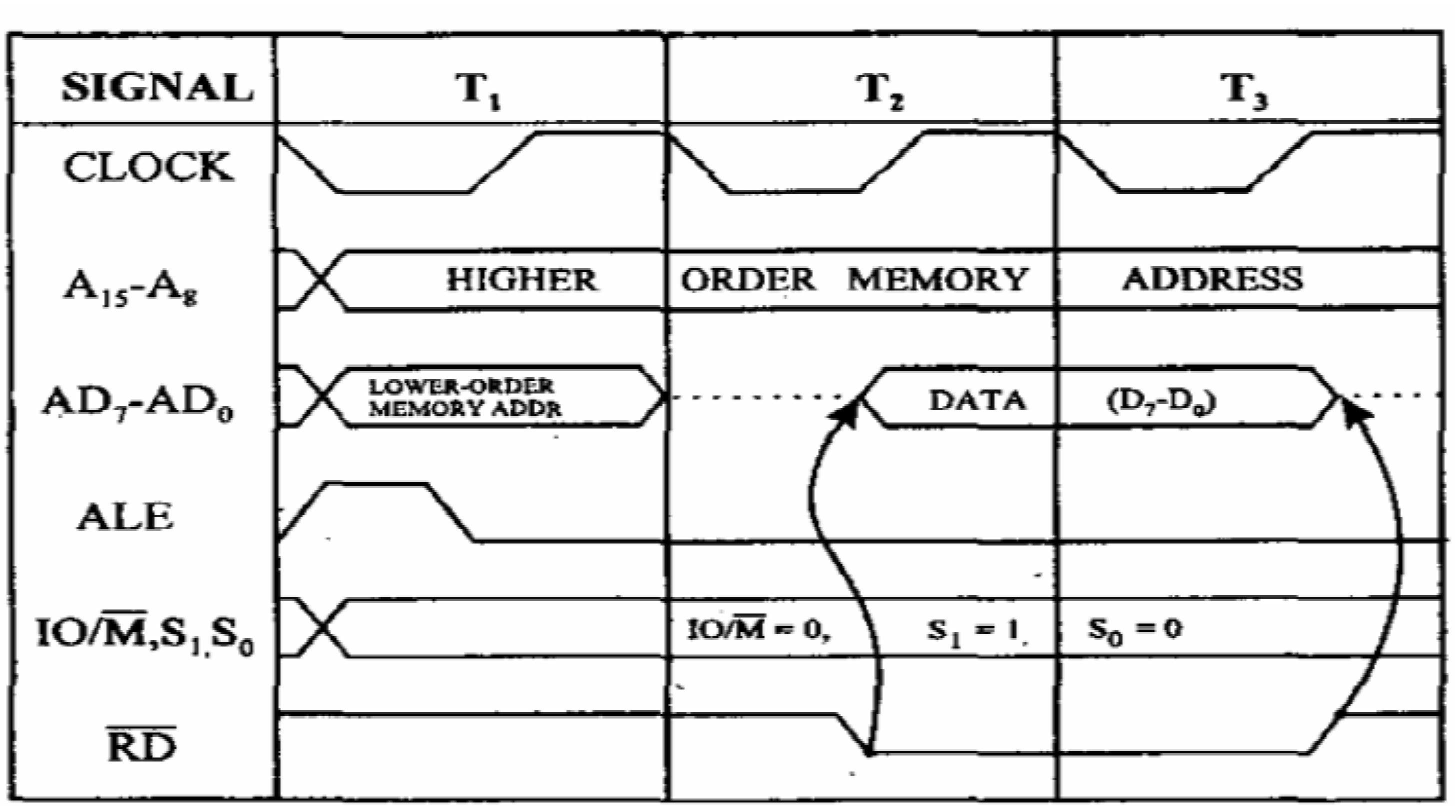
i) The MP places the 16-bit memory address from the program counter on address bus. At time period T1, the higher order memory address is placed on the address lines A15 – A8. When ALE is high, the lower address is placed on the bus AD7 – AD0. The status signal IO/M(bar) goes low indicating the memory operation and two status signals S1 = 1, S0 = 1 to indicate op-code fetch operation.

ii) At time period T2, the MP sends RD(bar) control line to enable the memory read. When memory is enabled with RD(bar) signal, the op-code value from the addressed memory location is placed on the data bus with ALE low.

iii) The op-code value is reached at processor register during T3 time period. When data (op-code value) is arrived, the RD(bar) signal goes high. It causes the bus to go into high impedance state.

iv) The op-code byte is placed in instruction decoder of MP and the op-code is decoded and executed. This happens during time period T4. In this time, the first, 3 T-states are used for fetching the opcode from memory and the remaining T-states are used for internal operations by the processor.

Timing diagram for memory read cycle



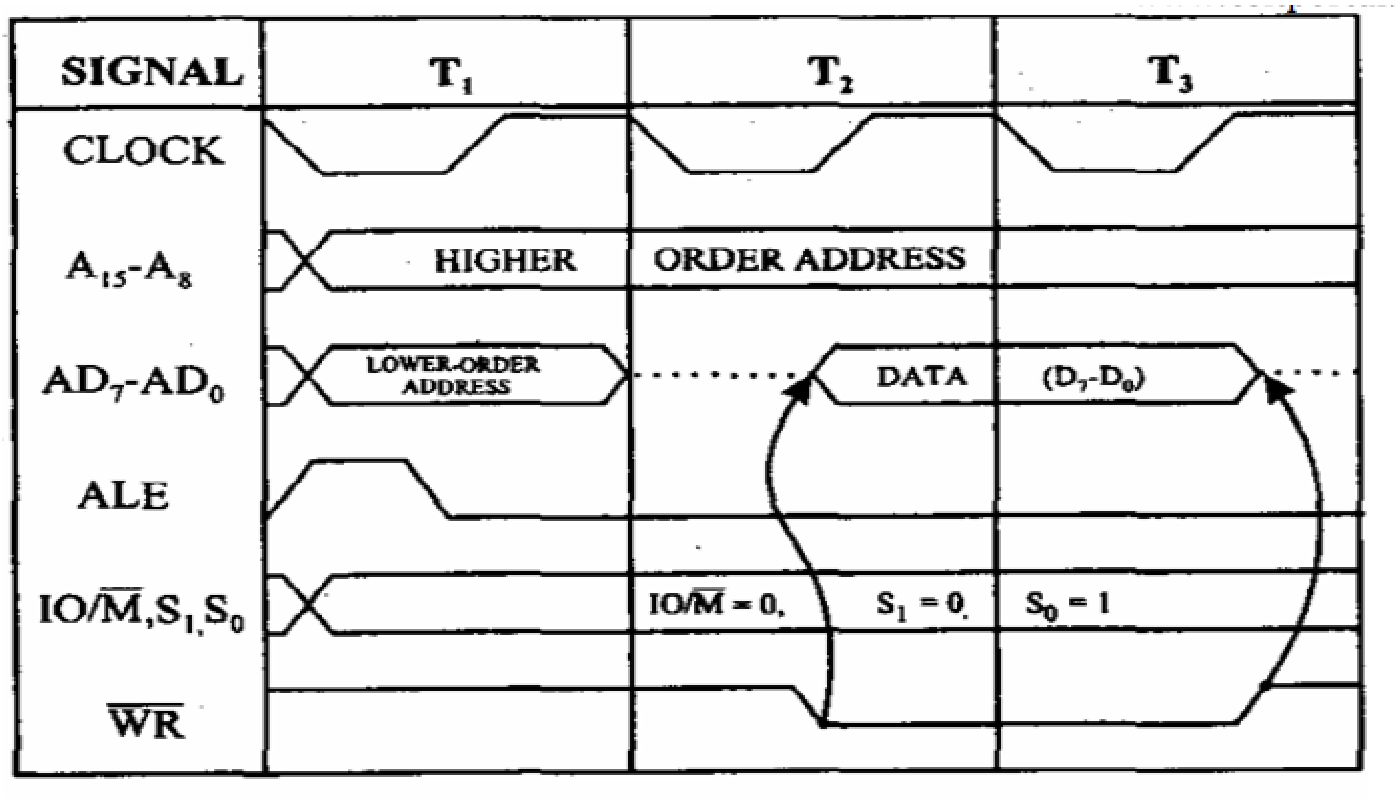
The memory read timing diagram can be explained as below:

i) The MP places the 16-bit memory address from the program counter on address bus. At time period T1, the higher order memory address is placed on the address lines A15 – A8. When ALE is high, the lower address is placed on the bus AD7 – AD0. The status signal IO/M(bar) goes low indicating the memory operation and two status signals S1 = 1, S0 = 0 to indicate memory read operation.

ii) At time period T2, the MP sends RD(bar) control line to enable the memory read. When memory is enabled with RD(bar) signal, the data from the addressed memory location is placed on the data bus with ALE low.

iii) The data is reached at processor register during T3 state. When data is arrived, the RD(bar) signal goes high. It causes the bus to go into high impedance state.

## Timing diagram for memory write cycle



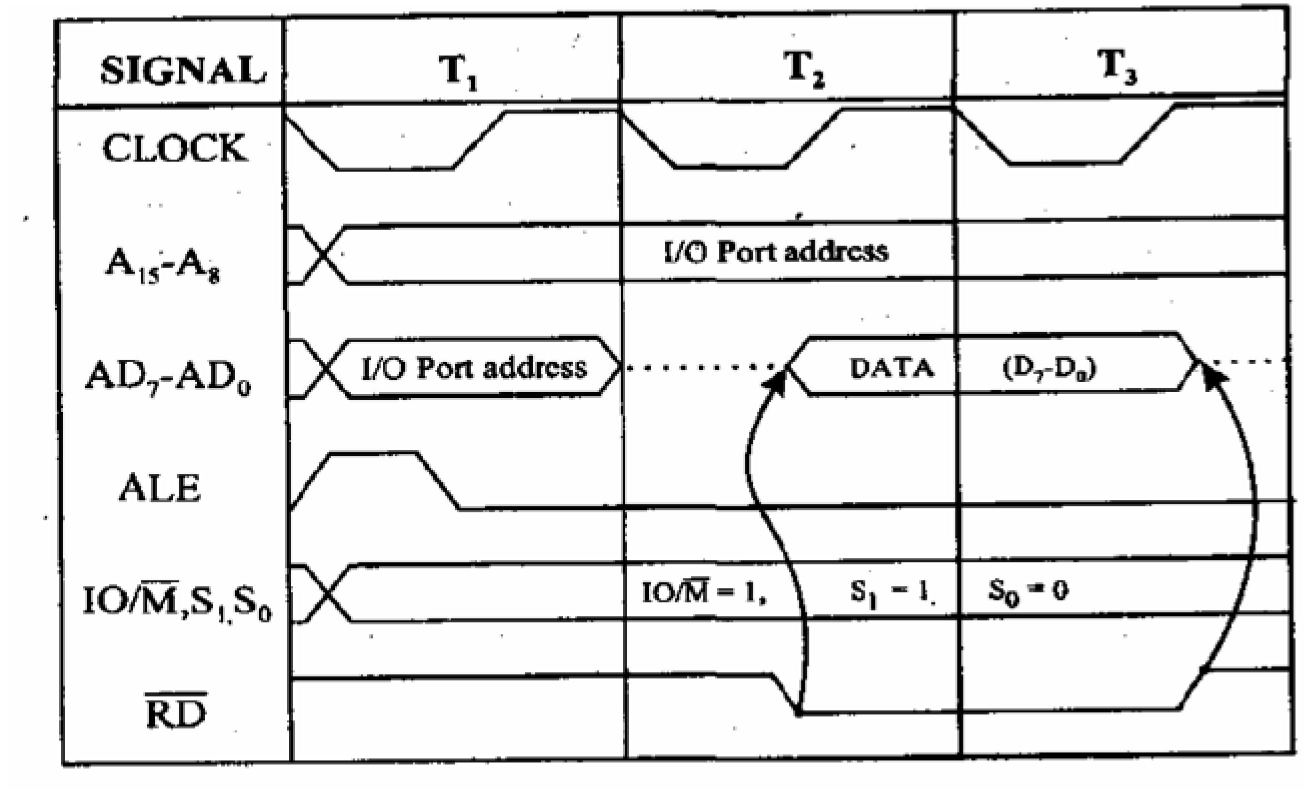
The memory write timing diagram can be explained as below:

i) The MP places the 16-bit memory address from the program counter on address bus. At time period T1, the higher order memory address is placed on the address lines A15 – A8. When ALE is high, the lower address is placed on the bus AD7 – AD0. The status signal IO/M(bar) goes low indicating the memory operation and two status signals S1 = 0, S0 = 1 to indicate memory write operation.

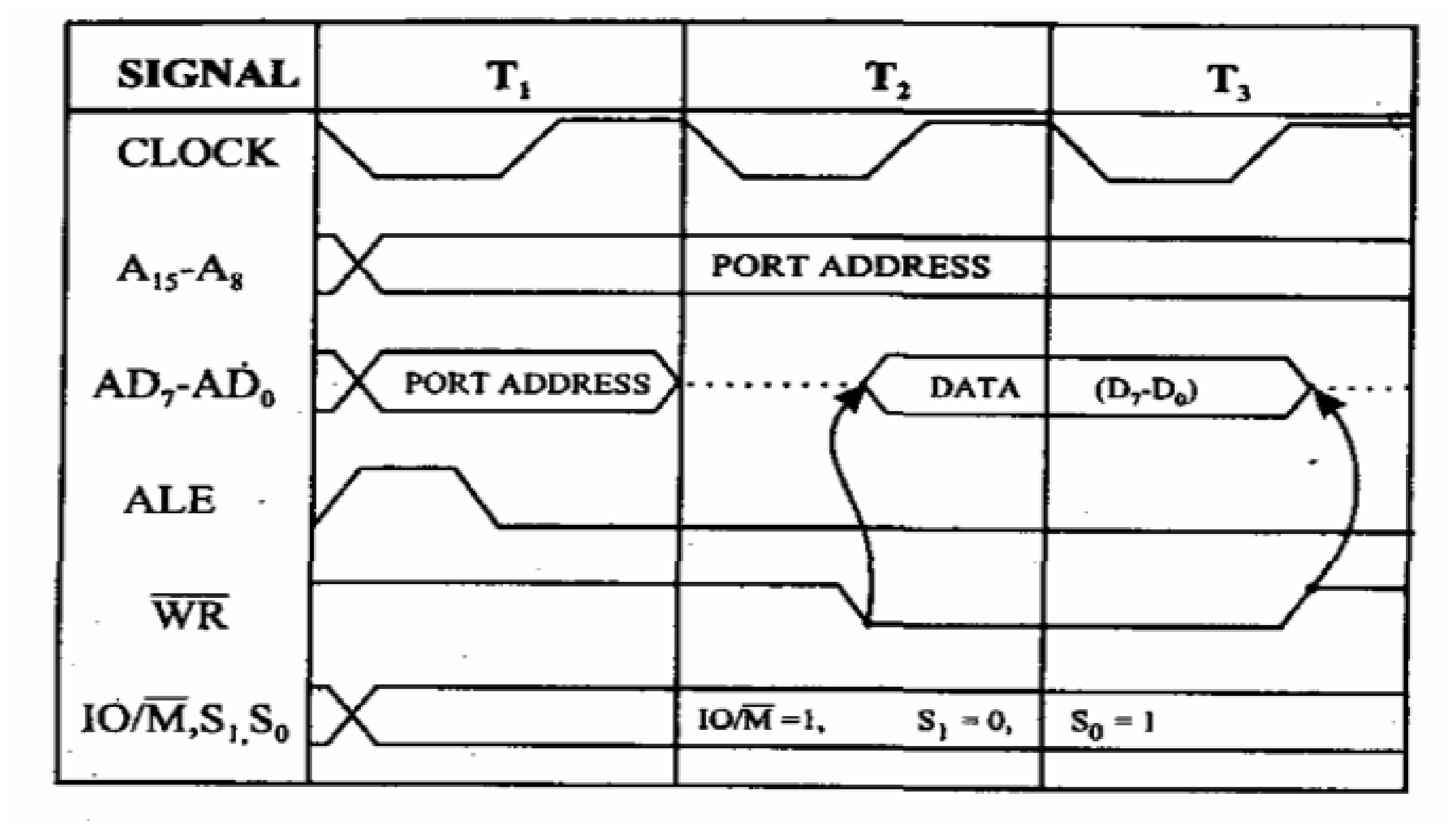
ii) At time period T2, the MP sends WR(bar) control line to enable the memory write. When memory is enabled with WR(bar) signal, the data from the processor is placed on the addressed location with ALE low.

iii) The data is reached at memory location during T3 state. When data is reached, the WR(bar) signal goes high. It causes the bus to go into high impedance state.

# Timing diagram for IO read cycle



# Timing diagram for IO write cycle



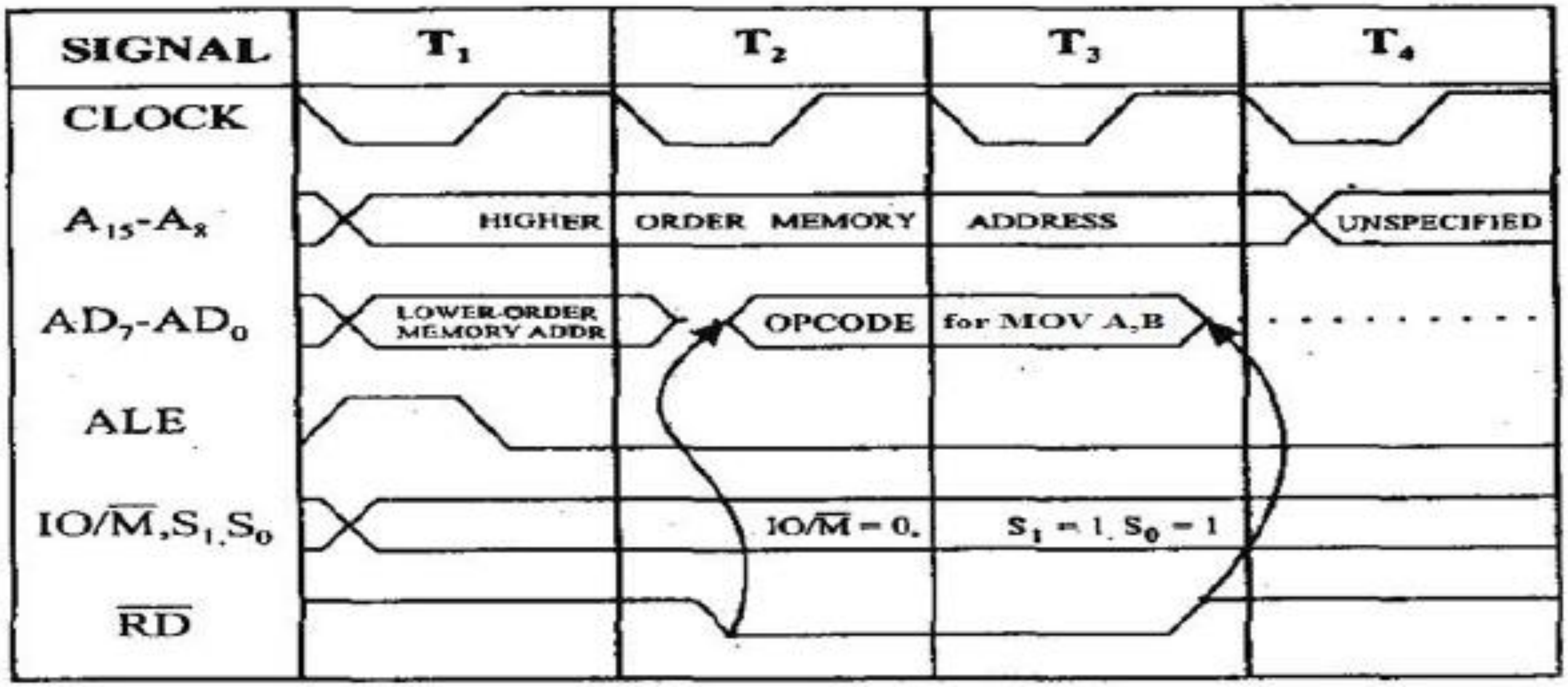
## Timing Diagram of MOV Eg: MOV A,B

- The instruction MOV A ,B is of 1 byte; therefore the complete instruction will be stored in a single memory address.
- For example: 2000: **MOV A, B**

**In Opcode fetch ( t1-t4 T states ):**

- 00 – lower bit of address where opcode is stored, i.e., 00
- 20 – higher bit of address where opcode is stored, i.e., 20.
- ALE – provides signal for multiplexed address and data bus. Only in t1 it used as address bus to fetch lower bit of address otherwise it will be used as data bus.
- RD (low active) – signal is 1 in t1 & t4 as no data is read by microprocessor. Signal is 0 in t2 & t3 because here the data is read by microprocessor.
- WR (low active) – signal is 1 throughout, no data is written by microprocessor.
- IO/M (low active) – signal is 1 in throughout because the operation is performing on memory.
- S0 and S1 – both are 1 in case of opcode fetching.

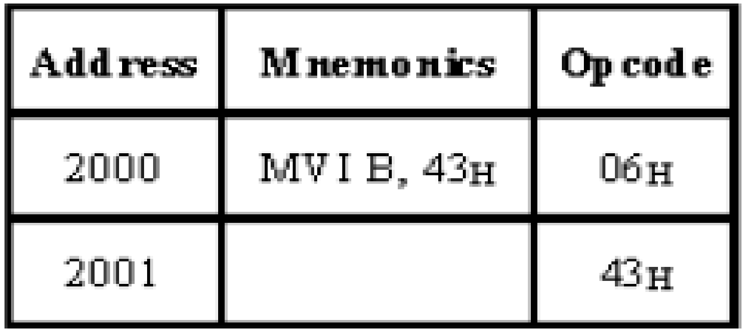
# Timing Diagram of MOV
Eg: MOV A,B



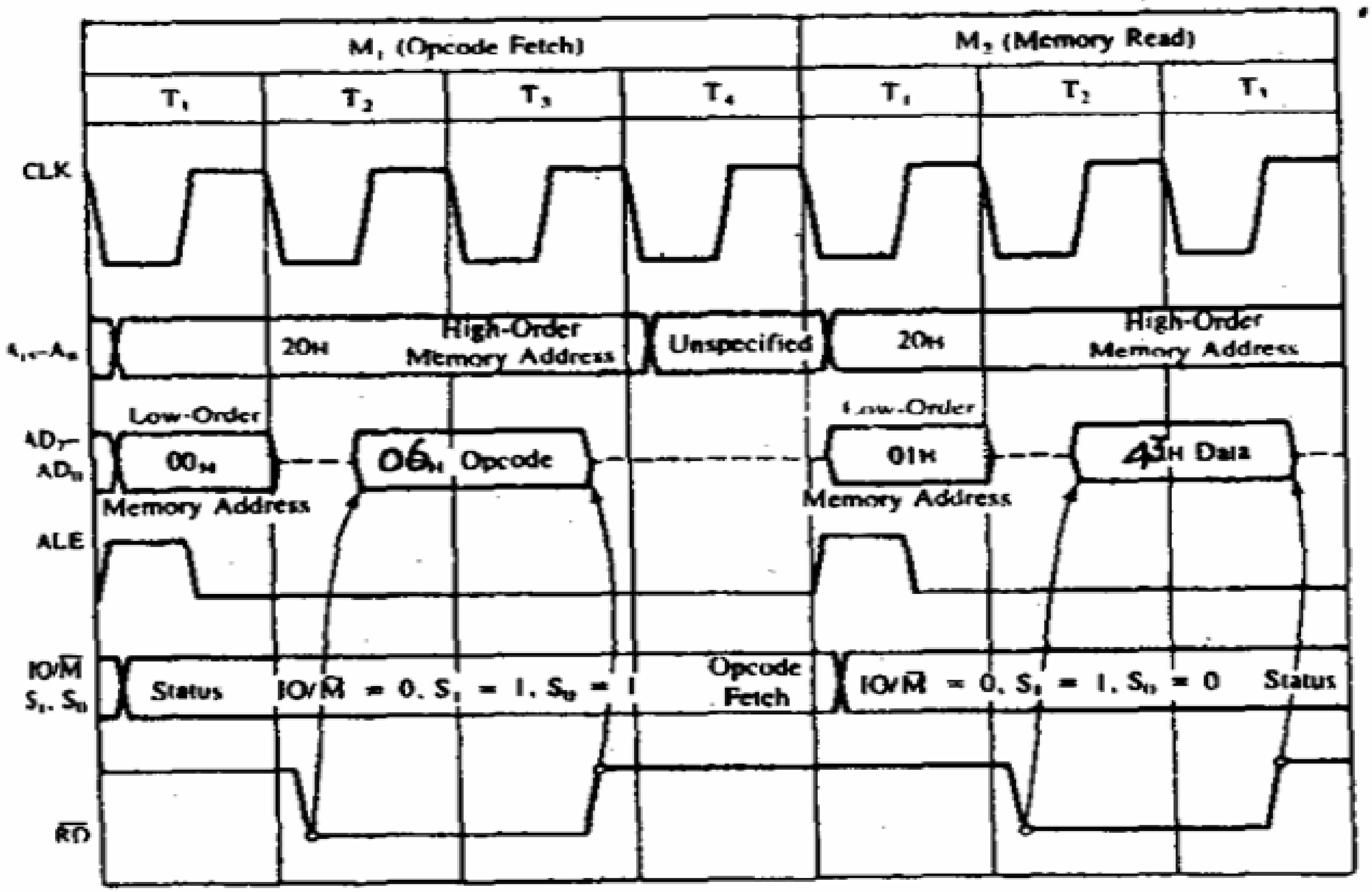
## Timing Diagram of MVI Eg: MVI B, 43H

Timing diagram for MVI B, 43H

- Fetching the Op-code 06H from the memory 2000H. (OF machine cycle)
- Read (move) the data 43H from memory 2001H. (memory read)

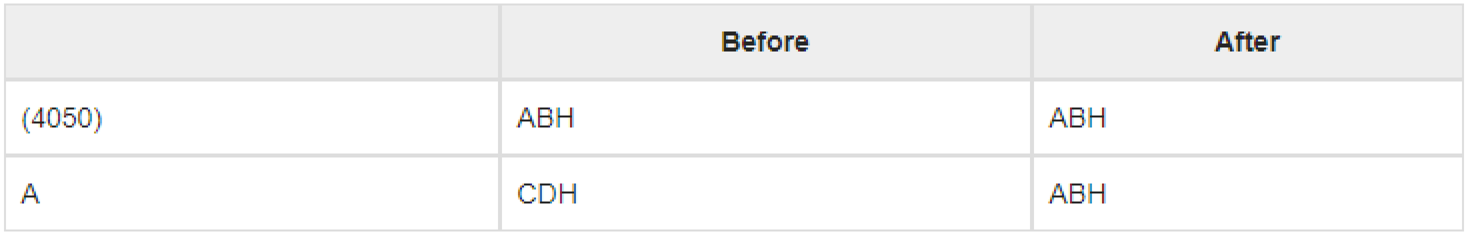


| Address | Mnemonics | Op code |
|---|---|---|
| 2000 | MVI B, 43H | 06H |
| 2001 | | 43H |

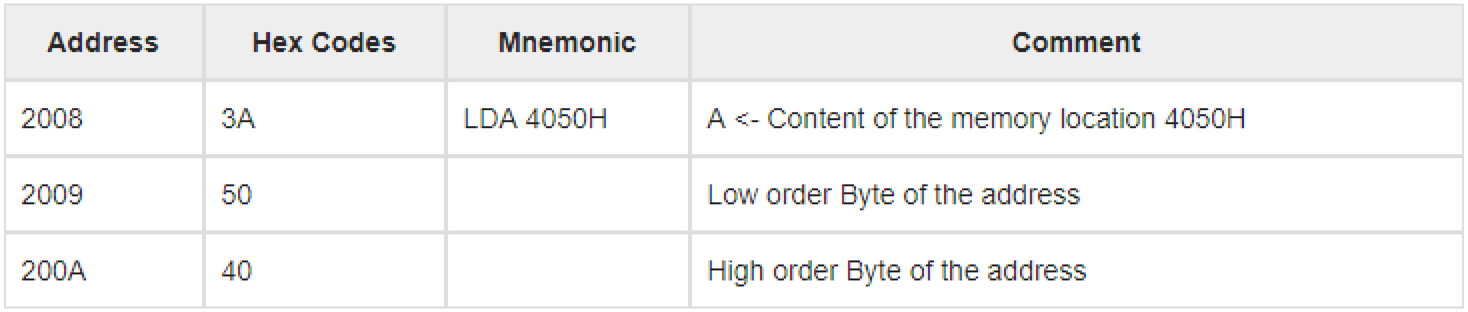

M₁ (Opcode Fetch)
M₂ (Memory Read)
T₁
T₂
T₃
T₄
T₁
T₂
T₃
CLK
A₁₅–A₈
20H
High-Order Memory Address
Unspecified
20H
High-Order Memory Address
AD₇–AD₀
Low-Order
00H
Memory Address
06H Opcode
Low-Order
01H
Memory Address
43H Data
ALE
IO/M̄ S₁, S₀
Status
IO/M̄ = 0, S₁ = 1, S₀ = 1
Opcode Fetch
IO/M̄ = 0, S₁ = 1, S₀ = 0
Status
RD̄

# Timing diagram for LDA 4050H

- Let us consider LDA 4050H as an example instruction of this type. It is a 3-Byte instruction. The initial content of memory address 4050H is ABH. Initially Accumulator content is CDH. As after execution A will be initialized with value ABH. Memory location 4050H will still remain with the content ABH. The results of execution of this instruction is as below –



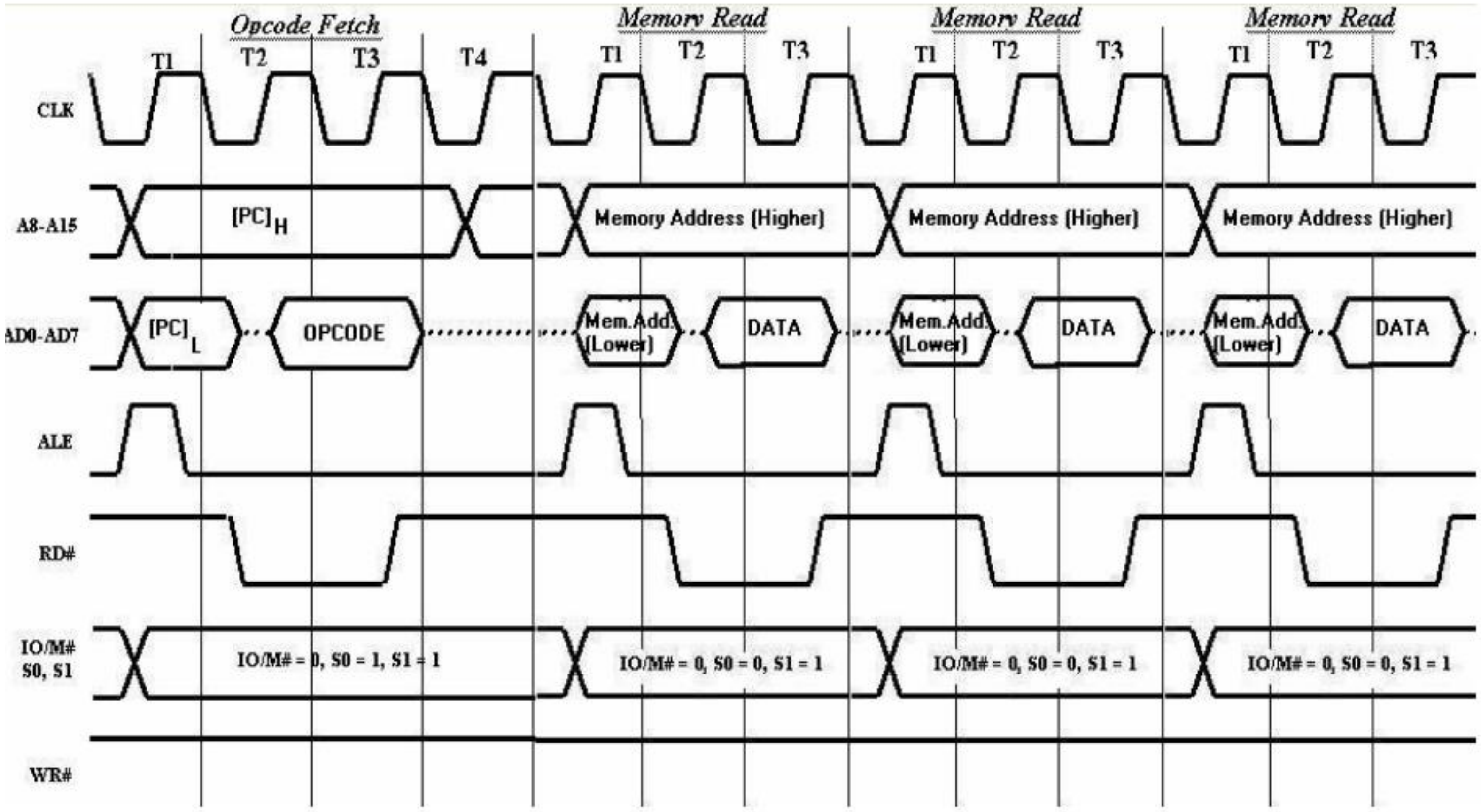
| | Before | After |
|---|---|---|
| (4050) | ABH | ABH |
| A | CDH | ABH |



| Address | Hex Codes | Mnemonic | Comment |
|---|---|---|---|
| 2008 | 3A | LDA 4050H | A <- Content of the memory location 4050H |
| 2009 | 50 | | Low order Byte of the address |
| 200A | 40 | | High order Byte of the address |

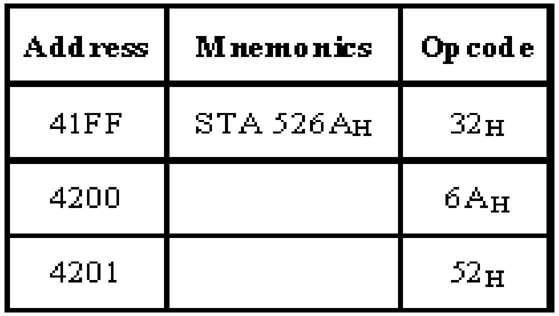
| Machine Cycle | T-states | Operation Performed |
|---|---|---|
| Opcode Fetch | 4 | Read Opcode from Memory to instruction register (3A) |
| Memory Read | 3 | Read Lower order memory address (50) |
| Memory Read | 3 | Read higher order memory address (40) |
| Memory Read | 3 | Read data from memory, using the address values read from previous 2 machine read cycle i.e from 4050 to accumulator. |

LDA 16bit Addr



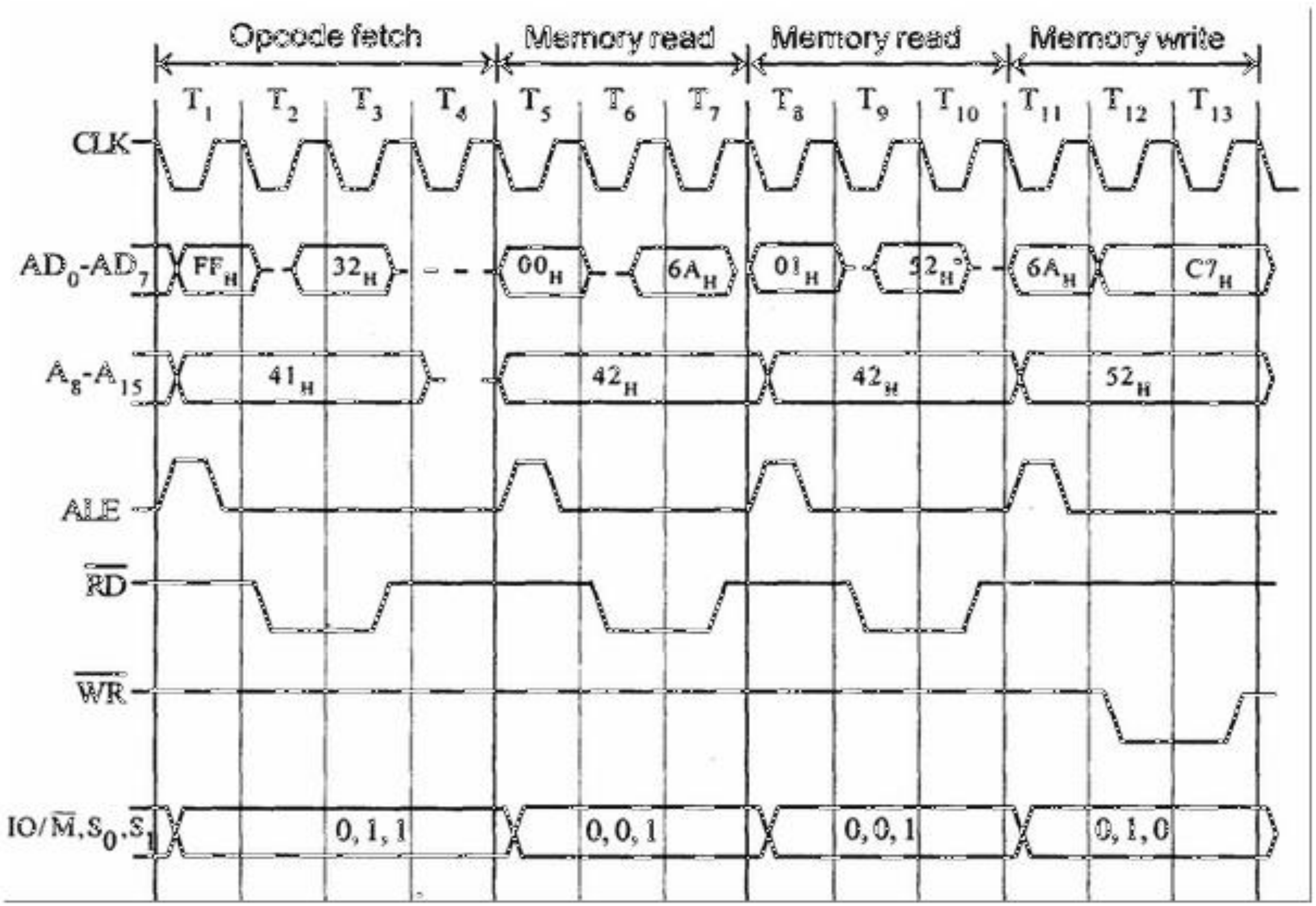
# Timing diagram for STA 526AH.

- STA means Store Accumulator -The contents of the accumulator is stored in the specified address(526A).
- The opcode of the STA instruction is said to be 32H. It is fetched from the memory 41FFH(see fig). - OF machine cycle
- Then the lower order memory address is read(6A). - Memory Read Machine Cycle
- Read the higher order memory address (52).- Memory Read Machine Cycle
- The combination of both the addresses are considered and the content from accumulator is written in 526A. - Memory Write Machine Cycle
- Assume the memory address for the instruction and let the content of accumulator is C7H. So, C7H from accumulator is now stored in 526A.

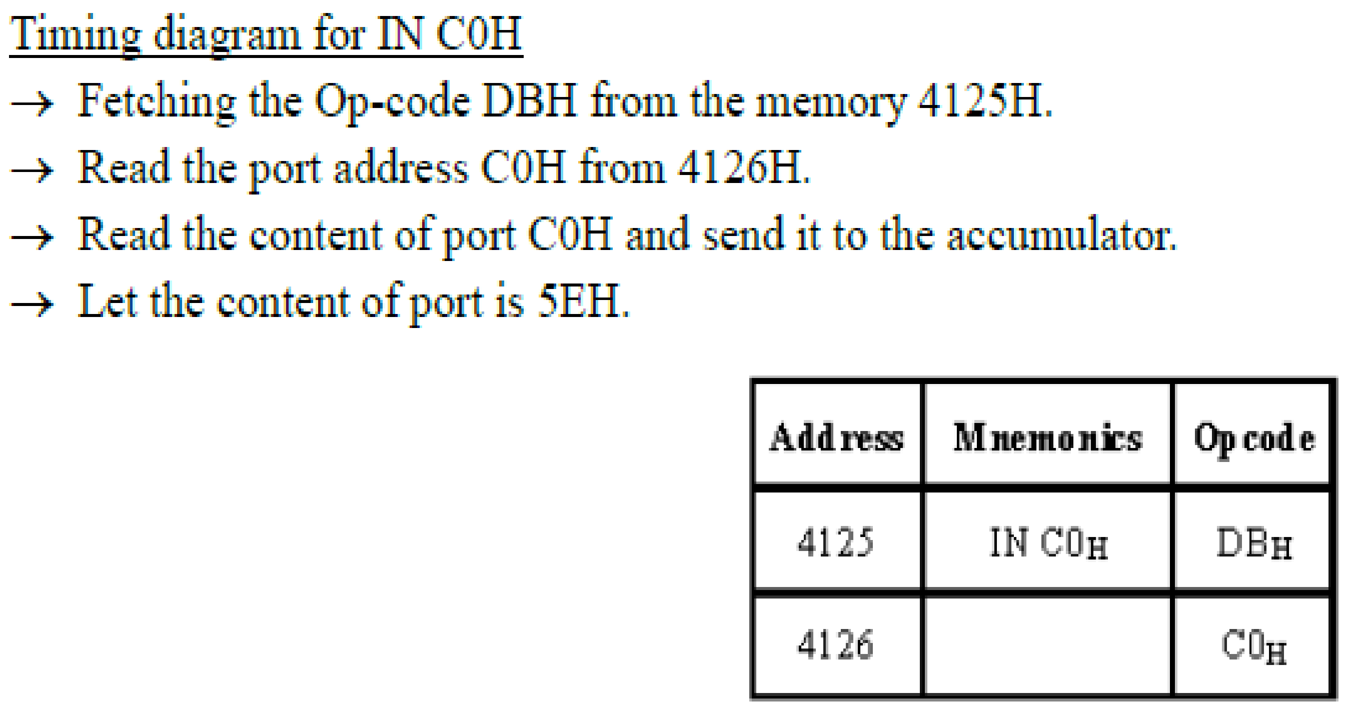


| Address | Mnemonics | Opcode |
|---|---|---|
| 41FF | STA 526$A_H$ | $32_H$ |
| 4200 | | $6A_H$ |
| 4201 | | $52_H$ |

| Machine Cycle | T-states | Operation Performed |
|---|---|---|
| Opcode Fetch | 4 | Read Opcode from Memory to instruction register (32) |
| Memory Read | 3 | Read Lower order memory address (6A) |
| Memory Read | 3 | Read higher order memory address (52) |
| Memory Write | 3 | Write data present in accumulator to memory, in the address values read from previous 2 machine read cycle i.e. to memory location 526AH |


Opcode fetch
Memory read
Memory read
Memory write
T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 T12 T13
CLK
AD0-AD7
FFH
32H
00H
6AH
01H
52H
6AH
C7H
A8-A15
41H
42H
42H
52H
ALE
RD
WR
IO/M, S0, S1
0, 1, 1
0, 0, 1
0, 0, 1
0, 1, 0

# Timing Diagram of IN



Timing diagram for IN C0H

→ Fetching the Op-code DBH from the memory 4125H.

→ Read the port address C0H from 4126H.

→ Read the content of port C0H and send it to the accumulator.

→ Let the content of port is 5EH.

| Address | Mnemonics | Opcode |
|---|---|---|
| 4125 | IN $C0_H$ | $DB_H$ |
| 4126 |  | $C0_H$ |

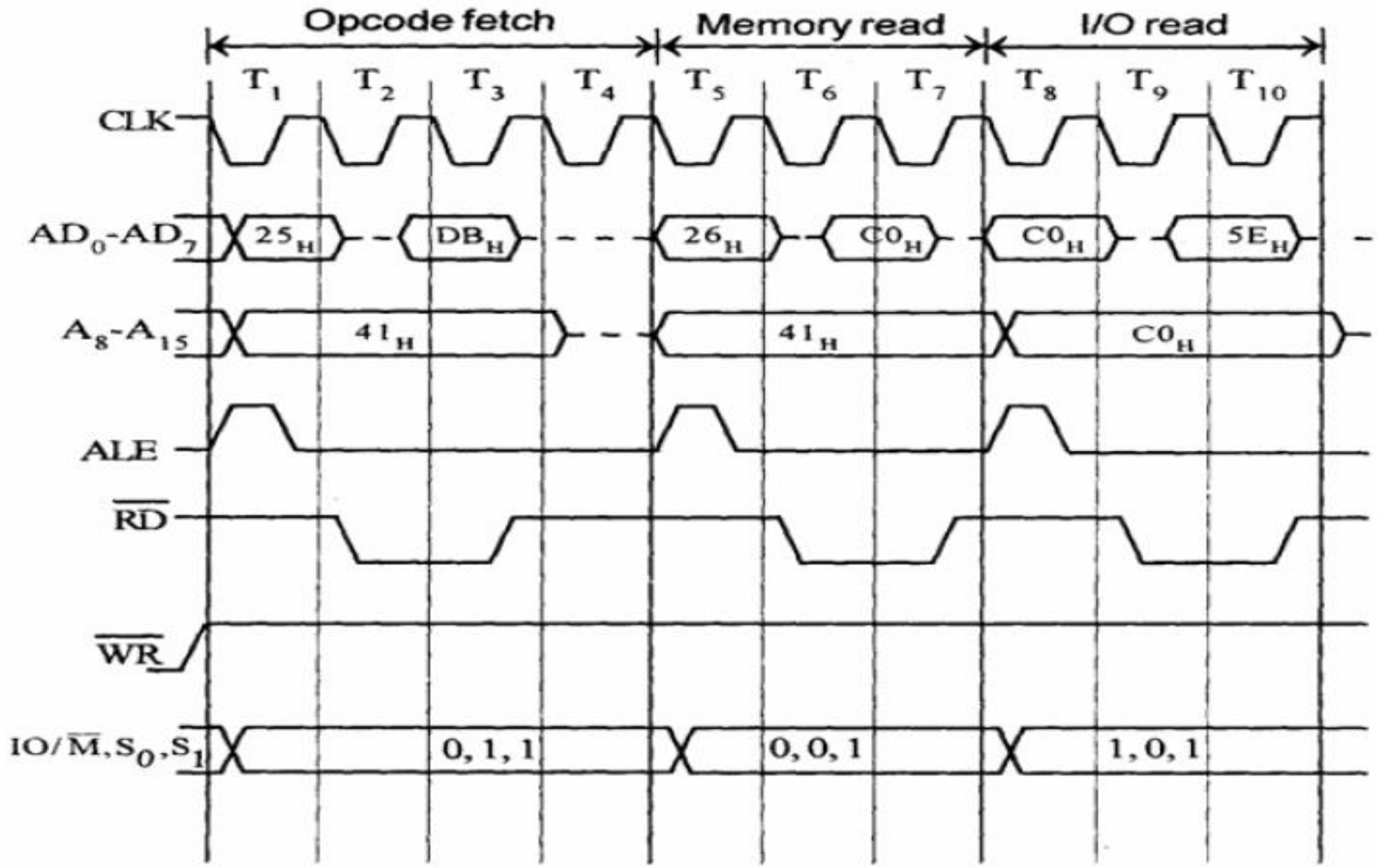


Fig: Timing Diagram for IN C0H

## Timing Diagram of OUT

- Instruction Format: OUT 8 bit Port Address
- Example: OUT 02H

| Address | Data |
|---|---|
| 8000 | Opcode of OUT (D3) |
| 8001 | 02 (Port Address) |

| Machine Cycle | T-states | Operation Performed |
|---|---|---|
| Opcode Fetch | 4 | Read Opcode from Memory to instruction register (D3) |
| Memory Read | 3 | Read Lower order memory address (02) |
| I/O Write | 3 | Write data to previously read port address i.e. to port 02 |

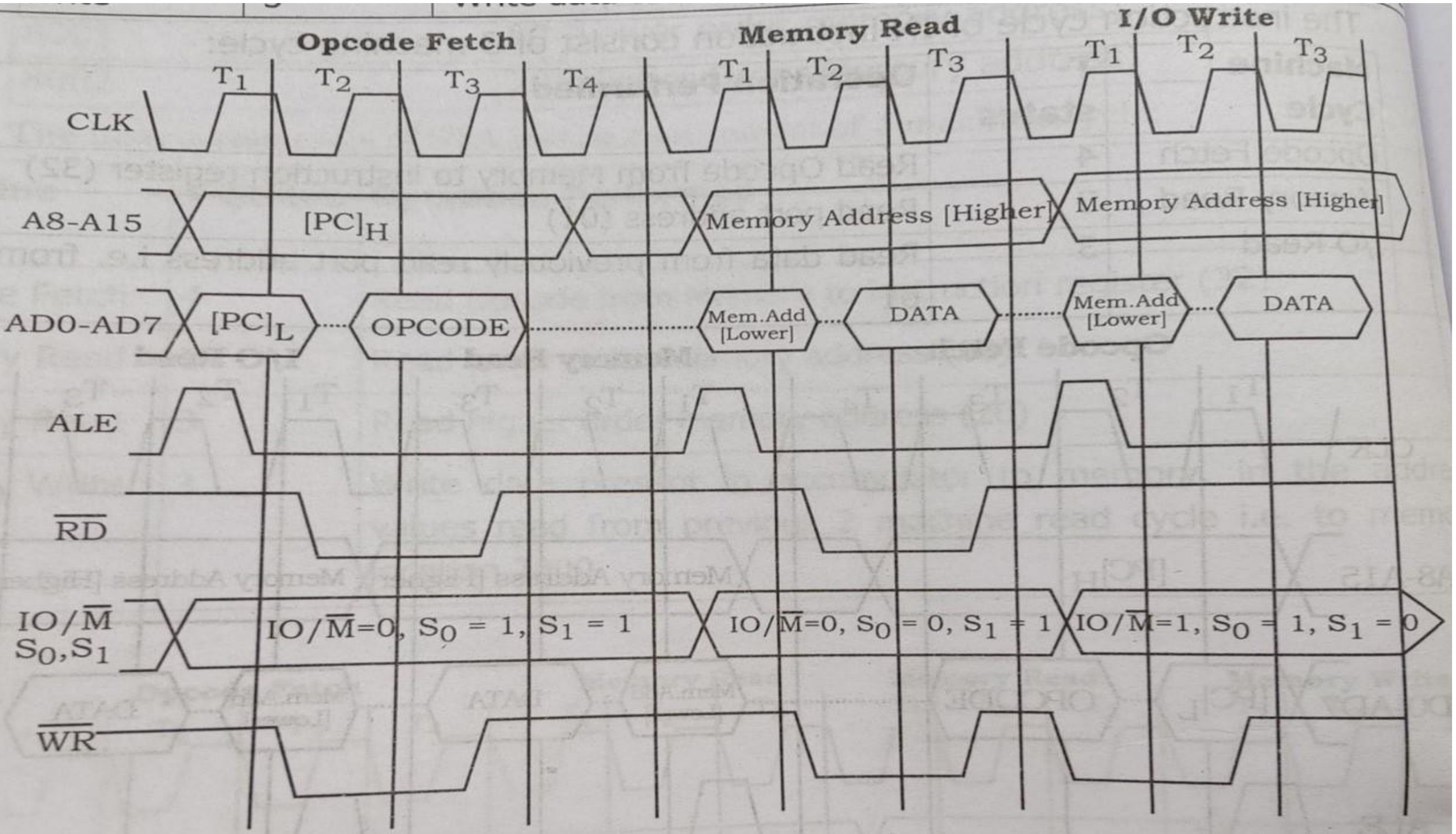
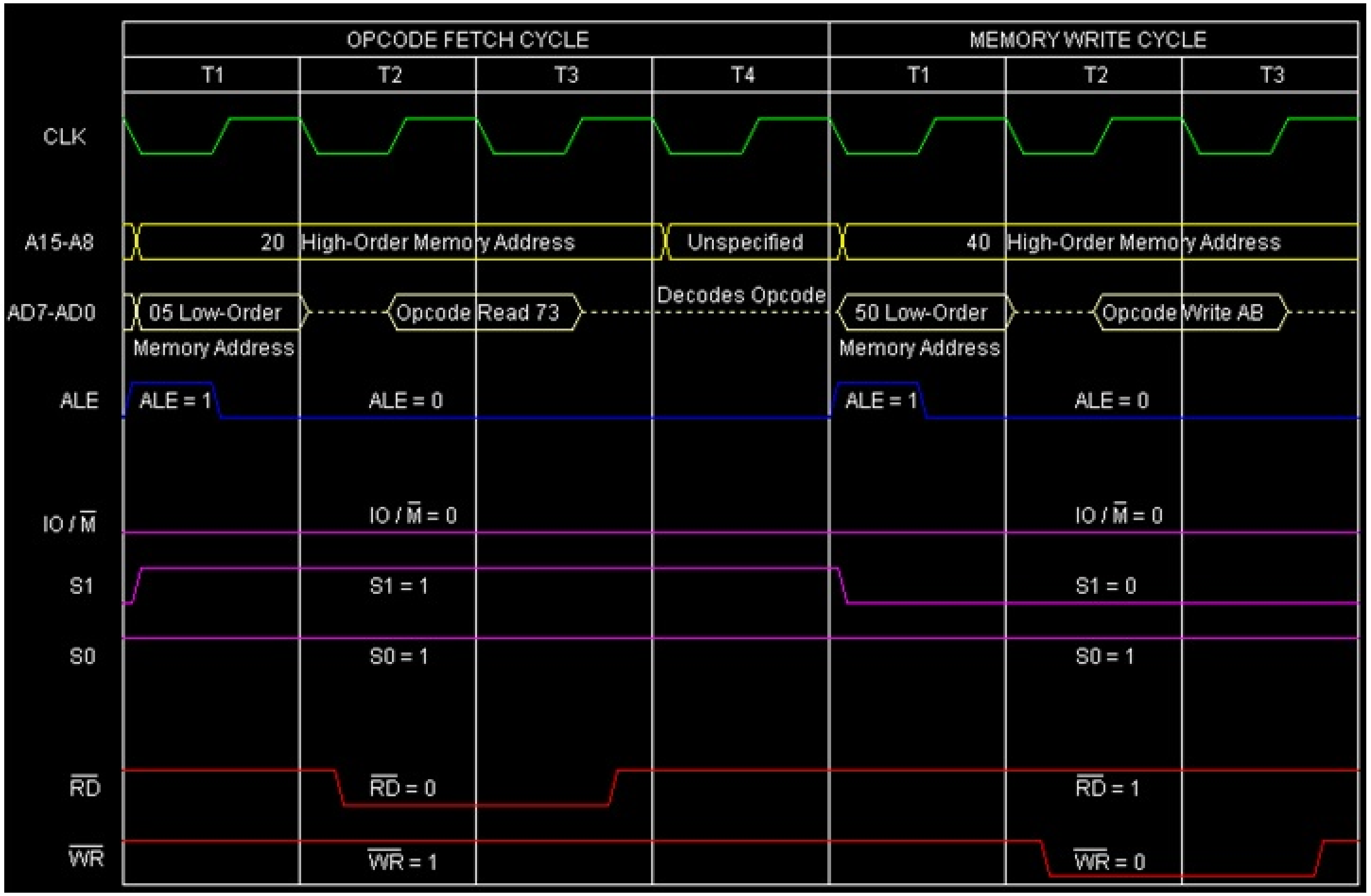
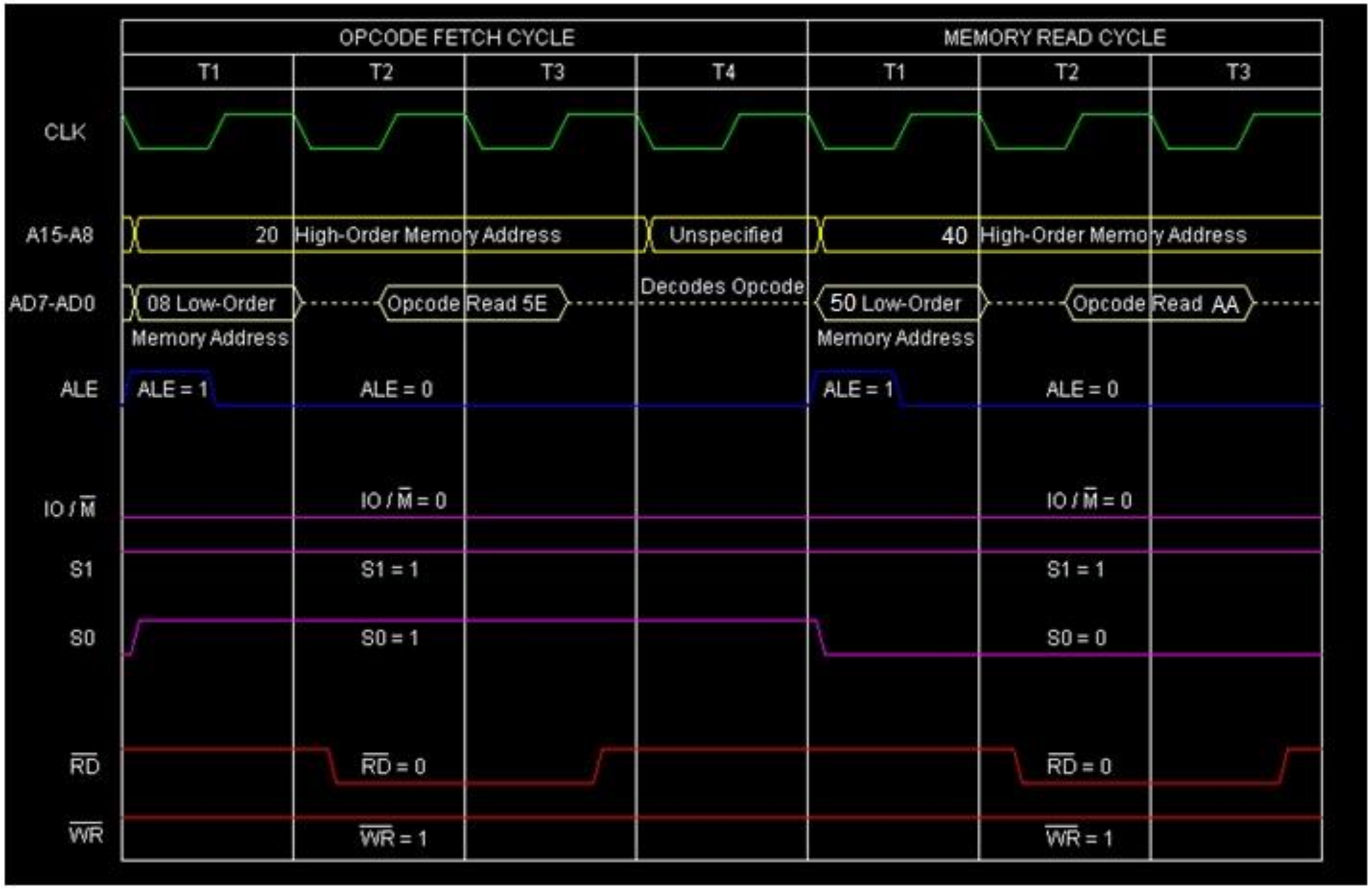


Fig: 2.34: Out instruction

## Timing Diagram of MOV M, E



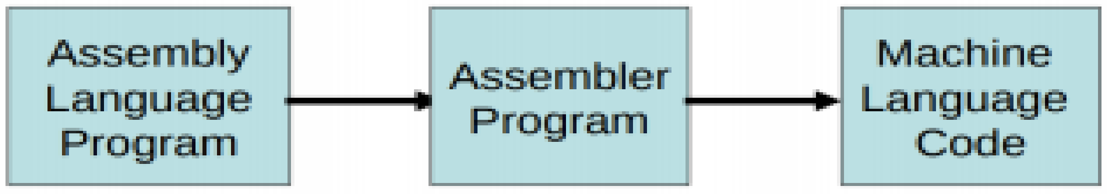
# Timing Diagram of MOV E, M

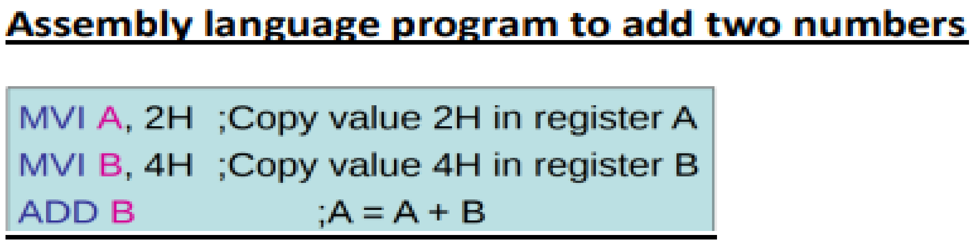


## Assembly Language Programming Basics

- An assembly language is the most basic programming language available for any processor.
- With assembly language, a programmer works only with operations that are implemented directly on the physical CPU.
- Assembly languages generally lack high-level conveniences such as variables and functions, and they are not portable between various families of processors.
- They have the same structures and set of commands as machine language, but allow a programmer to use names instead of numbers. This language is still useful for programmers when speed is necessary or when they need to carry out an operation that is not possible in high-level languages.
- Assembly language is specific to a given processor. For e.g. assembly language of 8085 is different than that of Motorola 6800 microprocessor.

- Microprocessor cannot understand a program written in Assembly language. A program known as Assembler is used to convert Assembly language program to machine language.




**Assembly language program to add two numbers**

```
MVI A, 2H  ;Copy value 2H in register A
MVI B, 4H  ;Copy value 4H in register B
ADD B                ;A = A + B
```

## Advantages of Assembly Language

a) The symbolic programming of Assembly Language is easier to understand and saves a lot of time and effort of the programmer.
b) It is easier to correct errors and modify program instructions.
c) Assembly Language has the same efficiency of execution as the machine level language.

## Disadvantages of Assembly Language

a) One of the major disadvantages is that assembly language is machine dependent. A program written for one computer might not run in other computers with different hardware configuration.

b) If you are programming in assembly language, you must have detailed knowledge of the particular microcomputer you are using.

c) Assembly language programs are not portable

- Machine language and Assembly language are both

–Microprocessor specific (**Machine dependent**)

–Low-level languages

•**Machine independent** languages are called

–High-level languages

–For e.g. BASIC, PASCAL,C++,C,JAVA, etc.

–A software called **Compiler** is required to convert a high-level language program to machine code

# 8085 Programming Model

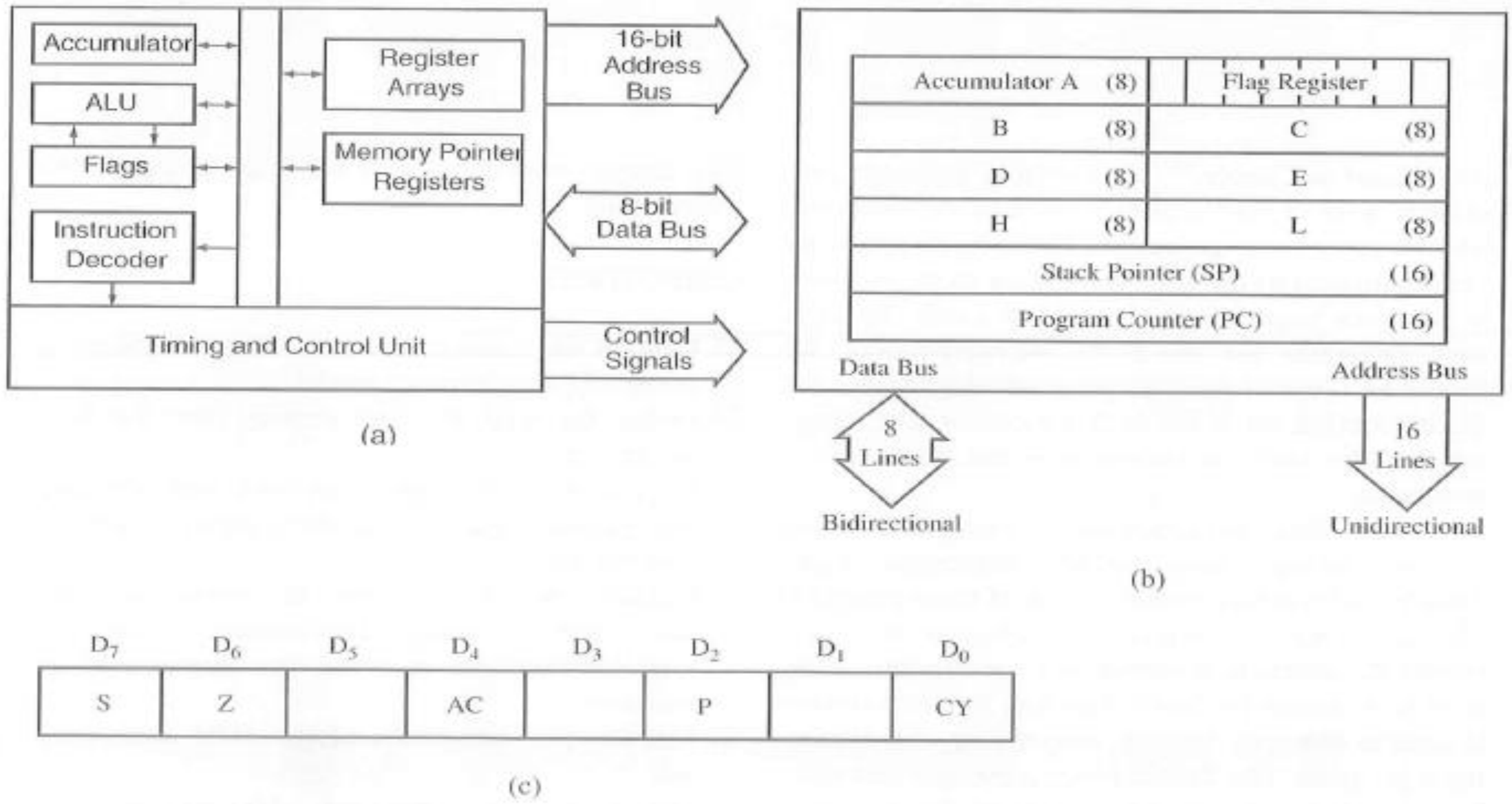


FIGURE 2.1
8085 Hardware Model (a), Programming Model (b), and Flag Register (c)

- **8085 HARDWARE MODEL**

The hardware model in fig (a) shows two major segments. One segment includes arithmetic logic unit [ALU] and an 8 bit register called an accumulator, instruction decoder, and flags. The second segment shows 8 bit and 16 bit registers. Both segments are connected with various internal connections called an internal bus. The arithmetic and logic operations are performed in the arithmetic logic unit [ALU]. Results are stored in the accumulator, and flip-flops, called flags, are set or reset to reflect the results. There are 3 buses- a 16 bit unidirectional address bus, an 8 bit bidirectional data bus, and a control bus.

- **The 8085 Programming Model**
- The 8085 programming model includes six registers, one accumulator, and one flag register. In addition, it has two 16-bit registers: the stack pointer and the program counter. They are described briefly as follows.
- **Registers**
  The 8085 has six general-purpose registers to store 8-bit data; these are identified as B,C,D,E,H, and L as shown in the figure. They can be combined as register pairs - BC, DE, and HL - to perform some 16-bit operations. The programmer can use these registers to store or copy data into the registers by using data copy instructions.

- **Accumulator**
  - The accumulator is an 8-bit register that is a part of arithmetic/logic unit (ALU). This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator. The accumulator is also identified as register A.

  **Data Bus Address Bus**
  - 8 Lines Bidirectional data bus and 16 Lines unidirectional address bus

- **Flags**
  The ALU includes five flip-flops, which are set or reset after an operation according to data conditions of the result in the accumulator and other registers. They are called Zero(Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry (AC) flags; their bit positions in the flag register are shown in the Figure below. The most commonly used flags are Zero, Carry, and Sign. The microprocessor uses these flags to test data conditions.
- For example, after an addition of two numbers, if the sum in the accumulator is larger than eight bits, the flip-flop uses to indicate a carry called the Carry flag (CY) is set to one. When an arithmetic operation results in zero, the flip-flop called the Zero(Z) flag is set to one. The first Figure shows an 8-bit register, called the flag register, adjacent to the accumulator. However, it is not used as a register; five bit positions out of eight are used to store the outputs of the five flip-flops. The flags are stored in the 8-bit register so that the programmer can examine these flags (data conditions) by accessing the register through an instruction.
- These flags have critical importance in the decision-making process of the microprocessor. The conditions (set or reset) of the flags are tested through the software instructions. For example, the instruction JC (Jump on Carry) is implemented to change the sequence of a program when CY flag is set. The thorough understanding of flag is essential in writing assembly language programs.

- **Program Counter (PC)**
  This 16-bit register deals with sequencing the execution of instructions. This register is a memory pointer. Memory locations have 16-bit addresses, and that is why this is a 16-bit register.
  The microprocessor uses this register to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte (machine code) is being fetched, the program counter is incremented by one to point to the next memory location

- **Stack Pointer (SP)**
  The stack pointer is also 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 16-bit address in the stack pointer.
  This programming model will be used in subsequent tutorials to examine how these registers are affected after the execution of an instruction.

## Instruction description and format:

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

- For example: ADD R1, R0
- Op-code address
- Here R0 is the source register and R1 is the destination register. The instruction adds the contents of R0 with the content of R1 and stores result in R1.

## Classification of instruction on the basisof address specified / Instruction Word Size

Depending on the number of address specified in instruction sheet, the instruction format can be classified into the categories.

**One address format (1 byte instruction):**

- Here 1 byte will be Op-code and operand will be default. E.g. ADD B, MOV A,B

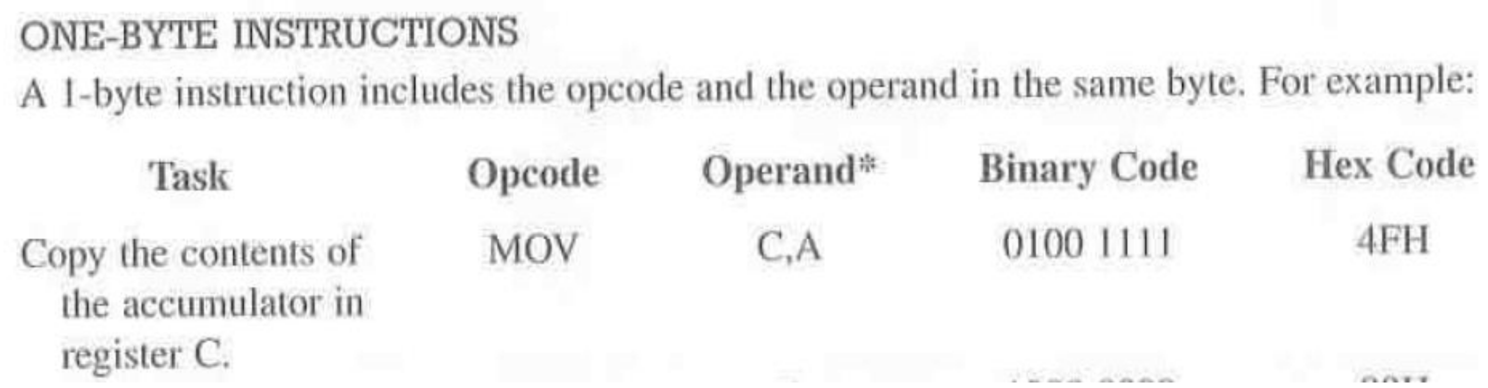
**Two address format (2 byte instruction) :**

- Here first byte will be Op-code and second byte will be the operand/data. E.g. IN 40H, MVI A, 8-bit Data

**Three address format (3 byte instruction):**

- Here first byte will be Op-code, second and third byte will be operands/data. That is 2nd byte- lower order data.
  3rd byte – higher order data
  E.g. LXI B, 4050

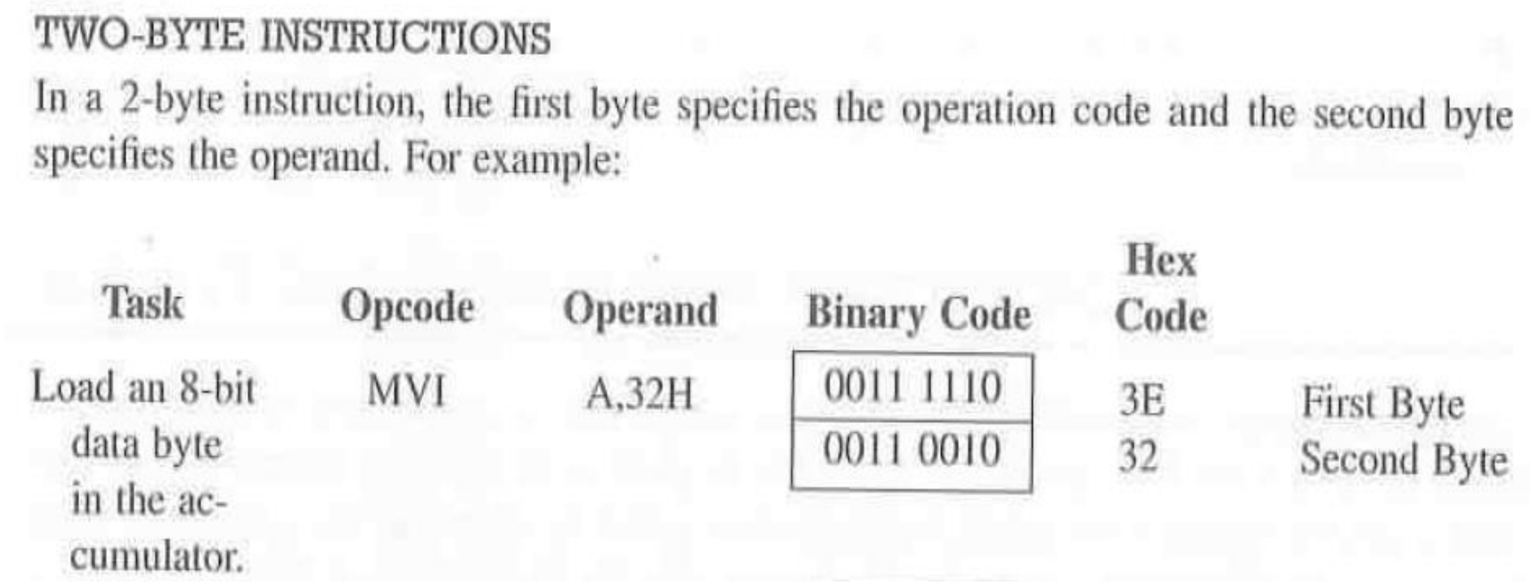
## Examples



ONE-BYTE INSTRUCTIONS

A 1-byte instruction includes the opcode and the operand in the same byte. For example:

| Task | Opcode | Operand* | Binary Code | Hex Code |
|---|---|---|---|---|
| Copy the contents of the accumulator in register C. | MOV | C,A | 0100 1111 | 4FH |

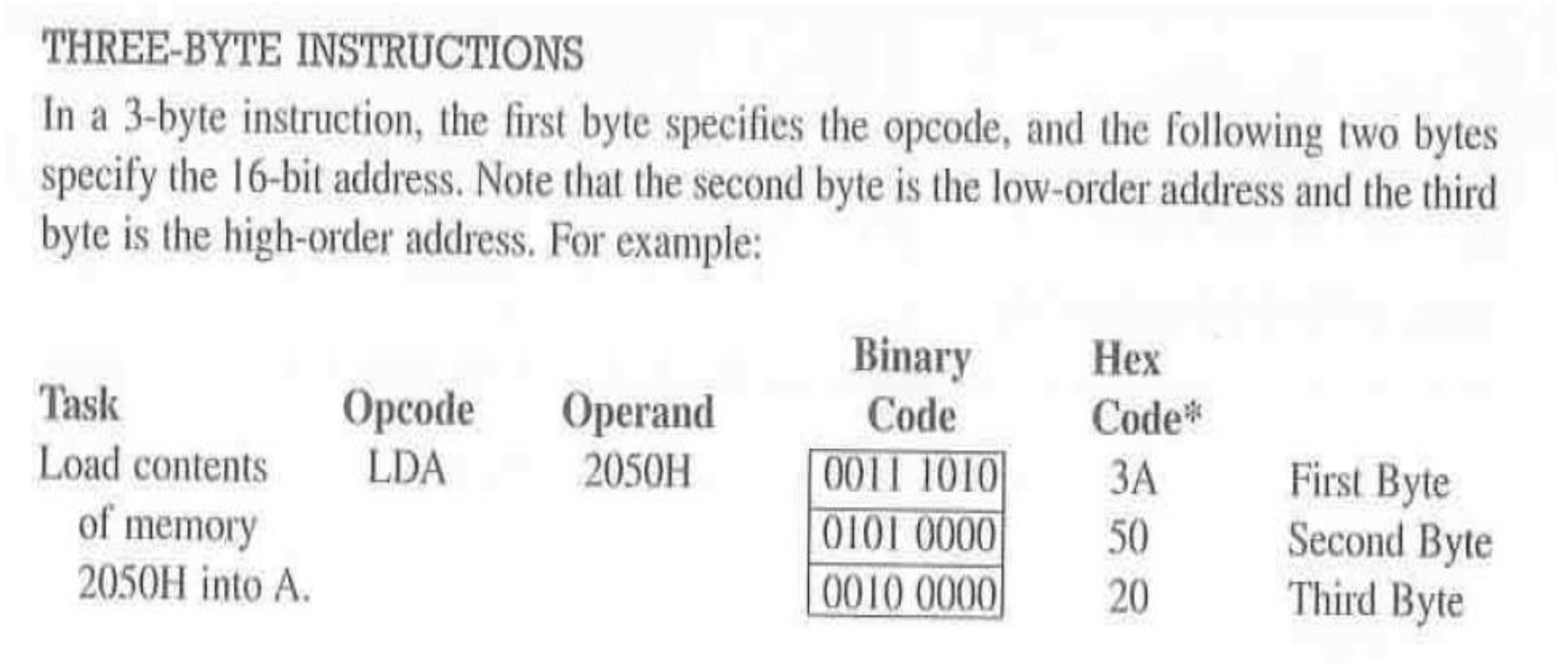


TWO-BYTE INSTRUCTIONS

In a 2-byte instruction, the first byte specifies the operation code and the second byte specifies the operand. For example:

| Task | Opcode | Operand | Binary Code | Hex Code | |
|---|---|---|---|---|---|
| Load an 8-bit data byte in the accumulator. | MVI | A,32H | 0011 1110 | 3E | First Byte |
| | | | 0011 0010 | 32 | Second Byte |

## Examples

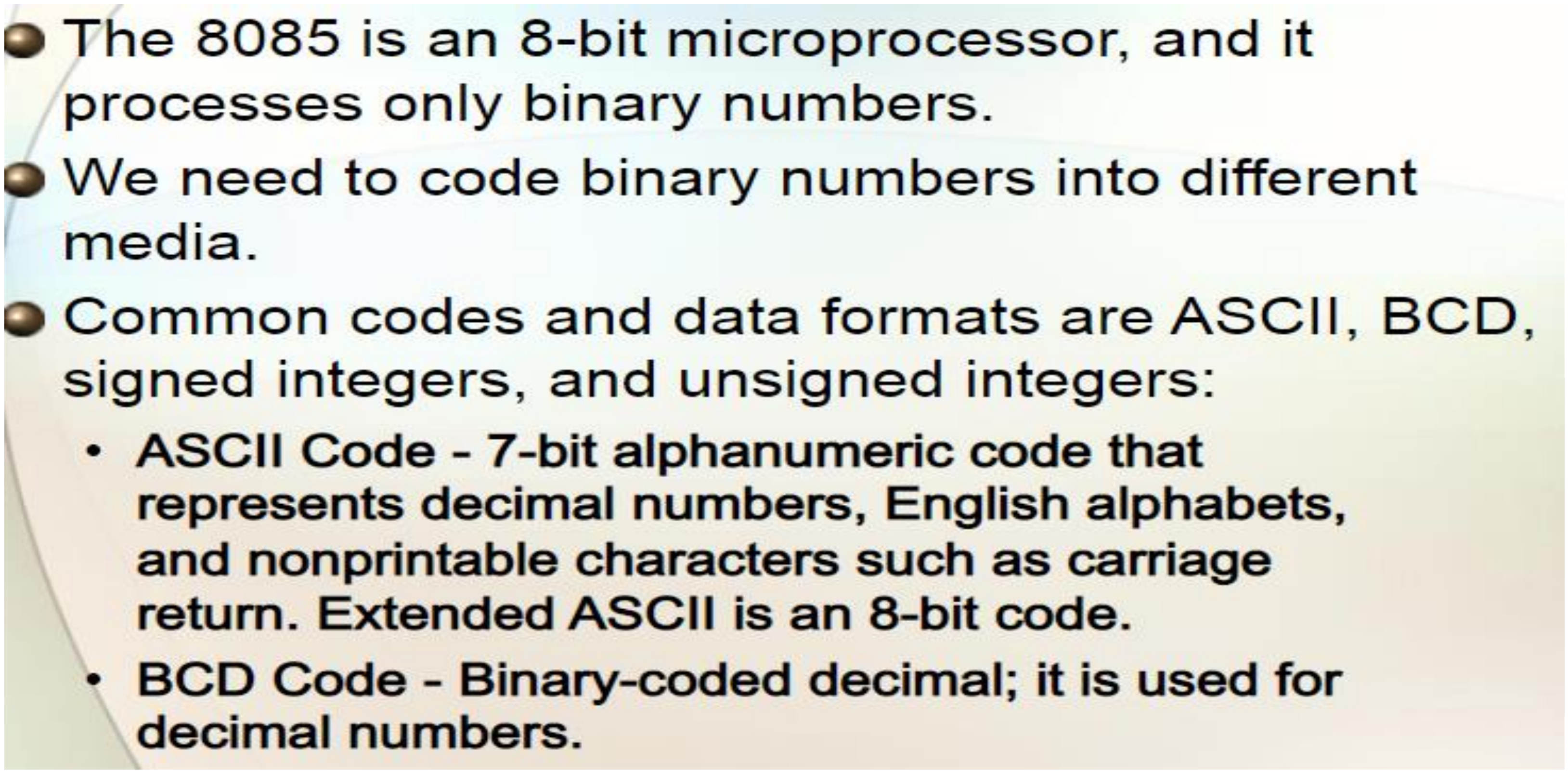


THREE-BYTE INSTRUCTIONS

In a 3-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit address. Note that the second byte is the low-order address and the third byte is the high-order address. For example:

| Task | Opcode | Operand | Binary Code | Hex Code* | |
|---|---|---|---|---|---|
| Load contents | LDA | 2050H | 0011 1010 | 3A | First Byte |
| of memory | | | 0101 0000 | 50 | Second Byte |
| 2050H into A. | | | 0010 0000 | 20 | Third Byte |

## Data Format



- The 8085 is an 8-bit microprocessor, and it processes only binary numbers.
- We need to code binary numbers into different media.
- Common codes and data formats are ASCII, BCD, signed integers, and unsigned integers:
  - **ASCII Code - 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 - Binary-coded decimal; it is used for decimal numbers.**

- Signed Integer - A signed integer is either a positive number or a negative number.
  - **In an 8-bit processor, the most significant digit, D7, is used for the sign; 0 represents the positive sign and l represents the negative sign.**
  - **The remaining seven bits, D6—D0, represent the magnitude of an integer. Therefore, the largest positive integer that can be processed by the 8085 at one time is 0111 1111 (7FH); the remaining Hex numbers, 80H to FFH, are considered negative numbers.**

- Unsigned Integers - An integer without a sign can be represented by all the 8 bits in a microprocessor register.
  - **Therefore, the largest number that can be processed at one time is FFH.**
  - **However, this does not imply that the 8085 microprocessor is limited to handling only 8-bit numbers. Numbers larger than 8 bits (such as 16-bit or 24-bit numbers) are processed by dividing them in groups of 8 bits.**

## Instruction Set of 8085

- An instruction is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions that a microprocessor supports is called Instruction Set.
- 8085 has 246 instructions. Each instruction is represented by an 8-bit binary value. These 8-bits of binary value is called Op-Code or Instruction Byte.
- Following are the classification of instructions:
- a) Data Transfer Instruction
- b) Arithmetic Instructions
- c) Logical Instructions
- d) Branching Instructions
- e) Control Instructions

**a) Data Transfer Instruction**

- These instructions move data between registers, or between memory and registers. These instructions copy data from source to destination. While copying, the contents of source are not modified.
- Example: MOV, MVI

**b) Arithmetic Instructions**

- These instructions perform the operations like addition, subtraction, increment and decrement.
- Example: ADD, SUB, INR, DCR

**c) Logical Instructions**

- These instructions perform logical operations on data stored in registers and memory. The logical operations are: AND, OR, XOR, Rotate, Compare and Complement.
- Example: ANA, ORA, RAR, RAL, CMP, CMA

**d) Branching Instructions**

- Branching instructions refer to the act of switching execution to a different instruction sequence as a result of executing a branch instruction. The three types of branching instructions are: Jump, Call and Return.

**e) Control Instructions**

- The control instructions control the operation of microprocessor. Examples: HLT, NOP, EI (Enable Interrupt), DI (Disable Interrupt).

## Instruction Set of 8085

8085 instructions can be classified as

1. **Data Transfer (Copy) instructions**
2. **Arithmetic instructions**
3. **Logical and Bit manipulation instructions**
4. **Branching instructions**
5. **Miscellaneous instructions / Control**

## Data Transfer Operations

- The data transfer instructions load given data into register, copy data from register to register, copy data from register to memory location, and vice versa.
- In other words we can say that data transfer instructions copy data from source to destination.
- Source can be data or contents of register or contents of memory location whereas destination can be register or memory location.
- These instructions do not affect the flag register of the processor.

# Instructions

**1) MOV Rd, Rs(move register instruction)**

- 1 byte instruction
- Copies data from source register to destination register.
- Rd & Rs may be A, B, C, D, E, H&L
- E.g. MOV A, B

**2) MVI R, 8 bit data (move immediate instruction)**

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

## Cont..

**3) MOV M, R(Move to memory from register)**

- Copy the contents of the specified register to memory. Here memory is the location specified by contents of the ***HL register pair***.
- E.g. MOV M, B

**4) MOV R, M (move to register from memory)**

- Copy the contents of memory location specified by **HL pair** to specified register.
- E.g. MOV B, M

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

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

## Cont..

**5) LXI $R_P$, 2 bytes data (load register pair)**

- 3-byte instruction
- Load immediate data to register pair – Register pair may be BC, DE, HL & SP(Stack pointer)
- 1st byte- Op-code
- 2nd byte – lower order data
- 3rd byte- higher order data
- E.g. LXI B, 4532H; $B \leftarrow 45$, $C \leftarrow 32H$

## Cont..

**6) MVI M, data (load memory immediate)**

- 2 byte instruction.
- Loads the 8-bit data to the memory location whose address is specified by the contents of HL pair. E.g. MVI M ,35H; [HL]$\leftarrow$35H

## 7) LDA 4035H (Load accumulator direct)

- 3-byte instruction
- Loads the accumulator with the contents of memory location whose address is specified by 16 bit address.
- $A \leftarrow [4035H]$

## Cont..

**8. STA 16-bit address (store accumulator contents direct)**

- 3-byte instruction.
- Stores the contents of accumulator to specified address
- E.g. STA FA00H

**9. LDAX RP (Load accumulator indirect)**

- 1 byte instruction.
- Loads the contents of memory location pointed by the contents of register pair to accumulator.
- E. g. LDAX B

```
    LXI B, 9000H        B=90, C=00
    LDAX B              A=[9000]
```

## Cont..

**10) STAX RP**

- Stores the contents of accumulator to memory location specified by the contents of register pair.
- 1 byte instruction
- Example

```
LXI B, 9500H
LXI D, 9501H
MVI  A,  32H
STAXB
MVI A, 7AH
STAX D
```

## Cont..

**11) IN 8-bit address**

- 2-byte instruction
- Read data from the input port address specified in the second byte and loads data into the accumulator.
- E.g. IN 40H

**12) OUT 8-bit address**

- 2-byte instruction
- Copies the contents of the accumulator to the output port address specified in the 2nd byte
- E.g. OUT 40H

## Cont..

**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.
- Eg.

```
LXI H, 9500H
MVI M, 32H
MVI L, 01H
MVI m, 7AH
LHLD 9500H
```

## Cont..

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

- Opposite to LHLD.
- Stores the contents of L register to specified memory location and contents of H register to next higher memory location.
- E.g. LXI H, 9500H

  SHLD 8500H

## Cont..

**15) XCHG (Exchange)**

- Exchanges DE pair with HL pair.
- E.g. LXI H, 7500H H= 75, L=00

  LXI D, 9532H D=95. E=32

  XCHG H=95, L=32

  D=75 E=00

## Arithmetic group Instructions

- 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.

## Arithmetic group Instructions

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

1) **ADD R/M**

- 1 byte add instruction.
- Adds the contents of register/memory to the contents of the accumulator and stores the result in accumulator.
- E.g. ADD B; $A \leftarrow A + B$

## Cont..

**2) ADI 8 bit data**

- 2 byte add immediate instruction.
- Adds the 8 bit data with the contents of accumulator and stores result in accumulator.
- Eg. ADI 9BH ; A$\leftarrow$A+9BH

**3) SUB R/M**

- 1 byte subtract instruction.
- Subtracts the contents of specified register / m with the contents of accumulator and stores the result in accumulator.
- E.g. SUB D ; A$\leftarrow$A-D

## Cont..

**4) SUI 8 bit data**

- 2 byte subtract immediate instruction.
- Subtracts the 8 bit data from the contents of accumulator stores result in accumulator.
- E.g. SUI D3H $A \leftarrow A - D3H$

# Cont..

**5) INR R/M, DCR R/M**

- 1 byte increment and decrement instructions
- Increase and decrease the contents of R(register) or M(memory) by 1 respectively.

```
E.g.    DCR B      ; B=B-1
        DCR M      ; [HL] = [HL]-1
        INR A      ; A=A+1
        INR M      ; [HL] +1
```

For these, all flags are affected except carry.

## Cont..

**6. INX Rp, DCX RP**

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

## Cont..

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

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

  ACI 70H ; A=A+ 70+CY

## Cont..

**8) SBB B/M**

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

**SBI 8 bit data**

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

## Cont..

**9) DAD Rp(double addition)**

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

  LXI B, 4220H

  DAD B; HL=HL+BC

## Cont..

**10) DAA (Decimal adjustment accumulator)**

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

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

## Logical Group Instructions:

- Microprocessor 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

**1) ANA R/M (the contents of register/memory)**

- Logically AND the contents of register/memory with the contents of accumulator.
- 1 byte instruction.
- **CY flag is reset and AC is set.**

2**) ANI 8 bit data**

- Logically AND 8 bit immediate data with the contents of accumulator.
- 2 byte instruction.
- **CY flag is reset and AC is set**. Others as per result

**3) ORA R/M**

- Logically OR the contents of register/memory with the contents of accumulator.
- 1 byte instruction.
- **CY and AC is reset and other as per result**.

4) ORI 8 bit data

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

**5) XRA R/M**

- Logically exclusive ORthe contents of register memory with the contents of accumulator.
- 1 byte instruction.
- **CYand ACis reset and other as per result.**

**6) XRI 8 bit data**

- Logically Exclusive OR8 bit data immediate with the content of accumulator.
- 2 byte instruction.
- **CYand ACis reset and other as per result**.

**7) CMA (Complement accumulator)**

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

# Logically Compare instructions

7. **CMP R/M (1 byte instruction)**

**CPI 8 bit data ( 2 byte instruction)**

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

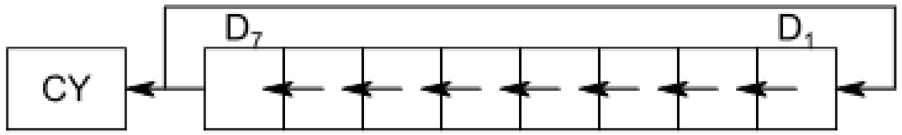
| Case | CY | Z | |
|---|---|---|---|
| [A]<[R/M] or 8bit | 1 | 0 | A-R<0 |
| [A]=[R/M] or 8bit | 0 | 1 | A-R=0 |
| [A]>[R/M] or 8bit | 0 | 0 | A-R>0 |

## Logical Rotate instructions

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

**1) RLC: Rotate accumulator left**

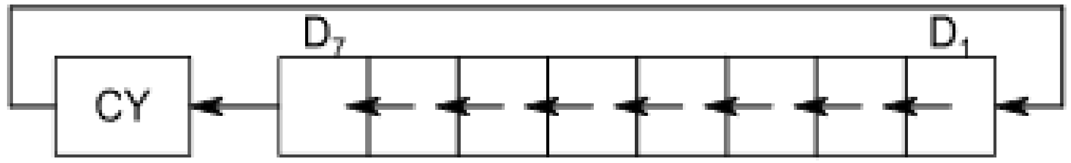
- Each bit is shifted to the adjacent left position.
- Bit D7 becomes D0.
- The carry flag is modified according to D7



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

## 2) RAL: Rotate accumulator left through carry

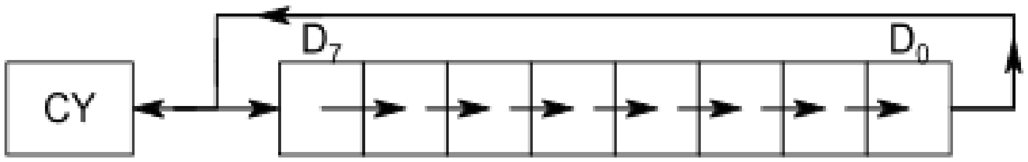
- Each bit is shifted to the adjacent left position.
- Bit D7 becomes the carry bit and the carry bit is shifted into D0.
- The carry flag is modified according to D7.



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

## 3) RRC:rotate accumulator right

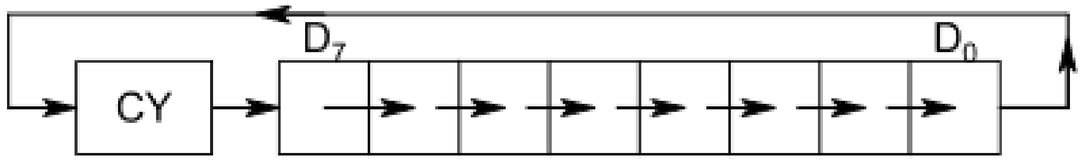
- Each bit is shifted right to the adjacent position.
- Bit D0 becomes D7.
- The carry flag is modified according to D0.



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

## 4) RAR: Rotate accumulator right through carry

- Each bit is shifted right to the adjacent position.
- Bit D0 becomes the carry bit and the carry bit is shifted into D7.



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

## 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 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.**

**1. Unconditional Jump**

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

E.g. JMP 4000H

## Conditional Jump

- The conditional jump instructions allow the microprocessor to make decisions based on certain conditions indicated by the flags.
- After logic and arithmetic operations, flags are set 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.

## Cont…

| Mnemonics | Description |
|---|---|
| JC16 bit | Jump on carry (if CY=1) |
| JNC16 bit | Jump on if no carry (if CY=0) |
| JZ16bit | Jump on zero (if Z=1) |
| JNZ16bit | jump on if no zero (if Z=0) |
| JP 16bit | jump on positive (if S=0) |
| JM 16bit | jump on negative (if S=1) |
| JPE16bit | Jump on parity even (if P=1) |
| JPO 16bit | Jump on parity odd (if P=0) |

## WAP to move 10 bytes of data from starting address 9500 H to 9600H

```
2000    MVI B, 0AH
2002    LXI H, 9500H
2005    LXI D, 9600H
2008    MOV A, M
2009    STAX D    ; Store the contents of accumulator to register pair.
200A    INX H     ; Increment the register pair by 1.
200B    INX D
200C    DCR B
200D    JNZ      2008
2010    HLT
```

## 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 atask.
- After the completion of subroutine task microprocessor returns to main program.
- The subroutine technique eliminates the need to write a subtask repeatedly, thus it uses memoryefficiently.

- To implement subroutine there are two instructions CALL and RET.

1. **CALL16 bit memory**

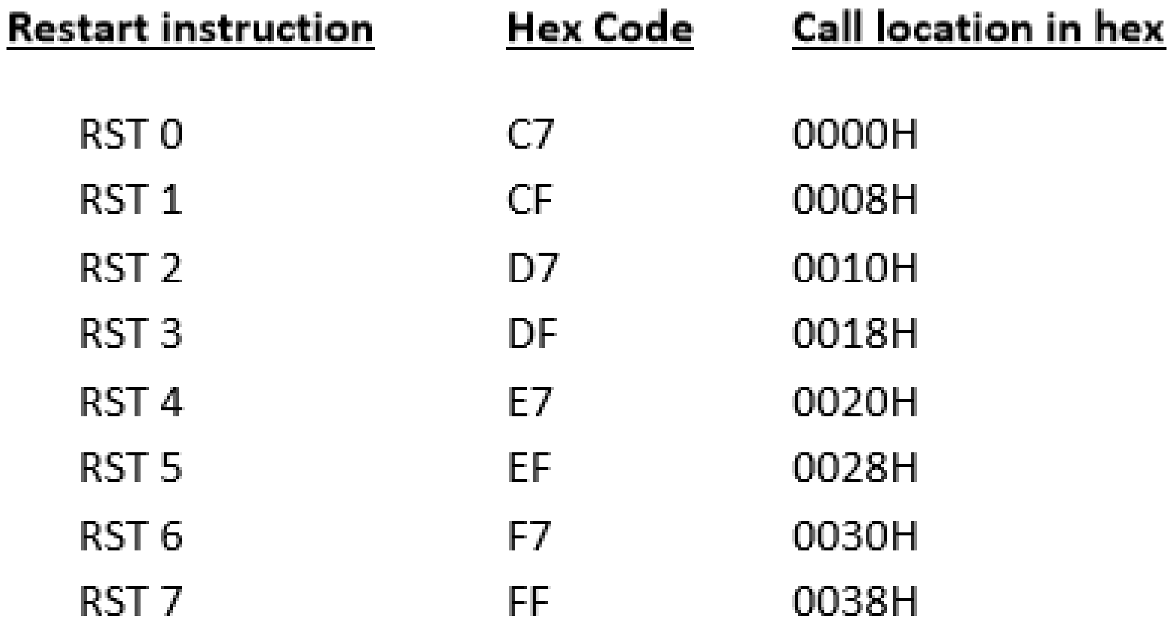
- Call subroutine unconditionally.
- 3 byte instruction.
- Saves the contents of program counter on the stack pointer. Loads the PC by jump address (16 bit memory) and executes the subroutine.

2. **RET**

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

## 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 in hex |
|---|---|---|
| RST 0 | C7 | 0000H |
| RST 1 | CF | 0008H |
| RST 2 | D7 | 0010H |
| RST 3 | DF | 0018H |
| RST 4 | E7 | 0020H |
| RST 5 | EF | 0028H |
| RST 6 | F7 | 0030H |
| RST 7 | FF | 0038H |

## Cont..

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

## Miscellaneous Group Instructions:

**STACK**

- The stack is defined as a set of memory location in R/W memory, specified by a programmer in a main memory. These memory locations are used to store binary information temporarily during the execution of a program.
- The beginning of the stack is defined in the program by using the instruction LXI SP,16 bit address.
- Once the stack location is defined, it loads 16 bit address in the stack pointer register. Storing of data bytes for this operation takes place at the memory location that is one less than the address
- e.g. LXI SP,2099H

- The stack instructions are:

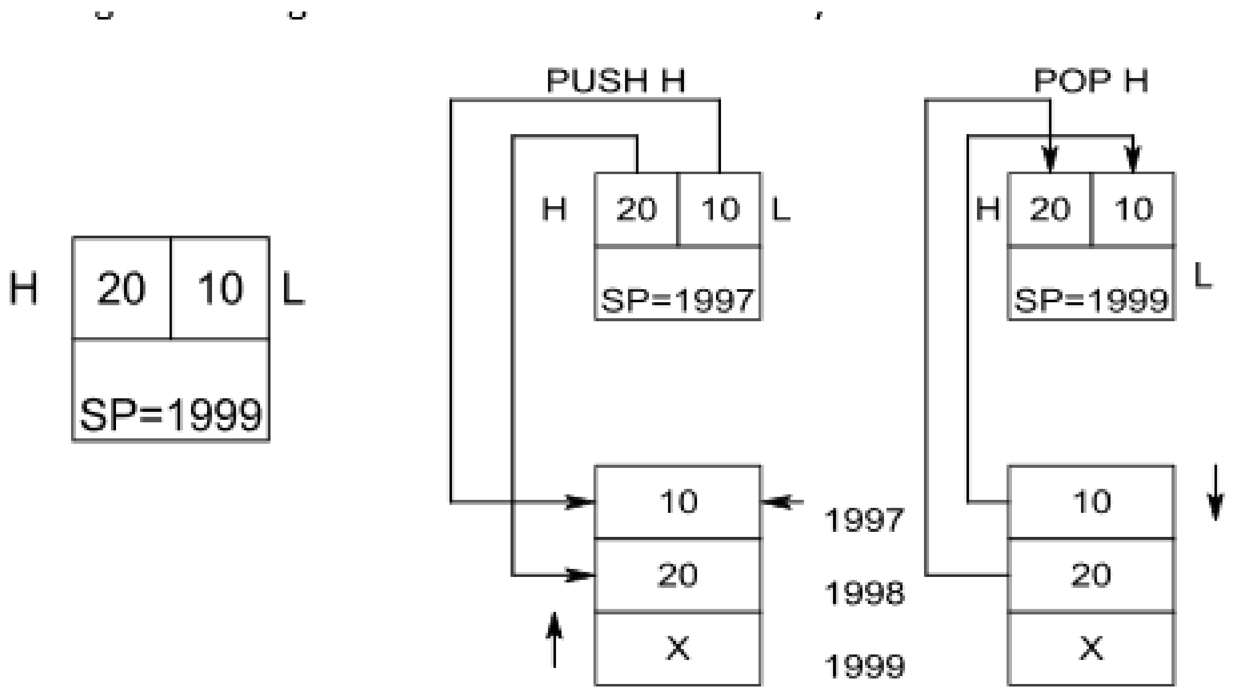
1. **PUSH Rp/PSW (Store register pair on stack)**

- 1 byte instruction.
- Copies the contents of specified register pair or program status word (accumulator and flag) on the stack.
- Stack pointer is decremented and content of high order register is copied. Then it is again decremented and content of low order register is copied.

2. **POP Rp/PSW (retrieve register pair from stack)**

- 1 byte instruction.
- Copies the contents of the top two memory locations of the stack into specified register pair or program status word.

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

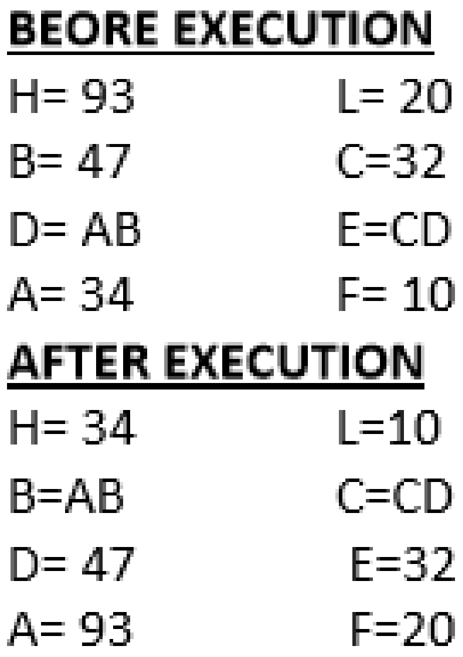


## Cont..

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

## Example

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



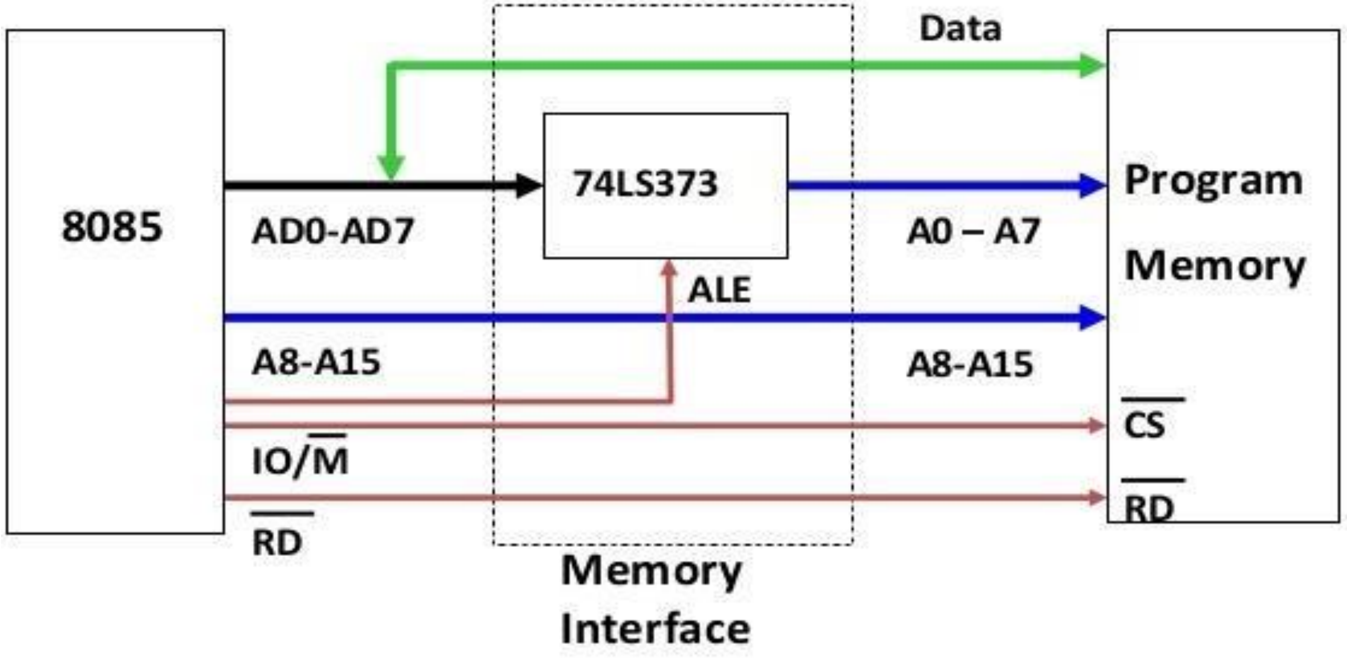
**BEORE EXECUTION**

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

**AFTER EXECUTION**

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

# 8085 Interfacing with Memory chips



- **PROGRAMS**