

**LECTURE NOTES**  
**ON**  
**MICROPROCESSOR & MICROCONTROLLER**

**DIPLOMA**

**Subject code-TH3**

**4<sup>TH</sup> SEMESTER , E&TC ENGINEERING**



**PREPARED BY**

**YASOBANTI NAYAK (Lecturer in E&TC)**  
**DEPT. OF ELECTRONICS & TELECOMMUNICATION ENGINEERING**  
**JHARSUGUDA ENGINEERING SCHOOL, JHARSUGUDA**

## **Unit-1:Microprocessor (Architecture and Programming-8085-8-bit)**

- 1.1 Introduction to Microprocessor and Microcomputer & distinguish between them.
- 1.2 Concept of Address bus, Data bus, Control bus & System Bus
- 1.3 General Bus structure Block diagram.
- 1.4 Basic Architecture of 8085 (8 bit) Microprocessor
- 1.5 Signal Description (Pin diagram) of 8085 Microprocessor
- 1.6 Register Organizations,Distinguish between SPR & GPR, Timing & Control Module,
- 1.7 Stack, Stack pointer &Stack top.

- 1.8 Interrupts:-8085 Interrupts, Masking of Interrupt(SIM,RIM)

## **Unit-2: Instruction Set and Assembly Language Programming**

- 2.1 Addressing data & Differentiate between one-byte, two-byte &three-byte instructions with examples.
- 2.2 Addressing modes in instructions with suitable examples.
- 2.3 Instruction Set of 8085(Data Transfer, Arithmetic, Logical, Branching, Stack& I/O , Machine Control)
- 2.4 Simple Assembly Language Programming of 8085
  - 2.4.1 Simple Addition & Subtraction
  - 2.4.2 Logic Operations (AND, OR, Complement 1's & 2's) & Masking of bits
  - 2.4.3 Counters & Time delay (Single Register, Register Pair, More than Two Register)
  - 2.4.4 Looping, Counting & Indexing (Call/JMP etc).
  - 2.4.5 Stack & Subroutine programmes.
  - 2.4.6 Code conversion, BCD Arithmetic & 16 Bit data Operation, Block Transfer.
  - 2.4.7 Compare between two numbers
  - 2.4.8 Array Handling (Largest number & smallest number in the array)
- 2.5 Memory & I/O Addressing,

## **Unit-3: TIMING DIAGRAMS.**

- 3.1 Define opcode, operand, T-State, Fetch cycle, Machine Cycle, Instruction cycle & discuss the concept of timing diagram.
- 3.2 Draw timing diagram for memory read, memory write, I/O read, I/O write machine cycle.
- 3.3 Draw a neat sketch for the timing diagram for 8085 instruction (MOV, MVI, LDA instruction).

## **Unit-4 Microprocessor Based System Development Aids**

- 4.1 Concept of interfacing
- 4.2 Define Mapping &Data transfer mechanisms - Memory mapping & I/O Mapping
- 4.3 Concept of Memory Interfacing:- Interfacing EPROM & RAM Memories
- 4.4 Concept of Address decoding for I/O devices
- 4.5 Programmable Peripheral Interface: 8255
- 4.6 ADC & DAC with Interfacing.
- 4.7 Interfacing Seven Segment Displays
- 4.8 Generate square waves on all lines of 8255
- 4.9 Design Interface a traffic light control system using 8255.
- 4.10 Design interface for stepper motor control using 8255.
- 4.11 Basic concept of other Interfacing DMA controller,USART

## **Unit-5 Microprocessor (Architecture and Programming-8086-16 bit)**

- 5.1 Register Organisation of 8086
- 5.2 Internal architecture of 8086
- 5.3 Signal Descriptionof 8086
- 5.4 General Bus Operation& Physical Memory Organisation
- 5.5 MinimumMode&Timings,
- 5.6 Maximum Mode&Timings,
- 5.7 Interrupts and Interrupt Service Routines, Interrupt Cycle, Non-Maskable Interrupt, Maskable Interrupt
- 5.8 8086 Instruction Set & Programming: Addressing Modes, Instruction Set, Assembler Directives and Operators,
- 5.9 Simple Assembly language programmingusing 8086 instructions.

## **Unit-6 Microcontroller (Architecture and Programming-8 bit):-**

- 6.1 Distinguish between Microprocessor & Microcontroller
- 6.2 8 bit & 16 bit microcontroller
- 6.3 CISC & RISC processor
- 6.4 Architectureof8051Microcontroller

## **1.1 Introduction to Microcomputer AND Microprocessor & distinguish between them.**

### **Microcomputer**

A microcomputer can be defined as a small sized, inexpensive, and limited capability computer. It has the same architectural block structure that is present on a computer. Present-day microcomputers are having smaller sizes. Nowadays, they are of the size of a notebook. But in the coming days also their sizes will get more reduced as well. Due to their lower costs, individuals can possess them as their personal computers. Because of mass production, they are becoming still cheaper. Initially, in the earlier days, they were not very much powerful. Their internal operations and instructions were very much limited and restricted. But at present days, microcomputers have not only multiplied and divide instructions on unsigned and signed numbers but are also capable of performing floating point arithmetic operations. In fact, they are becoming more powerful than the minicomputers and main computers of yesteryear.

As an example, the Commodore 64 was one of the most popular microcomputers of its era and is the best-selling model of home computer of all time.

So a microcomputer is a small, relatively inexpensive computer with a microprocessor as its central processing unit (CPU). It includes a single printed circuit board containing a microprocessor, memory, and minimal input/output(I/O) circuitry mounted. With the advent of increasingly powerful microprocessors, microcomputers became popular in the 1970s and 1980s. The predecessors to these computers, mainframes, and minicomputers, were comparatively much larger and more expensive(though indeed present-day mainframes such as the IBM System z machines use one or more custom microprocessors as their CPUs). Also, we can mention that many microcomputers, in the generic sense, (when equipped with a keyboard and screen for input and output) are also personal computers.

## **Microprocessor**

The processor on a single chip is called a Microprocessor which can process micro-instructions. Instructions in the form of 0s and 1s are called micro-instructions. The microprocessor is the CPU part of a microcomputer, and it is also available as a single integrated circuit. Thus as main components, the microprocessor will have the Control Unit (CU) and the Arithmetic Logic Unit (ALU) of a microcomputer. An example is Intel 8085 microprocessor. In addition to the microprocessor features, a microcomputer will have the following additional features:

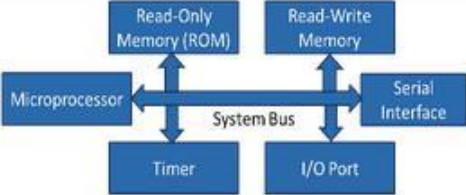
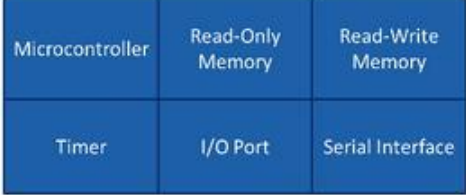
- ROM/PROM/EPROM/EEPROM for storing program;
- RAM for storing data, intermediate results, and final results;
- I/O devices for communication with the outside world;
- I/O ports for communication with the I/O devices.

In the present-day world, Microprocessors are extensively used. Before the microprocessor's invention, the logic design was done by hardware using gates, flip-flops, etc. A mini-computer was too much costly. With the advent of the microprocessor, logic design using hardware has been mostly replaced. It provides flexibility instrumentation where the characteristics of the system can be changed just by changing the software. Also, new generations of applications have surfaced, which were not thought of earlier because of the prohibitive cost of a minicomputer or the complexity of logic design using hardware.

Some of the applications where microprocessors have been used are listed below –

- Business applications such as desktop publishing;
- Industrial applications such as power plant control;
- Measuring instruments such as multimeter;

- Household equipment such as washing machine;
- Medical equipment such as blood pressure monitor;
- Defense equipment such as light combat aircraft;
- Computers such as a personal computer.

Microprocessor	Micro Controller
	
Microprocessor is heart of Computer system.	Micro Controller is a heart of embedded system.
It is just a processor. Memory and I/O components have to be connected externally	Micro controller has external processor along with internal memory and i/O components
Since memory and I/O has to be connected externally, the circuit becomes large.	Since memory and I/O are present internally, the circuit is small.
Cannot be used in compact systems and hence inefficient	Can be used in compact systems and hence it is an efficient technique
Cost of the entire system increases	Cost of the entire system is low
Due to external components, the entire power consumption is high. Hence it is not suitable to used with devices running on stored power like batteries.	Since external components are low, total power consumption is less and can be used with devices running on stored power like batteries.
Most of the microprocessors do not have power saving features.	Most of the micro controllers have power saving modes like idle mode and power saving mode. This helps to reduce power consumption even further.
Since memory and I/O components are all external, each instruction will need external operation, hence it is relatively slower.	Since components are internal, most of the operations are internal instruction, hence speed is fast.
Microprocessor have less number of registers, hence more operations are memory based.	Micro controller have more number of registers, hence the programs are easier to write.
Microprocessors are based on von Neumann model/architecture where program and data are stored in same memory module	Micro controllers are based on Harvard architecture where program memory and Data memory are separate
Mainly used in personal computers	Used mainly in washing machine, MP3 players

## 1.2 Concept of Address bus, Data bus, Control bus & System Bus

### SYSTEM BUSES

- Set of wires, that interconnects all the components (subsystems) of a computer
  - A source component sources out data onto the bus
  - A destination component inputs data from the bus
- May have a hierarchy of buses
  - Address, data and control buses to access memory and an I/O controller.
  - Second set of buses from I/O controller to attached devices/peripherals
  - Peripheral Component Interconnect(PCI) bus is an example of a very common local bus

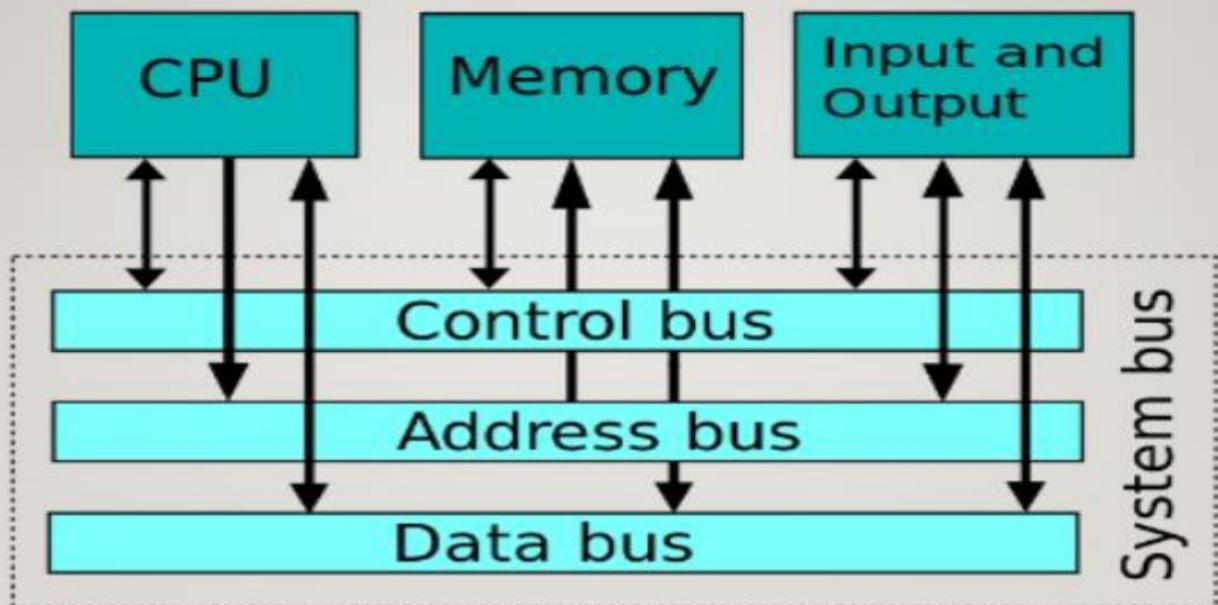


Fig: System Bus (Data,Address and Control Bus)

# ADDRESS BUS

---

- It is a channel which transmits addresses of data (not the data) from the CPU to memory.
  - The address bus consists of 16,24, or 32 parallel signal lines.
  - The number of lines (wires) determines the amount of memory that can be directly addressed as each line carries one bit of the address.
  - If the CPU has  $N$  address lines, then it can directly address  $2^N$  address lines.
  - For example, a computer with 32 bit address can address 4GB of physical memory.
- 
- CPU reads/writes data from the memory by addressing a unique location; outputs the location of the data (aka address) on the address bus; memory uses this address to access the proper data.
  - Each I/O device (such as monitor, keypad, etc) has a unique address as well (or a range of addresses); when accessing a I/O device, CPU places its address on the address bus. Each device will detect if it is its own address and act accordingly
  - Devices always receive data from the CPU; CPU never reads the address buss (it is never addressed)

# DATA BUS

---

- Data bus is a channel across which **actual data are transferred** between the CPU, memory and I/O devices.
  - The data bus consists of 8, 16, 32 or 64 parallel signal lines. Because each wire can transfer 1 bit of data at a time, an 8 wire bus can move 8 bits at a time which is a full byte.
  - The **number of wires in the bus affects the speed** at which data can travel between hardware components. The wider the data bus, more data it can carry at one time.
  - The data bus is **bidirectional** this means that the CPU can read data in from memory or it can send data out to memory.
- 
- When the CPU fetches data from memory, it first outputs the address on the address bus, then the memory outputs the data onto the data bus; the CPU reads the data from data bus
  - When writing data onto the memory, the CPU outputs first the address on the address bus, then outputs the data onto the output bus; memory then reads and stores the data at the proper location
  - The process to read/write to a I/O device is similar

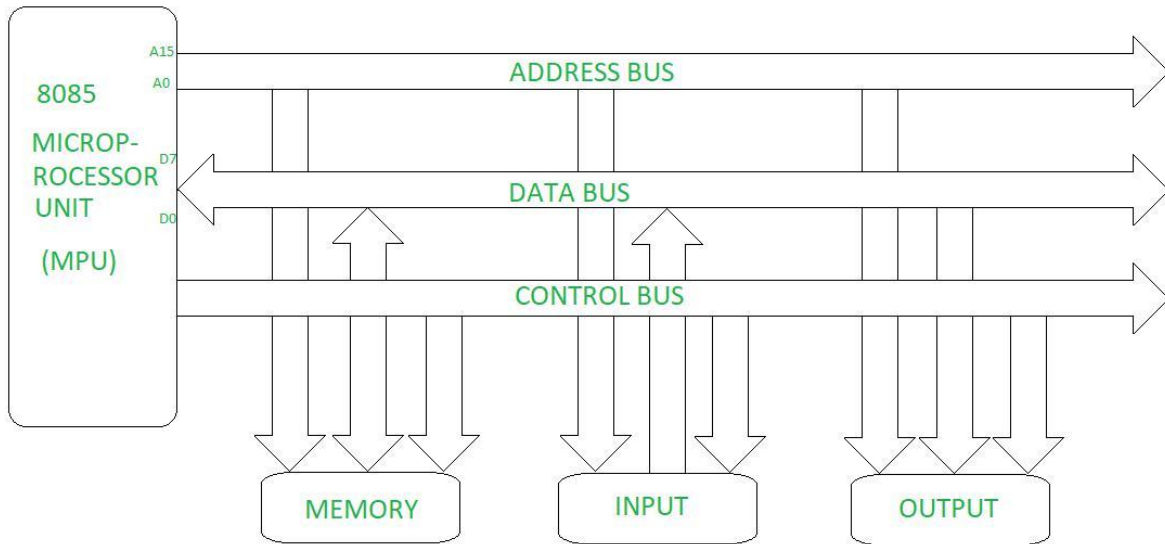


# CONTROL BUS

---

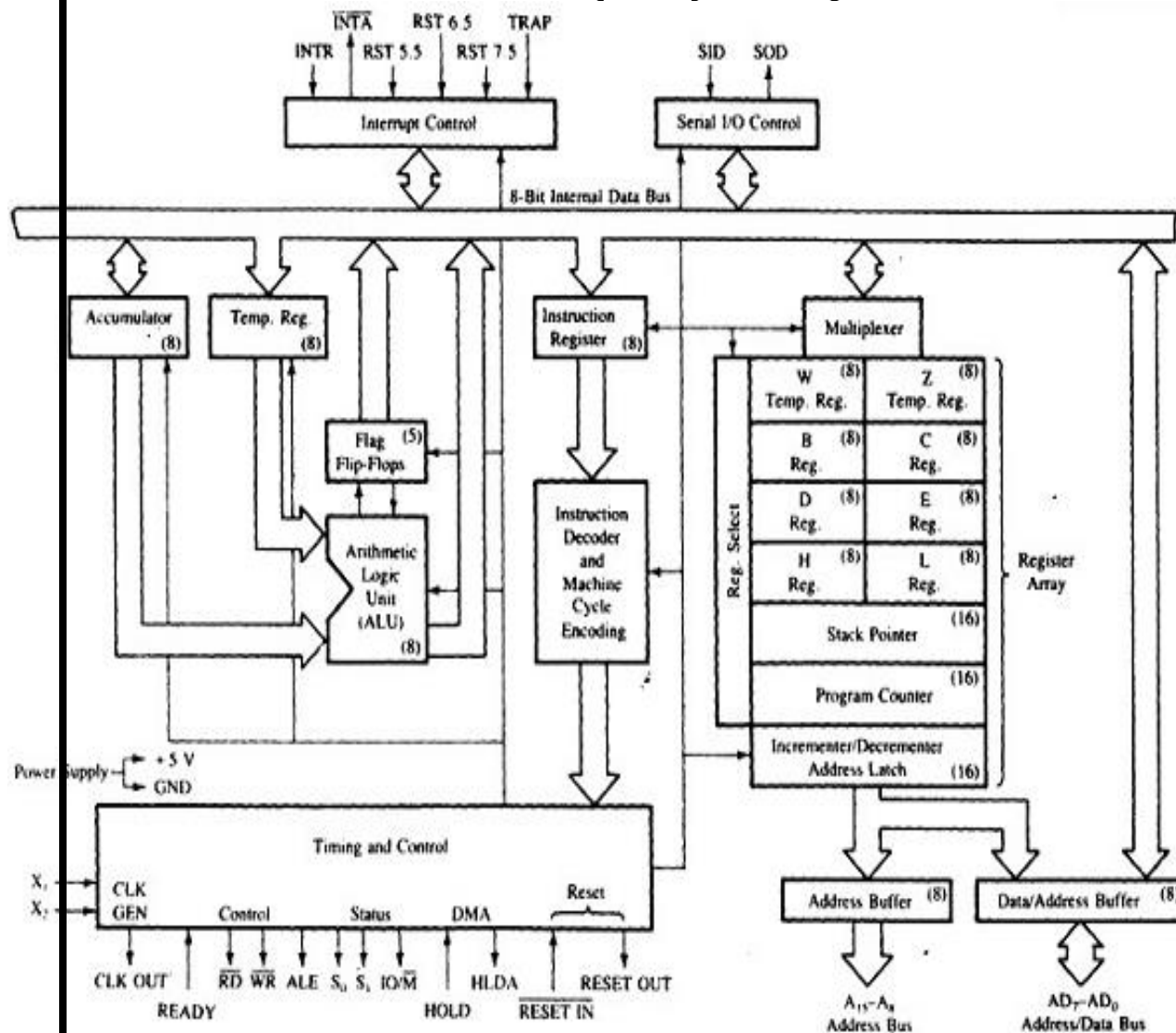
- The physical connections that carry **control information** between the CPU and other devices within the computer. This bus is mostly a collection of unidirectional signals.
- It is the **path for all timing and controlling functions** sent by the control units to other units of the system.
- It carries signals that **report the status of various devices**.
  - These signals indicate whether the data is to be read into or written out the CPU, whether the CPU is accessing memory or an IO device, and whether the I/O device or memory is ready for the data transfer
- For example, one line of the bus is used to indicate whether the CPU is currently reading from or writing to main memory. Others are *I/O Read/Write*

### 1.3 General Bus structure Block diagram



Bus organization system of 8085 Microprocessor

## 1.4 Basic Architecture of 8085 (8 bit) Microprocessor



8085 is pronounced as "eighty-eighty-five" microprocessor. It is an 8-bit microprocessor designed by Intel in 1977 using NMOS technology.

It has the following configuration –

- 8-bit data bus
- 16-bit address bus, which can address upto 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 – Functional Units

8085 consists of the following functional units –

#### 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.

#### Arithmetic and logic unit

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

### 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.

### 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.

### Stack pointer

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

### Temporary register

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

### 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 –

### 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.

### Timing and control unit

It provides timing and control signal to the microprocessor to perform operations. Following are the timing and control signals, which control external and internal circuits

–

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

### 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.

### Serial Input/output control

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

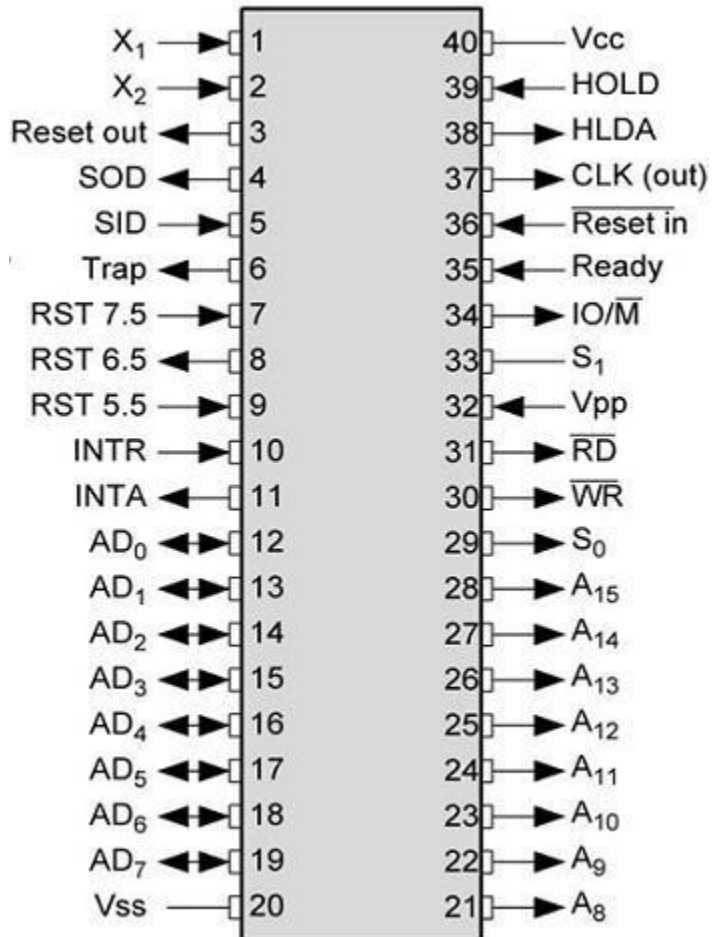
### Address buffer and address-data buffer

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.

### 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.

## 1.5 Signal Description (Pin diagram) of 8085 Microprocessor



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

#### Address bus

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

#### Data bus

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

#### 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.

- **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.
- **WR** – This signal indicates that the data on the data bus is to be written into a selected memory or IO location.
- **ALE** – 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.

#### Power supply

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

#### 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.

### 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. We will discuss interrupts in detail in interrupts section.

- **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.

### Serial I/O signals

There are 2 serial signals, i.e. SID and SOD and these signals are used for serial communication.

- **SOD (Serial output data line)** – The output SOD is set/reset as specified by the SIM instruction.
- **SID (Serial input data line)** – The data on this line is loaded into accumulator whenever a RIM instruction is executed.

## 1.6:- Register Organizations, Distinguish between SPR & GPR , Timing & Control Module



**(a) General Purpose Registers** – The 8085 has six general-purpose registers to store 8-bit data; these are identified as- B, C, D, E, H, and L. These can be combined as register pairs – BC, DE, and HL, to perform some 16-bit operation. These registers are used to store or copy temporary data, by using instructions, during the execution of the program.

**(b) Specific Purpose Registers –**

- **Accumulator:**

The accumulator is an 8-bit register (can store 8-bit data) that is the part of the arithmetic and logical unit (ALU). After performing arithmetical or logical operations, the result is stored in accumulator. Accumulator is also defined as register A.

- **Flag registers:**



fig(a)-Bit position of various flags in flag registers of 8085

The flag register is a special purpose register and it is completely different from other registers in microprocessor. It consists of 8 bits and only 5 of them are useful. The other three are left vacant and are used in the future Intel versions. These 5 flags are set or reset (when value of flag is 1, then it is said to be set and when value is 0, then it is said to be reset) after an operation according to data condition of the result in the accumulator and other registers. The 5 flag registers are:

1. **Sign Flag:** It occupies the seventh bit of the flag register, which is also known as the most significant bit. It helps the programmer to know whether the number stored in the accumulator is positive or negative. If the sign flag is set, it means that number stored in the accumulator is negative, and if reset, then the number is positive.
2. **Zero Flag:** It occupies the sixth bit of the flag register. It is set, when the operation performed in the ALU results in zero(all 8 bits are zero), otherwise it is reset. It helps in determining if two numbers are equal or not.
3. **Auxillary Carry Flag:** It occupies the fourth bit of the flag register. In an arithmetic operation, when a carry flag is generated by the third bit and passed on to the fourth bit, then Auxillary Carry flag is set. If not flag is reset. This flag is used internally for BCD(Binary-Coded decimal Number) operations.  
**Note** – This is the only flag register in 8085 which is not accessible by user.
4. **Parity Flag:** It occupies the second bit of the flag register. This flag tests for number of 1's in the accumulator. If the accumulator holds even number of 1's, then this flag is set and it is said to even parity. On the other hand if the number of 1's is odd, then it is reset and it is said to be odd parity.

5. **Carry Flag:** It occupies the zeroth bit of the flag register. If the arithmetic operation results in a carry (if result is more than 8 bit), then Carry Flag is set; otherwise it is reset.

### **(c) Memory Registers –**

There are two 16-bit registers used to hold memory addresses. The size of these registers is 16 bits because the memory addresses are 16 bits. They are :-

- **Program Counter:** This register is used 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
- **Sign Flag (7th bit):** It is reset(0), which means number stored in the accumulator is positive.
- **Zero Flag (6th bit):** It is reset(0), thus result of the operations performed in the ALU is non-zero.
- **Auxiliary Carry Flag (4th bit):** We can see that b3 generates a carry which is taken by b4, thus auxiliary carry flag gets set (1).
- **Parity Flag (2nd bit):** It is reset(0), it means that parity is odd. The accumulator holds odd number of 1's.
- **Carry Flag (0th bit):** It is set(1), output results in more than 8 bit.
- 

## **Distinguish between SPR & GPR**

### **Segment Registers:**

- **Segments are specific areas clear in a program for containing data, code and stack.**
- **There are 3 main segments – Code Segment – It contains all the instructions to be executed. A 16-bit Code Segment register or CS register supplies the starting address of the code segment.**

### **General purpose registers:**

- **General purpose registers are used to store momentary data within the microprocessor.**
- **It is of sixteen bits and is divided into two eight-bit registers**

### **1.7 Stack, Stack pointer & Stack top.**

A stack (also called a pushdown stack) operates in a last-in/first-out sense. When a new data item is entered or "pushed" onto the top of a stack, the stack pointer increments to the next physical memory address, and the new item is copied to that address. When a data item is "pulled" or "popped" from the top of a stack, the item is copied from the address of the stack pointer, and the stack pointer decrements to the next available item at the top of the stack

A stack pointer is a small [register](#) that stores the address of the last program request in a [stack](#). A stack is a specialized [buffer](#) which stores data from the top down. As new requests come in, they "push down" the older ones. The most recently entered request always resides at the top of the stack, and the program always takes requests from the top.

## **1.8 Interrupts:-8085 Interrupts, Masking of Interrupt(SIM,RIM)**

In 8085 Instruction set, **SIM** (Set Interrupt Mask) and **RIM** (Read Interrupt Mask) instructions can perform mask and unmask RST7.5, RST6.5, and RST5.5 interrupt pins and can also read their status.

In 8085 Instruction set, **SIM** stands for “Set Interrupt Mask”. It is 1-Byte instruction and it is a multi-purpose instruction. The main uses of **SIM** instruction are –

- Masking/unmasking of RST7.5, RST6.5, and RST5.5
- Reset to 0 RST7.5 flip-flop
- Perform serial output of data

When SIM instruction is executed then the content of the Accumulator decides the action to be taken. So before executing the SIM instruction, it is mandatory to initialize Accumulator with the required value. The meaning and purpose of the various bits of the accumulator when SIM is executed has been depicted below –

**Unit-2: Instruction Set and Assembly Language Programming**

## 2.1 Addressing data & Differentiate between one-byte, two-byte & three-byte instructions with examples

## & 2.2 Addressing modes in instructions with suitable examples

The 8085 instruction set is classified into 3 categories by considering the length of the instructions. In 8085, the length is measured in terms of “byte” rather than “word” because 8085 microprocessor has 8-bit data bus. Three types of instruction are: 1-byte instruction, 2-byte instruction, and 3-byte instruction.

### 1. One-byte instructions –

In 1-byte instruction, the opcode and the operand of an instruction are represented in one byte.

- **Example-1:**

Task- Copy the contents of accumulator in register B.

- 

**Mnemonic- MOV B, A**

- Opcode- MOV
- Operand- B, A
- Hex Code- 47H
- Binary code- 0100 0111

- **Example-2:**

Task- Add the contents of accumulator to the contents of register B.

- **Mnemonic- ADD B**

- Opcode- ADD
- Operand- B
- Hex Code- 80H
- Binary code- 1000 0000

- **Example-3:**

Task- Invert (complement) each bit in the accumulator.

- **Mnemonic- CMA**

- Opcode- CMA
- Operand- NA
- Hex Code- 2FH
- Binary code- 0010 1111

**Note** – The length of these instructions is 8-bit; each requires one memory location. The mnemonic is always followed by a letter (or two letters) representing the registers (such as A, B, C, D, E, H, L and SP).

## **2. Two-byte instructions –**

Two-byte instruction is the type of instruction in which the first 8 bits indicates the opcode and the next 8 bits indicates the operand.

- **Example-1:**

Task- Load the hexadecimal data 32H in the accumulator.

- **Mnemonic- MVI A, 32H**

- Opcode- MVI
- Operand- A, 32H
- Hex Code- 3E
- 32
- Binary code- 0011  
1110 0011 0010

- **Example-2:**

Task- Load the hexadecimal data F2H in the register B.

- **Mnemonic- MVI B, F2H**

- Opcode- MVI
- Operand- B, F2H
- Hex Code- 06
- F2
- Binary code- 0000  
0110 1111 0010

**Note** – This type of instructions need two bytes to store the binary codes. The mnemonic is always followed by 8-bit (byte) data.

## **3. Three-byte instructions –**

Three-byte instruction is the type of instruction in which the first 8 bits indicates the opcode and the next two bytes specify the 16-bit address. The low-order address is represented in second byte and the high-order address is represented in the third byte.

- **Example-1:**

Task- Load contents of memory 2050H in the accumulator.

- **Mnemonic- LDA 2050H**

- Opcode- LDA
- Operand- 2050H
- Hex Code- 3A
- 50
- 20

- Binary code- 0011 1010
- 0101 0000
- 0010 0000

- **Example-2:**

Task- Transfer the program sequence to the memory location 2050H.

- **Mnemonic- JMP 2085H**
- Opcode- JMP
- Operand- 2085H
- Hex Code- C3
- 85
- 20
- Binary code- 1100 0011
- 1000 0101
- 0010 0000

**Note** – These instructions would require three memory locations to store the binary codes. The mnemonic is always followed by 16-bit (or adr).

- 

- **Mnemonic- MOV B, A**

- Opcode- MOV
- Operand- B, A
- Hex Code- 47H
- Binary code- 0100 0111

- **Example-2:**

Task- Add the contents of accumulator to the contents of register B.

- **Mnemonic- ADD B**
- Opcode- ADD
- Operand- B
- Hex Code- 80H
- Binary code- 1000 0000

- **Example-3:**

Task- Invert (complement) each bit in the accumulator.

- **Mnemonic- CMA**
- Opcode- CMA
- Operand- NA
- Hex Code- 2FH
- Binary code- 0010 1111

**Note** – The length of these instructions is 8-bit; each requires one memory location. The mnemonic is always followed by a letter (or two letters) representing the registers (such as A, B, C, D, E, H, L and SP).

## 2. Two-byte instructions –

Two-byte instruction is the type of instruction in which the first 8 bits indicates the opcode and the next 8 bits indicates the operand.

- **Example-1:**

Task- Load the hexadecimal data 32H in the accumulator.

- **Mnemonic- MVI A, 32H**

- Opcode- MVI
- Operand- A, 32H
- Hex Code- 3E
- 32
- Binary code- 0011  
1110 0011 0010

- **Example-2:**

Task- Load the hexadecimal data F2H in the register B.

- **Mnemonic- MVI B, F2H**

- Opcode- MVI
- Operand- B, F2H
- Hex Code- 06
- F2
- Binary code- 0000  
0110 1111 0010

**Note –** This type of instructions need two bytes to store the binary codes. The mnemonic is always followed by 8-bit (byte) data.

## 3. Three-byte instructions –

Three-byte instruction is the type of instruction in which the first 8 bits indicates the opcode and the next two bytes specify the 16-bit address. The low-order address is represented in second byte and the high-order address is represented in the third byte.

- **Example-1:**

Task- Load contents of memory 2050H in the accumulator.

- **Mnemonic- LDA 2050H**

- Opcode- LDA
- Operand- 2050H
- Hex Code- 3A
- 50
- 20
- Binary code- 0011 1010  
0101 0000  
0010 0000



- **Example-2:**

Task- Transfer the program sequence to the memory location 2050H.

- **Mnemonic- JMP 2085H**
- Opcode- JMP
- Operand- 2085H
- Hex Code- C3
- 85
- 20
- Binary code- 1100 0011
- 1000 0101
- 0010 0000

**Note** – These instructions would require three memory locations to store the binary codes. The mnemonic is always followed by 16-bit (or adr).

### 2.3 Instruction Set of 8085(Data Transfer, Arithmetic, Logical, Branching, Stack& I/O , Machine Control)

#### Data transfer instructions in 8085 microprocessor

Data transfer instructions are the instructions which transfers data in the microprocessor. They are also called copy instructions.

Following is the table showing the list of logical instructions:

OPCODE	OPERAND	EXPLANATION	EXAMPLE
MOV	Rd, Rs	Rd = Rs	MOV A, B
MOV	Rd, M	Rd = Mc	MOV A, 2050
MOV	M, Rs	M = Rs	MOV 2050, A
MVI	Rd, 8-bit data	Rd = 8-bit data	MVI A, 50
MVI	M, 8-bit data	M = 8-bit data	MVI 2050, 50
OPCODE	OPERAND	EXPLANATION	EXAMPLE

LDA	16-bit address	A = contents at address	LDA 2050
STA	16-bit address	contents at address = A	STA 2050
LHLD	16-bit address	directly loads at H & L registers	LHLD 2050
SHLD	16-bit address	directly stores from H & L registers	SHLD 2050
LXI	r.p., 16-bit data	loads the specified register pair with data	LXI H, 3050
LDAX	r.p.	indirectly loads at the accumulator A	LDAX H
STAX	16-bit address	indirectly stores from the accumulator A	STAX 2050
XCHG	none	exchanges H with D, and L with E	XCHG
PUSH	r.p.	pushes r.p. to the stack	PUSH H
POP	r.p.	pops the stack to r.p.	POP H
IN	8-bit port address	inputs contents of the specified port to A	IN 15
OUT	8-bit port address	outputs contents of A to the specified port	OUT 15

Following is the table showing the list of Arithmetic instructions with their meanings.

Opcode	Operand	Meaning	Explanation
--------	---------	---------	-------------

ADD	R M	Add register or memory to the accumulator	The contents of the register or memory are added to the contents of the accumulator and the result is stored in the accumulator. <b>Example</b> – ADD K.
ADC	R M	Add register to the accumulator with carry	The contents of the register or memory & M the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. <b>Example</b> – ADC K
ADI	8-bit data	Add the immediate to the accumulator	The 8-bit data is added to the contents of the accumulator and the result is stored in the accumulator. <b>Example</b> – ADI 55K
ACI	8-bit data	Add the immediate to the accumulator with carry	The 8-bit data and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. <b>Example</b> – ACI 55K
LXI	Reg. pair, 16bit data	Load the register pair	The instruction stores 16-bit data into the pair designated register in the operand.

		immediate	<b>Example</b> – LXI K, 3025M
DAD	Reg. pair	Add the register pair to H and L registers	The 16-bit data of the specified register pair are added to the contents of the HL register. <b>Example</b> – DAD K
SUB	R M	Subtract the register or memory from the accumulator	The contents of the register or the memory are subtracted from the contents of the accumulator, and the result is stored in the accumulator. <b>Example</b> – SUB K

SBB	R  M	Subtract the source and borrow from the accumulator	The contents of the register or the memory & M the Borrow flag are subtracted from the contents of the accumulator and the result is placed in the accumulator. <b>Example –</b> SBB K
SUI	8-bit data	Subtract the immediate from the accumulator	The 8-bit data is subtracted from the contents of the accumulator & the result is stored in the accumulator. <b>Example –</b> SUI 55K
SBI	8-bit data	Subtract the immediate from the accumulator with borrow	The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E. <b>Example –</b> XCHG
INR	R  M	Increment the register or memory by 1	The contents of the designated register or memory are incremented by 1 and their result is stored at the same place. <b>Example –</b> INR K

INX	R	Increment register pair by 1	The contents of the designated register pair are incremented by 1 and their result is stored at the same place. <b>Example</b> – INX K
DCR	R M	Decrement the register or memory by 1	The contents of the designated register or memory are decremented by 1 and their result is stored at the same place. <b>Example</b> – DCR K
DCX	R	Decrement the register pair by 1	The contents of the designated register pair are decremented by 1 and their result is stored at the same place. <b>Example</b> – DCX K

DAA	None	Decimal adjust accumulator	<p>The contents of the accumulator are changed from a binary value to two 4-bit BCD digits.</p> <p>If the value of the low-order 4-bits in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low-order four bits.</p> <p>If the value of the high-order 4-bits in the accumulator is greater than 9 or if the Carry flag is set, the instruction adds 6 to the high-order four bits.</p>
-----	------	----------------------------	--

**Example – DAA**

### Logical instructions in 8085 microprocessor

Logical instructions are the instructions which perform basic logical operations such as AND, OR, etc. In 8085 microprocessor, the destination operand is always the accumulator. Here logical operation works on a bitwise level.

Following is the table showing the list of logical instructions:

OPCODE	OPERAND	DESTINATION	EXAMPLE
ANA	R	A=A AND R	ANA B

OPCODE	OPERAND	DESTINATION	EXAMPLE
ANI	8-bit data	A = A AND 8-bit data	ANI 50
ORA	R	A=AORR	ORA B
ORA	M	A = A OR Mc	ORA 2050
ORI	8-bit data	A = A OR 8-bit data	ORI 50
XRA	R	A=AXORR	XRA B
XRA	M	A = A XOR Mc	XRA 2050
XRI	8-bit data	A = A XOR 8-bit data	XRI 50
CMA	none	A = 1's compliment of A	CMA
CMP	R	Compares R with A and triggers the flag register	CMP B
CMP	M	Compares Mc with A and triggers the flag register	CMP 2050
CPI	8-bit data	Compares 8-bit data with A and triggers the flag register	CPI 50
RRC	none	Rotate accumulator right without carry	RRC
RLC	none	Rotate accumulator left without carry	RLC



OPCODE	OPERAND	DESTINATION	EXAMPLE
RAR	none	Rotate accumulator right with carry	RAR
RAL	none	Rotate accumulator left with carry	RAR
CMC	none	Compliments the carry flag	CMC
STC	none	Sets the carry flag	STC

## Branching instructions in 8085 microprocessor

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:

1. Jump (unconditional and conditional)
2. Call (unconditional and conditional)
3. Return (unconditional and conditional)

**1. Jump Instructions** – The jump instruction transfers the program sequence to the memory address given in the operand based on the specified flag. Jump instructions are 2 types: Unconditional Jump Instructions and Conditional Jump Instructions.

**(a) Unconditional Jump Instructions:** Transfers the program sequence to the described memory address.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
JMP	address	Jumps to the address	JMP 2050

**(b) Conditional Jump Instructions:** Transfers the program sequence to the described memory address only if the condition is satisfied.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
JC	address	Jumps to the address if carry flag is 1	JC 2050
JNC	address	Jumps to the address if carry flag is 0	JNC 2050
JZ	address	Jumps to the address if zero flag is 1	JZ 2050
JNZ	address	Jumps to the address if zero flag is 0	JNZ 2050
JPE	address	Jumps to the address if parity flag is 1	JPE 2050
JPO	address	Jumps to the address if parity flag is 0	JPO 2050
JM	address	Jumps to the address if sign flag is 1	JM 2050
JP	address	Jumps to the address if sign flag 0	JP 2050

**2. Call Instructions –** The call instruction transfers the program sequence to the memory address given in the operand. Before transferring, the address of the next instruction after CALL is pushed onto the stack. Call instructions are 2 types: Unconditional Call Instructions and Conditional Call Instructions.

**(a) Unconditional Call Instructions:** It transfers the program sequence to the memory address given in the operand.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
--------	---------	-------------	---------

CALL	address	Unconditionally calls	CALL 2050
------	---------	-----------------------	-----------

**(b) Conditional Call Instructions:** Only if the condition is satisfied, the instructions executes.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
--------	---------	-------------	---------

CC	address	Call if carry flag is 1	CC 2050
----	---------	-------------------------	---------

CNC	address	Call if carry flag is 0	CNC 2050
-----	---------	-------------------------	----------

CZ	address	Calls if zero flag is 1	CZ 2050
----	---------	-------------------------	---------

CNZ	address	Calls if zero flag is 0	CNZ 2050
-----	---------	-------------------------	----------

CPE	address	Calls if parity flag is 1	CPE 2050
-----	---------	---------------------------	----------

CPO	address	Calls if parity flag is 0	CPO 2050
-----	---------	---------------------------	----------

CM	address	Calls if sign flag is 1	CM 2050
----	---------	-------------------------	---------

CP	address	Calls if sign flag is 0	CP 2050
----	---------	-------------------------	---------

**3. Return Instructions** – The return instruction transfers the program sequence from the subroutine to the calling program. Jump instructions are 2 types: Unconditional Jump Instructions and Conditional Jump Instructions.

**(a) Unconditional Return Instruction:** The program sequence is transferred unconditionally from the subroutine to the calling program.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
--------	---------	-------------	---------

RET	none	Return from the subroutine unconditionally	RET
-----	------	--	-----

**(b) Conditional Return Instruction:** The program sequence is transferred unconditionally from the subroutine to the calling program only if the condition is satisfied.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
--------	---------	-------------	---------

RC	none	Return from the subroutine if carry flag is 1	RC
----	------	---	----

RNC	none	Return from the subroutine if carry flag is 0	RNC
-----	------	---	-----

RZ	none	Return from the subroutine if zero flag is 1	RZ
----	------	--	----

RNZ	none	Return from the subroutine if zero flag is 0	RNZ
-----	------	--	-----

RPE	none	Return from the subroutine if parity flag is 1	RPE
-----	------	--	-----

RPO	none	Return from the subroutine if parity flag is 0	RPO
-----	------	--	-----

RM	none	Returns from the subroutine if sign flag is 1	RM
----	------	---	----

RP	none	Returns from the subroutine if sign flag is 0	RP
----	------	---	----

## Stack I-O and Machine Control Instructions

The following instructions affect the Stack and/or Stack Pointer:

PUSH	Push Two bytes of Data onto the Stack
POP	Pop Two Bytes of Data off the Stack
XTHL	Exchange Top of Stack with H & L
SPHL	Move content of H & L to Stack Pointer

The I/O instructions are as follows:

IN	Initiate Input Operation
OUT	Initiate Output Operation

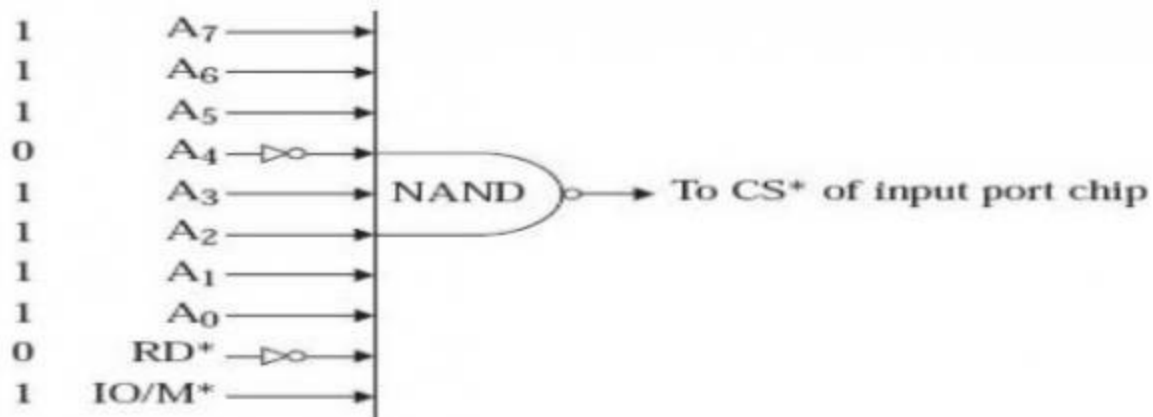
The Machine Control instructions are as follows:

EI	Enable Interrupt System
DI	Disable Interrupt System
HLT	Halt
NOP	No Operation

## 2.5 Memory & I/O Addressing,

It is possible to address an I/O port as if it were a memory location. For example, let us say, the chip select pin of an I/O port chip is activated when address = FFF0H, IO/M\* = 0, and RD\* = 0. This is shown in the following fig.

In this case, the I/O port chip is selected when the 8085 is thinking that it is addressing memory location FFF0H for a read operation. Note that 8085 thinks that it is addressing a memory location because it has sent out IO/M\* as a logic 0. But in reality, an input port has been selected, and the input port supplies information to the 8085. Such I/O ports that are addressed by the processor as if they were memory locations are called memory-mapped I/O ports.



In the memory location we address an Input Output port. An example to be cited as when address = FFF0H, IO/M\* = 0, and RD\* = 0. Here we select the Input Output port chip when 8085 microprocessor finds that it is memory allocated location as it is sent out like IO/M\* as a logic 0.

But in real world we select an Input Port which supplies information to 8085 Microprocessor. Like the memory locations 8085 microprocessor gets addressed by the processor which are called memory-mapped Input Output ports.

There is a set of instructions for this memory-mapped I/O operations. E.g. STA, LDA etc. Let us discuss STA instruction in detail for better understanding.

Register A is an 8-bit register used in 8085 to perform arithmetic, logical, I/O & LOAD/STORE operations. Register A is quite often called as an Accumulator. An accumulator is a register for short-term, intermediate storage of arithmetic and logic data in a computer's CPU (Central Processing Unit). In an arithmetic operation involving two operands, one operand has to be in this register. And the result of the arithmetic operation will be stored or accumulated in this register. Similarly, in a logical operation involving two operands, one operand has to be in the accumulator. Also, some other operations, like complementing and decimal adjustment, can be performed only on the accumulator.

Let us now consider a program segment which involves content of Accumulator only. In 8085 Instruction set, **STA** is a mnemonic that stands for STore Accumulator contents in memory. In this instruction, Accumulator 8-bit content will be stored to a memory location whose 16-bit address is indicated in the instruction as a16. This instruction uses absolute addressing for specifying the destination. This instruction occupies 3-Bytes of memory. First Byte is required for the opcode, and next successive 2-Bytes provide the 16-bit address divided into 8-bits each consecutively.

Mnemonics, Operand	Opcode (in HEX)	Bytes
STA Address	32	3

Let us consider **STA 4050H** as an example instruction of this type. It is a 3-Byte instruction. The first Byte will contain the opcode hex value 32H. As in 8085 assembly language coding supports low order Byte of the address should be mentioned at first then the high order Byte of the address should be mentioned next. So next Byte in memory will hold 50H and after that 40H will be kept in the last third Byte. Let us suppose the initial content of Accumulator is ABH and initial content of memory location 4050H is CDH. So after execution, Accumulator content will remain as ABH and 4050H location's content will become ABH replacing its previous content CDH. The content tracing of this instruction has been shown below –

	Before	After
(A)	ABH	ABH
(4050H)	CDH	ABH

The content tracing of this instruction has been shown below

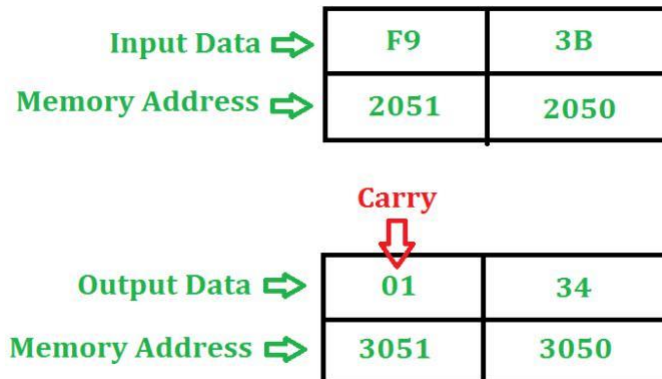
Address	Hex Codes	Mnemonic	Comment
2008	2A	STA 4050H	Content of the memory location 4050H A
2009	50		Low order Byte of the address
200A	40		High order Byte of the address

## 2.4 Simple Assembly Language Programming of 8085

## 8085 program to add two 8 bit numbers

**Problem –** Write an assembly language program to add two 8 bit numbers stored at address 2050 and address 2051 in 8085 microprocessor. The starting address of the program is taken as 2000.

**Example –**



**Algorithm –**

1. Load the first number from memory location 2050 to accumulator.
2. Move the content of accumulator to register H.
3. Load the second number from memory location 2051 to accumulator.
4. Then add the content of register H and accumulator using “ADD” instruction and storing result at 3050
5. The carry generated is recovered using “ADC” command and is stored at memory location 3051

**Program –**

MEMORY ADDRESS	MNEMONICS	COMMENT
2000	LDA 2050	A<-[2050]
2003	MOV H, A	H<-A
2004	LDA 2051	A<-[2051]



MEMORY ADDRESS	MNEMONICS	COMMENT
2007	ADD H	$A \leftarrow A + H$
2006	MOV L, A	$L \leftarrow A$
2007	MVI A 00	$A \leftarrow 00$
2009	ADC A	$A \leftarrow A + A + \text{carry}$
200A	MOV H, A	$H \leftarrow A$
200B	SHLD 3050	$H \rightarrow 3051, L \rightarrow 3050$
200E	HLT	

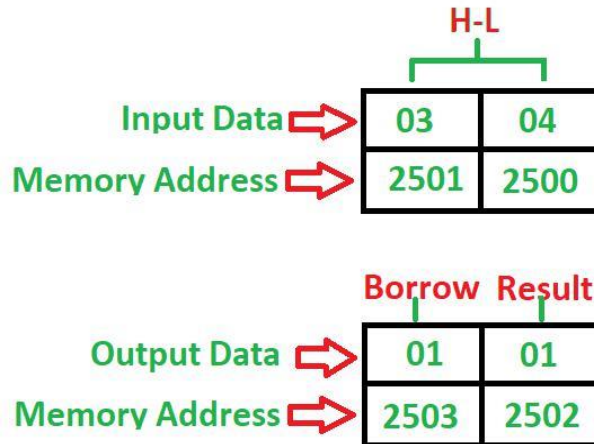
#### Explanation –

1. **LDA 2050** moves the contents of 2050 memory location to the accumulator.
2. **MOV H, A** copies contents of Accumulator to register H to A
3. **LDA 2051** moves the contents of 2051 memory location to the accumulator.
4. **ADD H** adds contents of A (Accumulator) and H register (F9). The result is stored in A itself. **For all arithmetic instructions A is by default an operand and A stores the result as well**
5. **MOV L, A** copies contents of A (34) to L
6. **MVI A 00** moves immediate data (i.e., 00) to A
7. **ADC A** adds contents of A(00), contents of register specified (i.e A) and carry (1). As ADC is also an arithmetic operation, A is by default an operand and A stores the result as well
8. **MOV H, A** copies contents of A (01) to H
9. **SHLD 3050** moves the contents of L register (34) in 3050 memory location and contents of H register (01) in 3051 memory location
10. **HLT** stops executing the program and halts any further execution

## 8085 program to subtract two 8-bit numbers with or without borrow

**Problem** – Write a program to subtract two 8-bit numbers with or without borrow where first number is at **2500** memory address and second number is at **2501** memory address and store the result into **2502** and borrow into **2503** memory address.

**Example** –



**Algorithm** –

1. Load 00 in a register C (for borrow)
2. Load two 8-bit number from memory into registers
3. Move one number to accumulator
4. Subtract the second number with accumulator
5. If borrow is not equal to 1, go to step 7
6. Increment register for borrow by 1
7. Store accumulator content in memory
8. Move content of register into accumulator
9. Store content of accumulator in other memory location
10. Stop

**Program** –

MEMORY	MNEMONICS	OPERANDS	COMMENT
2000	MVI	C, 00	[C] <- 00
2002	LHLD	2500	[H-L] <- [2500]

MEMORY	MNEMONICS	OPERANDS	COMMENT
2005	MOV	A, H	[A] <- [H]
2006	SUB	L	[A] <- [A] – [L]
2007	JNC	200B	Jump If no borrow
200A	INR	C	[C] <- [C] + 1
200B	STA	2502	[A] -> [2502], Result
200E	MOV	A, C	[A] <- [C]
2010	STA	2503	[A] -> [2503], Borrow
2013	HLT		Stop

**Explanation** – Registers A, H, L, C are used for general purpose:

1. **MOV** is used to transfer the data from memory to accumulator (1 Byte)
2. **LHLD** is used to load register pair directly using 16-bit address (3 Byte instruction)
3. **MVI** is used to move data immediately into any of registers (2 Byte)
4. **STA** is used to store the content of accumulator into memory(3 Byte instruction)
5. **INR** is used to increase register by 1 (1 Byte instruction)
6. **JNC** is used to jump if no borrow (3 Byte instruction)
7. **SUB** is used to subtract two numbers where one number is in accumulator(1 Byte)
8. **HLT** is used to halt the program.

# Machine cycles & Timing Diagrams Chapter - 3

## Instruction cycle

This is the timing required by the  $\mu p$  to fetch & execute one complete instruction. The instruction cycle is in two parts:

1. Fetch Cycle
2. Execute cycle

## Fetch Cycle

→ This is the time required by the  $\mu p$  to fetch all bytes of an instruction.

→ The length of the fetch cycle is thus determined by the no. of bytes in an instruction.

## Execution cycle

This is the time required by the  $\mu p$  to execute a fetched instruction.

## Machine cycle

It is the time required by the  $\mu p$  doing one operation & accessing one byte from the external module (memory/I/O)

## Machine cycle of 8085

The machine cycle of 8085 are given below:-

Name	IO/ $\bar{M}$	RD	WR	SI	SO	INTA	T-states
Opcode fetch	0	0	1	1	1	1	4/6
Mem Read	0	1	0	1	0	1	3
Mem Write	0	0	1	0	1	1	3
I/O Read	1	1	0	1	0	1	3
I/O Write	1	0	1	0	1	1	3
INTA	1	1	1	1	1	0	3 or 6
Bus Idle	0	1	1	0	0	1	3

## Opcode fetch Q1 Draw the timing diagram for opcode fetch.

- This cycle is used to fetch the opcode from the memory.
- This is the first machine cycle of every instruction.
- It is a compulsory machine cycle.
- It is generally a 4T-state but some instructions require a 6T-state opcode fetch.

### During T1

- A15-A8 contains the higher byte of the address (PC+1).
- A8-ALE is high AD7-AD0 contains the lower byte of the address (PCL).
- Since it is an opcode fetch cycle,  $\overline{SI}$  &  $\overline{SO}$  go high.
- Since it is a memory operation,  $\overline{IO}/\overline{M}$  goes low.

### During T2

- A8-ALE goes low address is removed from AD7-AD0.
- A8-RD goes low, data appears on AD7-AD0.
- The  $\mu p$  examines the state of READY pin. If it is low then the  $\mu p$  enters wait-state by executing wait cycles. It remains in the wait-state until READY goes high.

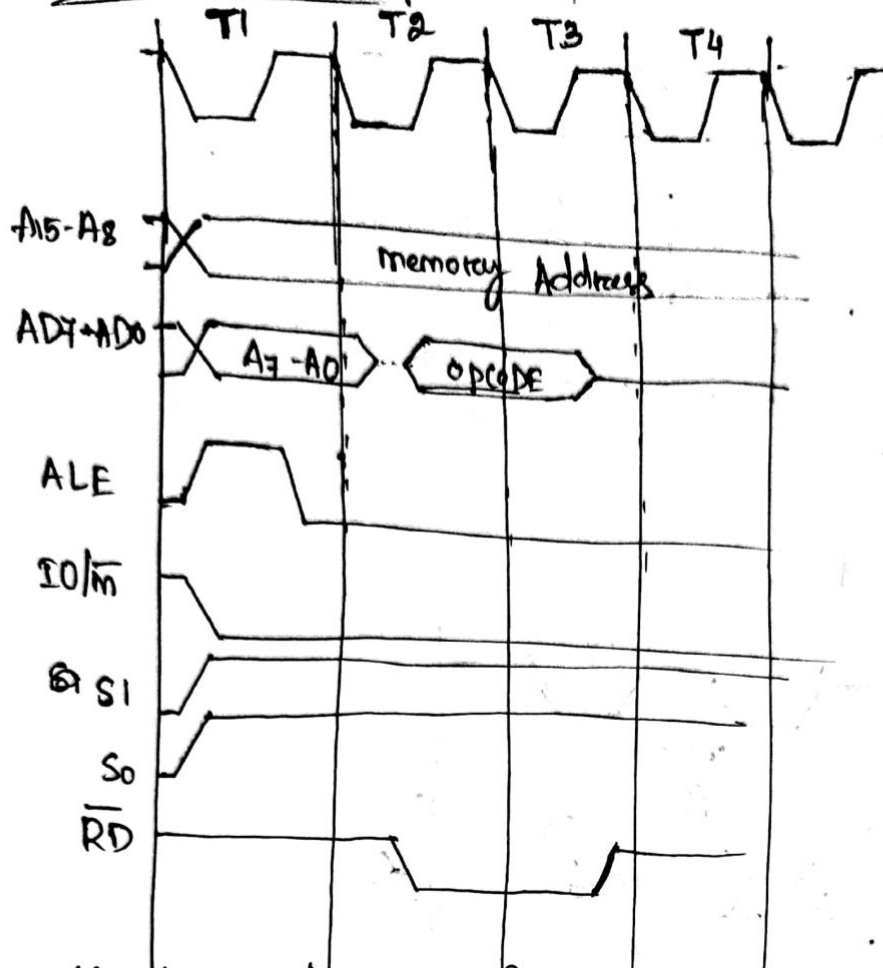
### During T3

- Data remains on AD7-AD0 till RD is low.

### During T4 :

- T4 state is used by the  $\mu p$  to decode the opcode.

## OPCODE FETCH CYCLE



Q11 Draw the timing diagram for memory read.  
Memory Read

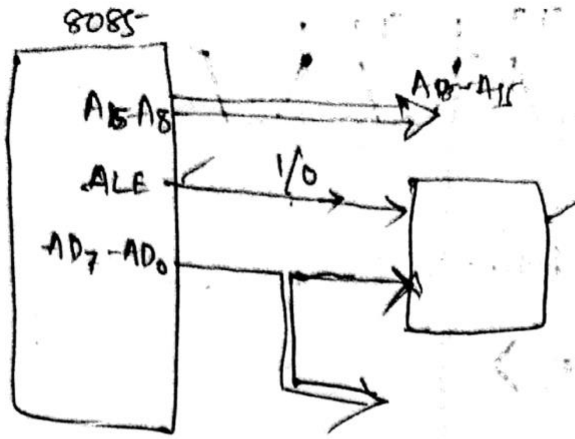
- This cycle is used to fetch one byte from memory.
- This cycle ~~is~~ can be used to fetch the operand bytes of an instruction or any data from the memory.
- It is not compulsory machine cycle.
- It requires 3 T-states.

### During T1

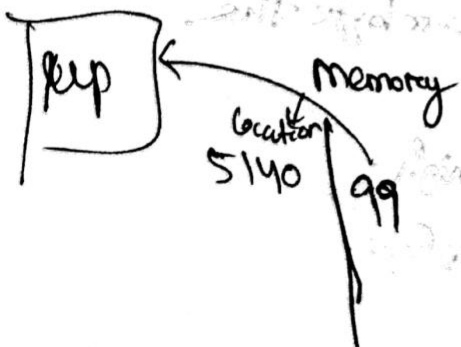
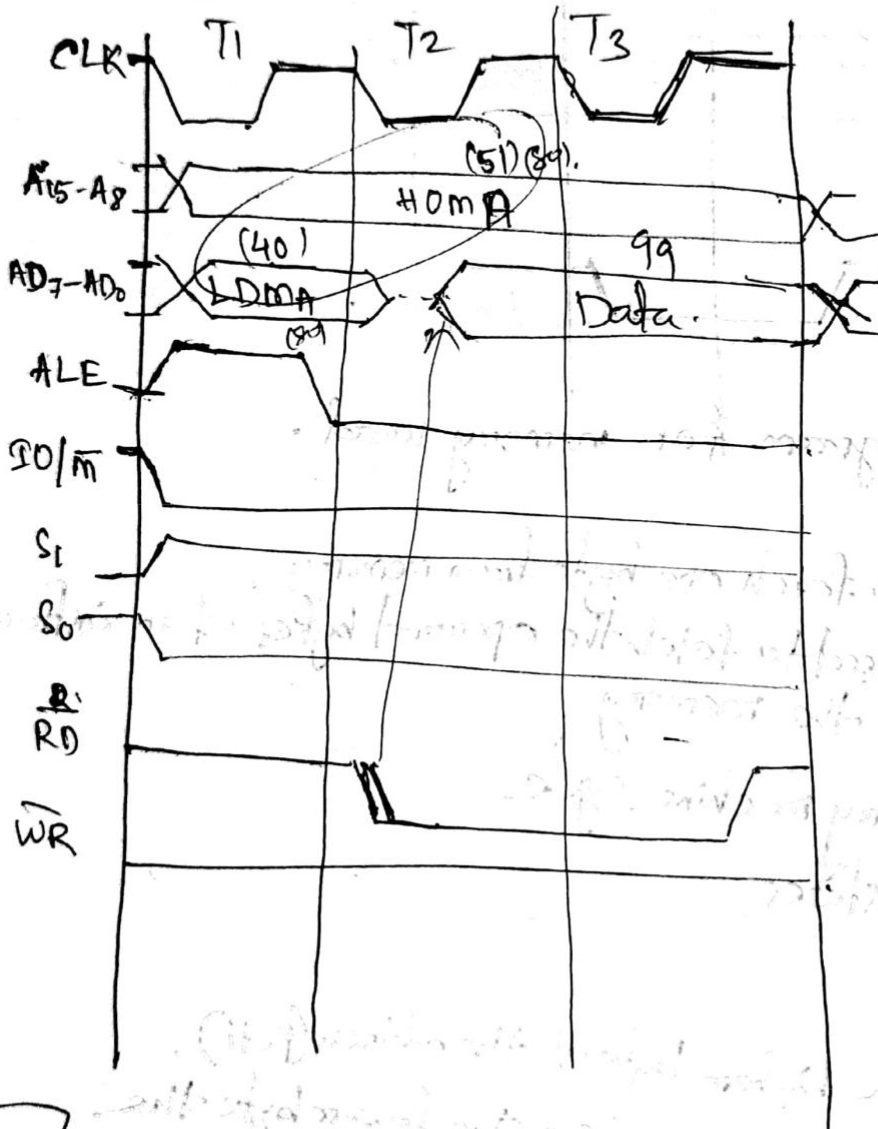
- A15-A8 contains the higher byte of the address (PCH).
- A8 ALE is high, AD7-AD0 contains the lower byte the address (PCL).
- Since it is a memory read cycle, S1 goes high.
- Since, it is a memory operation, IO/M goes low.

### During T2

- ALE goes low



Memory Read



## Memory Write Q1) Draw the timing Diagram for memory write.

- This cycle is used to send (write) one byte into the memory.
- It is not compulsory machine cycle.
- It requires 3T-states.

### During T1

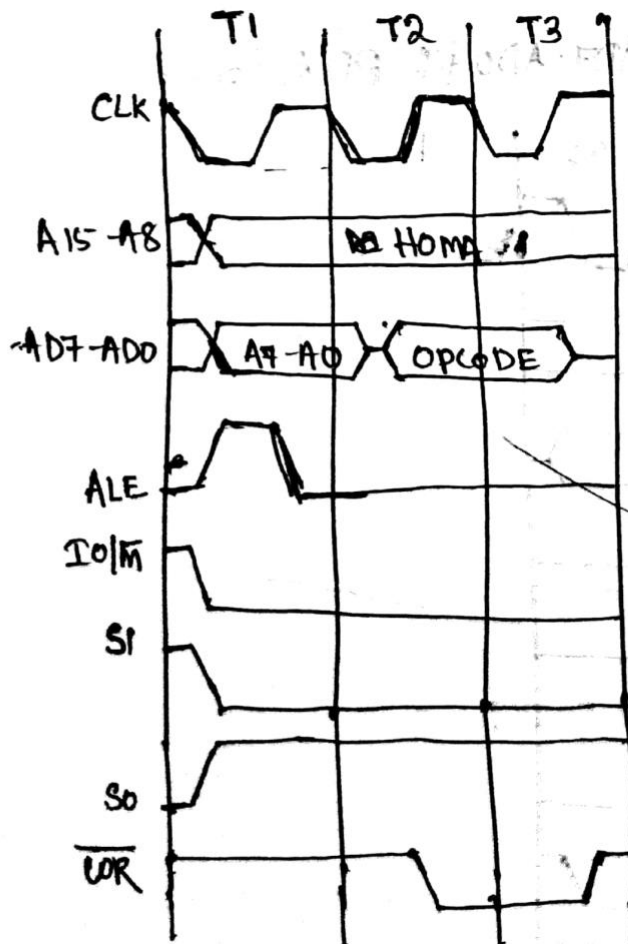
- A15-A8 Contains the higher byte of the address (PCH)
- A8-ALE is high, A7-A0 Contains the lower byte of the address (PCL).
- Since it is a memory write cycle,  $\overline{S_0}$  goes high.
- Since it is a memory operation,  $\overline{IO/\overline{M}}$  goes low.

### During T2

- ALE goes low
- Address is removed from A7-A0.
- Data appears on A7-A0 &  $\overline{WR}$  goes low.

### During T3

- Data remains on A7-A0 till  $\overline{WR}$  is low.



HOMA - Higher Order Memory Address



Q1 Draw the timing diagram for I/O Read.

### I/O Read Machine Cycle

- This cycle is used to fetch one byte from an I/O port.
- I/O read machine cycle is similar to the memory read machine cycle except that IO/M signal is high for I/O read.

#### During T1

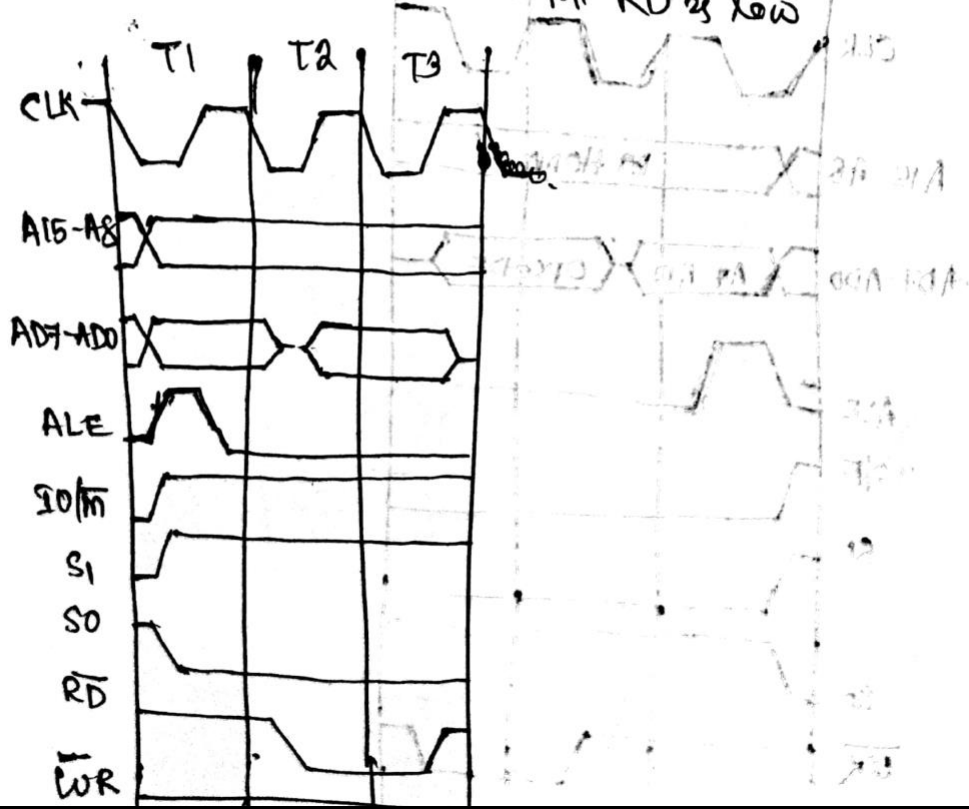
- The lower 8 bits of the I/O port Address are duplicated into the higher order address bus A15-A8.
- As ALE is high AD7-AD0 contains the lower byte of the address.
- Since it is an I/O operation IO/M goes high.

#### During T2

- ALE goes low.
- Address is removed AD7-AD0.
- As RD goes low, Data appears on AD7-AD0.

#### During T3

- Data remains on AD7-AD0 till RD is low.



## I/O write machine cycle // Draw the timing diagram for I/O write.

- This cycle is used to send (write) one byte into an I/O port.
- The I/O write machine cycle is similar to the memory write machine cycle except that  $\overline{IO/\overline{M}}$  is high for I/O write.
- It requires 3T-states.

### During T1

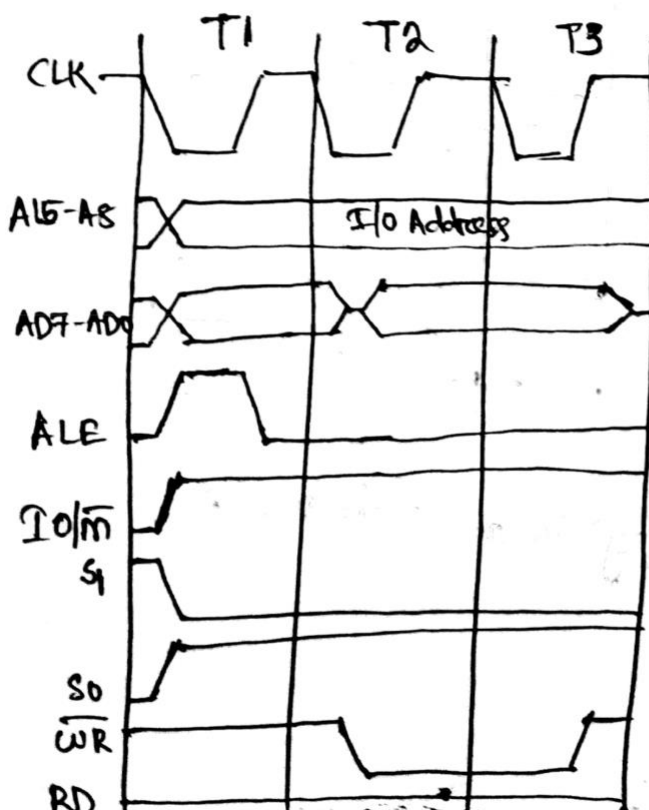
- The lower 8 bits of the I/O port address are duplicated into the higher order address bus A15-A8.
- As ALE is high AD7-AD0 contains the lower byte of the address.
- Since it is an I/O write cycle,  $\overline{SO}$  goes high.
- Since it is I/O operation,  $\overline{IO/\overline{M}}$  goes high.

### During T2

- ALE goes low.
- Address is removed from AD7-AD0.
- Data appears on AD7-AD0 &  $\overline{WR}$  goes low.

### During T3

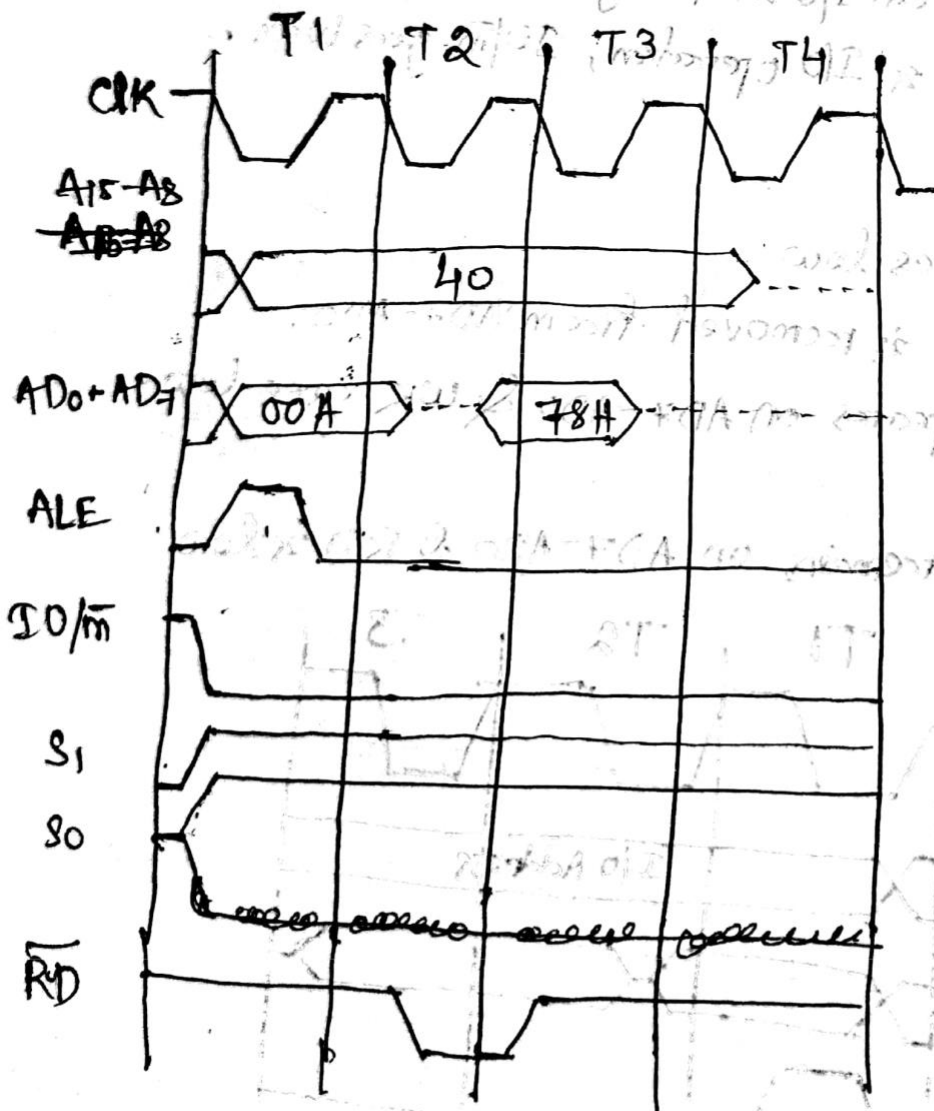
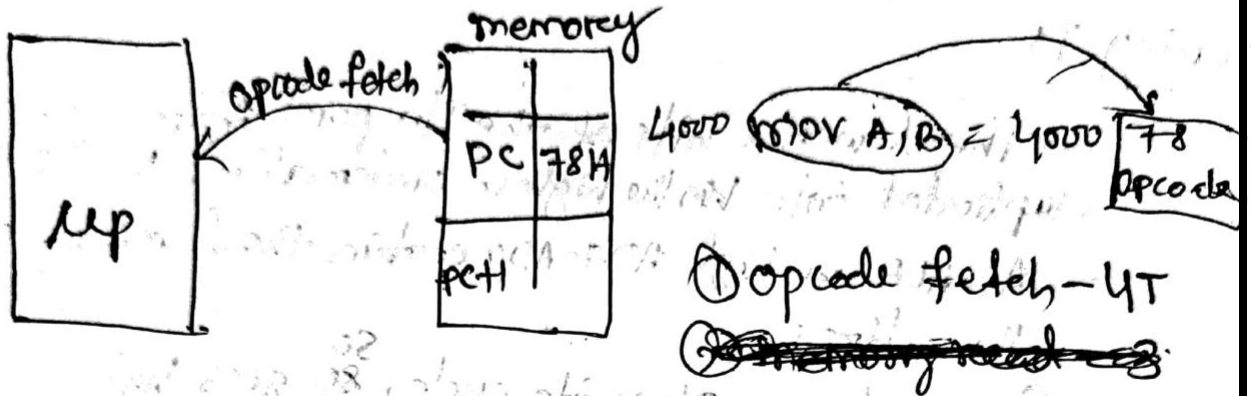
- Data remains on AD7-AD0 &  $\overline{RD}$  is low.



Q11 Draw the timing diagram of one 1 byte instruction.

00 Draw the timing diagram for the instruction MOV A, B

Ans The instruction MOV A, B is a 1 byte instruction.



Timing diagram for MOV A, B

Q1 Draw the timing diagram for the instruction MVI B, 25H.

Draw the timing diagram of a 2 byte instruction.

Soln

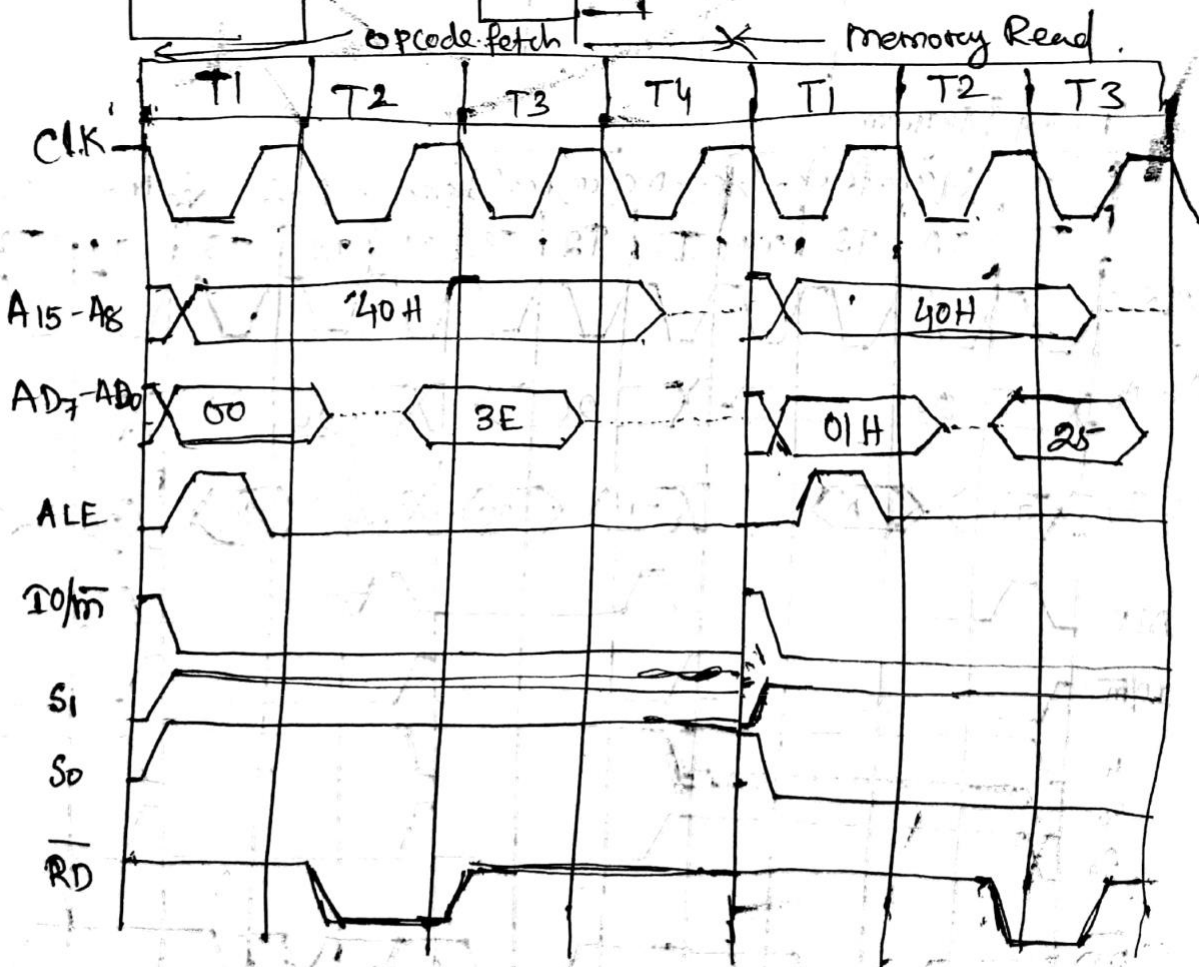
MVI B, 25H is a 2 byte instruction.

Machine cycle	Address Bus	Data Bus	T-states
Opcode fetch	PC	opcode (3E)	4
Memory Read	PC+1	25	3
			Total 7

4000 (MVI B, 25) =  $\begin{matrix} 4000 & 3E \\ 4001 & 25 \end{matrix}$



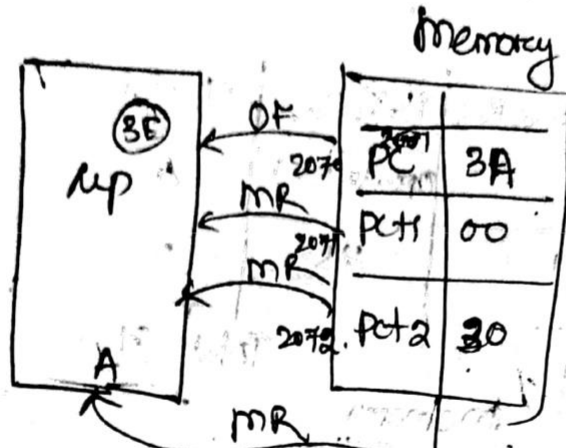
OF = Opcode fetch  
MR = memory read



Timing Diagram for MVI B, 25H

Q1 Draw the timing diagram for LDA 3000H.

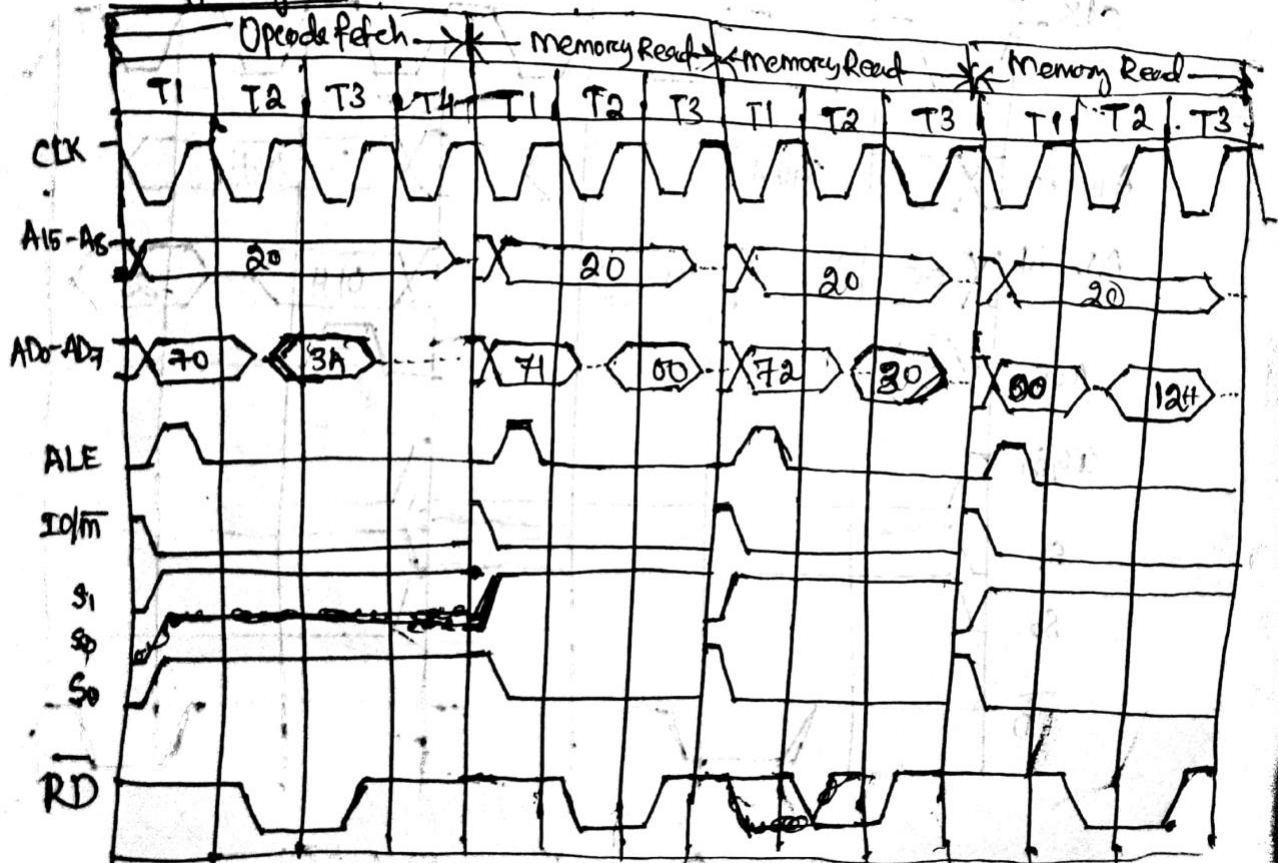
or  
Draw the timing diagram for load instruction.



Execution cycle 2070 2071 2072 2073  
12H

OF	MR	MR	MR
4T	3T	3T	3T
= 13T			

Timing Diagram



Timing Diagram for LDA 3000H