GOVERNMENT POLYTECHNIC, BHUBANESWAR
Government of Odisha
ସରକାରୀ ବହୁବୃତ୍ତି ଅନୁଷ୍ଠାନ, ଭୁବନେଶ୍ୱର

# LECTURE NOTES
# ON
# Microprocessor & Microcontroller

## 4th Semester
## Information Technology
## &
## Electronics & Telecommunication Engg

Prepared by:
P. Bhawani
Lecturer in Electronics & Telecommunication Engineering
Government polytechnic Bhubaneswar

# UNIT-1: MICROPROCESSOR (ARCHITECTURE AND PROGRAMMING-8085-8-BIT)

## 1.1 Introduction to Microprocessor and Microcomputer & distinguish between them.

### Microprocessor

- A microprocessor is a programmable electronics chip that has computing and decision making capabilities similar to central processing unit of a computer.

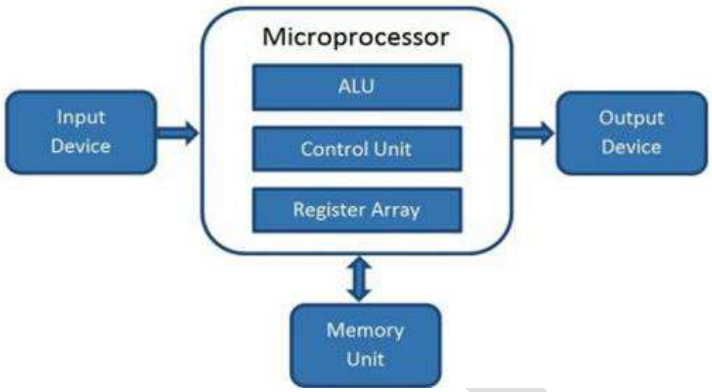
  Or

- Microprocessor is a digital device on a chip that can fetch instructions from memory, decode and execute them and give results.

### Microcomputer

- Microcomputer an electronic device with a microprocessor as its central processing unit (CPU).
- Microcomputers are designed to be used by individuals, whether in the form of PCs, workstations or notebook computers.
- A microcomputer contains a CPU on a microchip (the microprocessor), a memory system (typically ROM and RAM), a bus system and I/O ports, typically housed in a motherboard.

### Block Diagram of a Microcomputer



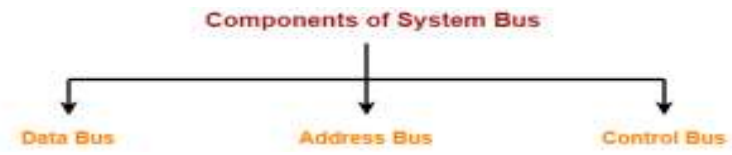

| Microcomputer | Microprocessor |
|---|---|
| Microcomputer is a complete computer similar to any other computer. | Microprocessor is one component of the microcomputer. |
| Examples: Laptops and desktops | Examples: 8085, 8086, Dual Core, etc |

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

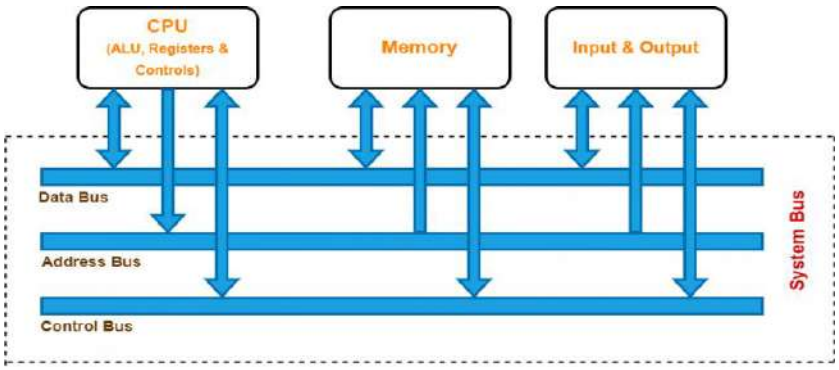
## 1.3 General Bus structure Block diagram.

- A Bus is a collection of wires that connects several devices.
- In computer system all the peripherals are connected to microprocessor through Bus.
- A bus that connects major components (CPU, memory and I/O devices) of a computer system is called as a System Bus.

The system bus consists of three major components-




1. Data Bus
2. Address Bus
3. Control Bus




**1. Address Bus:**

- Address bus is used to carry address from CPU to memory/IO devices.
- It is used to identify the particular location in memory.
- It carries the source or destination address of data i.e. where to store or from where to retrieve the data.
- It is uni-directional.

**2. Data Bus:**

- Data bus is used for transmitting the data / instruction from CPU to memory/IO and vice-versa.
- It is bi-directional.

**3. Control Bus:**

- Control bus is used to transfer the control and timing signals from one component to the other component.
- The CPU uses control bus to communicate with the devices that are connected to the computer system.
- The CPU transmits different types of control signals to the system components.
- It is bi-directional.

Typical control signals hold by control bus-

- Memory read – Data from memory address location to be placed on data bus.
- Memory write – Data from data bus to be placed on memory address location.
- I/O Read – Data from I/O address location to be placed on data bus.
- I/O Write – Data from data bus to be placed on I/O address location.

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

- **8085 Microprocessor** is a programmable electronics chip(IC).
- It is an 8-bit microprocessor that means it operates 8-bit at a time.
- It is used in almost all electronic devices like mobile phones, printers, washing machines, etc., and in advanced applications like radars, satellites, and flights.

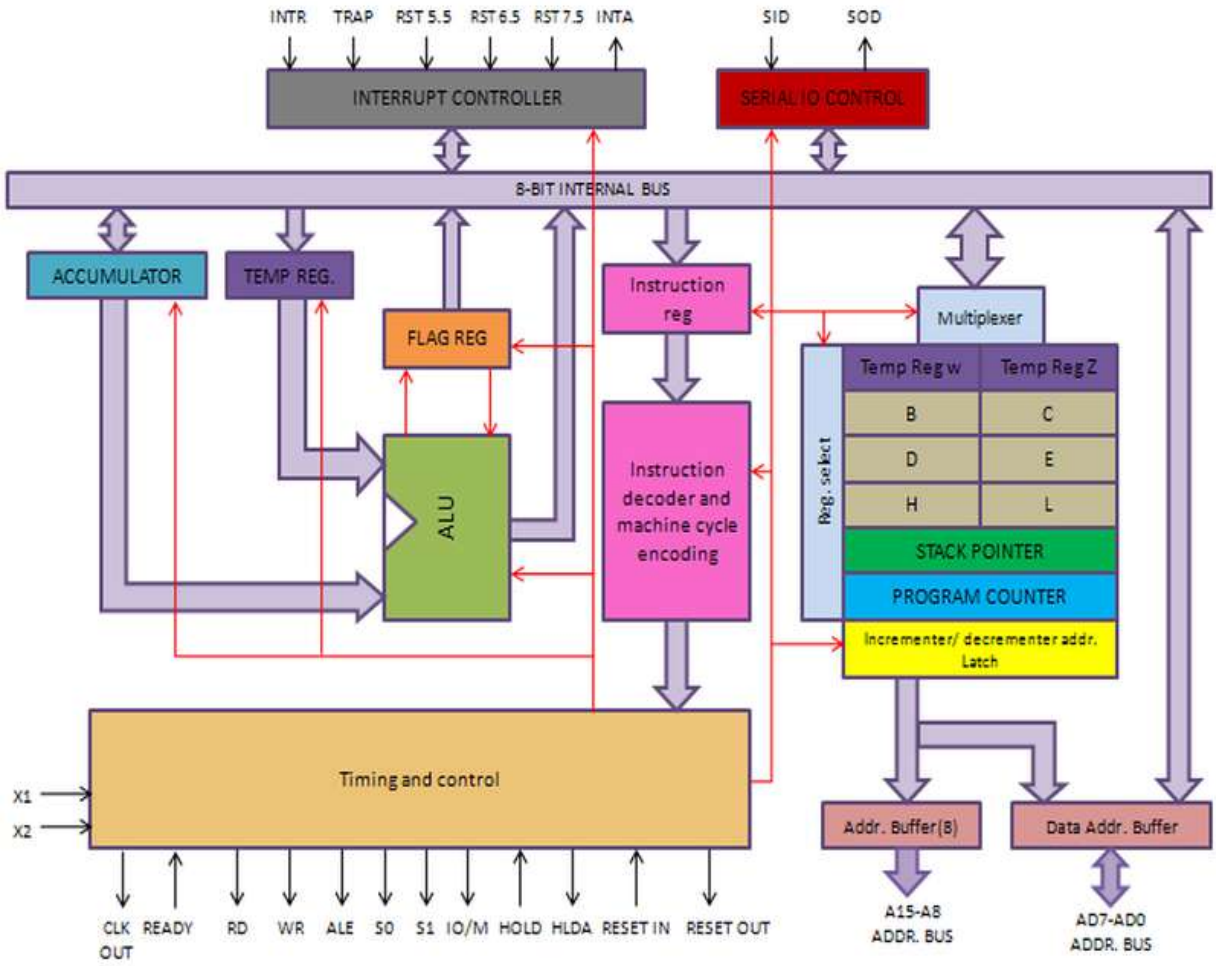
The salient features of 8085 µp are:

- It is an 8-bit processor available as a 40-pin IC package and uses +5 V for power.
- It is manufactured with N-MOS technology and operates at 3.2 MHZ single phase clock.
- Its data bus width is 8-bit and address bus width is 16-bit, thus it can address $2^{16}$ = 64 KB of memory.
- It supports external interrupt request.
- A 16-bit program counter (PC)
- A 16-bit stack pointer (SP)
- Six 8-bit general purpose register arranged in pairs: BC, DE, HL.

### Architecture of 8085 Microprocessor

8085 consists of various units as shown in figure and each unit performs its own functions. The various units of a microprocessor are listed below

- Accumulator
- Arithmetic and logic Unit
- General purpose register
- Program counter
- Stack pointer
- Temporary register
- Flags
- Instruction register and Decoder
- Timing and Control unit
- Interrupt control
- Serial Input/output control
- Address buffer and Address-Data buffer
- Address bus and Data bus




## Accumulator:

- The accumulator is an 8-bit register associated with the ALU. It is used to hold one of the operands of an arithmetic and logical operation.
- The final result of an arithmetic or logical operation is also placed in the accumulator.

## Arithmetic and logic Unit:

- The ALU is responsible for performing arithmetic and logical operations on data.
- It can perform operations such as addition, subtraction, logical AND, logical OR, and more.

## General purpose register:

- 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.

## 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:

- 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 stack.
- Each time when the data is loaded into stack, Stack pointer gets decremented. Conversely it is incremented when data is retrieved from stack.

## Temporary register:

- It is an 8-bit register, which holds the temporary data of arithmetic and logical operations.
- It is used by the microprocessor. It is not accessible to programmer.

## Flags:

- 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 |

If a flip-flop for a particular flag is set, then it indicates 1. When it is reset, it indicates 0.

## 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 microprocessors: INTR, RST 7.5, RST 6.5, RST 5.5, TRAP.
- **Priorities of Interrupts:** TRAP > RST 7.5 > RST 6.5 > RST 5.5 > INTR

**Serial Input/output control**:

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

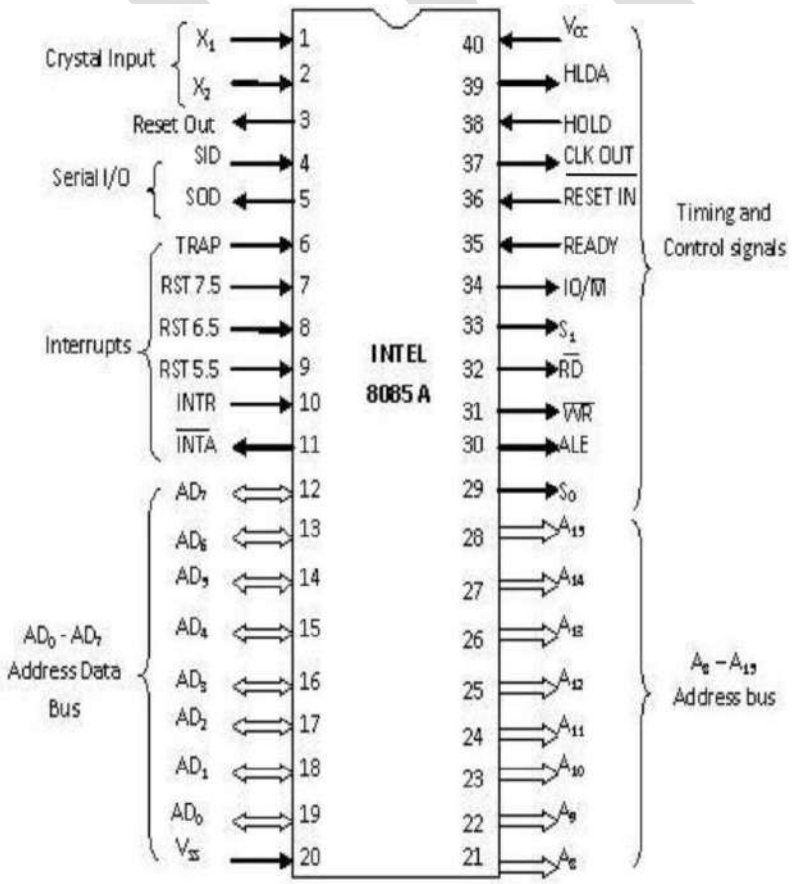
Address buffer and Address-Data buffer:

- 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



8085 is a 40 pin IC, DIP package. The pins of the 8085 microprocessor can be divided into six categories based on their function as follows:

1. Power supply and clock signals
2. Address bus
3. Data bus
4. Control and status signals
5. Interrupts and externally initiated signals
6. Serial I/O ports

## 1. Power supply and clock signals

### Power Signals

- There are two power supply signals Vcc and Vss. Vcc indicates +5V powers supply and Vss indicates ground reference.

### Clock signals

There are three clock signals i.e **X1, X2**, **CLK (OUT)**

- **X1, X2** pins are connected with a crystal or LC network to maintain the internal frequency of the clock generator.
- **CLK (OUT)** signal can be used as the system clock for other devices.

## 2. Address bus

- The address bus has 16 lines i.e. it can carry 16 bits at a time. However, out of 16, 8 are multiplexed with the data bus.
- A8 – A15 are unidirectional and are used to carry high-order address of 16-bit address

## 3. Data bus

- The address bus pins or data bus pins range from AD0 to AD7, and these pins are called multiplexing lines that can do both addresses as well as data transfer.

## 4. Control and status signals

### The Control signals are

#### RD’

- It is a signal to control READ operation. When it is low the selected memory or input-output device is read.

#### WR’

- It is a signal to control WRITE operation. When it goes low the data on the data bus is written into the selected memory or I/O location.

#### Address Latch Enable (ALE)

- This is an output signal used to give information on ($AD_0$-$AD_7$) contents.
- When the ALE pulse is HIGH, it indicates that the contents of ($AD_0$-$AD_7$), are addressed.

  When the ALE pulse is LOW, it indicates that the contents are data.

### The Status signals are

**IO/M'**

- It is a status signal which determines whether the address is for input-output or memory.
- When it is high (1) the address on the address bus is for input-output devices. When it is low (0) the address on the address bus is for the memory.

$S_1$ and $S_0$:

- These are output status signals used to get information on the operation performed by the microprocessor.

| $S_1$ | $S_0$ | Status |
|---|---|---|
| 0 | 0 | Halt |
| 0 | 1 | write |
| 1 | 0 | Read |
| 1 | 1 | Opcode Fetch |

5. Interrupts and externally initiated signals

Interrupts

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.

Externally initiated signals

INTA'

- It is an interrupt acknowledgement sent by the microprocessor after INTR is received.

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.
- This signal indicates that a peripheral such as DMA Controller is requesting the use of address & data buses

HLDA (HOLD Acknowledge)

- This active high signal is used to acknowledge HOLD request.

6. Serial I/O ports

**SID (Serial I/P Data):**

- This input signal is used to accept serial data bit by bit from the external device.

**SOD (Serial O/P Data):**

- This is an output signal which enables the transmission of serial data bit by bit to the external device.

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

Registers are a type of computer memory used to quickly accept, store and transfer data and instructions that are being used immediately by the processor.

There are two types of register in Intel 8085 Microprocessor.

a. General Purpose Registers (GPR)

b. Special Purpose Register (SPR)

### (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.

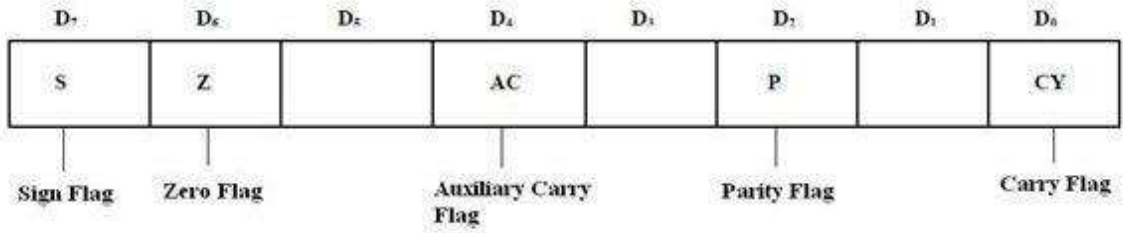
### (b) Special Purpose Registers

**Accumulator (A):**

- The accumulator is an 8-bit register used for arithmetic and logical operations. It holds one of the operands during calculations and stores the result.

**Flag registers:**

- 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. **Auxiliary 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 Auxiliary Carry flag is set. If not flag is reset.

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.

**Instruction Register:**

- The instruction register holds the opcode (operation code or instruction code) of the instruction which is being decoded and executed.

**Temporary Register:**

- It is an 8-bit register associated with the ALU. It holds data during an arithmetic/logical operation. It is used by the microprocessor. It is not accessible to programmer.

**Program Counter (PC):**

- It is a 16-bit special purpose register. It is used to hold the address of memory of the next instruction to be executed. It keeps the track of the instruction in a program while they are being executed.
- The microprocessor increments the content of the next program counter during the execution of an instruction so that at the end of the execution of an instruction it points to the next instructions address in the program.

**Stack Pointer (SP):**

- It is a 16-bit special function register used as memory pointer. A stack is nothing but a portion of RAM.
- In the stack, the contents of only those registers are saved, which are needed in the later part of the program.

Distinguish between SPR & GPR

| General Purpose Registers | Special purpose register |
|---|---|
| General purpose registers are used by programmer to store data | Special purpose registers are used by CPU for temporary storage of the data for calculations and other purposes. |

| | |
|---|---|
| It can be used for various operations, like arithmetic, data movement, or temporary storage. | These registers are designed for particular tasks, such as keeping track of program flow, managing the stack, or storing flags indicating the status of operations. |
| General purpose registers are symmetric and interchangeable. | Special purpose registers are symmetric. |
| It hold data and addresses | It hold the state of a program |
| Examples- B,C,D,E,H,L | Examples- Stack Pointer, Program Counter etc. |

## Timing & Control Module

The Timing and Control Unit generates the necessary timing signals to synchronize the activities of the microprocessor. There are certain timing and control signals like Control signals, DMA Signals, RESET signals and Status signals.

### Control Signals: RD', WR', ALE

- ALE is used for provide control signal to synchronize the components of microprocessor and timing for instruction to perform the operation.
- RD (Active low) and WR (Active low) are used to indicate whether the operation is reading the data from memory or writing the data into memory respectively.

### Status Signals: S0, S1, IO/M'

- IO/M (Active low) is used to indicate whether the operation belongs to the memory or peripherals.

| IO/M' | S1 | S0 | Data Bus Status |
|---|---|---|---|
| 0 | 1 | 1 | Opcode fetch |
| 0 | 1 | 0 | Memory read |
| 0 | 0 | 1 | Memory write |
| 1 | 1 | 0 | I/O read |
| 1 | 0 | 1 | I/O write |
| 1 | 1 | 1 | Interrupt acknowledge |
| 0 | 0 | 0 | Halt |

### DMA Signals: HOLD, HLDA

- HOLD: Indicates that another master is requesting the use of the address and data buses. The CPU, upon receiving the hold request, will relinquish the use of the bus as soon as the completion of the current bus transfer. Internal processing can continue. The processor can regain the bus only after the HOLD is removed

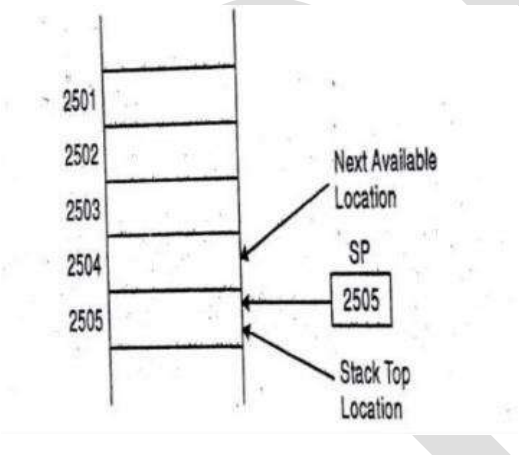
- HLDA: It is a signal which indicates that the hold request has been received after the removal of a HOLD request, the HLDA goes low.

### Reset Signals: Reset in, Reset Out

- 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.

## 1.7 Stack, Stack pointer &Stack top.

- The stack is a special area in memory used by the CPU to store register information or general data information during program execution. The stack has a top and a bottom.
- A stack is a region of memory that is used to store data temporarily. In the 8085 microprocessor, the stack is implemented using a dedicated block of memory known as the stack pointer (SP).
- The stack pointer (SP) register in 8085 contains the 16-bit offset from the start of the segment to the memory location where a word was most recently stored on the stack.
- Last memory location of the occupied portion of the stack is called Stack Top.




- The stack pointer points to the current top of the stack, which is the location where the next item will be stored.
- The stack used is a Last in first out (LIFO) structure that uses push and pull (pop) operations.
- When data is pushed onto the stack, the stack pointer is decremented by one to point to the next available memory location. When data is popped from the stack, the stack pointer is incremented by one to point to the next item on the stack.

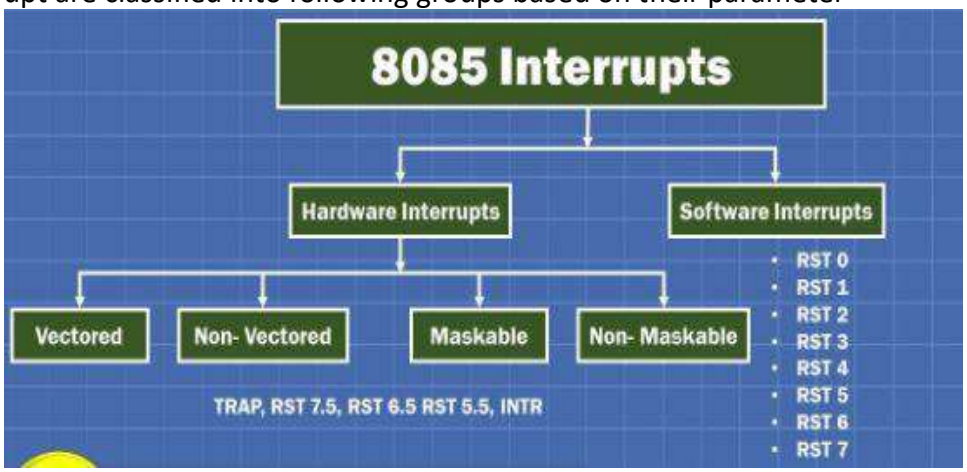
The 8085 microprocessor provides a set of instructions that can be used to manipulate the stack. These instructions include:

1. PUSH: This instruction is used to push a register or memory location onto the stack. The contents of the register or memory location are first copied to the memory location pointed to by the stack pointer, and then the stack pointer is decremented.
2. POP: This instruction is used to pop the topmost item from the stack. The contents of the memory location pointed to by the stack pointer are first copied to the register or memory location specified in the instruction, and then the stack pointer is incremented.

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

In the 8085 microprocessor, an interrupt is a signal that temporarily suspends the normal execution of a program and redirects the control to a specific interrupt service routine (ISR). Interrupts allow the microprocessor to respond to external events, such as user input, system events, or hardware signals, without the need for constant polling.

Interrupt are classified into following groups based on their parameter




1. **Hardware and Software Interrupts**
   - When microprocessors receive interrupt signals through pins (hardware) of microprocessor, they are known as *Hardware Interrupts*. There are 5 Hardware Interrupts in 8085 microprocessor. They are – *INTR, RST 7.5, RST 6.5, RST 5.5, TRAP*.
   - *Software Interrupts* are those which are inserted in between the program which means these are mnemonics of microprocessor. There are 8 software interrupts in 8085 microprocessor. They are – *RST 0, RST 1, RST 2, RST 3, RST 4, RST 5, RST 6, RST 7*.

Hardware Interrupts are classified as follows:

i. **Vector interrupt**
   - In this type of interrupt, the interrupt address is known to the processor. For example: RST7.5, RST6.5, RST5.5, TRAP

ii. **Non-Vector interrupt**
   - In this type of interrupt, the interrupt address is not known to the processor so, the interrupt address needs to be sent externally by the device to perform interrupts. For example: INTR.

iii. **Maskable Interrupts**
   - Maskable Interrupts are those which can be disabled or ignored by the microprocessor. These interrupts are either edge-triggered or level-triggered, so they can be disabled. *INTR, RST 7.5, RST 6.5, RST 5.5* are maskable interrupts in 8085 microprocessor.

iv. **Non-Maskable Interrupts**

- Non-Maskable Interrupts are those which cannot be disabled or ignored by microprocessor. *TRAP* is a non-maskable interrupt. It consists of both level as well as edge triggering and is used in critical power failure conditions.

There are five interrupt signals in the 8085 microprocessor:

1. **TRAP:**
   The TRAP interrupt is a non-maskable interrupt that is generated by an external device, such as a power failure or a hardware malfunction. The TRAP interrupt has the highest priority and cannot be disabled.
2. **RST 7.5:**
   The RST 7.5 interrupt is a maskable interrupt that is generated by a software instruction. It has the second highest priority.
3. **RST 6.5:**
   The RST 6.5 interrupt is a maskable interrupt that is generated by a software instruction. It has the third highest priority.
4. **RST 5.5:**
   The RST 5.5 interrupt is a maskable interrupt that is generated by a software instruction. It has the fourth highest priority.
5. **INTR:**
   The INTR interrupt is a maskable interrupt that is generated by an external device, such as a keyboard or a mouse. It has the lowest priority and can be disabled.

## Masking of Interrupt (SIM, RIM)

Masking of Interrupts in 8085 microprocessor is most important portion of interrupt. In 8085 microprocessor masking of interrupt we can do for four hardware interrupts INTR, RST 5.5, RST 6.5, and RST 7.5. The masking of 8085 interrupts can do at different levels.

- The maskable interrupts by default masked by the Reset signal. So no interrupt recognized by the hardware reset.
- The interrupts can enabled by the EI instruction.
- The three RST interrupts can selectively masked by loading the appropriate word in the accumulator and executing SIM instruction. This is called software masking.
- All maskable interrupts disabled whenever an interrupt recognized.
- All maskable interrupts can disabled by executing the DI instruction.

SIM (Set Interrupt Mask) and RIM (Read Interrupt Mask) commands are two important instructions inside the Intel 8085 microprocessor architecture.

### SIM instruction:

The SIM instruction generally used to mask or unmask RST hardware interrupts. When it executed, the **SIM** instruction reads the content of accumulator and accordingly mask or unmask the interrupts. The format of control word to be stored in the accumulator before executing SIM instruction shown in Fig

| Bit position | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| Name | SOD | SDE | X | R7.5 | MSE | M7.5 | M6.5 | M5.5 |
| Explanation | Serial data to be sent | Serial data enable—set to 1 for sending | Not used | Reset RST 7.5 flip-flop | Mask set enable—Set to 1 to mask interrupts | Set to 1 to mask RST 7.5 | Set to 1 to mask RST 6.5 | Set to 1 to mask RST 5.5 |

In addition to masking interrupts, SIM instruction can used to send serial data on the SOD line of the processor. Now need to send the data. So it placed in the MSB bit of the accumulator. And finally the serial data output enabled by making D6 bit to 1.

### RIM instruction:

RIM instruction is used to read the status of the interrupt mask bits. So when RIM instruction is executed. The accumulator then loaded with the current status of the interrupt masks and the pending interrupts also. The format and the meaning of the data stored in the accumulator after execution of RIM instruction shown in Fig

| Bit position | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| Name | SID | I7.5 | I6.5 | I5.5 | IE | M7.5 | M6.5 | M5.5 |
| Explanation | Serial input data in the SID pin | Set to 1 if RST 7.5 is pending | Set to 1 if RST 6.5 is pending | Set to 1 if RST 5.5 is pending | Set to 1 if interrupts are enabled | Set to 1 if RST 7.5 is masked | Set to 1 if RST 6.5 is masked | Set to 1 if RST 5.5 is masked |

In addition, RIM instruction is used to read the serial data on the SID pin of the processor. The data on the SID pin then stored in the MSB of the accumulator after the execution of the RIM instruction.

# UNIT-2: INSTRUCTION SET AND ASSEMBLY LANGUAGE PROGRAMMING

- An instruction is a command to the microprocessor to perform a given task on a specified data.
- Each instruction has two parts: one is task to be performed, called the operation code (opcode), and the second is the data to be operated on, called the operand.
- 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 8-bit (or 16-bit) address. In some instructions, the operand is implicit.
- An instruction is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions, called the instruction set, determines what functions the microprocessor can perform.

## 2.1 Addressing data & Differentiate between one-byte, two-byte & three-byte instructions with 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 one-byte instruction, the opcode and the operand of an instruction are represented in one byte.
- These type of instruction occupy only one memory location.
- Example:

| Opcode | Operand | Machine code/Hex code |
|---|---|---|
| MOV | A, B | 78 |
| ADD | M | 86 |

### 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.
- These type of instruction occupies two memory location.
- Example:

| Opcode | Operand | Machine code/Hex code | Byte description |
|---|---|---|---|
| MVI | A, 7FH | 3E | First byte |
| | | 7F | Second byte |
| ADI | 0FH | C6 | First byte |
| | | 0F | Second byte |

### 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.
- These type of instruction occupies three memory location.
- Examples:

| Opcode | Operand | Machine code/Hex code | Byte description |
|---|---|---|---|
| JMP | 9050H | C3 | First byte |
| | | 50 | Second byte |
| | | 90 | Third byte |
| LDA | 8850H | 3A | First byte |
| | | 50 | Second byte |
| | | 88 | Third byte |

## 2.2 Addressing modes in instructions with suitable examples.

Each instruction requires some data on which it has to operate. There are different techniques to specify data for instructions. These techniques are called **addressing modes.** The 8085 has the following five types of addressing modes:

### 1. Immediate Addressing Mode

- In immediate addressing mode, the data (8 / 16 bit) is specified in the instruction itself.
- The immediate addressing instructions are either 2 bytes or 3 bytes long.
- In 2-byte instruction, the first byte is OPCODE, and the second byte is the 8-bit data.
- In 3-byte instruction, the first byte is OPCODE, second and third bytes are 16-bit data.
- The instruction containing the letter "I" indicate immediate addressing mode.

Examples:

1. MVI A, A0 H: This instruction transfers immediate data (A0 H) to A register.
2. LXI H, C200 H: This instruction transfer 16-bit immediate data C200 to HL register pair.

   Lower order data(00H) to L register and high order data (C2 H) to H register.

### 2. Register Addressing Mode

- In register addressing mode, the source and destination operands are general-purpose registers.
- The register addressing instructions are generally of 1 byte i.e. OPCODE only.

Examples:

1. MOV D, B: This instruction copies the contents of register B to the D register. The source and destination operands are both registers.
2. ADD B: This instruction adds the content of the B register and A register, The data is present in both B and A registers. The result is stored in the accumulator.

### 3. Direct Addressing Mode

- In direct addressing mode, the 16-bit address of the operand is given within the instruction itself.
- The instruction in the direct addressing mode is 3-byte instructions. The first byte is OPCODE, the second slower order address mode, and the third is the higher-order address mode.

Examples:

1. LDA C200 H: Load accumulator directly from the memory location. In this instruction, the contents of the C200 memory location are transferred to the accumulator.
2. STA C200 H: Store accumulator directly to memory location. In this instruction, the content of the accumulator is stored at memory location C200 H.

### 4. Indirect Addressing Mode

- In indirect addressing mode, the instruction references the memory through a register pair.
- i.e. the memory address where the operand is located is specified by the content of a register pair.

Examples:

1. MOV A, M: In this case, M is a memory pointer specifying the HL register pair where the address is stored. The contents of the HL pair are used as addresses and the content of that memory location is transferred to the accumulator.
2. LDAX B: In this case, the BC register pair is used as an address and the content of the memory location specified by the BC register pair is copied to the accumulator.

### 5. Implied Addressing or Implicit Addressing Mode

- The implied mode of addressing does not require any operand.
- The data is specified within OPCODE itself.
- Generally, the implied addressing mode instruction is a 1-byte instruction.
- The data is supposed to be present generally in the accumulator.

Examples:

1. RAL: Rotate the accumulator left, it operates on the data in the accumulator only. So whenever RAL is used it is implied that the data to be operated on is available in the accumulator only.
2. CMC: Complement carry flag.

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

- In microprocessor, the **instruction** set is the collection of the instructions that the microprocessor is designed to execute.
- The programmer writes a program in assembly language using these instructions. These instructions have been classified into the following groups:

- Data Transfer Instruction
- Arithmetic Instructions
- Logical Instructions
- Branching Instructions
- Stack, I/O and Machine Control Instructions

## 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.

| Sl No | Instruction Set | Explanation | Example |
|---|---|---|---|
| 1 | MOV $r_1$, $r_2$<br>[r1] ← [r2] | Move the content of the one register to another | MOV A, B |
| 2 | MOV r, M<br>[r]←[[H-L]] | Move the content of memory to register | MOV B, M |
| 3 | MOV M, r<br>[[H-L]]←[r] | Move the content of register to memory | MOV M, C |
| 4 | MVI r, data<br>[r] ←data | Move immediate data to register | MVI A, 08 |
| 5 | MVI M, data<br>[[H-L]]←data | Move immediate data to memory | MVI M, 09 |
| 6 | LXI rp,data 16<br>[rp] ←data 16 bits,<br>[rh] ←8 MSBs,<br>[rl] ←8 LSBs of data | Load Register pair immediate | LXI H, 2500H |
| 7 | LDA addr<br>[A] ←[addr] | Load Accumulator direct | LDA 2400 H |
| 8 | STA Addr<br>[addr] ←[A] | Store accumulator direct.<br>(The content of the accumulator is stored in the memory location | STA 2000H |
| 9 | LHLD addr<br>[L] ←[addr],<br>[H] ← [addr + 1 ] | Load H-L pair direct | LHLD 2500H |

| | | | |
|---|---|---|---|
| 10 | SHLD addr<br>[addr] ←[L],<br>[addr +1] ← [H] | Store H-L pair direct.<br>(The content of register H,L are stored in the memory location) | SHLD 2500 H |
| 11 | LDAX rp<br>[A] ←[[rp]] | Load accumulator indirect | LDAX B |
| 12 | STAX rp<br>[[rp]] ←[A] | Store accumulator indirect.<br>The content of the accumulator is stored in the memory location whose address is in the register pair rp. | STAX D |
| 13 | XCHG<br>[H-L] ↔[D-E] | Exchange the contents of H-L with D-E pair | XCHG |

## Arithmetic Instructions

The instructions of this group perform arithmetic operations such as addition, subtraction, increment or decrement of the content of a register or a memory.

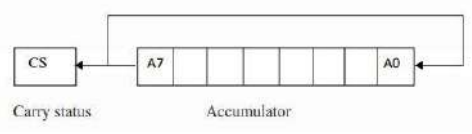
| Sl No | Instruction Set | Explanation | Example |
|---|---|---|---|
| 1 | ADD r<br>[A] ←[A]+[r] | Add register to accumulator | ADD C |
| 2 | ADD M<br>[A] ← [A] + [[H-L]] | Add memory to accumulator | ADD M |
| 3 | ADC r<br>[A] ← [A] + [r] + [CS] | Add register with carry to accumulator | ADC B |
| 4 | ADC M<br>[A] ← [A] + [[H-L]] +[CS] | Add memory with carry to accumulator | ADC M |
| 5 | ADI data<br>[A] ← [A] + data | Add immediate data to accumulator | ADI 55 |
| 6 | ACI data<br>[A] ← [A] + data + [CS] | Add with carry immediate data to accumulator | ACI 55 |
| 7 | SUB r<br>[A] ←[A]-[r] | Subtract register from accumulator | SUB C |
| 8 | SUB M<br>[A] ← [A] - [[H-L]] | Subtract memory from accumulator | SUB D |

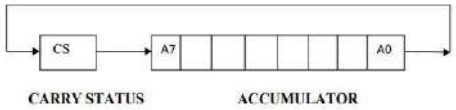
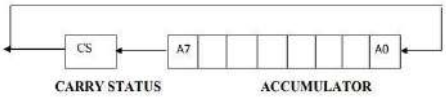
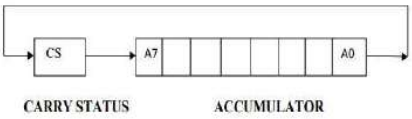
| | | | |
|---|---|---|---|
| 9 | SBB r<br>[A] ←[A]-[H-L]] - [CS] | Subtract memory from accumulator with borrow | SBB D |
| 10 | SUI data<br>[A] ←[A]-data | Subtract immediate data from accumulator | SUI 55 |
| 11 | SBI data<br>[A] ←[A]-data-[CS] | Subtract immediate data from accumulator with borrow | SBI 08 |
| 12 | INR r<br>[r] ←[r]+1 | Increment register content | INR D |
| 12 | INR M<br>[[H-L]] ←[[H-L]]+1 | Increment memory content | INR M |
| 13 | DCR r<br>[r] ←[r] -1 | Decrement register content | DCR C |
| 14 | DCR M<br>[[H-L]] ← [[H-L]]-1 | Decrement memory content | DCR M |
| 15 | INX rp<br>[rp] ←[rp]+1 | Increment memory content | INX B |
| 16 | DCX rp<br>[rp] ←[rp]-1 | Decrement register pair | DCX B |
| 17 | DAD rp<br>[H-L] ←[H-L] + [rp] | Add register paid to H-L pair | DAD D |
| 18 | DAA | Decimal adjust accumulator | DAA |

## Logical Group

The instructions in this group perform logical operation such as AND, OR, XOR, compare, rotate, etc.

| Sl No | Instruction Set | Explanation | Example |
|---|---|---|---|
| 1 | ANA r<br>[A] ←[A]∧[r] | AND register with accumulator | ANA B |
| 2 | ANA M<br>[A] ←[A]∧[[H-L] | AND memory with accumulator | ANA 2300H |

| | | | |
|---|---|---|---|
| 3 | ANI data<br>[A] ← [A] ∧ [data] | AND immediate data with accumulator | ANI 09 |
| 4 | ORA r<br>[A] ←[A]∨[r] | OR-register with accumulator | ORA C |
| 5 | ORA M<br>[A] ←[A]∨[[H-L]] | OR-memory with accumulator | ORA M |
| 6 | ORI data<br>[A]←[A] ∨ [data] | OR -immediate data with accumulator | ORI 98H |
| 7 | XRA r<br>[A]←[A] ⊕ [r] | XOR register with accumulator | XRA B |
| 8 | XRA M<br>[A] ← [A] ⊕ [[H-L]] | XOR memory with accumulator | XRA M |
| 9 | XRI data<br>[A] ←[A] ⊕ [data] | XOR immediate data with accumulator | XRI 07 |
| 10 | CMA<br>[A] ←[Ā] | Complement the accumulator | CMA |
| 11 | CMC<br>[CS] ←[ C̄S] | Complement the carry status | CMC |
| 12 | STC<br>[CS] ← 1 | Set carry status | STC |
| 13 | CMP r<br>[A]-[r] | Compare register with accumulator | CMP D |
| 14 | CMP M<br>[A] - [[H-L]] | Compare memory with accumulator | CMP M |
| 15 | CPI data<br>[A] - data | Compare immediate data with accumulator | CPI 21 |
| 16 | RLC | Rotate accumulator left<br>  | RLC |
| 17 | RRC | Rotate accumulator right | RRC |

| | | | |
|---|---|---|---|
| | |  | |
| 18 | RAL | Rotate accumulator left through carry  | RAL |
| 19 | RAR | Rotate accumulator right through carry  | RAR |

## 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 (unconditional and conditional)

1. Call (unconditional and conditional)

2. Return (unconditional and conditional)

**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 in satisfied.

| OPCODE | OPERAND | EXPLANATION | EXAMPLE |
|---|---|---|---|
| JC | address | Jumps to the address if carry flag is 1 | JC 2050 |

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

**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 |

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

**Return Instructions –** The return instruction transfers the program sequence from the subroutine to the calling program. Return 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 is 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 |

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

This group contains the instructions for input/output ports, stack and machine control.

| Instruction Set | Explanation | Example |
|---|---|---|
| IN port - address<br>[A] ← [Port] | Input to accumulator from I/O port | IN 08 |
| OUT port-address<br>[Port] ← [A] | Output from accumulator to I/O port | OUT 21 |
| PUSH rp<br>[[SP] - 1] ← [rh],<br>[[SP] - 2] ← [rh],<br>[SP] ← [SP] - 2 | Push the content of register pair to stack | PUSH B |
| PUSH PSW<br>[SP]-1] ← [A],<br>[[SP] -2] ← PSW,<br>[SP] ← [SP] - 2 | Push processor Status word | |
| POP rp<br>[rl] ← [ [ SP ] ],<br>[rh] ← [[SP]+1],<br>[SP] ← [SP] + 2 | Pop the content of register pair, which was saved, from the stack | POP D |
| HLT | Halt | HLT |
| XTHL<br>[L] ↔ [[SP]],<br>[H] ↔ [[SP] + 1] | Exchange top stack with H-L | XTHL |
| SPHL<br>[H-L] → [SP] | Moves the contents of H-L pair to stack pointer | SPHL |
| EI | Enable Interrupts | EI |

| | | |
|---|---|---|
| SIM | Set Interrupts Masks | SIM |
| RIM | Read Interrupts Masks | RIM |
| NOP | No Operation | NOP |

## 2.4 Simple Assembly Language Programming of 8085

### 2.4.1 Simple Addition & Subtraction

- Addition of two 8-bit number: Sum 8 bits

| *Mnemonics* | *Operands* | *Comments* |
|---|---|---|
| LXI | H, 2501 H | Get address of 1st number in H-L pair. |
| MOV | A,M | 1st number in accumulator. |
| INX | H | Increment content of H-L pair. |
| ADD | M | Add 1st and 2nd numbers. |
| STA | 2503 H | Store sum in 2503 H. |
| HLT | | Stop |

Example:

Data:

2501-49H

2502-56H

The sum is stored in the memory location 2503H

Result:

2503-9FH

- 8 Bit Subtraction

| *Mnemonics* | *Operands* | *Comments* |
|---|---|---|
| LXI | H, 2501 H | Get address of 1st number in H-L pair. |
| MOV | A, M | 1st number in accumulator. |
| INX | H | Content of H-L pair increases from 2501 to 2502 H. |
| SUB | M | 1st number - 2nd number. |
| INX | H | Content of H-L pair becomes 2503 H. |
| MOV | M, A | Store result in 2503 H. |
| HLT | | Halt |

Example:

Data:

2501-49H

2502-32H

The result is stored in the memory location 2503H

Result:

2503-17H

- Addition of Two 8 Bit numbers: Sum 16 Bits

| Labels | Mnemonics | Operands | Comments |
|---|---|---|---|
| | LXI | H, 2501 H | Address of 1st number in H-L pair. |
| | MVI | C,00 | MSBs of sum in register C. Initial value = 00. |
| | MOV | A, M | 1st number in accumulator. |
| | INX | H | Address of 2nd number 2502 in H-L pair. |
| | ADD | M | 1st number + 2nd number. |
| | JNC | AHEAD | Is carry? No, go to the label AHEAD. |
| | INR | C | Yes, increment C. |
| AHEAD | STA | 2503 H | LSBs of sum in 2503 H. |
| | MOV | A, C | MSBs of sum in accumulator. |
| | STA | 2504 H | MSBs of sum in 2504 H. |
| | HLT | | Halt |

Example:

Data:

2501-98H

2502-9AH

Result:

2503-32H, LSB of sum

2504-01H, MSB of sum

- Decimal Addition of two 8 bit numbers: sum 16 bits

| Labels | Mnemonics | Operands | Comments |
|---|---|---|---|
| | LXI | H, 2501 H | Address of 1st number in H-L pair. |
| | MVI | C, 00 | MSDs of sum in register C. Initial value = 00. |
| | MOV | A, M | 1st number in accumulator. |
| | INX | H | Address of 2nd number 2502 in H-L pair. |
| | ADD | M | 1st number + 2nd number. |
| | DAA | | Decimal adjust. |
| | JNC | AHEAD | Is carry? No, go to the label AHEAD. |
| | INR | C | Yes, increment C. |
| AHEAD | STA | 2503 H | LSDs of sum in 2503 H. |
| | MOV | A, C | MSDs of sum in accumulator. |
| | STA | 2504 H | MSDs of sum in 2504 H. |
| | HLT | | |

Example:

Data:

2501-84D

2502-75D

Result:

2503-59D, LSDs of the sum

2504-01D, MSDs of the sum

- Addition of two 16 bit numbers: sum 16 bits or more

| *Labels* | *Mnemonics* | *Operands* | *Comments* |
|---|---|---|---|
| | LHLD | 2501 H | 1st 16-bit number in H-L pair. |
| | XCHG | | Get 1st number in D-E pair. |
| | LHLD | 2503 H | 2nd 16-bit number in H-L pair. |
| | MVI | C, 00 | MSBs of the sum in register C. Initial value = 00. |
| | DAD | D | 1st number + 2nd number |
| | JNC | AHEAD | Is carry? No, go to the label AHEAD. |
| | INR | C | Yes, increment C. |
| AHEAD | SHLD | 2505 H | Store LSBs of sum in 2505 and 2506 H. |
| | MOV | A, C | MSBs of sum in accumulator. |
| | STA | 2507 H | Store MSBs of the sum in 2507 H. |
| | HLT | | Halt |

Example:

Data:

2501-98H, LSBs of 1st number

2502-5B H, MSBs of 1st number

2503-4C H, LSBs of 2nd number

2504-8E H, MSBs of 2nd number

Result:

2505-E4, LSBs of sum

2506-E9, LSBs of sum

2507-00, MSBs of sum

- 8 bit Decimal Subtraction

| *Mnemonics* | *Operands* | *Comments* |
|---|---|---|
| LXI | H, 2502 H | Get address of 2nd number in H-L pair. |
| MVI | A, 99 | Place 99 in accumulator. |
| SUB | M | 9's complement of 2nd number. |
| INR | A | 10's complement of 2nd number. |
| DCX | H | Get address of 1st number. |

| | | |
|---|---|---|
| ADD | M | Add 1st number and 10's complement of 2nd number. |
| DAA | | Decimal adjustment. |
| STA | 2503 H | Store result in 2503 H. |
| HLT | | Halt |

Example:

Data:

2501-96

2502-38

Result:

2503-58

## 2.4.2 Logic Operations (AND, OR, Complement 1's & 2's) & Masking of bits

- One's complement of an 8 Bit number

| *Mnemonics* | *Operands* | *Comments* |
|---|---|---|
| LDA | 2501 H | Get data in accumulator. |
| CMA | | Take its complement. |
| STA | 2502 H | Store result in 2502 H. |
| HLT | | Halt |

Example:

Data:

2501-96H

Result:

2502-69H

- One's complement of a 16 Bit number

| *Mnemonics* | *Operands* | *Comments* |
|---|---|---|
| LXI | H, 2501 H | Address of LSBs of the number. |
| MOV | A, M | 8 LSBs of the number in accumulator. |
| CMA | | Complement of 8 LSBs of the number. |
| STA | 2503 H | Store 8 LSBs of result. |
| INX | H | Address of 8 MSBs of the number. |
| MOV | A, M | 8 MSBs of the number in accumulator. |
| CMA | | Complement of 8 MSBs of the number. |
| STA | 2504 | Store 8 MSBs of the result. |
| HLT | | Halt |

Example:

Data:

2501-85H, LSBs of the number

2502-54 H, MSBs of the number

Result:

2503-7A H, LSBs of the result

2504-AB H, MSBs of the result

- Two's Complement of an 8 bit-number

| *Mnemonics* | *Operands* | *Comments* |
|---|---|---|
| LDA | 2501 H | Get data in accumulator. |
| CMA | | Take its 1's complement. |
| INR | A | Take 2's complement. |
| STA | 2502 H | Store result in 2502 H. |
| HLT | | Stop |

Example:

Data:

2501-96H

Result:

2502-6AH

- Two's Complement of a 16 bit-number

| *Label* | *Mnemonics* | *Operands* | *Comments* |
|---|---|---|---|
| | LXI | H, 2501 H | Address of 8 LSBs of the number. |
| | MVI | B, 00 | Use register B to store carry. |
| | MOV | A, M | 8 LSBs in accumulator. |
| | CMA | | 1's complement of 8 LSBs of the number. |
| | ADI | 01 | 2's complement of 8 LSBs of the number. |
| | STA | 2503 H | Store 8 LSBs of the result. |
| | JNC | GO | |
| | INR | B | Store carry. |
| GO | INX | H | Address of 8 MSBs of the number. |
| | MOV | A, M | 8 MSBs in accumulator. |
| | CMA | | 1's complement of 8 MSBs of the number. |
| | ADD | B | Add carry. |
| | STA | 2504 H | Store 8 MSBs of the result. |
| | HLT | | Stop. |

Example:

Data:

2501-8C, LSBs of the number

2502-5B, MSBs of the number

Result:

2503-74, LSBs of the result

2504-A4, MSBs of the result

2's complement of the number is A474

- Mask off least significant 4 bits of an 8-bit number

| *Mnemonics* | *Operands* | *Comments* |
|---|---|---|
| LDA | 2501 H | Get data in accumulator. |
| ANI | F0 | Mask off the least significant 4 bits. |
| STA | 2502 H | Store result in 2502 H. |
| HLT | | Stop |

Example:

Data:

2501-A6

Result:

2502-A0

- Mask off most significant 4 bits of an 8-bit number

| *Mnemonics* | *Operands* | *Comments* |
|---|---|---|
| LDA | 2501 H | Get data in accumulator. |
| ANI | 0F | Mask off the most significant 4 bits. |
| STA | 2502 H | Store result in 2502 H. |
| HLT | | Stop |

Example:

Data:

2501-A6

Result:

2502-06

### 2.4.3 Counters & Time delay (Single Register, Register Pair, More than Two Register)

- Delay subroutine using one Register

| Labels | Mnemonics | Operands | Comments |
|---|---|---|---|
| | MVI | B, 10H | Get 10 in register B. |
| LOOP | DCR | B | Decrement register B. |
| | JNZ | LOOP | Has the content of register B become zero? No, jump to LOOP. Yes, proceed ahead. |
| | RET | | |

- Delay subroutine using Register Pair

| Label | Mnemonics | Operands | Comments |
|---|---|---|---|
| | LXI | D, FFFF | Get FFFF in register pair D-E |
| LOOP | DCX | D | Decrement count. |
| | MOV | A, D | Move content of register D to accumulator. |
| | ORA | E | Check if D and E are zero. |
| | JNZ | LOOP | If D-E is not zero, jump to LOOP |
| | RET | | Return to main program. |

- Delay subroutine using three Register

| Labels | Mnemonics | Operands | Comments |
|---|---|---|---|
| | MVI | A, 98 H | Get control word. |
| | OUT | 03 | Initialize port for LED display. |
| | MVI | B, 50 H | |
| LOOP 1 | MVI | C, FF | |
| LOOP 2 | MVI | D, FF | |
| LOOP 3 | DCR | D | Delay subroutine with three registers. |
| | JNZ | LOOP 3 | |
| | DCR | C | |
| | JNZ | LOOP 2 | |
| | DCR | B | |
| | JNZ | LOOP 1 | |
| | MVI | A, 01 | |
| | OUT | 01 | Output for LED |
| | HLT | | Stop. |

## 2.4.6 Code conversion, BCD Arithmetic & 16 Bit data Operation, Block Transfer.

BCD Arithmetic & 16 Bit data Operation discuss in 2.4.1

- Code conversion
  - Decimal to Hexadecimal/ BCD to Binary

| Label | Mnemonic | Comment |
|---|---|---|
| | LDA 8200H | Get data in accumulator |
| | MOV B,A | Move the content A to B |
| | ANI 0FH | Mask off most significant bit |
| | MOV D,A | Move the content A to D |
| | MOV A,B | Move the content B to A |
| | ANI F0H | Mask off least significant bit |
| | RLC | Rotate the content of A to left |
| | RLC | Rotate the content of A to left |
| | RLC | Rotate the content of A to left |
| | RLC | Rotate the content of A to left |
| | MOV C,A | Move the content A to C |
| | MVI E,0AH<br>XRA A | Get 0A in register E<br>Exclusive or with A |
| LOOP | ADD C | Add Accumulator+ C |
| | DCR E | Decrement E |
| | JNZ: LOOP | |
| | ADD D | Add Accumulator+ D |
| | STA 8201H | Store result in 8201H |
| | HLT | Stop |

- Hexadecimal to Decimal / Binary to BCD

| Label | Mnemonic | Comment |
|---|---|---|
| | LDA 8000H | Get data in accumulator |
| | MVI B, 00H | Content of B=00 |
| | MOV C, B | Move the content B to C |
| | CPI 64H | Compare accumulator with 64H |
| | JC: LOOP | If carry then jump to LOOP |
| | SUI 64H | Subtract with 64H |
| | INR B | Increment B |

| | | |
|---|---|---|
| | JMP:AHEAD | Jump to AHEAD |
| LOOP | CPI 0AH | Compare accumulator with 0AH |
| | JC:LABEL | If carry then jump to LABEL |
| | SUI 0AH | Subtract with 0AH |
| | INR C | Increment C |
| | JMP:LOOP | Jump to LOOP |
| LABEL | STA 8001H | Store unit digit in 8001H |
| | MOV A,C | Move the content C to A |
| | STA 8002H | Store tenth digit in 8002H |
| | MOV A,B | Move the content B to A |
| | STA 8003 | Store hundredth digit in 8002H |
| | HLT | Stop |

- Binary to Gray code

| Mnemonic | Comment |
|---|---|
| LDA 4050H | Get data in accumulator |
| MOV B,A | Move data to Register B |
| RAR | Rotate right with carry |
| XRA B | Exclusive OR with Register B |
| STA 4051H | Store result in 4051 |
| HLT | Stop |

- Gray to Binary code

| Label | Mnemonic | Comment |
|---|---|---|
| | LXI H, 5000H | Get address of number in H-L pair |
| | MOV A,M | Move number in Accumulator |
| | MVI B, 07H | Get 07 in Register B |
| LOOP | RAR | Rotate right Accumulator data |
| | XRA M | Exclusive OR memory with Accumulator |
| | DCR B | Decrement register B |

| | | |
|---|---|---|
| | JNZ: LOOP | Jump to Loop if Register B is not 0 |
| | STA 5001H | Store result in 5001H |
| | HLT | Stop |

- Block Transfer

| Label | Mnemonic | Comment |
|---|---|---|
| | LXI H, 8050H | Source address of data in H-L pair |
| | LXI D, 8070H | Destiny address of data in D-E pair |
| | MVI C, 10H | Set up B to count 10 bytes |
| LOOP: | MOV A,M | Data from source address to Accumulator |
| | STAX D | Store data byte as destination |
| | INX H | Source address of next data |
| | INX D | Destiny address of next data |
| | DCR C | Decrement count |
| | JNZ LOOP | Jump to label Loop |
| | HLT | Stop |

Example:

2.4.7 Compare between two numbers

- Larger of two number

| *Labels* | *Mnemonics* | *Operands* | *Comments* |
|---|---|---|---|
| | LXI | H, 2501 H | Address of 1st number in H-L pair. |
| | MOV | A, M | 1st number in accumulator. |
| | INX | H | Address of 2nd number in H-L pair. |
| | CMP | M | Compare 2nd number with 1st number. Is the 2nd number > 1st ? |
| | JNC | AHEAD | No, larger number is in accumulator. Go to AHEAD. |
| | MOV | A, M | Yes, get 2nd number in accumulator. |
| AHEAD | STA | 2503 H | Store larger number in 2503 H. |
| | HLT | | Stop |

Example:

Data:

2501-98H

2502-87H

Result:

2503-98H

- Smaller of two number

| Labels | Mnemonics | Operands | Comments |
|---|---|---|---|
| | LXI | H, 2501 H | Address of 1st number in H-L pair. |
| | MOV | A, M | 1st number in accumulator. |
| | INX | H | Address of 2nd number in H-L pair. |
| | CMP | M | Compare 2nd number with 1st. Is 1st number < 2nd number? |
| | JC | AHEAD | Yes, smaller number is in accumulator. Go to AHEAD. |
| | MOV | A, M | No, Get 2nd number in accumulator |
| AHEAD | STA | 2503 H | Store smaller number in 2503 H. |
| | HLT | | Stop |

Example:

Data:

2501-84H 2502-99H

Result:

2503-84H

2.4.8 Array Handling (Largest number & smallest number in the array)

- Largest number in a data array

| Labels | Mnemonics | Operands | Comments |
|---|---|---|---|
| | LXI | H, 2500 H | Address for count in H-L pair. |
| | MOV | C, M | Count in register C. |
| | INX | H | Address of 1st number in H-L pair. |
| | MOV | A, M | 1st number in accumulator. |
| | DCR | C | Decrement count. |
| LOOP | INX | H | Address of next number. |
| | CMP | M | Compare next number with previous maximum. Is next number > previous maximum? |

| | | | |
|---|---|---|---|
| | JNC | AHEAD | No, larger number is in accumulator. Go to the lable AHEAD |
| | MOV | A, M | Yes, get larger number in accumulator. |
| AHEAD | DCR | C | Decrement count. |
| | JNZ | LOOP | |
| | STA | 2450 H | Store result in 2450 H. |
| | HLT | | Stop. |

Example:

Data:

2500-03 H

2501-98 H

2502-75 H

2503-99 H

Result:

2450-99 H

- Smallest number in a data array

| *Labels* | *Mnemonics* | *Operands* | *Comments* |
|---|---|---|---|
| | LXI | H, 2500 H | Get address for count in H-L pair. |
| | MOV | C, M | Count in register C. |
| | INX | H | Get address of 1st number in H-L pair. |
| | MOV | A, M | 1st number in accumulator. |
| | DCR | C | Decrement count. |
| LOOP | INX | H | Address of next number in H-L pair. |
| | CMP | M | Compare next number with previous smallest. Is previous smallest less than next number? |
| | JC | AHEAD | Yes, smaller number in accumulator. Go to AHEAD. |
| | MOV | A, M | No, get next number in accumulator. |
| AHEAD | DCR | C | Decrement count. |
| | JNZ | LOOP | |
| | STA | 2450 H | Store smallest number in 2450 H. |
| | HLT | | Stop. |

Example:

Data:

2500-03 H

2501-86 H

2502-58 H

2503-75 H

Result:

2450-58H

# UNIT-3: TIMING DIAGRAMS

## 3.1 Define opcode, operand, T-State, Fetch cycle, Machine Cycle, Instruction cycle & discuss the concept of timing diagram.

- Timing Diagram is a graphical representation.
- It represents the execution time taken by each instruction in a graphical format.
- The execution time is represented in T-states.

### Opcode:

- Opcodes mean "operation codes". An opcode is the first part of an instruction that tells the computer what function to perform.
- Every computer has an operation code or opcode for each of its functions. It is an instruction that tells the processor what to do with the variable or data written beside it.

### Operand:

- An operand is the second part of the instruction, which tells the computer where to find or store the data or instructions.
- The number of operands varies among computers. Each instruction tells the Control Unit of the CPU what to do and how to do it.

| Instruction Format | Example |
|---|---|
| I \| Opcode | CMA<br>CME |
| I \| Opcode \| Operand | ADD 06H<br>LDA 20H |
| I \| Opcode \| Operand 1 \| Operand 2 | MOV R1,R2<br>ADD AX, BX |
| I \| Opcode \| Operand 1 \| Operand 2 \| Operand 3 | ADD R1, R2, R3<br>SUB R1, R2, R3 |

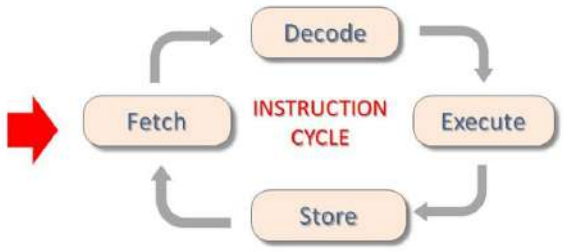
### Instruction cycle:

- The amount of time the microprocessor takes to fetch, decode and execute one Instruction. One Instruction cycle consists of several machine cycle.

OR

- The time required to execute an instruction is called instruction cycle.

The instruction cycle also known as the fetch–decode–execute cycle, or simply the fetch-execute cycle.




The instruction cycle of the 8085 microprocessor consists of four basic steps, which are:

- Fetch: In this step, the microprocessor fetches the instruction from the memory location pointed to by the program counter (PC). The PC is incremented by one after the fetch operation.

- Decode: Once the instruction is fetched, the microprocessor decodes it to determine the operation to be performed and the operands involved.
- Execute: In this step, the microprocessor performs the operation specified by the instruction on the operands.
- Store: Finally, the result of the execution is stored in the appropriate memory location or register.

### Fetch cycle:

- A fetch cycle is part of an instruction cycle and may occur more than once in the cycle.
- During the fetch execute cycle, the computer retrieves a program instruction from its memory. Every instruction cycle starts with a fetch, the opcode which the program counter points to.

### Machine Cycle:

- The amount of time the microprocessor takes to perform a memory or I/O access.
- One machine cycle consists of several clock cycle.

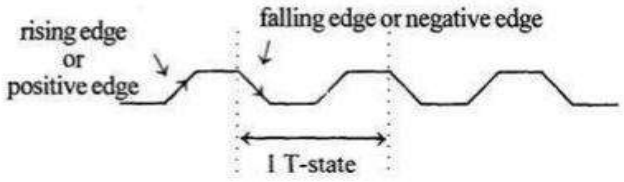
The 8085 microprocessor has 5 basic machine cycles. They are:

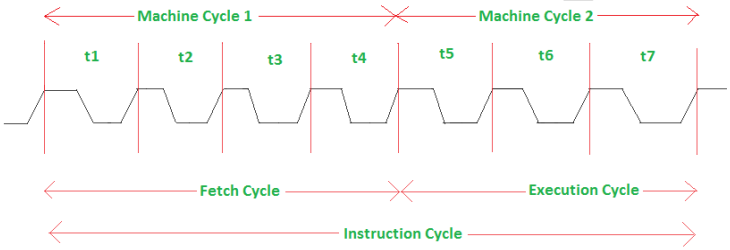
- Opcode Fetch {4T/6T- state}
- Memory Read {3T- state}
- Memory Write {3T- state}
- I/O Read {3T- state}
- I/O Write {3T- state}

### T-State:

- The time taken by the processor to execute a machine cycle is expressed in T-states.
- One T-state is equal to the time period of the internal clock signal of the processor.
- A t-state is measured from the falling edge of one clock pulse to the falling edge of the next clock pulse.

*Time period, T = 1/f ; where f = Internal clock frequency*



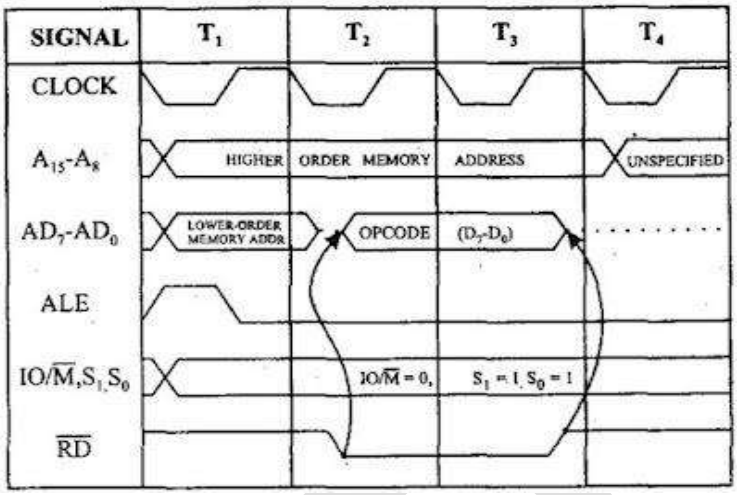




## 3.2 Draw timing diagram for memory read, memory write, I/O read, I/O write machine cycle.

### Opcode Fetch Machine Cycle of 8085

- Each processor instruction has a one-byte opcode.
- Operation codes are stored in memory. Thus, the processor performs an opcode load machine cycle to load the opcode from memory.
- Thus, each instruction begins with a machine cycle of opcodes.
- The time the processor takes to perform an opcode load cycle is 4T.

- At this time, the first, 3T-states are used to load the opcode from memory, and the remaining T-states are used for internal opcodes.




T1 clock cycle

- The content of PC is placed in the address bus; AD0 - AD7 lines contains lower bit address and A8 – A15 contains higher bit address.
- IO/M signal is low indicating that a memory location is being accessed. S1 and S0 also changed to 1.
- ALE is high, indicates that multiplexed AD0 – AD7 act as lower order bus.

T2 clock cycle

- Multiplexed address bus is now changed to data bus.
- The RD(bar) signal is made low by the processor. This signal makes the memory device load the data bus with the contents of the location addressed by the processor.

T3 clock cycle

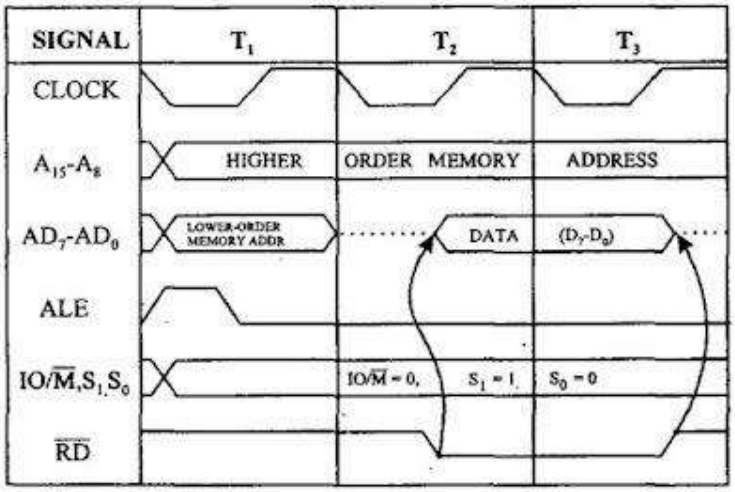
- The opcode available on the data bus is read by the processor and moved to the instruction register.
- The RD(bar) signal is deactivated by making it logic 1.

T4 clock cycle

- The processor decode the instruction in the instruction register and generate the necessary control signals to execute the instruction. Based on the instruction further operations such as fetching, writing into memory etc takes place.

## Memory Read Machine Cycle of 8085

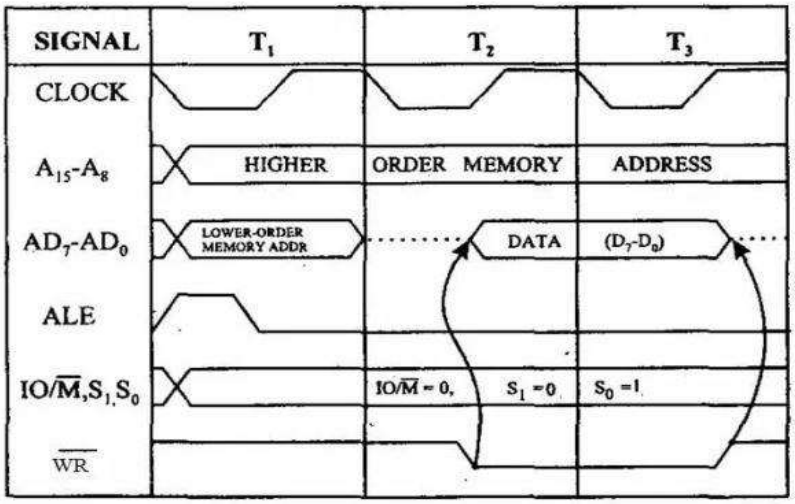
- A memory read machine cycle is executed by the processor to read a data byte from memory.
- The processor takes 3T states to perform this cycle.
- Instructions that have more than one byte word will use machine cycle after machine cycle to load the opcode.




- During T1, A8-A15 contains higher byte of address. At the same time ALE is high. Therefore, Lower byte of address A0-A7 is selected from AD0-AD7. Since it is memory ready operation, IO/M(bar) goes low.
- During T2 ALE goes low, RD(bar) goes low. Address is removed from AD0-AD7 and data D0-D7 appears on AD0-AD7.
- During T3, Data remains on AD0-AD7 till RD(bar) is at low signal.

## Memory Write Machine Cycle of 8085

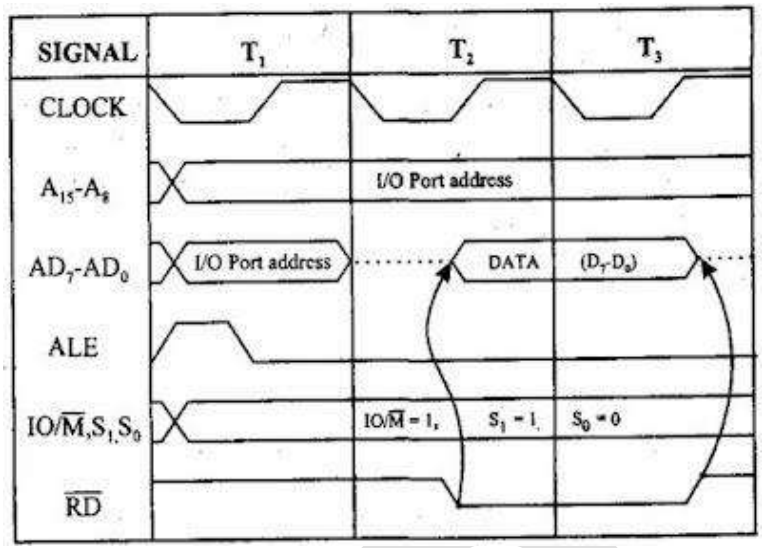
- A write-to-memory machine cycle is executed by the processor to write a data byte to memory.
- The processor takes 3T states to perform this cycle.
- Instructions that have more than one-byte word will use machine cycle after machine cycle to load the opcode.




- During T1, ALE is high and contains lower address A0-A7 from AD0-AD7. A8-A15 contains higher byte of address. As it is memory operation, IO/M goes low.
- During T2, ALE goes low, WR goes low and Address is removed from AD0-AD7 and then data appears on AD0-AD7.
- During T3 Data remains on AD0-AD7 till WR is low.

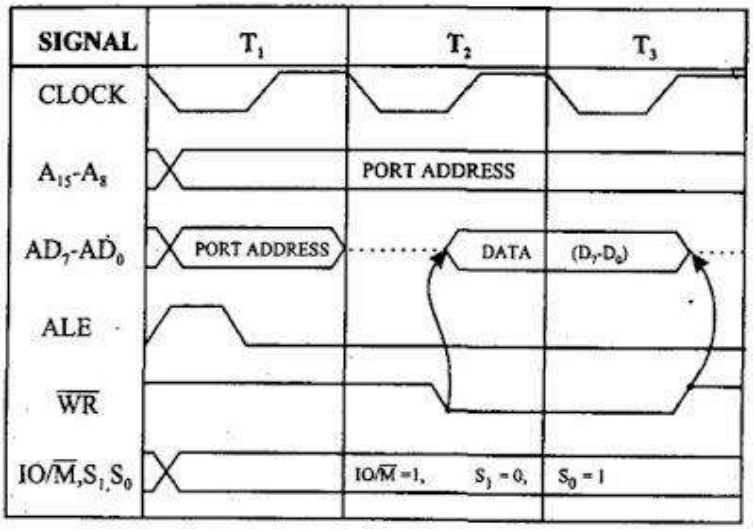
## I/O Read Machine Cycle

- A reader I/O cycle is performed by the processor to read a data byte from an I/O port or peripheral that is I/O mapped in the system.
- The processor needs 3T states to execute this machine cycle.




## I/O Write Machine Cycle

- A writer's I/O cycle is executed by the processor to write a data byte to an I/O port or peripheral that is I/O mapped in the system.
- The processor needs 3T states to execute this machine cycle.

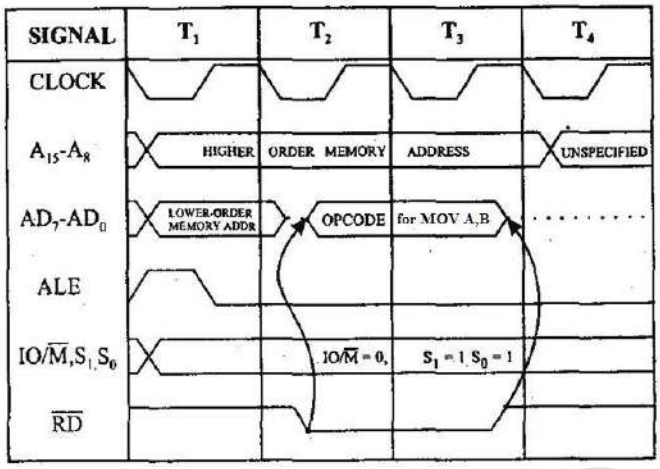



- During T1, the lower byte of address is duplicated into higher order address bus A8-A15. ALE is high and A0-A7 address is selected from AD0- AD7. As it is an IO operation IO/M goes low.
- During T2, ALE goes low, WR goes low and data appears on AD0-AD7 to write data into IO device.
- During T3, Data remains on AD0-AD7 till WR is low.

3.3 Draw a neat sketch for the timing diagram for 8085 instructions (MOV, MVI, LDA instruction).

## Timing Diagram of MOV $R_a$, $R_b$ instruction

The instruction MOV A,B is a 1-byte instruction. Microprocessor takes only one machine cycle (opcode fetch) to complete instruction. Hence, hex code for MOV A,B is paased to the microprocessor.



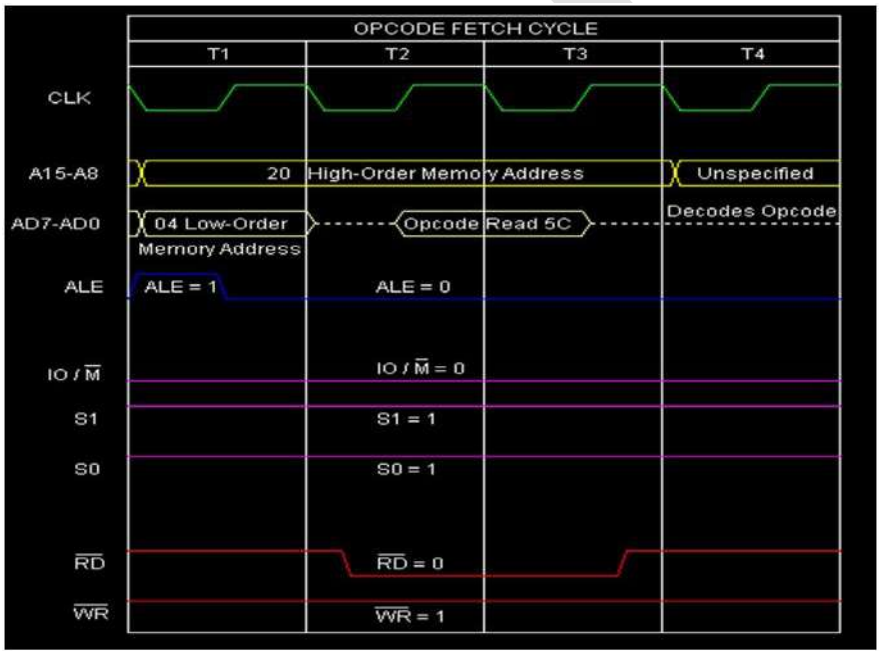

**OR**

**MOV E, H** is an example instruction of this type. It is a 1-Byte instruction. Suppose E register content is AB H, and H register content is 9C H. When the 8085 executes this instruction, the contents of E register will change to 9C H. This is shown as follows.

| | **Before** | **After** |
|---|---|---|
| **(E)** | AB H | 9C H |
| **(H)** | 9C H | 9C H |

| **Address** | **Hex Codes** | **Mnemonic** | **Comment** |
|---|---|---|---|
| 2004 | 5C | MOV E, H | E <- H |

The timing diagram of **MOV E, H** instruction is as follows.



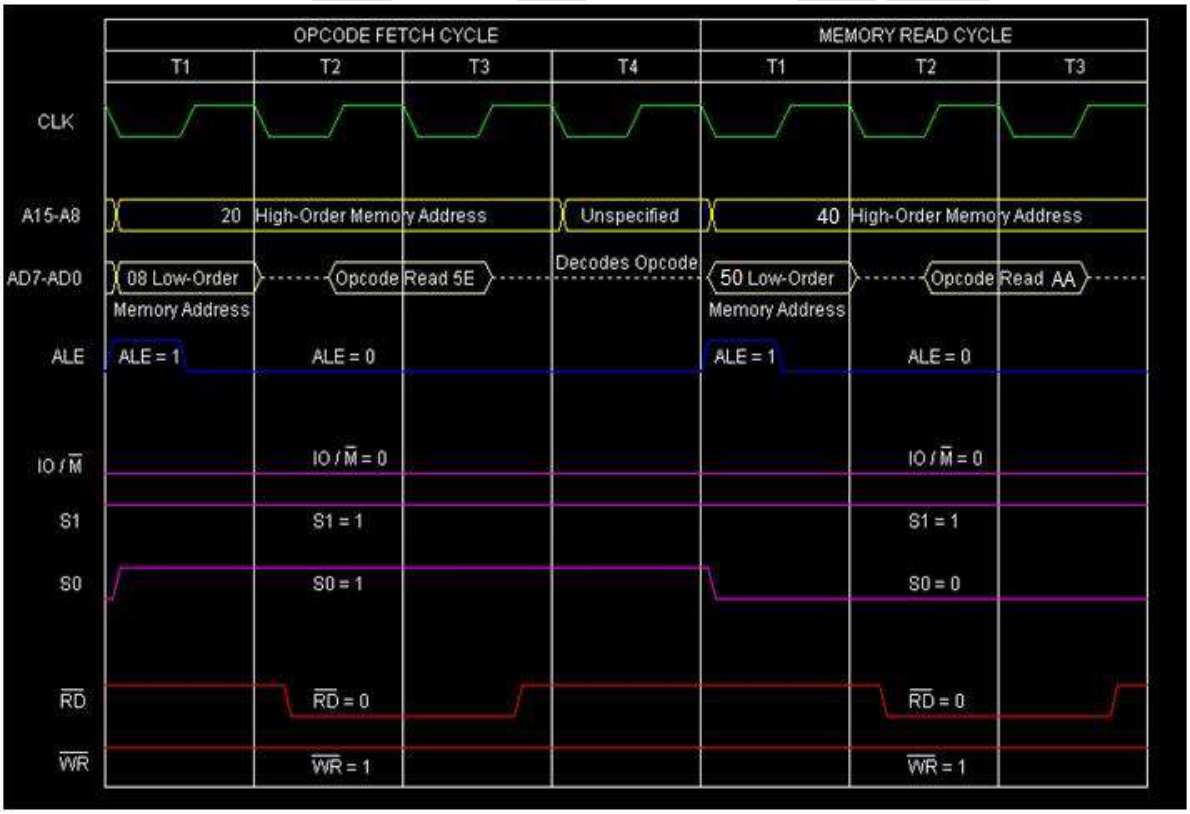
## Timing Diagram of **MOV $R_a$, M** instruction

**MOV E, M** is an example instruction of this type. It is a 1-Byte instruction. Suppose E register content is DBH, H register content is 40H, and L register content is 50H. Let us say location 4050H has the data value AAH. When the 8085 executes this instruction, the contents of E register will change to AAH, as shown below.

| | **Before** | **After** |
|---|---|---|
| **(E)** | DBH | AAH |
| **(HL)** | 4050H | 4050H |
| **(4050H)** | AAH | AAH |

| **Address** | **Hex Codes** | **Mnemonic** | **Comment** |
|---|---|---|---|
| 2008 | 2A | MOV E, M | Comment E <- Content of the memory location pointed by HL register pair |

The timing diagram for this **MOV E, M** instruction is as follows –



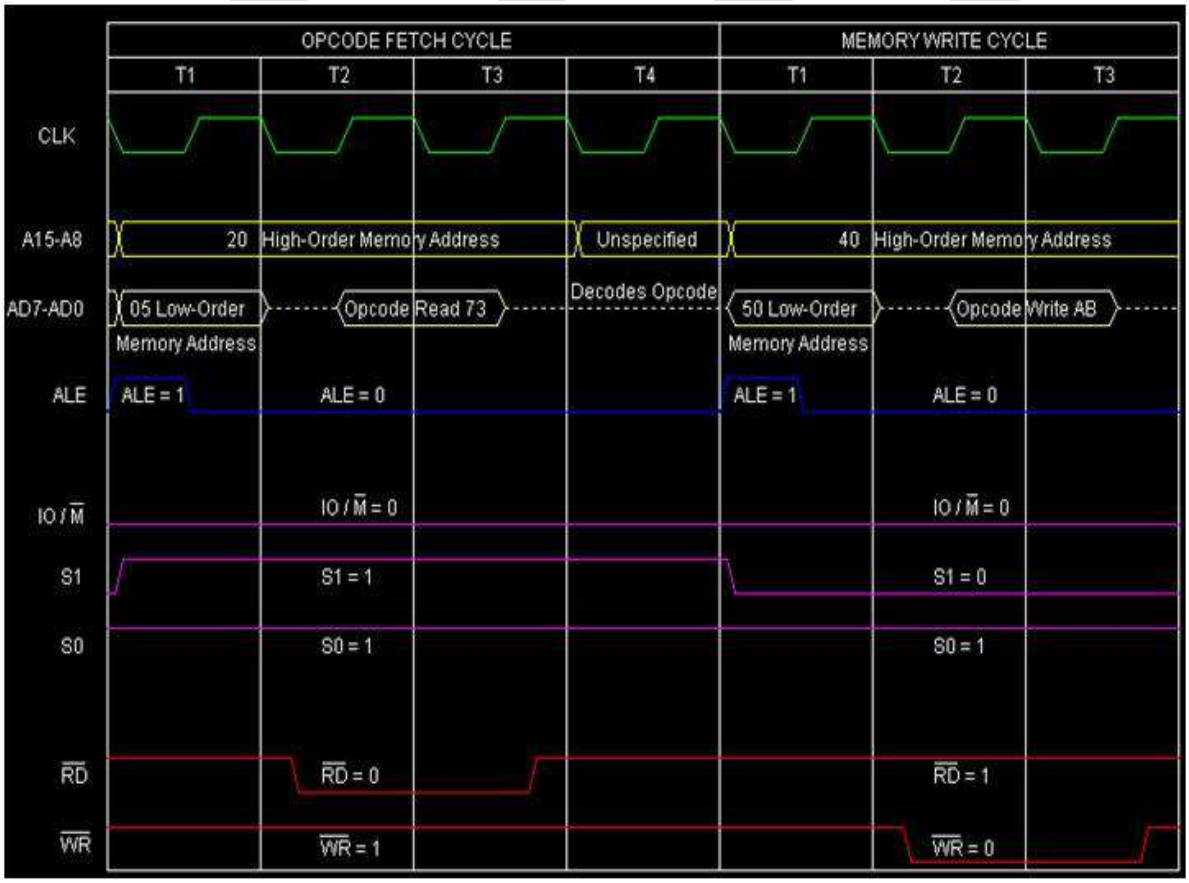
## Timing Diagram of **MOV M, $R_a$** instruction

**MOV M, E** is an example instruction of this type. It is a 1-Byte instruction. Let us suppose, E is having the initial value ABH, HL register pair is pointing to the memory location 4050H, 4050H memory location's content is CDH. Then after execution of the instruction MOV M, E, E register's content will be ABH. The result of execution of this instruction is shown below –

| | **Before** | **After** |
|---|---|---|
| **(E)** | ABH | ABH |
| **(HL)** | (4050H) | (4050H) |
| **(4050H)** | CDH | ABH |

| **Address** | **Hex Codes** | **Mnemonic** | **Comment** |
|---|---|---|---|
| 2005 | 273 | MOV M, E | Memory location pointer by HL register pair <- E |

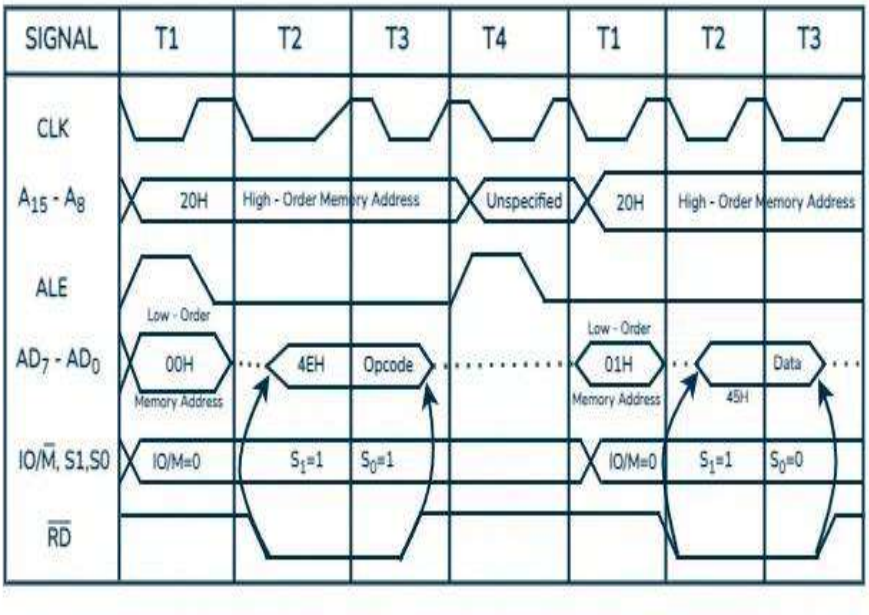
Here is the timing diagram of the instruction **MOV M, E** as below.



## Timing Diagram of MVI A,45H instruction

- Loading opcode 4EH from memory 2000H. (Operation Code Loading Machine Cycle)
- Read (move) data 45H from memory 2001H. (memory reading).

| Address | Mnemonics | Opcode |
|---|---|---|
| 2000H | MVI A,45H | 4EH |
| 2001H | | 45H |

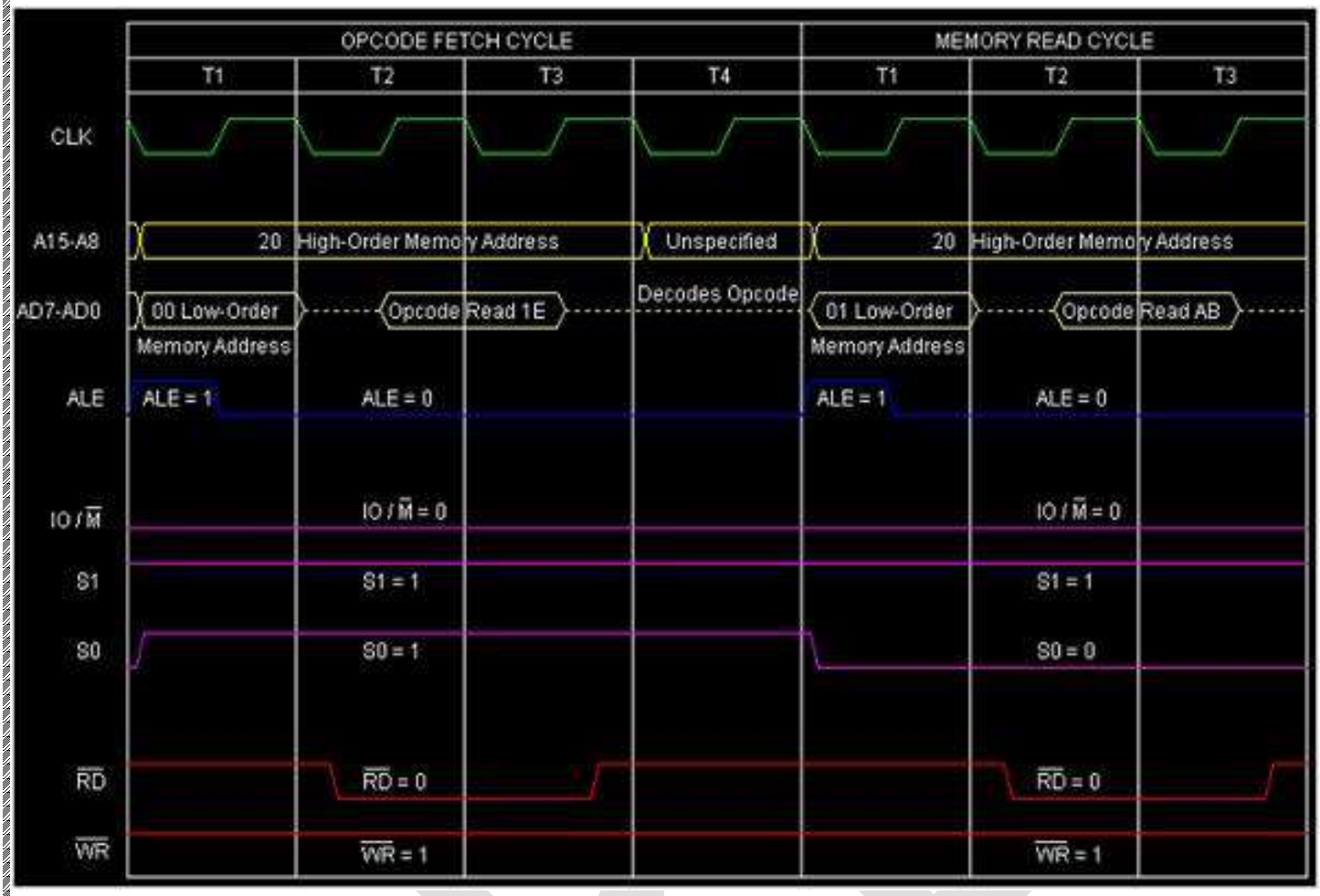



OR

**MVI E, ABH** then it means that **ABH** will be moved or copied to the register **E**. And as a result the previous value of E will get over written.

| Address | Hex Codes | Mnemonic | Comment |
|---|---|---|---|
| 2000 | 1E | MVI E, ABH | E <- ABH |
| 2001 | AB | | ABH as operand |

This instruction will have seven T-states as shown below.




## Timing Diagram of **LDA** instruction

**LDA** is a mnemonic that stands for Load Accumulator with the contents from memory. It occupies 3-Bytes in the memory. First Byte specifies the opcode, and the successive 2-Bytes provide the 16-bit address, i.e. 1-Byte each for each memory location.

| **Mnemonics, Operand** | **Opcode(in HEX)** | **Bytes** |
|---|---|---|
| LDA Address | 3A | 3 |

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.

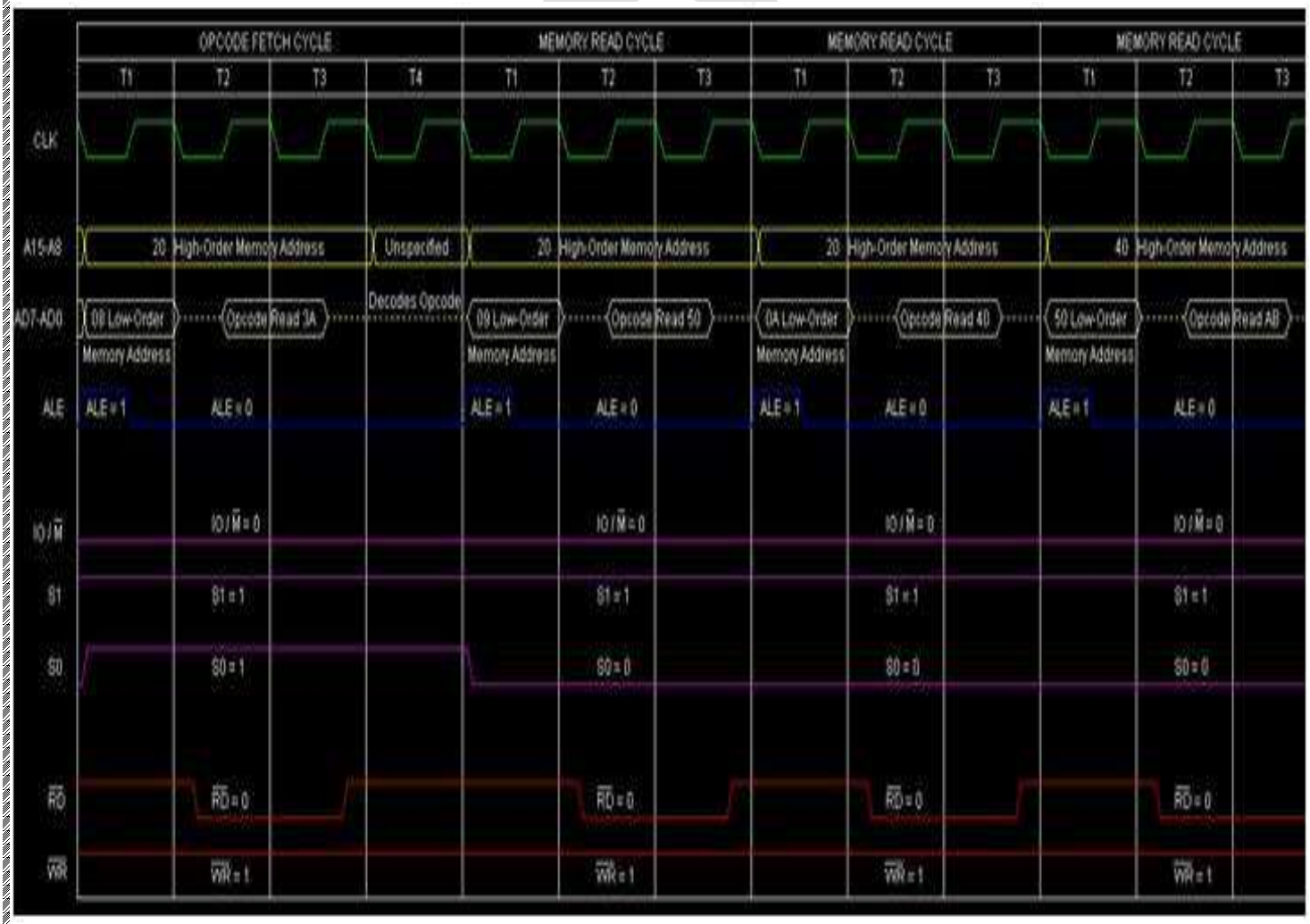
The results of execution of this instruction is as below −

| | **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 |

Here is the timing diagram of the instruction **LDA 4050H**

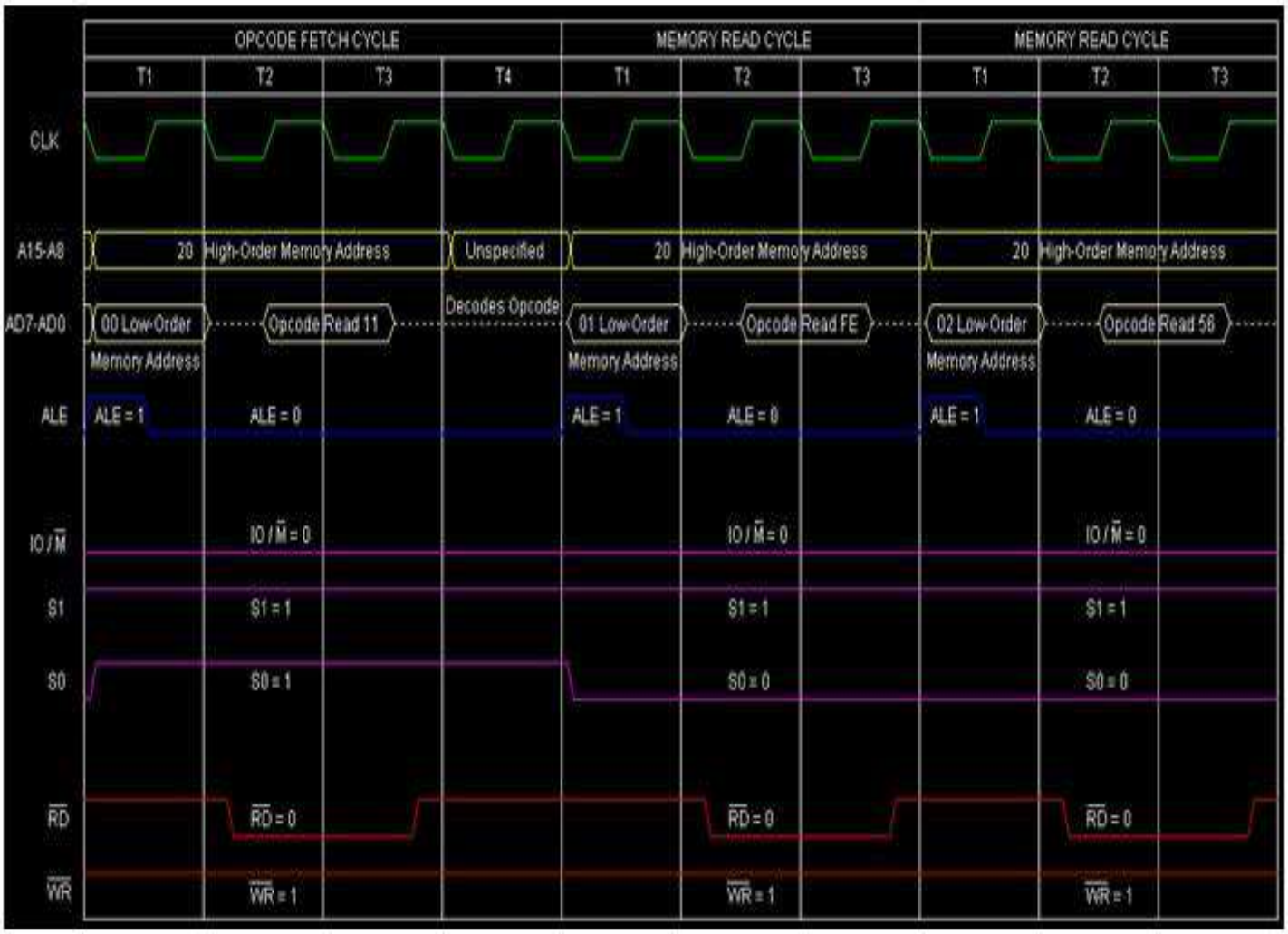



## Timing Diagram of **LXI**

LXI instruction can load 16-bit data into the Register Pairs.

Following is the timing diagram of the instruction **LXI D, 56FEH**

Load 56FE into the DE register pair and store the content of DE register pair into the location 8050 and 8051H.




Timing Diagram of **STA**

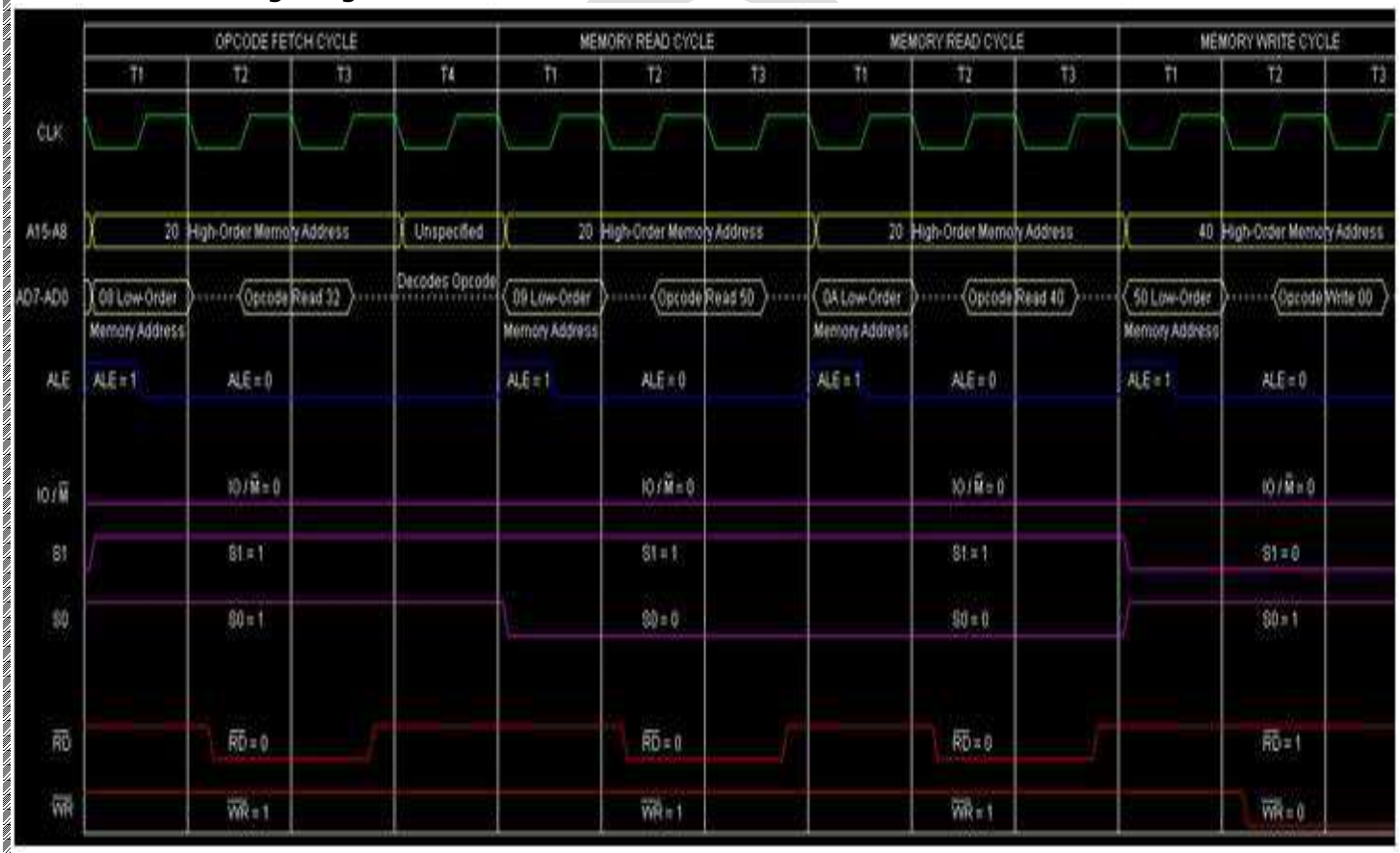
**STA 4050H** as an example instruction of this type. It is a 3-Byte instruction. The first Byte will contain the opcode hex value 32H. As in 8085 assembly language coding supports low order Byte of the address should be mentioned at first then the high order Byte of the address should be mentioned next. 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

| 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 |

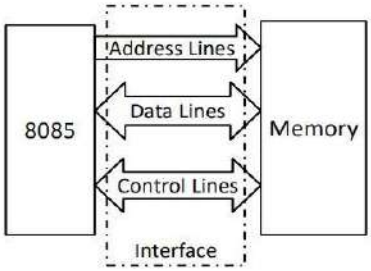
The timing diagram of this instruction **STA 4050H** is as follows –



# UNIT-4: MICROPROCESSOR BASED SYSTEM DEVELOPMENT AIDS

## 4.1 Concept of interfacing

The technique of connecting peripherals to the microprocessor is called interfacing.
OR
Interfacing involves connecting the microprocessor to external devices to exchange data and control signals.
The basic components involved in interfacing are ports, buffers, control lines.




## 4.2 Define Mapping &Data transfer mechanisms - Memory mapping & I/O Mapping

Interfacing I/O Devices

- Using I/O devices data can be transferred between the microprocessor and the outside world.
- This can be done in groups of 8 bits using the entire data bus. This is called parallel I/O.
- The other method is serial I/O where one bit is transferred at a time using the SID and SOD pins on the Microprocessor.
- There are two ways to interface 8085 with I/O devices in parallel data transfer mode:
  - Memory Mapped IO
  - IO Mapped IO

Interfacing is of two types, memory interfacing and I/O interfacing.

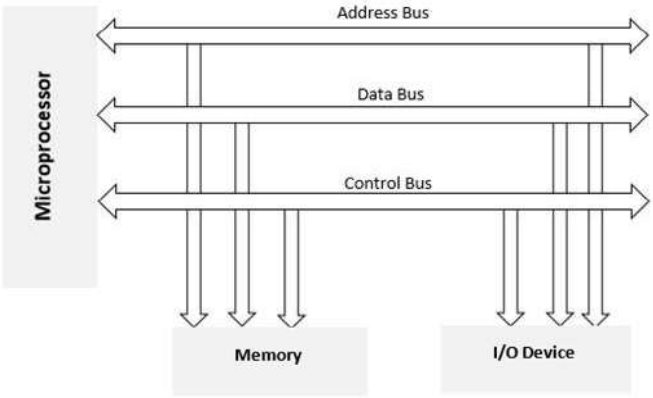
### Memory Interfacing

- When any instruction is executing, microprocessor access the memory for reading instruction codes and the data stored in the memory. For this, both the memory and the microprocessor requires some signals to read from and write to registers.
- The interfacing process includes some key factors to match with the memory requirements and microprocessor signals. The interfacing circuit therefore should be designed in such a way that it matches the memory signal requirements with the signals of the microprocessor.

### IO Interfacing

- There are various communication devices like the keyboard, mouse, printer, etc. So, we need to interface the keyboard and other devices with the microprocessor by using latches and buffers. This type of interfacing is known as I/O interfacing.

## Block Diagram of Memory and I/O Interfacing




# Memory Mapped IO and IO Mapped IO in 8085

| Memory Mapped IO | IO Mapped IO |
|---|---|
| ❖ Here IO devices treated as Memory | ❖ Here IO devices treated as IO. |
| ❖ 16 bits addressing ($A_0 - A_{15}$) | ❖ 8 bits addressing ($A_0 - A_7$) |
| ❖ It can address = $2^{16}$ = 64K Address | ❖ It can address = $2^8$ = 256 Address |
| ❖ Number of devices = 64K = 65536 | ❖ Number of devices = 256 |
| ❖ Due to many devices we need more decoder hardware in Memory Mapped IO. | ❖ Due to less devices we need less decoder hardware in IO Mapped IO. |
| ❖ Due to limited Address lines (16) with 8085 available memory less with memory mapped IO. | ❖ Available memory more with IO mapped IO as number of devices are less in IO mapped IO. |
| ❖ Here we use $\overline{MEMR}$ and $\overline{MEMW}$ control signals. | ❖ Here we use $\overline{IOR}$ and $\overline{IOW}$ control signals. |
| ❖ Data transfer happens between any registers and IO. | ❖ Data transfer only between Accumulator and IO. |
| ❖ ALU operation is done with all registers. | ❖ Not Available. |
| ❖ Instructions Example: LDA XXXXH, STA XXXXH, MOV A,M | ❖ Instructions Example: IN XXXXH, OUT XXXXH |

## 8085 Interfacing Pins

Following is the list of 8085 pins used for interfacing with other devices –

- $A_{15}$ - $A_8$ (Higher Address Bus)
- $AD_7$ - $AD_0$(Lower Address/Data Bus)
- ALE
- RD
- WR
- READY

## 4.3 Concept of Memory Interfacing:- Interfacing EPROM & RAM Memories

- The programs and data that are executed by the microprocessor have to be stored in ROM/EPROM and RAM, which are basically semiconductor memory chips.
- The programs and data that are stored in ROM/EPROM are not erased even when power supply to the chip is removed. Hence, they are called non-volatile memory. They can be used to store permanent programs.

- In a RAM, stored programs and data are erased when the power supply to the chip is removed. Hence, RAM is called volatile memory.
- Microprocessor 8085 can access 64Kbytes memory since address bus is 16-bit. But it is not always necessary to use full 64Kbytes address space. The total memory size depends upon the application.
- Generally, EPROM (or EPROMs) is used as a program memory and RAM (or RAMs) as a data memory. When both, EPROM and RAM are used, the total address space 64Kbytes is shared by them.
- The capacity of program memory and data memory depends on the application.
- It is not always necessary to select 1 EPROM and 1 RAM. We can have multiple EPROMs and multiple RAMs as per the requirement of application.
- EPROM/RAM can place anywhere in full 64 Kbytes address space. But program memory (EPROM) should be located from address 0000H since reset address of 8085 microprocessor is 0000H.
- It is not always necessary to locate EPROM and RAM in consecutive memory For example: If the mapping of EPROM is from 0000H to OFFFH, it is not must to locate RAM from 1000H. We can locate it anywhere between 1000H and FFFFH. Where to locate memory component totally depends on the application.

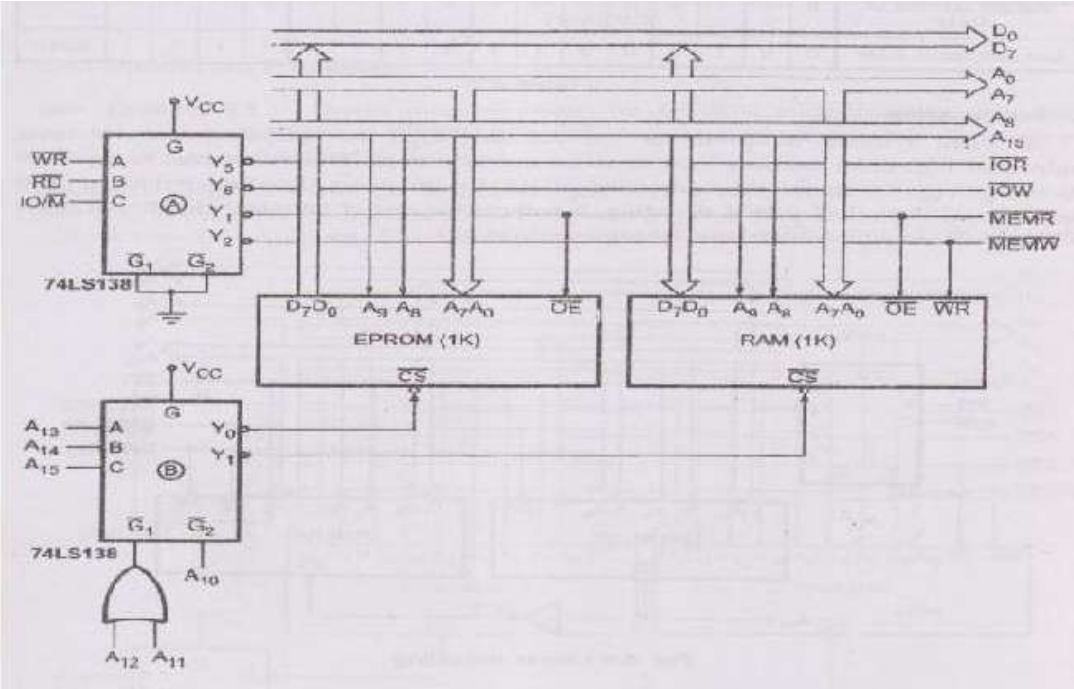
## 4.4 Concept of Address decoding for I/O devices

- A decoder is a logic circuit that accepts a set of inputs that represents a binary number and activates only the output that corresponds to the input number.
- Microprocessor can communicate (read/write) with only one device at a time. Since the data, address and control buses are common for all the devices. Decoding is necessary to communicate with devices (Memory/I/O). The common address decoding techniques are

  - Absolute decoding/ Full Decoding
  - Linear decoding / Partial Decoding

- Absolute Decoding:
- All the higher address lines are decoded to select the memory chip, and the memory chip is selected only for the specified logic level on these high-order address, no other logic levels can select the chip.
- The following figure shows the memory interface with absolute decoding. This addressing technique is normally used in large memory systems.

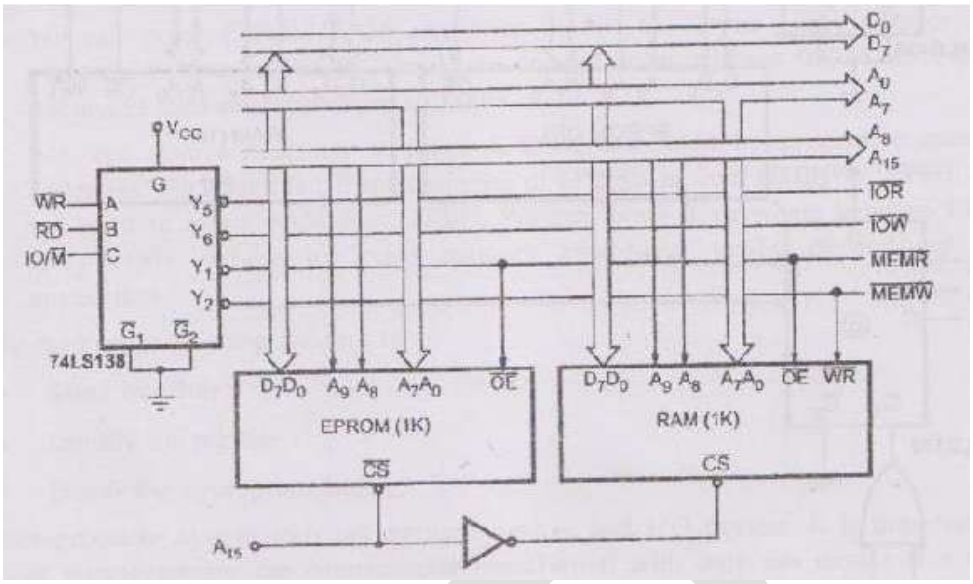
## Absolute Decoding:




| Memory ICs | $A_{15}$ | $A_{14}$ | $A_{13}$ | $A_{12}$ | $A_{11}$ | $A_{10}$ | $A_9$ | $A_8$ | $A_7$ | $A_6$ | $A_5$ | $A_4$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ | Address |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Starting address of EPROM | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0000H |
| End address of EPROM | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 03FFH |
| Starting address of RAM | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2000H |
| End address of RAM | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 23FFH |

## Linear Decoding:

- In small systems, h/w for the decoding logic can be eliminated by using individual high-order address lines to select memory chips.
- This is referred to as linear decoding.
- The figure below shows the addressing of RAM with linear decoding technique.
- This technique is also called partial decoding.
- It reduces the cost of the decoding cct., but it has a drawback of multiple address (shadow addresses)




| Memory ICs | $A_{15}$ | $A_{14}$ | $A_{13}$ | $A_{12}$ | $A_{11}$ | $A_{10}$ | $A_9$ | $A_8$ | $A_7$ | $A_6$ | $A_5$ | $A_4$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ | Address |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Starting address of EPROM | 0 | X | X | X | X | X | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0000H |
| End address of EPROM | 0 | X | X | X | X | X | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 03FFH |
| Starting address of RAM | 1 | X | X | X | X | X | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8000H |
| End address of RAM | 1 | X | X | X | X | X | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 83FFH |

# Absolute and Partial Decoding in 8085

- With Absolute decoding, we use all the address lines [A0-A7] for decoding of IO devices.
- With Partial decoding, few address lines are not used for decoding IO devices.
- With Absolute decoding, we have fixed address for decoding IO.
- With Partial decoding, multiple address may decode IO.
- Absolute and partial decoding may happen in case of Memory interface.

- Example of Absolute decoding.
- Port address is there with IN and OUT instruction as per all A0 to A7 lines.
- 5CH = 0101 1100 b
- Here, A7, A5, A1, A0 = 0 and A6, A4, A3, A2 = 1.
- So, device has fixed address.

- Example of Partial decoding.
- Port address is there with IN and OUT instruction as A0 to A7 lines, but few address lines are not used.
- 0101 11XX b = 5CH, 5DH, 5EH and 5FH.
- Here, A1 and A0 are unused, so total four address may decode given IO.

## 4.5 Programmable Peripheral Interface: 8255

- 8255 is a general purpose programmable device used for data transfer between processor and I/O devices.

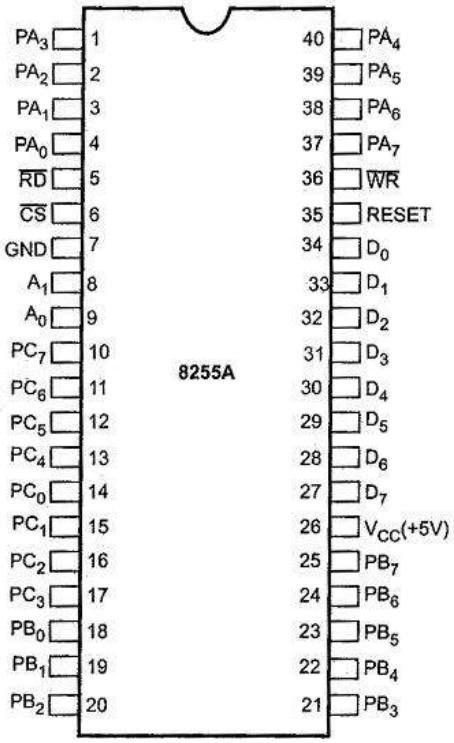
Or

- The 8255A is a general purpose programmable I/O device designed to transfer the data from I/O to interrupt I/O under certain conditions as required.
- It has three 8-bit bidirectional I/O ports (PA, PB &PC) and port operation (IN/OUT Port) is defined by control word in the control word register.
- Ports are operated in two modes:
  - i) I/O modes: Mode 0, Mode 1 & Mode 2
  - Ii) BSR (Bit set/Reset) mode

### 8255 Pin Configuration:

PPI has 40 pins and it has three distinct modes of operation.

- Port A (PA7-PA0) :8 pins
- Port B (PB7-PB0) :8 pins
- Port C (Pc: Upper: PC7-PC4) : 4 pins
- Port C (Pc: Lower: PC3-PC)) : 4 pins
- Data Bus (D7-D0) : 8 pins
- Control signals : 6 pins
- VCC and Gnd : 2 pins



PA7-PA0:

- These are eight port A lines that acts as either latched output or buffered input lines depending upon the control word loaded into the control word register.

PB0-PB7:

- These are the eight port B lines which are used as latched output lines or buffered input lines in the same way as port A.

PC0-PC7:

- Port C can be divided into two 4 bit ports namely PC7-PC4 & PC3-PCO and used for control signals to PA and PB

Data Bus (D7-D0):

- These are the data bus lines those carry data or control word to/from the microprocessor.

Control signals:

RD': This is the input line driven by the microprocessor and should be low to indicate read operation to 8255.

WR': This is an input line driven by the microprocessor. A low on this line indicates write operation.

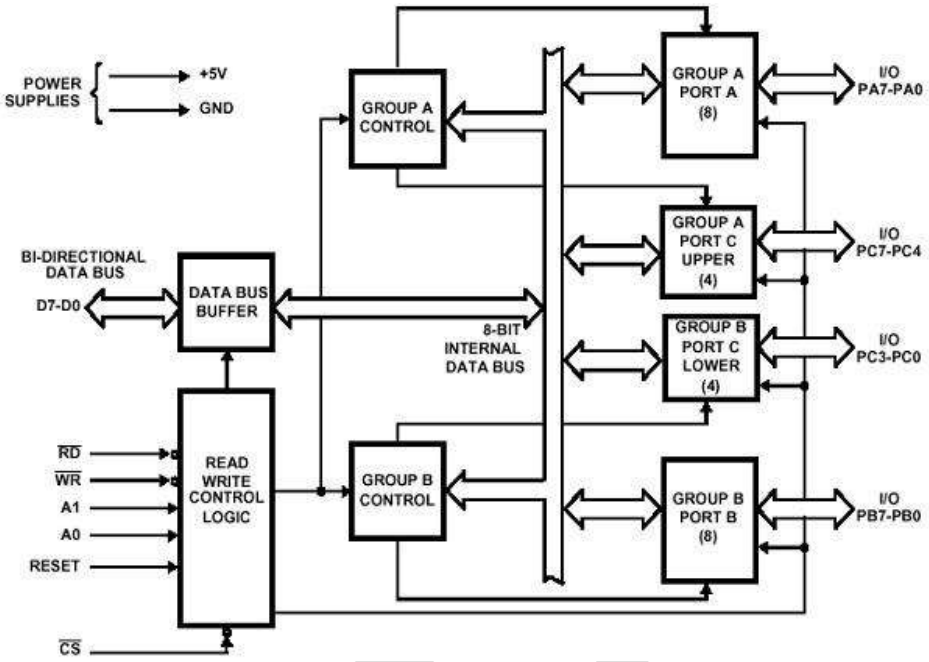
CS': The CS' is a chip select input pin which is responsible for selecting a chip. A low signal at CS' pin simply allows the communication between the 8255 & the processor.

A0 & A1: It simply make a decision about which port will be preferred for transmitting the data.
If A0 = 0 & A1=0 then Port-A is selected.
If A0 = 0 & A1=1 then Port-B is selected.
If A0 = 1 & A1=0 then Port-C is selected.
If A0 = 1 & A1=1 then the control register is selected.
RESET: Logic high on this line clears the control word register of 8255. All ports are set as input ports by default after reset.

VCC and Gnd:
5V input power supply and ground.

## 8255 Architecture

The block diagram of INTEL 8255 contains:
a) Data bus buffer
b) Read/write control logic
c) Group A, and group B control
d) Port A, Port B, Port C




### a) Data Bus Buffer

- It is a tri-state 8-bit buffer, which is used to interface the microprocessor to the system data bus.
- Data is transmitted or received by the buffer as per the instructions by the CPU.
- Control words and status information is also transferred using this bus.

### b) Read/Write Control Logic

- The function of this block is to control the internal operation of the device and to control the transfer of data and control or status words.
- It accepts inputs from the CPU address and control buses and in turn issues command to both the control groups.
- CS': It stands for Chip Select. A LOW on this input selects the chip and enables the communication between the 8255A and the CPU. It is connected to the decoded address, and $A_0$ & $A_1$ are connected to the microprocessor address lines.

Their result depends on the following conditions −

| **CS'** | **$A_1$** | **$A_0$** | **Result** |
|---|---|---|---|
| 0 | 0 | 0 | PORT A |
| 0 | 0 | 1 | PORT B |
| 0 | 1 | 0 | PORT C |
| 0 | 1 | 1 | Control Register |
| 1 | X | X | No Selection |

- WR': It stands for write. This control signal enables the write operation. When this signal goes low, the microprocessor writes into a selected I/O port or control register.
- RD': It stands for Read. This control signal enables the Read operation. When the signal is low, the microprocessor reads the data from the selected I/O port of the 8255.
- RESET: This is an active high signal. It clears the control register and sets all ports in the input mode.

### c) Group A and Group B control:

- Group A and B get the Control Signal from CPU and send the command to the individual control blocks.
- Group A send the control signal to port A and Port C (Upper) PC7-PC4.
- Group B send the control signal to port B and Port C (Lower) PC3-PC0.

### d) Port A, Port B & Port C

- Port A & Port B includes an 8-bit input latch and 8-bit buffered or latched output. The main function of these ports is also independent of the mode of operation.
- Port A can be programmed in 3 modes like modes 0, 1, and 2 whereas Port B can be programmed in modes 0 & mode 1.
- Port C includes an 8-bit data input buffer and 8-bit bidirectional data o/p latch or buffer. This port is divided mainly into two sections – port C upper PCU & port C lower PCL.

## 8255 Microprocessor Operating Modes

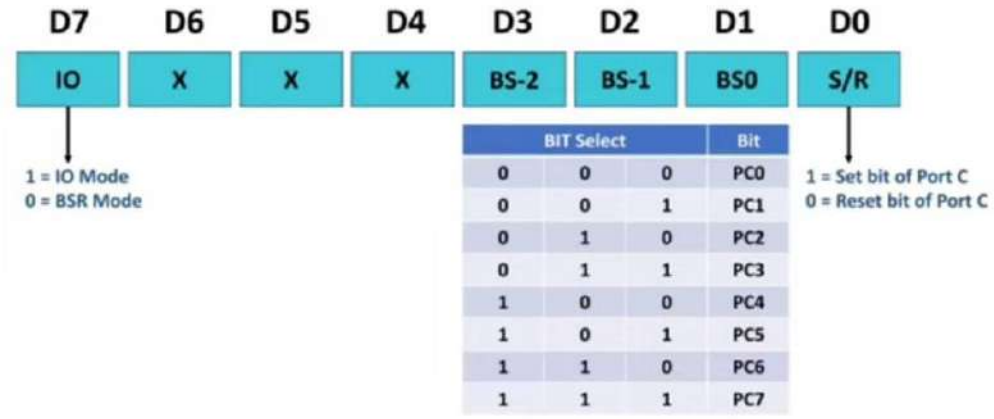
There are two basic operational modes of 8255:

- Bit set/reset Mode (BSR Mode).
- Input/output Mode (I/O Mode).

The two modes are selected on the basis of the value present at the D7 bit of the Control Word Register. When D7 = 1, 8255 operates in I/O mode and when D7 = 0, it operates in the BSR mode.

### Bit Set-Reset Mode

If MSB of control word (D7) is 0, PPI works in BSR mode. In this mode only port C bits are used for set or reset.




| BIT Select | | | Bit |
|---|---|---|---|
| 0 | 0 | 0 | PC0 |
| 0 | 0 | 1 | PC1 |
| 0 | 1 | 0 | PC2 |
| 0 | 1 | 1 | PC3 |
| 1 | 0 | 0 | PC4 |
| 1 | 0 | 1 | PC5 |
| 1 | 1 | 0 | PC6 |
| 1 | 1 | 1 | PC7 |

- D7 bit is always 0 for BSR mode.
- Bits D6, D5 and D4 are don't care bits.
- Bits D3, D2 and D1 are used to select the pin of Port C.
- Bit D0 is used to set/reset the selected pin of Port C.

### Input-Output mode

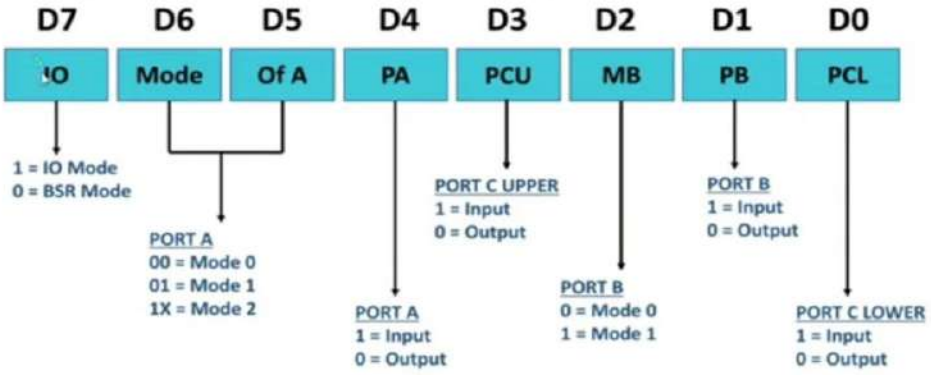
If MSB of control word (D7) is 1, PPI works in input-output mode. This is further divided into three modes:

- Mode 0 - Simple I/O
- Mode 1 - Handshake or strobbed I/O
- Mode 2 - Bi-directional I/O

**Mode 0 –**In this mode all the three ports (port A, B, C) can work as simple input function or simple output function. In this mode there is no interrupt handling capacity.

**Mode 1 –** Handshake I/O mode or strobed I/O mode. In this mode either port A or port B can work as simple input port or simple output port, and port C bits are used for handshake signals before actual data transmission. It has interrupt handling capacity and input and output are latched.

**Mode 2 –** Bi-directional data bus mode. In this mode only port A works, and port B can work either in mode 0 or mode 1. 6 bits port C are used as handshake signals. It also has interrupt handling capacity.




The common applications of 8255 are:

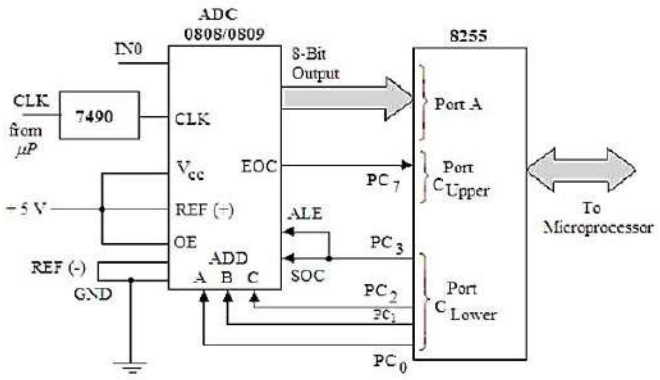
- Traffic light control
- Generating square wave
- Interfacing with DC motors and stepper motors

## 4.6 ADC & DAC with Interfacing.

### ADC

- The Analog to Digital Conversion is a quantizing process. Here the analog signal is represented by equivalent binary states.
- The ADC 0808/0809 is an 8-bit analog to digital converter. It has 8 channel multiplexer to interface with the microprocessor.

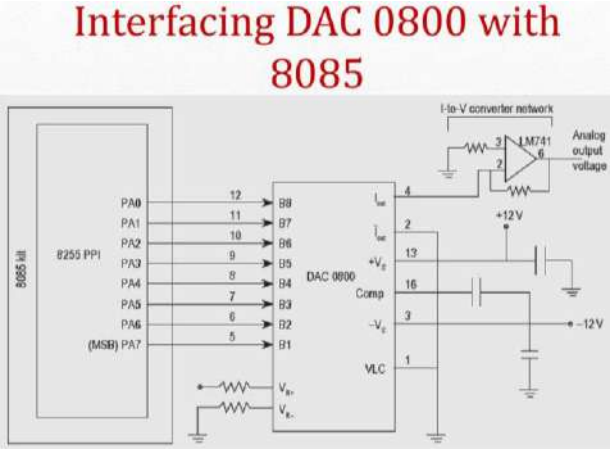
- ADC 0808/0809 is a monolithic CMOS device. This device uses successive approximation technique to convert analog signal to digital form. One of the main advantage of this chip is that it does not require any external zero and full scale adjustment, only +5V DC supply is sufficient.
- To interface the ADC with 8085, it need 8255 Programmable Peripheral Interface chip with it.




- The PortA of 8255 chip is used as the input port.
- The $PC_7$ pin of Port $C_{upper}$ is connected to the End of Conversion (EOC) Pin of the analog to digital converter. This port is also used as input port.
- The $C_{lower}$ port is used as output port. The $PC_{2\text{-}0}$ lines are connected to three address pins of this chip to select input channels. The $PC_3$ pin is connected to the Start of Conversion (SOC) pin and ALE pin of ADC 0808/0809.

## DAC

- The Digital to Analog converter (DAC) is a device, that is widely used for converting digital pulses to analog signals.
- Digital to Analog converter are used to get a proportional analog voltages or current for the digital data given out by the microprocessor.
- DAC 0800 is a common digital to analog converter chip that can be easily interfaced to 8085 through 8255.
- As the DAC chip can only be connected to an output port of a processor, the 8255 PPI is essential in the interface. Any one port is enough to interface am 8 bit DAC with 8255.
- The other control signals are directly correspondingly connected to either logic 0 or logic 1. The DAC chip gives a proportional current output.
- This current output in most cases is difficult to measure and so a current to voltage (I to V) converter is used at the output. DAC chips have an inbuilt latch
- This latch stores the digital input given by the port A and gives out a proportional voltage.




## 4.7 Interfacing Seven Segment Displays

- 7-segment display consists of a few LEDs and are arranged physically as shown in figure.
- It has seven segment from A to G that normally connected to data bus D0 to D6 respectively.
- If decimal point is used, D7 will be connected to DP; and left unconnected if it is unused.
- There are two different type of 7-segment display; common cathode and common anode.

  Common Anode – 0 will make an LED glow.

  Common Cathode – 1 will make an LED glow.
- Fig. shows the example to interface seven segment display and address decoder with an address of FDH.
- The common anode display is used therefore 0 logic is needed to activate the segment.
- Suppose to display number 4 at seven segment display, therefore the segment F, G, B and C have to be activated.

- Follows are the instructions to execute it: –

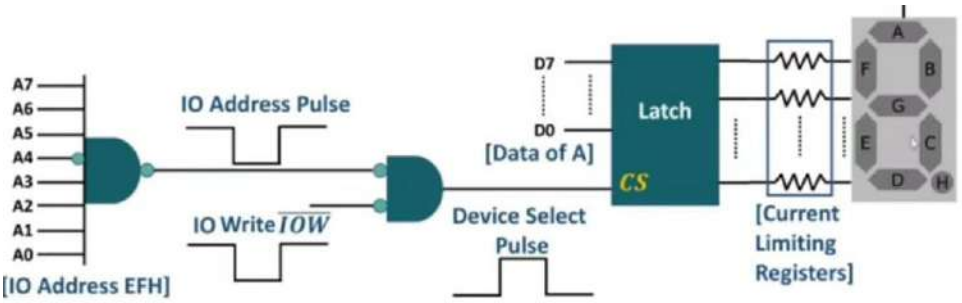
MVI A, 66H : For 7 display on seven segment display, A=66H
OUT FDH : Accumulator will get displayed at FDH port
HLT : Terminate program

Data lines: D7 D6 D5 D4 D3 D2 D1 D0

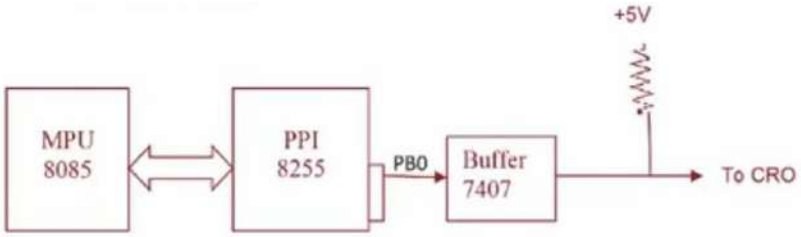
Bits: X 1 1 0 0 1 1 0 = 66 H

Segments: NC G F E D C B A

Vcc




## 4.8 Generate square waves on all lines of 8255




```
        LXI SP, 6000      ; Initialize stack pointer to use CALL
        MVI A, 98         ; Control word for 8255
        OUT 83            ; Load the control word into 8255
Loop:   MVI A, 00         ; Data for LOW pulse
        OUT 81            ; Send port B (PB0) as LOW
        CALL delay1       ; Set LOW pulse duration for LOW.
        MVI A, 01         ; Data for HIGH pulse
        OUT 81            ; Send port B (PB0) as HIGH
        CALL delay2       ; Pulse duration for HIGH.
        JMP loop          ; Repeat for continuous generation of square wave.
```

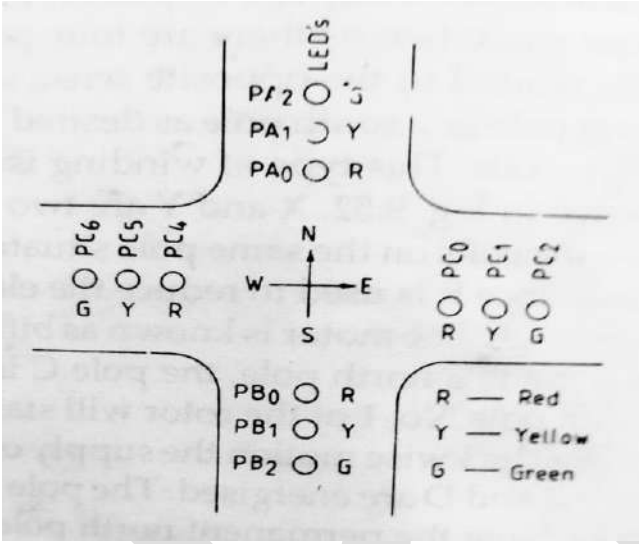
SUBROUTINE

```
delay1  MVI B, 10         ; Stores the count value in B register
loop:   DCR B             ; Count value is decremented by one
        JNZ Loop          ; If the count is not zero, go to Loop
        RET               ; If the count is zero, return to main
                            program.

Delay2  MVI C, 10         ; Stores the count value in C register
loop:   DCR C             ; Count value is decremented by one
        JNZ Loop          ; If the count is not zero, go to Loop
        RET               ; If the count is zero, return to main
```

## 4.9 Design Interface a traffic light control system using 8255.

- The figure shows a simple arrangement and port connections for microprocessor based traffic control.
- All ports of 8255 have been programmed as output ports.
- The control word to make all ports output ports in Mode 0 operation is 80H.
- The connection of pins of the ports to LED have been made through buffers (7407).
- Positive logic has been used to switch on LEDs. Three types of LEDs have been used red, yellow and green.
- Green light glows to allow crossing, yellow to make alert and red does not allow crossing.




**PROGRAM**

| *Memory address* | *Machine Codes* | *Lables* | *Mnemonics* | *Operands* | *Comments* |
|---|---|---|---|---|---|
| FC00 | 3E, 80 | | MVI | A, 80 H | Get control word for 8255. |
| FC02 | D3, 0B | | OUT | 0B | Initialize ports of 8255.2. |
| FC04 | 3E, 01 | LOOP | MVI | A, 01 | |
| FC06 | D3, 09 | | OUT | 09 | Red ON for south. |
| FC08 | D3, 08 | | OUT | 08 | Red ON for north. |
| FC0A | 3E, 44 | | MVI | A, 44 | Green ON for east and west. |
| FC0C | D3, 04 | | OUT | 0A | |
| FC0E | CD, 00, FD | | CALL | DELAY I | |
| FC11 | 3E, 22 | | MVI | A, 22 | Yellow ON for east and west. |
| FC13 | D3, 0A | | OUT | 0A | |
| FC15 | 3E, 02 | | MVI | A, 02 | |

| | | | | | |
|---|---|---|---|---|---|
| FC17 | D3, 09 | | OUT | 09 | Yellow ON for south |
| FC19 | D3, 08 | | OUT | 08 | Yellow ON for north. |
| FC1B | CD, 13, FD | | CALL | DELAY II | |
| FC1E | 3E, 11 | | MVI | A, 11 | |
| FC20 | D3, 0A | | OUT | 0A | Red ON for east and west. |
| FC22 | 3E, 04 | | MVI | A, 04 | |
| FC24 | D3, 08 | | OUT | 08 | Green ON for north. |
| FC26 | D3, 09 | | OUT | 09 | Green ON for south. |
| FC28 | CD, 00, FD | | CALL | DELAY I | |
| FC2B | 3E, 22 | | MVI | A, 22 | Yellow ON for east and west. |
| FC2D | D3, 0A | | OUT | 0A | |
| FC2F | 3E, 02 | | MVI | A, 02 | |
| FC31 | D3, 09 | | OUT | 09 | Yellow ON for south. |
| FC33 | D3, 08 | | OUT | 08 | Yellow ON for north. |
| FC35 | CD, 13, FD | | CALL | DELAY II | |
| FC38 | C3, 04, FC | | JMP | LOOP | |
| **DELAY I** | | | | | |
| FD00 | 06, 20 | | MVI | B, 20 H | |
| FD02 | 0E, FF | G03 | MVI | C, FF | |
| FD04 | 16, FF | G02 | MVI | D, FF | |
| FD06 | 15 | G01 | DCR | D | |
| FD07 | C2, 06, FD | | JNZ | G01 | |
| FD0A | 0D | | DCR | C | |
| FD0B | C2, 04, FD | | JNZ | G02 | |
| FD0E | 05 | | DCR | B | |
| FD0F | C2, 02, FD | | JNZ | G03 | |
| FD12 | C9 | | RET | | |
| **DELAY II** | | | | | |
| FD13 | 06, 10 | | MVI | B, 10 | |
| FD15 | C3, 02, FD | | JMP | FD02 | G03 |

## 4.10 Design interface for stepper motor control using 8255.

- A stepper motor is a device that translates electrical pulses into mechanical movement in steps of fixed step angle.
- The stepper motor rotates in steps in response to the applied signals.
- It is used in disk drives, dot matrix printers, plotters and robotics and process control circuits.

In this type of functioning, the following 4 binary sequence/code are used for rotation

| 4- Step sequence binary pattern | HEX code | Comments | | | |
|---|---|---|---|---|---|
| A | B | C | D | | |
| 1 | 0 | 1 | 0 | 0AH | Sequence for Clock wise rotation |
| 1 | 0 | 0 | 1 | 09H | |
| 0 | 1 | 0 | 1 | 05H | |
| 0 | 1 | 1 | 0 | 06H | |
| | | | | | |
| 0 | 1 | 1 | 0 | 06H | Sequence for anti-clockwise rotation |
| 0 | 1 | 0 | 1 | 05H | |
| 1 | 0 | 0 | 1 | 09H | |
| 1 | 0 | 1 | 0 | 0AH | |

**Chips select Logic:**

| Chip select address lines | Address lines to select port | HEX address | Selected I/O | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 | | |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 80H | PORT A |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 81H | PORT B |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 82H | PORT C |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 83H | Chip select register |

**Control word Format:**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| IO/BSR | MA | MA | PA | PCU | MB | PB | PCU |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

=80H

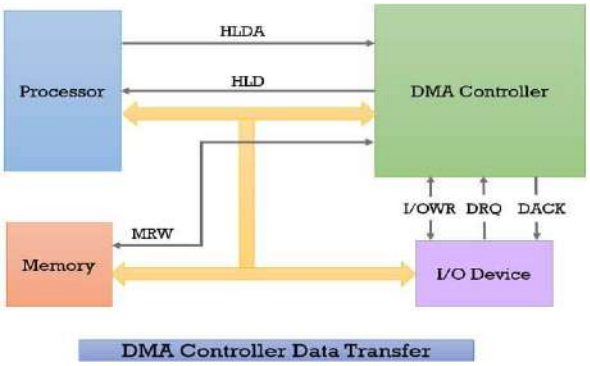
**Program:**

| LABEL | OPCODE | OPERAND | COMMENT |
|---|---|---|---|
| | LXI | SP,2800H | Initialize Stack pointer |
| | MVI | A, 80H | Initialize 8255 |
| | OUT | 83H (CWR) | |
| | MVI | B, 32H | Initialize repeated count |
| REPEAT: | LXI | H, 2100H | Initialize 4-step sequence |
| | MVI | C,04H | Initialize 4-step sequence from look up table |
| BACK: | MOV | A,M | |
| | OUT | 80H (PORT A) | Sends data to Port A |
| | CALL | DELAY | Provide time interval between steps |
| | INX | H | Increment look up table |
| | DCR | C | Decrement 4-step count |
| | JNZ | BACK | Is count= "00"? if no then jump to BACK |
| | DCR | B | Is count= "00"? if yes then decrement repeated count |
| | JNZ | REPEAT | Repeated count is repeated for further rotation |
| | HLT | | |

## 4.11 Basic concept of other Interfacing DMA controller, USART

### DMA controller

- DMA stands for Direct Memory Access. It is designed by Intel to transfer data at the fastest rate.
- It allows the device to transfer the data directly to/from memory without any interference of the microprocessor.
- Using a DMA controller, the device requests the processor to hold its data, address and control bus, so the device is free to transfer data directly to/from the memory.
- The DMA data transfer is initiated only after receiving HLDA signal from the processor.



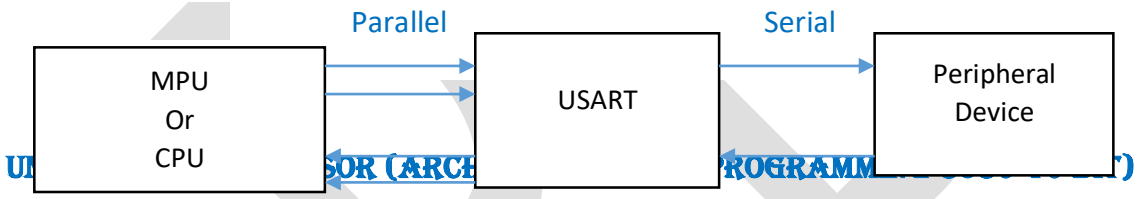
DMA Controller Data Transfer

Following is the sequence of operations performed by a DMA –

- Initially, when any device has to send data between the device and the memory, the device has to send DMA request (DRQ) to DMA controller.
- The DMA controller sends Hold request (HRQ) to the microprocessor and then microprocessor sends an acknowledge signal (HLDA) to the DMA controller.
- On receipt of this signal (HLDA), the DMA controller sends a DMA acknowledge signal (DMACK) to the I/O device.
- The DMA controller then takes over the control of the buses of microprocessor. And it takes control the data transfer between RAM and I/O device.
- When the data transfer is completed, DMA controller returns the control over the buses to the microprocessor by disabling the HOLD and DMACK signals.

USART

- The 8251 chip is Universal Synchronous Asynchronous Receiver Transmitter (USART).
- It acts as a mediator between the microprocessor and peripheral devices.
- It converts serial data to parallel form and vice versa.
- It takes data serially from peripheral (outside devices) and converts into parallel data. After converting the data into parallel form, it transmits it to the CPU.
- Similarly, it receives parallel data from microprocessor and converts it into serial form. After converting data into serial form, it transmits it to outside device (peripheral).




- Intel 8086 microprocessor is the enhanced version of Intel 8085 microprocessor.
- It built on a single semiconductor chip and packaged in a 40-pin IC package. The type of package is DIP (Dual Inline Package).
- 8086 uses 20 address lines and 16 data- lines. It can directly address up to $2^{20}$ = 1 Mbyte of memory.
- It consists of a powerful instruction set, which provides operation like division and multiplication very quickly.
- 8086 is designed to operate in two modes, i.e., Minimum and Maximum mode.

| 8085 Microprocessor | 8086 Microprocessor |
|---|---|
| It is an 8-bit microprocessor. | It is a 16-bit microprocessor. |
| It has a 16-bit address line. | It has a 20-bit address line. |
| It has a 8-bit data bus. | It has a 16-bit data bus. |
| The memory capacity is 64 KB. | The memory capacity is 1 MB. |

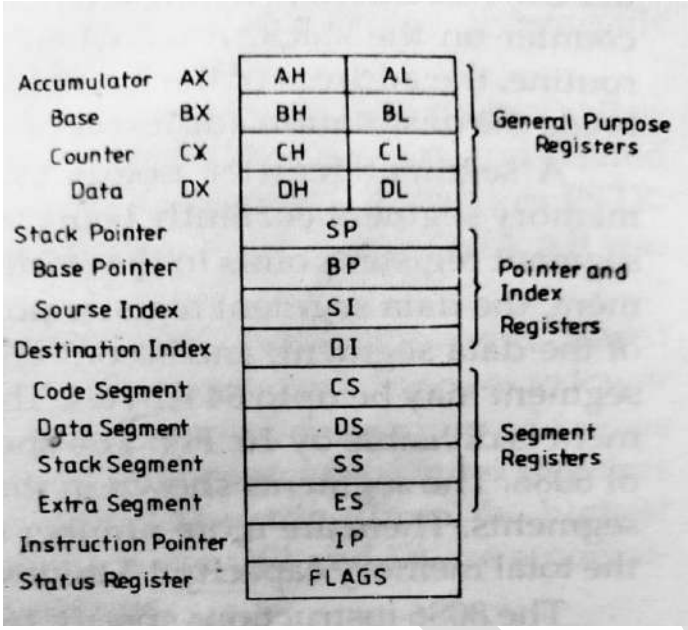
| | |
|---|---|
| The Clock speed of this microprocessor is 3 MHz. | The Clock speed of this microprocessor varies between 5, 8 and 10 MHz for different versions. |
| It has five flags. | It has nine flags. |
| 8085 microprocessor does not support memory segmentation. | 8086 microprocessor supports memory segmentation. |
| It does not support pipelining. | It supports pipelining. |
| It is accumulator based processor. | It is general purpose register based processor. |
| It has no minimum or maximum mode. | It has minimum and maximum modes. |
| In 8085, only one processor is used. | In 8086, more than one processor is used. An additional external processor can also be employed. |
| It contains less number of transistors compare to 8086 microprocessor. | It contains more number of transistors compare to 8085 microprocessor. |
| The cost of 8085 is low. | The cost of 8086 is high. |

Difference between 8085 and 8086 Microprocessor

## 5.1 Register Organisation of 8086

The register organization of the 8086 microprocessor can be divided into four categories:

1. General Purpose Registers
2. Segment Registers
3. Pointers and Index Registers
4. Flag or Status Register




### 1. General Purpose Registers

- General-purpose registers are used to store temporary data within the microprocessor.
- There are 4 general-purpose registers of 16-bit length each. Each of them is further divided into two 8-bit registers.
- Usually L and H specify the lower and higher bytes of a particular register.

i. AX = [AH:AL]
ii. BX = [BH:BL]
iii. CX = [CH:CL]
iv. DX = [DH:DL]

AX register: It is also known as accumulator register. It is used to store one of the operands for arithmetic operations.
BX register: It is used as a base register. It is used to store the starting base address of the memory area within the data segment.
CX register: It is referred to as counter. It is used in loop instruction to store the loop counter.
DX register: This register is used to hold I/O port address for I/O instruction.

### 2. Segment Registers

- The complete 1 Megabyte memory, which the 8086 addresses, is divided into 16 logical segments.
- Each segment thus contains 64 k bytes of memory.
- There are 4 segment registers: - Code segment register (CS) Code, Data segment register (DS) Data, Extra segment register (ES) data, Stack segment register (SS)
  - The CS register (code segment) points to the segment of memory where the currently executing program is stored.

- The DS register (data segment) points to the segment of memory where the program's data is stored.
- The SS register (stack segment) points to the segment of memory where the program's stack is stored.
- The ES register (extra segment) is an additional segment register that can be used for storing data.

Instruction Pointer (IP): The instruction pointer in the 8086 microprocessor acts as a program counter. It indicates to the address of the next instruction to be executed. Apart from this, it also acts as an offset for CS register.

### 3. Pointers and Index Registers

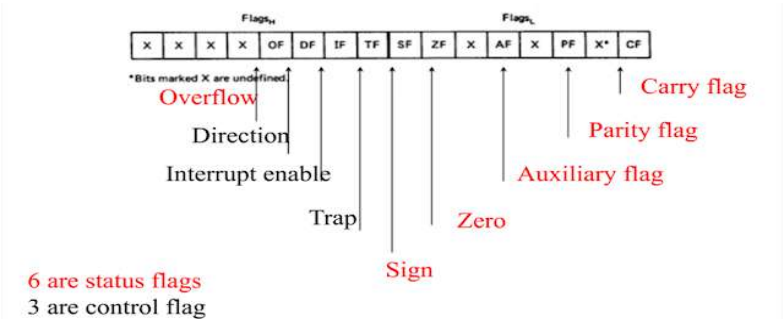
- The pointers will always store some address or memory location. In 8086 Microprocessor, they usually store the offset through which the actual address is calculated.
  - Base Pointer (BP): The Base pointer stores the base address of the memory. Also, it acts as an offset for Stack Segment (SS).
  - Stack Pointer (SP): The Stack Pointer Points at the current top value of the Stack. Like the BP, it also acts as an offset to the Stack Segment (SS).
- The indexes are used with the extra segment and they usually are used for copying the contents of a particular block of memory to a new location.
  - Source Index (SI): It stores the offset address of the source.
  - Destination Index (DI): It stores the offset address of the Destination.

### 4. Flag or Status Register

- The Flag or Status register is a 16-bit register which contains 9 flags, and the remaining 7 bits are idle in this register.
- These flags tell about the status of the processor after any arithmetic or logical operation.
- If the flag value is 1, the flag is set, and if it is 0, it is said to be reset.

There are two types of flags of flag register in 8086 Microprocessor:

1. Condition Flags
2. Control Flags




## 1. Condition Flags

The conditional flags are set or reset after any arithmetic or logical operation is performed on an 8 bit or 16-bit number. This category consists of the following 6 flags:

i. Carry Flag (CF): The carry flag will be set only if a carry is generated from the MSB of the result after doing any operation in 8086 Microprocessor.
ii. Parity Flag (PF): Parity is related to the number of 1's contained in the binary data. There exist two types of parity:
    - Even Parity: When the number of 1's in the binary data are even.
    - Odd Parity: When the number of 1's in the binary data are odd.

For the flag, the PF is set if there exists an even parity in data after the execution of the instruction. Else the flag is reset.

iii. Auxiliary-Carry Flag (AF): This flag is set if there is a generation of carrying from a nibble, i.e. 4 bits of data.
iv. Zero Flag (ZF): If the result after performing the required operation (Arithmetic or Logical) on the instructions is zero, in that case, the zero flags are set to 1. Else, it remains reset.
v. Sign Flag (SF): If the result after performing any arithmetic or logic operation in the given instruction is negative, then the sign flag is set to 1. Else, for a positive result, the sign flag remains reset.
vi. Overflow Flag (OF): This Flag will be set if the register gets overflowed with data after any arithmetic or logic operation. This happens in cases when the carry is getting in in MSB, but there is no space in the register to store the carried out bit.

### 2. Control Flags

The control flags are used to navigate the microprocessor for certain operations. There are 3 types of control flags:

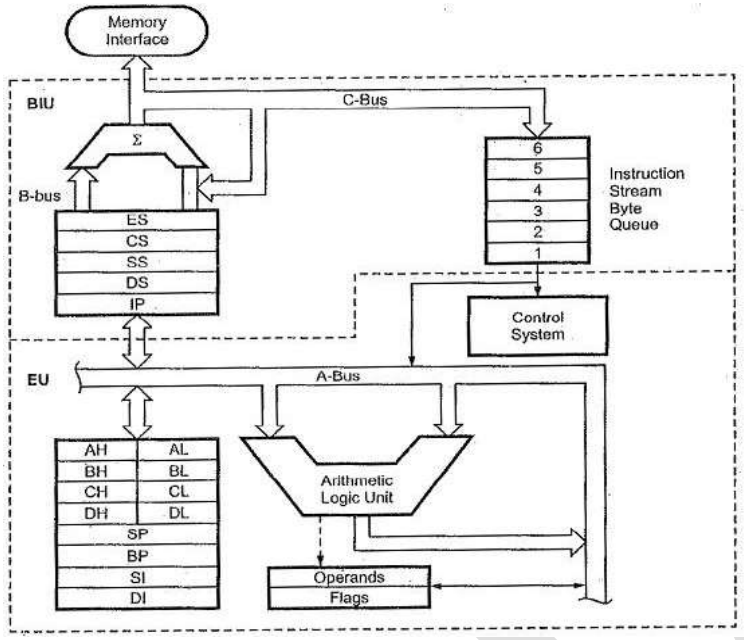
i. Trap Flag (TF): If the TF is set, then the execution will be done step by step. Otherwise, the free-running operation will be done.
ii. Interrupt Flag (IF): This flag is used to enable the Interrupt. The microprocessor is capable of handling interrupts only if this flag is in the set mode. Otherwise, any interrupt raised while the execution of the instructions will not be handled by the microprocessor.
iii. Direction Flag (DF): This flag is used for string operations. If this flag is set, the string will be read from higher-order bits to lower order bits and vice versa.

## 5.2 Internal architecture of 8086

The internal architecture of Intel 8086 is divided into 2 units: The Bus Interface Unit (BIU), and The Execution Unit (EU).

### Bus Interface Unit (BIU)

The segment registers, instruction pointer and 6-byte instruction queue are associated with the bus interface unit (BIU).




## The BIU:

- Handles transfer of data and addresses,
- Fetches instruction codes, stores fetched instruction codes in first-in-first-out register set called a queue**.**
- Reads data from memory and I/O devices,
- Writes data to memory and I/O devices,
- It relocates addresses of operands since it gets un-relocated operand addresses from EU. The EU tells the BIU from where to fetch instructions or where to read data.

It has the following functional parts:

- Instruction Queue: When EU executes instructions, the BIU gets 6-bytes of the next instruction and stores them in the instruction queue and this process is known as instruction pre fetch. This process increases the speed of the processor.
- Segment Registers: A segment register contains the addresses of instructions and data in memory which are used by the processor to access memory locations. It points to the starting address of a memory segment currently being used.
  There are 4 segment registers in 8086 as given below:
  - Code Segment Register (CS): Code segment of the memory holds instruction codes of a program.
  - Data Segment Register (DS): The data, variables and constants given in the program are held in the data segment of the memory.

- Stack Segment Register (SS): Stack segment holds addresses and data of subroutines. It also holds the contents of registers and memory locations given in PUSH instruction.
- Extra Segment Register (ES): Extra segment holds the destination addresses of some data of certain string instructions.
- Instruction Pointer (IP): The instruction pointer in the 8086 microprocessor acts as a program counter. It indicates to the address of the next instruction to be executed.

## Execution Unit (EU)

- The **EU** receives opcode of an instruction from the queue, decodes it and then executes it. While Execution, unit decodes or executes an instruction, then the BIU fetches instruction codes from the memory and stores them in the queue.
- The BIU and EU operate in parallel independently. This makes processing faster.
- General purpose registers, stack pointer, base pointer and index registers, ALU, flag registers (FLAGS), and instruction decoder constitute execution unit (EU).

❖ General Purpose Registers: There are four 16-bit general purpose registers: AX (Accumulator Register), BX (Base Register), CX (Counter) and DX. Each of these 16-bit registers are further subdivided into 8-bit registers as shown below:

| 16-bit registers | 8-bit high-order registers | 8-bit low-order registers |
|---|---|---|
| AX | AH | AL |
| BX | BH | BL |
| CX | CH | CL |
| DX | DH | DL |

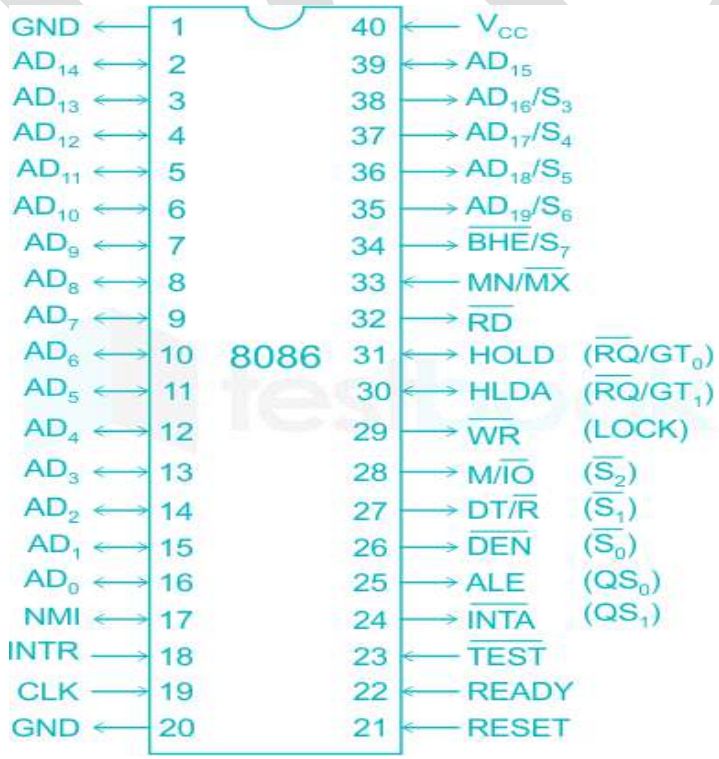
❖ Index Register: The following four registers are in the group of pointer and index registers:

- Stack Pointer (SP)
- Base Pointer (BP)
- Source Index (SI)
- Destination Index (DI)

❖ ALU: It handles all arithmetic and logical operations. Such as addition, subtraction, multiplication, division, AND, OR, NOT operations.

- Flag Register: It is a 16-bit register which exactly behaves like a flip-flop, means it changes states according to the result stored in the accumulator. It has 9 flags and they are divided into 2 groups i.e. conditional and control flags.
    - **Conditional Flags:** This flag represents the result of the last arithmetic or logical instruction executed. Conditional flags are:
        - Carry Flag
        - Auxiliary Flag
        - Parity Flag
        - Zero Flag
        - Sign Flag
        - Overflow Flag
    - **Control Flags:** It controls the operations of the execution unit. Control flags are:
        - Trap Flag
        - Interrupt Flag
        - Direction Flag

## 5.3 Signal Descriptionof 8086




### Power supply and frequency signals

It uses 5V DC supply at $V_{CC}$, and uses ground at $V_{SS}$ pin 1 and 20 for its operation.

## Clock signal

Clock signal provides timing to the processor for operations. Its frequency is different for different versions, i.e. 5MHz, 8MHz and 10MHz.

## Address/data bus

AD0-AD15. These are 16 address/data bus. AD0-AD7 carries low order byte data and AD8-AD15 carries higher order byte data. During the first clock cycle, it carries 16-bit address and after that it carries 16-bit data.

## Address/status bus

A16-A19/S3-S6. These are the 4 address/status buses. During the first clock cycle, it carries 4-bit address and later it carries status signals.

## S7/BHE'

BHE stands for Bus High Enable. It is used to indicate the transfer of data using data bus D8-D15. This signal is low during the first clock cycle, thereafter it is active.

## Read'

It is used to read signal for Read operation.

## Ready

It is an acknowledgement signal from I/O devices that data is transferred. It is an active high signal. When it is high, it indicates that the device is ready to transfer data. When it is low, it indicates wait state.

## RESET

It is used to restart the execution. It causes the processor to immediately terminate its present activity.

## INTR

It is an interrupt request signal, which is sampled during the last clock cycle of each instruction to determine if the processor considered this as an interrupt or not.

## NMI

It stands for non-maskable interrupt. It is an edge triggered input, which causes an interrupt request to the microprocessor.

## TEST'

This signal is like wait state. When this signal is high, then the processor has to wait for IDLE state, else the execution continues.

### MN/MX'

It stands for Minimum/Maximum. It indicates what mode the processor is to operate in; when it is high, it works in the minimum mode and when it is Low, it works in the maximum mode

### INTA'

It is an interrupt acknowledgement signal. When the microprocessor receives this signal, it acknowledges the interrupt.

### ALE

It stands for address enable latch. A positive pulse is generated each time the processor begins any operation. This signal indicates the availability of a valid address on the address/data lines.

### DEN'

It stands for Data Enable. It is used to enable Transreceiver 8286. The transreceiver is a device used to separate data from the address/data bus.

### DT/R'

It stands for Data Transmit/Receive signal. It decides the direction of data flow through the transreceiver. When it is high, data is transmitted out and vice-a-versa.

### M/IO'

This signal is used to distinguish between memory and I/O operations. When it is high, it indicates I/O operation and when it is low indicates the memory operation.

### WR'

It stands for write signal. It is used to write the data into the memory or the output device depending on the status of M/IO' signal.

### HLDA

It stands for Hold Acknowledgement signal. This signal acknowledges the HOLD signal.

### HOLD

This signal indicates to the processor that external devices are requesting to access the address/data buses.

### $QS_1$ and $QS_0$

These are queue status signals. These signals provide the status of instruction queue. Their conditions are shown in the following table −

| $QS_0$ | $QS_1$ | Status |
|---|---|---|
| 0 | 0 | No operation |
| 0 | 1 | First byte of opcode from the queue |
| 1 | 0 | Empty the queue |
| 1 | 1 | Subsequent byte from the queue |

### $S_0$, $S_1$, $S_2$

These are the status signals that provide the status of operation. Following is the table showing their status −

| $S_2$ | $S_1$ | $S_0$ | Status |
|---|---|---|---|
| 0 | 0 | 0 | Interrupt acknowledgement |
| 0 | 0 | 1 | I/O Read |
| 0 | 1 | 0 | I/O Write |
| 0 | 1 | 1 | Halt |
| 1 | 0 | 0 | Opcode fetch |
| 1 | 0 | 1 | Memory read |
| 1 | 1 | 0 | Memory write |
| 1 | 1 | 1 | Passive |

### LOCK

When this signal is active, it indicates to the other processors not to ask the CPU to leave the system bus. It is activated using the LOCK prefix on any instruction.

### RQ'/$GT_1$ and RQ'/$GT_0$

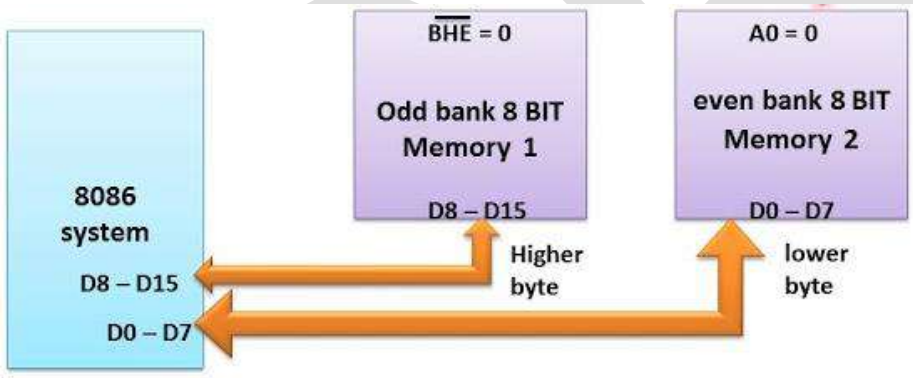
These are the Request/Grant signals used by the other processors requesting the CPU to release the system bus. When the signal is received by CPU, then it sends acknowledgment. RQ/$GT_0$ has a higher priority than RQ/$GT_1$.

## 5.4 General Bus Operation & Physical Memory Organisation

### General Bus Operation

- The 8086 has a combined address and data bus commonly referred as a time multiplexed address and data bus. The main reason behind multiplexing address and data over the same pins is the maximum utilization of processor pins and it facilitates the use of 40 pin standard DIP package.
- The bus can be demultiplexed using a few latches and transreceivers, when ever required.
- Basically, all the processor bus cycles consist of at least four clock cycles. These are referred to as T1, T2, T3, T4. The address is transmitted by the processor during T1. It is present on the bus only for one cycle.
- The negative edge of this ALE pulse is used to separate the address and the data or status information. In maximum mode, the status lines S0, S1 and S2 are used to indicate the type of operation.
- Status bits S3 to S7 are multiplexed with higher order address bits and the BHE signal. Address is valid during T1 while status bits S3 to S7 are valid during T2 through T4.

### Physical Memory Organization:




- The 8086 processor provides a 20-bit address to access any location of the 1 MB memory space. The memory is organized as a linear array of 1 million bytes, addressed as 00000(H) to FFFFF(H). The memory is logically divided into code, data, extra data, and stack segments of up to 64K bytes each.
- Physically, the memory is organized as a high bank (D15 - D8) and a low bank (D7 –D0) of 512 K 8- bit bytes addressed in parallel by the processor's address lines A19 -A1. Byte data with even addresses is transferred on the D7 – D0 bus lines while odd addressed byte data (A0 HIGH) is transferred on the D15-D8 bus lines. The processor provides two enable signals, BHE and A0, to selectively allow reading from or writing into either an odd byte location, even byte location, or both. The instruction stream is fetched from memory as words and is addressed internally by the processor to the byte level as necessary.

## 5.5 Minimum Mode &Timings,

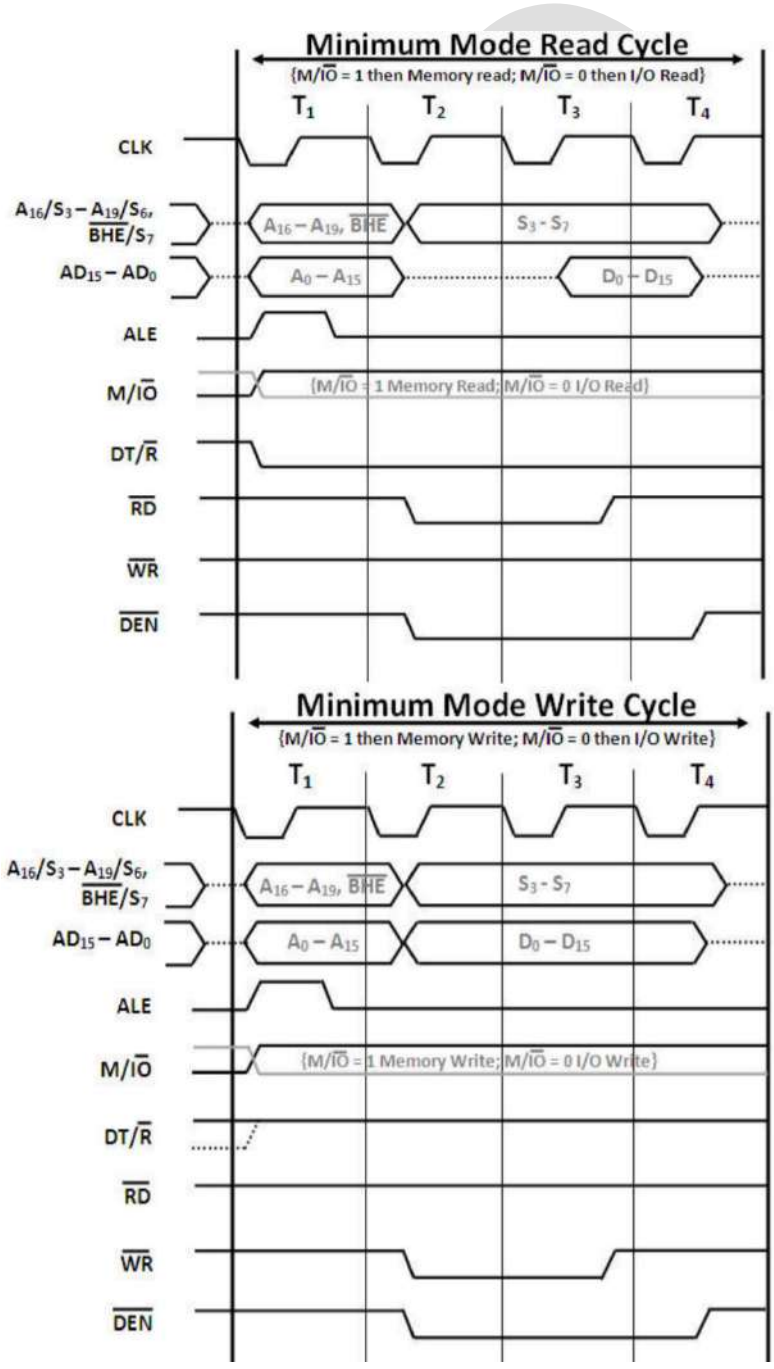
- The 8086 microprocessor operates in minimum mode when MN/MX' = 1.
- In minimum mode,8086 is the only processor in the system which provides all the control signals which are needed for memory operations and I/O interfacing.
- Here the circuit is simple but it does not support multiprocessing.

## Minimum mode of 8086 timing diagram:

Timing diagram when 8086 is on minimum mode. There are four stages in 8086 microprocessor. This timing diagram is of read and write signal.

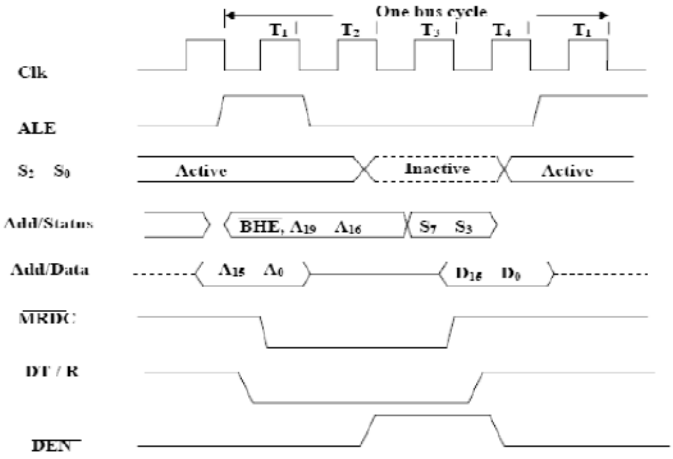
- Microprocessor gives address
- Microprocessor gives read/write data
- 8086 microprocessor waits for the data to get while reading.
- Microprocessor stores the data.

In the 3rd step the delay is known as propagation delay. This delay is only in read signal not in write signal.

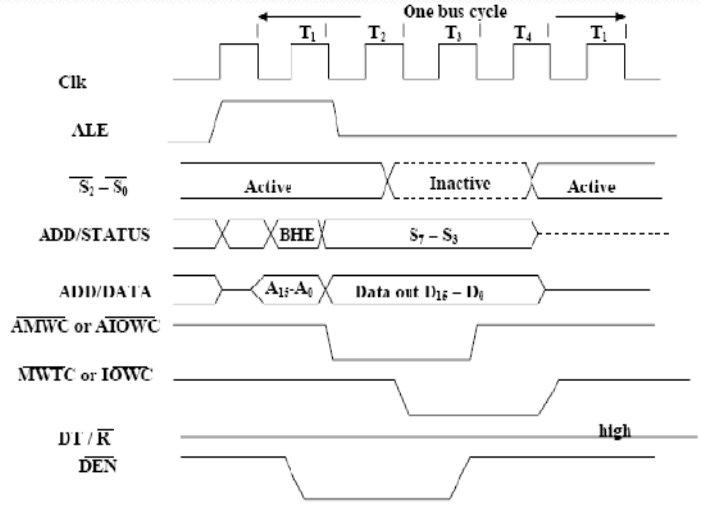


## 5.6 Maximum Mode &Timings,

- The 8086 microprocessor operates in maximum mode when MN/MX' = 0.
- In this we can connect more processors to 8086
- All processors execute their own program.




Memory Read Timing in Maximum Mode

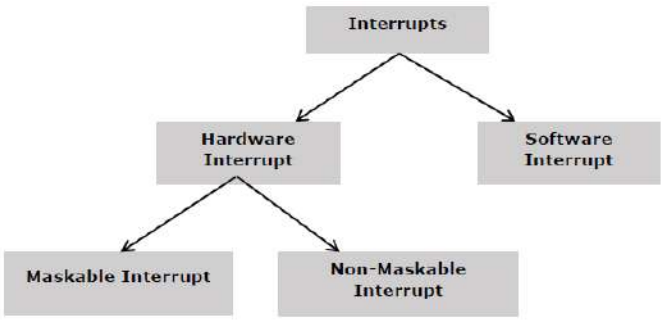



Memory Write Timing in Maximum mode.

## 5.7 Interrupts and Interrupt Service Routines, Interrupt Cycle, Non-Maskable Interrupt, Maskable

- Interrupt is the method of creating a temporary halt during program execution and allows peripheral devices to access the microprocessor.
- The microprocessor responds to that interrupt with an **ISR** (Interrupt Service Routine), which is a short program to instruct the microprocessor on how to handle the interrupt.

The following image shows the types of interrupts we have in an 8086 microprocessor –




### Hardware Interrupts

- Hardware interrupt is caused by any peripheral device by sending a signal through a specified pin to the microprocessor.
- The 8086 has two hardware interrupt pins, i.e. NMI and INTR.

NMI (Non-Maskable Interrupt): It is a single pin non-maskable hardware interrupt that cannot be disabled. It is the highest priority interrupt in the 8086 microprocessor.
INTR (Interrupt Request): It provides a single interrupt request and is activated by the I/O port. This interrupt can be masked or delayed. It is a level-triggered interrupt.

### Software Interrupts

- Some instructions are inserted at the desired position into the program to create interrupts.
- These interrupt instructions can be used to test the working of various interrupt handlers.
- There are 256 software interrupts in the 8086 microprocessor.
- The instructions are of the format INT type, where the type ranges from 00 to FF. The starting address ranges from 00000 H to 003FF H.

Some important software interrupts are:

- TYPE 0 corresponds to division by zero (0).
- TYPE 1 is used for single-step execution for debugging the program.
- TYPE 2 represents NMI and is used in power failure conditions.
- TYPE 3 represents a break-point interrupt.
- TYPE 4 is the overflow interrupt.

## 5.8 8086 Instruction Set & Programming: Addressing Modes, Instruction Set, Assembler Directives and Operators,

### Addressing Modes

The way of specifying data to be operated by an instruction is known as addressing modes. There are 8 different addressing modes in 8086 programming

#### 1.Immediate addressing mode

The addressing mode in which the data operand is a part of the instruction itself is known as immediate addressing mode.
Example:

MOV CX, 4929 H,

ADD AX, 2387 H,

MOV AL, FFH

2.Register addressing mode

In this type of addressing mode both the operands are registers.

Example:

MOV CX, AX

ADD BX, AX

3.Direct addressing mode

The addressing mode in which the effective address of the memory location is written directly in the instruction.

Example:

MOV AX, [1592H]

MOV AL, [0300H]

4.Register indirect addressing mode

This addressing mode allows data to be addressed at any memory location through an offset address held in any of the following registers: BP, BX, DI & SI.

**Example**

MOV AX, [BX]

ADD CX, [BX]

5.Based addressing mode

In this addressing mode, the offset address of the operand is given by the sum of contents of the BX/BP registers and 8-bit/16-bit displacement.

**Example**

MOV DX, [BX+04]

ADD CL, [BX+08]

6.Indexed addressing mode

In this addressing mode, the operands offset address is found by adding the contents of SI or DI register and 8-bit/16-bit displacements.

**Example**

MOV BX, [SI+16]

ADD AL, [DI+16]

7.Based-index addressing mode

In this addressing mode, the offset address of the operand is computed by summing the base register to the contents of an Index register.

### Example

ADD CX, [AX+SI]

MOV AX, [AX+DI]

### 8.Based indexed with displacement mode

In this addressing mode, the operands offset is computed by adding the base register contents. An Index registers contents and 8 or 16-bit displacement.

### Example

MOV AX, [BX+DI+08]

ADD CX, [BX+SI+16]

## Instruction Set of 8086

Instructions are classified on the basis of functions they perform. The 8086 microprocessor supports 8 types of instructions

### Data Transfer instruction

All the instructions which perform data movement come under this category. The source data may be a register, memory location, port etc. the destination may be a register, memory location or port. The following instructions come under this category:

| Instruction | Description |
|---|---|
| MOV | Moves data from register to register, register to memory, memory to register, memory to accumulator, accumulator to memory, etc. |
| LDS | Loads a word from the specified memory locations into specified register. It also loads a word from the next two memory locations into DS register. |
| LES | Loads a word from the specified memory locations into the specified register. It also loads a word from next two memory locations into ES register. |
| LEA | Loads offset address into the specified register. |
| LAHF | Loads low order 8-bits of the flag register into AH register. |
| SAHF | Stores the content of AH register into low order bits of the flags register. |

| | |
|---|---|
| XLAT/XLATB | Reads a byte from the lookup table. |
| XCHG | Exchanges the contents of the 16-bit or 8-bit specified register with the contents of AX register, specified register or memory locations. |
| PUSH | Pushes (sends, writes or moves) the content of a specified register or memory location(s) onto the top of the stack. |
| POP | Pops (reads) two bytes from the top of the stack and keeps them in a specified register, or memory location(s). |
| POPF | Pops (reads) two bytes from the top of the stack and keeps them in the flag register. |
| IN | Transfers data from a port to the accumulator or AX, DX or AL register. |
| OUT | Transfers data from accumulator or AL or AX register to an I/O port identified by the second byte of the instruction. |

## Arithmetic Instructions

Instructions of this group perform addition, subtraction, multiplication, division, increment, decrement, comparison, ASCII and decimal adjustment etc.The following instructions come under this category:

| Instruction | Description |
|---|---|
| ADD | Adds data to the accumulator i.e. AL or AX register or memory locations. |
| ADC | Adds specified operands and the carry status (i.e. carry of the previous stage). |
| SUB | Subtract immediate data from accumulator, memory or register. |
| SBB | Subtract immediate data with borrow from accumulator, memory or register. |
| MUL | Unsigned 8-bit or 16-bit multiplication. |
| IMUL | Signed 8-bit or 16-bit multiplication. |

| | |
|---|---|
| DIV | Unsigned 8-bit or 16-bit division. |
| IDIV | Signed 8-bit or 16-bit division. |
| INC | Increment Register or memory by 1. |
| DEC | Decrement register or memory by 1. |
| DAA | **Decimal Adjust after BCD Addition:** When two BCD numbers are added, the DAA is used after ADD or ADC instruction to get correct answer in BCD. |
| DAS | **Decimal Adjust after BCD Subtraction:** When two BCD numbers are added, the DAS is used after SUB or SBB instruction to get correct answer in BCD. |
| AAA | **ASCII Adjust for Addition:** When ASCII codes of two decimal digits are added, the AAA is used after addition to get correct answer in unpacked BCD. |
| AAD | **Adjust AX Register for Division:** It converts two unpacked BCD digits in AX to the equivalent binary number. This adjustment is done before dividing two unpacked BCD digits in AX by an unpacked BCD byte. |
| AAM | **Adjust result of BCD Multiplication:** This instruction is used after the multiplication of two unpacked BCD. |
| AAS | **ASCII Adjust for Subtraction:** This instruction is used to get the correct result in unpacked BCD after the subtraction of the ASCII code of a number from ASCII code another number. |
| CBW | Convert signed Byte to signed Word. |
| CWD | Convert signed Word to signed Doubleword. |
| NEG | Obtains 2's complement (i.e. negative) of the content of an 8-bit or 16-bit specified register or memory location(s). |
| CMP | Compare Immediate data, register or memory with accumulator, register or memory location(s). |

## Logical Instructions

Instruction of this group perform logical AND, OR, XOR, NOT and TEST operations. The following instructions come under this category:

| Instruction | Description |
|---|---|
| AND | Performs bit by bit logical AND operation of two operands and places the result in the specified destination. |
| OR | Performs bit by bit logical OR operation of two operands and places the result in the specified destination. |
| XOR | Performs bit by bit logical XOR operation of two operands and places the result in the specified destination. |
| NOT | Takes one's complement of the content of a specified register or memory location(s). |
| TEST | Perform logical AND operation of a specified operand with another specified operand. |

## Rotate Instructions

The following instructions come under this category:

| Instruction | Description |
|---|---|
| RCL | Rotate all bits of the operand left by specified number of bits through carry flag. |
| RCR | Rotate all bits of the operand right by specified number of bits through carry flag. |
| ROL | Rotate all bits of the operand left by specified number of bits. |
| ROR | Rotate all bits of the operand right by specified number of bits. |

## Shift Instructions

The following instructions come under this category:

| Instruction | Description |
|---|---|
| SAL or SHL | Shifts each bit of operand left by specified number of bits and put zero in LSB position. |
| SAR | Shift each bit of any operand right by specified number of bits. Copy old MSB into new MSB. |
| SHR | Shift each bit of operand right by specified number of bits and put zero in MSB position. |

## Branch Instructions

It is also called program execution transfer instruction. Instructions of this group transfer program execution from the normal sequence of instructions to the specified destination or target. The following instructions come under this category:

| Instruction | Description |
|---|---|
| JA or JNBE | Jump if above, not below, or equal i.e. when CF and ZF = 0 |
| JAE/JNB/JNC | Jump if above, not below, equal or no carry i.e. when CF = 0 |
| JB/JNAE/JC | Jump if below, not above, equal or carry i.e. when CF = 0 |
| JBE/JNA | Jump if below, not above, or equal i.e. when CF and ZF = 1 |
| JCXZ | Jump if CX register = 0 |
| JE/JZ | Jump if zero or equal i.e. when ZF = 1 |
| JG/JNLE | Jump if greater, not less or equal i.e. when ZF = 0 and CF = OF |
| JGE/JNL | Jump if greater, not less or equal i.e. when SF = OF |
| JL/JNGE | Jump if less, not greater than or equal i.e. when SF ≠ OF |

| | |
|---|---|
| JLE/JNG | Jump if less, equal or not greater i.e. when ZF = 1 and SF ≠ OF |
| JMP | Causes the program execution to jump unconditionally to the memory address or label given in the instruction. |
| CALL | Calls a procedure whose address is given in the instruction and saves their return address to the stack. |
| RET | Returns program execution from a procedure (subroutine) to the next instruction or main program. |
| IRET | Returns program execution from an interrupt service procedure (subroutine) to the main program. |
| INT | Used to generate software interrupt at the desired point in a program. |
| INTO | Software interrupts to indicate overflow after arithmetic operation. |
| LOOP | Jump to defined label until CX = 0. |
| LOOPZ/LOOPE | Decrement CX register and jump if CX ≠ 0 and ZF = 1. |
| LOOPNZ/LOOPNE | Decrement CX register and jump if CX ≠ 0 and ZF = 0. |

Here, CF = Carry Flag
ZF = Zero Flag
OF = Overflow Flag
SF = Sign Flag
CX = Register

## Flag Manipulation and Processor Control Instructions

Instructions of this instruction set are related to flag manipulation and machine control. The following instructions come under this category:

| Instruction | Description |
|---|---|
| CLC | **Clear Carry Flag:** This instruction resets the carry flag CF to 0. |

| | |
|---|---|
| CLD | **Clear Direction Flag:** This instruction resets the direction flag DF to 0. |
| CLI | **Clear Interrupt Flag:** This instruction resets the interrupt flag IF to 0. |
| CMC | This instruction take complement of carry flag CF. |
| STC | Set carry flag CF to 1. |
| STD | Set direction flag to 1. |
| STI | Set interrupt flag IF to 1. |
| HLT | Halt processing. It stops program execution. |
| NOP | Performs no operation. |
| ESC | **Escape:** makes bus free for external master like a coprocessor or peripheral device. |
| WAIT | When WAIT instruction is executed, the processor enters an idle state in which the processor does no processing. |
| LOCK | It is a prefix instruction. It makes the LOCK pin low till the execution of the next instruction. |

## String Instructions

String is series of bytes or series of words stored in sequential memory locations. The 8086 provides some instructions which handle string operations such as string movement, comparison, scan, load and store. The following instructions come under this category:

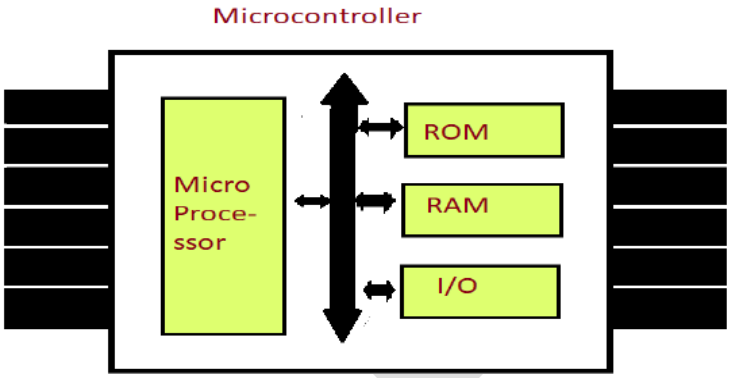
| Instruction | Description |
|---|---|
| MOVS/MOVSB/MOVSW | Moves 8-bit or 16-bit data from the memory location(s) addressed by SI register to the memory location addressed by DI register. |

| | |
|---|---|
| CMPS/CMPSB/CMPSW | Compares the content of memory location addressed by DI register with the content of memory location addressed by SI register. |
| SCAS/SCASB/SCASW | Compares the content of accumulator with the content of memory location addressed by DI register in the extra segment ES. |
| LODS/LODSB/LODSW | Loads 8-bit or 16-bit data from memory location addressed by SI register into AL or AX register. |
| STOS/STOSB/STOSW | Stores 8-bit or 16-bit data from AL or AX register in the memory location addressed by DI register. |
| REP | Repeats the given instruction until CX ≠ 0 |
| REPE/ REPZ | Repeats the given instruction till CX ≠ 0 and ZF = 1 |
| REPNE/REPNZ | Repeats the given instruction till CX ≠ 0 and ZF = 0 |

## 5.9 Simple Assembly language programming using 8086 instructions.

# UNIT-6 MICROCONTROLLER (ARCHITECTURE AND PROGRAMMING-8 BIT)

- A microcontroller is a compact integrated circuit designed to govern a specific operation in an embedded system.
- A typical microcontroller includes a processor, memory and input/output (I/O) peripherals on a single chip.
- The microcontroller used in Embedded System. for example:
  - Security Systems
  - Laser Printers
  - Automation System
  - Robotics




## 6.1 Distinguish between Microprocessor & Microcontroller

| | Microprocessor | Microcontroller |
|---|---|---|
| 1 | The microprocessor serves as the computer system's brain. | The microcontroller is the core of an embedded system. |
| 2 | It is a processor in which memory and I/O output component is connected externally. | It is a controlling device in which memory and I/O output component is present internally. |
| 3 | Since memory and I/O output is to be connected externally. Therefore the circuit is more complex. | Since on chip memory and I/O output component is available. Therefore the circuit is less complex. |

| | | |
|---|---|---|
| 4 | It cannot be used in compact system. Therefore microprocessor is inefficient. | It can be used in compact system. Therefore microcontroller is more efficient. |
| 5 | Microprocessor has less number of registers. Therefore most of the operations are memory based. | Microcontroller has more number of registers. Therefore a program is easier to write. |
| 6 | A microprocessor having a zero status flag. | A microcontroller has no zero flag. |
| 7 | It is mainly used in personal computers. | It is mainly used in washing machines, air conditioners etc. |
| 8 | It is complex and costly. | Simple and cheap. |

## 6.2 8 bit & 16 bit microcontroller

| 8-bit Microcontroller | 16-bit Microcontroller |
|---|---|
| A type of microcontroller that is capable of processing 8-bits of data at a time is called an 8-bit microcontroller. | A type of microcontroller that is capable of processing 16-bits of data at a time is called a 16-bit microcontroller. |
| 8-bit microcontroller has a data width equal to 8-bits. Hence, its CPU can process only 8-bit in parallel. | 16-bit microcontroller has a data width equal to 16-bits, which means its CPU can process 16-bits of data in parallel. |
| 8-bit microcontrollers have low processing power; hence they can handle only simple instructions. | 16-bit microcontrollers have high processing power; thus, they are capable of handling complex instructions. |
| 8-bit microcontroller has low memory capacity. | 16-bit microcontroller has high storage capacity. |
| 8-bit microcontrollers have lower clock speeds but are more reliable. | 16-bit microcontrollers offer double the clock speed but are less reliable. |
| 8-bit microcontrollers consume less power. | 16-bit microcontrollers consume more power than 8-bit microcontrollers. |

| 8-bit Microcontroller | 16-bit Microcontroller |
|---|---|
| 8-bit microcontrollers have low efficiency. | 16-bit microcontrollers are more efficient than 8-bit microcontrollers. |
| 8-bit microcontrollers are less expensive. | 16-bit microcontrollers are more expensive. |

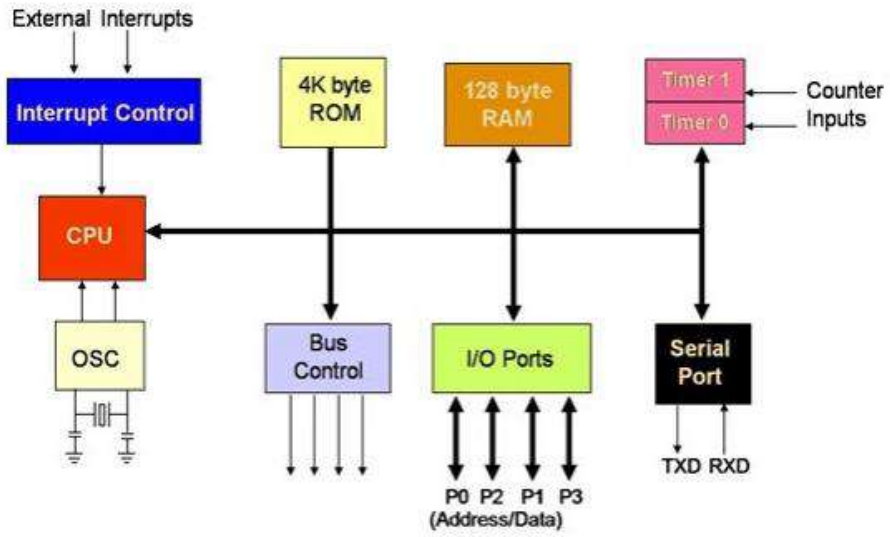
## 6.3 CISC & RISC processor

- RISC and CISC are two different types of computer architectures that are used to design the microprocessors that are found in computers.
- The fundamental difference between RISC and CISC is that RISC (Reduced Instruction Set Computer) includes simple instructions and takes one cycle, while the CISC (Complex Instruction Set Computer) includes complex instructions and takes multiple cycles.
- Difference between RISC and CISC Processor

| S.No. | RISC | CISC |
|---|---|---|
| 1. | RISC is a reduced instruction set. | CISC is a complex instruction set. |
| 2. | The number of instructions is less as compared to CISC. | The number of instructions is more as compared to RISC. |
| 3. | The addressing modes are less. | The addressing modes are more. |
| 4. | It works in a fixed instruction format. | It works in a variable instruction format. |
| 5. | The RISC consumes low power. | The CISC consumes high power. |
| 6. | The RISC processors are highly pipelined. | The CISC processors are less pipelined. |
| 7. | It optimizes the performance by focusing on software. | It optimizes the performance by focusing on hardware. |
| 8. | Requires more RAM. | Requires less RAM. |

## 6.4 Architecture of 8051Microcontroller

- 8051 Microcontroller is designed by Intel in 1981. It is an 8-bit microcontroller.
- It is built with 40 pins DIP (dual inline package).
- Its foundation is based on Harvard architecture and this microcontroller was developed principally for bringing it to be used in **Embedded Systems.**
- The internal architecture of 8051 Microcontroller represented in form of block diagram as shown below:




Basic components present internally inside 8051 Microcontroller architecture are:

### CPU (Central Processing Unit):

- CPU act as a mind of any processing machine. It synchronizes and manages all processes that are carried out in microcontroller.
- User has no power to control the functioning of CPU. It interprets the program stored in ROM and carries out from storage and then performs it projected duty.
- CPU manage the different types of registers available in 8051 microcontroller.

### Interrupts:

- Interrupts is a sub-routine call that given by the microcontroller when some other program with high priority is request for acquiring the system buses then interrupts occur in current running program.
- Interrupts provide a method to postpone or delay the current process, performs a sub-routine task and then restart the standard program again.
- The five sources of interrupts in 8051 Microcontroller:
    - Timer 0 overflow interrupt - TF0
    - Timer 1 overflow interrupt - TF1
    - External hardware interrupt - INT0
    - External hardware interrupt - INT1
    - Serial communication interrupt - RI/TI

### Memory:

- For operation Micro-controller required a program. This program guides the microcontroller to perform the specific tasks. This program installed in microcontroller required some on chip memory for the storage of the program.
- Microcontroller also required memory for storage of data and operands for the short duration. In microcontroller 8051 there is code or program memory of 4 KB that is it has 4 KB ROM and it also comprise of data memory (RAM) of 128 bytes.

### Bus:

- Bus is a group of wires which uses as a communication canal or acts as means of data transfer. The different bus configuration includes 8, 16 or more cables. Therefore, a bus can bear 8 bits, 16 bits all together.

Types of buses in 8051 Microcontroller:

- Address Bus: 8051 microcontrollers is consisting of 16 bit address bus. It is generally be used for transferring the data from Central Processing Unit to Memory.
- Data bus: 8051 microcontroller is consisting of 8 bits data bus. It is generally be used for transferring the data from one peripherals position to other peripherals.

### Oscillator:

- As the microcontroller is digital circuit therefore it needs timer for their operation. To perform timer operation inside microcontroller it required externally connected or on-chip oscillator.
- Microcontroller is used inside an embedded system for managing the function of devices. Therefore, 8051 uses the two 16 bit counters and timers. For the operation of this timers and counters the oscillator is used inside microcontroller.

### I/O Ports:

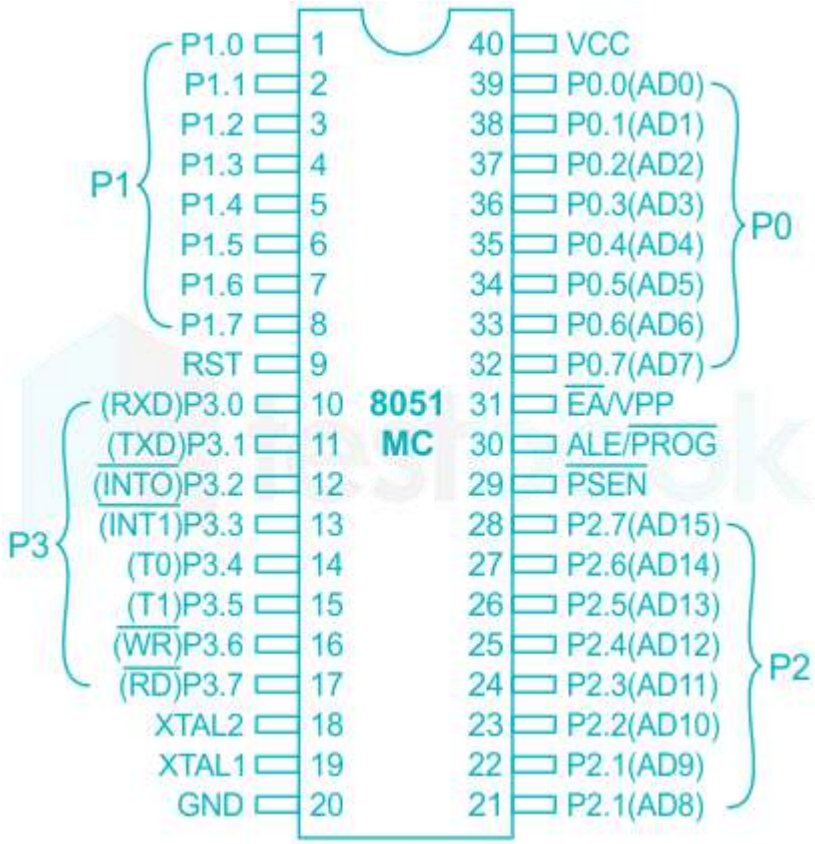
- It typically includes four 8-bit I/O ports (P0, P1, P2, and P3), totaling 32 I/O lines which can be used for input or output operations. These ports are bit-addressable, offering flexibility in interfacing with external devices.

### Timers/Counters:

- The 8051 includes two 16-bit timers/counters (Timer 0 and Timer 1) that can be used in various modes to perform tasks like time delays or counting external events.

## 6.5 Signal Descriptionof8051Microcontrollers

- 8051 is an 8-bit microcontroller that contains 40 pins of dual-in-line packaging. Out of the 40 pins, 32 pins are allotted to I/O ports. However, in 8051 there exist multiple pins which are assigned more than one signal.




**Pin 1 to Pin 8 (Port 1)**

Pin 1 to Pin 8 are assigned to Port 1 for simple I/O operations. They can be configured as input or output pins depending on the logic control.

**Pin 9**

It is a RESET pin, which is used to reset the microcontroller to its initial values.

**Pin 10 to Pin 17 (Port 3)**

Pin 10 to pin 17 are Port 3 pins which are also referred to as P3.0 to P3.7. These pins are similar to port 1 and can be used as universal input or output pins. These pins are bidirectional pins. These pins also have some additional functions which are as follows:

- P3.0 (RXD): 10th pin is RXD (serial data receive pin) which is for serial input. Through this input signal microcontroller receives data for serial communication.
- P3.1 (TXD): 11th pin is TXD (serial data transmit pin) which is serial output pin. Through this output signal microcontroller transmits data for serial communication.
- P3.2 and P3.3 (INT0', INT1'): 12th and 13th pins are for External Hardware Interrupt 0 and Interrupt 1 respectively. When this interrupt is activated(i.e. when it is low), 8051 gets interrupted in whatever it is doing and jumps to the vector value of the interrupt (0003H for INT0 and 0013H for INT1) and starts performing Interrupt Service Routine (ISR) from that vector location.
- P3.4 and P3.5 (T0 and T1): 14th and 15th pin are for Timer 0 and Timer 1 external input. They can be connected with 16 bit timer/counter.
- P3.6 (WR'): 16th pin is for external memory write i.e. writing data to the external memory.

- P3.7 (RD'): 17th pin is for external memory read i.e. reading data from external memory.

### Pin 18 and Pin 19 (XTAL2 and XTAL1)

These pins are connected to an external oscillator which is generally a quartz crystal oscillator. They are used to provide an external clock frequency of 4MHz to 30MHz.

### Pin 20

This pin provides the power supply to the circuit.

### Pins 21 to 28

These pins are known as Port 2. It serves as I/O port. Higher order address bus signals are also multiplexed using this port.

### Pin 29

This is PSEN pin which stands for Program Store Enable. It is used to read a signal from the external program memory.

### Pin 30 (ALE)

Address Latch Enable pin is an active high-output signal. If multiple memory chips are use, then this pin is used to distinguish between them. This Pin also gives program pulse input during programming of EPROM.

### Pin 31

Here EA is used for External Access. Whenever there is a need to access the program from external memory, then signal at this pin must be active low. While in case the code is not accessed by the external memory then it has an active high signal.

### Pins 32 to 39

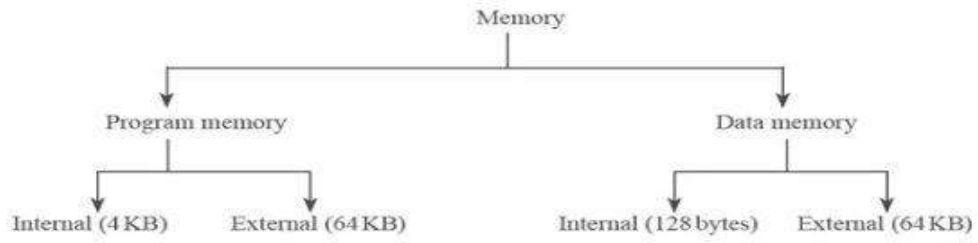
These pins are known as Port 0. It serves as I/O port. Lower order address and data bus signals are multiplexed using this port.

### Pin 40

This pin is used to provide power supply to the circuit.

## 6.6 Memory Organisation-RAM structure, SFR

- The 8051 has two types of memory and these are **Program Memory** and **Data Memory**.
- Program Memory (ROM) is used to permanently save the program being executed, while Data Memory (RAM) is used for temporarily storing data and intermediate results created and used during the operation of the microcontroller.




8051 has 128 bytes of internal RAM. It is divided into three sections.

1. Register banks
2. Bit/Byte addressable area

3. The General Purpose Memory Area

### Register banks

Register bank area has four register banks

- Bank 0
- Bank 1
- Bank 2
- Bank 3

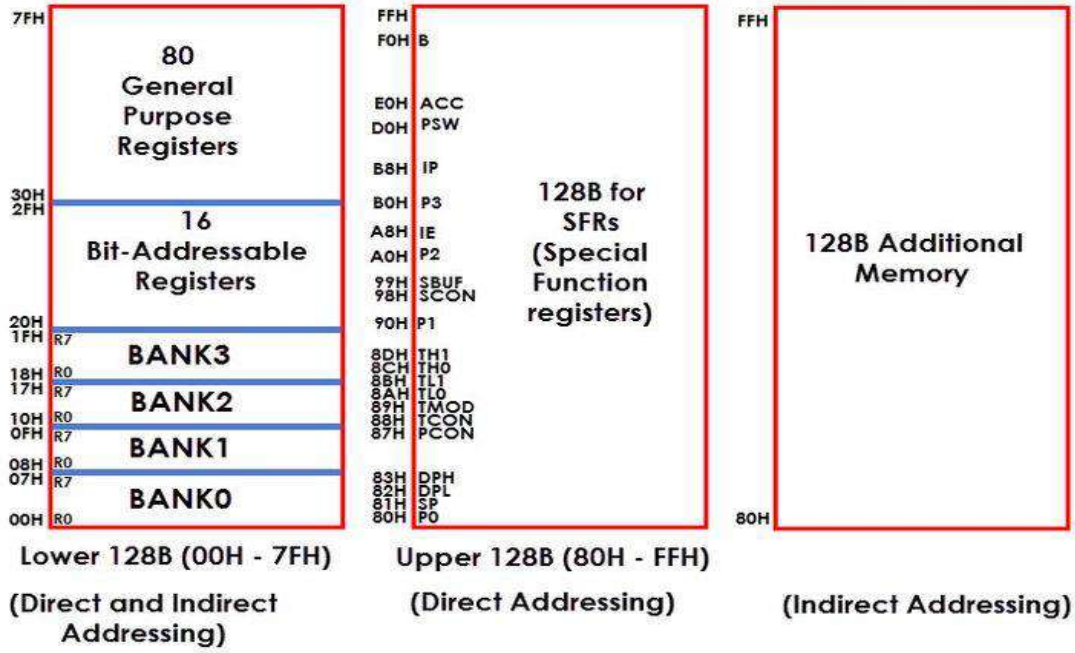
Each bank has 8 registers. Therefore there are 32 registers in the register bank area and each register is of 8 bit. Each bank has register R0 to R7. RS0 and RS1 bits of Program status word register are used to select register bank. The address of the register bank area start at 00H and ends at 1FH.

### Bit/Byte addressable area

Bit addressable area can store or remove 1-bit of data as well as 1-byte data. This area has total of 128 bit addresses starting from 00H to 07FH. They have byte address from 20H to 2FH. Separate instructions are used to read and write a single bit.

### General Purpose Memory Area

It is used to store any general purpose data. The addresses of the general purpose memory area is from 30H to 7FH.




## 6.7 Registers, timers, interrupts of 8051 Microcontrollers

A register is a small place in a CPU that can store small amounts of the data used for performing various operations such as addition and multiplication and loads the resulting data on the main memory. Registers contain the address of the memory location where the data is to be stored.

The 8051 microcontroller contains mainly two types of registers:

- General-purpose registers (Byte addressable registers)
- Special function registers (Bit addressable registers)

## General Purpose Registers in 8051

- Totally 34 general purpose registers or working registers. They are used to store data.
- Two of these A and B hold results of many instructions, particularly arithmetic and logical operations of 8051 CPU.
- The other 32 are in four register banks, RB0 –RB3 of eight registers each.

## Accumulator

- The Accumulator (A register) is used for many operations like Addition, Subtraction, Multiplication, Division and Boolean bit logical operations.
- It is also used for data transfer between 8051 and external memories, I/O devices.

## B Register

- The B register is used to do arithmetic operations such as multiplication and division with Accumulator.
- B register may be used to store the data and it has no other functions.

## R Registers (R0-R7)

- These registers are from the data memory location starting from 00h. There are 4 register banks each consisting of eight 8-bit registers.
- R registers is a common name for 8 general-purpose registers (R0, R1, R2 …R7). Similar to the accumulator, they are used for temporary storing variables and intermediate results during operation.

## Special Function Registers (SFR)

- The 8051 Microcontroller Special Function Registers are used to program and control different hardware peripherals like Timers, Serial Port, I/O Ports etc.
- In fact, by manipulating the 8051 Microcontroller Special Function Registers (SFRs), the operating mode of the 8051 Microcontroller can assess or change.
- The 8051 Microcontroller Special Function Registers act as a control table that monitor and control the operation of the 8051 Microcontroller. The Address Space for Internal RAM Structure from 80H to FFH is allocated to SFRs.
- Out of these 128 Memory Locations (80H to FFH), there are only 21 locations that are actually assigned to SFRs. Each SFR has one Byte Address and also a unique name which specifies its purpose.
- Since the SFRs are a part of the Internal RAM Structure, you can access SFRs as if you access the Internal RAM. The main difference is the address space: first 128 Bytes (00H to 7FH) is for regular Internal RAM and next 128 Bytes (80H to FFH) is for SFRs.

The 21 Special Function Registers of 8051 Microcontroller are categorized in to seven groups. They are:

- Math or CPU Registers: A and B
- Status Register: PSW (Program Status Word)
- Pointer Registers: DPTR (Data Pointer - DPL, DPH) and SP (Stack Pointer)
- I/O Port Latches: P0 (Port 0), P1 (Port 1), P2 (Port 2) and P3 (Port3)

- Peripheral Control Registers: PCON, SCON, TCON, TMOD, IE and IP
- Peripheral Data Registers: TL0, TH0, TL1, TH1 and SBUF

### Math or CPU Registers

A OR ACCUMULATOR (ACC): The Accumulator is used to hold the data for almost all the ALU Operations.

B (REGISTER B): The B Register is used along with the ACC in Multiplication and Division operations.

### Status Register

PROGRAM STATUS WORD (PSW): The PSW Register consists of Flag Bits, which help the programmer in checking the condition of the result and also make decisions

### Pointer Registers

DPTR: The Data Pointer is a 16-bit Register and is physically the combination of DPL (Data Pointer Low) and DPH (Data Pointer High) SFRs. The Data Pointer can be used as a single 16-bit register (as DPTR) or two 8-bit registers (as DPL and DPH).

SP or Stack Pointer: It points out to the top of the Stack and it indicates the next data to be accessed. Stack Pointer can be accessed using PUSH, POP, and CALL and RET Instructions.

### I/O Port Registers (P0, P1, P2 and P3)

The 8051 Microcontroller four Ports which can be used as Input and/or Output. These four ports are P0, P1, P2 and P3. Each Port has a corresponding register with same names (the Port Registers are also P0, P1, P2 and P3

### Peripheral Control Registers

PCON (Power Control): The PCON or Power Control register, as the name suggests is used to control the 8051 Microcontroller's Power Modes. Using two bits in the PCON Register, the microcontroller can be set to Idle Mode or Power down Mode.

SCON (Serial Control): The Serial Control or SCON SFR is used to control the 8051 Microcontroller's Serial Port.

TCON (Timer Control): Timer Control or TCON Register is used to start or stop the Timers of 8051 Microcontroller. It also contains bits to indicate if the Timers has overflowed. The TCON SFR also consists of Interrupt related bits.

TMOD (Timer Mode): The TMOD or Timer Mode register or SFR is used to set the Operating Modes of the Timers T0 and T1. The lower four bits are used to configure Timer0 and the higher four bits are used to configure Timer1.

IE (Interrupt Enable): The IE or Interrupt Enable Register is used to enable or disable individual interrupts. If a bit is SET, the corresponding interrupt is enabled and if the bit is cleared, the interrupt is disabled. The Bit7 of the IE register i.e. EA bit is used to enable or disable all the interrupts

IP (Interrupt Priority): The IP or Interrupt Priority Register is used to set the priority of the interrupt as High or Low. If a bit is CLEARED, the corresponding interrupt is assigned low priority and if the bit is SET, the interrupt is assigned high priority.

### Peripheral Data Registers

TL0/TH0 (Timer 0 Low/High): The Timer 0 consists of two SFRs: TL0 and TH0. The TL0 is the lower byte and the TH0 is the higher byte and together they form a 16-bit Timer0 Register.
TL1/TH1 (Timer 1 Low/High): The TL1 and TH1 are the lower and higher bytes of the Timer 1.
SBUF (Serial Data Buffer): The Serial Buffer or SBUF register is used to hold the serial data while transmission or reception.

## 6.8 Addressing Modes of 8051

- The way in which the data operands are accessed by different instructions is known as the addressing modes.
- There are various methods of denoting the data operands in the instruction.
- The 8051 microcontroller supports mainly 5 addressing modes.

- Immediate addressing mode
- Direct Addressing mode
- Register addressing mode
- Register Indirect addressing mode
- Indexed addressing mode

### Immediate addressing mode

- In this type of addressing mode the data is immediately or directly transfer to the destination register.
- Normally the data must be preceded by a # sign. If "#" is not present then it consider as an address.

**For Example:**

MOV A, #25H; load 25H into A
MOV R3, # 67H; load 67H into R4 register.
MOV P1, #21H: Move the data 21 to PORT1

### Direct Addressing mode

- In the Direct Addressing Mode, the source or destination address is specified by using 8-bit data in the instruction. Only the internal data memory can be used in this mode.

**For Example:**

MOV R1, 42H: Move the contents of RAM location 42 into R1 register
MOV 49H, A: Move the contents of the accumulator into the RAM location 49.
ADD A, 56H: Add the contents of the RAM location 56 to the accumulator

### Register addressing mode

- In this type of addressing mode both source and destination is register.

For Example:

MOV A, R0: Move the contents of the register R0 to the accumulator
ADD A, R6: Add the contents of R6 register to the accumulator
MOV P1, R2: Move the contents of the R2 register into port 1

### Register Indirect addressing mode

- The addressing mode in which a register is used as a pointer to the data memory block is known as Register indirect addressing mode.
- When R0 and R1 are used as pointers, they must be preceded by @ sign

For Example:

MOV A, @ R0: Move the contents of RAM location whose address is in R0 into A (accumulator)
MOV @ R1, B: Move the contents of B into RAM location whose address is held by R1

### Indexed addressing mode

- In the indexed addressing mode, the source memory can only be accessed from program memory only. The destination operand is always the register A.

For Example:

**MOV C A, @A+PC: The register holds the value 07H and the PC has the value 4100H. Then the data in memory location 4107H (4100+07) is moved to A register.**

MOVC A, @A+DPTR

## 6.9 Simple 8051 Assembly Language Programming Arithmetic & Logic Instructions, JUMP, LOOP, CALL Instructions, I/O Port Programming

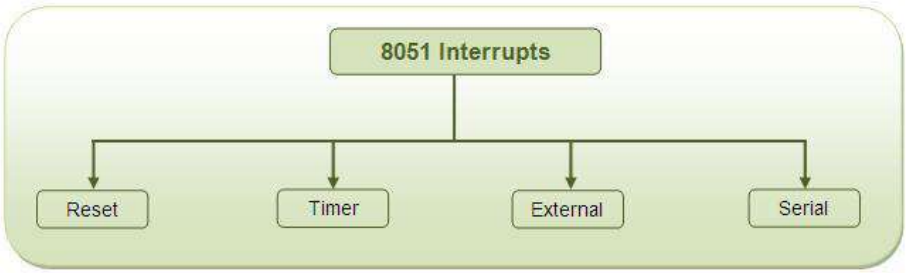
## 6.10 Interrupts, Timer & Counters

- Interrupts are the events that temporarily suspend the main program, pass the control to the external sources and execute their task. It then passes the control to the main program where it had left off.
- The interrupts can be either **hardware interrupts** or **software interrupts**.

### Hardware and Software interrupt

- The interrupts in a controller can be either hardware or software. If the interrupts are generated by the controller's inbuilt devices, like timer interrupts; or by the interfaced devices, they are called the hardware interrupts. If the interrupts are generated by a piece of code, they are termed as software interrupts.

The 8051 controller has following hardware interrupts. These are as follows:




**1. RESET interrupt –** This is also known as Power on Reset (POR). When the RESET interrupt is received, the controller restarts executing code from 0000H location. This is an interrupt which is not available to or, better to say, need not be available to the programmer.

**2. Timer interrupts –** Each Timer is associated with a Timer interrupt. A timer interrupt notifies the microcontroller that the corresponding Timer has finished counting.

**3. External interrupts –** There are two external interrupts EX0 and EX1 to serve external devices. Both these interrupts are active low. An external interrupt notifies the microcontroller that an external device needs its service.

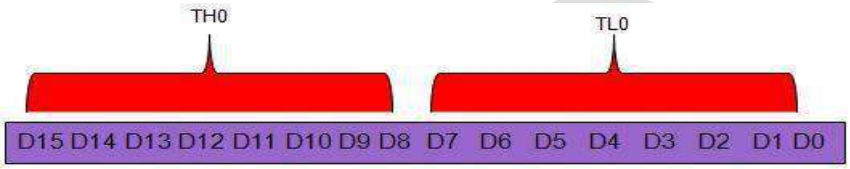
**4. Serial interrupt –** This interrupt is used for serial communication. When enabled, it notifies the controller whether a byte has been received or transmitted.

## Timer & Counters

The 8051 microcontroller has 2 timers/counters called T0 and T1. As their names suggest, their main purpose is to measure time and count external events. Besides, they can be used for generating clock pulses to be used in serial communication.

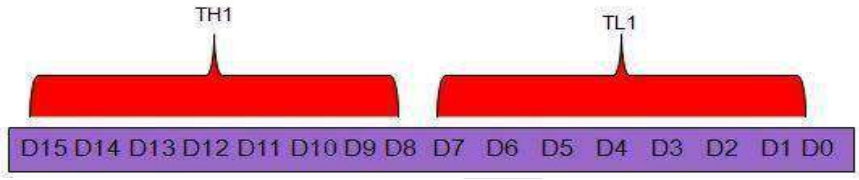
### Timer 0 Register

The 16-bit register of Timer 0 is accessed as low- and high-byte. The low-byte register is called TL0 (Timer 0 low byte) and the high-byte register is called TH0 (Timer 0 high byte). These registers can be accessed like any other register.




### Timer 1 Register

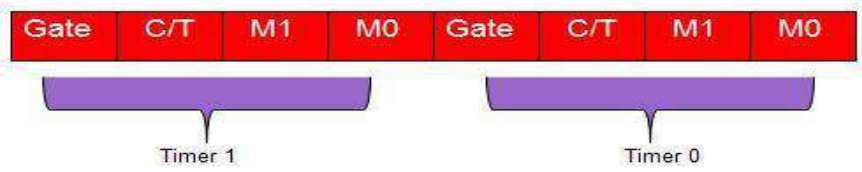
The 16-bit register of Timer 1 is accessed as low- and high-byte. The low-byte register is called TL1 (Timer 1 low byte) and the high-byte register is called TH1 (Timer 1 high byte). These registers can be accessed like any other register.




Two registers TMOD and TCON are closely connected to this timer and control its operation.

### TMOD (timer mode) Register:

This is an 8-bit register which is used by both timers 0 and 1 to set the various timer modes. In this TMOD register, lower 4 bits are set aside for timer0 and the upper 4 bits are set aside for timer1. In each case, the lower 2 bits are used to set the timer mode and upper 2 bits to specify the operation.




**Gate** – When set, the timer only runs while INT (0, 1) is high.

**C/T** – Counter/Timer select bit.

**M1** – Mode bit 1.

**M0** – Mode bit 0.

Bits of this register have the following function:

GATE1: Enables and disables Timer 1 by means of a signal brought to the INT1 pin (P3.3):

**1** - Timer 1 operates only if the INT1 bit is set.

**0** - Timer 1 operates regardless of the logic state of the INT1 bit.

C/T1: selects pulses to be counted up by the timer/counter 1

**1** - Timer counts pulses brought to the T1 pin (P3.5).

**0** - Timer counts pulses from internal oscillator.

T1M1, T1M0: These two bits select the operational mode of the Timer 1.

| T1M1 | T1M0 | MODE | DESCRIPTION |
|---|---|---|---|
| 0 | 0 | 0 | 13-bit timer |
| 0 | 1 | 1 | 16-bit timer |
| 1 | 0 | 2 | 8-bit auto-reload |
| 1 | 1 | 3 | Split mode |

GATE0: enables and disables Timer 1 using a signal brought to the INT0 pin (P3.2):

**1** - Timer 0 operates only if the INT0 bit is set.

**0** - Timer 0 operates regardless of the logic state of the INT0 bit.

C/T0: selects pulses to be counted up by the timer/counter 0:

**1** - Timer counts pulses brought to the T0 pin (P3.4).

**0** - Timer counts pulses from internal oscillator.

T0M1, T0M0: These two bits select the oprtaional mode of the Timer 0.

| T0M1 | T0M0 | MODE | DESCRIPTION |
|---|---|---|---|
| 0 | 0 | 0 | 13-bit timer |
| 0 | 1 | 1 | 16-bit timer |
| 1 | 0 | 2 | 8-bit auto-reload |
| 1 | 1 | 3 | Split mode |

### Timer Control (TCON) Register

TCON register is also one of the registers whose bits are directly in control of timer operation. Only 4 bits of this register are used for this purpose, while rest of them is used for interrupt

control to be discussed later.

| TCON | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Value after Reset |
|---|---|---|---|---|---|---|---|---|---|
| | TF1 | TR1 | TF0 | TR0 | IE1 | IT1 | IE0 | IT0 | Bit name |
| | bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0 | |

**TF1** bit is automatically set on the Timer 1 overflow.

**TR1** bit enables the Timer 1.

**1** - Timer 1 is enabled.

**0** - Timer 1 is disabled.

**TF0** bit is automatically set on the Timer 0 overflow.

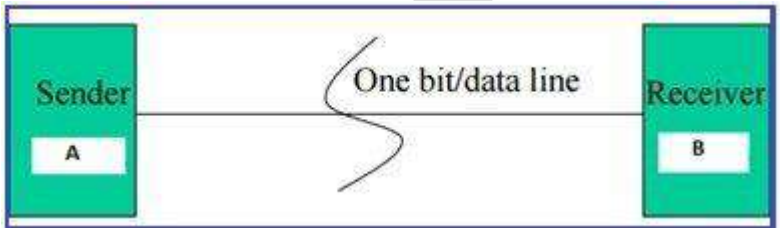
**TR0** bit enables the timer 0.

**1** - Timer 0 is enabled.

**0** - Timer 0 is disabled.

## 6.11 Serial Communication

- When digital data transfer in serial mode, this mode transmits the data bit by bit.
- Serial communication may be synchronous or asynchronous. In synchronous communication, the transmitter also transmits a clock along with data. In an asynchronous transfer of data, there is no clock.
- Serial communication may be simplex, half-duplex, or full-duplex.
- Simplex communication means that data transmission will be in one direction only, while half-duplex means data transmission will be in both directions, but at one time only one device can transmit, whereas full-duplex means data transmission may be in both directions at one time, and while one device is transmitting, it can also receive data transmitted from other devices at the same time.
- The transmitter and receiver need configuration to communicate at some data transfer rate before communication starts. This data transfer rate, or the transmission per second of the number of bits, is the baud rate for handling serial communication.




- The 8051 microcontroller is consisting of **Universal Asynchronous Receiver Transmitter (UART)** used for serial communication. The signals are transmitted and received by the Rx and Tx pins of microcontroller.
- The UART take individual bytes of data and sends the individual bits in a sequential manner. The registers are used for collecting and storing the data inside a memory. UART is based on half-duplex protocol.