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

1.1 INTRODUCTION TO MICROPROCESSOR AND MICROCOMPUTER-

MICROPROCESSOR:

- A Microprocessor is a multipurpose, Programmable clock driven, register based electronic device,
- That read binary instruction from a storage device called memory, accepts binary data as input and processes data according to those instructions and provides results as outputs.
- Microprocessor is clock driven semiconductor device which for is manufactured by using LSI and VLSI technique.

MICROCOMPUTER:

- A **microcomputer** is a small, relatively inexpensive computer with a microprocessor as its central processing unit (CPU). It includes a microprocessor, memory, and input/output (I/O) facilities.

- Microcomputers became popular in the 1970s and 80s with the advent of increasingly powerful microprocessors.
- Examples of Microcomputers are Intel 8051 controller-a single board computer,

- IBM PC and Apple Macintosh computer.

1.2 DIFFERENCE BETWEEN MICROCOMPUTER AND MICROPROCESSOR-

General Architecture of Microcomputer System:

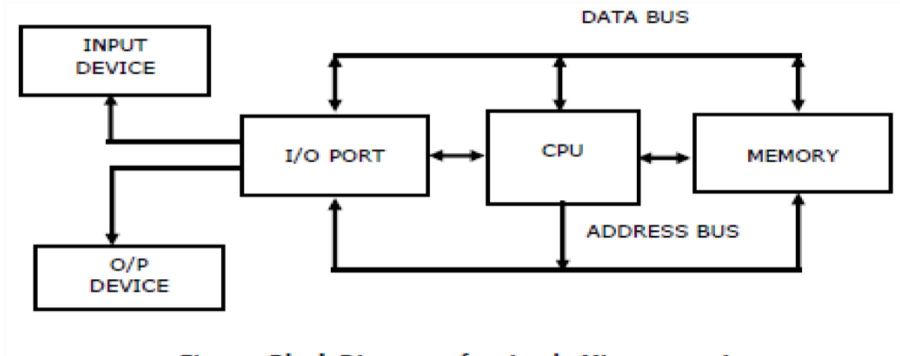


Figure: Block Diagram of a simple Microcomputer

The major parts are CPU, Memory and I/O

There are three buses, address bus, data bus and control bus;

MEMORY:

- Memory consist of RAM and ROM, the purpose of memory is to store binary codes for the sequences of instructions you want the computer to carry out.
- The second purpose of the memory is to store the binary-coded data with which the computer is going to be working.

INPUT / OUTPUT:

- The input/output or I/O Section allows the computer to take in data from the outside world or send data to the outside world.
- Peripherals such as keyboards, video display terminals, printers are connected to I/O Port.

CPU (CENTRAL PROCESSING UNIT):

- In a microcomputer CPU is a microprocessor.
- The fetches binary coded instructions from memory, decodes the instructions into a series of simple actions and carries out these actions in a sequence of steps.
- The CPU also contains an address counter or instruction pointer register, which holds the address of the next instruction or data item to be fetched from memory.

Architecture of microprocessor-

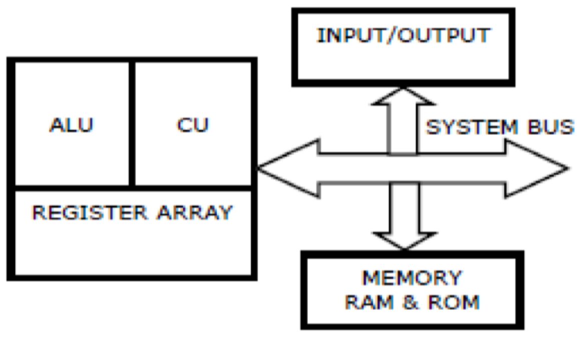


Figure: Microprocessor Based System with Bus Architecture.

Microprocessor is divided into three segments-

1. ALU
2. Register
3. Control Unit

Arithmetic Logic Unit:

- This is the area of Microprocessor where various computing functions are performed on data.

- The ALU performs operations such as addition, subtraction and logic operations such as AND, OR and exclusive OR.

Control Unit:

- The Control Unit Provides the necessary timing and control signals to all the operations in the Microcomputer
- It controls the flow of data between the Microprocessor and Memory and Peripherals.
- The Control unit performs 2 basic tasks

  →Sequencing

  →Execution

Register:

- These are storage devices to store data temporarily.

- There are different types of registers depending upon the microprocessor.
- These registers are primarily used to store data temporarily during the execution of a program and are accessible to the user through the instructions.

1.3 CONCEPT OF ADDRESS, DATA & CONTROL BUS-
ADDRESS BUS:

- The address bus consists of 16, 20, 24 or 32 parallel signal lines.
- On these lines the CPU sends out the address of the memory location that is to be written to or read from. The no of memory location that the CPU can address is determined by the number of address lines.

- If the CPU has N address lines, then it can directly address 2N memory locations i.e. CPU with 16 address lines can address 216 or 65536 memory locations.

DATA BUS:

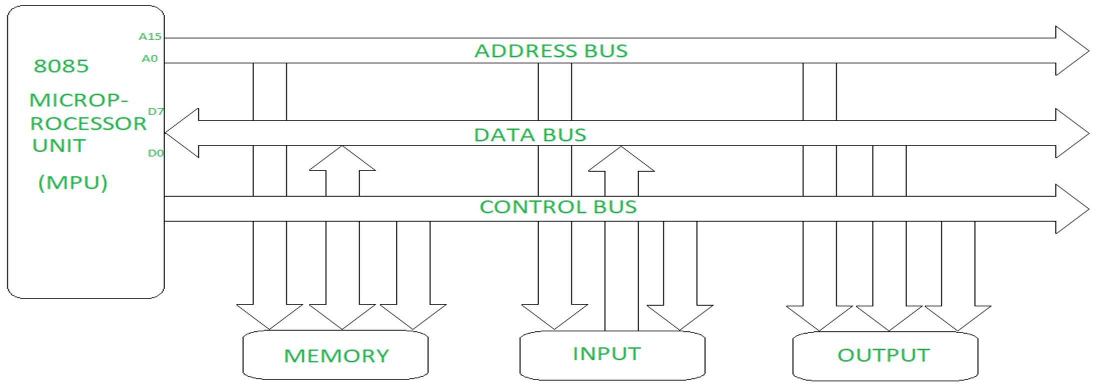
- The data bus consists of 8, 16 or 32 parallel signal lines.
- The data bus lines are bi-directional.
- This means that the CPU can read data in from memory or it can send data out to memory.

CONTROL BUS:

- The control bus consists of 4 to 10 parallel signal lines.

- The CPU sends out signals on the control bus to enable the output of addressed memory devices or port devices.
- Typical control bus signals are Memory Read, Memory Write, I/O Read and I/O Write.

1.4 GENERAL BUS STRUCTURE:



Bus organization system of 8085 Microprocessor

ADDRESS BUS:

- It is a group of conducting wires which carries address only.
- Address bus is unidirectional because data flow in one direction, from microprocessor to memory or from microprocessor to Input/output devices.
- Length of Address Bus of 8085 microprocessor is 16 Bit (i.e. Four Hexadecimal Digits), ranging from 0000 H to FFFF H, (H denotes Hexadecimal).
- The microprocessor 8085 can transfer maximum 16 bit address which means it can address 65,536 different memory location.
- The Length of the address bus determines the amount of memory a system can address.
- Such as a system with a 32-bit address bus can address 2^32 memory locations.
- If each memory location holds one byte, the addressable memory space is 4 GB. However, the actual amount of memory that can be accessed is usually much less than this theoretical limit due to chipset and motherboard limitations.

DATA BUS:

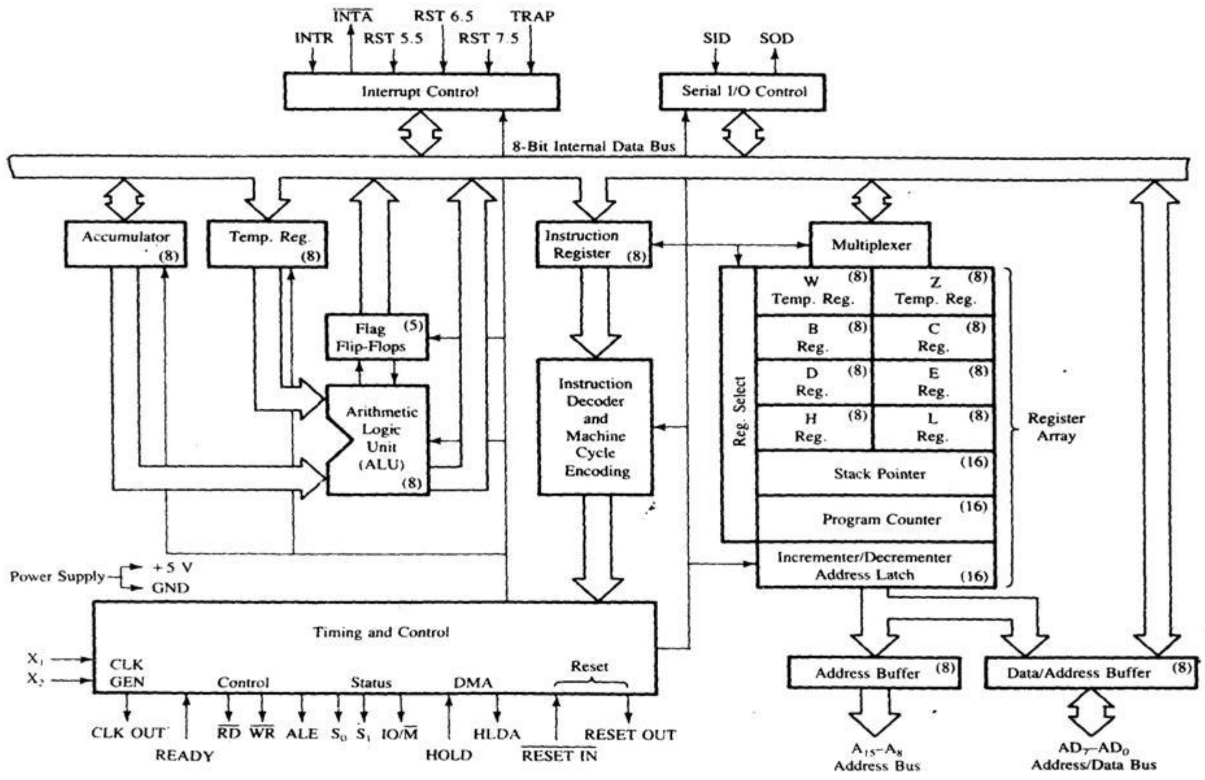
- It is a group of conducting wires which carries Data only.
- Data bus is bidirectional because data flow in both directions, from microprocessor to memory or Input/output devices and from memory or Input/output devices to microprocessor.
- Length of Data Bus of 8085 microprocessor is 8 Bit (That is, two Hexadecimal Digits), ranging from 00 H to FF H. (H denotes Hexadecimal).

- When it is write operation, the processor will put the data (to be written) on the data bus, when it is read operation, the memory controller will get the data from specific memory block and put it into the data bus.
- The width of the data bus is directly related to the largest number that the bus can carry, such as an 8 bit bus can represent 2 to the power of 8 unique values, this equates to the number 0 to 255.A 16 bit bus can carry 0 to 65535.

CONTROL BUS:

- It is a group of conducting wires, which is used to generate timing and control signals to control all the associated peripherals, microprocessor uses control bus to process data i.e. what to do with selected memory location. Some control signals are:
- Memory read
- Memory write
- I/O read
- I/O Write
- Opcode fetch

1.5 ARCHITECTURE OF 8085 MICROPROCESSOR:



Accumulator:

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

Arithmetic and logic unit:

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

General purpose register:

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

Program counter:

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

Stack pointer:

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

Temporary register:

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

Flag register:

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

These are the set of 5 flip-flops:

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

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

Instruction register and decoder:

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

Timing and control unit:

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

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

Interrupt control:

- As the name suggests it controls the interrupts during a process.
- When a microprocessor is executing a main program and whenever an interrupt occurs, the microprocessor shifts the control from the main program to process the incoming request.
- After the request is completed, the control goes back to the main program.
- There are 5 interrupt signals in 8085 microprocessor: INTR, RST 7.5, RST 6.5, RST 5.5, and TRAP.

Serial Input/output control:

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

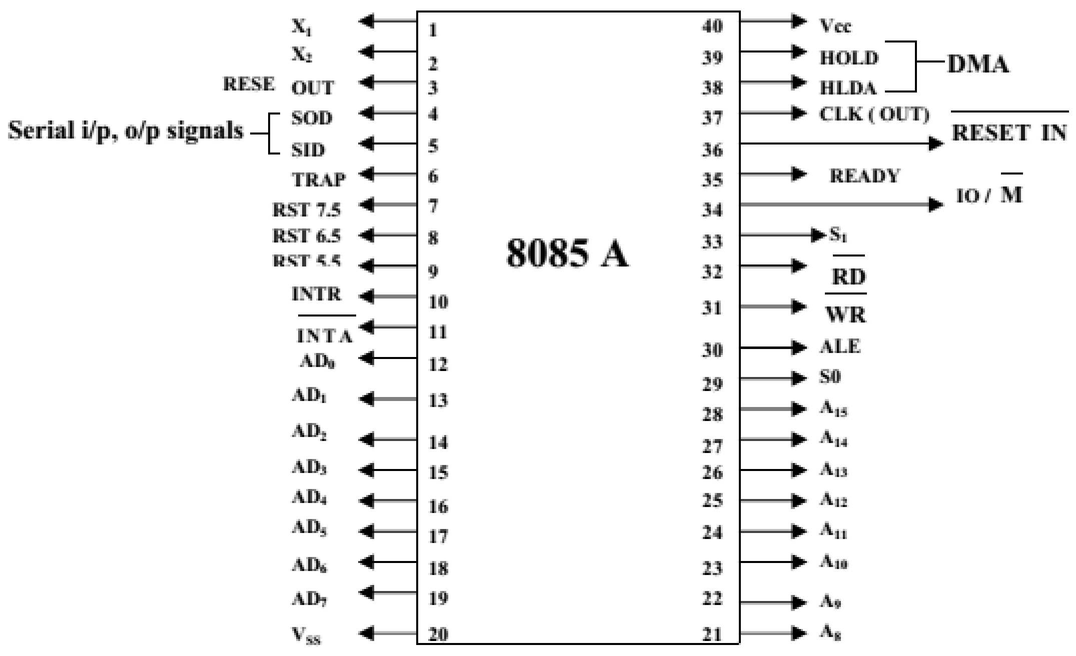
Address buffer and address-data buffer:

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

Address bus and data bus:

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

1.6 SIGNAL DESCRIPTION OF 8085:



Pin Diagram of 8085

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

Address bus:

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

Data bus:

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

Control and status signals:

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

Three control signals are RD', WR' & IO/M'.

RD':

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

WR':

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

IO/M’:

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

ALE:

It is a positive going pulse generated when a new operation is started by the microprocessor. When the pulse goes high, it indicates address. When the pulse goes down it indicates data.

S1 & S0:

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

S1	S0	Operation
0	0	Halt
0	1	Write
1	0	Read
1	1	Fetch

Power supply:

There are 2 power supply signals V_{cc} & V_{ss}. VCC indicates +5v power supply and VSS indicates ground signal.

Clock signals:

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

X1 X2:

A crystal (RC, LC N/W) is connected at these two pins and is used to set frequency of the internal clock generator. This frequency is internally divided by 2.

CLK OUT:

This signal is used as the system clock for devices connected with the microprocessor.

Interrupts & externally initiated signals:

- Interrupts are the signals generated by external devices to request the microprocessor to perform a task.
- There are 5 interrupt signals, i.e. TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR. We will discuss interrupts in detail in interrupts section.

TRAP:

- It is a non-maskable interrupt, having the highest priority among all interrupts. By default, it is enabled until it gets acknowledged. In case of failure, it executes as ISR and sends the data to backup memory. This interrupt transfers the control to the location 0024H.

RST7.5:

- It is a maskable interrupt, having the second highest priority among all interrupts. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 003CH address.

RST 6.5:

- It is a maskable interrupt, having the third highest priority among all interrupts. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 0034H address.

RST 5.5:

- It is a maskable interrupt. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 002CH address.

INTR:

It is a maskable interrupt, having the lowest priority among all interrupts. It can be disabled by resetting the microprocessor.

When **INTR signal goes high**, the following events can occur:

The microprocessor checks the status of INTR signal during the execution of each instruction.

- When the INTR signal is high, then the microprocessor completes its current instruction and sends active low interrupt acknowledge signal.
- When instructions are received, then the microprocessor saves the address of the next instruction on stack and executes the received instruction.

INTA':

It is an interrupt acknowledgment 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.

HLDA (HOLD Acknowledge):

It indicates that the CPU has received the HOLD request and it will relinquish the bus in the next clock cycle. HLDA is set to low after the HOLD signal is removed.

Serial I/O signals:

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

SOD (Serial output data line):

The output SOD is set/reset as specified by the SIM instruction.

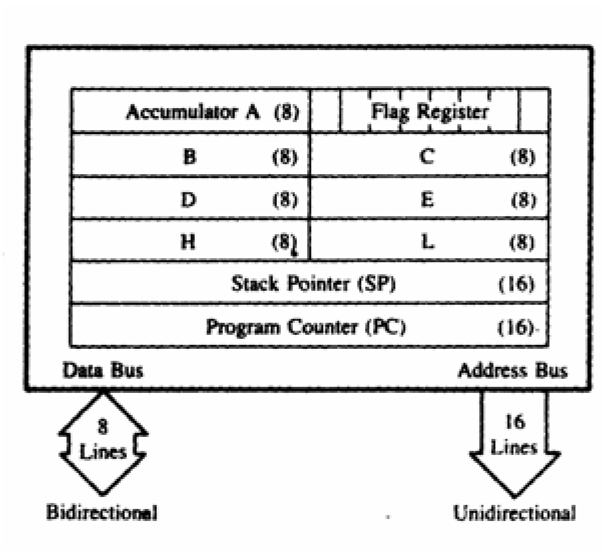
SID (Serial input data line):

The data on this line is loaded into accumulator whenever a RIM instruction is executed.

- When the INTR signal is high, then the microprocessor completes its current instruction and sends active low interrupt acknowledge signal.
- When instructions are received, then the microprocessor saves the address of the next instruction on stack and executes the received instruction.

1.7 REGISTER ORGANIZATION:

It has six addressable 8-bit registers: A, B, C, D, E, H, L and two 16-bit registers PC and SP. These registers can be classified as:



- **General Purpose Registers**
- **Temporary Registers:** Temporary data register, W and Z registers
- **Special Purpose Registers:** Accumulator, Flag registers, Instruction register
- **Sixteen-bit Registers:** Program Counter (PC), Stack Pointer (SP)

1. General Purpose Registers:

- Registers B, C, D, E, H, and L are general purpose registers in 8085 Microprocessor. All these GPRS are 8-bits wide. They are less important than the accumulator.
- They are used to store data temporarily during the execution of the program. For example, there is no instruction to add the contents of B and E registers.
- At least one of the operands has to be in A. Thus to add Band E registers, and to store the result in B register, the following have to be done.

➢ Move to A register the contents of B register.

➢ Then add A and E registers. The result will be in A.

➢ Move this result from A register to B register.

- It is possible to use these registers as pairs to store 16-bit information. Only B-C, D-E, and H-L can form register pairs.
- When they are used as register pairs in an instruction, the left register is understood to have the MSB byte and the right registers the LSB byte.

- For example, in D-E register pair, the content of the D register is treated as the MSB byte, and the content of E register is treated as the LSB byte.

2. Temporary Registers:

- **Temporary Data Register: -**
- The ALU has two inputs. One input is supplied by the accumulator and other from the temporary data register.
- The programmer cannot access this temporary data register. However, it is internally used for execution of most of the arithmetic and logical instructions.
- **W and Z register:-** Wand Z registers are temporary registers. These registers are used to hold 8-bit data during the execution of some instructions. These registers are not available for the programmer since 8085Microprocessor Architecture uses them internally.

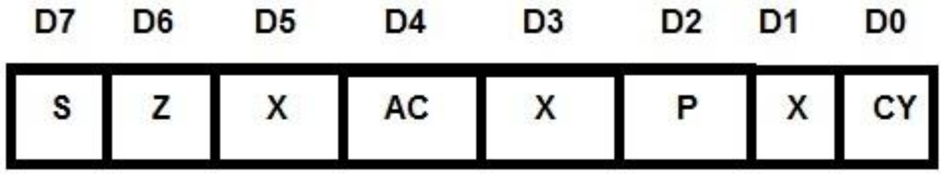
3. Special Purpose Registers:

Accumulator (A):

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

Flag Register:

- It is a 3-bit register, in which five of the bits carry significant information in the form of flags: S (Sign flag), Z (Zero flag), AC (Auxiliary carry flag), P (Parity flag), and CY (carry flag).



Flag Register of 8085

- **S-Sign flag: -** After the execution of arithmetic or logical operations, if bit D7 of the result is 1, the sign flag is set. In a given byte if D7 is1, the number will be viewed as a negative number. If D7 is U, the number will be considered as a positive number.
- **Z-Zero flag**:-The zero flag sets if the result of the operation in ALU is zero and flag resets if the result is non-zero. The zero flags are also set if a certain register content becomes zero following an increment or decrement operation of that register.
- **AC-auxiliary Carry flag: -** This flag is set if there is an overflow out of bit 3 i.e. carry from lower nibble to higher nibble (D3 bit to D4 bit). This flag is used for BCD operations and it is not available for the programmer.
- **P-Parity flag: -** Parity is defined by the number of one's present in the accumulator. After arithmetic or logical operation, if the result has an even number of ones, i.e. even parity, the flag is set. If the parity is odd, the flag is reset.
- **CY-Carry flag: -** This flag is set if there is an overflow out of bit 7. The carry flag also serves as a borrow flag for subtraction. In both the examples shown below, the carry flag is set.

Instruction Register:-

- In a typical processor operation, the processor first fetches the opcode of instruction from memory (i.e. it places an address on the address bus and memory responds by placing the data stored at the specified address on the data bus).
- The CPU stores this opcode in a register called the instruction register. This opcode is further sent to the instruction decoder to select one of the 256 alternatives.

4. Sixteen Bit Registers:

Program counter (PC):-

- Program is a sequence of instructions. Microprocessor fetches these instructions from the memory and executes them.
- The program counter is a special purpose register which, at a given time, stores the address of the next instruction to be fetched.
- Program Counter acts as a pointer to the next instruction.
- How processor increments program counter depends on the nature of the instruction; for one-byte instruction it increments program counter by one, for two-byte instruction it increments program counter by two and for three-byte instruction it increments

program counter by three such that program counter always points to the address of the next instruction.

Stack Pointer (SP):-

The stack is a reserved area of the memory in the RAM where temporary information may be stored. A 16-bit stack pointer is used to hold the address of the most recent stack entry.

1.8 DISTINGUISH BETWEEN GPR AND SPR:

GPR-

- It stands for General purpose registers.
- In these registers data can be accessed directly without requiring any intermediate.
- Examples of GPR are B, C, D, E, H, and L.
- These registers are of 8-bit.
- In order to hold 16 bit data, two 8 bit register can be combined or they can work in pairs such as B-C, D-E and H-L. These pairs are known as register pairs.
- The H-L pair works as a memory pointer.
- A memory pointer holds the address of a particular memory location.

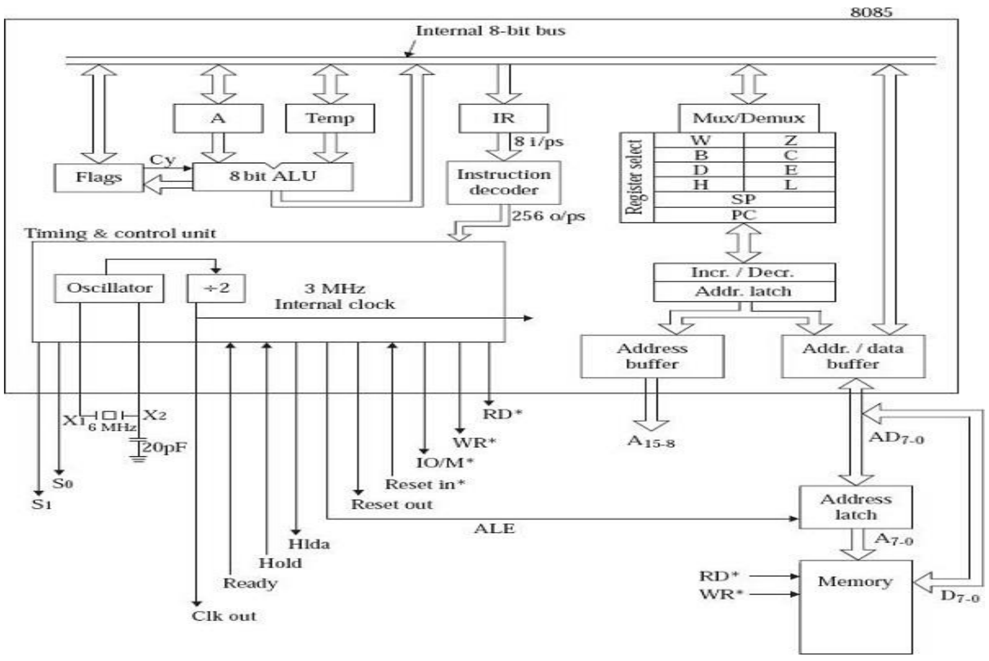
SPR-

- SPR stands for special purpose register.
- In special purpose register data cannot accessed directly and requires an intermediate.
- Examples of SPR are Accumulator, program counter, stack pointer.
- These registers are used only by microprocessor not by users.

1.9 TIMING AND CONTROL UNIT:

- We use Timing and controlling unit in 8085 for the generation of timing signals and the signals to control.
- All the operations and functions both interior and exterior of a microprocessor are controlled by this unit.
- X2 and CLK output pins: To do or rather perform the operations of timing in the microcomputer system, we have a generator called clock generator in the CU of 8085.
- Other than the quartz crystal the complete circuit of the oscillator is within the chip. The two pins namely X1 and X2 are taken out from the chip to give the connection to the crystal externally.
- We connect a capacitor of 20pF between the terminal X2 and ground just to analyze if the crystal is getting started.
- The frequency of the crystal is divided by 2 which divide the counter of the unit of control by 2.
- Internally 8085A works with a frequency of 3 MHz internally with clock frequency. Hence a crystal of frequency of 6-MHz crystal gets connected between X1 and X2.

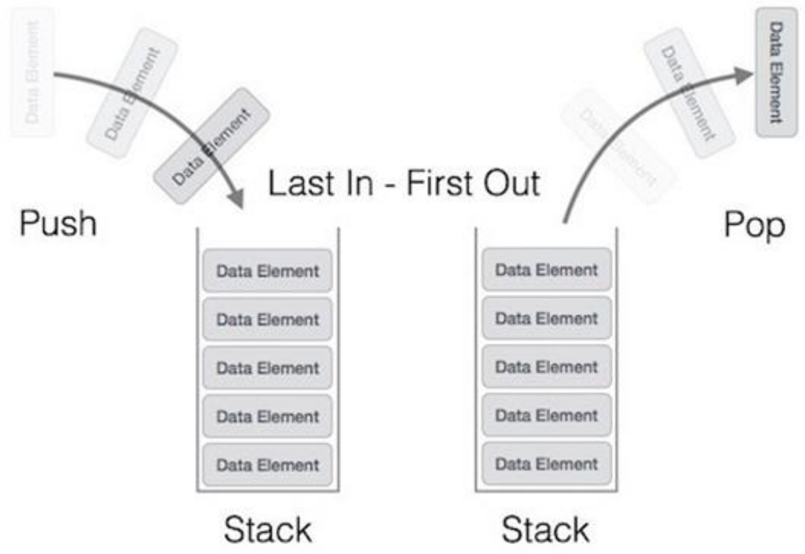
- Every operation in the entire 8085 system occurs with the given synchronization process with the clock. There are Peripheral chips like 8251 USART, which does not operate until a small clock signal is in need.



1.10 STACK, STACK POINTER AND STACK TOP:

STACK:

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



STACK POINTER:

- It is a special purpose 16-bit register that stores the address of the "**top of stack**"**.**
- "8085" provides the **"stack pointer"** which gives the address of the "top of stack". So, whenever you want to store an item it stacks, you just store it at the address provided by the stack pointer.

STACK operation in 8085 microprocessor.

The stack is a reserved area of the memory in RAM where temporary information may be stored. An 8-bit stack pointer is used to hold the address of the most recent stack entry. This location which has the most recent entry is called as the top of the stack.

When the information is written on the stack, the operation is called PUSH. When the information is read from the stack, the operation is called POP. The stack works on the principle of Last in First Out.

1.11 8085 INTERRUPTS:

- Interrupt is a process where an external device can get the attention of the microprocessor**.**
- An interrupt is considered to be an emergency signal that may be serviced.
- The Microprocessor may respond to it as soon as possible.
- The process starts from the I/O device
- The process is asynchronous

<u>Classification of Interrupts:</u>

Interrupts can be classified into two types:

- **Maskable Interrupts** (Can be delayed or Rejected)
- **Non-Maskable Interrupts** (Cannot be delayed or Rejected)

Interrupts can also be classified into:

- **Vectored** (the address of the service routine is hard-wired)
- **Non-vectored** (the address of the service routine needs to be supplied externally by the device)

What happens when MP is interrupted?

- When the Microprocessor receives an interrupt signal, it suspends the currently executing program and jumps to an Interrupt Service Routine (ISR) to respond to the incoming interrupt.
- Each interrupt will most probably have its own ISR.
- Responding to an interrupt may be immediate or delayed depending on whether the interrupt is maskable or non-maskable and whether interrupts are being masked or not.

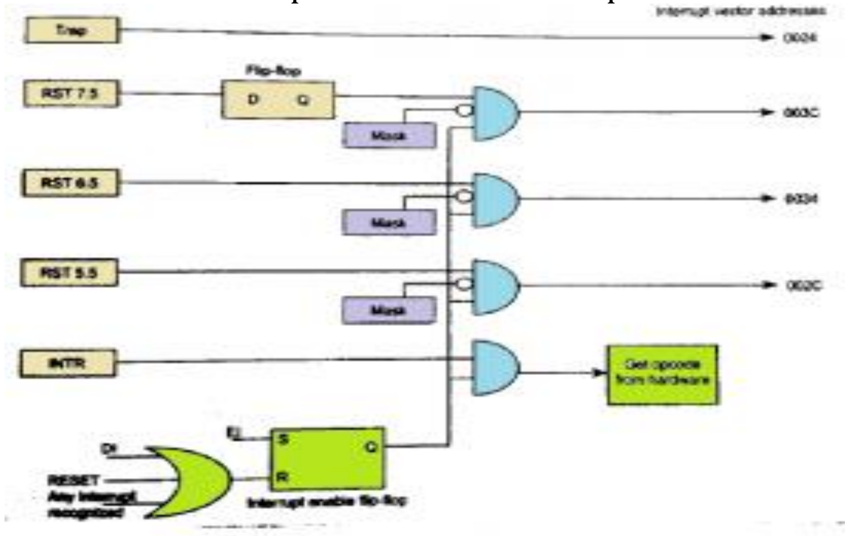
- There are two ways of redirecting the execution to the ISR depending on whether the interrupt is vectored or non-vectored.
- **Vectored:** The address of the subroutine is already known to the Microprocessor.
- **Non Vectored:** The device will have to supply the address of the subroutine to the Microprocessor.
- When a device interrupts, it actually wants the MP to give a service which is equivalent to asking the MP to call a subroutine. This subroutine is called **ISR** (Interrupt Service Routine)
- The 'EI' instruction is a one byte instruction and is used to enable the non-maskable interrupts.
- The 'DI' instruction is a one byte instruction and is used to disable the non-maskable interrupts.
- The 8085 has a single Non-Maskable interrupt. The non-maskable interrupt is not affected by the value of the Interrupt Enable flip flop.

The 8085 has 5 interrupt inputs.

- The **INTR** input is the only non-vectored interrupt. INTR is maskable using the EI/DI instruction pair.
- **RST 5.5, RST 6.5, RST 7.5** are allautomatically vectored and are mask able.
- **TRAP** is the only non-maskable interrupt in the 8085.it is also automatically vectored.

Masking of interrupt SIM, RIM:

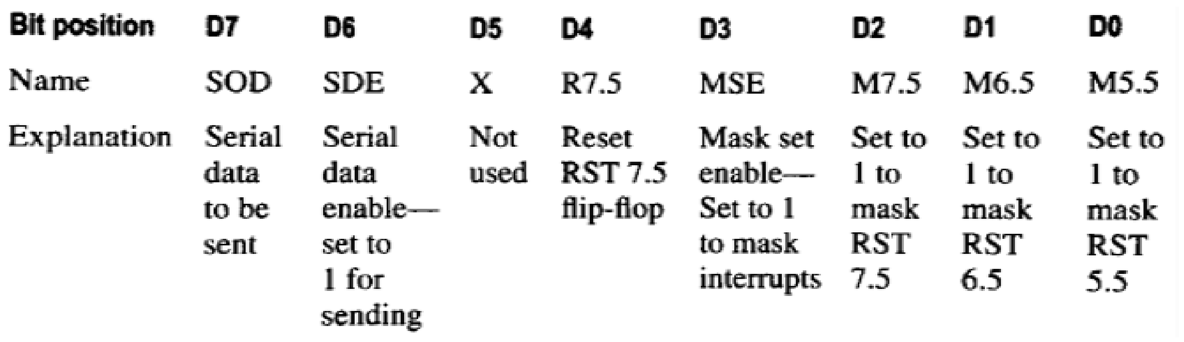
- When we study interrupts in 8085 microprocessor then we should know Masking of Interrupts in 8085 microprocessor.
- In 8085 microprocessor masking of interrupt can be done for four hardware interrupts INTR, RST 5.5, RST 6.5, and RST 7.5.
- The masking of 8085 interrupts is done at different levels. In bellow figure shows the organization of hardware interrupts in the 8085 microprocessor.



- The maskable interrupts are by default masked by the Reset signal. So no interrupt is recognized by the hardware reset.
- The interrupts can be enabled by the EI instruction.
- The three RST interrupts can be selectively masked by loading the appropriate word in the accumulator and executing SIM instruction. This is called software masking.
- All maskable interrupts are disabled whenever an interrupt is recognized.
- All maskable interrupts can be disabled by executing the DI instruction.
- If we talk about RST 7.5 interrupt. It alone has a flip-flop to recognize edge transition. The DI instruction reset interrupt enable flip-flop in the processor and the interrupts are disabled. To enable interrupts, EI instruction has to be executed.

SIM Instruction:

The SIM instruction is used to mask or unmask RST hardware interrupts. When 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 is as shown in Fig.

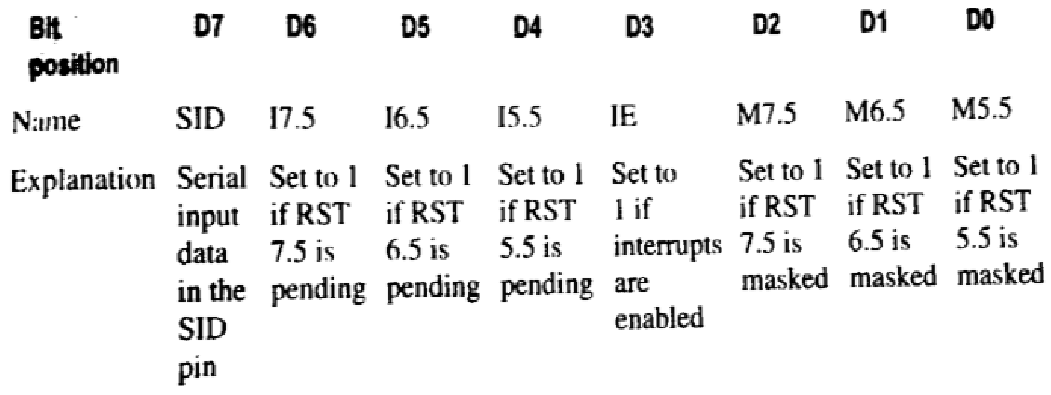


Bit position	D7	D6	D5	D4	D3	D2	D1	D0
Name	SOD	SDE	X	R7.5	MSE	M7.5	M6.5	M5.5
Explanation	Serial data to be sent	Serial data enable—set to 1 for sending	Not used	Reset RST 7.5 flip-flop	Mask set enable—Set to 1 to mask interrupts	Set to 1 to mask RST 7.5	Set to 1 to mask RST 6.5	Set to 1 to mask RST 5.5

- In addition to masking interrupts, **SIM** instruction can be used to send serial data on the **SOD** line of the processor.
- The data to be send is placed in the MSB bit of the accumulator and the serial data output is enabled by making D6 bit to 1.

RIM Instruction:

- RIM instruction is used to read the status of the interrupt mask bits.
- When **RIM** instruction is executed, the accumulator is loaded with the current status of the interrupt masks and the pending interrupts.
- The format and the meaning of the data stored in the accumulator after execution of RIM instruction is 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 also used to read the serial data on the **SID** pin of the processor.
- The data on the **SID** pin is stored in the MSB of the accumulator after the execution of the **RIM** instruction.

- E.g. write an assembly language program to enables all the interrupts in 8085 after reset**.** **EI** Enable interrupts MVI A, 08H: Unmask the interrupts **SIM:** Set the mask and unmask using SIM instruction.

UNIT-2: INSTRUCTION SET AND ASSEMBLY LANGUAGE PROGRAMMING

2.1 INSTRUCTION WORD SIZE:

- The total memory location required to feed the instruction in memory is called as **instruction word size**.
- The memory location of 8085 microprocessor can accommodate 8-bits of data.
- To store 16-bits data, they are stored in two consecutive memory locations (i.e. 2 Bytes).
- According to the instruction word size in 8085 microprocessor, there are three types of instructions:

a. 1-Byte instruction
b. 2-Byte instruction
c. 3-Byte instruction

1 – Byte Instructions:

- They include opcode and operands in the same byte.
- Operands are internal registers and coded into the instruction.
- Instructions require one memory location to store the single byte in the memory.

Note:
Instructions having the only register or register pair as the operand is 1 – Byte Instructions.
Instructions in the absence of operand are also 1 – Byte Instructions.
Examples:

MOV B, C

LDAX B

NOP

HLT

2 – Byte Instructions:

- 1st byte specifies opcode and 2nd byte specifies operand.
- Instructions require two memory locations to store in the memory.

Note:
Instructions having the 8-bit number either as an address or data as the operand is 2 – Byte Instructions.
Examples:

MVI B, 26 H

IN 56 H

3 – Byte Instructions:

- In a 3-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit address.
- The 2nd byte holds the low order address.
- The 3rd-byte holds the high order address.
- Instructions require three memory locations to store the single byte in the memory.

Note:
Instructions having the 16-bit number either as an address or data as the operand is 3 – Byte Instructions.
Examples:

LDA 2050 H

JMP 2085 H

2.2 ADDRESSING MODES:

- The various ways of specifying data (or operands) for instructions are called as **addressing modes.**
- The 8085 addressing modes are classified into following types:

1. Immediate addressing mode
2. Direct addressing mode
3. Register addressing mode
4. Register indirect addressing mode
5. Implicit addressing mode

1. Direct Addressing mode:

- In this addressing mode the address of the operand is specified in the instruction itself.

or

- The mode of addressing in which the 16-bit address of the operand is directly available in the instruction itself is called Direct Addressing mode. i.e., the address of the operand is available in the instruction itself. This is a 3-byte instruction.

Example:
LDA 9525H→ Load the contents of memory location into Accumulator.

STA 8000H→Store the contents of the Accumulator in the location 8000H

IN 01H→ Read the data from port whose address is 01H

2. Register addressing modes:

- In this addressing mode the address of the operand is one of the general purpose register.

or

- In this mode the operands are microprocessor registers only i.e. the operation is performed within various registers of the microprocessor.

Example:

- MOV A, B→Move the contents of B register to A register.
- SUB D→ Subtract the contents of D register from Accumulator.
- ADD B, C→Add the contents of C register to the contents of B register.

3. Register indirect addressing modes:

- In this addressing mode the address of the operand is specified by a register pair.

or

- The 16-bit address location of the operand stored in a register pair (H-L) is given in the instruction. The address of the operand is given in an indirect way with the help of a register pair. So it is called Register indirect addressing mode.

Example:

- LXIH 9570H→Load immediate the H-L pair with the address of the location 9570H
- MOV A, M→ Move the contents of the memory location pointed by the H-L pair to accumulator

4. Immediate Addressing mode:

- In this addressing mode the operand is specified in the instruction itself.

or

- In this mode operand is a part of the instruction itself is known as Immediate Addressing mode. If the immediate data is 8-bit, the instruction will be of two bytes. If the immediate data is 16 bit, the instruction is of 3 bytes.

Example:

ADI DATA →Add immediate the data to the contents of the accumulator.
LXIH 8500H→Load immediate the H-L pair with the operand 8500H
MVI 08H → Move the data 08 H immediately to the accumulator
SUI 05H →Subtract immediately the data 05H from the accumulator

5. Implicit Addressing mode:

- In this addressing mode the instruction don't require the address of the operand.

or

- The mode of instruction which do not specify the operand in the instruction but it is implicated, is known as implicit addressing mode. i.e., the operand is supposed to be present generally in accumulator.

Example:
CMA→complement the contents of Accumulator

CMC→ Complement carry

RLC→ Rotate Accumulator left by one bit

RRC→ Rotate Accumulator right by one bit

STC→Set carry.

2.3 INSTRUCTION SET OF 8085:

- An instruction is a binary bit pattern which performs a specific function in a system. The entire group of instructions of a system is called the instruction set.

- Instruction set determines what functions the microprocessor can perform with a single instruction.

- The instruction set in microprocessor 8085 can be classified into five functional categories:

OR

- 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.

1. Data transfer (copy) operations
2. Arithmetic operations
3. Logical operations
4. Branching operations and
5. Machine-control operations.

1. DATA TRANSFER INSTRUCTION:

- These instructions move data between registers, or between memory and registers.
- This group of instructions copies data from a location called as source to another location called as destination, without modifying the contents of the source
- These instructions are not the data transfer instructions but data copy instruction because the source is not modified.

Opcode Operand **Copy from source to destination**	**Description**
MOV Rd, Rs	This instruction copies the contents of the source
M, Rs	register into the destination register; the contents of
	The source register are not altered. If one of the operands is a memory location, its location is specified by the contents of the HL registers. Example: MOV B, C or MOV B, M

Rd, M Move immediate 8-bit

MVI Rd, data	The 8-bit data is stored in the destination register or
M, data	Memory. If the operand is a memory location, its location is specified by the contents of the HL registers. Example: MVI B, 57H or MVI M, 57H

Load accumulator

LDA 16-bit address	The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator. The contents of the source are not altered. Example: LDA 2034H

Load accumulator indirect

LDAX B/D Reg. pair	The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator. The contents of either the register pair or the memory location are not altered. Example: LDAX B

Load register pair immediate

LXI Reg. pair, 16-bit data	The instruction loads 16-bit data in the register pair designated in the operand. Example: LXI H, 2034H

Load H and L registers direct

LHLD 16-bit address	The instruction copies the contents of the memory location pointed out by the 16-bit address into register L and copies the contents of the next memory location into register H. The contents of source memory locations are not altered. Example: LHLD 2040H

Store accumulator direct

STA 16-bit address	The contents of the accumulator are copied into the memory location specified by the operand. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. Example: STA 4350H

Store accumulator indirect

STAX Reg. pair	The contents of the accumulator are copied into the memory location specified by the contents of the operand (register pair). The contents of the accumulator are not altered. Example: STAX B

Store H and L registers direct

SHLD 16-bit address	The contents of register L are stored into the memory location specified by the 16-bit address in the operand and the contents of H register are stored into the next memory location by incrementing the operand. The contents of registers

HL are not altered. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address.
Example: SHLD 2470H

Exchange H and L with D and E

XCHG none

The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E.
Example: XCHG

Arithmetic Operations:

They perform arithmetic operations, such as, addition, subtraction, increment, and decrement.

Addition:

- Addition of any 8-bit number, or the contents of a register or the contents of a memory location is added to the contents of the accumulator and the sum is stored in the accumulator.
- No two other 8-bit registers can be added directly.
- For example the contents of register B cannot be added directly to the contents of the register C. 8085 can also perform 16-bit. It can also perform BCD addition.

Subtraction:

- Subtraction of any 8-bit number, or the contents of a register, or the contents of a memory location can be subtracted from the contents of the accumulator and the results stored in the accumulator.
- The subtraction is performed in 2's compliment, and if the results is negative. Then they are expressed in 2's complement.
- No two other registers can be subtracted directly. 8085 do not perform 16-bit subtraction.

Increment or Decrement:

- The 8-bit contents of any register or a memory location can be incremented or decrement by 1.
- Similarly, the 16-bit contents of a register pair can be incremented or decrement by 1.
- These increment and decrement operations can be performed directly in the source itself. It means without using accumulator.

Opcode	Operand	Meaning	Explanation
ADD	R M	Add register or memory, to the accumulator	The contents of the register or memory are added to the contents of the accumulator and the result is stored in the accumulator. **Example** – ADD R,ADDM
ADC	R M	Add register to the accumulator with carry	The contents of the register or memory & M the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. **Example** – ADC R,ADDM
ADI	8-bit data	Add the immediate to the accumulator	The 8-bit data is added to the contents of the accumulator and the result is stored in the accumulator. **Example** – ADI 55
ACI	8-bit data	Add the immediate to the accumulator with carry	The 8-bit data and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. **Example** – ACI 55
LXI	Reg. pair, 16bit data	Load the register pair immediate	The instruction stores 16-bit data into the register pair designated in the operand. **Example** – LXI H, 3025H
DAD	Reg. pair	Add the register pair to H and L registers	The 16-bit data of the specified register pair are

			added to the contents of the HL register. **Example** – DAD
SUB	R M	Subtract the register or the memory from the accumulator	The contents of the register or the memory are subtracted from the contents of the accumulator, and the result is stored in the accumulator. **Example** – SUB R,SUB M
SBB	R M	Subtract the source and borrow from the accumulator	The contents of the register or the memory & M the Borrow flag are subtracted from the contents of the accumulator and the result is placed in the accumulator. **Example** – SBB R,SBBM
SUI	8-bit data	Subtract the immediate from the accumulator	The 8-bit data is subtracted from the contents of the accumulator & the result is stored in the accumulator. **Example** – SUI 55
SBI	8-bit data	Subtract the immediate from the accumulator with borrow	The 8-bit data and borrow is subtracted from the contents of the accumulator & the result is stored in the accumulator
INR	R M	Increment the register or the memory by 1	The contents of the designated register or the memory are incremented by 1 and their result is stored at the same place. **Example** – INR R, INR M

INX	R	Increment register pair by 1	The contents of the designated register pair are incremented by 1 and their result is stored at the same place. **Example** – INX R
DCR	R M	Decrement the register or the memory by 1	The contents of the designated register or memory are decremented by 1 and their result is stored at the same place. **Example** – DCR R,DCR M
DCX	R	Decrement the register pair by 1	The contents of the designated register pair are decremented by 1 and their result is stored at the same place. **Example** – DCX R
DAA	None	Decimal adjust accumulator	The contents of the accumulator are changed from a binary value to two 4-bit BCD digits. If the value of the low-order 4-bits in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low-order four bits. If the value of the high-order 4-bits in the accumulator is greater than 9 or if the Carry flag is set, the instruction adds 6 to the high-order four bits. **Example** – DAA

LOGICAL OPERATIONS:

These type instructions performs various logical operations with the contents of the accumulator. 8085 can perform six logical operation which are:

- AND
- OR
- Exclusive-OR
- NOT
- Compare
- Rotate

A 8-bit number can be logically ANDed with the contents of the accumulator. It can also be a content of register or of a memory location. The results are stored in the accumulator. The content of the accumulator can be complimented.

Rotate:
Each bit of the accumulator can be shifted either left or right to the next position.

Compare:

- Any 8-bit number or the content of a register, or content of a memory location can be compared for equality, greater than, or less than, with the contents of the accumulator.
- The result is reflected by zero and carry flags.

Opcode	**Operand**	**Meaning**	**Explanation**
CMP	R M	Compare the register or memory with the accumulator	The contents of the operand (register or memory) are M compared with the contents of the accumulator.
CPI	8-bit data	Compare immediate with the accumulator	The second byte data is compared with the contents of the accumulator.
ANA	R M	Logical AND register or memory with the accumulator	The contents of the accumulator are logically AND with M the contents of the register or memory, and the result is placed in the accumulator.
ANI	8-bit data	Logical AND immediate with the accumulator	The contents of the accumulator are logically AND with the 8-bit data and

			the result is placed in the accumulator.
XRA	R M	Exclusive OR register or memory with the accumulator	The contents of the accumulator are Exclusive OR with M the contents of the register or memory, and the result is placed in the accumulator.
XRI	8-bit data	Exclusive OR immediate with the accumulator	The contents of the accumulator are Exclusive OR with the 8-bit data and the result is placed in the accumulator.
ORA	R M	Logical OR register or memory with the accumulator	The contents of the accumulator are logically OR with M the contents of the register or memory, and result is placed in the accumulator.
ORI	8-bit data	Logical OR immediate with the accumulator	The contents of the accumulator are logically OR with the 8-bit data and the result is placed in the accumulator.
RLC	None	Rotate the accumulator left	Each binary bit of the accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the Carry flag. CY is modified according to bit D7.
RRC	None	Rotate the accumulator right	Each binary bit of the accumulator is rotated right by one position. Bit D0 is placed in the position of D7 as well as in the Carry flag. CY is modified according to bit D0.
RAL	None	Rotate the accumulator left through carry	Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0. CY is modified according to bit D7.
RAR	None	Rotate the accumulator	Each binary bit of the accumulator is rotated right by one position

		right through carry	through the Carry flag. Bit D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7. CY is modified according to bit D0.
CMA	None	Complement accumulator	The contents of the accumulator are complemented. No flags are affected.
CMC	None	Complement carry	The Carry flag is complemented. No other flags are affected.
STC	None	Set Carry	Set Carry

BRANCHING OPERATIONS:

This group of instruction transfers the control of microprocessor from one location to another location. 8085 can perform four types of branching operations. These are:

- JMP-Jump within a program.
- CALL-Jump from main program to sub-routine.
- RET-Jump from sub-routine to main program.
- RST-Jump from main program to instruction sub routine.

Jump:

- Conditional jumps are the important aspect of the decision-making process in the programming of a microprocessor.
- These instructions tests for a certain conditions and alter the program sequence when the condition is met.
- For example zero or carry flag, In addition, the instruction set also includes an instruction called unconditional jump.

Call, return, and restart:

- These type of instructions changes the sequence of a program either by calling a sub-routine or returning from a sub-routine.

- The conditional call and return instructions can also test the condition flags.

1. Jump Instructions: –

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

(a) Unconditional Jump Instructions:

- Transfers the program sequence to the described memory address.

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

(b) Conditional Jump Instructions:

- Transfers the program sequences 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
JNC	address	Jumps to the address if carry flag is 0	JNC 2050
JZ	address	Jumps to the address if zero flag is 1	JZ 2050
JNZ	address	Jumps to the address if zero flag is 0	JNZ 2050
JPE	address	Jumps to the address if parity flag is 1	JPE 2050
JPO	address	Jumps to the address if parity flag is 0	JPO 2050
JM	address	Jumps to the address if sign flag is 1	JM 2050
JP	address	Jumps to the address if sign flag 0	JP 2050

2. Call Instructions:–

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

(a) Unconditional Call Instructions:

- It transfers the program sequence to the memory address given in the operand.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
CALL	address	Unconditionally calls	CALL 2050

(b) Conditional Call Instructions:

Only if the condition is satisfied, the instructions executes.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
CC	address	Call if carry flag is 1	CC 2050
CNC	address	Call if carry flag is 0	CNC 2050
CZ	address	Calls if zero flag is 1	CZ 2050
CNZ	address	Calls if zero flag is 0	CNZ 2050
CPE	address	Calls if parity flag is 1	CPE 2050
CPO	address	Calls if parity flag is 0	CPO 2050
CM	address	Calls if sign flag is 1	CM 2050
CP	address	Calls if sign flag is 0	CP 2050

3. Return Instructions: –

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

(a) Unconditional Return Instruction:

- The program sequence is transferred unconditionally from the subroutine to the calling program.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
RET	none	Return from the subroutine unconditionally	RET

(b) Conditional Return Instruction:

The program sequence is transferred unconditionally from the subroutine to the calling program only 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
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 &MACHINE-CONTROL OPERATIONS:

These type of instructions controls the machine functions, such as halt, interrupt, or do nothing.

Opcode	Operand	Meaning	Explanation

NOP	None	No operation	No operation is performed, i.e., the instruction is fetched and decoded.
HLT	None	Halt and enter wait state	The CPU finishes executing the current instruction and stops further execution. An interrupt or reset is necessary to exit from the halt state.
DI	None	Disable interrupts	The interrupt enable flip-flop is reset and all the interrupts are disabled except TRAP.
EI	None	Enable interrupts	The interrupt enable flip-flop is set and all the interrupts are enabled.
RIM	None	Read interrupt mask	This instruction is used to read the status of interrupts 7.5, 6.5, 5.5 and read serial data input bit.
SIM	None	Set interrupt mask	This instruction is used to implement the interrupts 7.5, 6.5, 5.5, and serial data output.

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

I/O instructions are as follows:
IN - Initiate Input Operation
OUT - Initiate Output Operation

2.4 ASSEMBLY LANGUAGE PROGRAMMING OF 8085:

What is Assembly Language Program?

- Machine language and Hex code instructions are very difficult for the programmer.
- Hence for programmer, the instructions of microprocessor are made in the form of English abbreviation (short form). These instructions are name as Assembly Language instructions or mnemonics.
- The combinations of different mnemonics are known as Assembly Language Program and it is a low level language.

<u>Examples of assembly language program</u>

Loading Register or Memory with Data

Example 1: Write a program to transfer 07 H in register L.

Memory Address	Machine Code	Mnemonics	Operands	Comments
2000 H	2E, 07	MVI	L, 07	Move immediate 07 in register L
2002 H	76	HLT		Stop or terminate the program

- The instruction MVI L, 07 will move the data 07 to the register L.
- The instruction will stop the program.
- The machine code for the instruction MVI L, 07 is 2E, 07.
- The 1st byte of the machine code is 2E which is the Hex code for the instruction MVI L.
- The second byte is the data 07. The machine code for HLT is 76.
- The machine codes are fetch in the memory locations, starting from the memory locations 2000 H.
- Memory location 2000 H contains 2E, 2001 H contains 07 and memory location 2002 H contain 76, After the execution of a program, the contents of Register L can be examined which are 07.

Memory Address	Machine Code	Mnemonics	Operands	Comments
2000 H	3E, 08	MVI	A,08	Get 08 in register A
2002 H	4F	MOV	C,A	Move the contents of register A to register C
2003 H	76	HLT		Halt

Example 2 Write a program to load register A with 08 H and then move it to register C.

- In this program the instruction MVI A, ON H will place the given data 08 1H in the register A.
- The Hex code for MVI A, 08 H is 3E, 08 IH where 3E is the Hex code for MVI A.
- The instruction MOV C, A will move the contents of register A to the register C. Its machine code is 4F.
- With this instruction the data of register A is copies into the register C. It means the given data, is 08 H which was previously placed in register A is now copied into the register C.
- The instruction HLT whose machine code is 76 stops the program.
- The memory locations required for this program are 2000 H to 2003 H. Any other memory locations can be selected. After the execution of a program, the contents of register C can be examined.

Example 3. Write a program to load the contents of memory location 2050 H into accumulator and then move this data into register B

Memory Address	Machine Code	Mnemonics	Operands	Comments
2000 H	3A, 50, 20	LDA	2050 H	Load the contents of memory location 2050 H into the accumulator
2002 H	47	MOV	B,A	Move the contents of register A to register B
2004 H	76	HLT		Stop

- The instruction LDA 2050 H will load the contents of memory location 2050 H into the accumulator.
- The machine code for the instruction LDA is 3A.
- The instruction MOV B. A (Machine code 47) will move the contents of Accumulator to the register B.
- First of all data 07 is fetch in the memory location 2050.
- Then memory locations 2000 H contain 3A, 2001 H contain 50 H, 2002 H contains 20 H, 2003 H contains 47 H and 2004 H contains 76 H.

- After execution of a program, the contents of register B can be examined.

Example 4. Write a program to add two 8-bit numbers.

MEMORY ADDRESS	**MACHINE CODE**	**MNEMONICS**	**OPERANDS**	**COMMENTS**
2000	21,01,25	LXI	H,2501H	Get address of first number in H-L pair.
2003	7E	MOV	A,M	1st number in accumulator.
2004	23	INX	H	Increment content of H-L pair.
2005	86	ADD	M	Add 1st and 2nd numbers.
2006	32,03,25	STA	2503H	Store sum in 2503H.
2009	76	HLT		Stop the program.

EXPLANATION:

- The 1stnumber was stored in the memory location 2501H.
- 2501 was placed in H-L pair by the execution of the instruction LXI H, 2501H.
- The instruction MOV A,M moved the content of the memory location addressed by H-L pair to the accumulator.
- Thus the 1stnumber 49H which was in the 2501H was placed in the accumulator.
- The INX H increased the content of H-L pair from 2501 to 2502H.
- The instruction ADD M added the content of the memory location addressed by H-L pair with the accumulator.
- The result got stored in the accumulator.
  The instruction STA 2503H stored the sum in the memory location 2503H.
- The instruction HLT ended the program.

Example 5. Write a program to subtract two 8-bit numbers.

MEMORY ADDRESS	**MACHINE CODES**	**MNEMONICS**	**OPERAND**	**COMMENTS**
2000	21,01,25	LXI	H,2501	Get address of 1st in H-L pair.
2003	7E	MOV	A,M	1st number in accumulator.
2004	23	INX	H	Content of H-L pair increases from 2501 to 2502 H
2005	96	SUB	M	1stnumber- 2ndnumber.
2006	23	INX	H	Content of H-L pair becomes 2503 H.
2007	77	MOV	M,A	Store result in 2503 H.

2008	76	HLT		Stop the program

EXPLANATION:

- The first no. was stored in the memory location 2501 H.
- 2501 H was placed in H-L pair by the execution of the instruction LXI H, 2501 H.
- The instruction MOV A, M moved the content of the memory location addressed by H-L pair to the accumulator.
- Thus the first no. 49H which was in the 2501 H was placed in the accumulator.
- The INX H increased the content of H-L pair from 2501 to 2502 H.
- The instruction SUB M subtracted the content of the memory location addressed by H-L pair from the accumulator.
- The second no. which was in the memory location 2502 H was subtracted from the first no. which was in the accumulator.
- The result got stored in the accumulator.
- The INX H increased the content of H-L pair from 2502 to 2503 H.
- The instruction MOV M, A moved the content of the accumulator to the memory location addressed by H-L pair to the accumulator.
- The result which was stored in the accumulator got stored in the memory location 2503 H.
- The instruction HLT ended the program.

Example 6.Write an assembly language program in 8085 microprocessor to perform AND operation between lower and higher order nibble of 8 bit number.

Assumption – 8 bit number is stored at memory location 2050. Final result is stored at memory location 3050.

EXPLAINATION:

MEMORY ADDRESS	**MNEMONICS**	**COMMENT**
2000	LDA 2050	A ← M[2050]
2003	ANI 0F	A ←A (AND) 0F
2005	MOV B, A	B ←A
2006	LDA 2050	A ← M[2050]
2009	ANI F0	A ←A (AND) F0
200B	RLC	Rotate accumulator left by one bit without carry

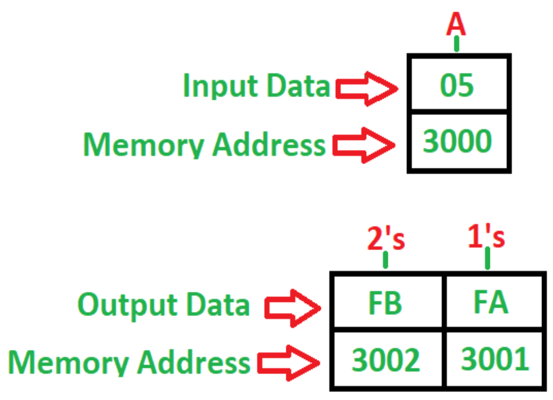
200C	RLC	Rotate accumulator left by one bit without carry
200D	RLC	Rotate accumulator left by one bit without carry
200E	RLC	Rotate accumulator left by one bit without carry
200F	ANA B	A ←A (AND) B
2010	STA 3050	M[3050] ← A
2013	HLT	END

EXPLANATION:

Registers A, B are used for general purpose.

1. **LDA 2050:** load the content of memory location 2050 in accumulator A.
2. **ANI 0F:** perform AND operation in A and 0F. Store the result in A.
3. **MOV B, A:** moves the content of A in register B.
4. **LDA 2050:** load the content of memory location 2050 in accumulator A.
5. **ANI F0:** perform AND operation in A and F0. Store the result in A.
6. **RLC:** rotate the content of A left by one bit without carry. Use this instruction 4 times to reverse the content of A.
7. **ANA B:** perform AND operation in A and B. Store the result in A.
8. **STA 3050:** store the content of A in memory location 3050.
9. **HLT:** stops executing the program and halts any further execution.

Example 7- Write a program to find 1's and 2's complement of 8-bit number where starting address is 2000 and the number is stored at 3000 memory address and store result into 3001 and 3002 memory address.



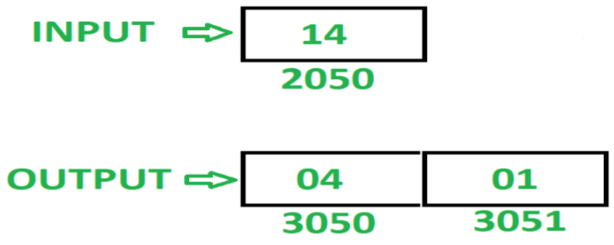
Program –

MEMORY ADDRESS	MNEMONICS	OPERANDS	COMMENT
2000	LDA	[3000]	[A] ← [3000]
2003	CMA		[A] ← [A^]
2004	STA	[3001]	1's complement
2007	ADI	01	[A] ← [A] + 01
2009	STA	[3002]	2's complement
200C	HLT		Stop

EXPLANATION:

1. **A** is an 8-bit accumulator which is used to load and store the data directly
2. **LDA** is used to load accumulator direct using 16-bit address (3 Byte instruction)
3. **CMA** is used to complement content of accumulator (1 Byte instruction)
4. **STA** is used to store accumulator direct using 16-bit address (3 Byte instruction)
5. **ADI** is used to add data into accumulator immediately (2 Byte instruction)
6. **HLT** is used to halt the program

Example8:– Write an assembly language program in 8085 microprocessor to show masking of lower and higher nibble of 8 bit number.
Example –



Assumption: - 8 bit number is stored at memory location 2050. After masking of nibbles, lower order nibble is stored at memory location 3050 and higher order nibble is stored at memory location 3051.

Program –

MEMORY ADDRESS	MNEMONICS	COMMENT
2000	LDA 2050	A ←M[2050]
2003	MOV B, A	B ←A
2004	ANI 0F	A ←A (AND) 0F

2006	STA 3050	M[3050] 🡐 A
2009	MOV A, B	A 🡐 B
200A	ANI 0F	A 🡐A (AND) 0F
200C	RLC	rotate content of A left by 1 bit without carry
200D	RLC	rotate content of A left by 1 bit without carry
200E	RLC	rotate content of A left by 1 bit without carry
200F	RLC	rotate content of A left by 1 bit without carry
2010	STA 3051	M[3051] 🡐A
2013	HLT	END

EXPLANATION:

Registers A, B are used:

1. **LDA 2050:** load the content of memory location 2050 in accumulator A.
2. **MOV B, A:** moves the content of A to B.
3. **ANI 0F:** perform AND operation of A with 0F and store the result back to A.
4. **STA 3050:** store content of A in memory location 3050.
5. **MOV A, B:** moves the content of B in A.
6. **ANI 0F:** perform AND operation of A with 0F and store the result back to A.
7. **RLC:** rotate content of A left by 1 bit without carry. Use this instruction 4 times to reverse the content of A.
8. **STA 3051:** store the content of A in memory location 3051.
9. **HLT:** stops executing the program and halts any further execution.

COUNTER:

- A counter is designed simply by loading appropriate number into one of the registers and using INR or DNR instructions.
- Loop is established to update the count.
- Each count is checked to determine whether it has reached final number; if not, the loop is repeated. C

TIME DELAY:

- Procedure used to design a specific delay.
- A register is loaded with a number, depending on the time delay required and then the register is decremented until it reaches zero by setting up a loop with conditional jump instruction.

Using 8-bit register as counter:

- Counter is another approach to generate a time delay. In this case the program size is smaller. So in this approach we can generate more time delay in less space. The following program will demonstrate the time delay using 8-bit counter.

Program	Time (T-States)
• MVI B,FFH • LOOP: DCR B • JNZ LOOP • RET	7 4 7/10 10

- Here the first instruction will be executed once, it will take 7 T-states. DCR C instruction takes 4 T-states. This will be executed 255 (FF) times. The JNZ instruction takes 10 T-states when it jumps (It jumps 254 times), otherwise it will take 7 T-States. And the RET instruction takes 10 T-States.
- 7 + ((4*255) + (10*254)) + 7 + 10 = 3584. So the time delay will be 3584 * 1/3μs = 1194.66μs. So when we need some small delay, then we can use this technique with some other values in the place of FF.
- This technique can also be done using some nested loops to get larger delays. The following code is showing how we can get some delay with one loop into some other

Using 16-bit register-pair as counter:

- Instead of using 8-bit counter, we can do that kind of task using 16-bit register pair. Using this method more time delay can be generated. This method can be used to get more than 0.5 seconds delay. Let us see and example.

Program	Time (T-States)
LXI B,FFFFH **LOOP: DCX B** **MOV A,B** **ORA C** **JNZ LOOP** **RET**	10 6 4 4 10 (For Jump), 7(Skip) 10

- In the above table we have placed the T-States. From that table, if we calculate the time delay, it will be like this:
- 10 + (6 + 4 + 4 + 10) * 65535H – 3 + 10 = 17 + 24 * 65535H = 1572857. So the time delay will be 1572857 * 1/3μs = 0.52428s. Here we are getting nearly 0.5s delay.

- In different program, we need 1s delay. For that case, this program can be executed twice. We can call the Delay subroutine twice or use another outer loop for two-time execution.

<u>Looping, counting and indexing (Call/JMP)</u>

To perform a repetitive task, commonly used techniques are looping, counting, and indexing. To add data bytes stored in memory, for example, the following steps are necessary.

LOOPING

- The programming technique used to instruct the microprocessor to repeat tasks is called looping.
- This task is accomplished by using jump instructions.
- Define the task to be repeated is called **Looping**.
- A loop is set up by using either a conditional Jump or an unconditional Jump as illustrated in Examples.

COUNTING:

- Specify how many times the task is to be repeated is called **Counting**.
- The counter is set by loading a count (number of times the task is to be repeated) .into a register or a register pair, and the counting is done by decrementing the count every time the loop is repeated. The counter can also be set up to count from 0 to the final count using increment instructions.

INDEXING:

- Specify the location of the data is called **Indexing**.
- The starting location of the data can be specified by loading the memory address into a register pair and using the register pair as a memory pointer or index.

SETTING FLAGS:

- Indicate the end of the repetitive task is called **Setting Flags**.
- The end of repetition is indicated by the flag of the conditional Jump instruction. When the condition is true, the loop is repeated; when the condition is false, the loop execution is terminated, and the execution goes to the next instruction in memory.

CLASSIFICATION OF LOOPS:
1 Conditional loop
2.Unconditional loop

CONTINUOUS LOOP:

- Repeats a task continuously.
- A continuous loop is set up by using the unconditional jump instruction
- A program with a continuous loop does not stop repeating the tasks until the system is reset.

CONDITIONAL LOOP:

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

CONDITIONAL LOOP AND COUNTER:

- A counter is a typical application of the conditional loop.
- A microprocessor needs a counter, flag to accomplish the looping task.
- Counter is set up by loading an appropriate count in a register.
- Counting is performed by either increment or decrement the counter.
- Loop is set up by a conditional jump instruction.
- End of counting is indicated by a flag.

Example:

- Steps to add ten bytes of data stored in memory locations starting ata given location and display the sum.
- The microprocessor needs

1. A counter to count 10 data bytes.
2. An index or a memory pointer to locate where data bytes are stored.
3. To transfer data from a memory location to the microprocessor(ALU)
4. To perform addition
5. Registers for temporary storage of partial answers
6. A flag to indicate the completion of the stack
7. To store or output the result.

Stack and Subroutines programs:

- The stack is a reserved area of the memory in RAM where we can store temporary information.
- Interestingly, the stack is a shared resource as it can be shared by the microprocessor and the programmer.
- The programmer can use the stack to store data. And the microprocessor uses the stack to execute subroutines.

- The 8085 has a 16-bit register known as the 'Stack Pointer.'
- This register's function is to hold the memory address of the stack. This control is given to the programmer.
- The programmer can decide the starting address of the stack by loading the address into the stack pointer register at the beginning of a program.
- The stack works on the principle of First in Last Out. The memory location of the most recent data entry on the stack is known as the Stack Top.

How does a stack work in assembly language?

- We use two main instructions to control the movement of data into a stack and from a stack. These two instructions are **PUSH** and **POP.**
- PUSH – This is the instruction we use to write information on the stack.
- POP – This is the instruction we use to read information from the stack.
- There are two methods to add data to the stack: **Direct method and indirect method**

Direct method:

In the direct method, the stack pointers address is loaded into the stack pointer register directly.

```
LXI SP, 8000H

LXI H, 1234H

PUSH H

POP D

HLT
```

Explanation of the code:

- LXI SP, 8000H – The address of the stack pointer is set to 8000H by loading the number into the stack pointer register.
- LXI H, 1234H – Next, we add a number to the HL pair. The most significant two bits will enter the H register. The least significant two bits will enter the L register.
- PUSH H – The PUSH command will push the contents of the H register first to the stack. Then the contents of the L register will be sent to the stack. So the new stack top will hold 34H.
- POP D – The POP command will remove the contents of the stack and store them to the DE register pair. The top of the stack clears first and enters the E register. The new top of the

stack is 12H now. This one clears last and enters the D register. The contents of the DE register pair is now 1234H.

- HLT – HLT indicates that the program execution needs to stop.

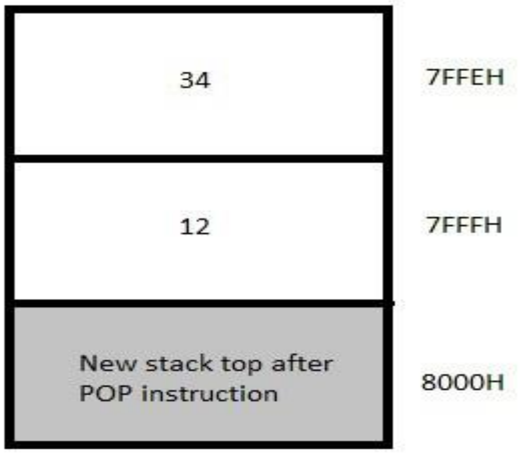
Indirect method:

In the indirect method, the stack pointers address is loaded into the stack pointer register via another register pair.

```
LXI H, 8000H
SPHL
LXI H, 1234H
PUSH H
POP D
HLT
```

Explanation of the code

- LXI H, 8000H – The number that we wish to enter into the stack pointer, 8000H, is loaded into the HL pair register.
- SPHL – This is a special command that we can use to transfer data from HL pair to stack pointer (SP). Now, the contents of the HL pair are in the SP.
- LXI H, 1234H – Next, we add a number to the HL pair. The most significant two bits will enter the H register. The least significant two bits will enter the L register.
- PUSH H – The PUSH command will push the contents of the H register first to the stack. Then the contents of the L register will be sent to the stack. So the new stack top will hold 34H.
- POP D – The POP command will remove the contents of the stack and store them to the DE register pair. The top of the stack clears first and enters the E register. The new top of the stack is 12H now. This one clears last and enters the D register. The contents of the DE register pair is now 1234H.
- HLT – HLT indicates that the program execution needs to stop.
- Both the methods can be shown diagrammatically with the following diagram.



What is a Subroutine is assembly language?

- A subroutine is a small program written separately from the main program to perform a particular task that you may repeatedly require in the main program.
- Essentially, the concept of a subroutine is that it is used to avoid the repetition of smaller programs.
- Subroutines are written separately and are stored in a memory location that is different from the main program.
- Call a subroutine multiple times from the main program using a simple CALL instruction.

BCD to binary conversion in 8085:

(2200H) = 67H
(2300H) = 6 x OAH + 7 = 3CH + 7 = 43H

Source Program:

```
LDA 2200H        : Get the BCD number
MOV B, A         : Save it
ANI OFH          : Mask most significant four bits
MOV C, A         : Save unpacked BCDI in C register
MOV A, B         : Get BCD again
ANI FOH          : Mask least significant four bits
RRC              : Convert most significant four bits into unpacked BCD2
RRC
RRC
RRC
MOV B, A         : Save unpacked BCD2 in B register
```

```
      XRA A                  : Clear accumulator (sum = 0)
      MVI D, 0AH             : Set D as a multiplier of 10
Sum: ADD D                 : Add 10 until (B) = 0
      DCR B                  : Decrement BCD2 by one
      JNZ SUM                : Is multiplication complete? i if not, go back and add again
      ADD C                : Add BCD1
      STA 2300H              : Store the result
      HLT                    : Terminate program execution
```

BCD to HEX conversion in 8085 Microprocessor:

Program

```
LXI H,5000
MOV A,M      ;Initialize memory pointer
ADD A        ;MSD X 2
MOV B,A      ;Store MSD X 2
ADD A        ;MSD X 4
ADD A        ;MSD X 8
ADD B        ;MSD X 10
INX H        ;Point to LSD
ADD M        ;Add to form HEX
INX H
MOV M,A      ;Store the result
HLT
```

Result

Input:

Data 0: 02H in memory location 5000

Data 1: 09H in memory location 5001

Output:

Data 0: 1DH in memory location 5002

Program to find larger of two numbers

PROGRAM:

MEMORY ADDRESS	**MACHINE CODE**	**LABELS**	**MNEMONICS**	**OPERANDS**	**COMMENTS**
2000	21,01,25		LXI	H,2501H	Address of 1st number in H-L pair.
2003	7E		MOV	A,M	1stnumber in accumulator.
2004	23		INX	H	Address of 2nd number in H-L pair.
2005	BE		CMP	M	Compared 2nd number with 1st number. Is the 2nd number> 1st?
2006	D2,0A,20		JNC	AHEAD	No, larger number is in accumulator. Go to AHEAD
2009	7E		MOV	A,M	Yes, get 2nd number in accumulator.
200A	32,03,25	AHEAD	STA	2503H	store larger number in 2503H
200D	76		HLT		Stop the program.

__**Example-1:**__

Data:
2501→98 H
2502→87 H
Result:
2503→98 H and it is stored in the memory location 2503 H.

Program to find smaller of two numbers

PROGRAM:-

ADDRESS	MACHINE CODES	LABELS	MNEMONICS	OPERANDS	COMMENTS
2000	21,01,25		LXI	H,2501H	Address of the 1st number in H-L pair
2003	7E		MOV	A,M	1st number in accumulator
2004	23		INX	H	Address of the 2nd number in H-L pair.
2005	BE		CMP	M	Compare 2nd number with 1st .Is 1st number < 2nd number?
2006	DA,0A,20		JNC	AHEAD	Yes, smaller number is in accumulator. Go to AHEAD.
2009	7E		MOV	A,M	No ,get 2nd number in accumulator
200A	32,03,25	AHEAD	STA	2503H	Store smaller number in 2503H.
200D	76		HLT		stop

EXAMPLE:

DATA:

2501-84H

2502-99H

RESULT:

2503-84H

Program to find the largest number in a data array

PROGRAM:

MEMORY ADDRESS	**MACHINE CODES**	**LABELS**	**MNEMONICS**	**OPERANDS**	**COMMENTS**
2000	21,00,25		LXI	H, 2500H	Address for count in H-L pair.
2003	4E		MOV	C,M	Count in register C.
2004	23		INX	H	Address of the 1st number in H-L pair.
2005	7E		MOV	A,M	1st number in accumulator.
2006	OD		DCR	C	Decrement count.
2007	23		INX	H	Address of next number.
2008	BE		CMP	M	Compare next number with previous maximum. Is next number> previous maximum.
2009	D2,0D,20		JNC	AHEAD	NO, Larger number is in accumulator. GO to the label AHEAD.
200C	7E		MOV	A,M	Yes, get larger number in accumulator.
200D	0D		DCR	C	Decrement Count.
200E	C2, 07, 20		JNZ	LOOP	
2011	32,04,25		STA	2504H	Store result in 2504H.
2014	76		HLT		Stop the Program.

Example-1:

Data:

2500→03

2501→98

2502→75

2503→99

Result: 2504→99

Program to find the smallest number in a data array

PROGRAM:

MEMORY ADDRESS	**MACHINE CODES**	**LABLES**	**MNEMONICS**	**OPERANDS**	**COMMENTS**
2000	21,00,25		LXI	H,2500 H	Get the address for count in the H-L pair
2003	4E		MOV	C,M	Count in register C.
2004	23		INX	H	Get address of 1st number in H-L pair.
2005	7E		MOV	A,M	1stnumber in accumulator.
2006	0D		DCR	C	Decrement count.
2007	23	LOOP	INX	H	Address of next number in H-L pair.
2008	BE		CMP	M	Compare next number with previous smallest. Is previous smallest < next no?
2009	DA,0D,20		JC	AHEAD	Yes, smaller number in the accumulator .Go to AHEAD.
200C	7E		MOV	A,M	No, get next number in accumulator.
200D	0D	AHEAD	DCR	C	Decrement count.
200E	C2,07,20		JNZ	LOOP	
2011	32,50,24		STA	2450 H	Store smallest number in 2450 H.
2014	76		HLT		Stop the program.

2.5 MEMORY AND I/O ADDRESSING-

Memory Addressing-

- A memory address is a unique identifier used by a device or CPU for data tracking.
- This binary address is defined by an ordered and finite sequence allowing the CPU to track the location of each memory byte.

- Modern computers are addressed by bytes which are assigned to memory addresses – binary numbers assigned to a random access memory (RAM) cell that holds up to one byte. Data greater than one byte is consecutively segmented into multiple bytes with a series of corresponding addresses.
- Hardware devices and CPUs track stored data by accessing memory addresses via data buses.
- Before CPU processing, data and programs must be stored in unique memory address locations.

OR

Memory Addressing:

- The bus determines a fixed number of CPU memory addresses assigned according to CPU requirements. The CPU then processes physical memory in individual segments.
- The operating system's read-only memory (ROM) basic input/output system (BIOS) programs and device drivers require memory addresses. Before processing, input device/keyboard data, stored software or secondary storage must be copied to RAM with assigned memory addresses.
- Memory addresses are usually allocated during the boot process. This initiates the startup BIOS on the ROM BIOS chip, which becomes the assigned address. To enable immediate video capability, the first memory addresses are assigned to video ROM and RAM, followed by the following assigned memory addresses:

- Expansion card ROM and RAM chips
- Motherboard dual inline memory modules, single inline memory modules or Rambus inline memory modules
- Other devices

I/O addressing:

- Input/output (I/O) port addresses are used to communicate between devices and software.
- The I/O port address is used to send and receive data for a component.
- As with IRQs, each component will have a unique I/O port assigned.
- There are 65,535 I/O ports in a computer, and they are referenced by a hexadecimal address in the range of 0000h to FFFF H.

UNIT-3: TIMING DIAGRAMS

3.1 Definitions:

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.

Instruction Cycle:

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

or

The time taken by the processor to complete the execution of an instruction. An instruction cycle consists of one to six machine cycles.

Machine Cycle:

The time required to access the memory or input/output devices is called machine cycle.

or

The time required to complete one operation; accessing either the memory or I/O device. A machine cycle consists of three to six T-states.

T-State:

The machine cycle and instruction cycle takes multiple clock periods. A portion of an operation carried out in one system clock period is called as T-state.

or

Time corresponding to one clock period. It is the basic unit to calculate execution of instructions or programs in a processor.

Fetch cycle:

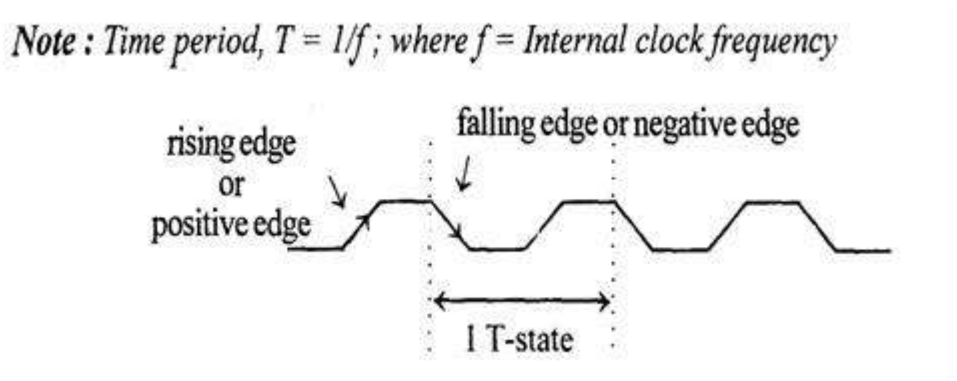
The fetch cycle in a microprocessor comprises of several time states during which the next instruction to be executed is copied (fetched) from the memory location (whose address is in the Program Counter) to the Instruction Register.

3.2 CONCEPT OF TIMING DIAGRAM:

The 8085 microprocessor has 5 (seven) basic machine cycles. They are

1. Opcode fetch cycle (4T)
2. Memory read cycle (3 T)
3. Memory write cycle (3 T)
4. I/O read cycle (3 T)

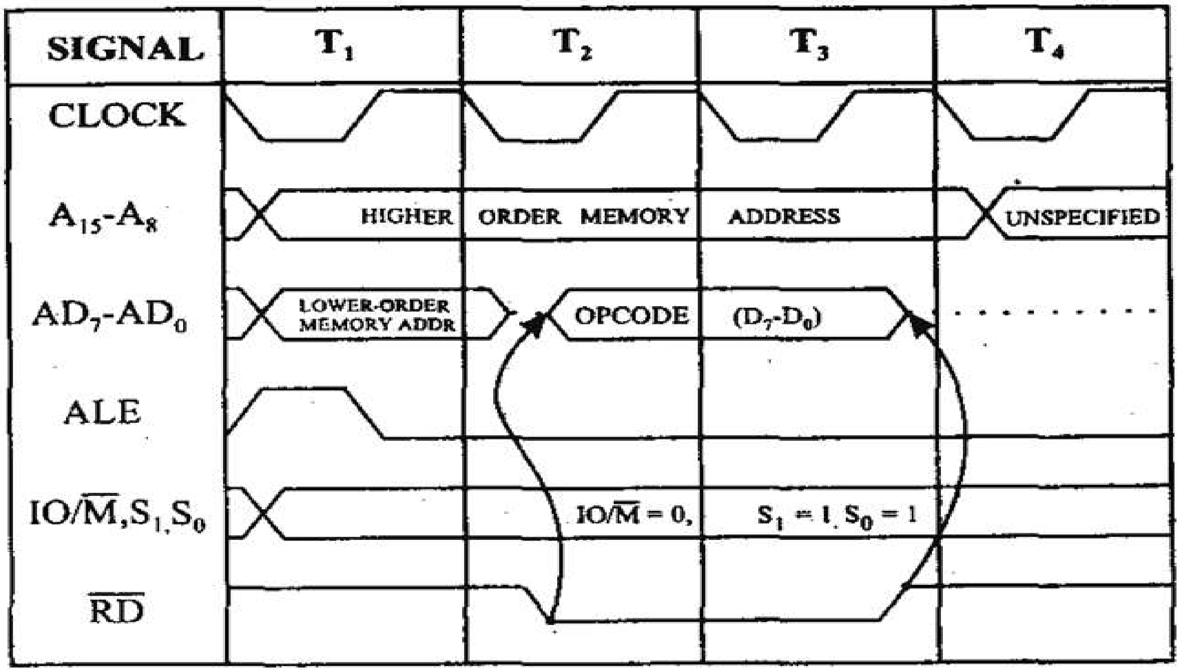
5. I/O write cycle (3 T)


Note : Time period, T = 1/f ; where f = Internal clock frequency
rising edge or positive edge
falling edge or negative edge
1 T-state

- Each instruction of the 8085 processor consists of one to five machine cycles, i.e., when the 8085 processor executes an instruction, it will execute some of the machine cycles in a specific order.
- The processor takes a definite time to execute the machine cycles. 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.
- The T-state starts at the falling edge of a clock.

Opcode Fetch Machine Cycle:

- It is the first step in the execution of any instruction. The timing diagram of this cycle is given below.



- The following points explain the various operations that take place and the signals that are changed during the execution of opcode fetch machine cycle:

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 the levels.
- 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)’** 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)’** signal is deactivated by making it logic 1.

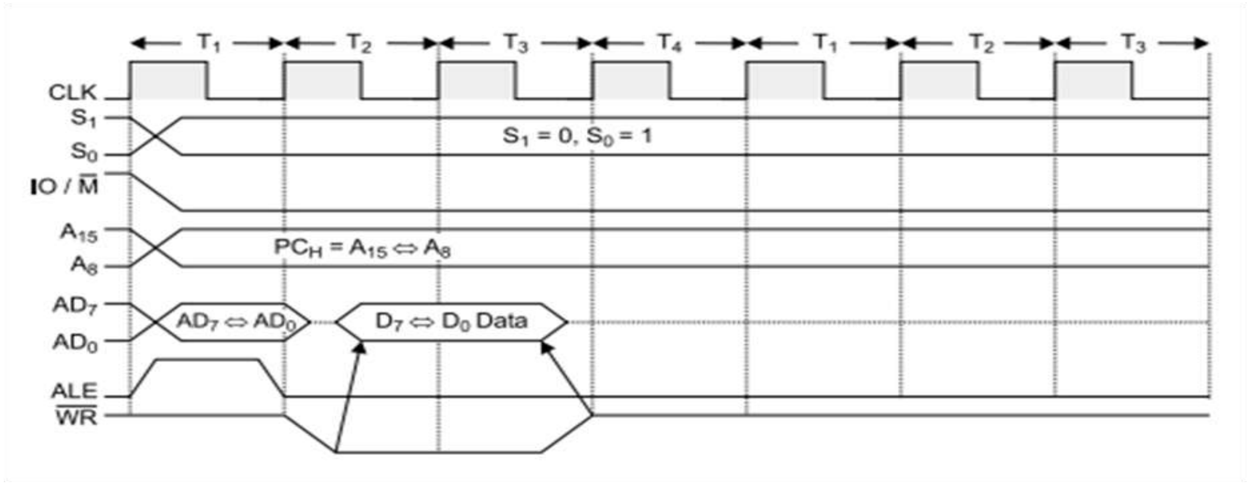
T4 clock cycle:

- 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.

3.3 DRAW TIMING DIAGRAM FOR MEMORY READ, MEMORY WRITE, I/O READ, I/O WRITE MACHINE CYCLE:

Memory Read Machine Cycle:

- The memory read cycle is executed by the processor to read a data byte from memory. The machine cycle is exactly same to opcode fetch except: a) It has three T-states b) The S0 signal is set to 0.



T1 state:

- The higher order address bus (A8-A15) and lower order address and data multiplexed (AD0-AD7) bus.
- ALE goes high so that the memory latches the (AD0-AD7) so that complete 16-bit address are available.
- The microprocessor identifies the memory read machine cycle from the status signals IO/M'=0, S1=1, S0=0. This condition indicates the memory read cycle.

T2 state:

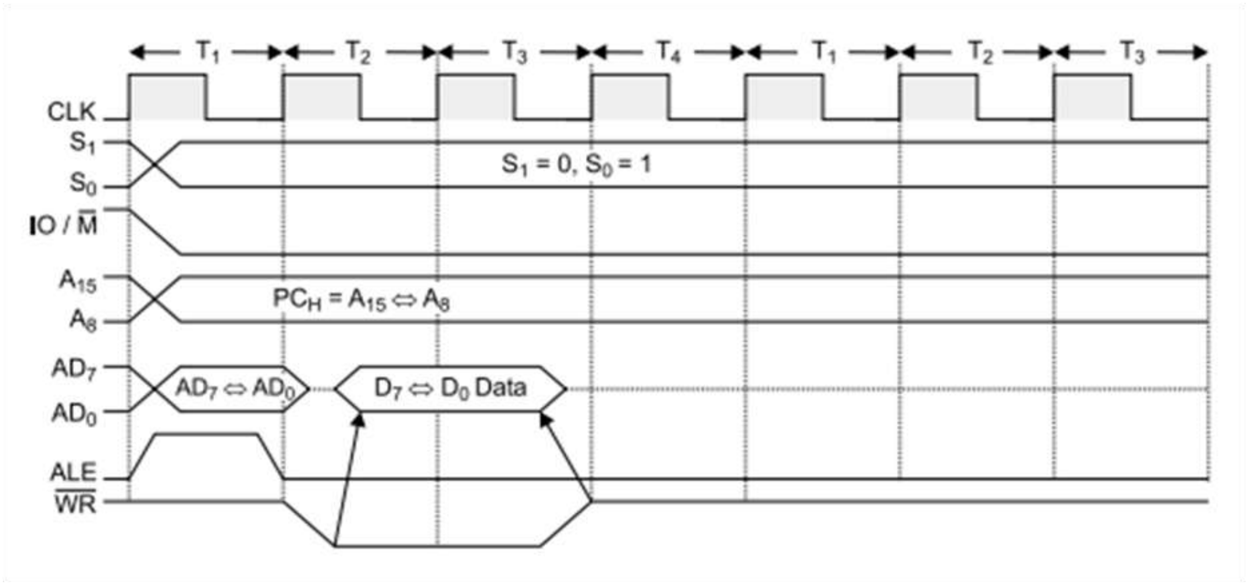
- Selected memory location is placed on the (D0-D7) of the A/D multiplexed bus. RD' goes LOW

T3 State:

- The data which was loaded on the previous state is transferred to the microprocessor.
- In the middle of the T3 state RD' goes high and disables the memory read operation.
- The data which was obtained from the memory is then decoded.

<u>Memory Write Machine Cycle:</u>

- The memory write cycle is executed by the processor to write a data byte in a memory location. The processor takes three T-states and (**WR)'**signal is made low.



T1 state:

- The higher order address bus (A8-A15) and lower order address and data multiplexed (AD0-AD7) bus.
- ALE goes high so that the memory latches the (AD0-AD7) so that complete 16-bit address are available.
- The microprocessor identifies the memory read machine cycle from the status signals IO/M'=0, S1=0, S0=1. This condition indicates the memory read cycle.

T2 state:

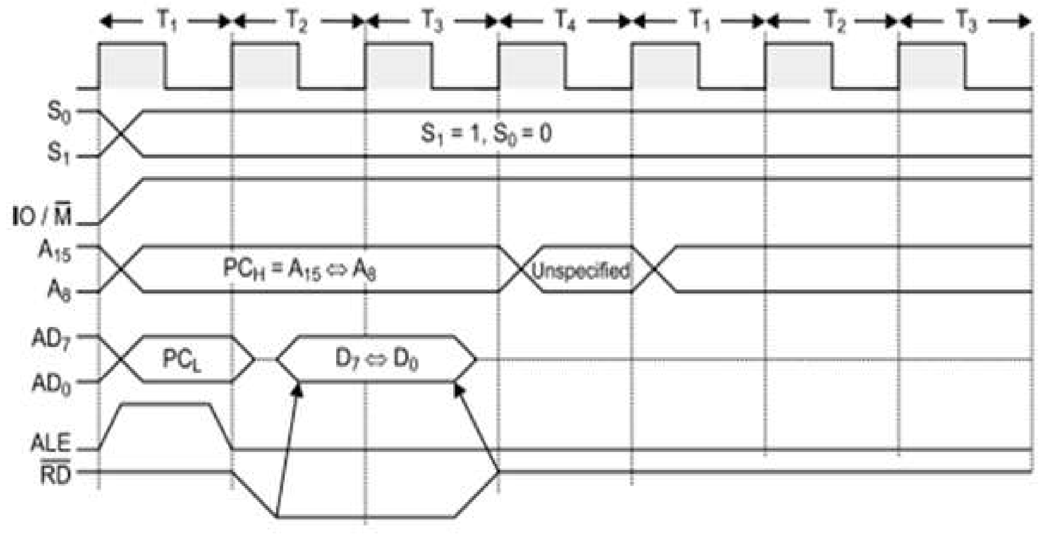
- Selected memory location is placed on the (D0-D7) of the A/D multiplexed bus. WR' goes LOW

T3 State:

- In the middle of the T3 state WR' goes high and disables the memory write operation. The data which was obtained from the memory is then decoded.

<u>I/O Read Cycle:</u>

The I/O read cycle is executed by the processor to read a data byte from I/O port or from peripheral, which is I/O mapped in the system. The 8-bit port address is placed both in the lower and higher order address bus. The processor takes three T-states to execute this machine cycle.



T1 state:

- The higher order address bus (A8-A15) and lower order address and data multiplexed (AD0-AD7) bus.
- ALE goes high so that the memory latches the (AD0-AD7) so that complete 16-bit address are available.
- The microprocessor identifies the I/O read machine cycle from the status signals IO/M'=1, S1=1, S0=0. This condition indicates the I/O read cycle.

T2 state:

- Selected memory location is placed on the (D0-D7) of the A/D multiplexed bus. RD' goes LOW

T3 State:

- The data which was loaded on the previous state is transferred to the microprocessor.
- In the middle of the T3 state RD' goes high and disables the I/O read operation.
- The data which was obtained from the I/O is then decoded.

<u>I/O Write Cycle:</u>

The I/O write cycle is executed by the processor to write a data byte to I/O port or to a peripheral, which is I/O mapped in the system. The processor takes three T-states to execute this machine cycle.

T1 state:

- The higher order address bus (A8-A15) and lower order address and data multiplexed (AD0-AD7) bus.
- ALE goes high so that the memory latches the (AD0-AD7) so that complete 16-bit address are available.

- The microprocessor identifies the I/O read machine cycle from the status signals IO/M'=1, S1=0, S0=1. This condition indicates the I/O read cycle.

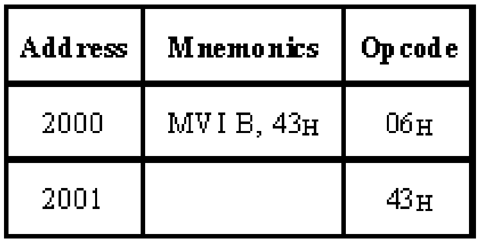
  T2 state:
- Selected memory location is placed on the (D0-D7) of the A/D multiplexed bus. WR' goes LOW

  T3 State:
- In the middle of the T3 state WR' goes high and disables the I/O write operation. The data which was obtained from the I/O is then decoded.

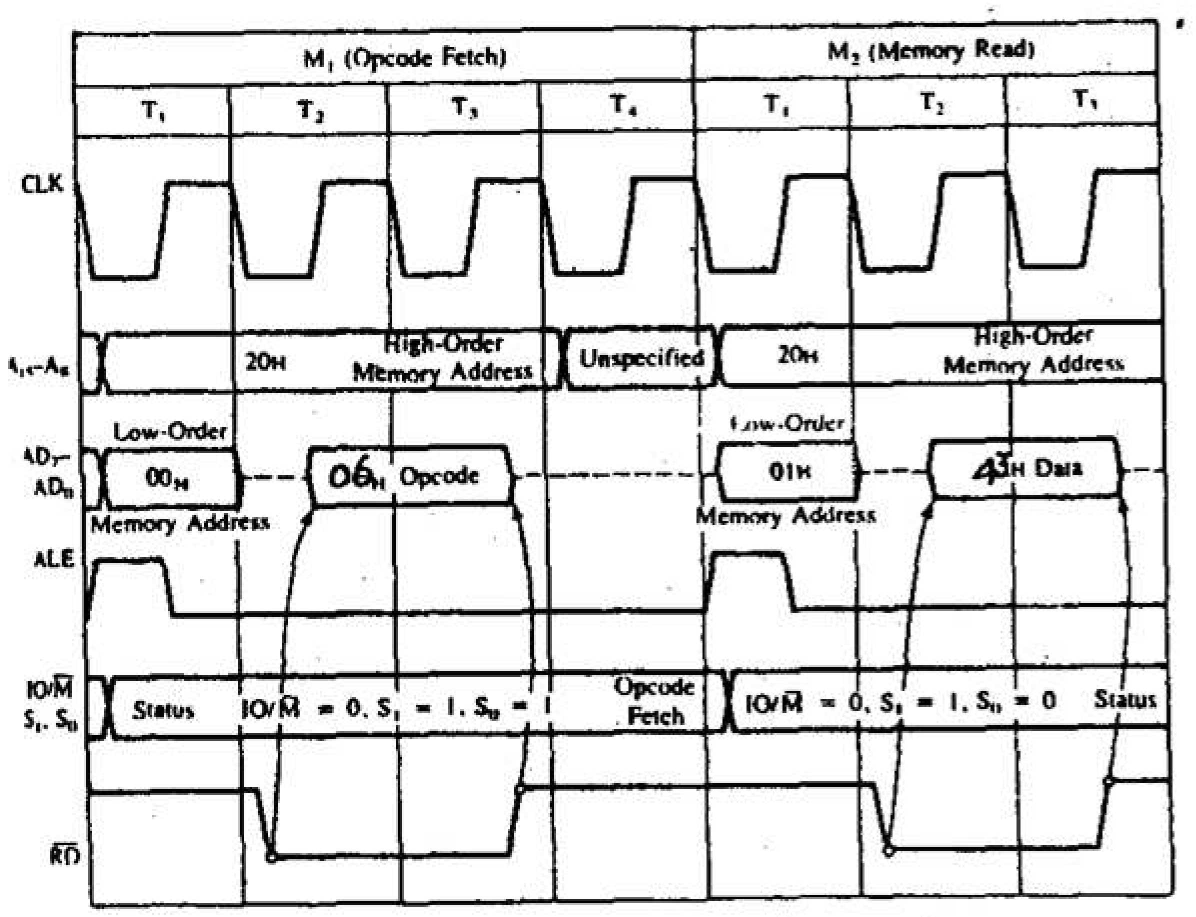
3.4 DRAW A NEAT SKETCH FOR THE TIMING DIAGRAM FOR 8085 INSTRUCTION:

Timing diagram for MVI B, 43H.

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



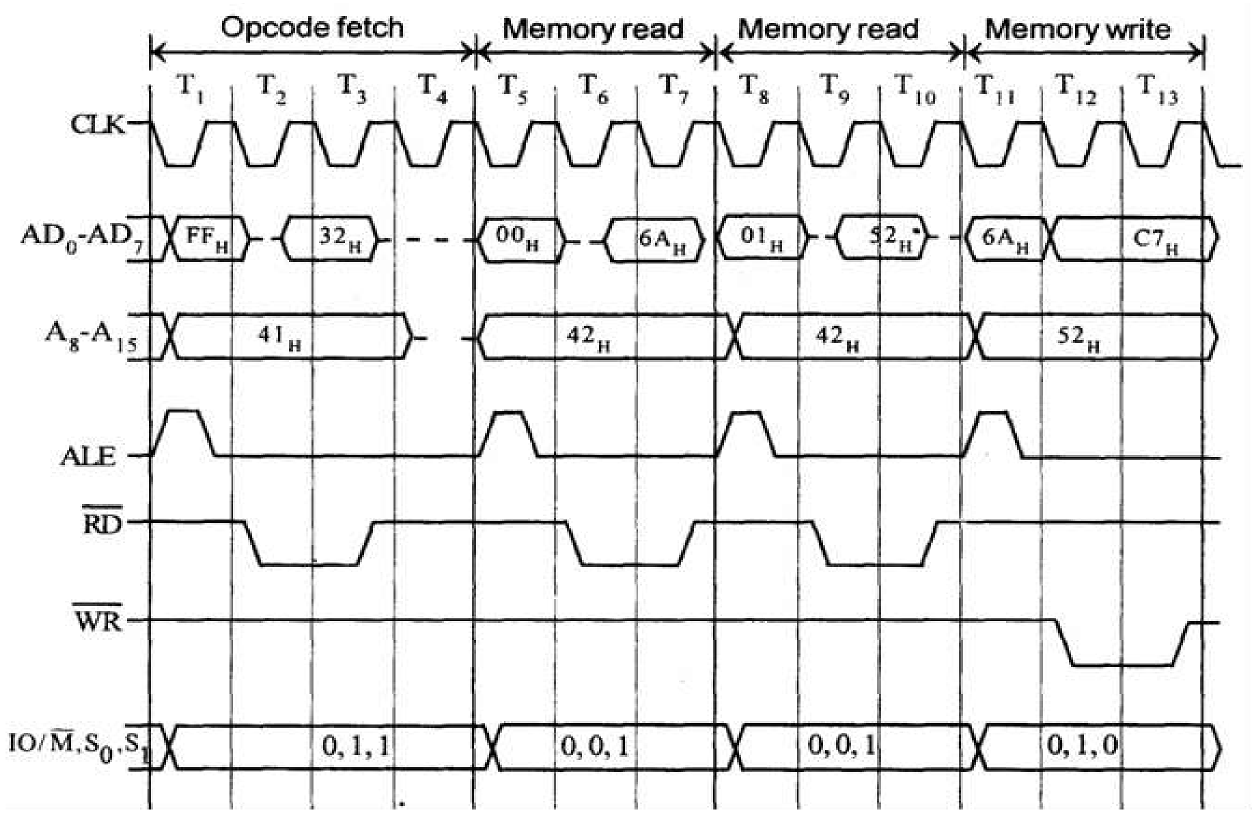
Address	Mnemonics	Opcode
2000	MVI B, 43_H	06_H
2001		43_H



Timing diagram for STA 526AH.

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

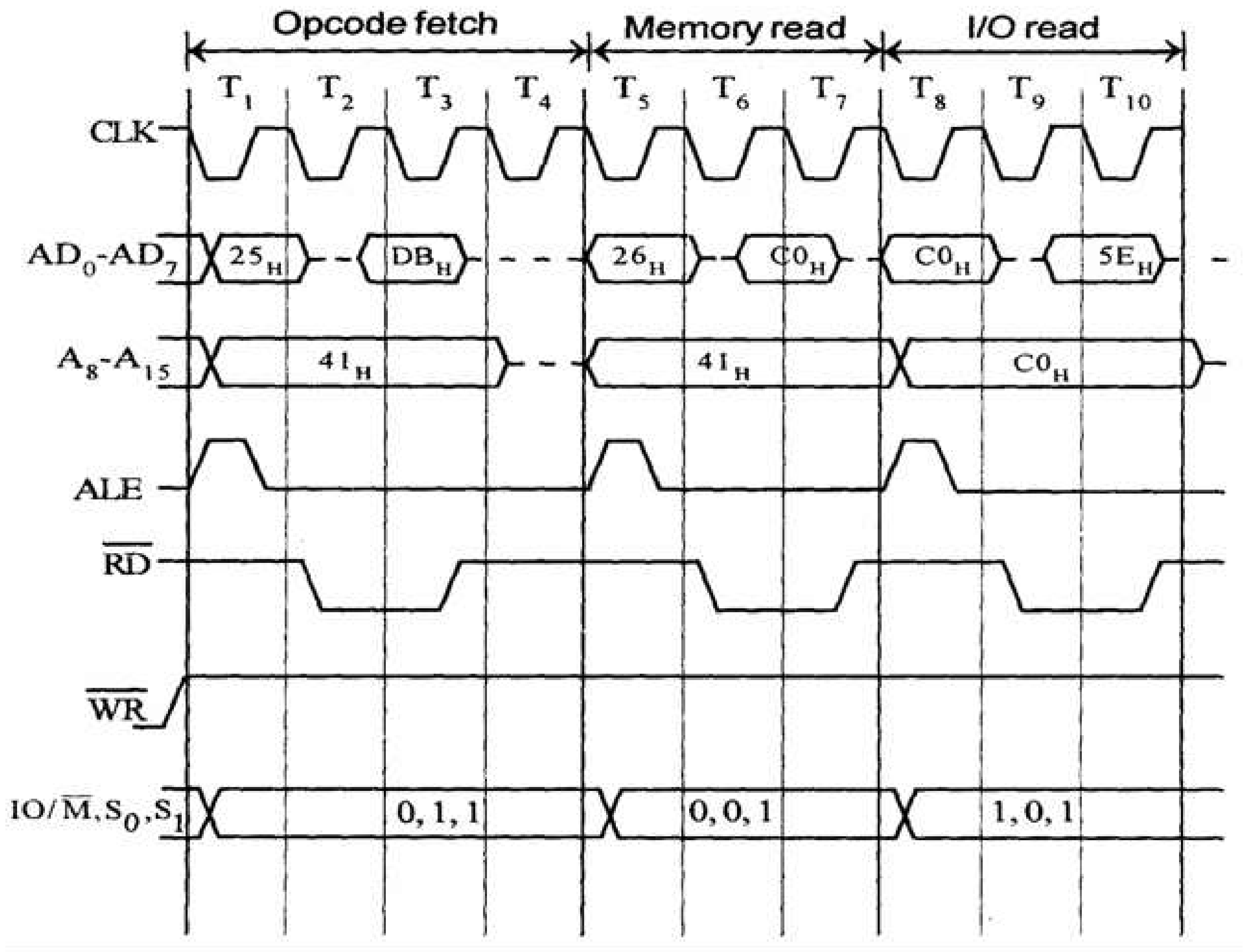
Address	Mnemonics	Opcode
41FF	STA 526A$_H$	32$_H$
4200		6A$_H$
4201		52$_H$



Timing diagram for IN C0H.

- Fetching the Opcode DBH from the memory 4125H.
- Read the port address C0H from 4126H.
- Read the content of port C0H and send it to the accumulator.
- Let the content of port is 5EH.

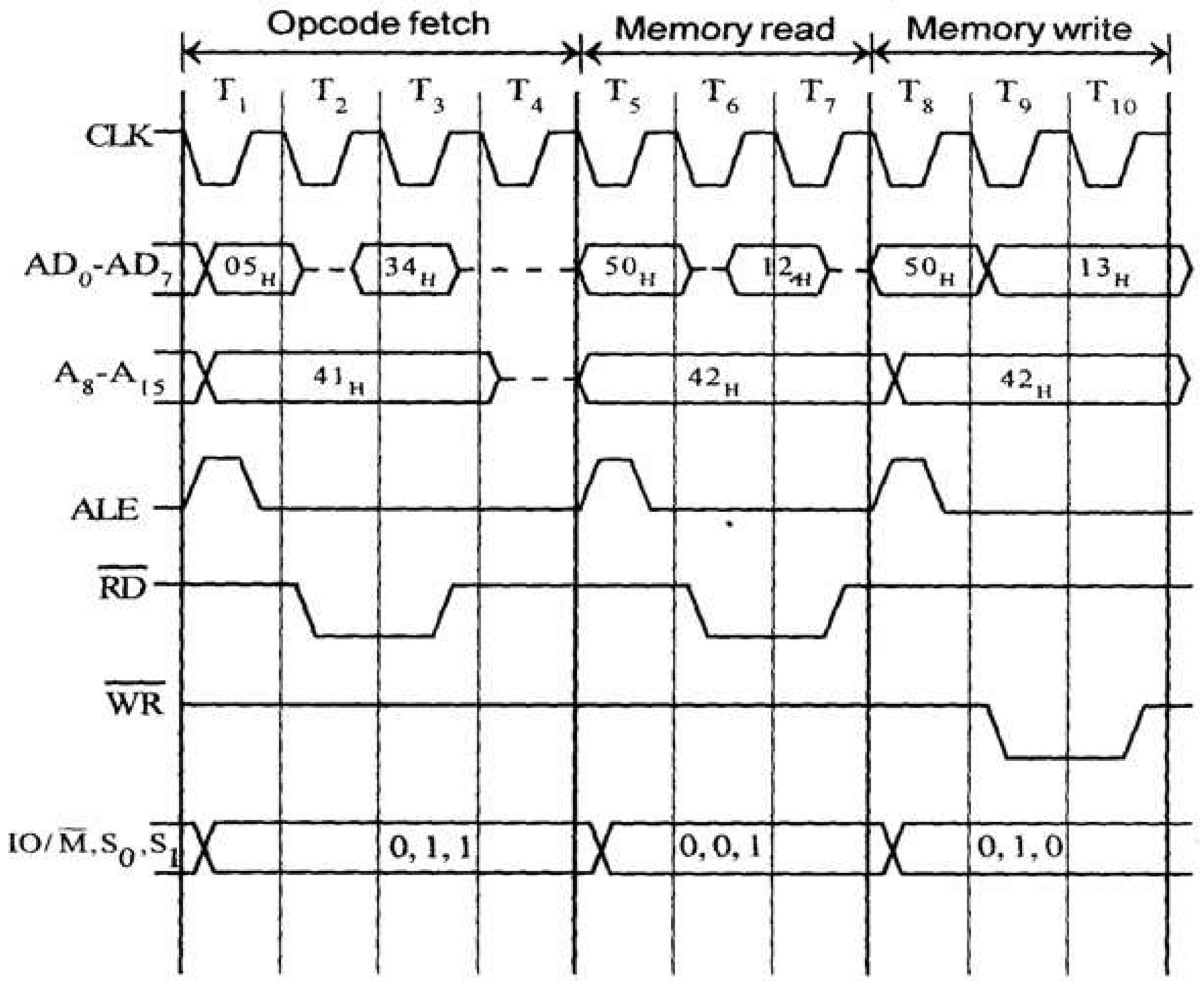
Address	Mnemonics	Opcode
4125	IN C0$_H$	DB$_H$
4126		C0$_H$



Timing diagram for INR M

- Fetching the Opcode 34H from the memory 4105H. (OF cycle)
- Let the memory address (M) be 4250H. (MR cycle -To read Memory address and data)
- Let the content of that memory is 12H.
- Increment the memory content from 12H to 13H. (MW machine cycle)

Address	Mnemonics	Opcode
4105	INR M	34_H


Opcode fetch
Memory read
Memory write
T1 T2 T3 T4 T5 T6 T7 T8 T9 T10
CLK
AD0-AD7
05H
34H
50H
12H
50H
13H
A8-A15
41H
42H
42H
ALE
RD
WR
IO/M, S0, S1
0, 1, 1
0, 0, 1
0, 1, 0

UNIT-4: MICROPROCESSOR BASED SYSTEM DEVELOPMENT AIDS

4.1 CONCEPT TO INTERFACING:

- We know that a microprocessor is the CPU of a computer. A microprocessor can perform some operation on a data and give the output. But to perform the operation we need an input to enter the data and an output to display the results of the operation. So we are using a keyboard and monitor as Input and output along with the processor. Microprocessors engineering involves a lot of other concepts and we also interface memory elements like ROM, EPROM to access the memory.
- Interfacing a microprocessor is to connect it with various peripherals to perform various operations to obtain a desired output.

INTERFACING TYPES:

There are two types of interfacing in 8085 processor.

- Memory Interfacing.
- I/O interfacing.

Purpose of interfacing:

- The interfacing process involves matching the memory requirements with the 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.
- For example for carrying out a READ process, the microprocessor should initiate a read signal which the memory requires to read a data.
- In simple words, the primary function of a memory interfacing circuit is to aid the microprocessor in reading and writing a data to the given register of a memory chip.

Disadvantages of interfacing:

- The main disadvantage with this interfacing is that the microprocessor can perform only one function.
- It functions as an input device if it is connected to buffer.
- It function as an output device if it is connected to latch.
- Thus the capability is very limited in this type of interfacing.

Types of Communication Interface

There are two ways in which a microprocessor can connect with outside world or other memory systems.

1. Serial Communication Interface

2. Parallel Communication interface

Serial Communication Interface:

- In serial communication interface, the interface gets a single byte of data from the microprocessor and sends it bit by bit to other system serially

- The interface also receives data bit by bit serially from the external systems and converts the data into a single byte and transfers it to the microprocessor.

Parallel Communication Interface:

- This interface gets a byte of data from microprocessor and sends it bit by bit to the other systems in simultaneous or parallel.

- The interface also receives data bit by bit simultaneously from the external system and converts the data into a single byte and transfers it to microprocessor.

4.2 MEMORY MAPPING & I/O MAPPING:

MEMORY MAPPING:

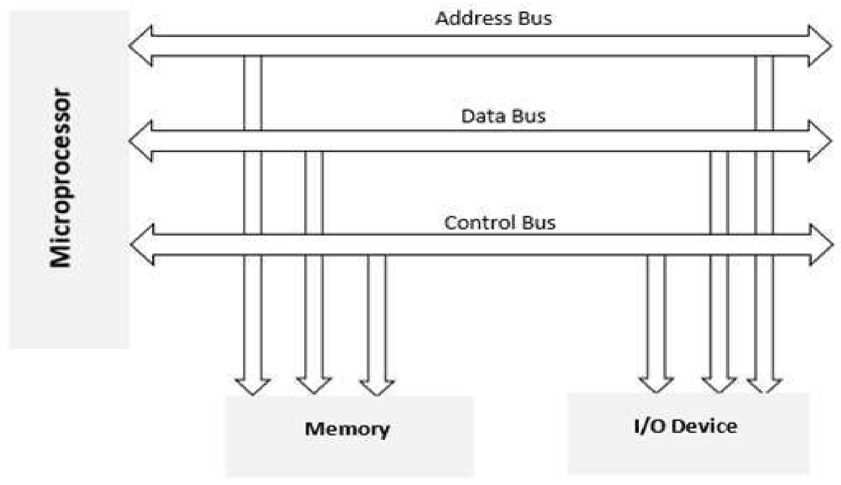
- Memory mapping is a method to expand the memory of the microprocessor.
- Being limited in memory resources, microprocessor needs to be connected to external memory devices like RAM/ROM/EEPROM.
-

- The interfacing between the microprocessor and the memory device by connecting the data and address bus is called memory mapping.

I/O INTERFACING:

- I/O Interfacing is achieved by connecting keyboard (input) and display monitors (output) with the microprocessor.
- We know that keyboard and Displays are used as communication channel with outside world. So it is necessary that we interface keyboard and displays with the microprocessor. This is called I/O interfacing. In this type of interfacing we use latches and buffers for interfacing the keyboards and displays with the microprocessor.

Block diagram of memory and I/O interfacing



I/O Mapping in 8085 Microprocessor:

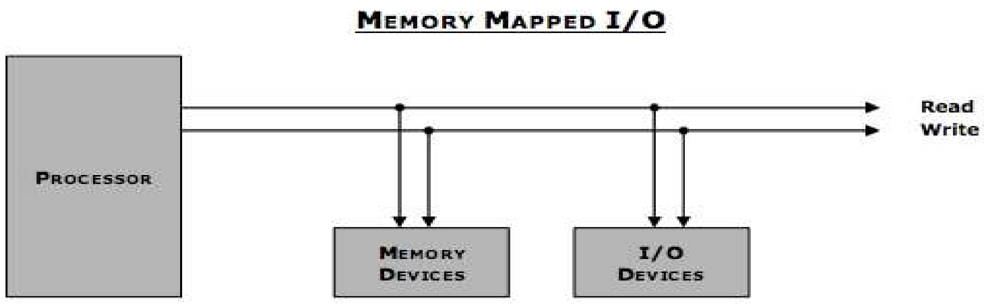
<u>I/O interfacing:</u>
There are two methods of interfacing the Input / Output devices with the microprocessor. They are,

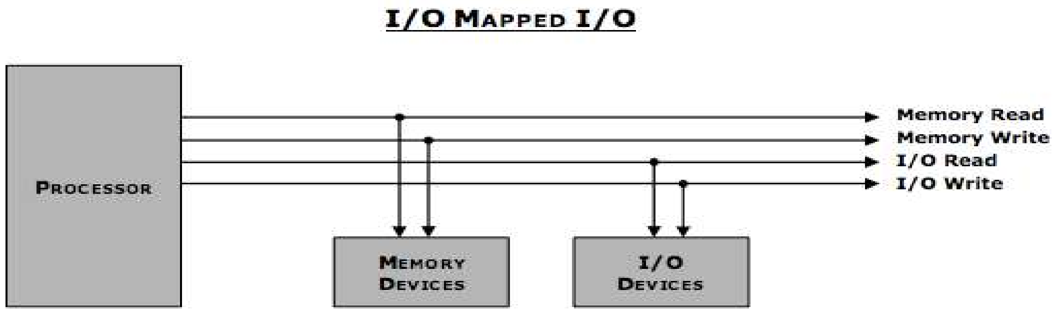
1) Memory mapped I/O and

2) I/O mapped I/O.

	MEMORY MAPPED I/O	I/O MAPPED I/O
1	**I/O devices are mapped into memory space.**	**I/O devices are mapped into I/O space.**
2	I/O devices are allotted **memory addresses**.	I/O devices are allotted **I/O addresses**.
3	Processor **does not differentiate** between memory and I/O. Treats I/O devices also like memory devices.	Processor **differentiates between I/O devices and memory**. It isolates I/O devices.
4	I/O addresses are as big as memory addresses. E.g.in 8085, **I/O addresses will be 16 bit** as memory addresses are also 16-bit.	I/O addresses are smaller than memory addresses. E.g. in 8085, **I/O addresses will be 8 bit** though memory addresses are 16-bit.

5	This allows us to increase the number of I/O devices. E.g. in 8085, we can access up to **2^{16} = 65536 I/O devices**.	This allows us to access limited number of I/O devices. E.g. in 8085, we can access only up to **2^{8} = 256 I/O devices**.
6	We can transfer data from I/O devices using **any instruction like MOV** etc.	We can transfer data from I/O device using **dedicated I/O instructions like IN and OUT ONLY**.
7	Data can be transferred using **any register** of the processor.	Data can be transferred only using a fixed register. E.g. in 8085 only **"A" register**.
8	We need **only two control** signals in the system: Read and Write.	We need **four control signals**: Memory Read, Memory Write and I/O Read and I/O Write
9	Memory addresses are big so **address decoding will be slower**.	I/O addresses are smaller so **address decoding will be faster**.
10	Address decoding will be **more complex and costly**.	Address decoding will be **simpler and cheaper**.





MEMORY MAPPED I/O:

In this method the I/O devices are treated like the memory. A part of the memory address space is used for the I/O devices. The memory mapped I/O scheme is shown in figure.

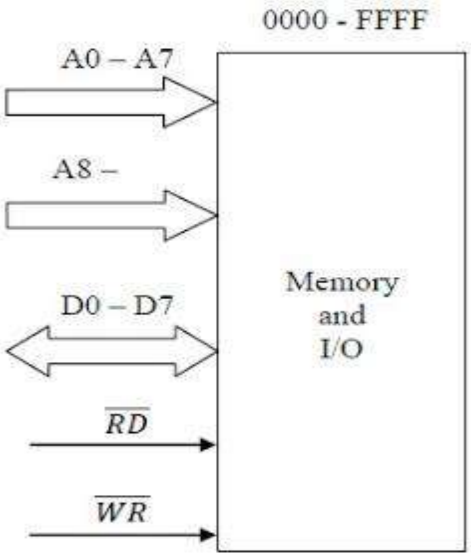


Figure: Memory mapped I/O scheme

• In memory mapped I/O scheme, the same address space is used for both memory and I/O devices.

• The microprocessor uses the sixteen address line $A_0 – A_7$ and $A_8 – A_{15}$ for the memory as well as for the I/O devices.

• The I/O devices share the address space with the memory. All the memory related instructions are used for addressing I/O devices also.

• No separate IN and OUT instructions are required in memory mapped I/O scheme.

• IO/M’ pin is not required.

Steps for memory operations (memory read and memory write):

- When the memory related instructions like LDA and STA are used, the microprocessor places the 16-bit address on the address bus.
- □□’ is activated for read operation and □□’ is activated for write operation.

Steps for I/O operations (I/O read and I/O write):

- When the memory related instructions like LDA and STA are used, the microprocessor places the 16-bit address on the address bus.
- □□’ is activated for read operation and □□’ is activated for write operation.

I/O MAPPED I/O:

In this method, I/O devices are treated as I/O devices and memory is treated as memory. Separate address space is used for memory and I/O. The I/O mapped I/O scheme is shown in figure.

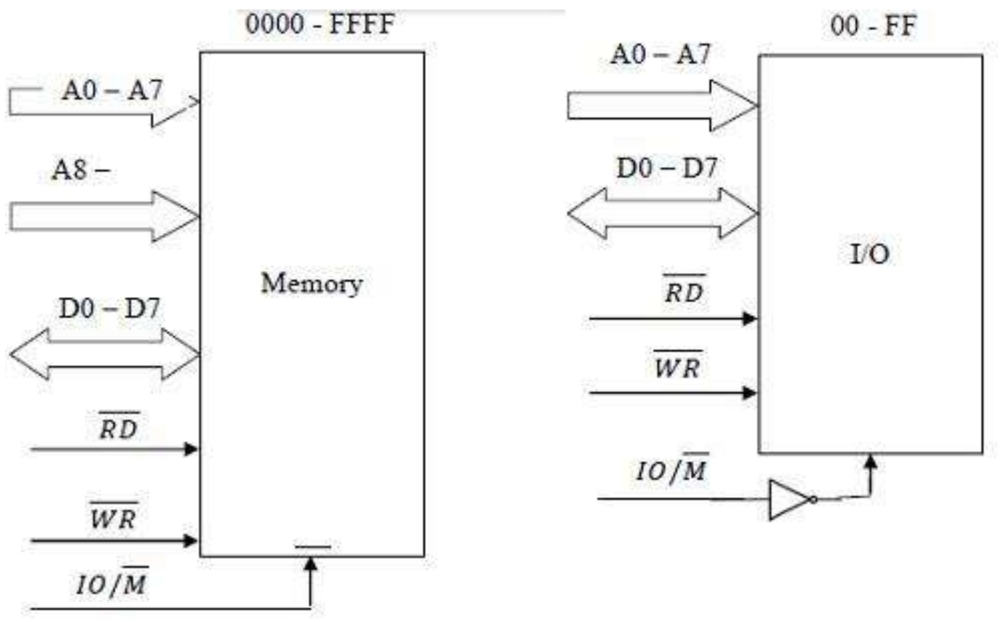


Figure: I/O mapped I/O scheme

• In I/O mapped I/O scheme, the microprocessor uses the sixteen address lines A_0 – A_7 and A_8 – A_{15} for the memory and eight address lines A_0 to A_7 to identify an input / output device.

• Here, the full address space 0000 – FFFF is used for the memory and a separate address space 00 – FF is used for the I/O devices.

• Hence, the microprocessor can address 65536 (2^{16}) memory locations 256 (2^8) input devices and 256 (2^8) output devices separately.

• IN and OUT instructions are used to activate the IO/▯’ signal.

• When IO/▯’ is low, the memory is selected for reading and writing operations.

• When IO/▯’ is high, the I/O port is selected for reading and writing operations.

Steps for memory operations (memory read and memory write):

- When the memory related instructions like LDA and STA are used, the microprocessor places the 16-bit address on the address bus.
- The microprocessor makes the IO/▯’ line low.
- The microprocessor makes the ▯▯’ low for read operation and ▯▯’ low for write operation.

Steps for I/O operations (I/O read and I/O write):

- 1 When the I/O related instructions like IN and OUT are used, the microprocessor places the 8-bit address on the address bus $A_0 – A_7$ as well as $A_8 – A_{15}$.
- IO/M’ line is made high.
- The microprocessor makes the □□’ low for read operation and □□’ low for write operation.

4.3 MEMORY INTERFACING:

- While executing an instruction, there is a necessity for the microprocessor to access memory frequently for reading various instruction codes and data stored in the memory.
- The read/write operations are monitored by control signals. The microprocessor activates these signals when it wants to read from and write into memory.
- The interfacing circuit aids in accessing the memory requires some signals to read from and write to registers. Similarly the microprocessor transmits some signals for reading or writing a data.

OR

- Memory Interfacing is used when the microprocessor needs to access memory frequently for reading and writing data stored in the memory. It is used when reading/writing to a specific register of a memory chip.

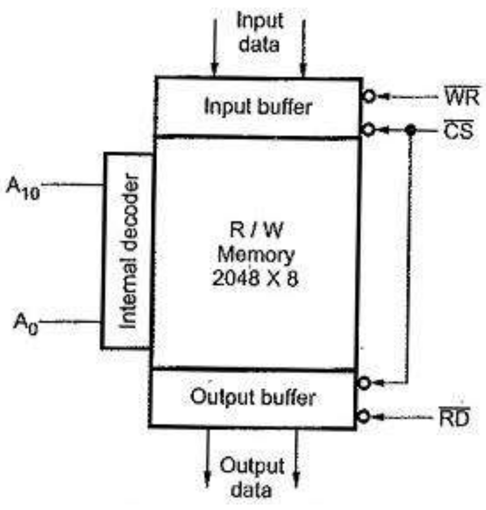
<u>The memory interfacing requires to:</u>

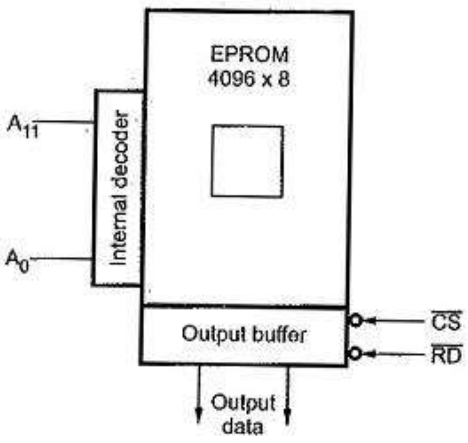
- ➢ Select the chip
- ➢ Identify the register
- ➢ Enable the appropriate buffer.

- Microprocessor system includes memory devices and I/O devices. It is important to note that 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.
- In order to communicate with memory or I/O devices, it is necessary to decode the address from the microprocessor.
- Due to this each device (memory or I/O) can be accessed independently. The following section describes common address decoding techniques.

Memory Structure and its Requirements:

As mentioned earlier, read/write memories consist of an array of registers, in which each register has unique address. The size of the memory is N x M, where N is the number of registers and M is the word length, in number of bits.





Logic diagram for RAM **Logic diagram for EPROM**

INTERFACING EPROM & RAM MEMORIES:

- 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.
- We can place EPROM/RAM 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

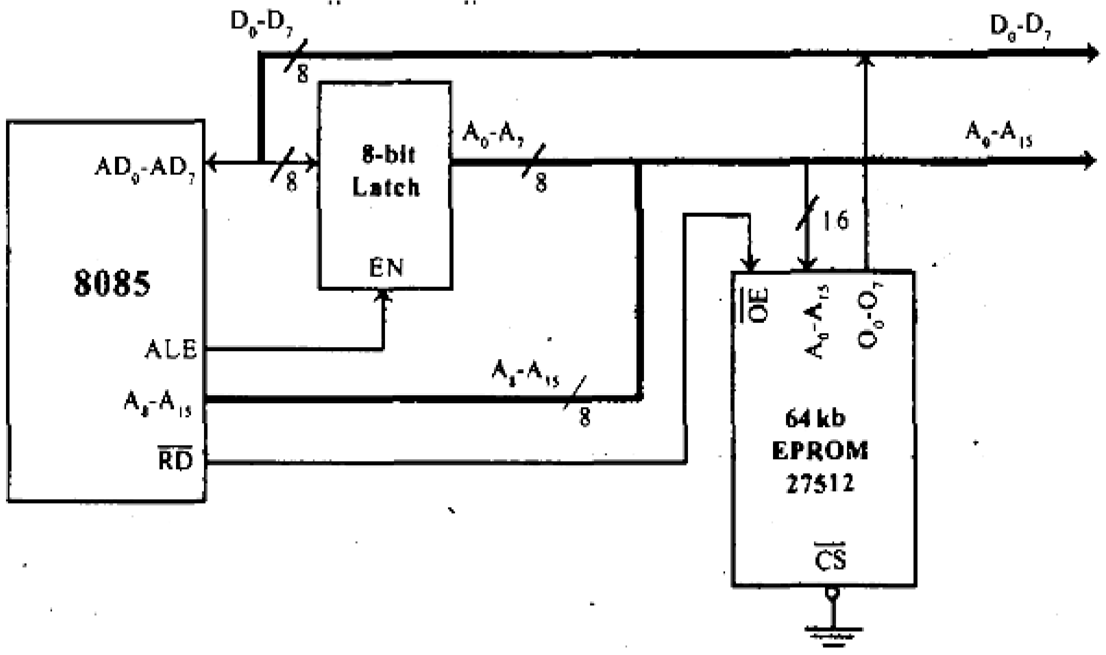
<u>EXAMPLES OF MEMORY INTERFACING</u>

EXAMPLE-1

Consider a system in which the full memory space 64kb is utilized for EPROM memory. Interface the EPROM with 8085 processor.

- The memory capacity is 64 Kbytes. i.e.
- 2^n = 64 x 1000 bytes where n = address lines.

- So, n = 16.
- In this system the entire 16 address lines of the processor are connected to address input pins of memory IC in order to address the internal locations of memory.
- The chip select (CS) pin of EPROM is permanently tied to logic low (i.e., tied to ground).
- Since the processor is connected to EPROM, the active low RD pin is connected to active low output enable pin of EPROM.
- The range of address for EPROM is 0000H to FFFFH.

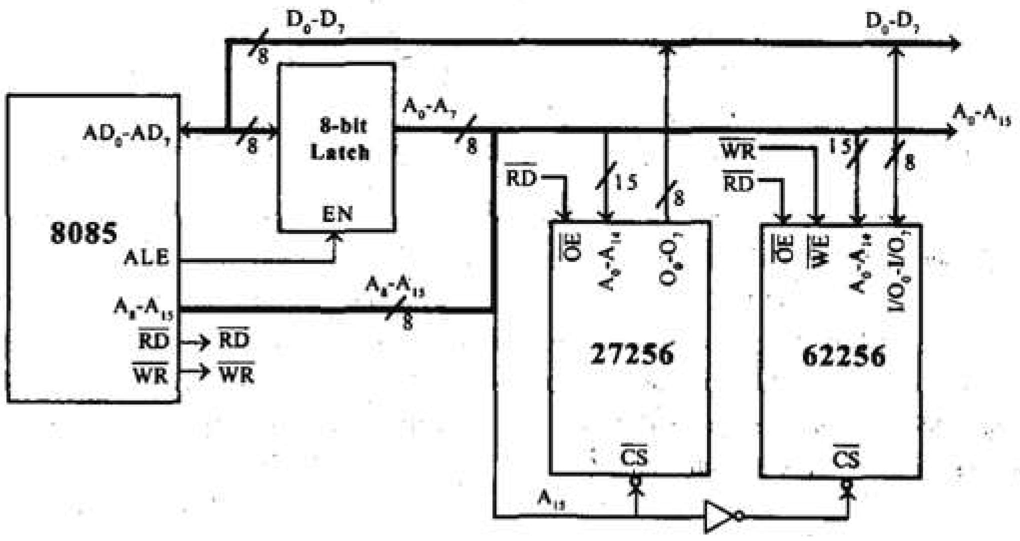


Interfacing 64Kb EPROM with 8085

EXAMPLE-2

Consider a system in which the available 64kb memory space is equally divided between EPROM and RAM. Interface the EPROM and RAM with 8085 processor.

- Implement 32kb memory capacity of EPROM using single IC 27256.
- 32kb RAM capacity is implemented using single IC 62256.
- The 32kb memory requires 15 address lines and so the address lines A0 - A14 of the processor are connected to 15 address pins of both EPROM and RAM.
- The unused address line A15 is used as to chip select. If A15 is 1, it select RAM and if A15 is 0, it select EPROM.
- Inverter is used for selecting the memory.
- The memory used is both Ram and EPROM, so the low RD and WR pins of processor are connected to low WE and OE pins of memory respectively.
- The address range of EPROM will be 0000H to 7FFFH and that of RAM will be 7FFFH to FFFFH.

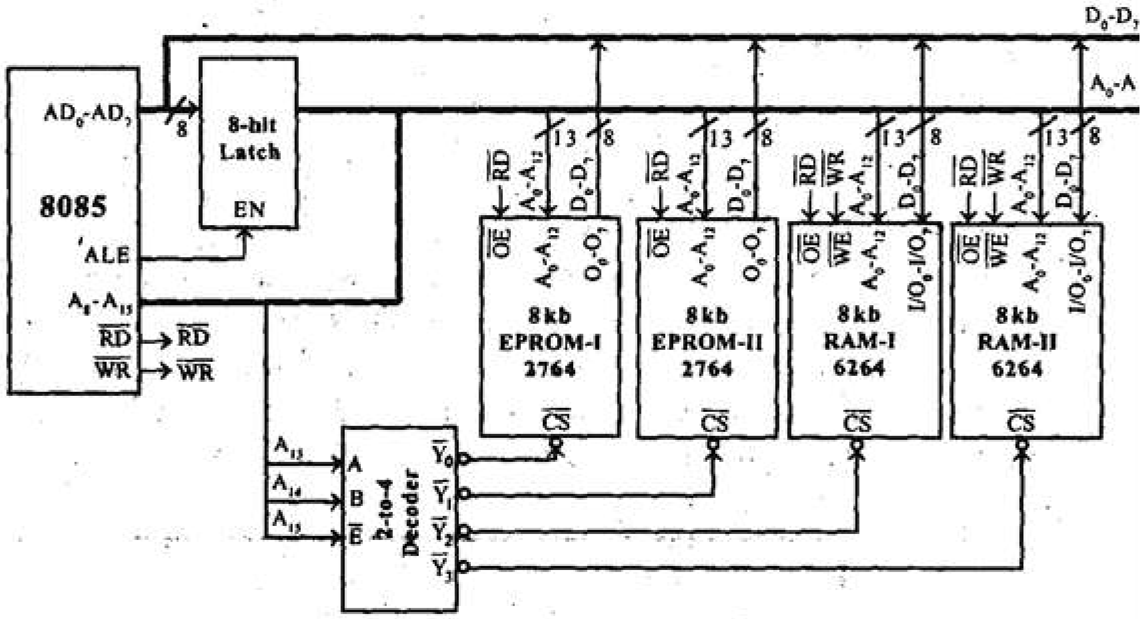


Interfacing 32Kb EPROM and 32Kb RAM with 8085

EXAMPLE-3

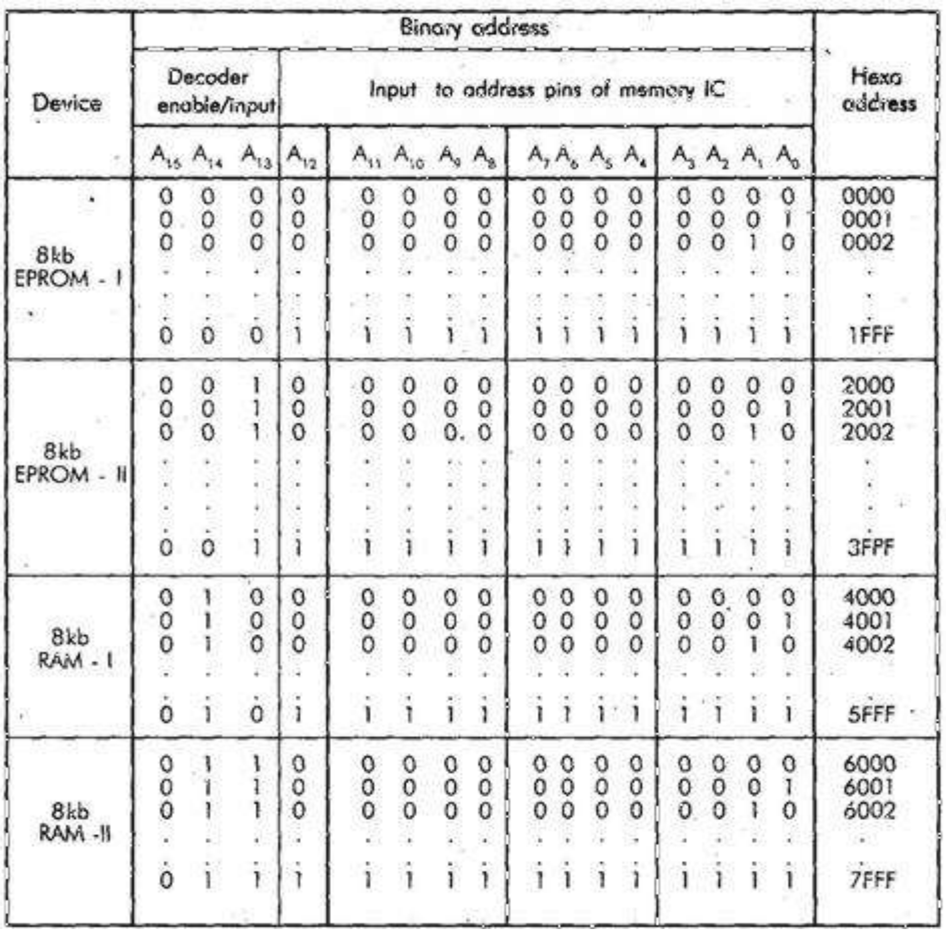
Consider a system in which 32kb memory space is implemented using four numbers of 8kb memory. Interface the EPROM and RAM with 8085 processor.

- The total memory capacity is 32Kb. So, let two number of 8kb n memory be EPROM and the remaining two numbers be RAM.
- Each 8kb memory requires 13 address lines and so the address lines A0- A12 of the processor are connected to 13 address pins of all the memory.
- The address lines and A13 - A14 can be decoded using a 2-to-4 decoder to generate four chip select signals.
- These four chip select signals can be used to select one of the four memory IC at any one time.
- The address line A15 is used as enable for decoder.
- The simplified schematic memory organization is shown.



Interfacing 16Kb EPROM and 16Kb RAM with 8085

➢ The address allotted to each memory IC is shown in following table.

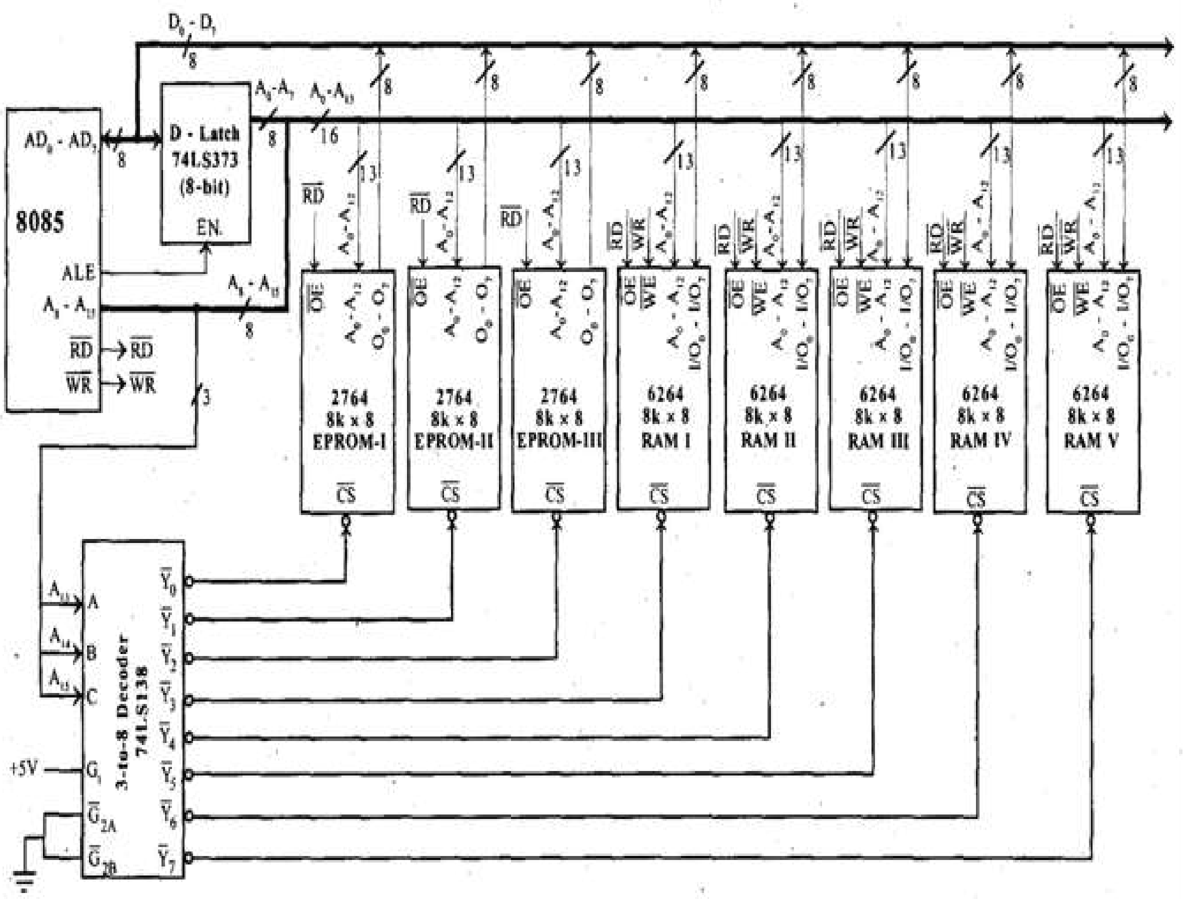


Device	Binary address																Hexa address
	Decoder enable/input			Input to address pins of memory IC													
	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
8kb EPROM - I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0001
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFF
8kb EPROM - II	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000
	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	2001
	0	0	1	0	0	0	0.	0	0	0	0	0	0	0	1	0	2002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFF
8kb RAM - I	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4001
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	4002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	5FFF
8kb RAM -II	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	6000
	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	6001
	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	6002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7FFF

EXAMPLE-4

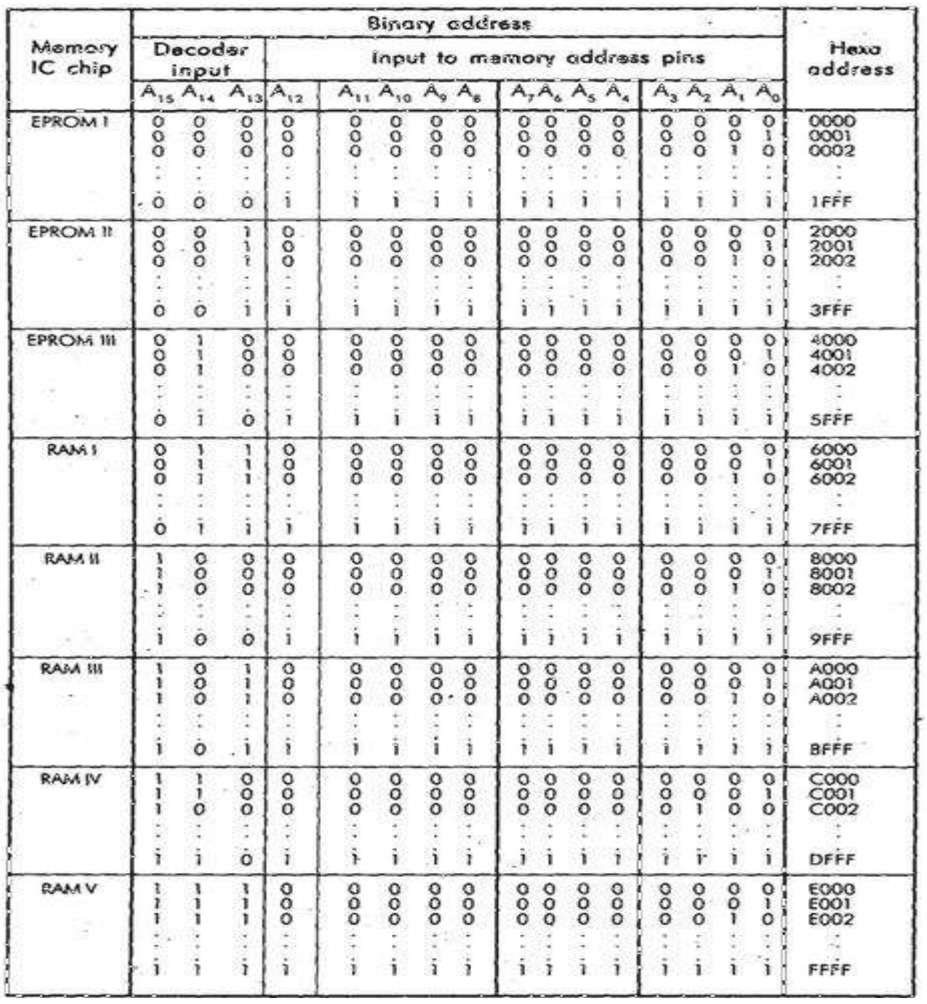
Consider a system in which the 64kb memory space is implemented using eight numbers of 8kb memory. Interface the EPROM and RAM with 8085 processor.

- The total memory capacity is 64Kb. So, let 4 numbers of 8Kb EPROM and 4 numbers of 8Kb RAM.
- Each 8kb memory requires 13 address lines. So the address line A0 - A12 of the processor are connected to 13address pins of all the memory lCs.
- The address lines A13, A14 and A15 are decoded using a 3-to-8 coder to generate eight chip select signals. These eight chip select signals can be used to select one of the eight memories at any one time.
- The memory interfacing is shown in following figure.



Interfacing 4 no. 8Kb EPROM and 4 no. 8Kb RAM with 8085

- The address allocation for Interfacing 4 no. 8Kb EPROM and 4 no. 8Kb RAM with 8085 is



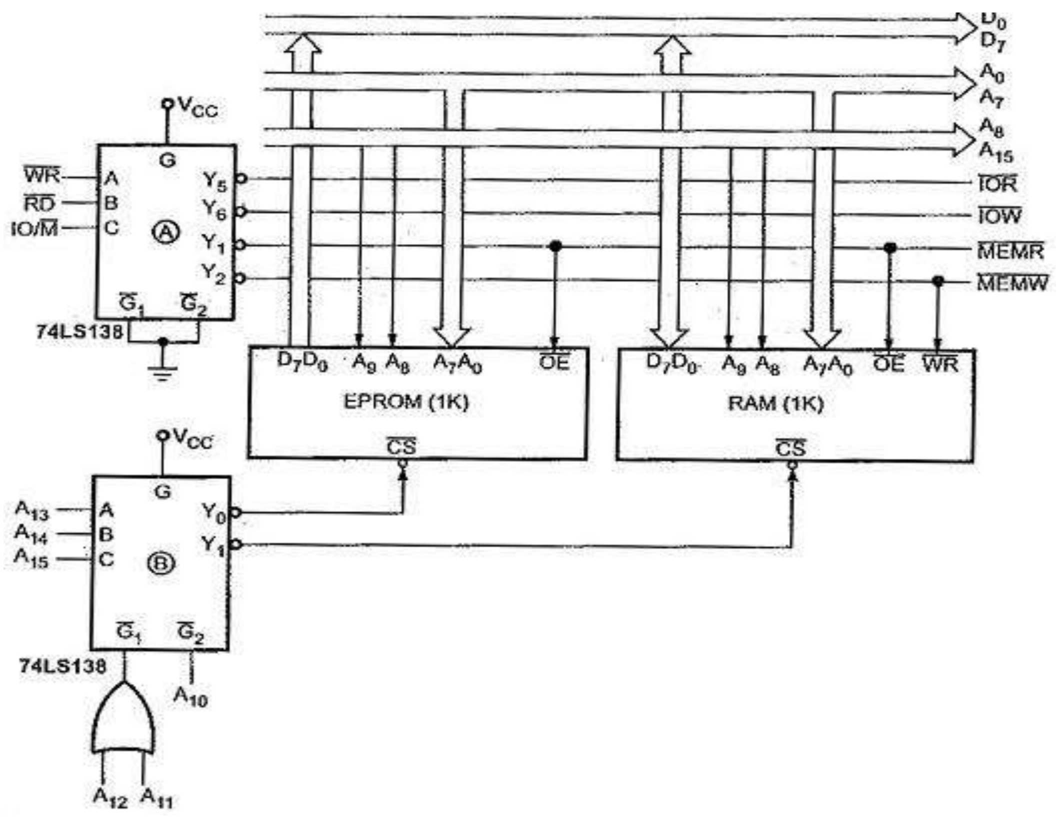
Memory IC chip	Binary address																Hexa address
	Decoder input			Input to memory address pins													
	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
EPROM I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0001
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0002
	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFF
EPROM II	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000
	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	2001
	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	2002
	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFF
EPROM III	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4001
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	4002
	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	5FFF
RAM I	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	6000
	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	6001
	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	6002
	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7FFF
RAM II	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000
	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	8001
	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	8002
	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	9FFF
RAM III	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	A000
	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	A001
	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	A002
	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	BFFF
RAM IV	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C000
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	C001
	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	C002
	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	DFFF
RAM V	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	E000
	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	E001
	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	E002
	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FFFF

4.4 ADDRESS DECODING TECHNIQUES:

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

Absolute decoding:

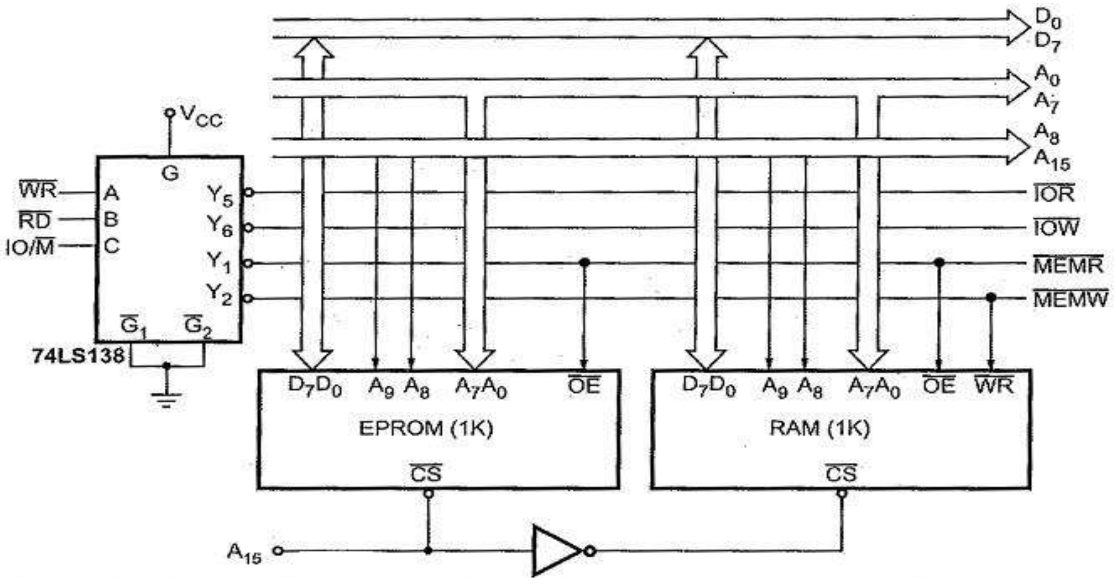
- In absolute decoding technique, all the higher address lines are decoded to select the memory chip, and the memory chip is selected only for the specified logic levels on these high-order address lines; no other logic levels can select the chip.
- This figure shows the Memory Interfacing in 8085 with absolute decoding. This addressing technique is normally used in large memory system



Absolute decoding technique diagram

Linear decoding:

- In small systems, hardware 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.
- This figure shows the addressing of RAM with linear decoding technique.
- This technique is also called **partial decoding**. It reduces the cost of decoding circuit, but it has a drawback of multiple addresses (shadow addresses).



Linear decoding technique diagram

- It shows the addressing of RAM with linear decoding technique. A_{15} address line, is directly connected to the chip select signal of EPROM and after inversion it is connected to the chip select signal of the RAM.
- Therefore, when the status of A_{15} line is 'zero', EPROM gets selected and when the status of A_{15} line is 'one' RAM gets selected.
- The status of the other address lines is not considered, since those address lines are not used for generation of chip select signals.

4.5 Programmable Peripheral Interface: 8255

- 8255 is a programmable I/O device that acts as interface between peripheral devices and the microprocessor for parallel data transfer.
- 8255 PPI (programmable peripheral interface) is programmed in a way so as to have transfer of data in different conditions according to the need of the system.
- In 8255, 24 pins are assigned to the I/O ports. Basically it has three, 8-bit ports that are used for simple or interrupt I/O operations.

- The three ports are **Port A, Port B and Port C** and as each port has 8 lines, but the 8 bits of port C is divided into 2 groups of 4-bit each.
- These are given as port C lower i.e., $PC_3 – PC_0$ and port C upper i.e., $PC_7 – PC_4$.
- And are arranged in group of 12 pins each thus designated as Group A and Group B. The two modes in which 8255 can be programmed are as follows:

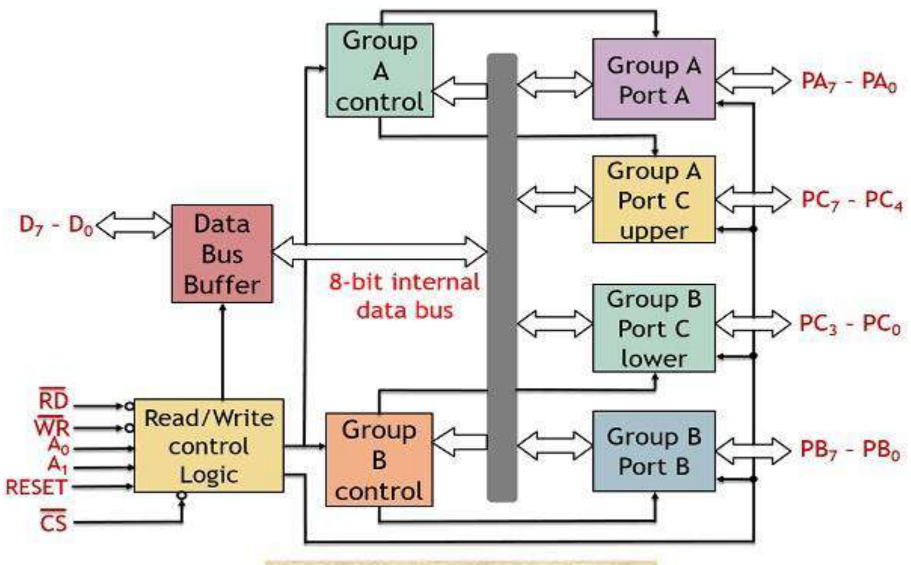
1. Bit set/reset mode
2. I/O mode

- The bits of port C gets set or reset in the BSR mode. The other mode of 8255 i.e., I/O mode is further classified into.

  Mode 0: Simple input/output
  Mode 1: Input output with handshaking
  Mode 2: Bidirectional I/O handshaking

- **Mode 1 and Mode 2** both are same but the only difference is mode 1 does not support bidirectional handshaking.
- This means if 8255 is programmed to **mode 1** input, then it will particularly be connected to an input device and performs the input handshaking with the processor.
- But if it is programmed to **mode 2** then due to bidirectional nature, the PPI will perform both input and output operation with the processor according to the command received.

Architecture of 8255 PPI:



Architecture of 8255 PPI

The figure above represents the architectural representation of 8255 PPI:

Let us understand the operation performed by each unit separately.

Data bus buffer:

- It is used to connect the internal bus of 8255 with the system bus so as to establish proper interfacing between the two.
- The data bus buffer allows the read/write operation to be performed from/to the CPU.
- The buffer allows the passing of data from ports or control register to CPU in case of write operation and from CPU to ports or status register in case of read operation.

Read/ Write control logic:

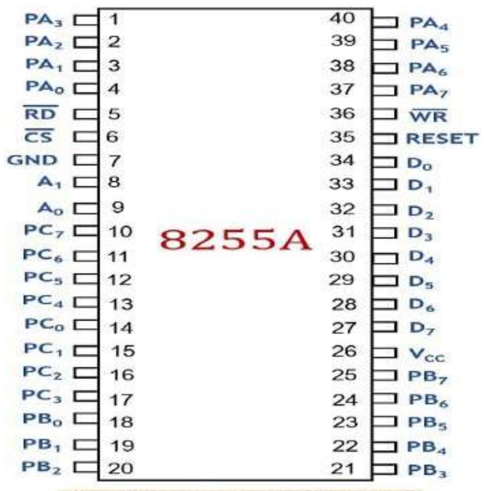
- This unit manages the internal operations of the system. This unit holds the ability to control the transfer of data and control or status words both internally and externally.
- Whenever there exists a need for data fetch then it accepts the address provided by the processor through the bus and immediately generates command to the 2 control groups for the particular operation.

Group A and Group B control:

- These two groups are handled by the CPU and functions according to the command generated by the CPU.
- The CPU sends control words to the group A and group B control and they in turn sends the appropriate command to their respective port.
- **Group A** the access of the **port A** and higher order bits of **port C**. While **group B** controls port B with the lower order bits of **port C.**

Pin Diagram of 8255 PPI

The figure below represents the 40 pin configuration of 8255 programmable peripheral interface:



Pin Diagram of 8255 PPI

<u>CS</u>:

- It stands for chip select. A low signal at this pin shows the enabling of communication between the 8255 and the processor.
- More specifically we can say that the data transfer operation gets enabled by an active low signal at this pin.

<u>RD</u>:

- It is the signal used for read operation.
- A low signal at this pin shows that CPU is performing read operation at the ports or status word.
- Or we can say that 8255 is providing data or information to the CPU through data buffer.

<u>WR</u>:

- It shows write operation. A low signal at this pin allows the CPU to perform write operation over the ports or control register of 8255 using the data bus buffer.

<u>A_0 and A_1</u>:

- These are basically used to select the desired port among all the ports of the 8255 and it do so by forming conjunction with RD and WR.
- It forms connection with the LSB of the address bus.

The table below shows the operation of the control signals:

A_1	**A_0**	**RD'**	**WR'**	**CS'**	**Input/Output Operation**
0	0	0	1	0	Port A - Data Bus
0	1	0	1	0	Port B - Data Bus
1	0	0	1	0	Port C - Data Bus
0	0	1	0	0	Data Bus - Port A
0	1	1	0	0	Data Bus - Port B

A_1	A_0	RD'	WR'	CS'	Input/Output Operation
1	0	1	0	0	Data Bus - Port C
1	1	1	0	0	Data Bus - Control register

Reset:

- It is an active high signal that shows the resetting of the PPI.
- A high signal at this pin clears the control registers and the ports are set in the input mode.
- Initializing the ports to input mode is done to prevent circuit breakdown.
- As in case of reset condition, if the ports are initialized to output mode then there exist chances of destruction of 8255 along with the processor.

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.

PC7-PC4:

- Upper nibble of port C lines. They may act as either output latches or input buffers lines. This port also can be used for generation of handshake lines in mode 1 or mode 2.

PC3-PC0:
These are the lower port C lines, other details are the same as PC7-PC4 lines.

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.

D0-D7: These are the data bus lines those carry data or control word to/from the microprocessor.

VCC:
It is a supply voltage. The 8255 requires +5V supply to operate.
GND:
It is the ground reference. If there is excessive power supply then it passed to ground.

MODES OF OPERATION:

As we have already discussed that 8255 has two modes of operation. These are as follows:

1. Bit set/reset mode
2. I/O mode

Bit Set-Reset mode:

When port C is utilized for control or status operation, then by sending an OUT instruction, each individual bit of port C can be set or reset.

I/O mode:

As we know that I/ O mode is sub-classified into 3 modes. So, let us now discuss the 3 modes here.

Mode 0: Input/output mode:

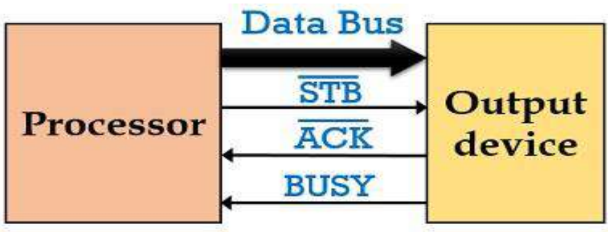
This mode is the simple input output mode of 8255 which allows the programming of each port as either input or output port. The input/output feature of mode 0 includes:

- It does not support handshaking or interrupt capability.
- The input ports are buffered while outputs are latched.

Mode 1: Input/output with handshaking:

Mode 1 of 8255 supports handshaking with the ports programmed as either input or output mode. We know that it is not necessary that all the time the data is transferred between two devices operating at same speed. So, handshaking signals are used to synchronize the data transfer between two devices that operates at different speeds.

The figure below shows the data transferring between CPU and an output device having different operating speeds:



Data Transfer using handshaking signals

- Here STB signal is used to inform the output device that data is available on the data bus by the processor.
- Here port A and port B can be separately configured as either input or output port.
- Both the port utilizes 3-3 lines of port C for handshaking signals. The rest two lines operates as input/output port.
- It supports interrupt logic.
- The data at the input or output ports are latched.

Mode 2: Bidirectional I/O port with handshaking:

- In this mode, the ports can be utilized for the bidirectional flow of information by handshaking signals.
- The pins of group A can be programmed to acts as bidirectional data bus and the port C upper (PC_7 – PC_4) are used by the handshaking signal.
- The rest 4 lower port C bits are utilized for I/O operations.
- As the data bus exhibits bidirectional nature thus when the peripheral device request for a data input only then the processor load the data in the data bus.
- Port B can be programmed in mode 0 and 1. And in mode 1 the lower bits of port C of group B are used for handshaking signals.

4.6 DAC & ADC WITH INTERFACING:

DAC INTERFACING:

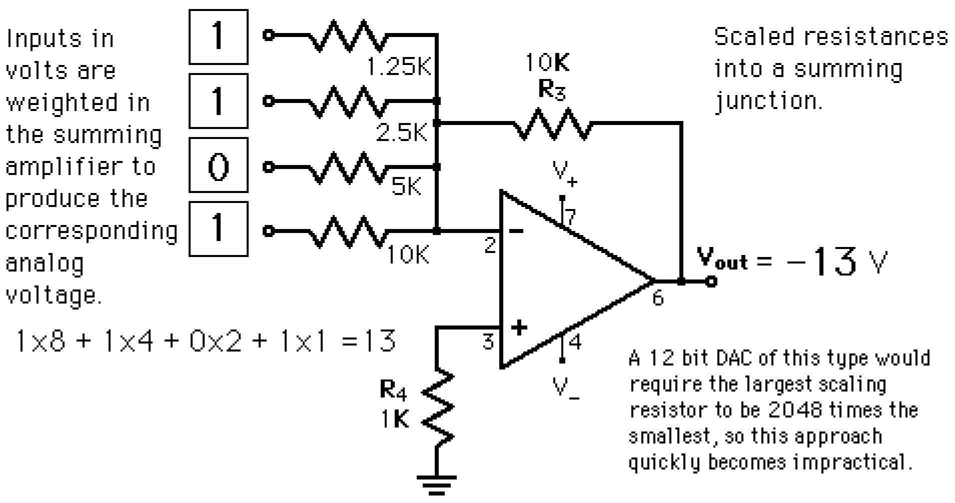
- Digital-to-Analog Conversion or simply DAC, is a device that is used to convert a digital (usually binary) code into an analog signal (current, voltage, or electric charge).
- Digital-to-analog conversion is the primary means by which digital equipment such as computer-based systems are able to translate digital data into real-world signals that are more understandable to or useable by humans, such as music, speech, pictures, video.
- It also allows digital control of machines, equipment, household appliances.
- When data is in binary form, the 0's and 1's may be of several forms such as the TTL form where the logic zero may be a value up to 0.8 volts and the 1 may be a voltage from 2 to 5 volts.
- The data can be converted to clean digital form using gates which are designed to be on or off depending on the value of the incoming signal.
- Data in clean binary digital form can be converted to an analog form by using a summing amplifier.
- Here is a simplified functional diagram of an 8-bit DAC. There are mainly two techniques used for digital to analog conversion

1. Weighted Summing Amplifier

2. R-2R Network

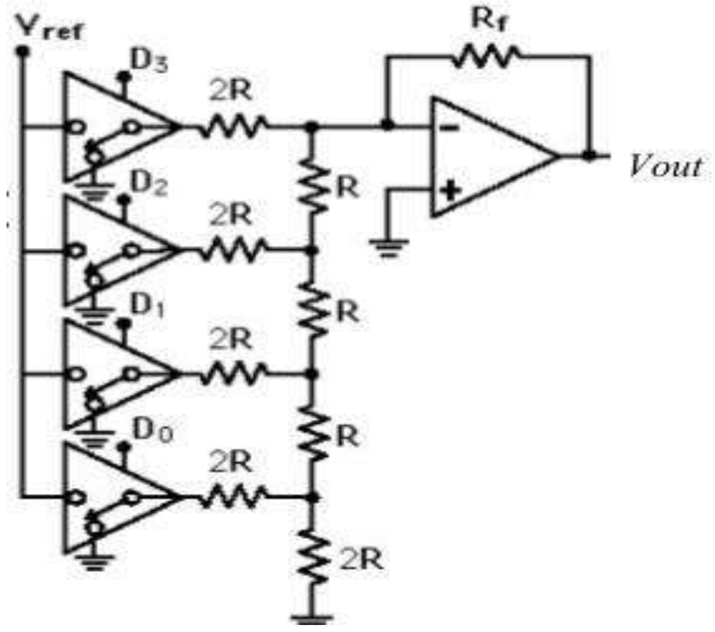
Weighted Sum DAC:

- One way to achieve D/A conversion is to use a summing amplifier.
- This approach is not satisfactory for a large number of bits because it requires too much precision in the summing resistors.
- This problem is overcome in the R-2R network DAC.

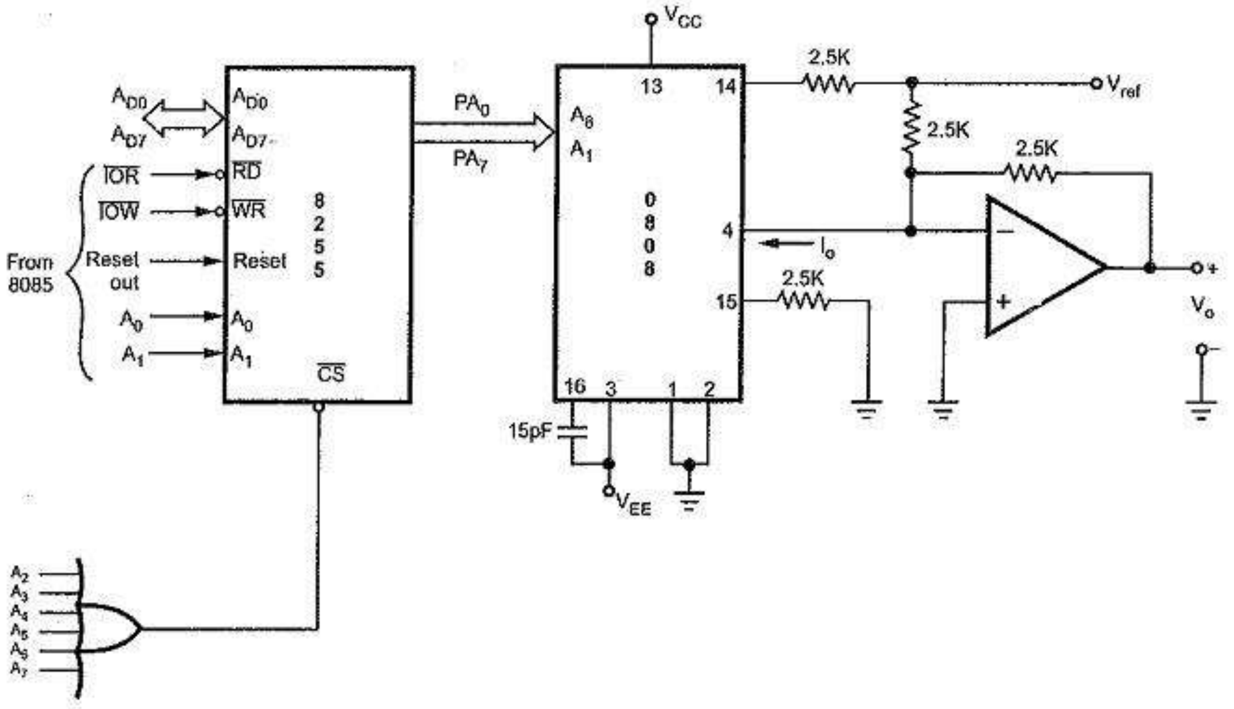

Inputs in volts are weighted in the summing amplifier to produce the corresponding analog voltage.
1
1
0
1
1.25K
2.5K
5K
10K
10K
R_3
Scaled resistances into a summing junction.
V_+
7
2
3
6
4
V_-
V_{out} = −13 V
1x8 + 1x4 + 0x2 + 1x1 =13
R_4
1K
A 12 bit DAC of this type would require the largest scaling resistor to be 2048 times the smallest, so this approach quickly becomes impractical.

R-2R Ladder DAC:

- The summing amplifier with the R-2R ladder of resistances shown produces the output where the D's take the value 0 or 1.
- The digital inputs could be TTL voltages which close the switches on a logical 1 and leave it grounded for a logical 0.
- This is illustrated for 4 bits, but can be extended to any number with just the resistance values R and 2R.



- The interfacing of DAC 0808 with microprocessor 8085 is shown below. Here, programmable peripheral interface, 8255 is used as parallel port to send the digital data to DAC.



Interfacing of 0808 with microprocessor

Interfacing Digital-To-Analog converter to 8085 using 8255"

- Figure below shows the interfacing of DAC 0808 with microprocessor 8085. Here, programmable peripheral interface, 8255 is used as parallel port to send the digital data to DAC.
- **I/O Map for 8255**

PORT/REGISTER	ADDRESS
Port A	00
Port B	01
Port C	02
Control Register	03

<u>Program:</u>

MVI A, 80H; Initialization -control word for 8255 to configure all ports as output Ports

OUT 03

MVI A, DATA; Load 8-bit data to be sent at the input of 0808 DAC

OUT 00; send data on port A.

A Circuit Description of DAC module:

- When chip select of DAC is enabled then DAC will convert digital input value given through portliness PB0-PB7 to analog value.
- The analog output from DAC is a current quantity. This current is converted to voltage using OPAMP based current-to-voltage converter.
- The voltage outputs (+/- 8V for bipolar 0 to 8V for unipolar mode) of OPAMP are connected to CRO to see the wave form. Port A & Port B are connected to channel 1 and channel 2 respectively.
- A reference voltage of 8V is generated using 723 and is given to Verify points of the DAC 0800. The standard output voltage will be 7.98V when FF is outputted and will be 0V when 00 is outputted.
- The Output of DAC-0800 is fed to the operational amplifier to get the final output as X OUT and Y OUT.

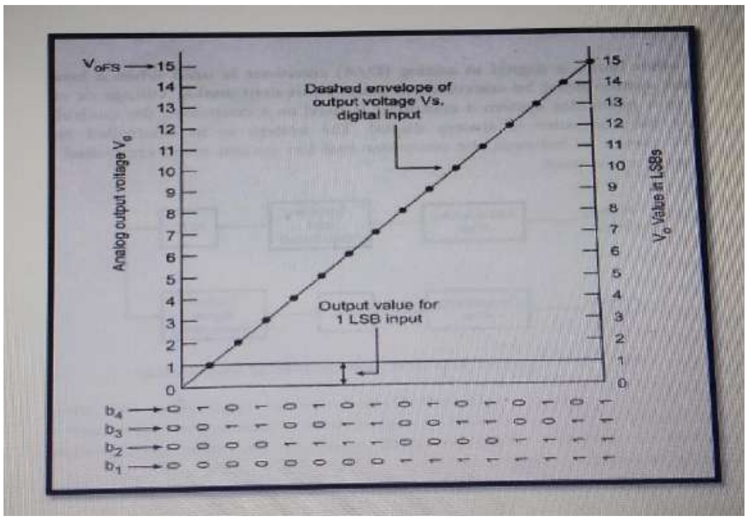


Figure shows analog output voltage v0 is plotted against all 16 possible digital input words.

Performance Parameters of DAC:

The performance parameters of DAC are:

1. <u>**Resolution**</u>:

- Resolution is defined in two ways. o Resolution is the number of different analog output values that can be provided by a DAC.
- For an n-bit DAC **Resolution = 2^n ……… (1)**

or

- Resolution is also defined as the ratio of a change in output voltage resulting from a change of 1 LSB at the digital inputs.
- For an n-bit DAC it can be given as: **Resolution= V_o F_s /2^n -1 ………(2)**
- Where, **V_o F_s** = Full scale output voltage from equation (1), we can say that, the resolution can be determined by the number of bits in the input binary word.
- For an 8-bit resolution can be given as resolution = 2^n = 2^8 = 256

- If the full scale output voltage is 10.2 V then by second definition the resolution for an 8-bit can be given as Resolution= = **V_o F_s /2^n -1 =10.2/2^8-1 =10.2/255 = 40 mV/LSB**
- Therefore, we can say that an input change of 1 LSB causes the output to change by 40 mv

2. <u>Accuracy</u>:

- lt is a comparison of actual output voltage with expected output. It is expressed in percentage. Ideally, the accuracy of DAC should be, at worst, ±1/2, of its LSB.
- If the full scale output voltage is 10.2 V then for an 8-bit DAC accuracy can be given as

  Accuracy = Vo Fs / (2^n-1)2 = 10.2/255x2 = 20 mV

ADC CONVERTER:

- It is a converter which converts analog quantity into digital quantity.
- There are many types of ADC available such as

1. RAMP type ADC
2. Dual slope ADC
3. Flash type ADC
4. Successive approximation type ADC

- Mostly the successive approximation type ADC are used.

SPECIFICATION OF ADC:

<u>Input voltage range:</u>

- The analog input can be either unipolar or bipolar.
- Unipolar means the voltage have one polarity i.e. (0 to +5V or -5V to 0).
- Bipolar means the voltage have the range from one polarity to other polarity i.e. (+5V to -5V or -10V to +10V)

<u>Output voltage range:</u>

- The resolution of ADC is defined as the smallest change input voltage can be sensed or detected at the output. Which is given by

  $$\text{Resolution} = \frac{\text{range of input voltage}}{\text{range of output voltage}}$$

  Example→if the input voltage range from 0 to +5V and output has 8 bit then the resolution =5/2^8=19.5mv

Conversion time:

It is the time required to convert the analog input into digital output by ADC chip is known as conversion time.

Example of ADC chip:

- ADC 0800 IC
- ADC0804 IC
- ADC 0808 IC
- ADC 0816 IC

ADC INTERFACING:

The Analog to Digital Conversion is a quantizing process. Here the analog signal is represented by equivalent binary states. The A/D converters can be classified into two groups based on their conversion techniques.

- In the first technique it compares given analog signal with the initially generated equivalent signal. In this technique, it includes successive approximation, counter and flash type converters.
- In another technique it determines the changing of analog signals into time or frequency. This process includes integrator-converters and voltage-to-frequency converters.
- The first process is faster but less accurate, the second one is more accurate. As the first process uses flash type, so it is expensive and difficult to design for high accuracy.

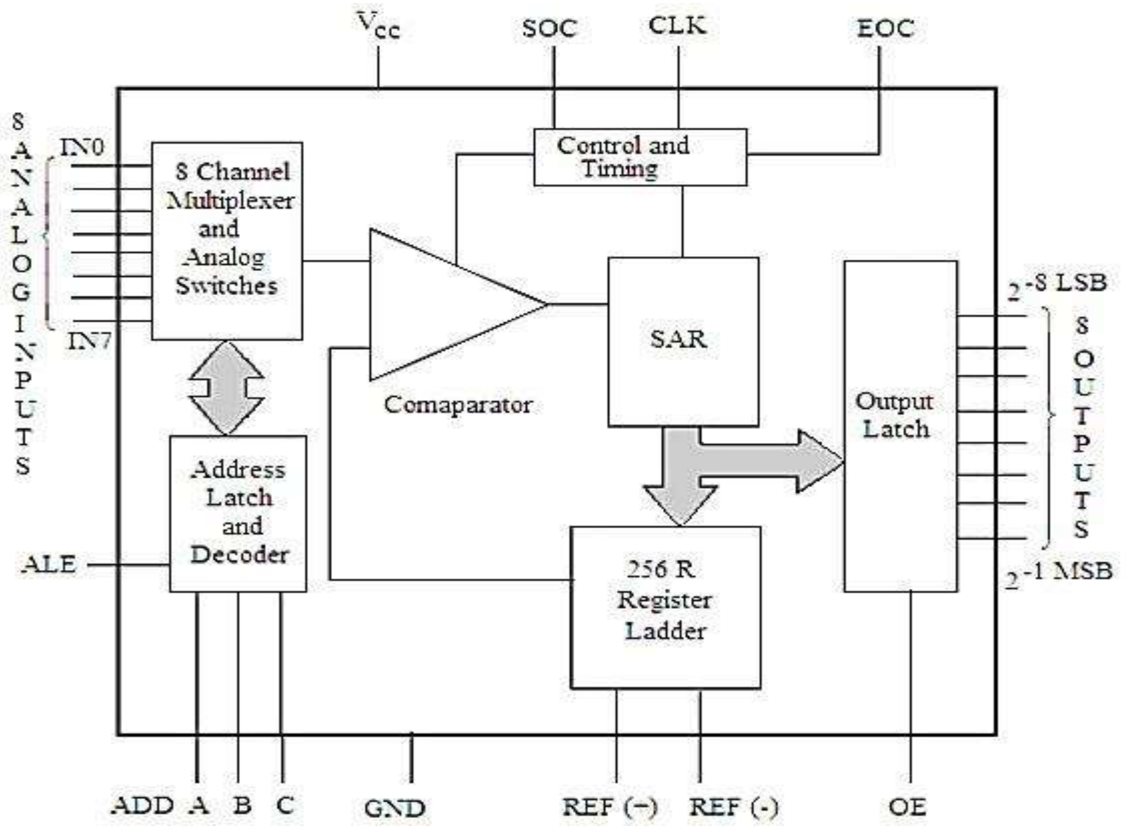
The ADC 0808/0809 Chip

- The ADC 0808/0809 is an 8-bit analog to digital converter.
- It has 8 channel multiplexer to interface with the microprocessor.
- This chip is popular and widely used ADC.
- 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.

Let us see some good features of ADC 0808/0809

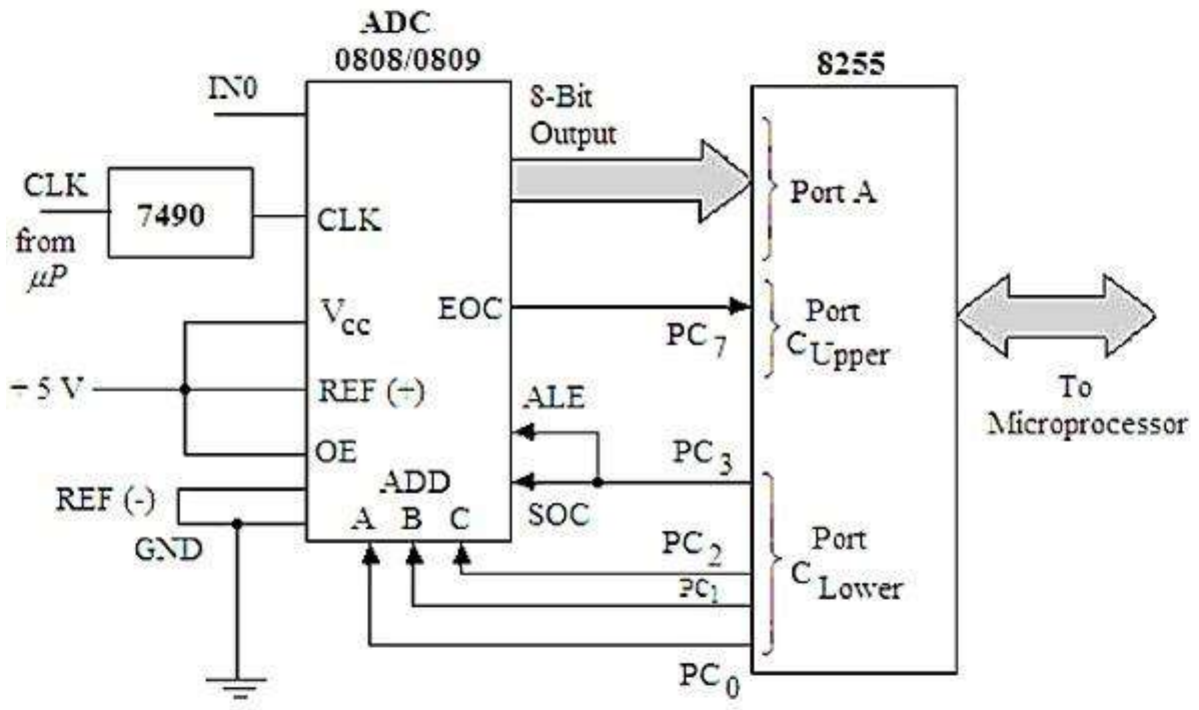
- ➢ The conversion speed is much higher
- ➢ The accuracy is also high
- ➢ It has minimal temperature dependence
- ➢ Excellent long term accuracy and repeatability
- Less power consumption

The functional block diagram of this chip is like this



Interfacing ADC with 8085 Microprocessor:

To interface the ADC with 8085, we need 8255 Programmable Peripheral Interface chip with it. Let us see the circuit diagram of connecting 8085, 8255 and the ADC converter.



- The Port A 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_{2-0} 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.

  Now let us see a program to generate digital signal from analog data. We are using IN0 as input pin, so the pin selection value will be 00H.

MVI A, 98H; Set Port A and Cupper as input, C_{Lower} as output

OUT 03H; Write control word 8255-I to control Word register

XRA A; Clear the accumulator

OUT 02H; send the content of accumulator to Port C_{lower} to select

IN0

MVI A, 08H; Load the accumulator with 08H

OUT 02H; ALE and SOC will be 0

XRA A; Clear the accumulator

OUT 02H; ALE and SOC will be low.

READ: IN 02H; Read from EOC (PC7)

RAL: Rotate left to check C7 is 1.

JNC READ; if C7 is not 1, go to READ

IN 00H: Read digital output of ADC

STA 8000H: Save result at 8000H

HLT: Stop the program

4.7 INTERFACING SEVEN SEGMENT DISPLAYS

SEVEN SEGMENT DISPLAYS:

- A seven-segment display is a form of electronic display device for displaying decimal numerals that is an alternative to the more complex dot matrix displays.
- Seven-segment displays are widely used in digital clocks, electronic meters, basic calculators, and other electronic devices that display numerical information.
- The binary information can be displayed in the form of decimal using this seven segment display. Its wide range of applications is in microwave ovens, calculators, washing machines, radios, digital clocks etc.

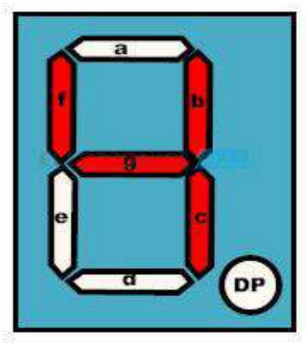
- The seven segment displays are made up of either LEDs (Light emitting diode) or LCDs (Liquid crystal display). LED or light emitting diode is P-N junction diode which emits the energy in the form of light, differing from normal P-N junction diode which emits in the form of heat.

- Liquid crystal displays (LCD) use the properties of liquid crystal for displaying. LCD will not emit the light directly. These LED's or LCD are used to display the required numeral or alphabet. Single seven segment or number of segments arranged in an order meets our requirements.

Working of Seven Segment Display:

- Seven segment display works, by glowing the required respective LEDS in the numeral. The display is controlled using pins that are left freely. Forward biasing of these pins in a sequence will display the particular numeral or alphabet. Depending on the type of seven segment the segment pins are applied with logic high or logic zero and in the similar way to the common pins also.

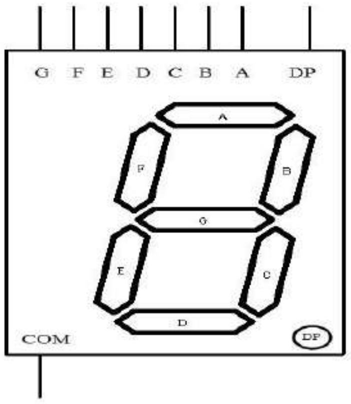
- For example to display numeral '1' segments b and c are to be switched on and the remaining segments are required to be switched off. In order to display two digits two seven segments are used.



- Depending on either the common pin is anode or cathode, seven segments are divided into following types.

INTERFACING SEVEN SEGMENT DISPLAY:

- Seven Segment Display Interfacing are generally used as numerical indicators and consists of a number of LEDs arranged in seven segments as shown in the Figure

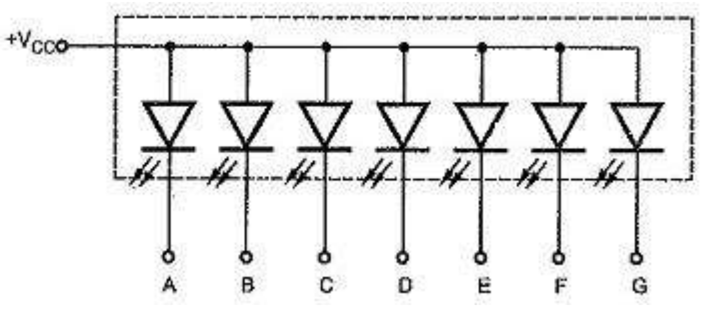


- Any number between 0 and 9 can be indicated by lighting the appropriate segments. The 7-segment displays are of two types:

  1. Common anode type
  2. Common cathode type.

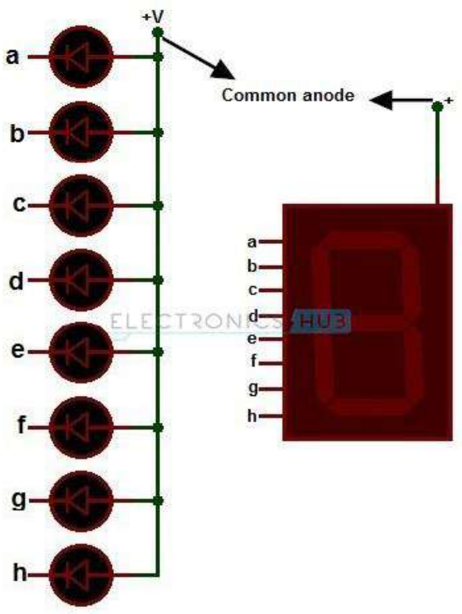
  Common Anode type:

- In common anode, all anodes of LEDs are connected together as shown in Fig.

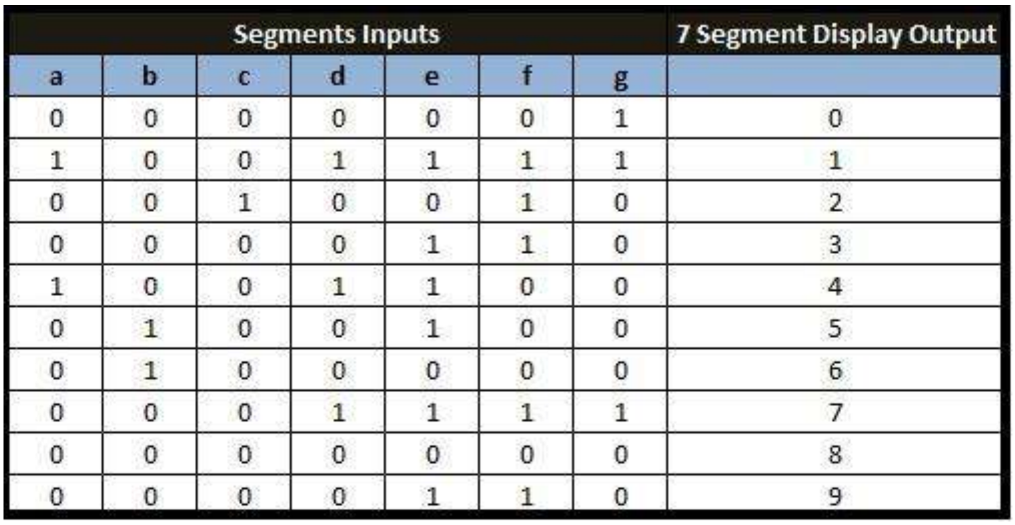


Common anode type

or



- In common anode type, all the anodes of 8 LED's are connected to the common terminal and cathodes are left free. Thus, in order to glow the LED, these cathodes have to be connected to the logic '0' and anode to the logic '1'.
- Below truth table gives the information required for driving the common anode seven segments.

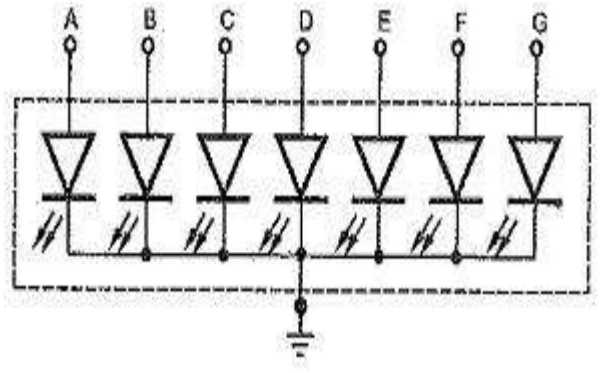


Segments Inputs							7 Segment Display Output
a	b	c	d	e	f	g	
0	0	0	0	0	0	1	0
1	0	0	1	1	1	1	1
0	0	1	0	0	1	0	2
0	0	0	0	1	1	0	3
1	0	0	1	1	0	0	4
0	1	0	0	1	0	0	5
0	1	0	0	0	0	0	6
0	0	0	1	1	1	1	7
0	0	0	0	0	0	0	8
0	0	0	0	1	1	0	9

- In order to display zero on this segment one should enable logic high on a, b, c, d, e and f segments and logic low on segment 'g'. Thus, the above table provides data on seven segments for displaying numerals from 0-9.

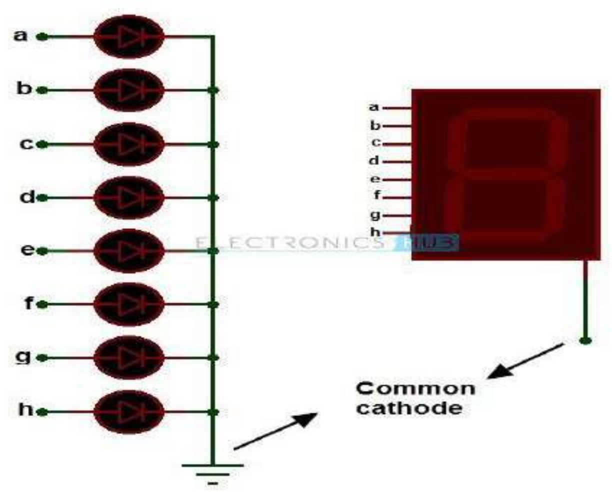
Common cathode type:

- As the name indicates cathode is the common pin for this type of seven segments and remaining 8 pins are left free. Here, logic low is applied to the common pin and logic high to the remaining pins.
- in common cathode, all cathodes are connected together, as shown in Fig

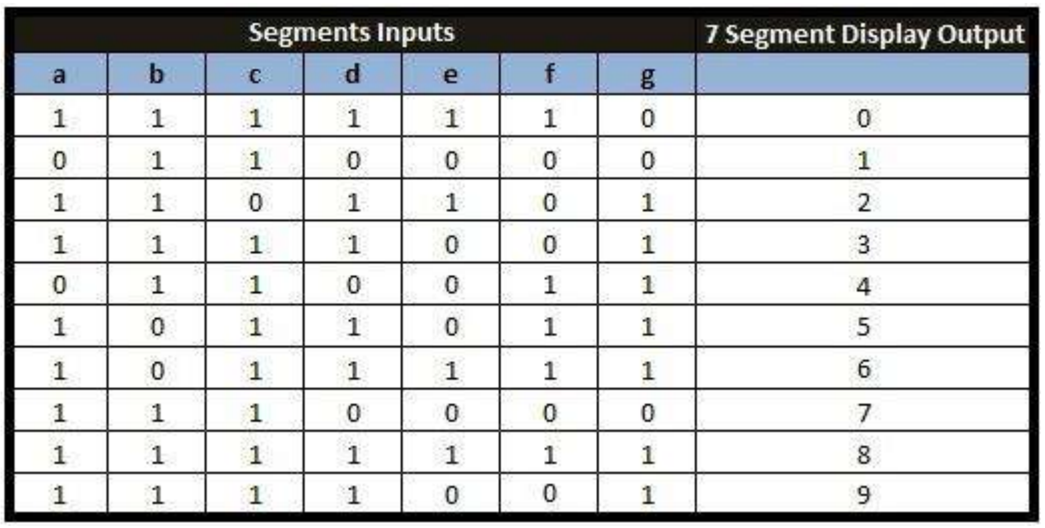


Common cathode type

Or



The truth table of seven segment display is shown below.

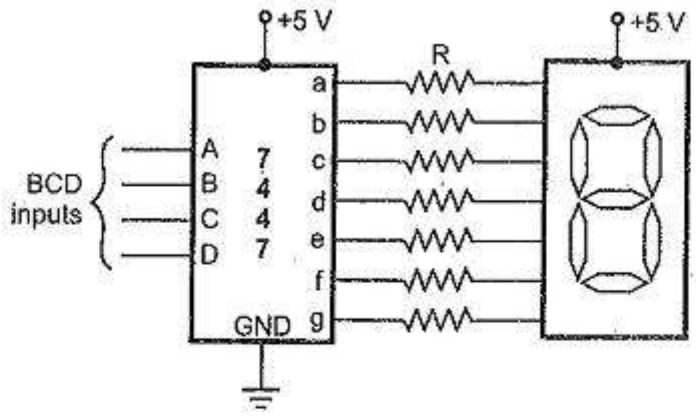


Segments Inputs							7 Segment Display Output
a	b	c	d	e	f	g	
1	1	1	1	1	1	0	0
0	1	1	0	0	0	0	1
1	1	0	1	1	0	1	2
1	1	1	1	0	0	1	3
0	1	1	0	0	1	1	4
1	0	1	1	0	1	1	5
1	0	1	1	1	1	1	6
1	1	1	0	0	0	0	7
1	1	1	1	1	1	1	8
1	1	1	1	0	0	1	9

- Above truth table shows the data to be applied to the seven segments to display the digits. In order to display digit'0' on seven segment , segments a , b , c , d , e and f are applied with logic high and segment g is applied with logic low.

Driving a Seven Segment Display:

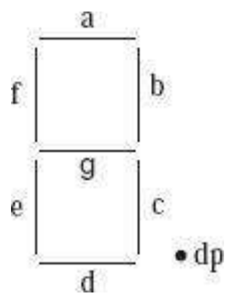
- It shows a circuit to drive a single, Seven Segment Display Interfacing, common anode LED display.
- For common anode, when anode is connected to positive supply, a low voltage is applied to a cathode to turn it on.
- Here, BCD to seven segment decoder, IC 7447 is used to apply low voltages at cathodes according to BCD input applied to 7447.
- To limit the current through LED segments resistors are connected in series with the segments.
- This circuit connection is referred to as a static display because current is being passed through the display at all times.

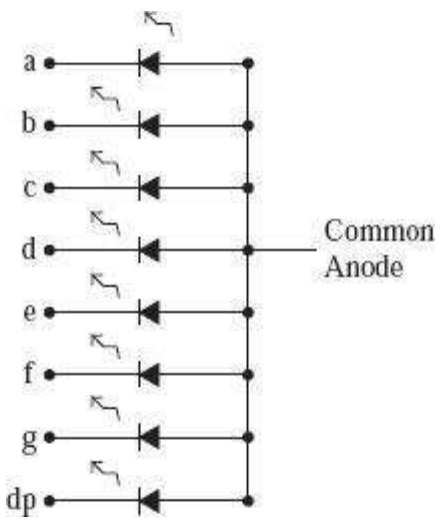


Circuit for driving single seven segment LED display

INTERFACING:

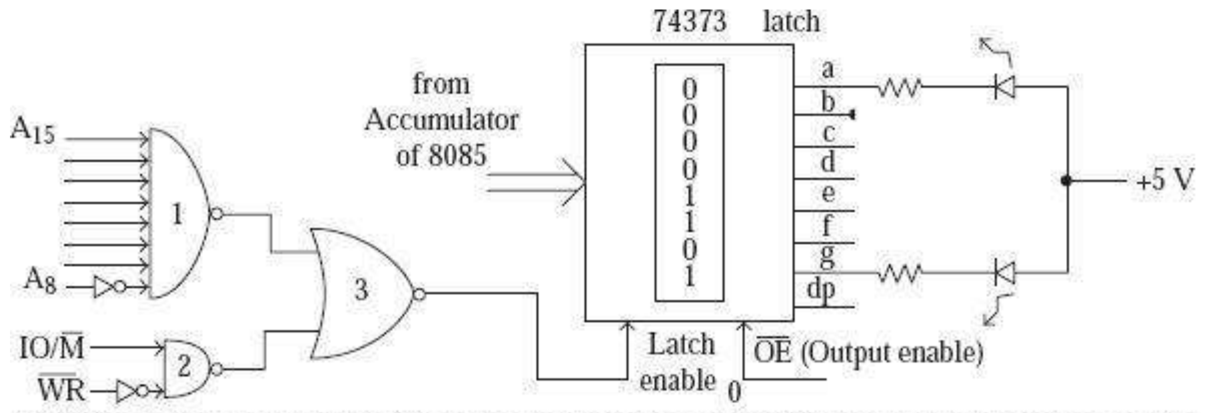
- An output device which is very common is, especially in the kit of 8085 microprocessor and it is the Light Emitting Diode consisting of seven segments.
- Moreover, we have eight segments in a LED display consisting of 7 segments which, consist of character 8 and having a decimal point just next to it.
- We denote the segments as ‘a, b, c, d, e, f, g, and dp’ where dp signifies ‘.’ which is the decimal point.
- Moreover, these are LEDs or together a series of Light Emitting Diodes. We have shown the internal circuit comprising of a display of seven segment





- We have discussed the common anode-type which is 7 segmented Light Emitting Diode.
- In the LED which is common anode and is 7-segmented, here we connect all the eight LED anodes together and the eight external pin is brought to display.
- And this pin gets connected to a DC supply of +5 Volt.
- The cathode ends of the eight segments are brought out on the pins of the display.

The use of 74373 latch for interfacing a 7-segment display is shown in the following Fig.



Note: To keep the figure simple, only two of the eight segment connections are shown. The other six segment connections are similar.

- In the 74373 latch is used as an I/O mapped I/O port with the port address as FEH.
- This could be easily verified from the chip select circuit used in the figure.
- The following instructions are to be executed to display character '3' on the 7-segment display.
- The corresponding program to send 0DH to the port FEH will be -

MVI A, 0DH

OUT FEH

- Using MVI instruction we are initializing Accumulator (A) with Byte 0DH i.e. 0000 1101.
- Then it will be sent to the port FEH by the instruction OUT.

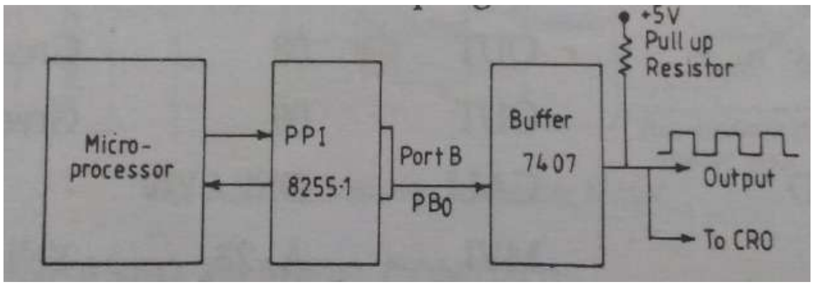
4.8 GENERATE SQUARE WAVES ON ALL LINES OF 8255:

- A square wave or pulse can easily be generated by microprocessor.
- The microprocessors sends high and then low signals to generate square wave or pulse.

- A pulse or square wave can be generated using I/O port or SOD line or timer/counter (Intel 8253).

  To generate square wave or pulse using I/O port:

- To generate square wave connections are made as shown in figure below.



To generate square wave using microprocessor

- The pin PB_0 of the port B is used for taking output. This is connected to a buffer 7407. The final output is taken from the buffer terminal.
- The 8255 has been designed as general purpose programmable I/O devices, compatible with Intel microprocessor.
- It contain three 8-bit port which can be configured by software means to prove any one of 3 programmable data transfer modes available with 8255.
- The control word used in the program is 98H to make port B an output port.

PROGRAM:

MEMORY ADDRESS	**MACHINE CODES**	**LABLES**	**MNEMONICS**	**OPERANDS**	**COMMENTS**
2400	3E, 98		MVI	A,98 H	Get control word.
2402	D3, 0B		OUT	08	Initialize ports.
2404	3E, 00	LOOP	MVI	A,00	Move '00' into accumulator
2406	D3,09		OUT	09	Make PB_0 LOW
2408	CD, 00, 25		CALL	DELAY 1	Call the DELAY 1 subroutine.
240B	3E, 01		MVI	A,01	Move '01' into accumulator
240D	D3, 09		OUT	09	Make PB_0 HIGH
240F	CD, 09, 25		CALL	DELAY 2	Call the DELAY 2 subroutine.
2412	C3, 04, 24		JMP	LOOP	Jump to LOOP.
SUBROUTINES DELAY 1					
2500	06, 02		MVI	B,02	Get count for delay.
2502	05	GO	DCR	B	Decrement register B by 1.

2503	C3, 02, 25		JNZ	GO	Is B=0? No, then jump to GO.
2506	C9		RET		
DELAY 2					
2509	0E, 02		MVI	C,02	Get count for delay
250B	0D	BACK	DCR	C	Decrement register C by 1.
250C	C2, 0B, 25		JNZ	BACK	Is C=0? No, then go to BACK.
250F	C9		RET		

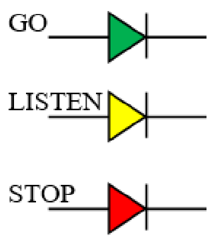
4.9 DESIGN INTERFACE A TRAFFIC LIGHT CONTROL SYSTEM USING 8255:

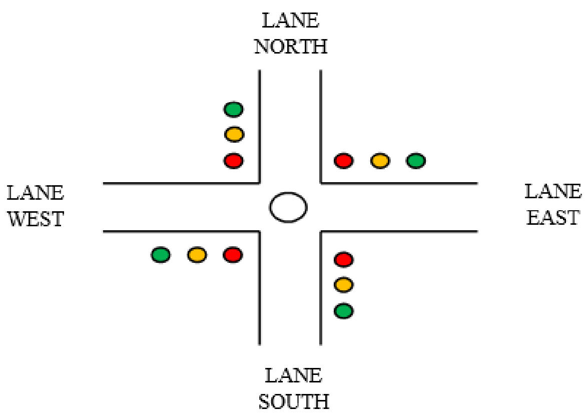
TRAFFIC LIGHT CONTROL

Traffic lights, which may also be known as stoplights, traffic lamps, traffic signals, signal lights, robots or semaphore, are signaling devices positioned at road intersections, pedestrian crossings and other locations to control competing flows of traffic.

ABOUT THE COLORS OF TRAFFIC LIGHT CONTROL

- Traffic lights alternate the right of way of road users by displaying lights of a standard color (red, yellow/amber, and green), using a universal color code (and a precise sequence to enable comprehension by those who are color blind).
- Illumination of the red signal prohibits any traffic from proceeding. Usually, the red light contains some orange in its hue, and the green light contains some blue, for the benefit of people with red-green color blindness, and "green" lights in many areas are in fact blue lenses on a yellow light (which together appear green).

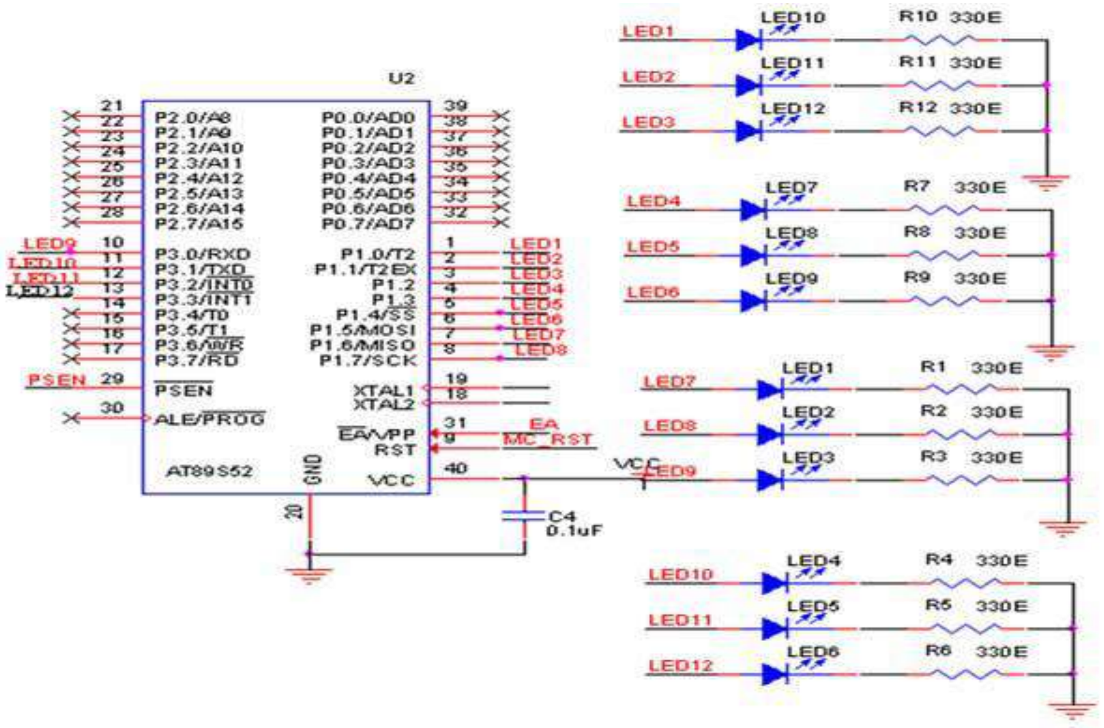




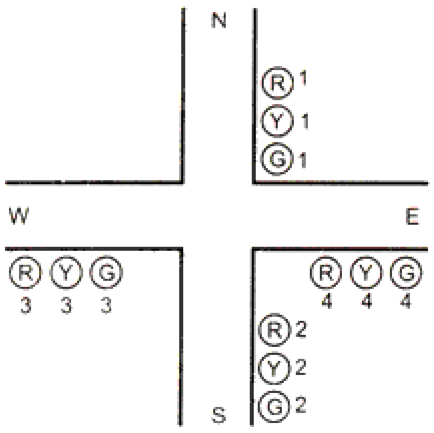
INTERFACING TRAFFIC LIGHT WITH 8085

The Traffic light controller section consists of 12 Nos. point LEDS are arranged by 4Lanes in Traffic light interface card. Each lane has Go (Green), Listen (Yellow) and Stop (Red) LED is being placed.

CIRCUIT DIAGRAM TO INTERFACE TRAFFIC LIGHT WITH 8085



HARDWARE FOR TRAFFIC LIGHT CONTROL



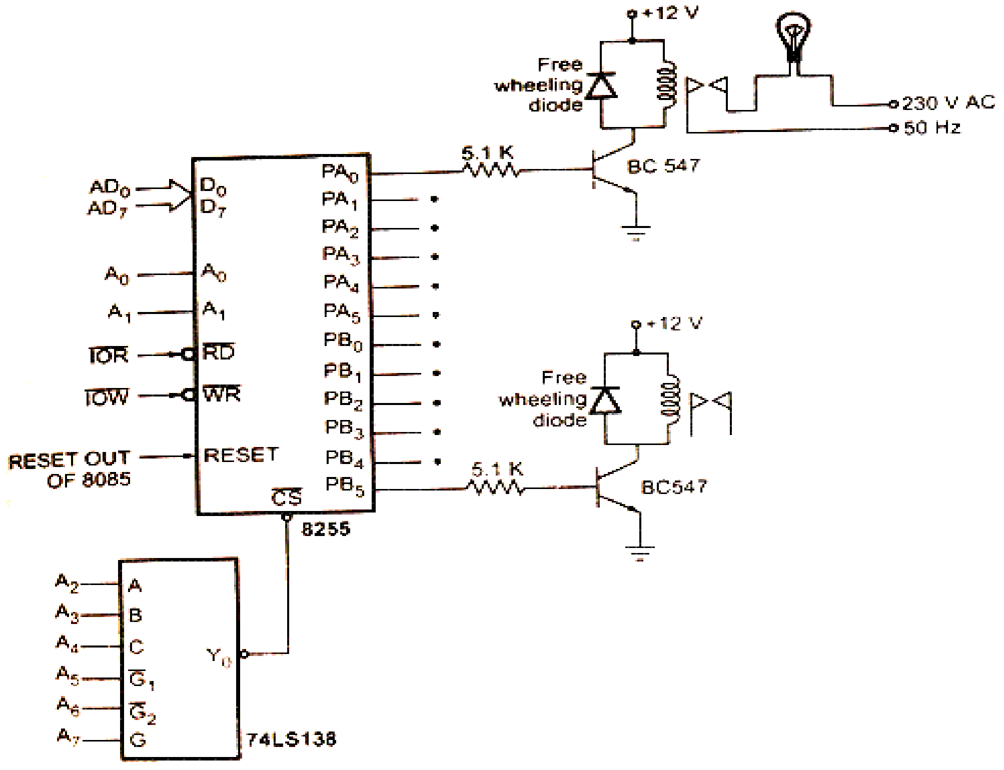
- The interfacing diagram to control 12 electric bulbs. Port A is used to control lights on N-S road and Port B is used to control lights on W-E road. Actual pin connections are listed in Table 1 below.

Pins	Light	Pins	Light
PA_0	R_1	PB_0	R_3
PA_1	Y_1	PB_1	Y_3
PA_2	G_1	PB_2	G_3
PA_3	R_2	PB_3	R_4
PA_4	Y_2	PB_4	Y_4
PA_5	G_2	PB_5	G_4

Table 1

- The electric bulbs are controlled by relays. The 8255 pins are used to control relay on-off action with the help of relay driver circuits. The driver circuit includes 12 transistors to drive 12 relays. Fig. also shows the interfacing of 8255

INTERFACING DIAGRAM:

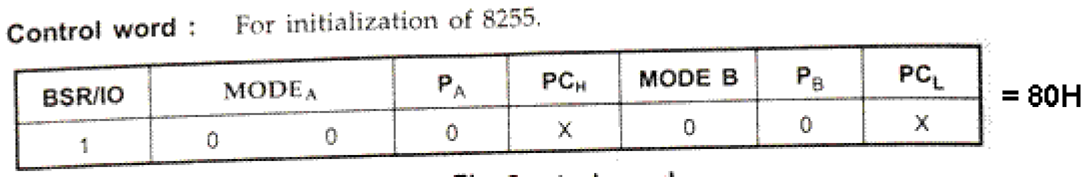


I/O MAP:

Ports / Control Register	Address lines A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0	Address
Port A	1 0 0 0 0 0 0 0	80H
Port B	1 0 0 0 0 0 0 1	81H
Port C	1 0 0 0 0 0 1 0	82H
Control Register	1 0 0 0 0 0 1 1	83H

Table 2

SOFTWARE FOR TRAFFIC LIGHT CONTROL:



Control word : For initialization of 8255.

BSR/IO	MODE A		P_A	PC_H	MODE B	P_B	PC_L
1	0	0	0	X	0	0	X

= 80H

Fig. Control word

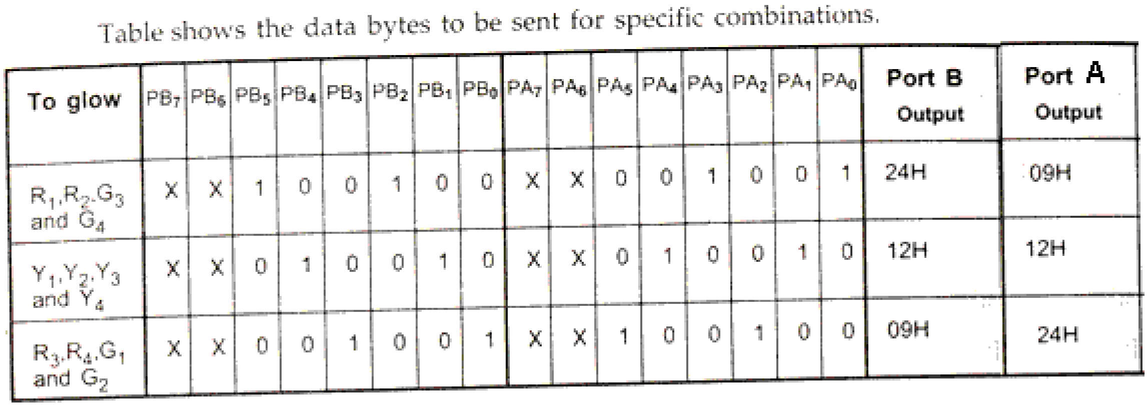


Table shows the data bytes to be sent for specific combinations.

To glow	PB_7	PB_6	PB_5	PB_4	PB_3	PB_2	PB_1	PB_0	PA_7	PA_6	PA_5	PA_4	PA_3	PA_2	PA_1	PA_0	Port B Output	Port A Output
R_1,R_2,G_3 and G_4	X	X	1	0	0	1	0	0	X	X	0	0	1	0	0	1	24H	09H
Y_1,Y_2,Y_3 and Y_4	X	X	0	1	0	0	1	0	X	X	0	1	0	0	1	0	12H	12H
R_3,R_4,G_1 and G_2	X	X	0	0	1	0	0	1	X	X	1	0	0	1	0	0	09H	24H

PROGRAM:

```
MVI A, 80H                 : Initialize 8255, port A and port B
OUT 83H (CR)                 : in output mode
START: MVI A, 09H
OUT 80H (PA)                 : Send data on PA to glow R1 and R2
MVI A, 24H
OUT 81H (PB)                  : Send data on PB to glow G3 and G4
MVI C, 28H                   : Load multiplier count (40ıo) for delay
CALL DELAY                     : Call delay subroutine
MVI A, 12H
OUT (81H) PA                   : Send data on Port A to glow Y1 and Y2
OUT (81H) PB                   : Send data on port B to glow Y3 and Y4
MVI C, 0AH                   : Load multiplier count (10ıo) for delay
CALL: DELAY                    : Call delay subroutine
MVI A, 24H
OUT (80H) PA                   : Send data on port A to glow G1 and G2
MVI A, 09H
OUT (81H) PB                    : Send data on port B to glow R3 and R4
MVI C, 28H                    : Load multiplier count (40ıo) for delay
CALL DELAY                      : Call delay subroutine
MVI A, 12H
OUT PA                       : Send data on port A to glow Y1 and Y2
OUT PB                       : Send data on port B to glow Y3 and Y4
MVI C, 0AH                   : Load multiplier count (10ıo) for delay
CALL DELAY                     : Call delay subroutine
JMP START
```

Delay Subroutine:

```
DELAY: LXI D, Count         : Load count to give 0.5 sec delay
BACK: DCX D                 : Decrement counter
MOV A, D
ORA E                       : Check whether count is 0
JNZ BACK                    : If not zero, repeat
DCR C                       : Check if multiplier zero, otherwise repeat
JNZ DELAY
RET                         : Return to main program
```

4.10 DESIGN INTERFACE FOR STEPPER MOTOR CONTROL USING 8255:

STEPPER MOTOR:

- 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 mainly used for position control.
- ✓ It is used in disk drives, dot matrix printers, plotters and robotics and process control circuits.

Structure:

- Stepper motors have a permanent magnet called rotor (also called the shaft) surrounded by a stator.
- The most common stepper motors have four stator windings that are paired with a center-tap.
- This type of stepper motor is commonly referred to as a four-phase or unipolar stepper motor.
- The center tap allows a change of current direction in each of two coils when a winding is grounded, thereby resulting in a polarity change of the stator.

Interfacing:

- Even a small stepper motor require a current of 400 mA for its operation. But the ports of the microcontroller cannot source this much amount of current.
- If such a motor is directly connected to the microprocessor/microcontroller ports, the motor may draw large current from the ports and damage it.
- So a suitable driver circuit is used with the microprocessor/microcontroller to operate the motor.

Motor Driver Circuit (ULN2003)

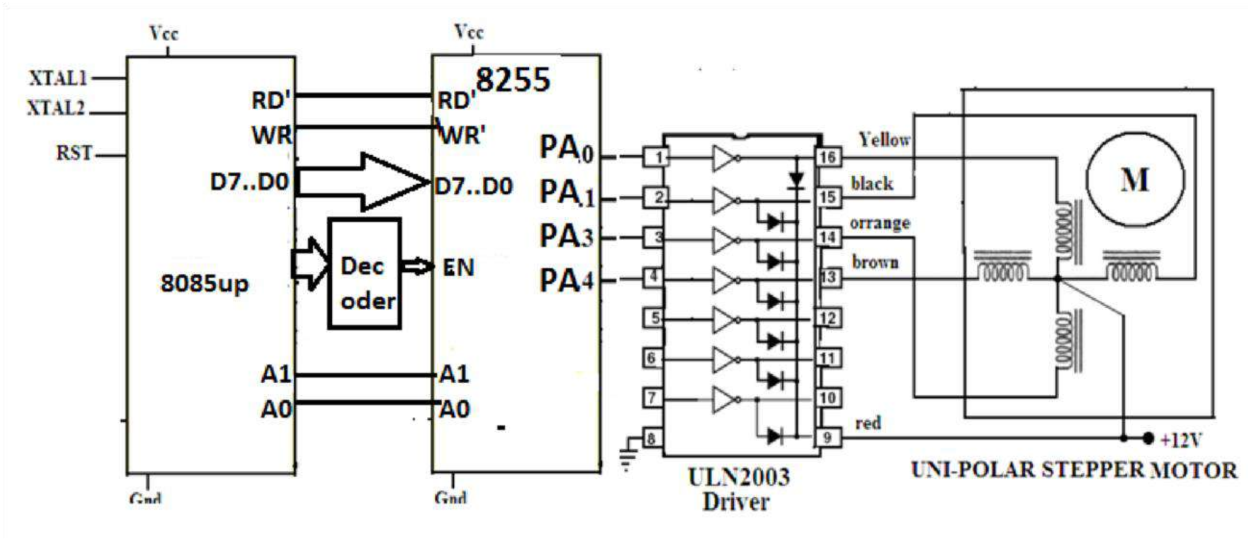
- Stepper motor driver circuits are available readily in the form of ICs.

- ULN2003 is one such driver IC which is a High-Voltage High-Current Darlington transistor array and can give a current of 500mA.
- This current is sufficient to drive a small stepper motor. Internally, it has protection diodes used to protect the motor from damage due to back emf and large eddy currents.
- So, this ULN2003 is used as a driver to interface the stepper motor to the microcontroller.

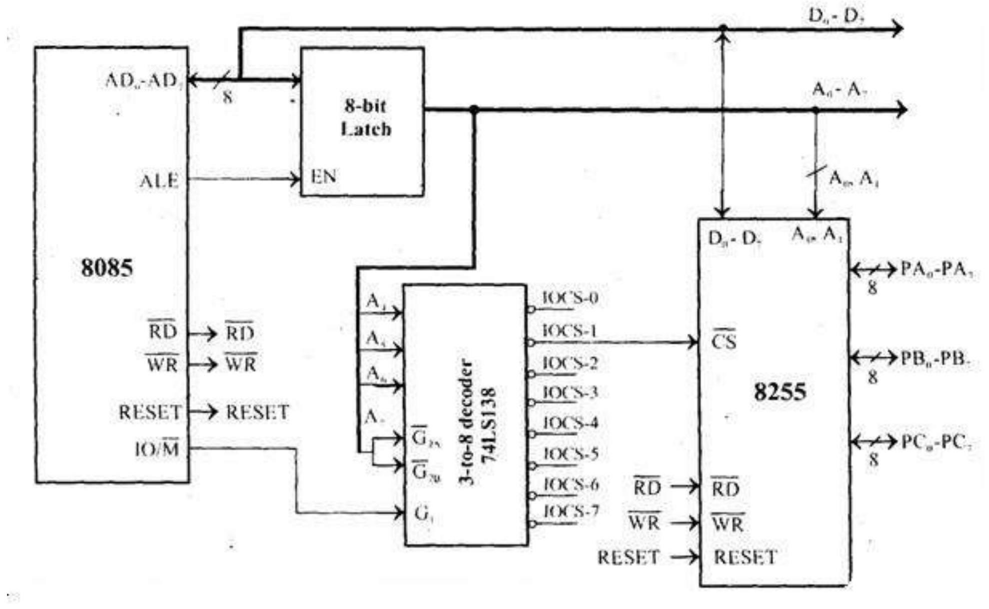
Operation:

- The important parameter of a stepper motor is the **step angle**.
- It is the minimum angle through which the motor rotates in response to each **excitation pulse**.
- In a four phase motor if there are 200 steps in one complete rotation then then the step angle is 360/200 = 1.8^0 .
- So to rotate the stepper motor we have to apply the excitation pulse. For this the controller should send a hexa decimal code through one of its ports.
- **The hex code mainly depends on the construction of the stepper motor**. So, all the stepper motors do not have the same Hex code for their rotation. (Refer the operation manual supplied by the manufacturer.)
- For example, let us consider the hex code for a stepper motor to rotate in clockwise direction is 77H, BBH, DDH and EEH. This hex code will be applied to the input terminals of the driver through the assembly language program. To rotate the stepper motor in anti-clockwise direction the same code is applied in the reverse order.

Stepper Motor interface- Schematic Diagram (for 8085):



Detailed Connection diagram between 8085 and 8255:



ASSEMBLY LANGUAGE PROGRAM (8085)

```
Main   :  MVI                    ;   80H → Control word to
A, 80                            configure PA,PB,PC in    O/P
OUT                              ;  Write control
CWR_Address                      word in CWR
                                 of 8255
MVI A, 77                        ; Code for the
                                 Phase 1
OUT                              ; sent to motor
PortA_Address                    via port A of
                                 8255             ;
CALL DELAY                       ;   Delay
                                 subroutine
MVI A,  BB                       ; Code for the
                                 Phase II
OUT                              ; sent to motor
PortA_Address                    via port A of
                                 8255
CALL DELAY                       ;   Delay
                                 subroutine.
MVI A,  DD                       ; Code for the
                                 Phase III
OUT                              ; sent to motor
PortA_Address                    via port A of
                                 8255;
CALL DELAY                       ; Delay
                                 subroutine
```

```
MVI A,  EE H                ;  Code for the Phase 1
OUT PortA_Address           ; sent to motor via port A of 8255          ;
CALL DELAY                  ;  Delay subroutine
JMP MAIN                    ; Keep the motor rotating continuously.
```

DELAY Subroutine

```
MVI C, FF                   ;  Load C with FF -- Change it for the speed variation
LOOP1:  MVI D,FF            ; Load D with FF
LOOP2:  DCR D
JNZ  LOOP2
DCR C
JNZ LOOP1
RET                         ; Return to main program.
```

4.11 DMA CONTROLLER: (8257)

- 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 CPU.
- Using a DMA controller, the device requests the CPU 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 CPU.

How DMA Operations are performed?

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 CPU and waits for the CPU to assert the HLDA.

- Then the microprocessor tri-states all the data bus, address bus, and control bus. The CPU leaves the control over bus and acknowledges the HOLD request through HLDA signal.
- Now the CPU is in HOLD state and the DMA controller has to manage the operations over buses between the CPU, memory, and I/O devices.

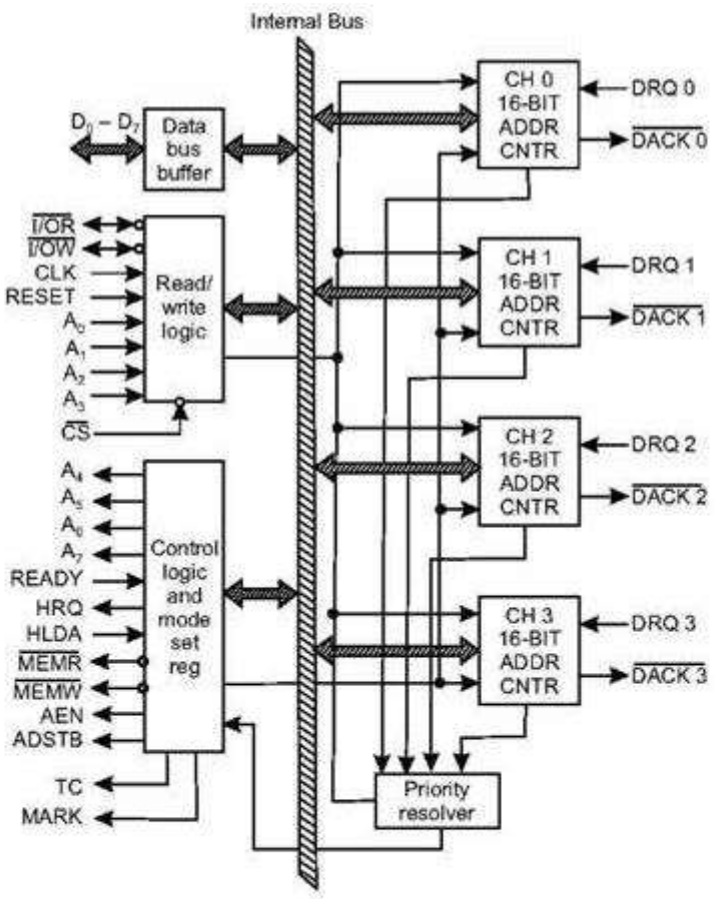
Features of 8257:

Here is a list of some of the prominent features of 8257 −

- It has four channels which can be used over four I/O devices.
- Each channel has 16-bit address and 14-bit counter.
- Each channel can transfer data up to 64kb.
- Each channel can be programmed independently.
- Each channel can perform read transfer, write transfer and verify transfer operations.
- It generates MARK signal to the peripheral device that 128 bytes have been transferred.
- It requires a single phase clock.
- Its frequency ranges from 250Hz to 3MHz.
- It operates in 2 modes, i.e., **Master mode** and **Slave mode**.

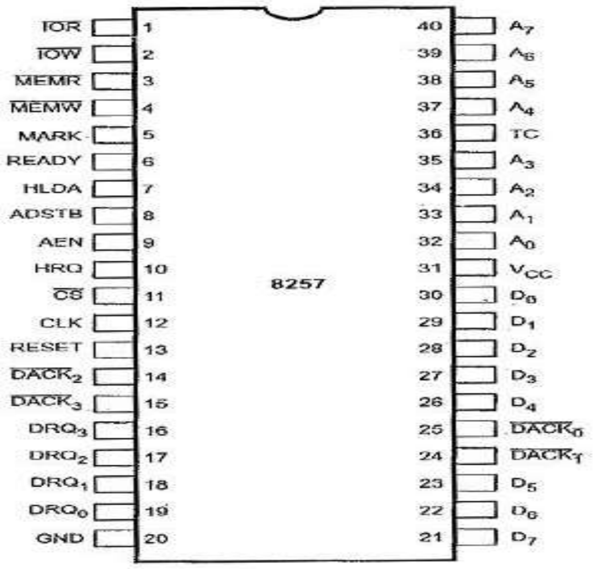
8257 Architecture:

The following image shows the architecture of 8257 −



8257 Pin Description:

The following image shows the pin diagram of an 8257 DMA controller –



DRQ_0–DRQ_3

These are the four individual channel DMA request inputs, which are used by the peripheral devices for using DMA services. When the fixed priority mode is selected, then DRQ_0 has the highest priority and DRQ_3 has the lowest priority among them.

$DACK_0$ – $DACK_3$

These are the active-low DMA acknowledge lines, which updates the requesting peripheral about the status of their request by the CPU. These lines can also act as strobe lines for the requesting devices.

D_0 – D_7

These are bidirectional, data lines which are used to interface the system bus with the internal data bus of DMA controller. In the Slave mode, it carries command words to 8257 and status word from 8257. In the master mode, these lines are used to send higher byte of the generated address to the latch. This address is further latched using ADSTB signal.

IOR

It is an active-low bidirectional tri-state input line, which is used by the CPU to read internal registers of 8257 in the Slave mode. In the master mode, it is used to read data from the peripheral devices during a memory write cycle.

IOW

It is an active low bi-direction tri-state line, which is used to load the contents of the data bus to the 8-bit mode register or upper/lower byte of a 16-bit DMA address register or terminal count register. In the master mode, it is used to load the data to the peripheral devices during DMA memory read cycle.

CLK

It is a clock frequency signal which is required for the internal operation of 8257.

RESET

This signal is used to RESET the DMA controller by disabling all the DMA channels.

A_0 - A_3

These are the four least significant address lines. In the slave mode, they act as an input, which selects one of the registers to be read or written. In the master mode, they are the four least significant memory address output lines generated by 8257.

CS

It is an active-low chip select line. In the Slave mode, it enables the read/write operations to/from 8257. In the master mode, it disables the read/write operations to/from 8257.

A_4 - A_7

These are the higher nibble of the lower byte address generated by DMA in the master mode.

READY

It is an active-high asynchronous input signal, which makes DMA ready by inserting wait states.

HRQ

This signal is used to receive the hold request signal from the output device. In the slave mode, it is connected with a DRQ input line 8257. In Master mode, it is connected with HOLD input of the CPU.

HLDA

It is the hold acknowledgement signal which indicates the DMA controller that the bus has been granted to the requesting peripheral by the CPU when it is set to 1.

MEMR

It is the low memory read signal, which is used to read the data from the addressed memory locations during DMA read cycles.

MEMW

It is the active-low three state signal which is used to write the data to the addressed memory location during DMA write operation.

ADST

This signal is used to convert the higher byte of the memory address generated by the DMA controller into the latches.

AEN

This signal is used to disable the address bus/data bus.

TC

It stands for 'Terminal Count', which indicates the present DMA cycle to the present peripheral devices.

MARK
The mark will be activated after each 128 cycles or integral multiples of it from the beginning. It indicates the current DMA cycle is the 128th cycle since the previous MARK output to the selected peripheral device.

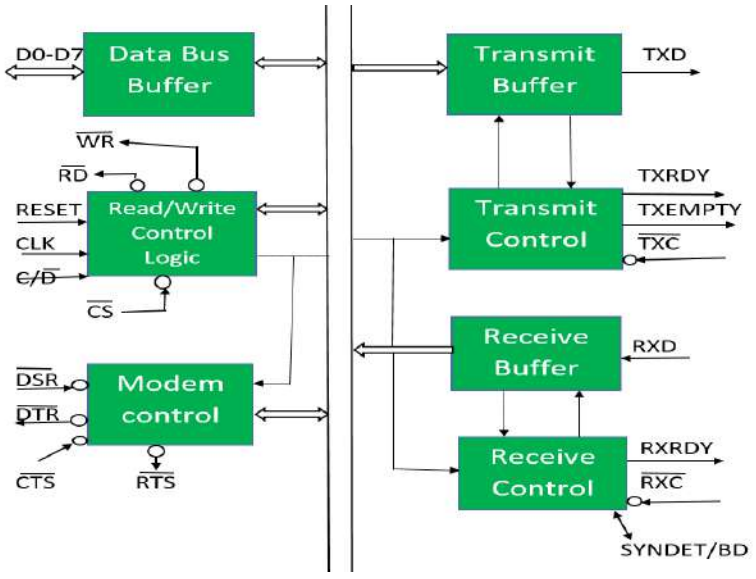
V_{cc}
It is the power signal which is required for the operation of the circuit

4.12 USART: (8251)

8251 universal synchronous asynchronous receiver transmitter (USART) acts as a mediator between microprocessor and peripheral to transmit serial data into parallel form and vice versa.

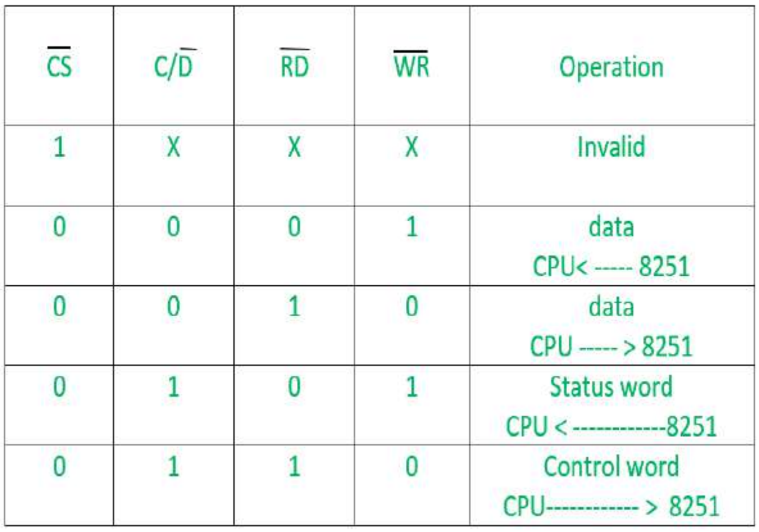
1. It takes data serially from peripheral (outside devices) and converts into parallel data.
2. After converting the data into parallel form, it transmits it to the CPU.
3. Similarly, it receives parallel data from microprocessor and converts it into serial form.
4. After converting data into serial form, it transmits it to outside device (peripheral).

Block Diagram of 8251 USART –



Data bus buffer:
This block helps in interfacing the internal data bus of 8251 to the system data bus. The data transmission is possible between 8251 and CPU by the data bus buffer block.

Read/Write control logic:
It is a control block for overall device. It controls the overall working by selecting the operation to be done. The operation selection depends upon input signals as:



$\overline{CS}$	C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	Operation
1	X	X	X	Invalid
0	0	0	1	data CPU< ----- 8251
0	0	1	0	data CPU -----> 8251
0	1	0	1	Status word CPU <------------8251
0	1	1	0	Control word CPU------------> 8251

In this way, this unit selects one of the three registers- data buffer register, control register, status register.

<u>Modem control (modulator/demodulator):</u>
A device converts analog signals to digital signals and vice-versa and helps

the computers to communicate over telephone lines or cable wires. The following are active-low pins of Modem.

- **DSR:** Data Set Ready signal is an input signal.
- **DTR:** Data terminal Ready is an output signal.
- **CTS:** It is an input signal which controls the data transmit circuit.
  RTS: It is an output signal which is used to set the status RTS.

Transmit buffer:

This block is used for parallel to serial converter that receives a parallel byte for conversion into serial signal and further transmission onto the common channel.

- **TXD:** It is an output signal, if its value is one, means transmitter will transmit the data.

Transmit control:

This block is used to control the data transmission with the help of following pins:

- **TXRDY:** It means transmitter is ready to transmit data character.
- **TXEMPTY:** An output signal which indicates that TXEMPTY pin has transmitted all the data characters and transmitter is empty now.
- **TXC:** An active-low input pin which controls the data transmission rate of transmitted data.

Receive buffer:

This block acts as a buffer for the received data.

- **RXD:** An input signal which receives the data.

Receive control:

This block controls the receiving data.

- **RXRDY:** An input signal indicates that it is ready to receive the data.
- **RXC:** An active-low input signal which controls the data transmission rate of received data.
- **SYNDET/BD:** An input or output terminal. External synchronous mode-input terminal and asynchronous mode-output terminal.

UNIT-5: MICROPROCESSOR (ARCHITECTURE AND PROGRAMMING -8086-16 BIT)

5.1 INTRODUCTION:

- 8086 Microprocessor is an enhanced version of 8085Microprocessor that was designed by Intel in 1976.
- It is a 16-bit Microprocessor having 20 address lines and16 data lines that provides up to 1MB storage.
- It consists of powerful instruction set, which provides operations like multiplication and division easily.
- It supports two modes of operation, i.e. Maximum mode and Minimum mode.
- Maximum mode is suitable for system having multiple processors and Minimum mode is suitable for system having a single processor.

5.2 8086 MICROPROCESSOR FEATURES:

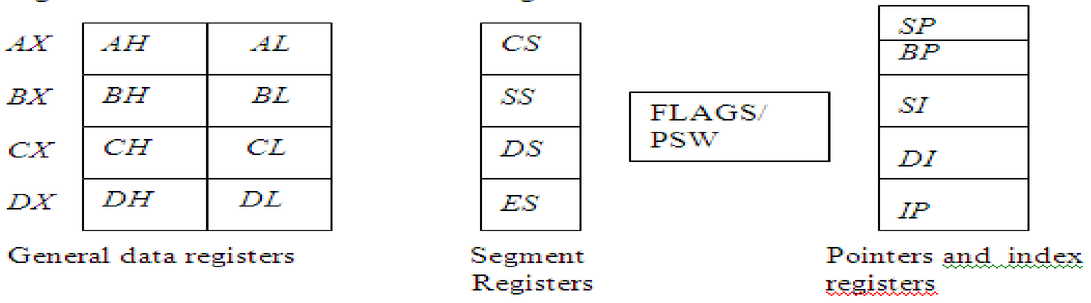
- It is 16-bit microprocessor
- It has a 16-bit data bus, so it can read data from or write data to memory and ports either 16-bit or 8-bit at a time.
- It has 20 bit address bus and can access up to 2^{20} memory locations (1 MB).
- It can support up to 64K I/O ports
- It provides 14, 16-bit registers
- It has multiplexed address and data bus AD_0-AD_{15} & A_{16}-A_{19}
- It requires single phase clock with 33% duty cycle to provide internal timing.
- Pre fetches up to 6 instruction bytes from memory and queues them in order to speed up the processing.
- 8086 supports 2 modes of operation
  1. Minimum mode
  2. Maximum mode
- It is available in 3 versions based on the frequency of operation –
  1. 8086 → 5MHz
  2. 8086-2 → 8MHz
  3. (c)8086-1 → 10 MHz
- It uses two stages of pipelining, i.e. fetch Stage and Execute Stage, which improves performance.
- Fetch stage can pre fetch up to 6 bytes of instructions and stores them in the queue.
- Execute stage executes these instructions.
- It has 256 vectored interrupts.
- It consists of 29,000 transistors.

5.3 REGISTER ORGANIZATION:

- 8086 has a powerful set of registers known as general purpose registers and special purpose registers.
- All of them are 16-bit registers.
- **General purpose registers:**
  - ✓ These registers can be used as either 8-bit registers or 16-bit registers.
  - ✓ They may be either used for holding data, variables and intermediate results temporarily or for other purposes like a counter or for storing offset address for some particular addressing modes etc.
- **Special purpose registers:**
  - ✓ These registers are used as segment registers, pointers, index registers or as offset storage registers for particular addressing modes.

- **The 8086 registers are classified into the following types:**
  - ✓ General Data Registers
  - ✓ Segment Registers
  - ✓ Pointers and Index Registers
  - ✓ Flag Register

The register set of 8086 can be categorized into 4 different groups. The register organization of 8086 is shown in the figure.



Register organization of 8086

1. <u>General Data Registers:</u>

- The registers AX, BX, CX and DX are the general purpose 16-bit registers.
- AX is used as 16-bit accumulator. The lower 8-bit is designated as AL and higher 8-bit is designated as AH.
- AL Can be used as an 8-bit accumulator for 8-bit operation.
- All data register can be used as either 16 bit or 8 bit. BX is a 16 bit register, but BL indicates the lower 8-bit of BX and BH indicates the higher 8-bit of BX.
- The register BX is used as offset storage for forming physical address in case of certain addressing modes.
- The register CX is used default counter in case of string and loop instructions.
- DX register is a general purpose register which may be used as an implicit operand or destination in case of a few instructions.

2. **Segment Registers:**

- There are 4 segment registers. They are:
  - ✓ Code Segment Register(CS)
  - ✓ Data Segment Register(DS)
  - ✓ Extra Segment Register(ES)
  - ✓ Stack Segment Register(SS)

- The 8086 architecture uses the concept of **segmented memory**. 8086 able to address a memory capacity of 1 megabyte and it is byte organized. This 1 megabyte memory is divided into 16 logical segments. Each segment contains 64 Kbytes of memory.

- **Code segment register (CS):**
  It is used for addressing memory location in the code segment of the memory, where the executable program is stored.

- **Data segment register (DS):**
  It points to the data segment of the memory where the data is stored.

- **Extra Segment Register (ES) :**
  It also refers to a segment in the memory which is another data segment in the memory.

- **Stack Segment Register (SS):**
  - ➢ It is used for addressing stack segment of the memory. The stack segment is that segment of memory which is used to store stack data.
  - ➢ While addressing any location in the memory bank, the **physical address** is calculated from two parts:

    Physical address= segment address + offset address
  - ➢ The first is segment address, the segment registers contain 16-bit segment base addresses, related to different segment.
  - ➢ The second part is the offset value in that segment.

3. **Pointers and Index Registers:**

- The index and pointer registers are given below:
  - ✓ IP—Instruction pointer-store memory location of next instruction to be executed
  - ✓ BP—Base pointer
  - ✓ SP—Stack pointer
  - ✓ SI—Source index
  - ✓ DI—Destination index

- The pointers registers contain offset within the particular segments.
  - ✓ The pointer register IP contains offset within the code segment.
  - ✓ The pointer register BP contains offset within the data segment.
  - ✓ Thee pointer register SP contains offset within the stack segment.

- The index registers are used as general purpose registers as well as for offset storage in case of indexed, base indexed and relative base indexed addressing modes.
- The register SI is used to store the offset of source data in data segment.
- The register DI is used to store the offset of destination in data or extra segment.
- The index registers are particularly useful for string manipulation.

4. 8086 flag register and its functions:

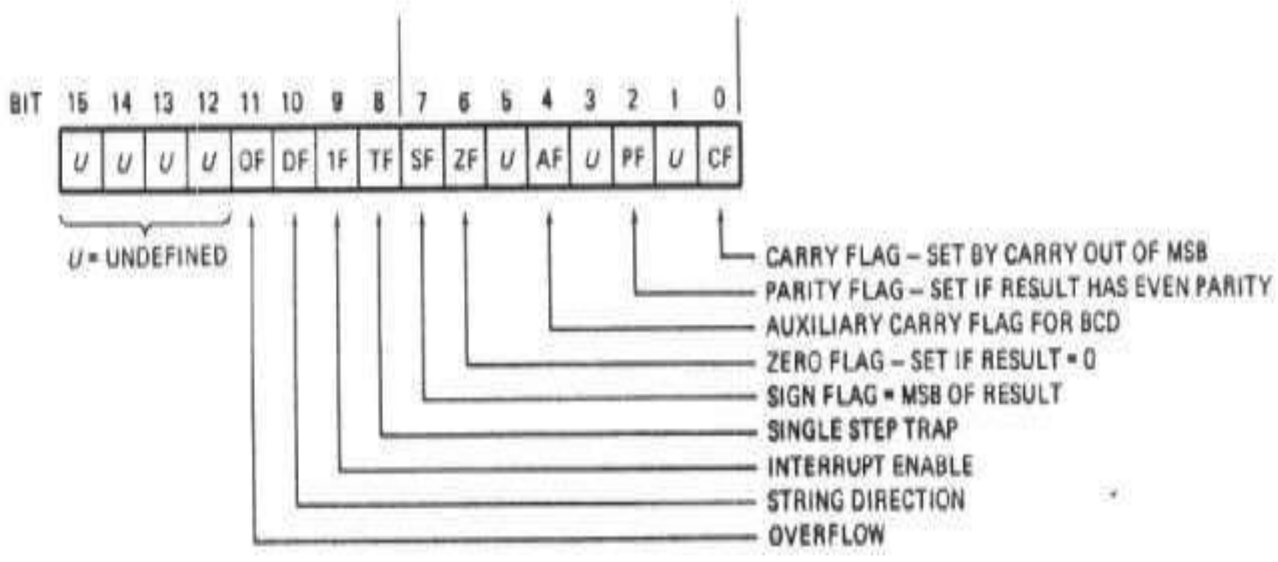
- The 8086 flag register contents indicate the results of computation in the ALU. It also contains some flag bits to control the CPU operations.

- A 16 bit flag register is used in 8086. It is divided into two parts.
  - ✓ Condition code or status flags
  - ✓ Machine control flags

- The **condition code flag register** is the lower byte of the 16-bit flag register. The condition code flag register is identical to 8085 flag register, with an additional overflow flag.

- The **control flag register** is the higher byte of the flag register. It contains three flags namely direction flag (D), interrupt flag (I) and trap flag (T).

Flag register configuration


BIT 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
U U U U OF DF IF TF SF ZF U AF U PF U CF
U = UNDEFINED
CARRY FLAG – SET BY CARRY OUT OF MSB
PARITY FLAG – SET IF RESULT HAS EVEN PARITY
AUXILIARY CARRY FLAG FOR BCD
ZERO FLAG – SET IF RESULT = 0
SIGN FLAG = MSB OF RESULT
SINGLE STEP TRAP
INTERRUPT ENABLE
STRING DIRECTION
OVERFLOW

The description of each flag bit is as follows:

SF (Sign Flag):
This flag is set, when the result of any computation is negative. For signed computations the sign flag equals the MSB of the result.

ZF (Zero Flag):

This flag is set, if the result of the computation or comparison performed by the previous instruction is zero.

PF (Parity Flag):
This flag is set to 1, if the lower byte of the result contains even number of 1's.

CF (Carry Flag):
This flag is set, when there is a carry out of MSB in case of addition or a borrow in case of subtraction.

AF (Auxiliary Carry Flag):
This is set, if there is a carry from the lowest nibble, i.e., bit three during addition, or borrow for the lowest nibble, i.e., bit three, during subtraction.

OF (Over flow Flag):
This flag is set, if an overflow occurs, i.e., if the result of a signed operation is large enough to accommodate in a destination register. The result is of more than 7-bits in size in case of 8-bit signed operation and more than 15-bits in size in case of 16-bit sign operations, and then the overflow will be set.

TF (Tarp Flag):

If this flag is set, the processor enters the single step execution mode. The processor executes the current instruction and the control is transferred to the Trap interrupt service routine.

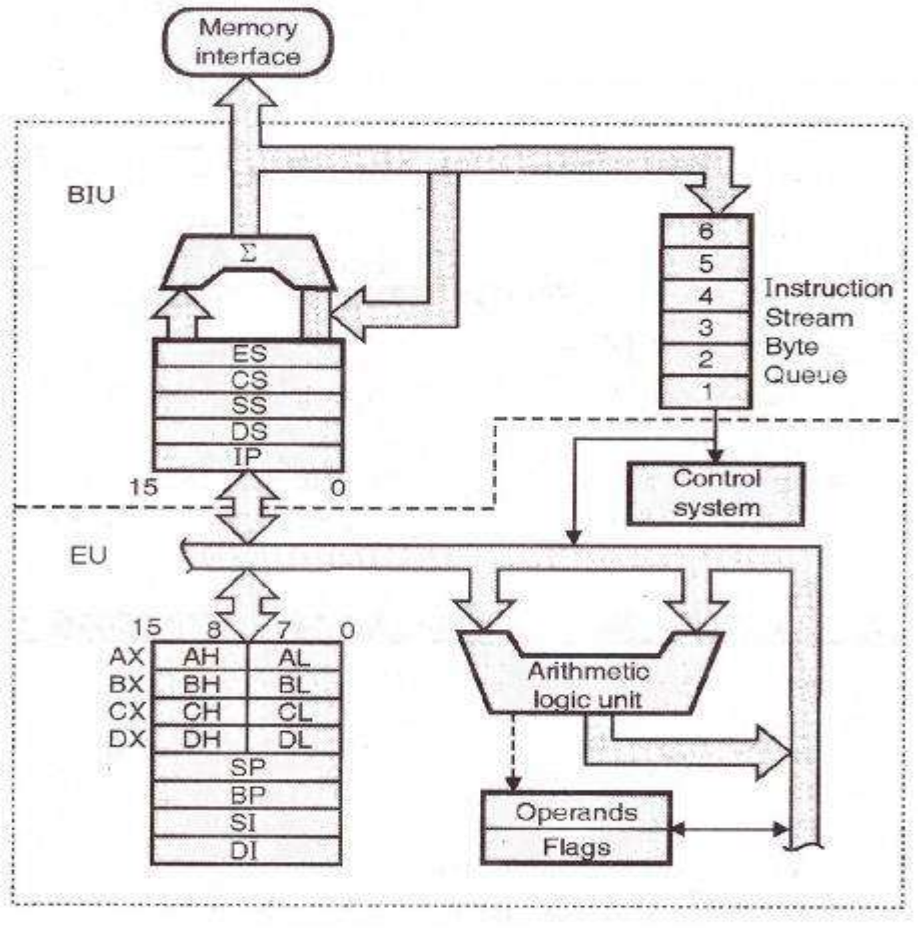
IF (Interrupt Flag):
If this flag is set, the mask able interrupts are recognized by the CPU, otherwise they are ignored.

D (Direction Flag):
This is used by string manipulation instructions. If this flag bit is '0', the string is processed beginning from the lowest address to the highest address, i.e., auto incrementing mode. Otherwise, the string is processed from the highest address towards the lowest address, i.e., auto decrementing mode.

5.4 ARCHITECTURE OF 8086 MICROPROCESSOR:

- As shown in the below figure, the 8086 CPU is divided into two independent functional parts
  - ✓ Bus Interface Unit(BIU)
  - ✓ Execution Unit(EU)

Dividing the work between these two units' speeds up processing.



8086 internal architecture

The Execution Unit (EU):

- The execution unit of the 8086 tells the BIU where to fetch instructions or data from, decodes instructions, and executes instructions.
- The EU contains **control circuitry,** which directs internal operations.
- A decoder in the EU translates instructions fetched from memory into a series of actions, which the EU carries out.
- The EU has a 16-bit **arithmetic logic unit** (ALU) which can add, subtract, AND, OR, XOR, increment, decrement, complement or shift binary numbers.

- **The main functions of EU are:**
  - ✓ Decoding of Instructions
  - ✓ Execution of instructions

- ✓ **Steps:**
- ✓ EU extracts instructions from top of queue in BIU
- ✓ Decode the instructions
- ✓ Generates operands if necessary
- ✓ Passes operands to BIU & requests it to perform read or write bus cycles to memory or I/O
- ✓ Perform the operation specified by the instruction on operands

Or

- Execution unit gives instructions to BIU stating from where to fetch the data and then decode and execute those instructions. Its function is to control operations on data using the instruction decoder & ALU. EU has no direct connection with system buses as shown in the above figure, it performs operations over data through BIU.

Bus Interface Unit (BIU):

- The BIU sends out addresses, fetches instructions from memory, reads data from ports and memory, and writes data to ports and memory.
- In simple words, the BIU handles all transfers of data and addresses on the buses for the execution unit.

Or

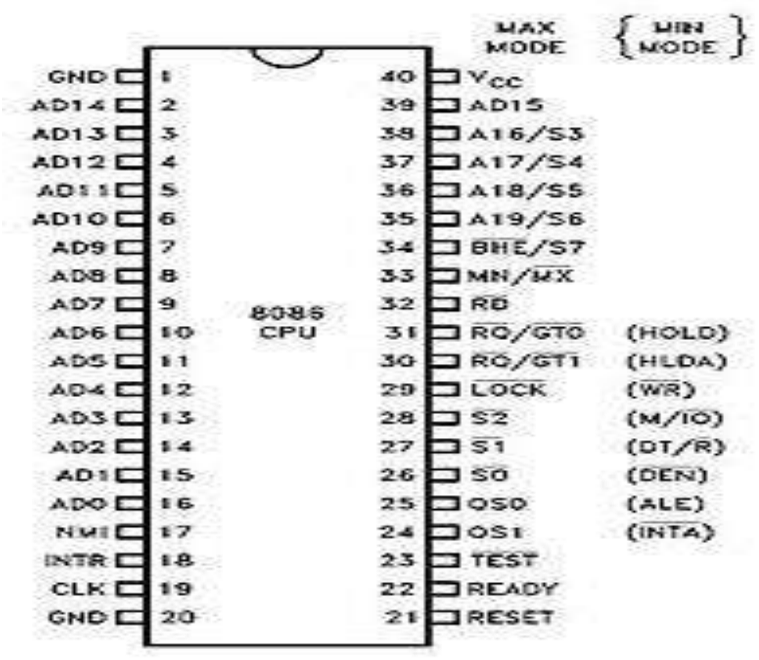
- BIU takes care of all data and addresses transfers on the buses for the EU like sending addresses, fetching instructions from the memory, reading data from the ports and the memory as well as writing data to the ports and the memory.

- EU has no direction connection with System Buses so this is possible with the BIU. EU and BIU are connected with the Internal Bus.

8086 HAS PIPELINING ARCHITECTURE:

- While the EU is decoding an instruction or executing an instruction, which does not require use of the buses, the BIU fetches up to six instruction bytes for the following instructions.
- The BIU stores these pre-fetched bytes in a first-in-first-out register set called a queue.
- When the EU is ready for its next instruction from the queue in the BIU. This is much faster than sending out an address to the system memory and waiting for memory to send back the next instruction byte or bytes.
- Except in the case of JMP and CALL instructions, where the queue must be dumped and then reloaded starting from a new address, this pre-fetch and queue scheme greatly speeds up processing.
- Fetching the next instruction while the current instruction executes is called **pipelining**.

5.5 PIN DIAGRAM OF 8086:



Intel 8086 is a 16-bit HMOS microprocessor. It is available in 40 pin DIP chip. It uses a 5V DC supply for its operation. The 8086 uses 20-line address bus. It has a 16-line data bus. The 20 lines of the address bus operate in multiplexed mode. The 16-low order address bus lines have been multiplexed with data and 4 high-order address bus lines have been multiplexed with status signals.

AD0-AD15:
Address/Data bus. These are low order address bus. They are multiplexed with data. When AD lines are used to transmit memory address the symbol A is used instead of AD, for example A0-A15. When data are transmitted over AD lines the symbol D is used in place of AD, for example D0-D7, D8-D15 or D0-D15.

A16-A19:
High order address bus. These are multiplexed with status signals.

S2, S1, S0:
Status pins. These pins are active during T4, T1 and T2 states and is returned to passive state (1, 1, 1 during T3 or Tw (when ready is inactive). These are used by the 8288 bus controller for generating all the memory and I/O operation) access control signals. Any change in S2, S1, and S0 during T4 indicates the beginning of a bus cycle.

S2	S1	S0	Characteristics
0	0	0	Interrupt acknowledge
0	0	1	Read I/O port

0	1	0	Write I/O port
0	1	1	Halt
1	0	0	Code access1 0 1 Read memory
1	1	0	Write memory
1	1	1	Passive State

A16/S3, A17/S4, A18/S5, A19/S6:

- The specified address lines are multiplexed with corresponding status signals.
- These are the 4 address/status buses. During the first clock cycle, it carries 4-bit address and later it carries status signals.

BHE'/S7:

- Bus High Enable/Status. During T1 it is low. It is used to enable data onto the most significant half of data bus, D8-D15.
- 8-bit device connected to upper half of the data bus use BHE (Active Low) signal.
- It is multiplexed with status signal S7.
- S7 signal is available during T2, T3 and T4.

RD':
This is used for read operation. It is an output signal. It is active when low.

READY:
This is the acknowledgement from the memory or slow device that they have completed the data transfer. The signal made available by the devices is synchronized by the 8284A clock generator to provide ready input to the microprocessor. The signal is active high (1).

INTR:
Interrupt Request. This is triggered input. This is sampled during the last clock cycles of each instruction for determining the availability of the request. If any interrupt request is found pending, the processor enters the interrupt acknowledge cycle. This can be internally masked after resulting the interrupt enable flag. This signal is active high (1) and has been synchronized internally.

NMI:
Non maskable interrupt. This is an edge triggered input which results in a type II interrupt. A subroutine is then vectored through an interrupt vector lookup table which is located in the system memory. NMI is non-maskable internally by software. A transition made from low (0) to high (1) initiates the interrupt at the end of the current instruction. This input has been synchronized internally.

INTA:
Interrupt acknowledge. It is active low (0) during T2, T3 and Tw of each interrupt acknowledge cycle.

MN/MX':
Minimum/Maximum. This pin signal indicates what mode the processor will operate in.

RQ'/GT1', RQ'/GT0':
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.

LOCK':

- It's an active low pin. It indicates that other system bus masters have not been allowed to gain control of the system bus while LOCK' is active low (0). The LOCK signal will be active until the completion of the next instruction.
- 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

RESET:
This pin requires the microprocessor to terminate its present activity immediately. The signal must be active high (1) for at least four clock cycles.

TEST':
This examined by a 'WAIT' instruction. If the TEST pin goes low (0), execution will continue, else the processor remains in an idle state. The input is internally synchronized during each of the clock cycle on leading edge of the clock.

CLK:

- Clock Input. The clock input provides the basic timing for processing operation and bus control activity. It's an asymmetric square wave with a 33% duty cycle.

Vcc:
Power Supply (+5V D.C.)

GND:
Ground

QS1, QS0:
Queue Status. These signals indicate the status of the internal 8086 instruction queue according to the table shown below

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

DT/R:
Data Transmit/Receive. This pin is required in minimum systems that want to use an 8286 or 8287 data bus transceiver. The direction of data flow is controlled through the transceiver.

DEN:
Data enable. This pin is provided as an output enable for the 8286/8287 in a minimum system which uses transceiver. DEN is active low (0) during each memory and input-output access and for INTA cycles.

HOLD/HOLDA:
HOLD indicates that another master has been requesting a local bus .This is an active high (1). The microprocessor receiving the HOLD request will issue HLDA (high) as an acknowledgement in the middle of a T4 or T1 clock cycle.

ALE:
Address Latch Enable. ALE is provided by the microprocessor to latch the address into the 8282 or 8283 address latch. It is an active high (1) pulse during T1 of any bus cycle. ALE signal is never floated, is always integer.

5.6 GENERAL BUS OPERATION OF 8086:

- 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 de multiplexed using a few latches and transceivers, whenever required.
- Basically, all the processor bus cycles consist of at least four clock cycles. These are referred to as T1, T2, T3, and 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.

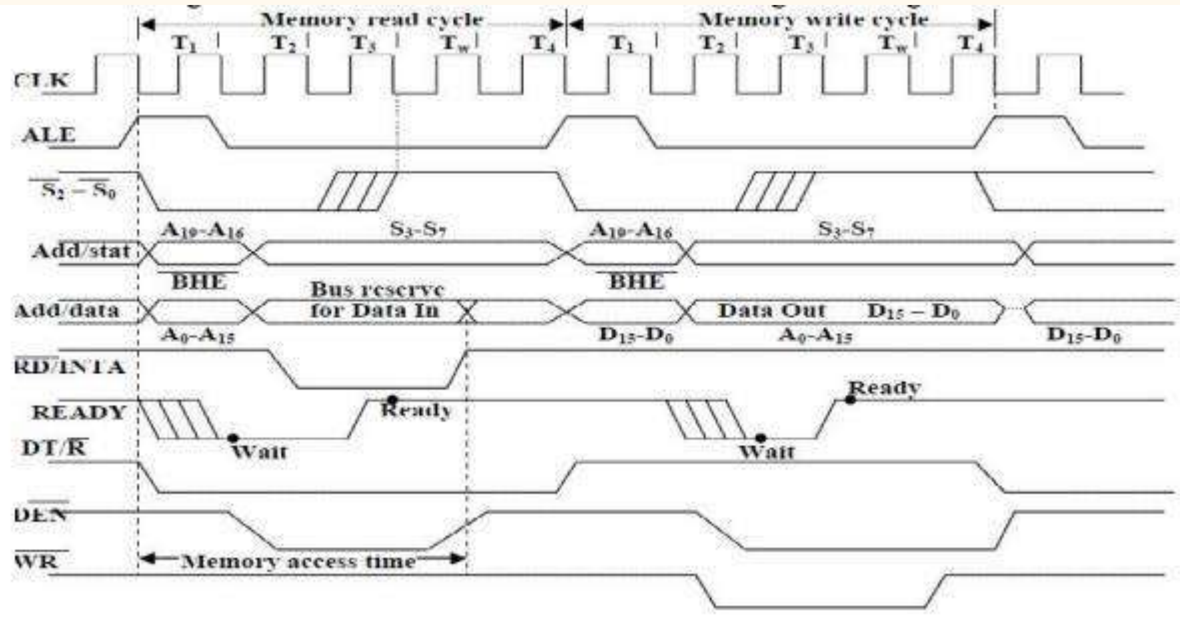


Fig.2.2. General Bus operation cycle

Maximum mode

- In the maximum mode, the 8086 is operated by strapping the MN/MX pin to ground.
- In this mode, the processor derives the status signal S2, S1, S0. Another chip called bus controller derives the control signal using this status information.
- In the maximum mode, there may be more than one microprocessor in the system configuration.

Minimum mode

- In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/MX pin to logic 1.
- In this mode, all the control signals are given out by the microprocessor chip itself.
- There is a single microprocessor in the minimum mode system.

5.7 8086 MEMORY ORGANIZATION:

- Segmented Memory Two types of memory organization are used:
- Linear addressing where the entire memory is available to the processor at all the times (Motorola 68000 family).
- Segmented addressing where the memory space is divided into several segments and the processor is limited to access program instructions and data in specific segments.
- 8086 Memory Organization Each memory location 8086 is a byte while the 8086 is a 16-bits microprocessor.

Memory Segmentation:

- The memory in an 8086 based system is organized as segmented memory.
- The CPU 8086 is able to access 1MB of physical memory. The complete 1MB of memory can be divided into 16 segments, each of 64KB size and is addressed by one of the segment register.
- The 16-bit contents of the segment register actually point to the starting location of a particular segment. The address of the segments may be assigned as 0000H to F000h respectively.
- To address a specific memory location within a segment, we need an offset address. The offset address values are from 0000H to FFFFH so that the physical addresses range from 00000H to FFFFFH.

Advantages of the segmented memory scheme are as follows:

- Allows the memory capacity to be 1MB although the actual addresses to be handled are of 16-bit size.
- Allows the placing of code, data and stack portions of the same program in different parts (segments) of memory, for data and code protection.
- Permits a program and/or its data to be put into different areas of memory each time the program is executed, i.e., provision for relocation is done.

Overlapping and Non-overlapping Memory segments:

- In the overlapping area locations physical address = CS1+IP1 = CS2+IP2. Where '+' indicates the procedure of physical address formation.

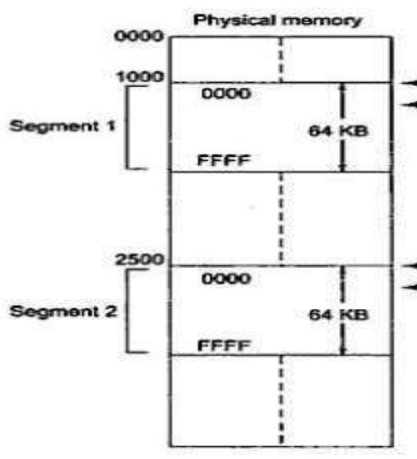


Fig. 1.3(a) *Non-overlapping Segments*

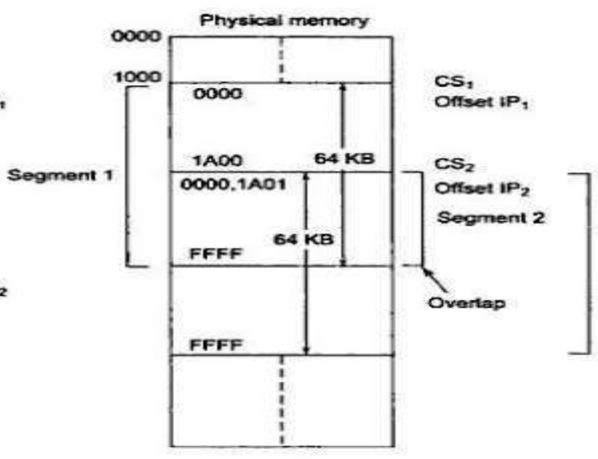
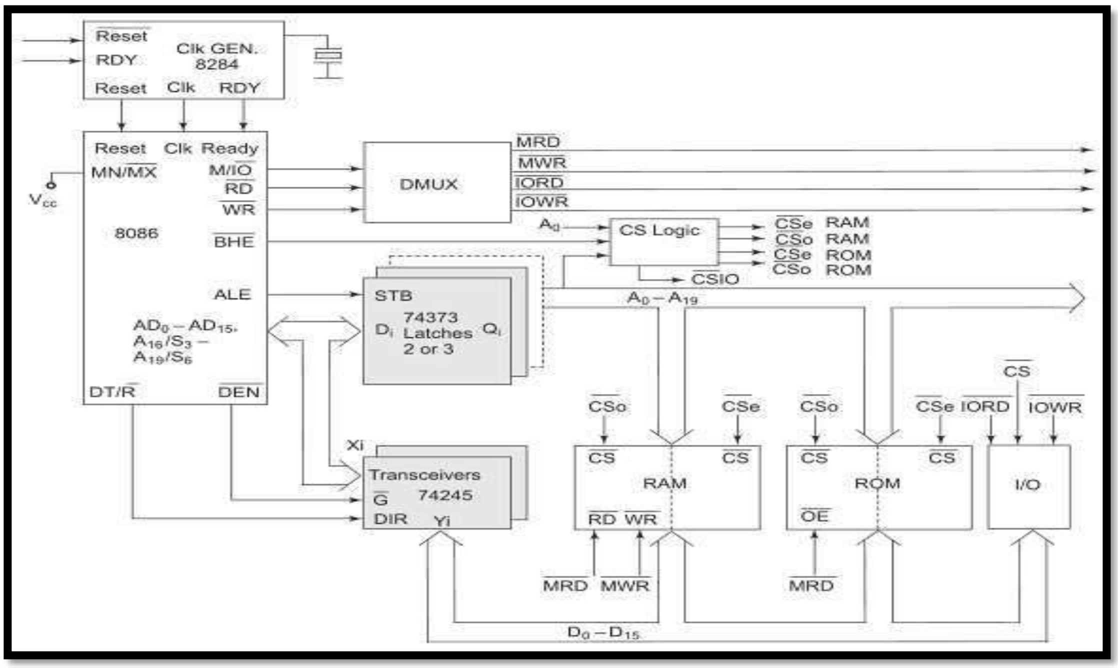


Fig. 1.3(b) *Overlapping Segments*

5.8 MINIMUM MODE & TIMINGS:

Minimum Mode 8086 System:

- The microprocessor 8086 is operated in minimum mode by strapping its MN/MX pin to logic 1.
- In this mode, all the control signals are given out by the microprocessor chip itself. There is a single microprocessor in the minimum mode system.
- The remaining components in the system are latches, transreceivers, clock generator, memory and I/O devices.
- Some type of chip selection logic may be required for selecting memory or I/O devices, depending upon the address map of the system
- Latches are generally buffered output D-type flip-flops like 74LS373 or 8282. They are used for separating the valid address from the multiplexed address/data signals and are controlled by the ALE signal generated by 8086.



Minimum Mode Configuration for 8086

- Since it has 20 address lines and 16 data lines, the 8086 CPU requires three octal address latches and two octal data buffers for the complete address and data separation.
- Transceivers are the bidirectional buffers and sometimes they are called as data amplifiers. They are required to separate the valid data from the time multiplexed address/data signal.
- They are controlled by two signals, namely, DEN’ and DT/R’. The DEN’ signal indicates that the valid data is available on the data bus, while DT/R’ indicates the direction of data, i.e. from or to the processor.
- The system contains memory for the monitor and users program storage. Usually, EPROMS are used for monitor storage, while RAMs for users program storage.
- A system may contain I/O devices for communication with the processor as well as some special purpose I/O devices.
- The clock generator generates the clock from the crystal oscillator and then shapes it and divides to make it more precise so that it can be used as an accurate timing reference for the system.
- The clock generator also synchronizes some external signals with the system clock.
- The working of the minimum mode configuration system can be better described in terms of the timing diagrams rather than qualitatively describing the operations.
- The opcode fetch and read cycles are similar. Hence the timing diagram can be categorized in two parts, the first is the timing diagram for read cycle and the second is the timing diagram for write cycle.

Timing Diagrams:

- Timing diagram is graphical representation of the operations of microprocessor with respect to the time.
- **State**: one cycle of the clock is called state.
- **Machine cycle:** The basic microprocessor operation such as reading a byte from memory or writing a byte to a port is called machine cycle and made up of more than one state.
- **Instruction cycle:** The time required for microprocessor to fetch and execute an entire instruction is called Instruction cycle and made up of more than one machine cycle.

Note: An instruction cycle is made up of machine cycles, and a machine cycle is made up of states. The time for a state is determined by the frequency of the clock signal.

Read cycle timing diagram for Minimum mode:

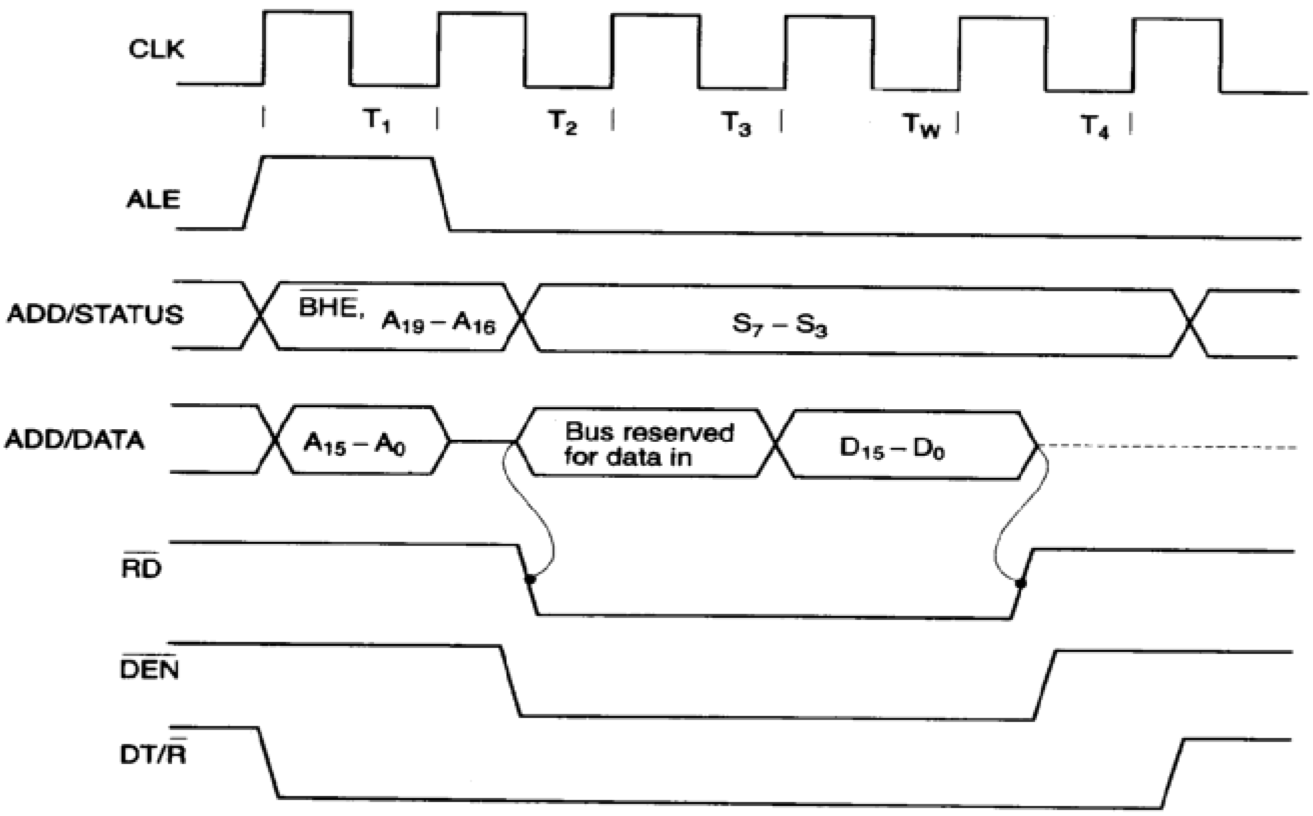


Fig. 1.9(a) *Read Cycle Timing Diagram for Minimum Mode*

- The best way to analyze a timing diagram such as the one to think of time as a vertical line moving from left to right across the diagram.
- **The read cycle** begins in T_1 with the assertion of the address latch enable (ALE) signal and also M/IO' signal.
- During the negative going edge of this signal, the valid address is latched on the local bus. The BHE' and A0 signals address low, high or both bytes.
- From T_1 to T_4, the M/IO' signal indicate a memory or I/O $\overline{RD}$ operation. At T_2, the address is removed from the local bus and is sent to the output. The bus is then tristated. The read (RD') control signal is also activated in T_2.

- The read (RD') signal causes the addressed device to enable its data bus driver. After goes low, the valid data is available on the data bus. The addressed device will drive the READY line high. When the processor returns the read signal to high level, the addressed device will again tristate its bus drivers.

Write cycle timing diagram for Minimum mode:

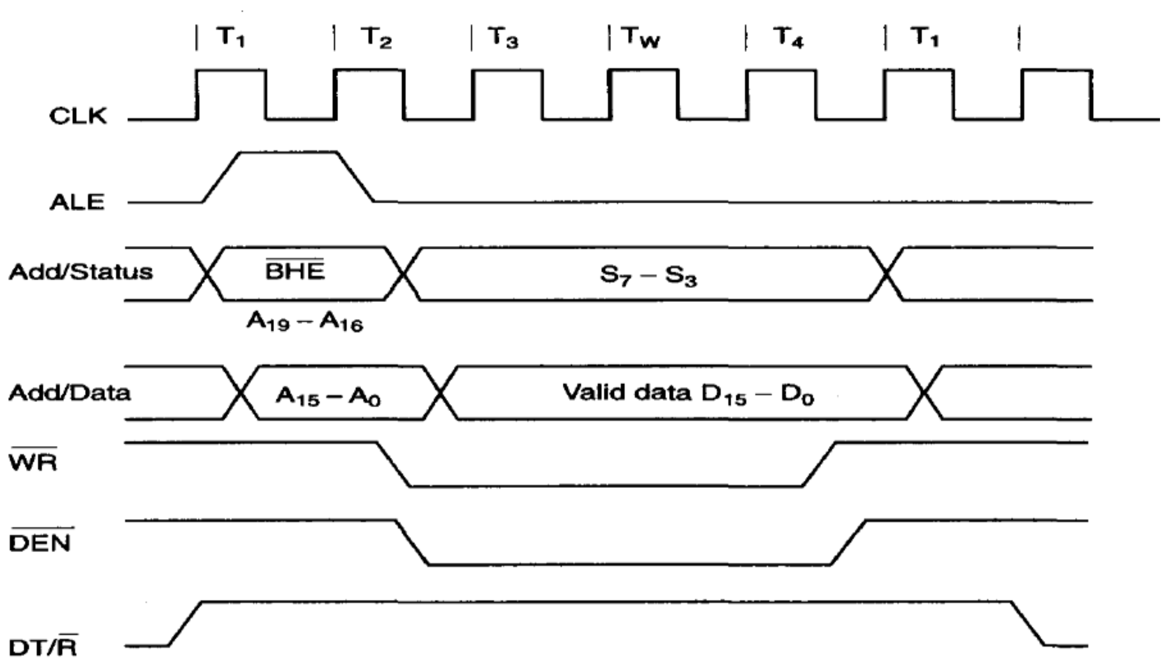
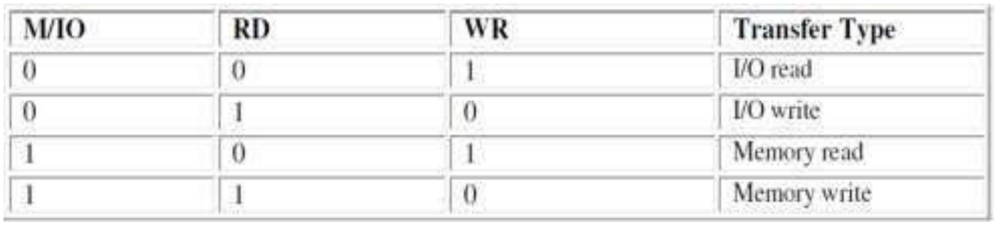


Fig. 1.9(b) *Write Cycle Timing Diagram for Minimum Operation*

- **A write cycle** also begins with the assertion of ALE and the emission of the address. The M/IO' signal is again asserted to indicate a memory or I/O operation.
- In T_2, after sending the address in T_1, the processor sends the data to be written to the addressed location. The data remains on the bus until middle of T_4 state. The WR' becomes active at the beginning of T_2 (unlike RD' is somewhat delayed in T_2 to provide time for floating).
- The BHE' and A_0 signals are used to select the proper byte or bytes of memory or I/O word to be read or written.

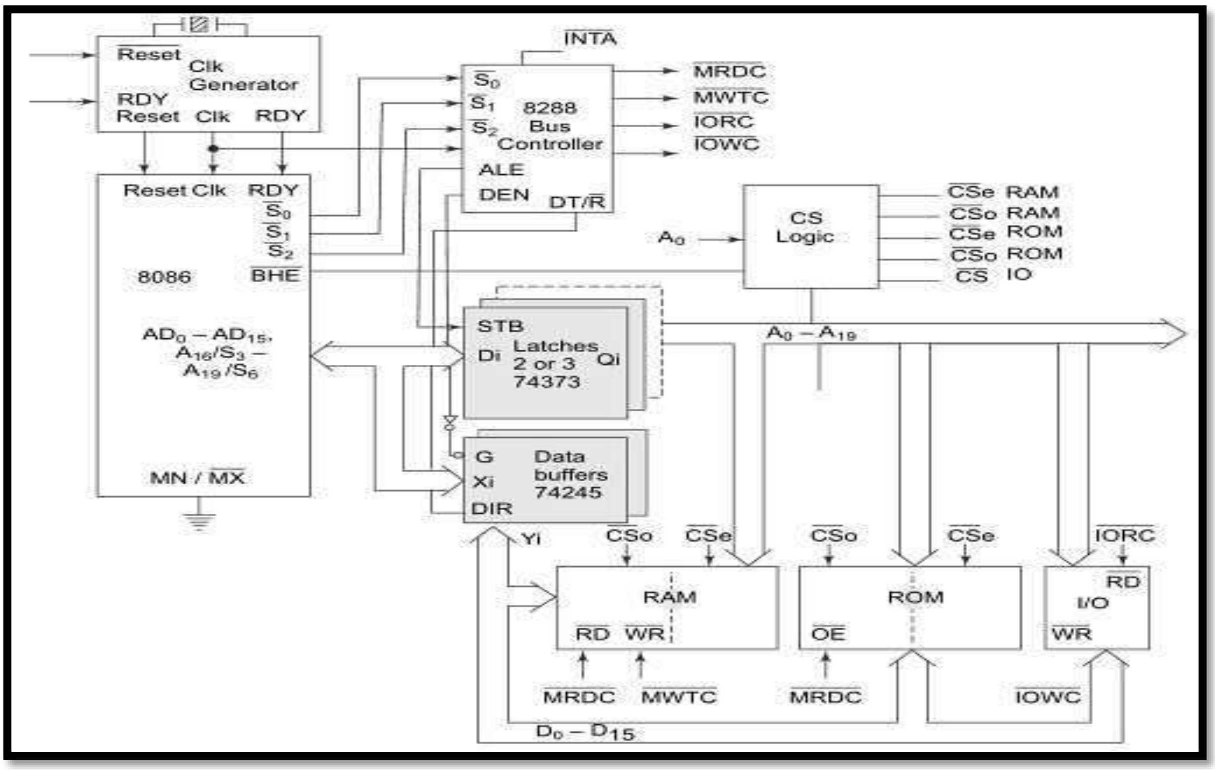


M/IO	RD	WR	Transfer Type
0	0	1	I/O read
0	1	0	I/O write
1	0	1	Memory read
1	1	0	Memory write

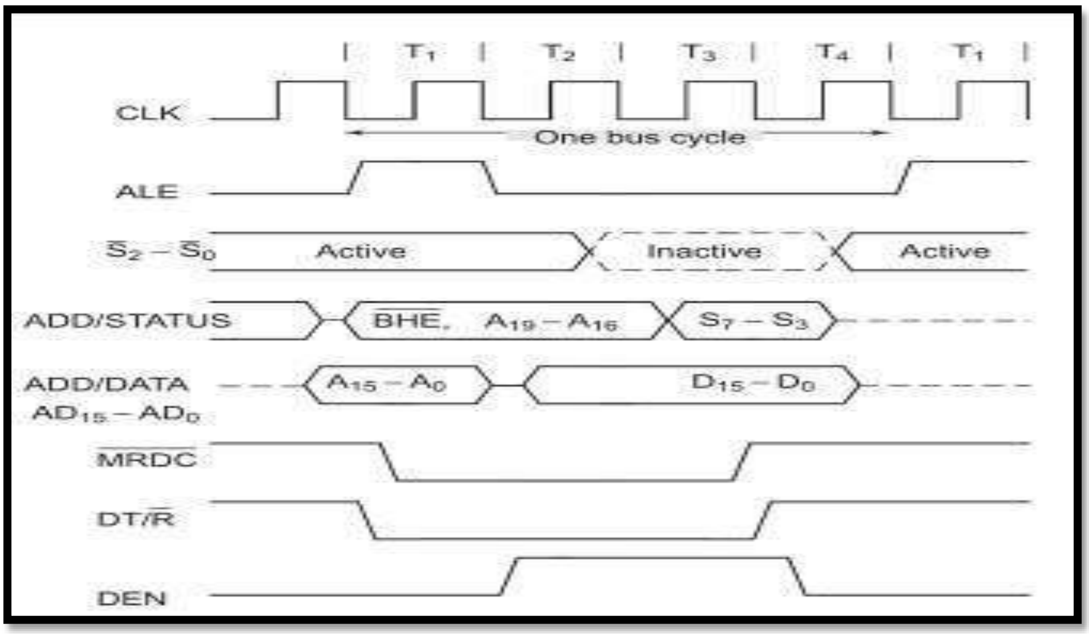
- The M/IO', RD' and WR' signals indicate the types of data transfer as specified in Table.

5.9 MAXIMUM MODE &TIMINGS

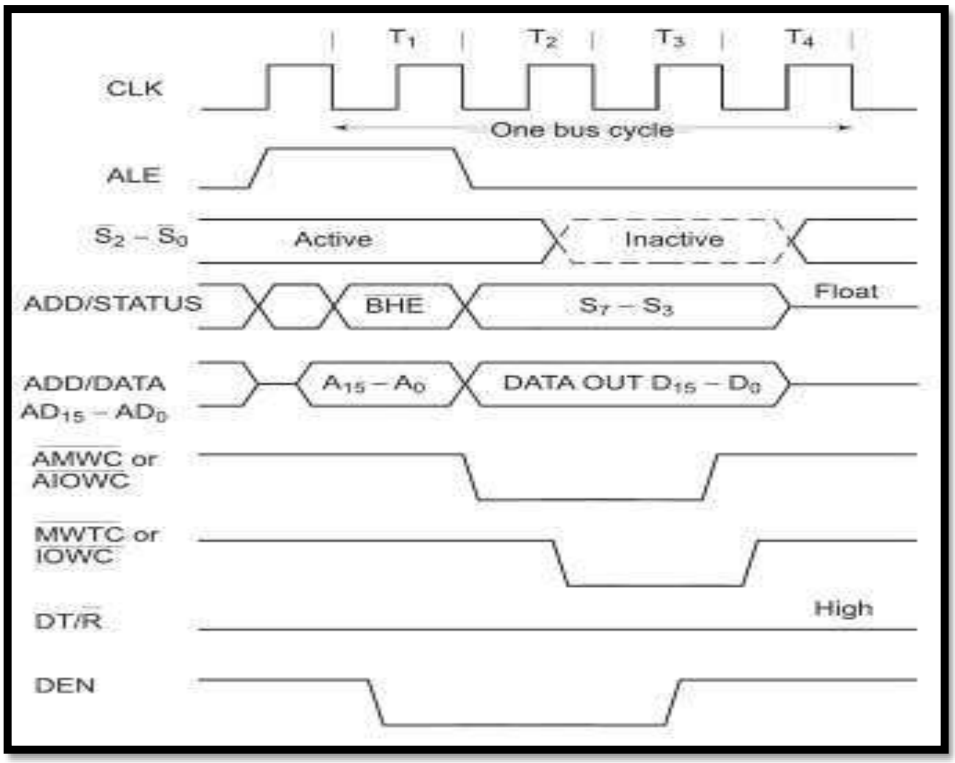
- In the maximum mode, the 8086 is operated by strapping the MN/MX’ pin to ground. In this mode, the processor derives the status signals S2’, S1’ and S0’. Another chip called bus controller derives the control signals using this status information.
- In the maximum mode, there may be more than one microprocessor in the system configuration. The other components in the system are the same as in the minimum mode system. The general system organization is as shown in the below figure. The basic functions of the bus controller chip IC8288, is to derive control signals like RD’ and WR’ (for memory and I/O devices), DEN, DT/R’, ALE, etc. using the information made available by the processor on the status lines.
- The bus controller chip has input lines S2’, S1’ and S0’ and CLK. These inputs to 8288 are driven by the CPU. It derives the outputs ALE, DEN, DT/R’, MWTC’, MRDC’, IORC’, IOWC’ and INTA’.
- INTA’ pin is used to issue two interrupt acknowledge pulses to the interrupt controller or to an interrupting device.
- IORC*, IOWC* are I/O read command and I/O write command signals respectively. These signals enable an IO interface to read or write the data from or to the addressed port. The MRDC*, MWTC* are memory read command and memory write command signals respectively and may be used as memory read and write signals. All these command signals instruct the memory to accept or send data from or to the bus.
- The maximum mode system timing diagrams are also divided in two portions as read (input) and write (output) timing diagrams. The address/data and address/status timings are similar to the minimum mode. ALE is asserted in T1, just like minimum mode. The only difference lies in the status signals used and the available control and advanced command signals.



Read cycle timing diagram for Maximum mode:



Write cycle timing diagram for Maximum mode:



5.10 INTERRUPTS:

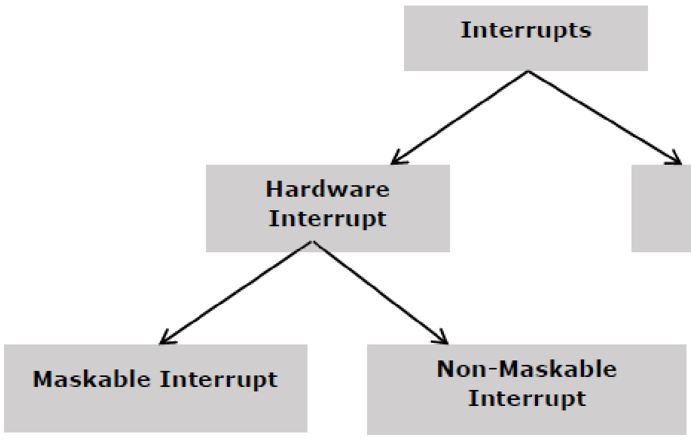
Definition:

The meaning of „interrupts" is to break the sequence of operation. While the CPU is executing a program, on „interrupt" breaks the normal sequence of execution of instructions, diverts its execution to some other program called Interrupt Service Routine (ISR).After executing ISR , the control is transferred back again to the main program. Interrupt processing is an alternative to polling.

Or

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 a 8086 microprocessor –



Need for Interrupt:
Interrupts are particularly useful when interfacing I/O devices that provide or require data at relatively low data transfer rate.

Types of Interrupts:
There are two types of Interrupts in 8086.
1. Hardware Interrupts
2. Software Interrupts

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 is a non-maskable interrupt and INTR is a maskable interrupt having lower priority. One more interrupt pin associated is INTA called interrupt acknowledge.

NMI (non-maskable):

It is a single non-maskable interrupt pin (NMI) having higher priority than the maskable interrupt request pin (INTR) and it is of type 2 interrupt.

When this interrupt is activated, these actions take place: –

- Completes the current instruction that is in progress.
- Pushes the Flag register values on to the stack.
- Pushes the CS (code segment) value and IP (instruction pointer) value of the return address on to the stack.
- IP is loaded from the contents of the word location 00008H.
- CS is loaded from the contents of the next word location 0000AH.
- Interrupt flag and trap flag are reset to 0.

INTR (Maskable):

- The INTR is a maskable interrupt because the microprocessor will be interrupted only if interrupts are enabled using set interrupt flag instruction. It should not be enabled using clear interrupt Flag instruction.
- The INTR interrupt is activated by an I/O port. If the interrupt is enabled and NMI is disabled, then the microprocessor first completes the current execution and sends '0' on INTA pin twice. The first '0' means INTA informs the external device to get ready and during the second '0' the microprocessor receives the 8 bit, say X, from the programmable interrupt controller.

These actions are taken by the microprocessor: –

- First completes the current instruction.
- Activates INTA output and receives the interrupt type, say X.
- Flag register value, CS value of the return address and IP value of the return address are pushed on to the stack.

- IP value is loaded from the contents of word location X × 4
- CS is loaded from the contents of the next word location.
- Interrupt flag and trap flag is reset to 0

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. **It includes: –**

INT- Interrupt instruction with type number

- It is 2-byte instruction. First byte provides the op-code and the second byte provides the interrupt type number. There are 256 interrupt types under this group.

→Its execution includes the following steps: –

- Flag register value is pushed on to the stack.
- CS value of the return address and IP value of the return address are pushed on to the stack.
- IP is loaded from the contents of the word location 'type number' × 4
- CS is loaded from the contents of the next word location.
- Interrupt Flag and Trap Flag are reset to 0
- The starting address for type0 interrupt is 000000H, for type1 interrupt is 00004H similarly for type2 is 00008H and ……so on. The first five pointers are dedicated interrupt pointers. i.e. –
- **TYPE 0** interrupt represents division by zero situation.
- **TYPE 1** interrupt represents single-step execution during the debugging of a program.
- **TYPE 2** interrupt represents non-maskable NMI interrupt.
- **TYPE 3** interrupt represents break-point interrupt.
- **TYPE 4** interrupt represents overflow interrupt.

- The interrupts from Type 5 to Type 31 are reserved for other advanced microprocessors, and interrupts from 32 to Type 255 are available for hardware and software interrupts.

INT 3-Break Point Interrupt Instruction

- It is a 1-byte instruction having op-code is CCH. These instructions are inserted into the program so that when the processor reaches there, then it stops the normal execution of program and follows the break-point procedure.

→Its execution includes the following steps: –

- Flag register value is pushed on to the stack.
- CS value of the return address and IP value of the return address are pushed on to the stack.
- IP is loaded from the contents of the word location 3×4 = 0000CH
- CS is loaded from the contents of the next word location.
- Interrupt Flag and Trap Flag are reset to 0

INTO - Interrupt on overflow instruction

- It is a 1-byte instruction and their mnemonic **INTO**. The op-code for this instruction is CEH. As the name suggests it is a conditional interrupt instruction, i.e. it is active only when the overflow flag is set to 1 and branches to the interrupt handler whose interrupt type number is 4. If the overflow flag is reset then, the execution continues to the next instruction.

→Its execution includes the following steps: –

- Flag register values are pushed on to the stack.
- CS value of the return address and IP value of the return address are pushed on to the stack.
- IP is loaded from the contents of word location 4×4 = 00010H
- CS is loaded from the contents of the next word location.
- Interrupt flag and Trap flag are reset to 0

5.11 ADDRESSING MODES:

The way of specifying data to be operated by an instruction is known as **addressing modes**. This specifies that the given data is an immediate data or an address. It also specifies whether the given operand is register or register pair.

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**

2. Register mode:

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

Or

- It means that the register is the source of an operand for an instruction.

- **Example:**
- ✓ MOV CX, AX ; copies the contents of the 16-bit AX register into the 16-bit CX register
- ✓ ADD BX, AX

3. **Displacement or direct mode:**

- In this type of addressing mode the effective address is directly given in the instruction as displacement.

- **Example:**
- ✓ MOV AX, [DISP]
- ✓ MOV AX, [0500]

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] ; Suppose the register BX contains 4895H, then the contents 4895H are moved to AX
- ✓ ADD CX, [BX]
- ✓ ADD AL, [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 (Index register) or DI (displacement) 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 (BX or BP) to the contents of an Index register (SI or DI).

- Offset= [BX or BP]+[SI or DI]
- BX is used as a base register for data segment, and BP is used as a base register for stack segment.
- **Example:**

✓ ADD AX, [BX+SI]
✓ MOV CX,[BX+SI]
✓ MOV AX,[AX+DI]

8. **Based indexed with displacement mode:**

- In this type of addressing mode the effective address is the sum of index register, base register and displacement.
- Offset= [BX+BP] + [SI or DI] +8-bit or 16-bit displacement.
- **Example:**

✓ MOV AX, [BX+SI+05]→ an example of 8-bit displacement.
✓ MOV AX, [BX+SI+1235H]→ an example of 16-bit displacement.
✓ MOV AL, [SI+BP+2000]

5.12 INSTRUNCTION SET:

The 8086 instructions are categorized into the following main types

- Data transfer instructions
- Arithmetic instructions
- Program control transfer instructions
- Machine control instructions
- Shift/rotate instructions
- Flag manipulation instructions
- String instructions

1. **<u>DATA COPY /TRANSFER INSTRUCTIONS:</u>**

- These type of instructions are used to transfer data from source operand to destination operand. All the store, load, move, exchange input and output instructions belong to this category.

MOV instruction

- It is a general purpose instruction to transfer byte or word from register to register, memory to register, register to memory or with immediate addressing
- MOV destination, source
- Here the source and destination needs to be of the same size that is both 8-bit and both 16-bit.
- MOV instruction does not affect any flags.

MOV BX, 00F2H ;	load the immediate number 00F2H in BX Register
MOV CL, [2000H] ;	Copy the 8 bit content of the memory Location, at a displacement of 2000H from data segment base to the CL register
MOV [589H], BX ;	Copy the 16 bit content of BX register on to the memory location, which at a displacement of 589H from the data segment base.
MOV DS, CX ;	Move the content of CX to DS

PUSH instruction:

- The PUSH instruction decrements the stack pointer by two and copies the word from source to the location where stack pointer now points. Here the source must of word size data. Source can be a general purpose register, segment register or a memory location.
- The PUSH instruction first pushes the most significant byte to sp-1, then the least significant to the sp-2.
- Push instruction does not affect any flags.
- **Example:-**
- PUSH CX ; Decrements SP by 2, copy content of CX to the stack (figure shows execution of this instruction)
- PUSH DS ; Decrement SP by 2 and copy DS to stack

POP instruction:

- The POP instruction copies a word from the stack location pointed by the stack pointer to the destination. The destination can be a General purpose register, a segment register or a memory location. Here after the content is copied the stack pointer is automatically incremented by two.
- The execution pattern is similar to that of the PUSH instruction.
- **Example:**
- POP CX; Copy a word from the top of the stack to CX and increment SP by 2.

IN & OUT instructions

The IN instruction will copy data from a port to the accumulator. If 8 bit is read the data will go to AL and if 16 bit then to AX. Similarly OUT instruction is used to copy data from accumulator to an output port.

Both IN and OUT instructions can be done using direct and indirect addressing modes.

- **Example:**

• IN AL, 0F8H	;	Copy a byte from the port 0F8H to AL
• MOV DX, 30F8H	;	Copy port address in DX
• IN AL, DX	;	Move 8 bit data from 30F8H port
• IN AX, DX	;	Move 16 bit data from 30F8H port
• OUT 047H, AL	;	Copy contents of AL to 8 bit port 047H
• MOV DX, 330F8H	;	Copy port address in DX
• OUT DX, AL	;	Move 8 bit data to the 30F8H port
• OUT DX, AX	;	Move 16 bit data to the 30F8H port

XCHG instruction

- The XCHG instruction exchanges contents of the destination and source. Here destination and source can be register and register or register and memory location, but XCHG cannot interchange the value of 2 memory locations.
- XCHG Destination, Source
  - **Example:**

• XCHG BX, CX	;	exchange word in CX with the word in BX
	;	exchange byte in CL with the byte in AL
	;	Here physical address, which is DS +SUM+ [BX]. The content at physical address and the content of AX are interchanged

2. **Arithmetic and Logical instructions:**
   All the instructions performing arithmetic, logical, increment, decrement, compare and ASCII instructions belong to this category.

ADD instruction:

- Add instruction is used to add the current contents of destination with that of source and store the result in destination. Here we can use register and/or memory locations.
- AF, CF, OF, PF, SF, and ZF flags are affected.

- ADD Destination, Source
- **Example:**
  - ADD AL, 0FH ; Add the immediate content, 0FH to the content of AL and store the result in AL
  - ADD AX, BX ;   AX <= AX+BX
  - ADD AX,0100H – IMMEDIATE
  - ADD AX,BX – REGISTER
  - ADD AX,[SI] – REGISTER INDIRECT OR INDEXED
  - ADD AX, [5000H] – DIRECT
  - ADD [5000H], 0100H – IMMEDIATE
  - ADD 0100H – DESTINATION AX (IMPLICT)

ADC: ADD WITH CARRY

This instruction performs the same operation as ADD instruction, but adds the carry flag bit (which may be set as a result of the previous calculation) to the result. All the condition code flags are affected by this instruction. The examples of this instruction along with the modes are as follows:

Example:

- ADC AX,BX – REGISTER
- ADC AX,[SI] – REGISTER INDIRECT OR INDEXED
- ADC AX, [5000H] – DIRECT
- ADC [5000H], 0100H – IMMEDIATE
- ADC 0100H – IMMEDIATE (AX IMPLICT)

SUB instruction:

- SUB instruction is used to subtract the current contents of destination with that of source and store the result in destination. Here we can use register and/or memory locations. AF, CF, OF, PF, SF, and ZF flags are affected
- SUB Destination, Source

- **Example:**
  - SUB AL, 0FH ; subtract the immediate content, 0FH from the content of AL and store the result in AL
  - SUB AX, BX   ; AX <= AX-BX
  - SUB AX,0100H – **IMMEDIATE (DESTINATION AX)**
  - SUB AX,BX – **REGISTER**
  - SUB AX,[5000H] – **DIRECT**
  - SUB [5000H], 0100H – **IMMEDIATE**

SBB: SUBTRACT WITH BORROW:

- To subtract with borrow instruction subtracts the source operand and the borrow flag (CF) which may reflect the result of the previous calculations, from the destination operand. Subtraction with borrow, here means subtracting 1 from the subtraction obtained by SUB, if carry (borrow) flag is set.

- The result is stored in the destination operand. All the flags are affected (condition code) by this instruction. The examples of this instruction are as follows:

- **Example:**

- SBB AX, 0100H – **IMMEDIATE (DESTINATION AX)**
- SBB AX, BX – **REGISTER**
- SBB AX,[5000H] – **DIRECT**
- SBB [5000H], 0100H – **IMMEDIATE**

CMP: COMPARE:

- The instruction compares the source operand, which may be a register or an immediate data or a memory location, with a destination operand that may be a register or a memory location.
- For comparison, it subtracts the source operand from the destination operand but does not store the result anywhere. The flags are affected depending upon the result of the subtraction.
- If both of the operands are equal, zero flag is set. If the source operand is greater than the destination operand, carry flag is set or else, carry flag is reset. The examples of this instruction are as follows:

- **Example:**
- CMP BX, 0100H – IMMEDIATE
- CMP AX, 0100H – IMMEDIATE
- CMP [5000H], 0100H – DIRECT
- CMP BX,[SI] – REGISTER INDIRECT OR INDEXED

- CMP BX, CX – REGISTER

INC & DEC instructions:

1. INC and DEC instructions are used to increment and decrement the content of the specified destination by one. AF, CF, OF, PF, SF, and ZF flags are affected.
2. **Example:**

INC AL	;	AL←AL + 1
INC AX	;	AX←AX + 1

DEC AL	;	AL← AL – 1
DEC AX	;	AX←AX – 1

AND instruction:

- This instruction logically ANDs each bit of the source byte/word with the corresponding bit in the destination and stores the result in destination. The source can be an immediate number, register or memory location, register can be a register or memory location.

- The CF and OF flags are both made zero, PF, ZF, SF are affected by the operation and AF is undefined.
- AND Destination, Source

- **Example:**

- **AND BL, AL;** suppose BL=1000 0110 and AL = 1100 1010 then after the operation BL would be BL= 1000 0010.
- **AND CX, AX;** CX ← CX AND AX
- **AND CL, 08;** CL← CL AND (0000 1000)

OR instruction:

- This instruction logically ORs each bit of the source byte/word with the corresponding bit in the destination and stores the result in destination. The source can be an immediate number, register or memory location, register can be a register or memory location.

- The CF and OF flags are both made zero, PF, ZF, SF are affected by the operation and AF is undefined.

- OR Destination, Source
- **Example:**
- **OR BL, AL** ; suppose BL=1000 0110 and AL = 1100 1010 then after the operation BL would be BL= 1100 1110.
- OR CX, AX ; CX←AX AND AX
- OR CL, 08 ; CL←CL AND (0000 1000)

NOT instruction:

- The NOT instruction complements (inverts) the contents of an operand register or a memory location, bit by bit. The examples are as follows:
  1. **Example:**
  2. NOT AX (BEFORE AX= (1011)2= (B) 16 AFTER EXECUTION AX= (0100)2= (4)16).
  3. NOT [5000H]

XOR instruction:

- The XOR operation is again carried out in a similar way to the AND and OR operation. The constraints on the operands are also similar. The XOR operation gives a high output, when the 2 input bits are dissimilar. Otherwise, the output is zero. The example instructions are as follows:

- **Example:**

- XOR AX, 0098H
- XOR AX, BX
- XOR AX, [5000H]

3. Shift / Rotate Instructions:

1) Shift instructions move the binary data to the left or right by shifting them within the register or memory location. They also can perform multiplication of powers of 2^{+n} and division of powers of 2^{-n}.

2) There are two type of shifts logical shifting and arithmetic shifting, later is used with signed numbers while former with unsigned.

SHL/SAL instruction:

- Both the instruction shifts each bit to left, and places the MSB in CF and LSB is made 0. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.

- All flags are affected.
- SAL/SHL destination, count

→Example:

MOV BL, B7H	;	BL is made B7H
SAL BL, 1	;	Shift the content of BL register one place to left.

Before Execution

CY	**B7**	**B6**	**B5**	**B4**	**B3**	**B2**	**B1**	**B0**
0	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**

After Execution,

CY	**B7**	**B6**	**B5**	**B4**	**B3**	**B2**	**B1**	**B0**
1	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**

SHR instruction:

- This instruction shifts each bit in the specified destination to the right and 0 is stored in the MSB position. The LSB is shifted into the carry flag. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.

- All flags are affected
- **Before execution,**

B7	**B6**	**B5**	**B4**	**B3**	**B2**	**B1**	**B0**	
CY 1	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**

- **After execution,**

B7	**B6**	**B5**	**B4**	**B3**	**B2**	**B1**	**B0**	**CY**
0	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**

ROL instruction:

- This instruction rotates all the bits in a specified byte or word to the left some number of bit positions. MSB is placed as a new LSB and a new CF. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.
- All flags are affected
- ROL destination, count
  →**Example:**

```
MOV BL, B7H    ;    BL is made B7H

ROL BL, 1      ;    rotates the content of BL register one place to the left.
```

Before Execution:

CY	**B7**	**B6**	**B5**	**B4**	**B3**	**B2**	**B1**	**B0**
0	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**

After the execution,

CY	**B7**	**B6**	**B5**	**B4**	**B3**	**B2**	**B1**	**B0**
1	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**1**

ROR instruction:

- This instruction rotates all the bits in a specified byte or word to the right some number of bit positions. LSB is placed as a new MSB and a new CF. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.

- All flags are affected.
- ROR destination, count
- **Example:**

MOV BL, B7H ; BL is made B7H

ROR BL, 1 ; shift the content of BL register one place to the right.

- **<u>Before execution</u>,**

B7 B6 B5 B4 B3 B2 B1 B0 CY

1 0 1 1 0 1 1 1 0

- **<u>After execution</u>,**

B7 B6 B5 B4 B3 B2 B1 B0 CY

1 1 0 1 1 0 1 1 1

RCR instruction

- This instruction rotates all the bits in a specified byte or word to the right some number of bit positions along with the carry flag. LSB is placed in a new CF and previous carry is placed in the new MSB. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.
- All flags are affected
- **General Format:** RCR destination, count
- **Example:**

MOV BL, B7H ; BL is made B7H

RCR BL, 1 ; shift the content of BL register one place to the right.

- **<u>Before execution</u>,**

B7 B6 B5 B4 B3 B2 B1
B0 CY 1 0 1 1 0 1 1
1 0

- **<u>After execution</u>,**

B7 B6 B5 B4 B3 B2 B1

B0 CY 0 1 0 1 1 0 1
1 1

4. PROGRAM CONTROL TRANSFER INSTRUCTIONS:

- These instructions transfer control of execution to the specified address. All the call, jump, interrupt and return instruction belong to this class.
- There are 2 types of such instructions.

1. Unconditional transfer instructions – CALL, RET, JMP
2. Conditional transfer instructions – J condition

CALL instruction:

- The CALL instruction is used to transfer execution to a subprogram or procedure. There are two types of CALL instructions, near and far.
- A **near CALL** is a call to a procedure which is in the same code segment as the CALL instruction. 8086 when encountered a near call, it decrements the SP by 2 and copies the offset of the next instruction after the CALL on the stack. It loads the IP with the offset of the procedure then to start the execution of the procedure.
- A **far CALL** is the call to a procedure residing in a different segment. Here value of CS and offset of the next instruction both are backed up in the stack. And then branches to the procedure by changing the content of CS with the segment base containing procedure and IP with the offset of the first instruction of the procedure.
- **Example:**

  Near call

  CALL PRO ; PRO is the name of the procedure

  CALL CX ; Here CX contains the offset of the first instruction
  of the procedure, that is replaces the content IP with the content of CX

Far call

CALL DWORD PTR [8X]; New values for CS and IP are fetched from four Memory locations in the DS. The new value for CS is fetched from [8X] and [8X+1], the new IP is fetched from [8X+2] and [8X+3].

RET instruction:

- RET instruction will return execution from a procedure to the next instruction after the CALL instruction in the calling program. If it was a near call, then IP is replaced with the value at the top of the stack, if it had been a far call, then another POP of the stack is required. This second popped data from the stack is put in the CS, thus resuming the execution of the calling program.

- RET instruction does not affect any flags.

JMP INSTRUCTION:

- This is also called as unconditional jump instruction, because the processor jumps to the specified location rather than the instruction after the JMP instruction. Jumps can be **short jumps** when the target address is in the same segment as the JMP instruction or **far jumps** when it is in a different segment.

Conditional Jump (J cond)

- Conditional jumps are always short jumps in 8086. Here jump is done only if the condition specified is true/false. If the condition is not satisfied, then the execution proceeds in the normal way.

Iteration control instructions:

- These instructions are used to execute a series of instructions some number of times. The number is specified in the CX register, which will be automatically decremented in course of iteration. But here the destination address for the jump must be in the range of -128 to 127 bytes.

Example:

LOOP : loop through the set of instructions until CX is 0
LOOPE/LOOPZ : here the set of instructions are repeated until CX=0 or ZF=0 LOOPNE/LOOPNZ: here repeated until CX=0 or ZF=1

5. MACHINE CONTROL INSTRUCTIONS:

- These instructions control the machine status. NOP, HLT, WAIT and LOCK instructions belong to this class.

HLT instruction

- The HLT instruction will cause the 8086 microprocessor to fetching and executing instructions.
- The 8086 will enter a halt state. The processor gets out of this Halt signal upon an interrupt signal in INTR pin/NMI pin or a reset signal on RESET input

WAIT instruction

- When this instruction is executed, the 8086 enters into an idle state. This idle state is continued till a high is received on the TEST input pin or a valid interrupt signal is received. Wait affects no flags. It generally is used to synchronize the 8086 with a peripheral device(s).

ESC instruction

- This instruction is used to pass instruction to a coprocessor like 8087. There is a 6 bit instruction for the coprocessor embedded in the ESC instruction. In most cases the 8086 treats ESC and a NOP, but in some cases the 8086 will access data items in memory for the coprocessor

LOCK instruction

- In multiprocessor environments, the different microprocessors share a system bus, which is needed to access external devices like disks. LOCK Instruction is given as prefix in the case when a processor needs exclusive access of the system bus for a particular instruction.
- It affects no flags.
- **LOCK XCHG SEMAPHORE, AL** : The XCHG instruction requires two Bus accesses.
- The lock prefix prevents another processor from taking control of the system bus between the 2 accesses

NOP instruction

- At the end of NOP instruction, no operation is done other than the fetching and decoding of the instruction. It takes 3 clock cycles. NOP is used to fill in time delays or to provide space for instructions while trouble shooting. NOP affects no flags.

6. FLAG MANIPULATION INSTRUCTIONS:

- All the instructions which directly affect the flag register come under this group of instructions. Instructions like CLD, STD, CLI, STI etc.., belong to this category of instructions.
- **STC instruction**
  This instruction sets the carry flag. It does not affect any other flag.
- **CLC instruction**
  This instruction resets the carry flag to zero. CLC does not affect any other flag.

- **CMC instruction**
  This instruction complements the carry flag. CMC does not affect any other flag.

- **STD instruction**
  This instruction is used to set the direction flag to one so that SI and/or DI can be decremented automatically after execution of string instruction. STD does not affect any other flag.

- **CLD instruction**
  This instruction is used to reset the direction flag to zero so that SI and/or DI can be incremented automatically after execution of string instruction. CLD does not affect any other flag.

- **STI instruction**
  This instruction sets the interrupt flag to 1. This enables INTR interrupt of the 8086. STI does not affect any other flag.

- **CLI instruction**
  This instruction resets the interrupt flag to 0. Due to this the 8086 will not respond to an interrupt signal on its INTR input. CLI does not affect any other flag.

7. STRING MANIPULATION INSTRUCTIONS:

These instructions involve various string manipulation operations like Load, move, scan, compare, store etc.

- **MOVS/MOVSB/MOVSW**
  - These instructions copy a word or byte from a location in the data segment to a location in the extra segment. The offset of the source is in SI and that of destination is in DI. For multiple word/byte transfers the count is stored in the CX register.
  - When direction flag is 0, SI and DI are incremented and when it is 1, SI and DI are decremented.
  - MOVS affect no flags. MOVSB is used for byte sized movements while MOVSW is for word sized.

Example:

```
CLD                ;          clear the direction flag to auto increment SI
and DI MOV AX, 0000H;

MOV DS, AX        ;           initialize data segment
register to 0 MOV ES, AX  ;           initialize extra
segment register to 0 MOV SI, 2000H   ;           Load the
offset of the string1 in SI MOV DI, 2400H  ;     Load the
offset of the string2 in DI MOV CX, 04H            ;
                   load length of the string in CX
```

```
REP MOVSB        ;          decrement CX and MOVSB until CX will be 0
```

- **REP/REPE/REP2/REPNE/REPNZ**
  - REP is used with string instruction; it repeats an instruction until the specified condition becomes false.

Example:	**Comments**
REP	CX=0
REPE/REPZ	CX=0 OR ZF=0
REPNE/REPNZ	CX=0 OR ZF=1

- **LODS/LODSB/LODSW**
  - This instruction copies a byte from a string location pointed to by SI to AL or a word from a string location pointed to by SI to AX.LODS does not affect any flags. LODSB copies byte and LODSW copies word.

  Example:

```
CLD                          ; clear direction flag to auto increment SI MOVSI,
OFFSET S_STRING              ; point SI at string
LODS S_STRING                ;
```

- **STOS/STOSB/STOSW**
- The STOS instruction is used to store a byte/word contained in AL/AX to the offset contained in the DI register. STOS does not affect any flags. After copying the content DI is automatically incremented or decremented, based on the value of direction flag.

Example:

- MOV DL, OFFSET D_STRING;   assign DI with destination address.
- STOS D_STRING   ;   assembler uses string name to determine byte or Word, if byte then AL is used and if of word size, AX is used.

5. CMPS/CMPSB/CMPSW

- CMPS is used to compare the strings, byte wise or word wise. The comparison is affected by subtraction of content pointed by DI from that pointed by SI. The AF, CF, OF, PF, SF and ZF flags are affected by this instruction, but neither operand is affected.

Example:		Comments
MOV SI, OFFSET STRING_A	;	Point first string
MOV DI, OFFSET STRING_B	;	Point second string
MOV CX, 0AH	;	Set the counter as 0AH
CLD	;	Clear direction flag to auto increment
REPE CMPSB	;	Repeatedly compare till unequal or counter =0

5.13 ASSEMBLER DIRECTIVES AND OPERATOR:

- There are some instructions in the assembly language program which are not a part of processor instruction set. These instructions are instructions to the assembler, linker and loader. These are referred to as pseudo-operations or as assembler directives. The assembler directives enable us to control the way in which a program assembles and lists. They act during the assembly of a program and do not generate any executable machine code.

- There are many specialized assembler directives. Let us see the commonly used assembler directive in 8086 assembly language programming.

<u>ASSUME:</u>

- It is used to tell the name of the logical segment the assembler to use for a specified segment.

- E.g.: ASSUME CS: CODE tells that the instructions for a program are in a logical segment named CODE.

<u>DB -Define Byte:</u>

- The DB directive is used to reserve byte or bytes of memory locations in the available memory. While preparing the EXE file, this directive directs the assembler to allocate the specified number of memory bytes to the said data type that may be a constant, variable, string, etc. Another option of this directive also initializes the reserved memory bytes with the ASCII codes of the characters specified as a string. The following examples show how the DB directive is used for different purposes.
  - ✓ **RANKS DB 01H, 02H, 03H, 04H**
    This statement directs the assembler to reserve four memory locations for a list named RANKS and initialize them with the above specified four values.

- ✓ **MESSAGE DB „GOOD MORNING"**
  This makes the assembler reserve the number of bytes of memory equal to the number of characters in the string named MESSAGE and initializes those locations by the ASCII equivalent of these characters.

- ✓ **VALUE DB 50H**
  This statement directs the assembler to reserve 50H memory bytes and leave them uninitialized for the variable named VALUE.

DD:

- Define Double word - used to declare a double word type variable or to reserve memory locations that can be accessed as double word.

  E.g.: ARRAY _POINTER DD 25629261H declares a double word named ARRAY_POINTER.

DQ -Define Quad word:

- This directive is used to direct the assembler to reserve 4 words (8 bytes) of memory for the specified variable and may initialize it with the specified values.

DT -Define Ten Bytes:

- The DT directive directs the assembler to define the specified variable requiring 10-bytes for its storage and initialize the 10-bytes with the specified values. The directive may be used in case of variables facing heavy numerical calculations, generally processed by numerical processors.

DW -Define Word:

- The DW directives serves the same purposes as the DB directive, but it now makes the assembler reserve the number of memory words (16-bit) instead of bytes. Some examples are given to explain this directive.

- ✓ **WORDS DW 1234H, 4567H, 78ABH, 045CH**
  This makes the assembler reserve four words in memory (8 bytes), and initialize the words with the specified values in the statements. During initialization, the lower bytes are stored at the lower memory addresses, while the upper bytes are stored at the higher addresses.

- ✓ **NUMBER1 DW 1245H**
  This makes the assembler reserve one word in memory.

END-End of Program:

- The END directive marks the end of an assembly language program. When the assembler comes across this END directive, it ignores the source lines available later on. Hence, it should be ensured that the END statement should be the last statement in the file and should not appear in between. Also, no useful program statement should lie in the file, after the END statement.

ENDP:

- End Procedure - Used along with the name of the procedure to indicate the end of a procedure.
- E.g.: SQUARE_ROOT PROC: start of procedure
  SQUARE_ROOT ENDP: End of procedure

ENDS-End of Segment:

- This directive marks the end of a logical segment. The logical segments are assigned with the names using the ASSUME directive. The names appear with the ENDS directive as prefixes to mark the end of those particular segments. Whatever are the contents of the segments, they should appear in the program before ENDS. Any statement appearing after ENDS will be neglected from the segment. The structure shown below explains the fact more clearly.

EQU:

- Equate - Used to give a name to some value or symbol. Each time the assembler finds the given name in the program, it will replace the name with the vale.
- E.g.: CORRECTION_FACTOR EQU 03H MOV
  AL, CORRECTION_FACTOR

EVEN:

- Tells the assembler to increment the location counter to the next even address if it is not already at an even address.
- Used because the processor can read even addressed data in one clock cycle

EXTRN:

- Tells the assembler that the names or labels following the directive are in some other assembly module.
- For example if a procedure in a program module assembled at a different time from that which contains the CALL instruction ,this directive is used to tell the assembler that the procedure is external

GLOBAL:

- Can be used in place of a PUBLIC directive or in place of an EXTRN directive.
- It is used to make a symbol defined in one module available to other modules.
- E.g.: GLOBAL DIVISOR makes the variable DIVISOR public so that it can be accessed from other modules.

GROUP:

- Used to tell the assembler to group the logical statements named after the directive into one logical group segment, allowing the contents of all the segments to be accessed from the same group segment base.

- E.g.: SMALL_SYSTEM GROUP CODE, DATA, STACK_SEG

INCLUDE:
- Used to tell the assembler to insert a block of source code from the named file into the current source module.
- This will shorten the source code.

LABEL:
- Used to give a name to the current value in the location counter.
- This directive is followed by a term that specifies the type you want associated with that name.
- E.g: ENTRY_POINT LABEL FAR

- NEXT: MOV AL, BL

NAME:
- Used to give a specific name to each assembly module when programs consisting of several modules are written.
- E.g.: NAME PC_BOARD

OFFSET:
- Used to determine the offset or displacement of a named data item or procedure from the start of the segment which contains it.
- E.g.: MOV BX, OFFSET PRICES

ORG:
- The location counter is set to 0000 when the assembler starts reading a segment. The ORG directive allows setting a desired value at any point in theprogram.
- E.g.: ORG 2000H

PROC:
- Used to identify the start of a procedure.
- E.g.:SMART_DIVIDE PROC FAR identifies the start of a procedure named SMART_DIVIDE and tells the assembler that the procedure is far

PTR:
- Used to assign a specific type to a variable or to a label.
- E.g.: NC BYTE PTR[BX]tells the assembler that we want to increment the byte pointed to by BX

PUBLIC:
- Used to tell the assembler that a specified name or label will be accessed from other modules.

- E.g.: PUBLIC DIVISOR, DIVIDEND makes the two variables DIVISOR and DIVIDEND available to other assembly modules.

SEGMENT:

- Used to indicate the start of a logical segment.
- E.g.: CODE SEGMENT indicates to the assembler the start of a logical segment called CODE

SHORT:

- Used to tell the assembler that only a 1 byte displacement is needed to code a jump instruction.
- E.g.: JMP SHORT NEARBY_LABEL

TYPE:

- Used to tell the assembler to determine the type of a specified variable.
- E.g.: ADD BX, TYPE WORD_ARRAY is used where we want to increment BX to point to the next word in an array of words.

5.14 SIMPLE ASSEMBLY LANGUAGE PROGRAMMING USING 8086 INSTRUCTIONS:

PROGRAM-1: (ADDITION)

EFFECTIVE ADDRESS	MNEMONIC CODES	LABLE	MNEMONICS	OPERANDS	COMMENTS
2000	8B,06,00,17		MOV	AX,[1700]	Move the contents of 1700 in register AX
2004	8B,1E,02,17		MOV	BX, [1702]	Move the contents of 1702 in register BX
2008	01,D8		ADD	AX,BX	Data of AX and BX are added and result stored in AX
200A	CC		INT 3		Interrupt program

PROGRAM-2: (SUBTRACTION)

EFFECTIVE ADDRESS	MNEMONIC CODES	LABLE	MNEMONICS	OPERANDS	COMMENTS
2000	8B,06,00,17		MOV	AX,[1700]	Move the contents of 1700 in register AX
2004	8B,1E,02,17		MOV	BX, [1702]	Move the contents of 1702 in register BX
2008	29,D8		SUB	AX,BX	Data of AX and BX are added and result stored in AX
200A	CC		INT 3		Interrupt program

PROGRAM-3: (MULTIPLICATION)

EFFECTIVE ADDRESS	OPCODES	MNEMONICS	OPERANDS	COMMENTS
1100	BE 00 15	MOV	SI,1500	Load 1500 into SI
1103	AD	LOD	SW	Load the multiplicand value
1104	89 C3	MOV	BX, AX	Load AX value into BX
1106	AD	LOD	SW	Load the multiplier value
1107	F7 E3	MUL	BX	Multiply two data
1109	BF 0 5 15	MOV	DI, 1520	Load 1520 address into DI
110C	89 05	MOV	[DI], AX	Store AX value into DI
110E	47	INC DI		
110F	47	INC	DI	Increment the DI
1110	89 15	MOV	[DI], BX	Store BX value into DI
1112	CC	INT 3		Break point

PROGRAM-4: (DIVISION)

EFFECTIVE ADDRESS	OPCODES	MNEMONICS	OPERANDS	COMMENTS
1100	BA 00 00	MOV	DX, 0000	Clear DX registers
1103	B8 83 00	MOV	AX, 0083	Load the dividend in AX
1106	B9 00 02	MOV	BX, 02	Load the divisor value in BX
1109	F7 F1	DIV	BX	Divide the two data’s
110B	BF 20 15	MOV	DI, 1520	Load 1520 address into DI
110E	88 05	MOV	[DI], AL	Load AL value into DI
1110	47	INC	DI	Increment DI

1111	88 25	MOV	[DI], AH	Load AH value into DI
1113	47	INC	DI	Increment DI
1114	89 15	MOV	[DI], DX	Load DX value into DI
1116	CC	INT3		Break point

PROGRAM-5: (LARGEST NUMBER IN DATA ARRAY)

EFFECTIVE ADDRESS	MNEMONIC CODES	LABLE	MNEMONICS	OPERANDS	COMMENTS
0101	B8, 00, 00		MOV	AX, 0000	;Initial value for comparison
0104	BE, 00, 02		MOV	SI, 0200	;memory address in SI
0107	8B, 0C		MOV	CX, [SI]	;count in CX
0109	46	BACK	INC	SI	;increment SI
010A	46		INC	SI	;increment SI
010B	3B, 04		CMP	AX, [SI]	;compare previous largest number with next number
010D	73, 02		JAE	GO	;Jump if number in AX is greater i.e. CF = 0
010F	8B, 04		MOV	AX, [SI]	;save next larger number in AX
0111	E2, F6	GO	LOOP	BACK	;jump to BACK until CX become zero
0113	A3, 51, 02		MOV	[0251], AX	;store largest number in memory
0116	CC		INT3		;interrupt program

PROGRAM-6: (SMALLEST NUMBER IN DATA ARRAY)

EFFECTIVE ADDRESS	MNEMONICS CODES	LABEL	MNEMONICS	OPERANDS	COMMENTS
0101	B8,FF,FF		MOV	AX,FFFF	Initial value for comparison.
0104	BE,00,02		MOV	SI,0200	Memory address in SI.
0107	8B,0C		MOV	CX,[SI]	Count in CX
0109	46	BACK	INC	SI	Increment SI
010A	46		INC	SI	Increment SI
010B	3B,04		CMP	AX,[SI]	Compare previous smallest with next number
010D	72,02		JB	GO	Jump if number in AX is smaller i.e. CF=1
010F	8B,04		MOV	AX,[SI]	Save next smaller
0111	E2,F6	GO	LOOP	BACK	Jump to back until CX becomes zero.
0113	A3,51,02		MOV	[0251],AX	Store smallest number in memory
0116	CC		INT 3		Interrupt program.

Branch Instructions:

- These instructions transfer control of execution to the specified address. All the call, jump, interrupt and return instruction belong to this class.

Loop instructions:

- These instructions can be used to implement unconditional and conditional loops. The LOOP, LOOP NZ, LOOP Z instructions belong to this category.

Machine control instructions:

- These instructions control the machine status. NOP, HLT, WAIT and LOCK instructions belong to this class.

Flag manipulation instructions:

- All the instructions which directly affect the flag register come under this group of instructions. Instructions like CLD, STD, CLI, STI etc.., belong to this category of instructions.

Shift and Rotate instructions:

- These instructions involve the bit wise shifting or rotation in either direction with or without a count in CX.

String manipulation instructions:

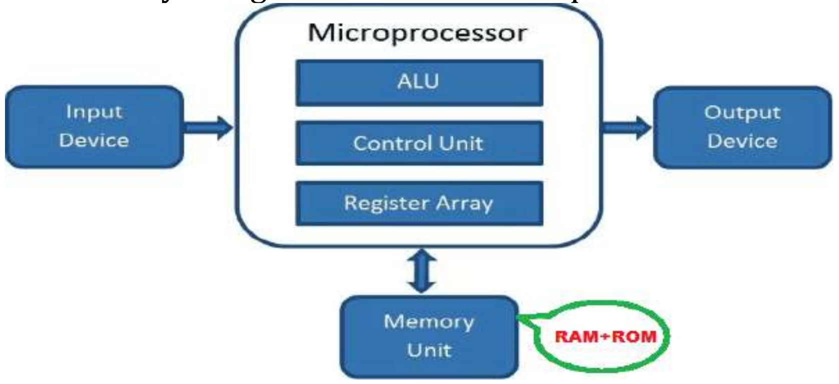
- These instructions involve various string manipulation operations like Load, move, scan, compare, store etc..,

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

6.1 DISTINGUISH BETWEEN MICROPROCESSOR & MICROCONTROLLER

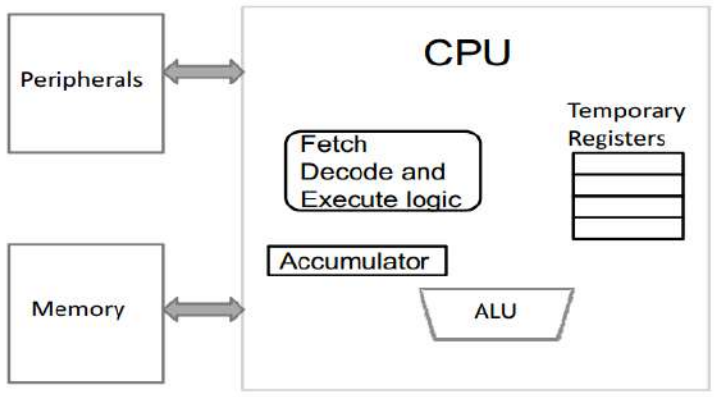
MICROPROCESSOR:

- A Microprocessor is a multipurpose, Programmable clock driven, register based electronic device,
- That read binary instruction from a storage device called memory, accepts binary data as input and processes data according to those instructions and provides results as outputs.
- Microprocessor is clock driven semiconductor device which for is manufactured by using LSI and VLSI technique.



MICROCONTROLLER:

- Microcontroller is like a mini computer with a CPU along with RAM, ROM, serial ports, timers, and IO peripherals all embedded on a single chip.
- It's designed to perform application specific tasks that require a certain degree of control such as a TV remote, LED display panel, smart watches, vehicles, traffic light control, temperature control, etc.
- It's a high-end device with a microprocessor, memory, and input/output ports all on a single chip.
- It's the brains of a computer system which contains enough circuitry to perform specific functions without external memory.
- Since it lacks external components, the power consumption is less which makes it ideal for devices running on batteries.
- Simple speaking, a microcontroller is complete computer system with less external hardware.



DIFFERENCE BETWEEN MICROPROCESSOR AND MICROCONTROLLER:

MICROPROCESSOR	MICROCONTROLLER
Microprocessor contains ALU, General purpose registers, stack pointer, program counter, clock timing circuit, interrupt circuit	Microcontroller contains the circuitry of microprocessor, and in addition it has built in ROM, RAM, I/O Devices, Timers/Counters etc.
It has many instructions to move data between memory and CPU	It has few instructions to move data between memory and CPU
Few bit handling instruction	It has many bit handling instructions
Less number of pins are multifunctional	More number of pins are multifunctional
Single memory map for data and code (program)	Separate memory map for data and code (program)
Access time for memory and IO are more	Less access time for built in memory and IO.
Microprocessor based system requires additional hardware	It requires less additional hardware's
More flexible in the design point of view	Less flexible since the additional circuits which is residing inside the microcontroller is fixed for a particular microcontroller
Large number of instructions with flexible addressing modes	Limited number of instruction with few addressing modes

6.2 8 BIT & 16 BIT MICROCONTROLLER:

8 bit Microcontroller:

- 8 bit microcontroller is type of microcontroller having all traits of microcontroller and its information gadgets are largely 8 bits big.
- 8 bits big means your CPU can use 8 bit information bus or pipe and can entry the similar dimension information by a single machine instruction.
- For every cycle of instruction its fluctuate is zero to 255. It requires 20mA current to work. Intel 8008 was the first model having 8 bit micro-controller.

16 bit Microcontroller:

- 16 bit microcontroller is additional superior than 8 bit microcontroller.
- It is additional right and precise in performing mathematical and technical duties.

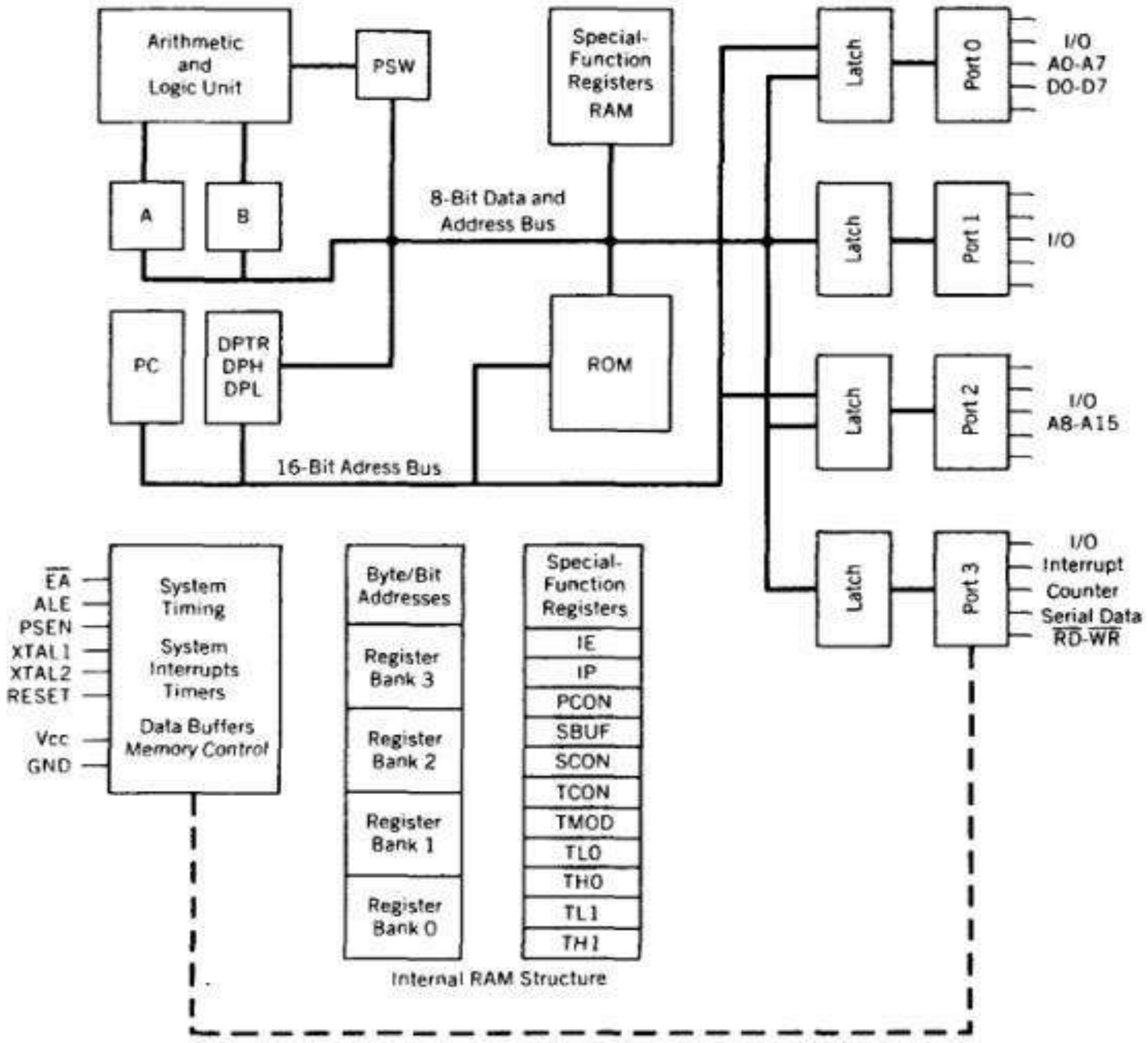
- Unlike 8 bit microcontroller it makes use of 16 bits information bus or pipe for a single instruction.
- For every cycle of instruction its bit fluctuate is extended from zero to 65,535. As 16 bit controller is 2 time better than 8 bit controller, it would probably work on two 16 bit numbers. It requires 10mA current to hold out.

6.3 CISC AND RISC CPU ARCHITECTURES:

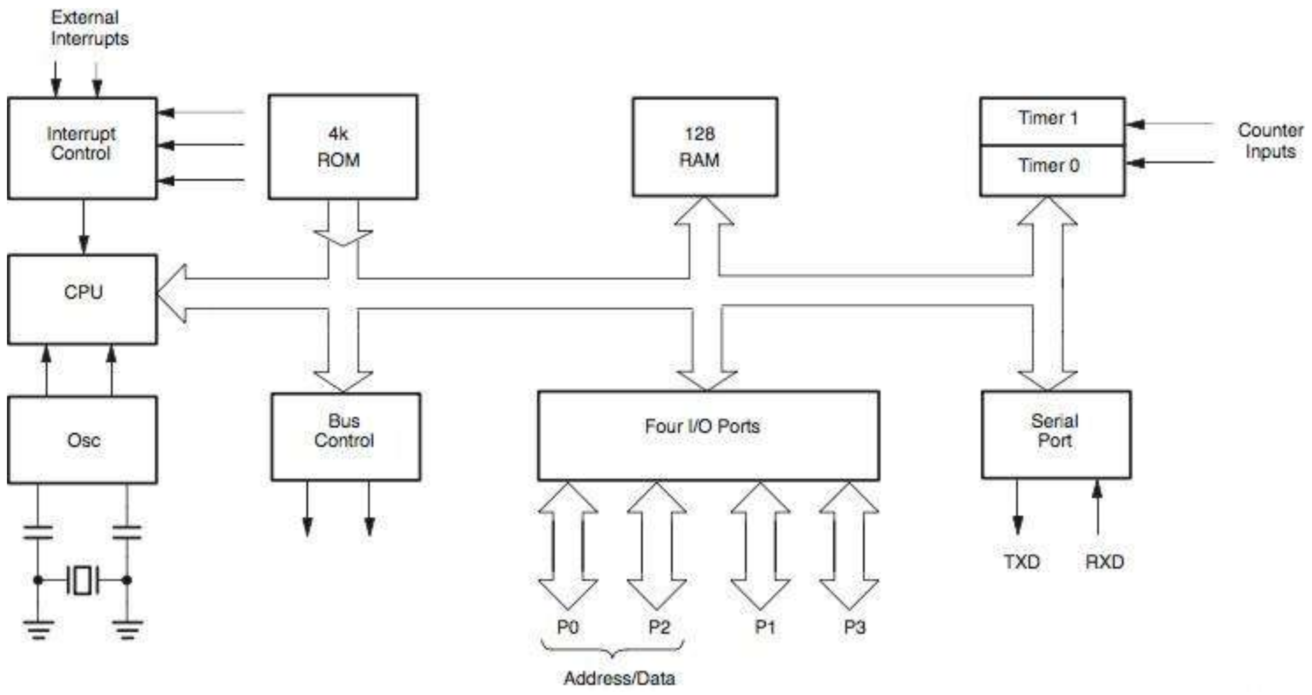
- Microcontrollers with small instruction set are called reduced instruction set computer (RISC) machines and those with complex instruction set are called complex instruction set computer (CISC).
- Intel 8051 is an example of CISC machine whereas microchip PIC 18F87X is an example of RISC machine.

RISC	CISC
Instruction takes one or two cycles	Instruction takes multiple cycles
Only load/store instructions are used to access memory	In additions to load and store instructions, memory access is possible with other instructions also.
Instructions executed by hardware	Instructions executed by the micro program
Fixed format instruction	Variable format instructions
Few addressing modes	Many addressing modes
Few instructions	Complex instruction set
Most of the have multiple register banks	Single register bank
Highly pipelined	Less pipelined
Complexity is in the compiler	Complexity in the microprogram

6.4 ARCHITECTURE OF 8051 MICROCONTROLLER:



Alternate diagram …



✓ **8051** is a microcontroller. This means it has an internal processor, internal memory and an I/O section. The architecture of 8051 is thus divided into three main sections:

- The CPU
- Internal Memory
- I/O components.

CPU:

- 8051 has an 8 bit CPU.
- This is where all 8-bot arithmetic and logic operations are performed.
- It has the following components.

ALU – ARITHMETIC LOGIC UNIT:

- It performs 8-bit arithmetic and logic operations.
- It can also perform some bit operations.

- **Example:**
  ADD A, R0 ; Adds contents of A register and R0 register and stores the result in A register.

ANL A, R0 ; Logically ANDs contents of A register and R0 register and stores the result in A register.

CPL P0.0 ; Complements the value of P0.0 pin.

<u>A – REGISTER (ACCUMULATOR):</u>

- It is an 8-bit register.
- In most arithmetic and logic operations, A register hold the first operand and also gets the result of the operation.
- Moreover, it is the only register to be used for data transfers to and from external memory.

- **Example:**

  ADD A, R1 ; Adds contents of A register and R1 register and stores the result in A register.

  MOVX A, @DPTR ; A gets the data from External RAM location pointed by DPTR

<u>B – REGISTER:</u>

- It is an 8-bit register.
- It is dedicated for Multiplication and Division.
- It can also be used in other operations.

- **Example:**

  MUL AB; Multiplies contents of A and B registers. Stores 16-bit result in B and A registers.

  DIV AB; Divides contents of A by those of B. Stores quotient in A and remainder in B.

<u>PC – PROGRAM COUNTER</u>

- It is a 16-bit register.
- It holds address of the next instruction in program memory (ROM).
- PC gets automatically incremented as soon as any instruction is fetched.
- That's what makes the program move ahead in a sequential manner.

- In the case of a branch, a new address is loaded into PC.

<u>DPTR – DATA POINTER</u>

- It is a 16-bit register.
- It holds address data in data memory (RAM).
- DPTR is divided into two registers DPH (higher byte) and DPL (lower byte).
- It is typically used by the programmer to transfer data from External RAM.
- It can also be used as a pointer to a look up table in ROM, using Indexed addressing mode.
- **Example:**

MOVX A, @DPTR ; A gets the data from External RAM location pointed by DPTR

MOVC A, @A+DPTR ; A gets the data from ROM location pointed by A + DPTR

<u>SP – STACK POINTER</u>

- It is an 8-bit register.
- It contains address of the top of stack. The Stack is present in the Internal RAM.
- Internal RAM has 8-bit addresses from 00H… 7FH. Hence SP is an 8-bit register.
- It is affected during Push and Pop operations. During a Push, SP gets incremented.
- During a Pop, SP gets decremented.

<u>PSW – PROGRAM STATUS WORD</u>

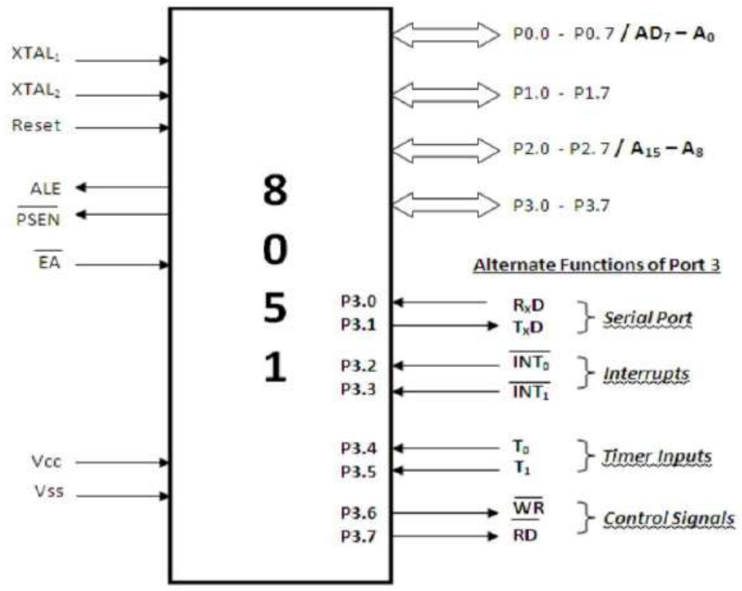
- It is an 8-bit register.
- It is also called the “Flag register”, as it mainly contains the status flags. These flags indicate status of the current result.
- They are changed by the ALU after every arithmetic or logic operation. The flags can also be changed by the programmer.
- PSW is a bit addressable register.
- Each bit can be individually set or reset by the programmer.

- The bits can be referred to by their bit numbers (**PSW.4**) or by their name (**RS1**).

- **Example:**
  SETB PSW.3 ; Makes PSW.3 ← 1

  CLR PSW.4 ; Makes PSW.4 ← 0

6.5 SIGNAL DESCRIPTION OF 8051:



- 8051 has 40 pins.
  The function of these pins is briefly explained as follows.

<u>XTAL1 & XTAL2:</u>

- These are connected to the crystal oscillator.
- The typical operate in frequency is 12 MHz
- In Serial communication based applications, the operating frequency is chosen to be 11.0592 MHz, in order to derive the standard universal baud rates. This will be discussed in detail in the further chapters.

RESET:

- It is used to reset the 8051 microcontroller. On reset PC becomes 0000H.
- This address is called the reset vector address.
- From here, 8051 executes the BIOS program also called the Booting program or the monitor program.
- It is used to set-up the system and make it ready, to be used by the end-user.

ALE:

- It is used to enable the latching of the address. The address and data buses are multiplexed.
- This is done to reduce the number of pins on the 8051 IC.
- Once out of the chip, address and data have to be separated that is called de-multiplexing.
- This is done by a latch, with the help of ALE signal. ALE is "1" when the bus carries address and "0" when the bus carries data.
- This informs the latch, when the bus is carrying address so that the latch captures only address and not the data.

EA'

- It decides whether the first 4 KB of program memory space (0000H… 0FFFH) will be assigned to internal ROM or External ROM.
- If EA = 0, the External ROM begins from 0000H.
- In this case the Internal ROM is discarded. 8051 now uses only External ROM.
- If EA = 1, the External ROM begins from 1000H.
- In this case the Internal ROM is used. It occupies the space 0000H…0FFFH.
- In modern **FLASH ROM versions**, this pin also acts as **VPP** (12 Volt programming voltage) to write into the FLASH ROM.

PSEN'

- 8051 has a 16-bit address bus (A_{15}-A_0).
  This should allow 8051 to access 64 KB of external Memory as 2^{16} = 64 KB. Interestingly though, 8051 can access 64 KB of External ROM and 64 KB of External RAM, making a total of 128

KB.

- Both have the same address range 0000H to FFFFH.
- This does not lead to any confusion because there are separate control signals for External RAM and External ROM.
- RD and WR are control signals for External RAM.
- PSEN is the READ signal for External ROM.
- It is called Program Status Enable as it allows reading from ROM also known as Program Memory. Having separate control signals for External RAM and External ROM actually allows us to double the size of the external memory to a total of 128 KB from the original 64 KB.

<u>VCC & GND:</u>

- These are power supply pins.
- 8051 works at +5V / 0V power supply.

<u>P0.0-P0.7</u>

- These are 8 pins of Port 0.
- We can perform a byte operation (8-bit) on the whole port 0.
- We can also access every bit of port 0 individually by performing bit operations like set, clear, complement etc.
- The bits are called P0.0… P0.7.
- Additionally, Port 0 also has an alternate function.
- It carries the multiplexed address data lines.
- A0-A7 (the lower 8 bits of address) and D0-D7 (8 bits of data) are multiplexed into AD0-AD7.
- In any operation address and data are not issued simultaneously. First, address is given, then data is transferred. Using a common bus for both, reduces the number of pins.
- To identify if the bus is carrying address or data, we look at the ALE signal. If ALE = 1, the bus carries address,
- If ALE = 0, the bus carries data.

<u>P1.0-P1.7</u>

- These are 8 pins of Port 1.
- We can perform a byte operation (8-bit) on the whole port 1.
- We can also access every bit of port 1 individually by performing bit operations like set, clear, complement etc. on P1.0… P1.7.
- Port 1 also has NO alternate function

P2.0-P2.7

- These are 8 pins of Port 2.
- We can perform a byte operation (8-bit) on the whole port 2.
- We can also access every bit of port 2 individually by performing bit operations like set, clear, complement etc. on P2.0… P2.7.
- Additionally, Port 2 also has an alternate function. It carries the higher order address lines A8-A15.

P3.0-P3.7

- These are 8 pins of Port 3.
- We can perform a byte operation (8-bit) on the whole port 3. We can also access every bit of port 3 individually.
- The bits are called P0.0… P0.7.
- The various pins of Port 3 have a lot of alternate functions.

P3.0 (RXD) and P3.1 (TXD):

- They are used to receive and transmit serial data.
- This forms the serial port of 8051.

P3.2 (INT0) and P3.3 (INT1):

- They are external hardware interrupts of 8051.
- If they occur simultaneously, INTO is by default higher priority.

P3.4 (T0) and P3.5 (T1):

- They are used **timer clock inputs**.
- They provide external clock inputs to Timer 0 and Timer 1.

P3.6 (WR) and P3.7 (RD):

- They are used as **control signals for External RAM**.
- 8051 can access 64 KB External RAM from 0000H to FFFFH.

6.6 MEMORY ORGANISATION-RAM STRUCTURE:

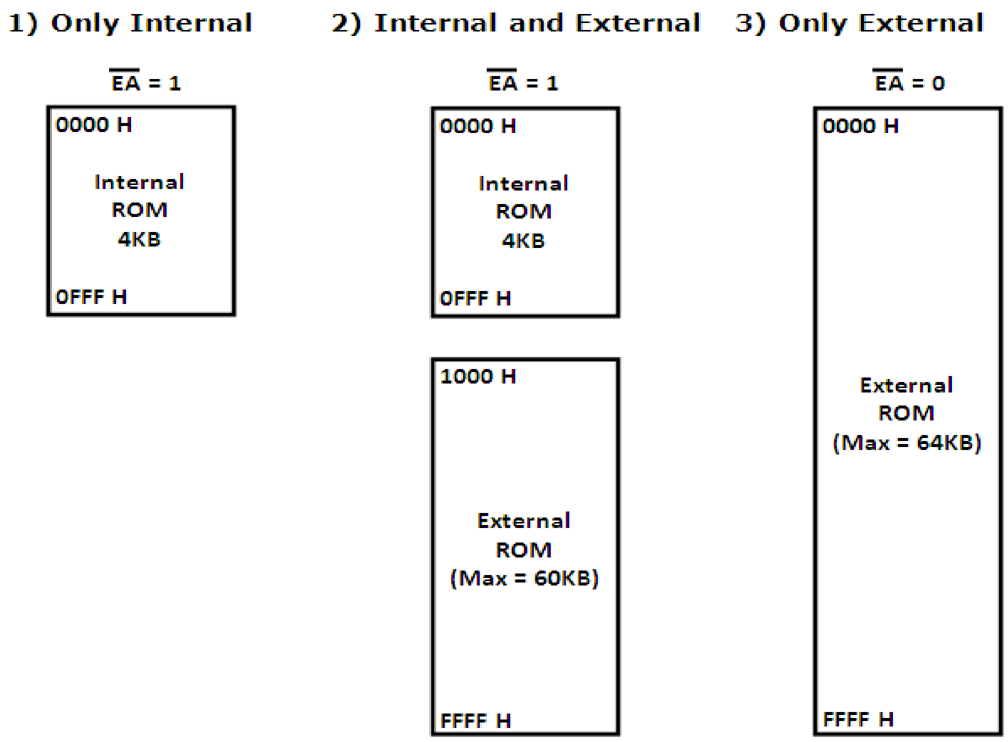
✓ 8051 operates with 4 different memories:

- Internal ROM

- External ROM
- Internal RAM
- External RAM

- Being based on Harvard Model, 8051 stores programs and data in separate memory spaces. Programs are stored in ROM, whereas data is stored in RAM.
- Microcontrollers are used in appliances.
- Washing machines, remote controllers, microwave ovens are some of the
- **EXAMPLES:**
  Here programs are generally permanent in nature and very rarely need to be modified. Moreover, the programs must be retained even after the device is completely switched off. Hence programs are stored in permanent (non-volatile) memory like ROM.
- Data on the other hand is continuously changed at runtime. For example current temperature, cooking time etc. in an oven.
- Such data is not permanent in nature and will certainly be modified in every usage of the device.
- Hence Data is stored in writeable memory like RAM.
- However, sometimes there is permanent data, such as ASCII codes or 7-segment display codes. Such data is stored in ROM, in the form of Look up tables and is accessed using a dedicated addressing mode called Indexed Addressing mode. We will discover this in more depth in further topics.
- We are now going to take a closer look at all four memories.

ROM ORGANIZATION / CODE MEMORY / PROGRAM MEMORY



✓ We can implement ROM in three different ways in 8051.

1. ONLY INTERNAL ROM:

- 8051 has 4 KB internal ROM.
- In many cases this size is sufficient and there is no need for connecting External ROM. Such systems use only Internal ROM of 8051.
- All addresses from 0000H… 0FFFH will be accessed from Internal ROM. Any address beyond that will be invalid.
- In such systems **EA** will be “1” as Internal ROM is being used.

2. INTERNAL AND EXTERNAL ROM:

- 8051 has 4 KB internal ROM.
- In many cases this size is may be insufficient and we may need to add some External ROM. Such systems use a combination of Internal ROM and External ROM.

- The “total” ROM that can be accessed is 64 KB.
- Since we are using the Internal ROM of 4 KB, the maximum amount of External ROM that can be connected is 60 KB.
- All addresses from 0000H… 0FFFH will be accessed from Internal ROM. Addresses 1000H… FFFH will be accessed from External ROM.
- In such systems EA will be “1” as Internal ROM is being used.

<u>3. ONLY EXTERNAL ROM:</u>

- This is the most interesting case.
- Though 8051 has 4 KB of Internal ROM, the user may choose the discard it and connect only External ROM.
- This may happen due to several reasons.
- The program stored in the Internal ROM may have become invalid or outdated, or the system may need to be upgraded etc.
- Such systems use only External ROM, and the Internal ROM is discarded. Here we can connect up to 64 KB of External ROM.
- All addresses from 0000H… FFFFH will be accessed from External ROM. But do keep in mind, that the Internal ROM is still present in 8051.
- We need to clearly indicate to 8051 that the Internal ROM must be ignored and every address from 0000H… FFFFH must be accessed externally. This is indicated by us to 8051 using **EA.**
- By making **EA = 0**, we inform 8051 that the Internal ROM must be discarded and all ROM must be accessed externally.

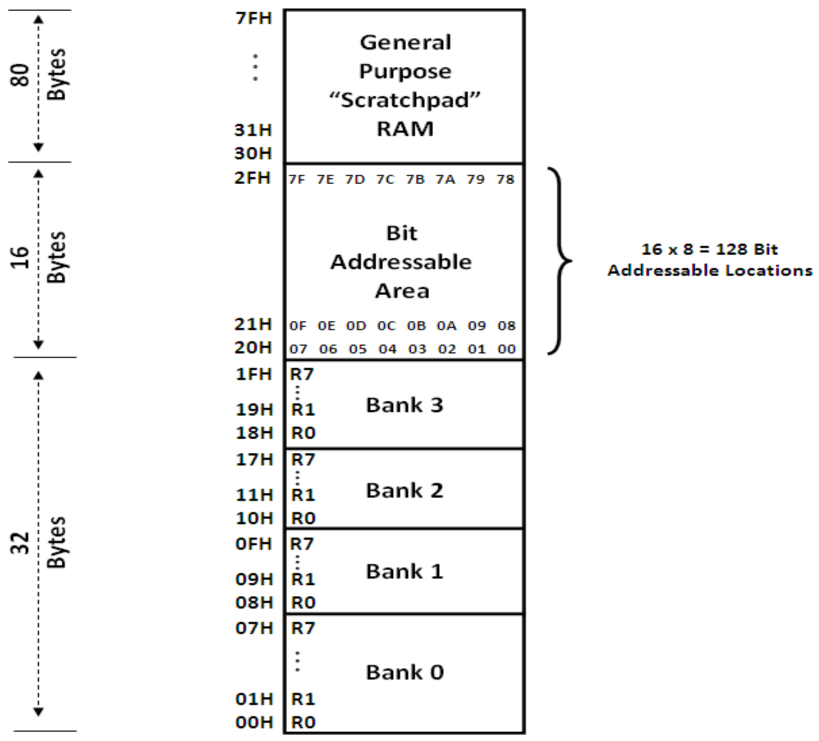
✓ **<u>Use of EA pin of 8051:</u>**

- **The EA pin of 8051 decides whether the Internal ROM will be used or not.**
- If the Internal ROM has to be used we must make **EA** = 1.
- Now 8051 will Access the internal ROM for all addresses from 0000H to 0FFFH and will only access external ROM for addresses 1000H and beyond.

- But if **EA** = 0, then the Internal ROM is completely discarded.
- Now 8051 will access the External ROM for all addresses from 0000H to FFFFH, hence discarding the internal ROM.

- 8051 checks **EA** pin during every ROM operation where the address is 0000H…
- 0FFFH. If **EA = 1,** this location is accessed from internal ROM.
- If **EA = 0,** this location is accessed from external ROM.
- If the address is 1000H or more, 8051 does not check **EA** as this location can only be present in External ROM.

STRUCTURE OF INTERNAL RAM:



- 8051 has a 128 Bytes of internal RAM. These are 128 locations of 1 Byte each.
- The address range is 00H… 7FH.
- This RAM is used for storing data.
- It is divided into three main parts: Register Banks, Bit addressable area and a general purpose area.

<u>REGISTER BANKS:</u>

- The first 32 locations (Bytes) of the Internal RAM from 00H… 1FH, are used by the programmer as general purpose registers.
- Having so many general purpose registers makes programming easier and faster.

- But as a downside, this also vastly increases the number of opcodes (refer my class lectures for detailed understanding of this).
- Hence the 32 registers are divided into 4 banks, each having 8 Registers R0… R7.
- The first 8 locations 00H… 07H are registers R0… R7 of bank 0.
- Similarly locations 08H… 0FH are registers R0… R7 of bank 1 and so on. A register can be addressed using its name, or by its address.
- E.g. Location 00H can be accessed as R0, if Bank 0 is the active bank.
  MOV A, R0; "A" register gets data from register R0.
  It can also be accessed as Location 00H, irrespective of which bank is the active bank.
  MOV A, 00H; "A" register gets data from Location 00H.
- The appropriate bank is selected by the RS1, RS0 bits of PSW.
  Since PSW is available to the programmer, any Bank can be selected at run-time.
- Bank 0 is selected by default, on reset.

<u>BIT ADDRESSABLE AREA:</u>

- The next 16-bytes of RAM, from 20H… 2FH, is available as Bit Addressable Area.
- We can perform ordinary byte operations on these locations, as well as bit operations.
- As each location has 8-bits, we have a total of →16 X 8 = 128 Addressable Bits.
- These bits can be addressed using their individual address 00H … 7FH. SETB 00H; Will store a "1" on the LSB of location 20H
  CLR 07H; Will store a "0" on the MSB of location 20H

- Normal "BYTE" operations can also be performed at the addresses: 20H … 2FH. MOV 20H, #00H; Will store a "0" on all 8-bits of location 20H.

- Here is something very interesting to know and will also help you understand further topics. The entire internal RAM is of 128 bytes so the address range is 00H… 7FH.
- The bit addressable area has 128 bits so its bit addresses are also 00h… 7FH.

- This means every address 00H… 7FH can have two meanings, it could be a byte address or a bit address.

- This does not lead to any confusion, because the instruction in which we use the address, will clearly indicate whether it is a bit operation or a byte operations.

- SETB, CLR etc. are bit ops whereas ADD, SUB etc. are byte operations.
- SETB 00H; this is a bit operation. It will make Bit location 00H contain a value "1".
- MOV A, 00H; this is a byte operation. A" register will get 8-bit data from byte location 00H.

GENERAL PURPOSE AREA

- The general-purpose area ranges from location 30H … 7FH.
- This is an 80-byte area which can be used for general data storage.

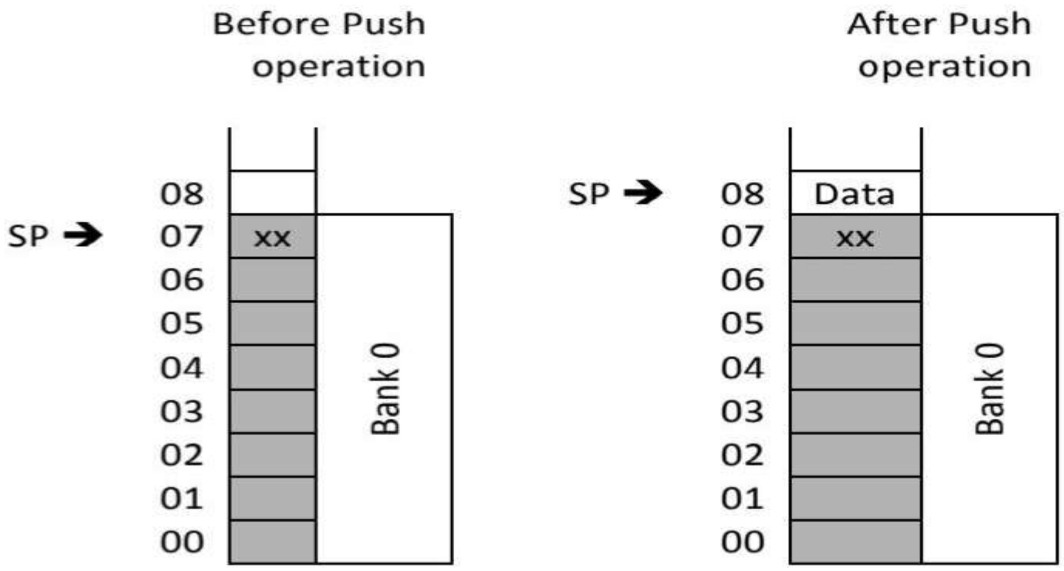
STACK OF 8051:

- Another important element of the Internal RAM is the Stack.
- Stack is a set of memory locations operating in Last in First out (LIFO) manner.
- It is used to store return addresses during ISRs and also used by the programmer to store data during programs.
- In 8051, the Stack can only be present in the Internal RAM.
- This is because, SP which is an 8-bit register, can only contain an 8-bit address and External RAM has 16-bit address. (#Viva)
- On reset SP gets the value 07H.
- Thereafter SP is changed by every PUSH or POP operation in the following manner:

PUSH:	**POP:**
SP → SP + 1	Data → [SP]
[SP] → New data	**SP→SP – 1**

- The reset value of SP is 07H because, on the first PUSH, SP gets incremented and then data is pushed on to the stack. This means the very first data will be stored at location 08H.

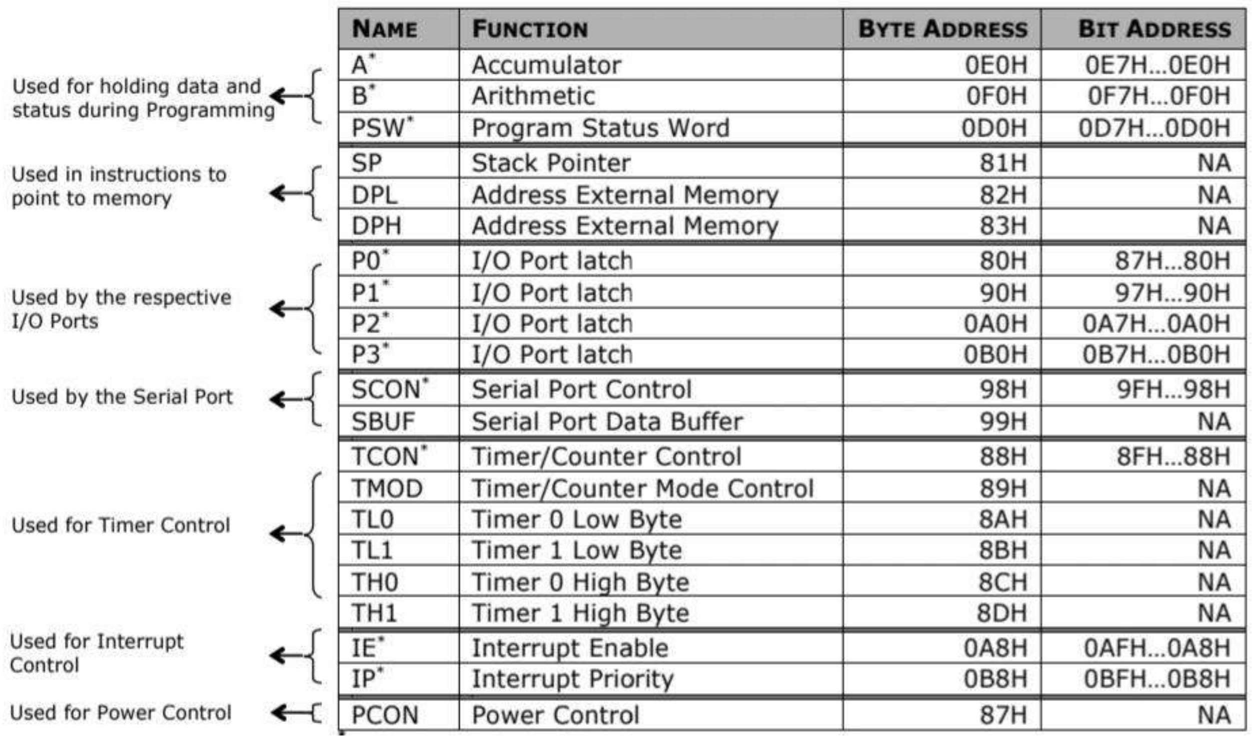
- This does not affect the default bank (0) and still gives the stack, the maximum space to grow.



- The programmer can relocate the stack to any desired location by simply putting a new value into SP register.

6.7 SFR (SPECIAL FUNCTION REGISTER):

- 8051 has 21, 8-bit Special Function registers.



	NAME	FUNCTION	BYTE ADDRESS	BIT ADDRESS
Used for holding data and status during Programming	A*	Accumulator	0E0H	0E7H...0E0H
	B*	Arithmetic	0F0H	0F7H...0F0H
	PSW*	Program Status Word	0D0H	0D7H...0D0H
Used in instructions to point to memory	SP	Stack Pointer	81H	NA
	DPL	Address External Memory	82H	NA
	DPH	Address External Memory	83H	NA
Used by the respective I/O Ports	P0*	I/O Port latch	80H	87H...80H
	P1*	I/O Port latch	90H	97H...90H
	P2*	I/O Port latch	0A0H	0A7H...0A0H
	P3*	I/O Port latch	0B0H	0B7H...0B0H
Used by the Serial Port	SCON*	Serial Port Control	98H	9FH...98H
	SBUF	Serial Port Data Buffer	99H	NA
Used for Timer Control	TCON*	Timer/Counter Control	88H	8FH...88H
	TMOD	Timer/Counter Mode Control	89H	NA
	TL0	Timer 0 Low Byte	8AH	NA
	TL1	Timer 1 Low Byte	8BH	NA
	TH0	Timer 0 High Byte	8CH	NA
	TH1	Timer 1 High Byte	8DH	NA
Used for Interrupt Control	IE*	Interrupt Enable	0A8H	0AFH...0A8H
	IP*	Interrupt Priority	0B8H	0BFH...0B8H
Used for Power Control	PCON	Power Control	87H	NA

● SFRs are 8-bit registers. Each SFR has its own special function.

● They are placed inside the Microcontroller.

● They are used by the programmer to perform special functions like controlling the timers, the serial port, the I/O ports etc.

● As SFRs are available to the programmer, we will use them in instructions. This causes another problem. SFRs are registers after all, and hence using them would tremendously increase the number of opcodes to reduce the number of opcodes, SFRs are allotted addresses. These addresses must not clash with any other addresses of the existing memories

● Incidentally, the internal RAM is of 128 bytes and uses addresses only from 00H… 7FH. This gives an entire range of addresses from 80H… FFH completely unused and can be freely allotted to the SFRs.

● Hence SFRs are allotted addresses from 80H… FFH.

● It is not a co-incidence that these addresses are free. The Internal RAM was restricted to 128 bytes instead of 256 bytes so that these addresses are free for SFRs.

● To avoid this problem, even the bits of the SFRs are allotted addresses. These are bit addresses, which are different from byte addresses. These bit addresses must not clash with those of the bit addressable area of the Internal RAM. Amazingly, even the bit addresses in the Internal RAM are 00H… 7FH (again 128 bits), keeping bit addresses 80H… FFH free to be used by the SFR bits.

- So bit addresses 80H… FFH are allotted to the bits of various SFRs.
- Port 0 has a byte address of 80H and its bit addresses are from 80H… 87H.
- A byte operation at address 80H will affect entire Port0.
- **E.g.-**MOV A, P0; this refers to Byte address 80H that's whole Port 0. 12) A bit

  Operation at 80H will affect only P0.0.

- E.g. SETB P0.0; this refers to bit address 80H that's Port0.0

6.8 REGISTERS OF 8051 MICROCONTROLLER:

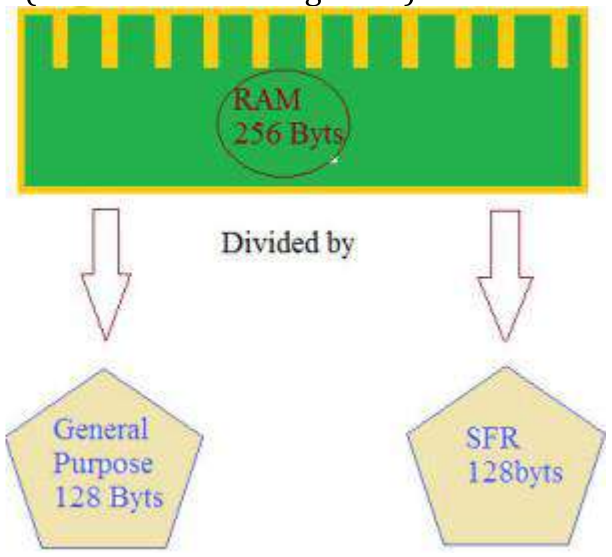
- In the CPU, registers are used to store information temporarily.
- That information could be a byte of data to be processed, or an address pointing to the data to be fetched.
- The vast majority of 8051 registers are 8-bit registers.in the 8051 there is only one data type 8-bits.
- With an 8-bit data type, any data larger than 8-bits must be broken into 8-bit chunks before it is processed.
- Since there are a larger number of registers in the 8051.
- The most widely used registers of the 8051 are A (accumulator), B, R0, R1, R2, R3, R4, R5, R6, R7, DPTR (data pointer) and PC (program counter).
- All of the above registers are 8-bits except DPTR and the program counter (PC).
- The accumulator, register A is used for all arithmetic and logic instructions.

Types of Registers:

The 8051 microcontroller contains mainly two types of registers:

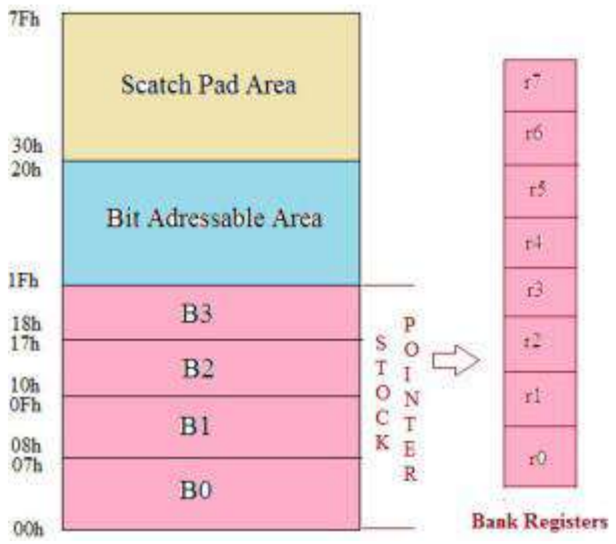
- General purpose registers (Byte addressable registers)

- Special function registers (Bit addressable registers)



- The 8051 microcontroller consists of 256 bytes of RAM memory, which is divided into two ways, such as 128 bytes for general purpose and 128 bytes for special function registers (SFR) memory.
- The memory which is used for general purpose is called as RAM memory, and the memory used for SFR contains all the peripheral related registers like Accumulator, 'B' register, Timers or Counters, and interrupt related registers.

General Purpose Registers:

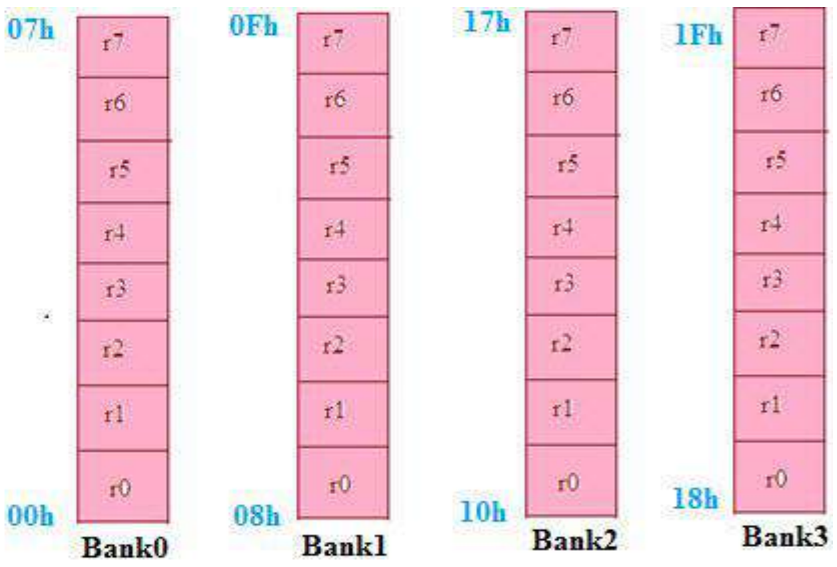


General Purpose Memory

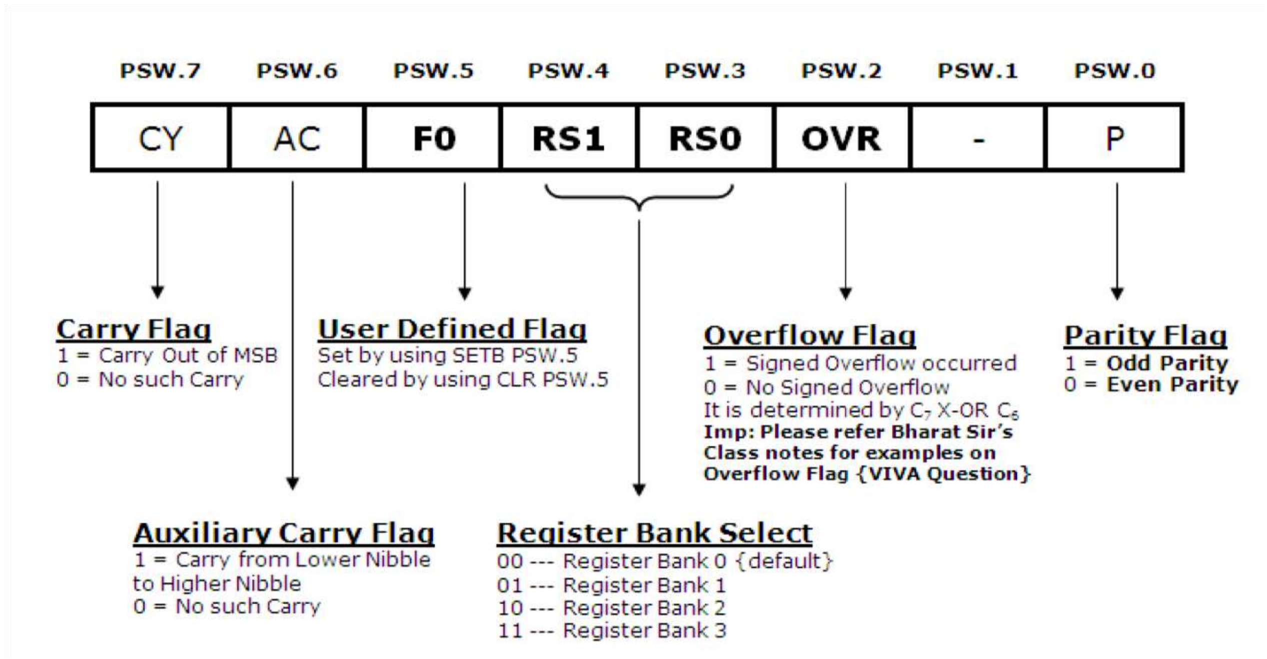
- The general purpose memory is called as the RAM memory of the 8051 microcontroller, which is divided into 3 areas such as **banks**, **bit-addressable area,** and **scratch-pad area.**
- The banks contain different general purpose registers such as R0-R7, and all such registers are byte-addressable registers that store or remove only 1-byte of data.

Banks and Registers:

- The B0, B1, B2, and B3 stand for banks and each bank contains eight general purpose registers ranging from ‘R0’ to ‘R7’.
- All these registers are byte-addressable registers. Data transfer between general purpose registers to general purpose registers is not possible. These banks are selected by the Program Status Word **(PSW**) register.



FLAG REGISTER (PSW) OF 8051:



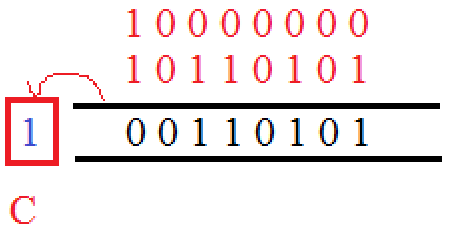
PSW – PROGRAM STATUS WORD

- It is an 8-bit register.
- It is also called the "Flag register", as it mainly contains the status flags.

- These flags indicate status of the current result.
- They are changed by the ALU after every arithmetic or logic operation.
- The flags can also be changed by the programmer.
- PSW is a bit addressable register.
- Each bit can be individually set or reset by the programmer.
- The bits can be referred to by their bit numbers (PSW.4) or by their name (RS1).

CY - CARRY FLAG

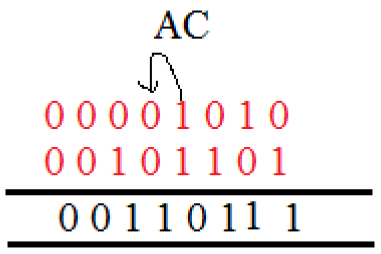
- It indicates the carry out of the MSB, after any arithmetic operation.
- If CY = 1, There was a carry out of the MSB
- If CY = 0, There was no carry out of the MSB



AC – AUXILIARY CARRY FLAG

- It indicates the carry from lower nibble (4-bits) to higher nibble.
- If the 8bits are numbered Bit 7 --- Bit 0, this is the carry from Bit 3 to Bit 4.
- If AC = 1, There was an auxiliary carry
- If AC = 0, There was no auxiliary carry

✓ Note: It is particularly useful in an operation called DA A (Decimal Adjust after Addition).

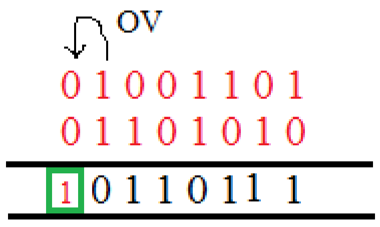


OVR - OVERFLOW FLAG

- It indicates if there was an overflow during a signed operation.
- An 8-bit signed number has the range -80H… 00H… +7FH. Any result, out of this range causes an overflow.

- If OVR = 1, There was an overflow in the result If OVR = , There was no overflow in the result
- Overflow is determined by doing an Ex-Or between the 2nd last carry (C_6) and the last carry (C_7)

✓ Note: After an overflow, the Sign (MSB) of the result becomes wrong.



P - PARITY FLAG

- It indicates the Parity of the result.
- Parity is determined by the number of 1's in the result.
- If PF = 1, The result has ODD parity
- If PF = 0, The result has EVEN parity

F0 – USER DEFINED FLAG

- This flag is available to the programmer.
- It can be used by us to store any **user defined information**.
- For example: In an Air Conditioning unit, programmer can use this flag indicate whether the compressor is ON or OFF (1 or 0).
- This flag can be changed by simple instructions like SETB and CLR.
- SETB PSW.5; This makes F0 bit →1
- CLR PSW.5; This makes F0 bit→0

RS1, RS0 – REGISTER BANK SELECT

- The initial 32 locations (bytes) of the Internal RAM are available to the programmer as registers.
- Having so many registers makes programming easier and faster.
- Naming R0… R31, would tremendously increase the number of opcodes.
- Hence the registers are divided into 4 banks: Bank0… Bank3.
- Each bank has 8 registers named R0… R7.
- At a time, only of the four banks is the "active bank".
- RS1 and RS0 are used by the programmer to select the active bank.

RS1 RS0	REGISTER BANK	SELECTED BY INSTRUCTIONS
0 0	Bank 0	CLR PSW.4 CLR PSW.3
0 1	Bank 1	CLR PSW.4 SETB PSW.3
1 0	Bank 2	SETB PSW.4 CLR PSW.3
1 1	Bank 3	SETB PSW.4 SETB PSW.3

NUMERICAL EXAMPLES FOR FLAG REGISTER:
Example 1:

32 H→0011 0001

<u>23 H</u>→<u>0010 0011</u>

54 H→0101 0100

- Flag Affected: CY=0, AC=0, OVR=0, P=1

Example 2:

39 H→0011 1001

<u>27 H</u>→<u>0010 0111</u>

60 H→0110 0000

- Flag Affected: CY=0, AC=1, OVR=0, P=0

Example 3:

42 H→0100 0010

<u>44 H</u>→<u>0100 0100</u>

86 H→1000 0110

- Flag Affected: CY=0, AC=0, OVR=1, P=1

- The result 86H is out of range for a "Signed" Number as it has become greater than +7FH.
- Such an event is called a "Signed Overflow".
- In such a case the MSB of the result gives a wrong sign.
- Though the result is +ve (+86H) the MSB is "1" indicating that the result is –ve.
- Overflow is determined by doing an Ex-Or between the 2nd last Carry and the last Carry.
- Here the 2nd last Carry (the one coming into the MSB) is "1".
- The final carry (The one going out of the MSB) is "0". As "1" Ex-Or "0" = "1", the Overflow flag is "1".

Example 1:

Assembly program to move 6 natural numbers in bank0 register R0-R5

```
Org 0000h (starting addresses declaration)
MOV PSW, #00h (open the bank0 memory)
MOV r0, #00h (starting address of bank0 memory)
MOV r1, #01h
MOV r2, #02h
MOV r2, #03h
MOV r3, #04h
MOV r4, #05h
END
```

Example 2:

Assembly program to move 6 natural numbers in bank1 register R0-R7

```
Org 0000h (starting addresses declaration)
MOV PSW, #08h (open the bank1 memory)
MOV r0, 00h (value send to the bank1 memory)
MOV r1, 02h
MOV r2, 02h
MOV r2, 03h
MOV r3, 04h
MOV r4, 05h
MOV r5, 06h
MOV r6, 07h
MOV r7, 08h
END
```

6.9 INTERRUPTS OF 8051:

- 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.
- 8051 has 5 interrupt signals, i.e. INT0, TFO, INT1, TF1, and RI/TI. Each interrupt can be enabled or disabled by setting bits of the IE register and the whole interrupt system can be disabled by clearing the EA bit of the same register.

Or

- An interrupt is an event that occurs randomly in the flow of continuity. It is just like a call you have when you are busy with some work and depending upon call priority you decide whether to attend or neglect it.
- Same thing happens in microcontrollers. 8051 architecture handles **5 interrupt sources,** out of which **two are internal (Timer Interrupts),** two are **external** and one is a **serial interrupt**. Each of these interrupts has their interrupt vector address. Highest priority interrupt is the Reset, with vector address 0x0000.

Vector Address:

- This is the address where controller jumps after the interrupt to serve the ISR (interrupt service routine).

Interrupt	**Flag**	**Interrupt vector address**
Reset	-	0000H
INT0 (Ext. int. 0)	IE0	0003H
Timer 0	TF0	000BH
INT1 (Ext. int. 1)	IE1	0013H
Timer 1	TF1	001BH
Serial	TI/RI	0023H

Reset

- Reset is the highest priority interrupt, upon reset 8051 microcontroller start executing code from 0x0000 address.

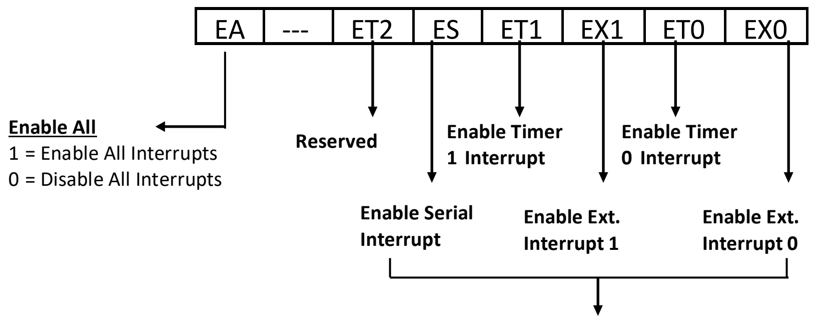
Internal interrupt (Timer Interrupt)

- 8051 has two internal interrupts namely **timer0** and **timer1**. Whenever timer overflows, timer overflow flags (TF0/TF1) are set. Then the microcontroller jumps to their vector address to serve the interrupt. For this, global and timer interrupt should be enabled.

Serial Port Interrupt (Common for RI or TI)

- All interrupts are **vectored** i.e. they cause the program to execute an ISR from a pre-determined address in the Program Memory.
- Interrupts are controlled mainly by **IE** and **IP** SFR's and also by some bits of **TCON** SFR.

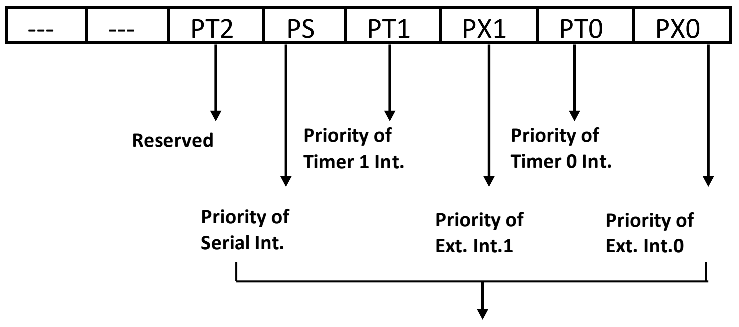
- **<u>IE - Interrupt Enable (SFR)</u> [Bit-Addressable As IE.7 to IE.0]**



1 = Enable respective Interrupt

0 = Disable respective Interrupt

- **<u>IP - Interrupt Priority (SFR)</u> [Bit-Addressable As IP.7 to IP.0]**



- 10 = Priority of respective Interrupt = Priority of respective Interrupt i.e. HIGHLOW

Timer Overflow Interrupts (TF1 and TF0)

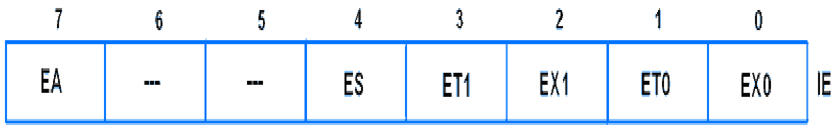
- When any of the 2 Timers overflow, their respective bit TFX (TF1 or TF0) is set in TCON SFR.
- If Timer Interrupts are enabled then the timer interrupt occurs. The TFX bits are cleared when their respective ISR is executed.

Serial Port Interrupt (RI or TI)

- While receiving serial data, when a complete byte is received the RI (receive interrupt) bit is set in the SCON.
- During transmission, when a complete byte is transmitted the TI (transmit interrupt) bit is set in the SCON.
- ANY of these events can cause the Serial Interrupt (provided Serial Interrupt is enabled).
- The RI/TI bit is not cleared automatically on executing the ISR. The program should explicitly clear this bit to allow further Serial Interrupts.

IE register: Interrupt Enable Register

- IE register is used to enable/disable interrupt sources.



- **Bit 7 – EA:** Enable All Bit

  1 = Enable all interrupts

  0 = Disable all interrupts

- **Bit 6,5 –** Reserved bits
- **Bit 4 – ES:** Enable Serial Interrupt Bit

  1 = Enable serial interrupt

  0 = Disable serial interrupt

- **Bit 3 – ET1:** Enable Timer1 Interrupt Bit

  1 = Enable Timer1 interrupt

  0 = Disable Timer1 interrupt

- **Bit 2 – EX1:** Enable External1 Interrupt Bit

  1 = Enable External1 interrupt

  0 = Disable External1 interrupt

- **Bit 1 – ET0:** Enable Timer0 Interrupt Bit

  1 = Enable Timer0 interrupt

  0 = Disable Timer0 interrupt

- **Bit 0 – EX0:** Enable External0 Interrupt Bit

  1 = Enable External0 interrupt

  0 = Disable External0 interrupt

Interrupt priority

- Priority to the interrupt can be assigned by using **interrupt priority register (IP)**

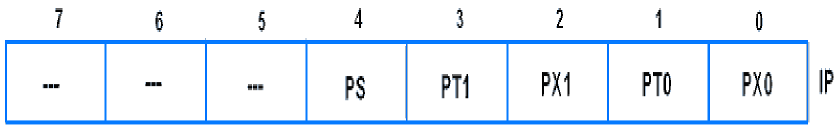
Interrupt priority after Reset:

Priority	Interrupt source	Intr. bit / flag
1	External Interrupt 0	INT0
2	Timer Interrupt 0	TF0
3	External Interrupt 1	INT1
4	Timer Interrupt 1	TF1
5	Serial interrupt	(TI/RI)

- In the table, interrupts priorities upon reset are shown. As per 8051 interrupt priorities, lowest priority interrupts are not served until microcontroller is finished with higher priority ones. In a case when two or more interrupts arrives microcontroller queues them according to priority.

IP Register: Interrupt priority register

- 8051 has interrupt priority register to assign priority to interrupts.



- **Bit 7, 6, 5 –** Reserved bits.
- **Bit 4 – PS:** Serial Interrupt Priority Bit

  1 = Assign high priority to serial interrupt.
  0 = Assign low priority to serial interrupt.

- **Bit 3 – PT1:** Timer1 Interrupt Priority Bit

  1 = Assign high priority to Timer1 interrupt.
  0 = Assign low priority to Timer1 interrupt.

- **Bit 2 – PX1:** External Interrupt 1 Priority Bit

1 = Assign high priority to External1 interrupt.
0 = Assign low priority to External1 interrupt.

- **Bit 1 – PT0:** Timer0 Interrupt Priority Bit

  1 = Assign high priority to Timer0 interrupt.
  0 = Assign low priority to Timer0 interrupt.

- **Bit 0 – PX0:** External0 Interrupt Priority Bit

  1 = Assign high priority to External0 interrupt.
  0 = Assign low priority to External0 interrupt.

Or

External Interrupts (INT1 and INT0):

- Pins INT1 and INT0 are inputs for external interrupts.
- These interrupts can be -ve edge or low-level triggered depending upon the IT0 and IT1 bit in
- TCON SFR. (ITX = 1 è -ve edge triggered)
- Whew any of these interrupts occur the respective bits TE1 or IE0 are set in the TCON SFR. If External Interrupts are enabled then the ISR is executed from the respective address.

Interrupt Sequence

- The following sequence is executed to service an interrupt:
- Address of next instruction of the main program i.e. PC is pushed into the Stack.
- All interrupts are disabled, by making EA bit in IE SFR$\leftarrow$0.
- Program Control is shifted to the Vector Address (location) of the ISR. The ISR begins.

Returning Sequence

- RETI instruction denotes the end of the ISR.
- It causes the processor to POP the contents of the Stack Top into the PC.
- It also re-enables interrupts by making EA bit in IE SFR$\leftarrow$1. The main program resumes.

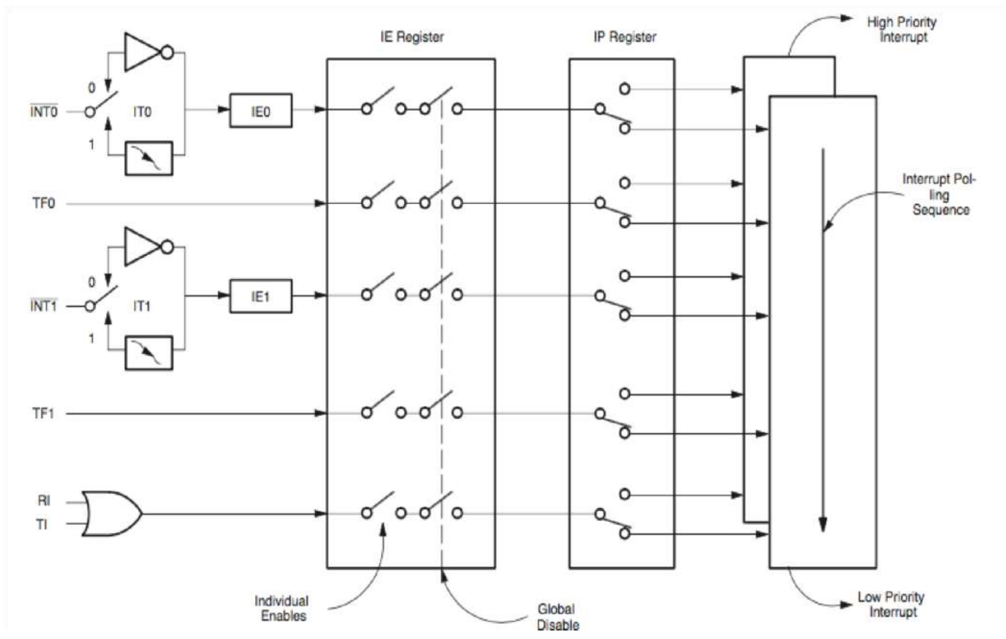
Interrupt Priorities

- 8051 has only two priority levels for the interrupts: Low and High.

- Interrupt priorities are set using the IP SFR.
- As the name suggests, a high priority interrupt can interrupt a low priority interrupt.
- It two or more interrupts at the same level occur simultaneously then priorities are decided as follows:

INTERRUPT	**PRIORITY**	**VECTOR ADDRESS**
$\overline{\text{INT0}}$	1	0003H
TF0	2	000BH
$\overline{\text{INT1}}$	3	0013H
TF1	4	001BH
Serial (RI or TI)	5	0023H

<u>DIAGRAM FOR INTERRUPTS</u>



OR

External interrupts in 8051

- 8051 has two external interrupt **INT0 and INT1.**

- 8051 controller can be interrupted by external Interrupt, by providing level or edge on external interrupt pins PORT3.2, PORT3.3.
- External peripherals can interrupt the microcontroller through these external interrupts if global and external interrupts are enabled.
- Then the microcontroller will execute current instruction and jump to the Interrupt Service Routine (ISR) to serve to interrupt.
- In polling method microcontroller has to continuously check for a pulse by monitoring pin, whereas, in interrupt method, the microcontroller does not need to poll. Whenever an interrupt occurs microcontroller serves the interrupt request.
- **External interrupt has two types of activation level**

1. Edge triggered (Interrupt occur on rising/falling edge detection)
2. Level triggered (Interrupt occur on high/low-level detection)

- **In 8051, two types of activation level are used. These are,**

<u>Low level triggered</u>

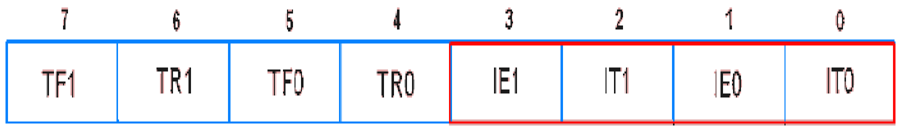
- Whenever a low level is detected on the INT0/INT1 pin while global and external interrupts are enabled, the controller jumps to interrupt service routine (ISR) to serve interrupt.

<u>Falling edge triggered</u>

- Whenever falling edge is detected on the INT0/INT1 pin while global and ext. interrupts are enabled, the controller jumps to interrupt service routine (ISR) to serve interrupt.

- There are lower four flag bits in **TCON register** required to select and monitor the external interrupt type and ISR status.

TCON: Timer/ counter Register



7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

✓ **<u>Bit 3- IE1</u>:**

- External Interrupt 1 edge flag, set by hardware when interrupt on INT1 pin occurred and cleared by hardware when interrupt get processed.

✓ **Bit 2- IT1:**

- This bit selects external interrupt event type on INT1 pin,
- 1= sets interrupt on falling edge
- 0= sets interrupt on low level

✓ **Bit 1- IE0:**

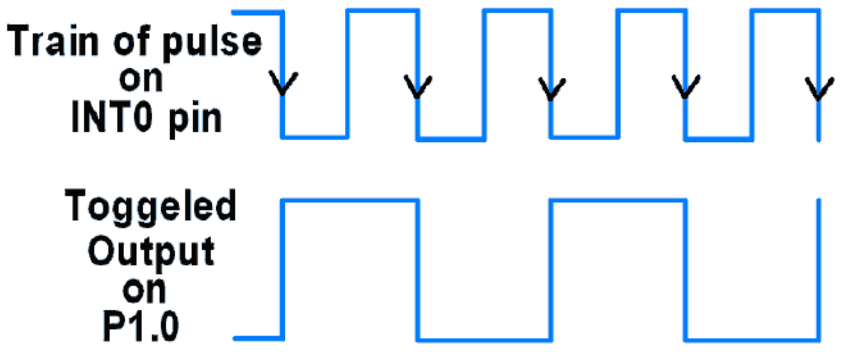
- Interrupt0 edge flag, set by hardware when interrupt on INT0 pin occurred and cleared by hardware when an interrupt is processed.

✓ **Bit 0 - IT0:**

- This bit selects external interrupt event type on INT0 pin.
- 1= sets interrupt on falling edge
- 0= sets interrupt on low level

Example

Let's program the external interrupt of AT89C51 such that, when falling edge is detected on INT0 pin then the microcontroller will toggle the P1.0 pin.



6.10 TIMER SECTION OF 8051:

- The 8051 has two timers: timer0 and timer1. They can be used either as timers or as counters. Both timers are 16 bits wide. Since the 8051 has an 8-bit architecture, each 16-bit is accessed as two separate registers of low byte and high byte.

- There are two 16-bit timers and counters in 8051 microcontroller: **timer 0 and timer 1**. Both timers consist of 16-bit register in which the lower byte is stored in TL and the higher byte is stored in TH. Timer can be used as a counter as well as for timing operation that depends on the source of clock pulses to counters.
- 8051 has 2, 16-bit Up Counters T1 and T0.
- If the counter counts internal clock pulses it is known as timer.
- If it counts external clock pulses it is known as counter.
- Each counter is divided into 2, 8-bit registers TH1 - TL1 and TH0 - TL0.
- The timer action is controlled mainly by the TCON and the TMOD registers.

<u>TCON - Timer Control (SFR)</u> [Bit-Addressable As TCON.7 to TCON.0]

TF1	**TR1**	**TF0**	**TR0**	IE1	IT1	IE0	IT0

TF1 and TF0: (<u>Timer Overflow Flag</u>)

- Set (1) when Timer 1 or Timer 0 overflows respectively i.e. its bits roll over from all 1's to all 0's.
- Cleared (0) when the processor executes ISR (address 001BH for Timer 1 and 000BH for Timer 0).

TR1 and TR0: (<u>Timer Run Control Bit</u>)

- Set (1) - Starts counting on Timer 1 or Timer 0 respectively.
- Cleared (0) - Halts Timer 1 or Timer 0 respectively.

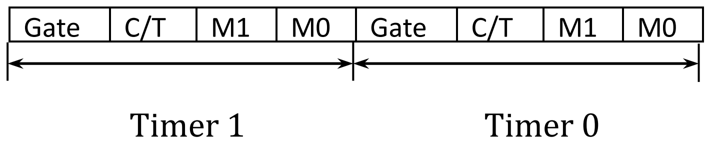
IE1 and IE0: (<u>External Interrupt Edge Flag</u>)

- Set (1) when external interrupt signal received at INT1 or INT0 respectively.
- Cleared (0) when ISR executed (address 0013H for Timer 1 and 0003H for Timer 0).

IT1 and IT0: (<u>External Interrupt Type Control Bit</u>)

- Set (1) - Interrupt at INT1 or INT0 must be -ve edge triggered.
- Cleared (0) - Interrupt at INT1 or INT0 must be low-level triggered.

<u>TMOD - Timer Mode Control (SFR)</u> [NOT Bit-Addressable]



C/T: (<u>Counter/Timer</u>)

- Set (1) - Acts as Counter (Counts external frequency on T1 and T0 pin inputs).
- Cleared (0) - Acts as Timer (Counts internal clock frequency, fosc/12).

Gate: (<u>Gate Enable Control bit</u>)

- Set (1) - Timer controlled by hardware i.e. INTX signal.
- Cleared (0) – Counting independent of INTX signal.

M1, M0: (<u>Mode Selection bits</u>)

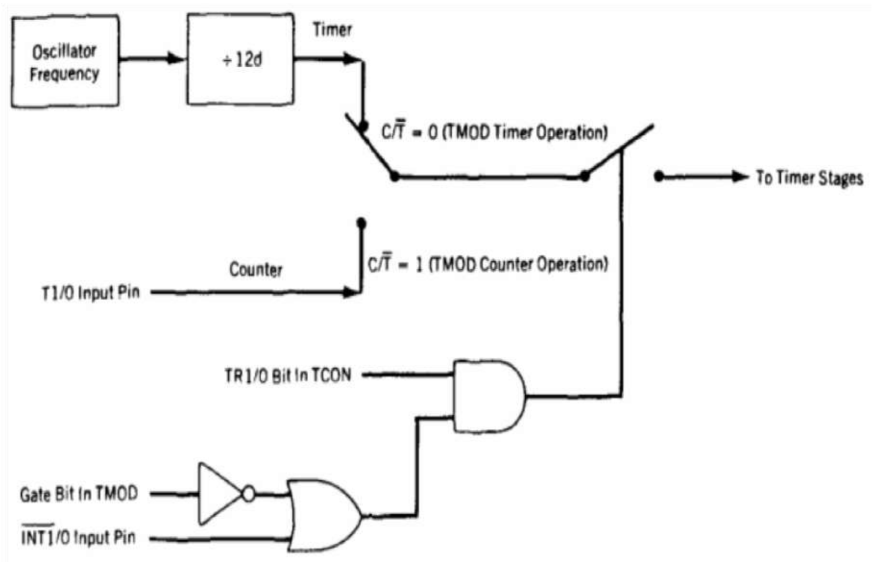
- Used to select the operational modes of the respective Timer.

M1 M0	**Timer Mode**
0 0	Mode 0
0 1	Mode 1
1 0	Mode 2
1 1	Mode 3

<u>Timer Counter Interrupts</u>

- To use the timer, a certain count value is placed in the Count Register.
- This value is the →**Max count—desired count+1**
- On each count (rising edge of the input clock) the counter increments its value.
- When the counter rolls over (i.e. form all 1's to all 0's) it is said to overflow.
- Thus the Timer Overflow Flag, TFX (TF1 or TF0) is set.
- If timer interrupt is enabled then the Timer Interrupt will occur on overflow.

<u>Timer Counter Logic</u>

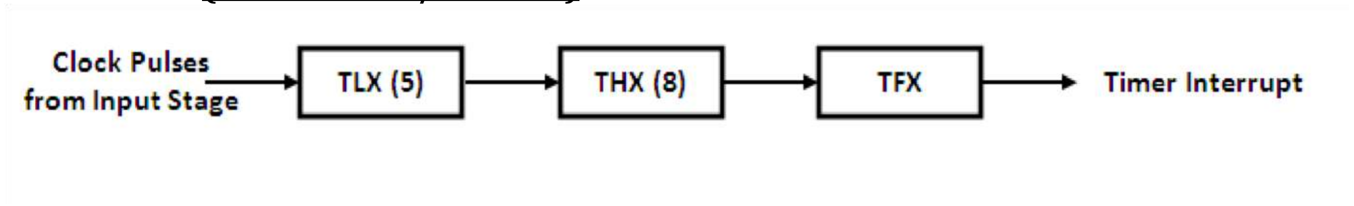


- As shown above, based on C/T bit the timer functions as a Counter or as a Timer.
- If it is a Timer, it will count the internal clock frequency of 8051 divided by 12_d (f/12).
- If it is a Counter, the input clock signal is applied at the TX (T1 or T0) input pins for Timer1 or
- Timer0 respectively. #please refer Bharat Sir's Lecture Notes for this...
- As shown the Timer is running only if the TRX bit (TR1 or TR0) is set.

- Also if the Gate bit is set in the TMOD then the INTX (INT1 or INT0) pin must be "high (1)" for the timer to count.

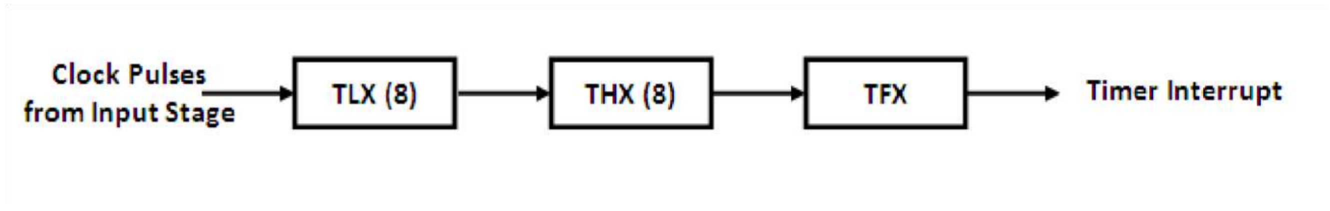
TIMER MODES:

a) Timer Mode 0 <u>(13-bit Timer/Counter)</u>



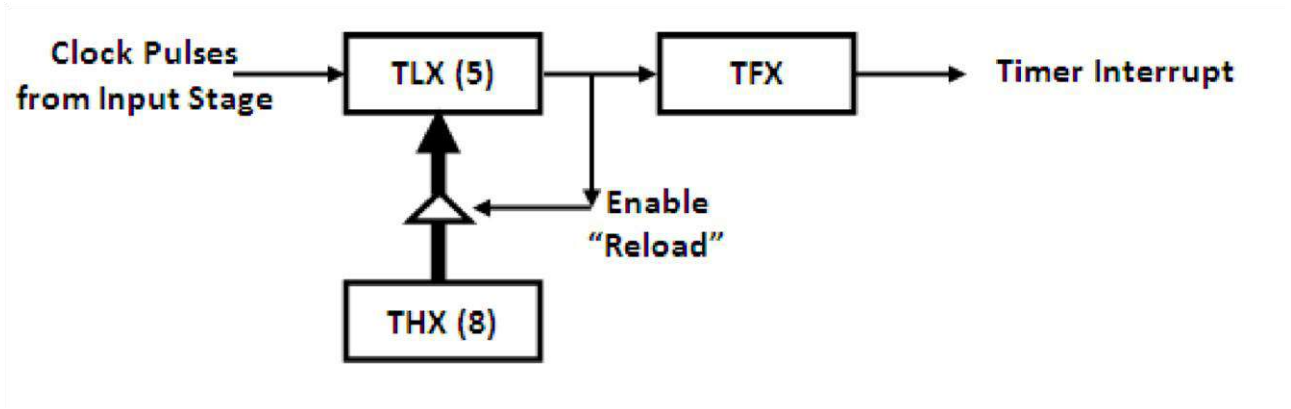
- THX is used as an 8-bit counter.
- TLX is used as a 5-bit pre-set. Hence 13-bits are used for counting.
- On each count the TLX increments.
- Each time TLX rolls-over, THX increments.
- Thus the input frequency is divided by 32 (5-bits of TLX and 2^5 = 32).
- The timer overflow flag TFX is set only when THX overflows i.e. rolls from FFH to 00H. Max Count = 2^{13} = 8K = 8192 (1FFFH). Hence Max Delay→ 8192(12/f)

b) Timer Mode 1 (16-bit Timer/Counter)



- All 16-bits of the Counter are used (8 bits of THX and 8 bits of TLX).
- On each count the 16-bit Timer increments.
- The timer overflow flag TFX is set when the Timer rolls-over from FFFFH to 0000H. Max Count => 2^{16} = 16K = 65536 (FFFFH). Hence Max Delay →65536(12/f).

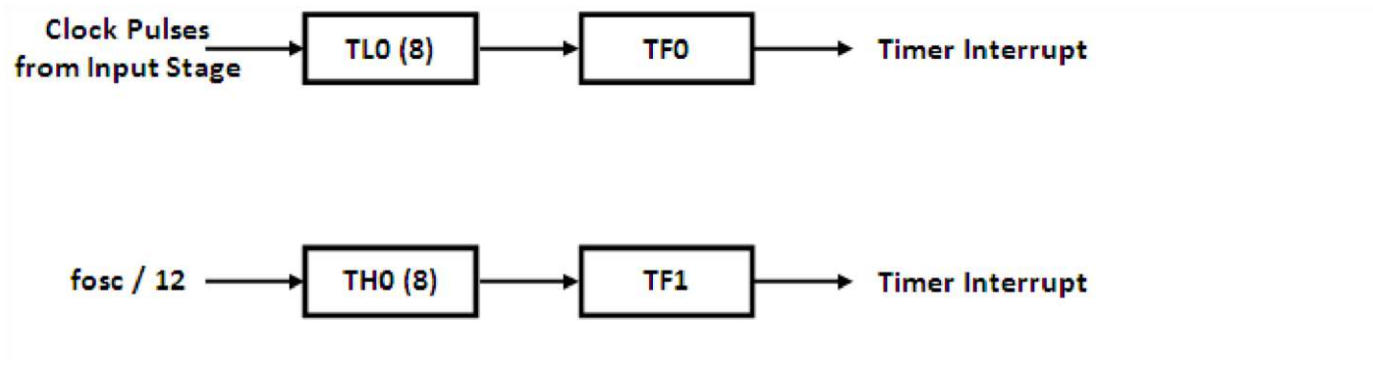
c) Timer Mode 2 (Auto reload TL from TH)



- TLX is used as an 8-bit counter.
- THX holds the count value to be reloaded.
- On each count TLX increments.
- When TLX rolls-over (i.e. from FFH to 00H), the following events take place:
  1. Timer overflow flag TFX is set, hence timer interrupt occurs.
  2. The value of THX is copied into TLX. Hence TLX is auto-reloaded form THX, and the process repeats.

- Thus the timer interrupt occurs at regular intervals "Continuously".
- This mode is used to generate a desired frequency using the Timer Flag. Max Count è 2^8 = 256 (FFH). Hence Max Delay è 256(12/f).

d) Timer Mode 3 (Two 8-bit Timers Using Timer0)



- Timer 0 is used as 2 separate 8-bit timers TH0 and TL0.
- TL0 uses the control bits (TR0 and TF0) of Timer 0.
- It can work as a Timer or a Counter.
- TH0 uses the control bits (TR1 and TF1) of Timer 1.

- It can work only as a Timer. #please refer Bharat Sir's Lecture Notes for this...
- Timer 1 can be in Mode 0, Mode 1, or Mode 2, but will not generate an interrupt.

6.11 8051 TIMER/COUNTER (HARDWARE DELAY) PROGRAMMING:

Example 1:

- **WAP to generate a delay of 20 µsec using internal timer-0 of 8051. After the delay send a "1" through Port3.1. Assume Suitable Crystal Frequency**

NOTE: In 8051, if we select a Crystal of 12 MHz, then Timer freq will be f_{osc}/12 è 1MHz. Hence each count will require 1/1MHz è 1 µsec. Thus for 20 µsec, the Desired Count will be 20 14H. For an Up-Counter (Mode 1):
Count = Max Count – Desired Count + 1 Count = FFFF – 14 + 1
<u>Count = FFECH</u> #Please refer Bharat Sir's Lecture Notes for this...

```
SOLN: MOV TMOD, #01H     ;   Program TMOD→(0000 0001)2 ...Timer0 Mode1
      MOV TL0, #0ECH     ;   Load lower byte of Count
      MOV TH0, #0FFH     ;   Load upper byte of Count
      MOV TCON, #10H     ;   Program TCON→ (0001 0000)2...start Timer0
WAIT: JNB TCON.5, WAIT   ;   Wait for overflow
      SETB P3.1          ;   Send a "1" through Port3.1
      MOV TCON, #00H     ;   Stop Timer0
HERE: SJMP HERE          ;   End of program
```

Example 2:

- **WAP to generate a Square wave of 1 KHz from the TxD pin of 8051, Q11 usingTimer1. Assume Clock Frequency of 12 MHz**

NOTE: For a Square wave of 1 KHz, the delay required is .5 msec.
We know, each count will require 1/1MHz è 1 µsec.
Thus for 500 µsec, the Desired Count will be 500_d è 01F4H. For an Up-Counter (Mode 1):
Count = Max Count – Desired Count + 1
Count = FFFF – 01F4 + 1
Count = FE0CH

```
SOLN: CLR P3.1               ;    Clear Txd Line initially
MOV TMOD, #10H               ;    Program TMOD→(0001 0000)2...Timer1 Mode1
REPEAT: MOV TL1, #0CH        ;    Load lower byte of Count
MOV TH1, #0FEH               ;    Load upper byte of Count
MOV TCON, #40H               ;    Program TCON→(0100 0000)2…start Timer1
  WAIT: JNB TCON.7, WAIT     ;    Wait for overflow
        CPL P3.1             ;    Toggle Txd pin after the delay
MOV TCON, #00H               ;    Stop Timer1
        SJMP REPEAT          ;    Repeat the process
```

Example 3:

- **WAP to generate a Rectangular wave of 1 KHz, having a 25% Duty Cycle from the TxD pin of 8051, using Timer1. Assume XTAL of 12 MHz**

NOTE: For a Rectangular wave of 1 KHz, having 25% Duty Cycle: T_{ON} = 250 µsec; T_{OFF} = 750 µsec.

For T_{ON}: Desired Count = 250_d è 00FAH
Count $_{ON}$ = Max Count – Desired Count + 1

Count $_{ON}$ = FFFF – 00FA + 1
Count $_{ON}$ = FF06H

For T$_{OFF}$: Desired Count = 750 $_d$ è 02EEH
Count $_{OFF}$ = Max Count – Desired Count + 1
Count $_{OFF}$ = FFFF – 02EE + 1
Count $_{OFF}$ = FD12H

```
  SOLN: MOV TMOD, #10H      ;    Program TMOD→ (0001
                                 0000)2...Timer1 Mode1


REPEAT: MOV TL1, #06H       ;    Load lower byte of
                                 Count ON
 MOV TH1, #0FFH             ;    Load upper byte of
                                 Count ON
       SETB P3.1            ;    Display"1"at Txd
 MOV TCON, #40H             ;    Program TCON
                                 (0100 0000)2...
                                 startTimer1
    ON: JNB TCON.7, ON      ;    Maintain"1" at Txd
       CLR P3.1             ;    Clear Txd
 MOV TCON, #00H             ;    Stop Timer1

 MOV TL1, #12H              ;    Load lower byte of Count
 MOV TH1, #0FDH             ;    Load upper byte of
                                 Count OFF
 MOV TCON, #40H             ;    Program TCON (0100
                                 0000)2...start Timer1
   OFF: JNB TCON.7, OFF     ;    Maintain "0" at Txd
 MOV TCON, #00H             ;    Stop Timer1

       SJMP REPEAT          ;    Repeat the process
```

Note: If System Freq = 12MHz, it is clear that 1 Count requires 1 msec.
In Mode 1, we have a 16bit Count.
Hence max pulses that can be desired is 2^{16} = 65536.
Count = Max Count – Desired Count + 1
= 65535 – 65536 + 1

= 0.
Thus we will get max delay if we load the count as 0000H, as it will have to "roll-over" back to 0000H to overflow.
Hence Max delay if XTAL is of 12 MHz … is 65536 µsec è **65.536 msec.**
Similarly Max delay if XTAL is of 11.0592 MHz … is 71106 µsec è **71.106 msec.**

Example 4:

- **WAP to generate a delay of <u>1 SECOND</u> using Timer1. Assume Clock Frequency of 12 MHz (Popular Question in College!)**

NOTE: Max delay if XTAL is of 12 MHz … is 65536 µsec è 65.536 msec. Hence to get a delay of 1 second, we will have to perform the counting repeatedly in a loop.
Let's keep the Desired Count 50000. (50 msec delay)
Now 50000_d = C350H
Count = Max Count – Desired Count + 1 Count = FFFF – C350 + 1
<u>Count = 3CB0H</u>
We will have to perform this counting 1sec/50msec times è **<u>20 times</u>**

```
SOLN: MOV TMOD, #10H        ;   Program TMOD→ (0001 0000)2
                                Timer1 Mode1
 MOV R0, #14H               ;   Load count 20 in R0
REPEAT: MOV TL1, #0B0H      ;   Load lower byte of Count ON
 MOV TH1, #3CH              ;   Load upper byte of Count ON
 MOV TCON, #40H             ;   Program TCON (0100 0000)2…start
                                Timer1
   WAIT: JNB TCON.1, WAIT   ;   Wait for an overflow
 MOV TCON, #00H             ;   Stop Timer1
 DJNZ R0, REPEAT            ;   repeat the process 20 times
   HERE: SJMP HERE:         ;   End of program
```

Example 5:

- **WAP to read the data from Port1, 10 times, each after a 1 sec delay. Store the data from RAM locations 20H onwards. When the operation is complete, ring an "Alarm" connected at Port3.1. Assume CLK = 12 MHz**

NOTE: As seen from the previous program, for a delay of 1 second, we have **<u>Count = 3CB0H.</u>** Counting has to be performed **<u>20 times.</u>** Also note that all ports of 8051 are o/p ports by default. To program a port as i/p ports, **<u>all "1"s</u>** must be sent through it.

```
  SOLN: CLR P3.1           ;   Clear  Port3.1    line

 MOV TMOD, #10H            ;   Program TMOD→(0001
                               0000)2...Timer1Mode1
 MOV 90H, #0FFH            ;   Program Port1      as
                               i/p by sending all "1"s
                               through it

 REPEAT: MOV R0, #0AH      ;   Load  Data  Count of
                               10 in R0

 MOV R1, #20H              ;   Load Storage address
                               in R1
 MOV @R1, 90H              ;   Read  data  from  Port
 INC R1                    ;   Increment  data  storage
                               address from next
                               Iteration
ACALL DELAY                ;   Call delay of 1 sec before
                               going into next
                               Iteration
DJNZ R0, REPEAT            ;   Repeat till all 10 bytes
                               are read
        SETB P3.1          ;   Ring "Alarm" at Port3.1
  HERE: SJMP HERE:         ;   End of program

 DELAY: MOV R2, #14H       ;   Load count 20 in R0
REPEAT: MOV TL1, #0B0H     ;   Load  lower byte   of
                               Count ON
MOV TH1, #3CH              ;   Load upper byte    of
                               Count ON
  MOV TCON, #40H              ;       Program TCON
                                      (0100 0000)...start Timer1

 WAIT: JNB TCON.1, WAIT        ;       Wait for an overflow
     MOV TCON, #00H          ;       Stop Timer1
     DJNZ R2, REPEAT         ;       Repeat the process 20
                             times
     RET                     ;       End of delay routine
```

6.12 ADDRESSING MODES OF 8051:

Addressing Modes is the manner in which operands are given in the instruction. 8051 supports the following 5 addressing modes:

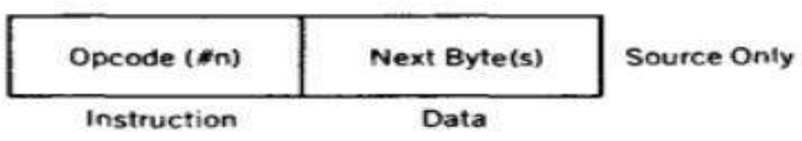
1. Immediate addressing mode
2. Register addressing mode
3. Direct addressing mode
4. Indirect addressing mode
5. Index addressing mode

1. <u>IMMEDIATE ADDRESSING MODE</u>

- In this addressing mode, the Data is given in the Instruction itself.
- We put a "#" symbol, before the data, to identify it as a data value and not as an address.
- **Example**

  MOV A, #35H ; A🡐35H

  MOV DPTR, #3000H; DPTR🡐 3000H



Immediate Addressing Mode

2. <u>REGISTER ADDRESSING MODE</u>

- In this addressing mode, Data is given by a Register in the instruction.
- The permitted registers are A, R7 … R0 of each memory bank.

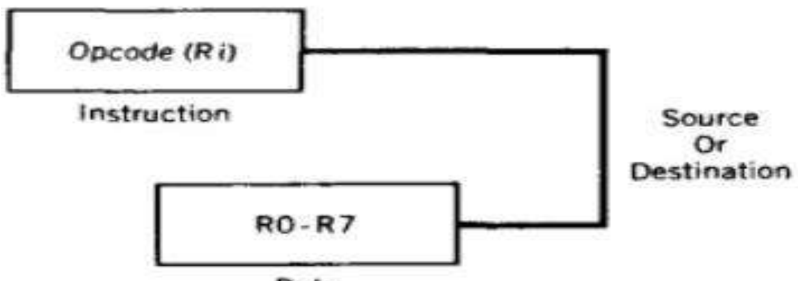
- Note: Data transfer between two RAM registers is not allowed.
- **Example**

MOV A, R0 ; A← R0 … If R0 = 25H, then A gets the Value 25H.

MOV R5, A ; R5← A

MOV Rx, Ry ; NOT ALLOWED. That's because this would allow 64 combinations of register.

; As registers invite opcodes, this would need 64 opcodes!



Register Addressing Mode

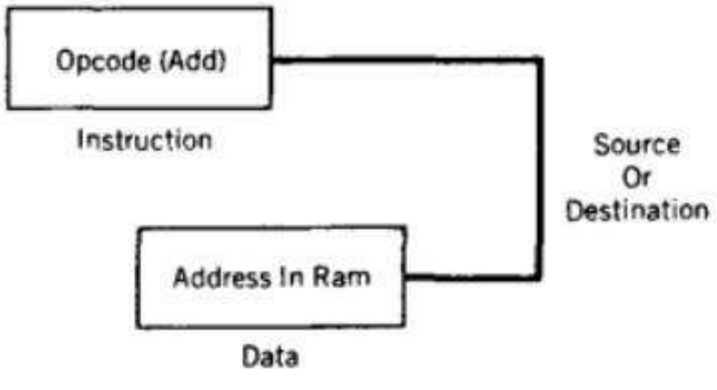
3. DIRECT ADDRESSING MODE

- Here, the address of the operand is given in the instruction.
- Only Internal RAM addresses (00H…7FH) and SFR addresses (from 80H to FFH) allowed.

- **Example**

MOV A, 35 ; A←Contents of RAM location 35H
MOV A, 80H ; A←contents of port 0 (SFR at address 80H)
MOV 20H, 30H ; [20H]←[30H]
i.e. Location 20H gets the contents of location

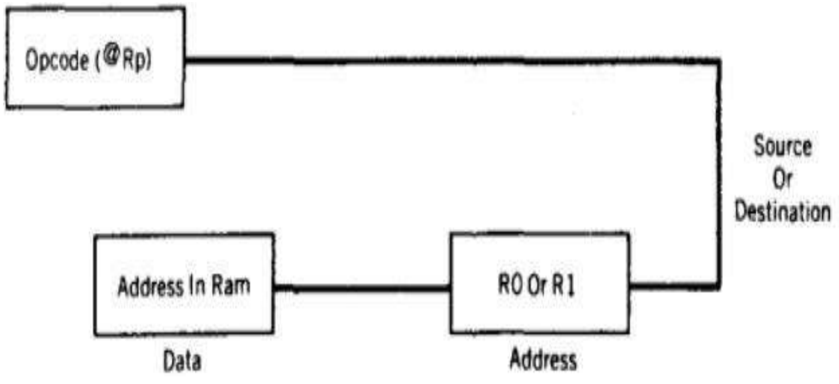
30H.



Direct Addressing Mode

4. INDIRECT ADDRESSING MODE

- Here, the address of the operand is given in a register.
- Internal RAM and External RAM can be accessed using this mode.
- The advantage of giving an address using a register is that we can increment the address in a loop, by simply incrementing the register, and hence access a series of locations.



Indirect Addressing Mode

INTERNAL RAM: (8-BIT ADDRESS GIVEN BY R0 OR R1):

- ONLY R1 or R0, called as Data Pointers, can be used to specify address (00H ... 7FH).
- A "@" sign is present before the register to indicate that the register is giving an address.

- **Example:**

  MOV A, @R0 ; A← [R0]

; i.e. **A**←Contents of Internal RAM Location whose address is given by R0.

; if R0 = 25H, then A gets the contents of Location 25H from Internal RAM.

MOV @R1, A ; [R1]← A

; i.e. Internal RAM Location pointed by R1 gets value of A.

EXTERNAL RAM: (16 BIT ADDRESS GIVEN BY DPTR):

- For the External RAM, address is provided by R1 or R0, or by DPTR.
- If DPTR is used to give an address, then the full 64KB range of External RAM from 0000H… FFFFH is available. This is because DPTR is 16-bit and 2^{16} = 65536.
- An "X" is present in the instruction, to indicate External RAM.

- **Example**

MOVX A, @DPTR ; A ←[DPTR] ^

; A gets the contents of External RAM location whose address is given by DPTR.

; If DPTR=2000H, then A gets contents of location 0025H from the external RAM

MOVX @DPTR, A ; [DPTR] ^←A

; i.e. A is stored at the External RAM location whose address is given by DPTR.

EXTERNAL RAM: (8 BIT ADDRESS GIVEN BY R0 OR R1):

- If R0 or R1 is used to give an address, then only the first 256 locations of External RAM is available from 0000 H to 00FF H.
- This is because R0 or R1 are 8-bit and 2^8 = only 256.
- Example

MOVX A,@R0 ; A← [R0] ^

; i.e. **A** gets the contents of External RAM Location whose address is given by R0.

; If R0 = 25H, then A gets contents of Location 0025H from the External RAM

MOVX @R1, A ; [R1] ^←A

; i.e. **A** is stored at the External RAM Location whose address is given by R1

5. INDEXED ADDRESSING MODE

- This mode is used to access data from the Code memory (Internal ROM or External ROM).
- In this addressing mode, address is indirectly specified as a "SUM" of (A and DPTR) or (A and PC).
- This is very useful because ROM contains permanent data which is stored in the form of Look Up tables.
- To access a Look Up table, address is given as a SUM or two registers, where one acts as the base and the other acts as the index within the table.
- A "C" is present in such instructions, to indicate Code Memory.

- **Example**

MOVC A, @A+DPTR; A← Contents of a ROM Location pointed by A+DPTR.

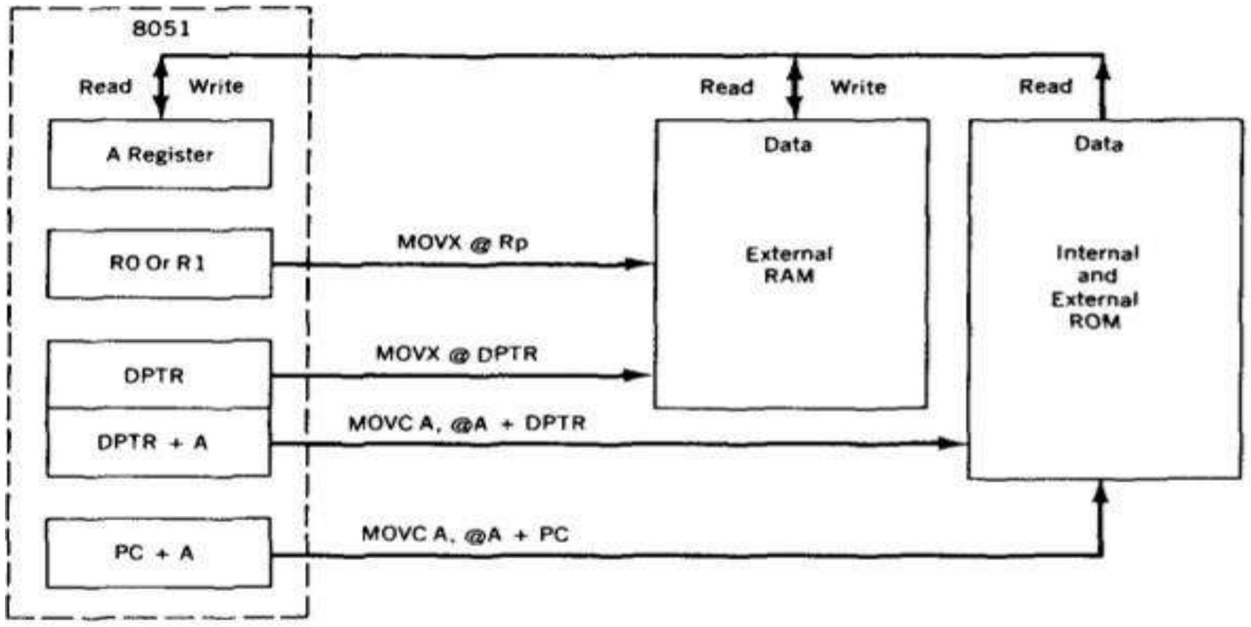
; If DPTR = 0400H and A = 05H,

; Then **A** gets the contents of ROM Location whose address is 0405 H.

MOVC A, @A+PC ; A← Contents of a ROM Location pointed by A+PC.

- The same instruction may operate on Internal or External ROM, depending upon the address and on the value of **EA** pin of 8051.
- If the address is in the range of 0000… 0FFFH, then **EA** pin will decide if it operates on Internal
- ROM or External ROM. IF **EA** = 0, External ROM else Internal ROM. If Address is 1000H and more, it will certainly be External ROM.

External Addressing using MOVX and MOVC



6.13 8051 ASSEMBLY LANGUAGE PROGRAMMING:

1. **Write a program to add the values of locations 50H and 51H and store the result in locations in 52h and 53H.**

```
        ORG 0000H              ; Set program counter 0000H
        MOV A,50H              ; Load the contents of Memory location 50H into A
        ADD A,51H              ; Add the contents of memory 51H with contents A
        MOV 52H,A              ; Save the LS byte of the result in 52H
        MOV A, #00             ; Load 00H into A
        ADDC A, #00            ; Add the immediate data and carry to A
        MOV 53H,A              ; Save the MS byte of the result in location 53h
        END
```

2. **Write a program to store data FFH into RAM memory locations 50H to 58H using direct addressing mode**

```
 ORG 0000H                          ; Set program counter 0000H
MOV A, #0FFH                        ; Load FFH into A
 MOV 50H, A     ; Store contents of A in location 50H
 MOV 51H, A     ; Store contents of A in location 5IH
 MOV 52H, A     ; Store contents of A in location 52H
 MOV 53H, A     ; Store contents of A in location 53H
 MOV 54H, A     ; Store contents of A in location 54H
 MOV 55H, A     ; Store contents of A in location 55H
 MOV 56H, A     ; Store contents of A in location 56H
 MOV 57H, A     ; Store contents of A in location 57H
 MOV 58H, A     ; Store contents of A in location 58H
 END
```

3. Write a program to subtract a 16 bit number stored at locations 51H-52H from 55H-56H and store the result in locations 40H and 41H. Assume that the least significant byte of data or the result is stored in low address. If the result is positive, then store 00H, else store 01H in 42H.

```
            ORG 0000H                        ; Set program counter 0000H
           MOV A, 55H                       ; Load the contents of memory location 55 into
A
           CLR C                             ; Clear the borrow flag
           SUBB A,51H                        ; Sub the contents of memory 51H from
contents of A
             MOV 40H, A                       ; Save the LS Byte of the result in location
40H
           MOV A, 56H                         ; Load the contents of memory location
56H into A                    SUBB A, 52H                        ; Subtract the content
of memory 52H from the
                                              Content A.
           MOV 41H,                          ;Save the MS byte of the result in location
415.
           MOV A, #00                        ; Load 005 into A
           ADDC A, #00                        ; Add the immediate data and the carry flag
to A
            MOV 42H, A                         ; If result is positive, store00H, else store
0lH in 42H
            END
```

4. Write a program to add two 16 bit numbers stored at locations 51H-52H and 55H-56H and store the result in locations 40H, 41H and 42H. Assume that the least significant byte of data and the result is stored in low address and the most significant byte of data or the result is stored in high address.

```
        ORG 0000H                  ; Set program counter 0000H
        MOV A, 51H                 ; Load the contents of memory location 51H
into A
        ADD A, 55H                 ; add the contents of 55H with contents of A
        MOV 40H, A                 ; Save the LS byte of the result in location
40H
        MOV A, 52H                 ; Load the contents of 52H into A
        ADDC A, 56H                ; Add the contents of 56H and CY flag with A
        MOV 41H, A                 ; Save the second byte of the result in 41H
        MOV A, #00                 ; Load 00H into A
        ADDC A, #00                ; Add the immediate data 00H and CY to A
        MOV 42H, A                 ; Save the MS byte of the result in location
42H
        END
```

5. **Write a program to store data FFH into RAM memory locations 50H to 58H using indirect addressing mode.**

```
            ORG 0000H       ; Set program counter 0000H
            MOV A, #0FFH    ; Load FFH into A
            MOV RO, #50H    ; Load pointer, R0-50H
            MOV R5, #08H    ; Load counter, R5-08H
Start:      MOV @RO, A      ; Copy contents of A to RAM pointed by R0
            INC RO          ; Increment pointer
            DJNZ R5, start  ; Repeat until R5 is zero
            END
```

6. **Write a program to add two Binary Coded Decimal (BCD) numbers stored at locations 60H and 61H and store the result in BCD at memory locations 52H and 53H. Assume that the least significant byte of the result is stored in low address.**

```
ORG 0000H        ; Set program counter 00004
MOV A, 60H       ; Load the contents of memory location 6.0.H into A
```

```
        ADD A, 61H          ; Add the contents of memory location 61H with
contents of A
       DA A                ; Decimal adjustment of the sum in A
        MOV 52H, A          ; Save the least significant byte of the result in location
52H
       MOV A, #00          ; Load 00H into .A
       ADDC A, #00H         ; Add the immediate data and the contents of carry
flag to A

       MOV 53H, A           ; Save the most significant byte of the result in
location 53
       END
```

7. **Write a program to clear 10 RAM locations starting at RAM address 1000H.**

```
        ORG 0000H                  ; Set program counter 0000H
        MOV DPTR, #1000H       ; Copy address 1000H to DPTR
        CLR A                      ; Clear A
        MOV R6, #0AH      ; Load 0AH to R6 again:
        MOVX @DPTR, A              ; Clear RAM location pointed by DPTR
        INC DPTR                   ; Increment DPTR
         DJNZ R6, again             ; Loop until counter R6=0
         END
```

8. **Write a program to compute 1 + 2 + 3 + N (say N=15) and save the sum at70H**

```
              ORG 0000H               ; Set program counter 0000H
             N EQU 15
              MOV R0, #00              ; Clear R0
             CLR A                    ; Clear A
Again:       INC R0                   ; Increment R0
              ADD A, R0                ; Add the contents of R0 with A
             CJNE R0, #N, again     ; Loop until counter, R0, N
              MOV 70H, A               ; Save the result in location 70H END
```

9. **Write a program to multiply two 8 bit numbers stored at locations 70H and 71H and store the result at memory locations 52H and 53H.**

Assume that the least significant byte of the result is stored in low address.

```
ORG 0000H                ; Set program counter 00 OH
MOV A, 70H               ; Load the contents of memory location 70h into A
MOV B, 71H               ; Load the contents of memory location 71H into B
MUL AB                   ; Perform multiplication
MOV 52H, A                  ; Save the least significant byte of the result in
location 52H MOV 53H, B              ; Save the most significant byte of the
result in location 53
END
```

10.Write a program to exchange the lower nibble of data present in external memory 6000H and 6001H

```
ORG 0000H                ; Set program counter 00h
MOV DPTR, #6000H         ; Copy address 6000H to DPTR
MOVX A, @DPTR                 ; Copy contents of 60008 to A
MOV R0, #45H                  ; Load pointer, R0=45H
MOV @R0, A                    ; load pointer, r0=45H
INC DPL                       ; increment pointer
MOVX A, @DPTR                 ; Copy contents of 60018 to A
                         XCHD A, @R0
MOVX @DPTR, A                 ; Copy contents of A to 6001 8
DEC DPL                       ; Decrement pointer

MOV A, @R0                    ; Copy contents of  RAM pointed by R
                         to A
MOVX @DPTR, A                 ; Copy contents of A to RAM pointed
                         by DPTR
END
```

11. Write a program to count the number of and o's of 8 bit data stored in location 6000H.

```
ORG 00008                ; Set program counter 00008
MOV DPTR, #6000h         ; Copy address 6000H to DPTR
MOVX A, @DPTR             ; Copy number to A
MOV R0, #08              ; Copy 08 in RO
MOV R2, #00              ; Copy 00 in R2
```

```
                 MOV R3, #00                    ; Copy 00 in R3
               CLR C                            ; Clear carry flag
       BACK:   RLC A                          ; Rotate A through carry flag
               JC NEXT                       ; If CF=1, branch to next
               INC R2                           ; If CF=0, increment R2 AJMP NEXT2
                 NEXT:  INC R3                  ; If CF=1, increment R3
               NEXT2: DJNZ RO, BACK   ; Repeat until R0 is zero
               END
```

12.Write a program to shift a 24 bit number stored at 57H-55H to the left Logically four Places.
Assume that the least significant byte of data is stored in lower address.

```
                 ORG 0000H              ; Set program counter 0000h
                 MOV R1, #04            ; Set up loop count to 4
Again:          MOV A, 55H            ; Place the least significant byte of data in A
                CLR C                   ; Clear the carry flag
                  RLC A                  ; Rotate contents of A (55H) left through carry
                MOV 55H, A
                MOV A, 56H
                  RLC A                  ; Rotate contents of A (56H) left through carry
                MOV 56H, A
                MOV A, 57H
                  RLC A                  ; Rotate contents of A (57H) left through carry
                MOV 57H, A
                DJNZ R1, again        ; Repeat until R1 is zero
                END
```

6.14 SERIAL COMMUNICATION:

DATA COMMUNICATION:

- The 8051 microcontroller is parallel device that transfers eight bits of data simultaneously over eight data lines to parallel I/O devices.
- Parallel data transfer over a long is very expensive. Hence, a serial communication is widely used in long distance communication.
- In serial data communication, 8-bit data is converted to serial bits using a parallel in serial out shift register and then it is transmitted over a single data line.

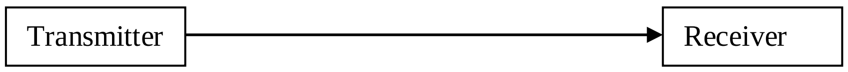
- The data byte is always transmitted with least significant bit first.

BASICS OF SERIAL DATA COMMUNICATION,

Communication Links

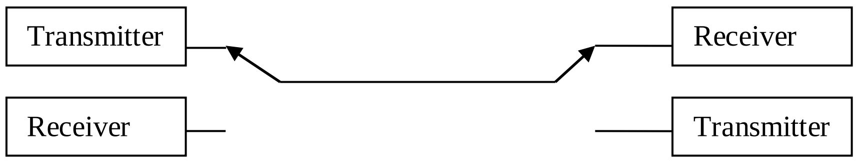
1. **Simplex communication link:**

- In simplex transmission, the line is dedicated for transmission. The transmitter sends and the receiver receives the data.

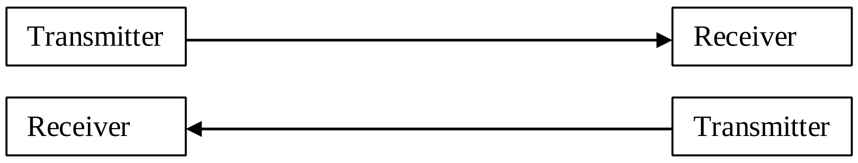


2. **Half duplex communication link:**

- In half duplex, the communication link can be used for either transmission or reception. Data is transmitted in only one direction at a time.



3. **Full duplex communication link:**
    - If the data is transmitted in both ways at the same time, it is a full duplex i.e. transmission and reception can proceed simultaneously. This communication link requires two wires for data, one for transmission and one for reception.



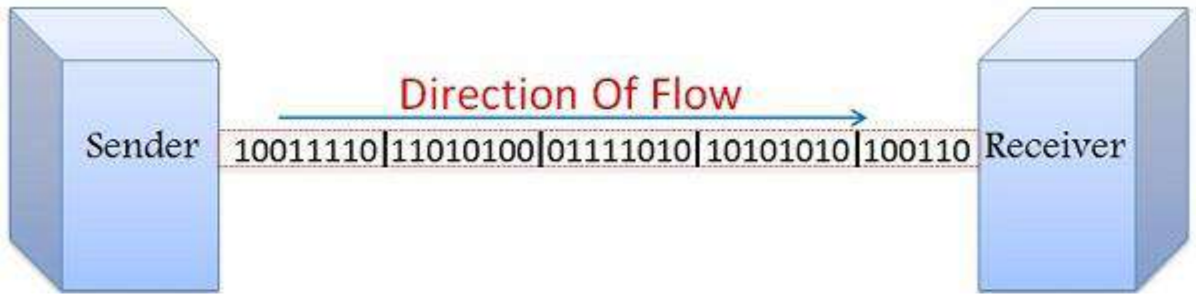
Types of Serial communication:

Serial data communication uses two types of communication.

1. Synchronous serial data communication:

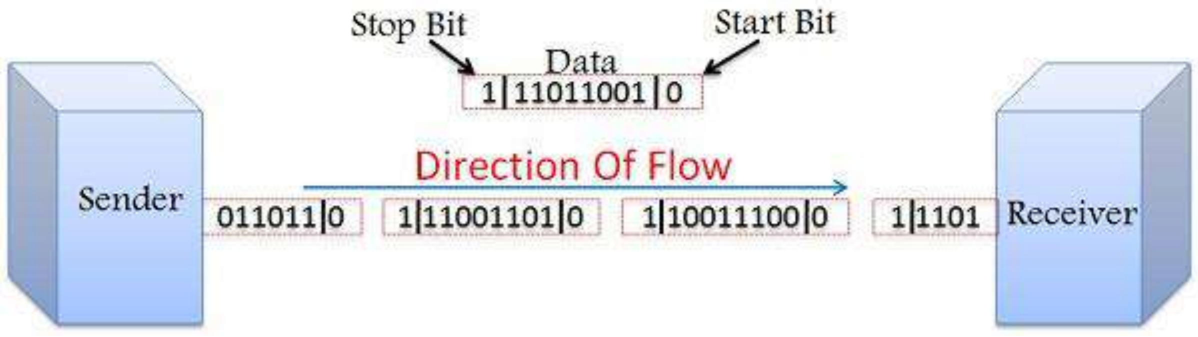
- In this transmitter and receiver are synchronized.
- It uses a common clock to synchronize the receiver and the transmitter.
- First the synch character is sent and then the data is transmitted.
- This format is generally used for high speed transmission.

- In Synchronous serial data communication a block of data is transmitted at a time.



2. Asynchronous Serial data transmission:

- In this, different clock sources are used for transmitter and receiver.
- In this mode, data is transmitted with start and stop bits.
- A transmission begins with start bit, followed by data and then stop bit.
- For error checking purpose parity bit is included just prior to stop bit.
- In Asynchronous serial data communication a single byte is transmitted at a time.



Baud rate:

- The rate at which the data is transmitted is called baud or transfer rate.
- The baud rate is the reciprocal of the time to send one bit.
- In asynchronous transmission, baud rate is not equal to number of bits per second.
- This is because; each byte is preceded by a start bit and followed by parity and stop bit.
- **For example**, in synchronous transmission, if data is transmitted with 9600 baud_it means that 9600 bits are transmitted in one second.
- For bit transmission time = 1 second/ 9600 = 0.104 ms.

8051 SERIAL COMMUNICATION:

The 8051 supports a full duplex serial port.

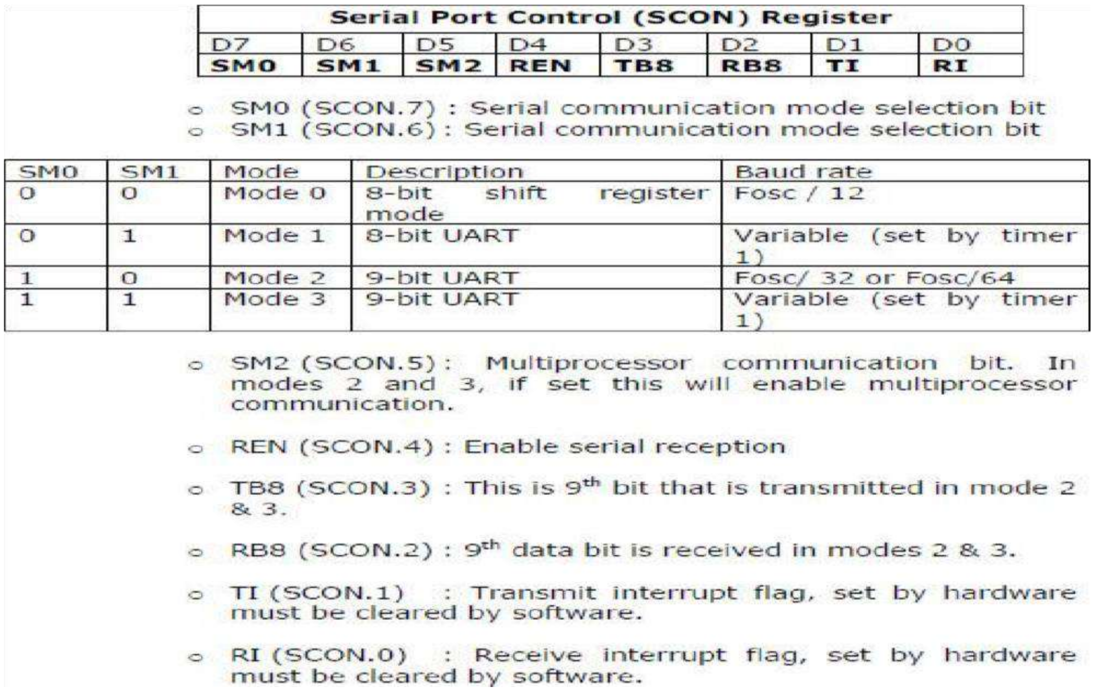
Three special function registers support serial communication.

1. **SBUF Register:**

   Serial Buffer (SBUF) register is an 8-bit register. It has separate SBUF registers for data transmission and for data reception. For a byte of data to be transferred via the TXD line, it must be placed in SBUF register. Similarly, SBUF holds the 8-bit data received by the RXD pin and read to accept the received data.

2. **SCON register:**

   The contents of the Serial Control (SCON) register are shown below. This register contains mode selection bits, serial port interrupt bit (TI and RI) and also the ninth data bit for transmission and reception (TB8 and RB8).



Serial Port Control (SCON) Register							
D7	D6	D5	D4	D3	D2	D1	D0
SM0	**SM1**	**SM2**	**REN**	**TB8**	**RB8**	**TI**	**RI**

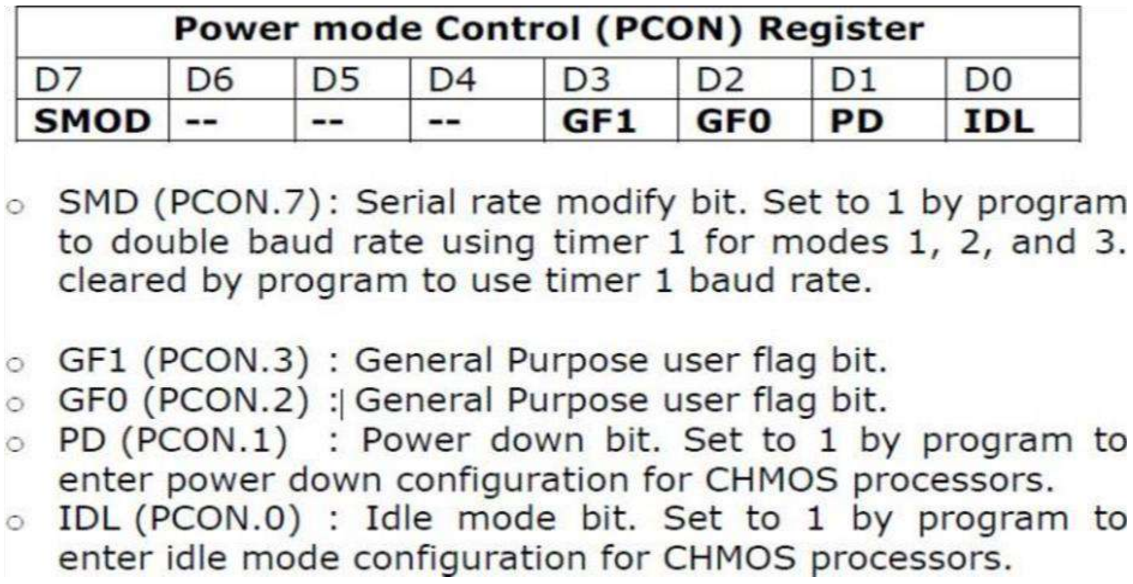
- SM0 (SCON.7) : Serial communication mode selection bit
- SM1 (SCON.6) : Serial communication mode selection bit

SM0	SM1	Mode	Description	Baud rate
0	0	Mode 0	8-bit shift register mode	Fosc / 12
0	1	Mode 1	8-bit UART	Variable (set by timer 1)
1	0	Mode 2	9-bit UART	Fosc/ 32 or Fosc/64
1	1	Mode 3	9-bit UART	Variable (set by timer 1)

- SM2 (SCON.5) : Multiprocessor communication bit. In modes 2 and 3, if set this will enable multiprocessor communication.
- REN (SCON.4) : Enable serial reception
- TB8 (SCON.3) : This is 9th bit that is transmitted in mode 2 & 3.
- RB8 (SCON.2) : 9th data bit is received in modes 2 & 3.
- TI (SCON.1) : Transmit interrupt flag, set by hardware must be cleared by software.
- RI (SCON.0) : Receive interrupt flag, set by hardware must be cleared by software.

3. **PCON register:**

   The SMOD bit (bit 7) of PCON register controls the baud rate in asynchronous mode transmission.



Power mode Control (PCON) Register							
D7	D6	D5	D4	D3	D2	D1	D0
SMOD	**--**	**--**	**--**	**GF1**	**GF0**	**PD**	**IDL**

- SMD (PCON.7): Serial rate modify bit. Set to 1 by program to double baud rate using timer 1 for modes 1, 2, and 3. cleared by program to use timer 1 baud rate.

- GF1 (PCON.3) : General Purpose user flag bit.
- GF0 (PCON.2) :| General Purpose user flag bit.
- PD (PCON.1) : Power down bit. Set to 1 by program to enter power down configuration for CHMOS processors.
- IDL (PCON.0) : Idle mode bit. Set to 1 by program to enter idle mode configuration for CHMOS processors.

SERIAL COMMUNICATION MODES:

1. Mode 0

In this mode serial port runs in synchronous mode. The data is transmitted and received through RXD pin and TXD is used for clock output. In this mode the baud rate is 1/12 of clock frequency.

2. Mode 1

In this mode SBUF becomes a 10 bit full duplex transceiver. The ten bits are 1 start bit, 8 data bit and 1 stop bit. The interrupt flag TI/RI will be set once transmission or reception is over. In this mode the baud rate is variable and is determined by the timer 1 overflow rate.

Baud rate = $[2^{smod}/32]$ x Timer 1 overflow Rate

= $[2^{smod}/32]$ x [Oscillator Clock Frequency] / [12 x [256 – [TH1]]]

3. Mode 2

This is similar to mode 1 except 11 bits are transmitted or received. The 11 bits are, 1 start bit, 8 data bit, a programmable 9th data bit, 1 stop bit.

Baud rate = $[2^{smod}/64]$ x Oscillator Clock Frequency

4. Mode 3

This is similar to mode 2 except baud rate is calculated as in mode 1

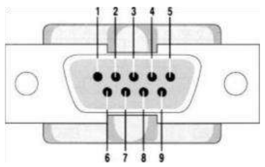
CONNECTIONS TO RS-232

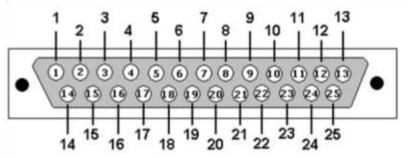
RS-232 standards:

To allow compatibility among data communication equipment made by various manufactures, and interfacing standard called RS232 was set by the Electronics Industries Association (EIA) in 1960. Since the standard was set long before the advent of logic family, its input and output voltage levels are not TTL compatible.

In RS232, a logic one (1) is represented by -3 to -25V and referred as MARK while logic zero (0) is represented by +3 to +25V and referred as SPACE. For this reason to connect any RS232 to a microcontroller system we must use voltage converters such as MAX232 to convert the TTL logic level to RS232 voltage levels and vice-versa. MAX232 IC chips are commonly referred as line drivers.

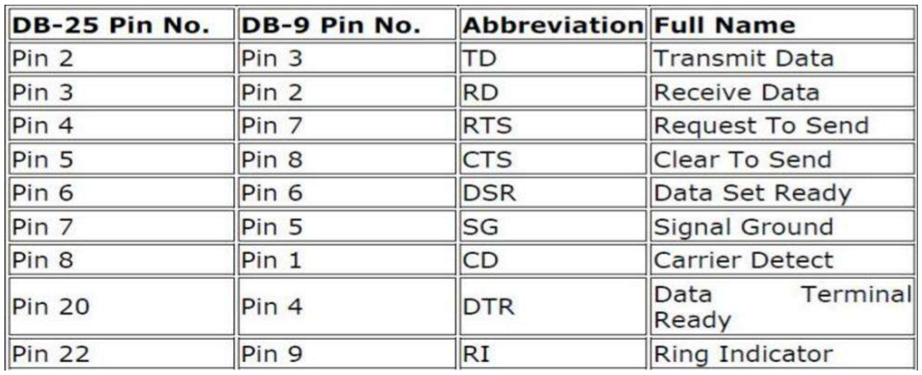
In RS232 standard we use two types of connectors. DB9 connector or DB25 connector.





DB9 Male Connector   DB25 Male Connector

The pin description of DB9 and DB25 Connectors are as follows



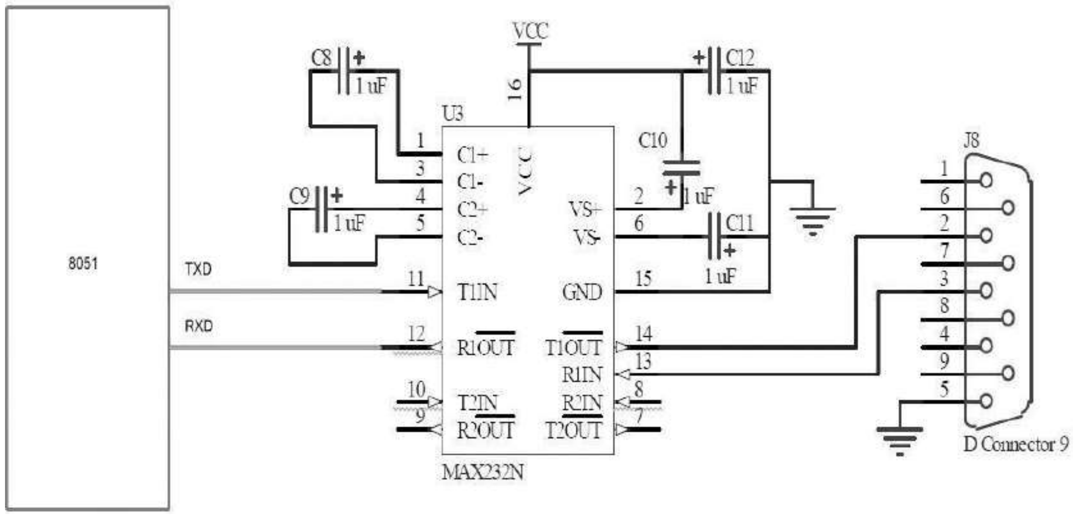
DB-25 Pin No.	DB-9 Pin No.	Abbreviation	Full Name
Pin 2	Pin 3	TD	Transmit Data
Pin 3	Pin 2	RD	Receive Data
Pin 4	Pin 7	RTS	Request To Send
Pin 5	Pin 8	CTS	Clear To Send
Pin 6	Pin 6	DSR	Data Set Ready
Pin 7	Pin 5	SG	Signal Ground
Pin 8	Pin 1	CD	Carrier Detect
Pin 20	Pin 4	DTR	Data Terminal Ready
Pin 22	Pin 9	RI	Ring Indicator

The 8051 connection to MAX232 is as follows:

- The 8051 has two pins that are used specifically for transferring and receiving data serially.
- These two pins are called TXD, RXD. Pin 11 of the 8051 (P3.1) assigned to TXD and pin 10 (P3.0) is designated as RXD. These pins TTL compatible; therefore they require line driver (MAX 232) to make them RS232

compatible.

- MAX 232 converts RS232 voltage levels to TTL voltage levels and vice versa.
- One advantage of the MAX232 is that it uses a +5V power source which is the same as the source voltage for the 8051.
- The typical connection diagram between MAX 232 and 8051 is shown below.



SERIAL COMMUNICATION PROGRAMMING IN ASSEMBLY AND C.

Steps to programming the 8051 to transfer data serially

1. The TMOD register is loaded with the value 20H, indicating the use of the Timer 1 in mode 2 (8-bit auto reload) to set the baud rate.
2. The TH1 is loaded with one of the values in table 5.1 to set the baud rate for serial data transfer.
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8-bit data is framed with start and stop bits.
4. TR1 is set to 1 start timer 1.
5. TI is cleared by the "CLR TI" instruction.
6. The character byte to be transferred serially is written into the SBUF register.
7. The TI flag bit is monitored with the use of the instruction JNB TI, target to see if the character has been transferred completely. 8. To transfer the next character, go to step 5.

Example 1. Write a program for the 8051 to transfer letter 'A' serially at 4800-baud rate, 8 bit data, and 1 stop bit continuously.

```
ORG 0000H
LJMP START
ORG 0030H
    START: MOV TMOD, #20H       ; select timer 1 mode 2
     MOV TH1, #0FAH             ; load count to get baud rate of 4800
     MOV SCON, #50H             ; initialize UART in mode 2
                                ; 8 bit data and 1 stop bit
      SETB TR1                  ; start timer
    AGAIN: MOV SBUF, #'A'       ; load char 'A' in SBUF
BACK: JNB TI, BACK          ; Check for transmit interrupt flag
      CLR TI                    ; Clear transmit interrupt flag
    SJMP AGAIN
    END
```

Example 2. Write a program for the 8051 to transfer the message 'EARTH' serially at 9600 baud, 8 bit data, and 1 stop bit continuously.

```
ORG 0000H
LJMP START


ORG 0030H
  START: MOV TMOD, #20H                                  ; select timer 1 mode 2
MOV TH1, #0FDH                 ; load count to get required baud rate of 9600 MOV
SCON, #50H                     ; initialize uart in mode 2
                                     ; 8 bit data and 1 stop bit
   SETB TR1                          ; start timer
  LOOP: MOV A, #'E'         ; load 1st letter 'E' in a
   ACALL LOAD                        ; call load subroutine
   MOV A, #'A'              ; load 2nd letter 'A' in a
   ACALL LOAD                        ; call load subroutine
   MOV A, #'R'              ; load 3rd letter 'R' in a
   ACALL LOAD                        ; call load subroutine
```

```
 MOV A, #'T'                         ; load 4th letter 'T' in a
 ACALL LOAD                          ; call load subroutine
 MOV A, #'H'                         ; load 4th letter 'H' in a
 ACALL LOAD                          ; call load subroutine
 SJMP LOOP                           ; repeat steps

LOAD:  MOV SBUF, A
HERE:  JNB TI, HERE                  ; Check for transmit interrupt flag
 CLR TI                 ; Clear transmit interrupt flag RET

  END
```

6.15 INTERFACING:

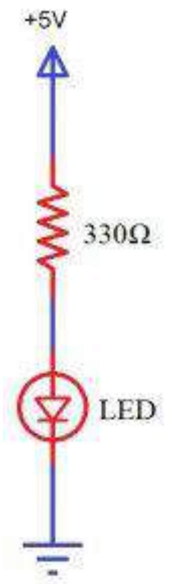
Interfacing is the process of connecting devices together so that they can exchange the information and that proves to be easier to write the programs. There are different type of input and output devices as for our requirement such as LEDs, LCDs, 7segment, keypad, motors and other devices.

Interfacing LED with 8051

- Light Emitting Diodes or LEDs are the mostly commonly used components in many applications. They are made of semiconducting material. In this project, I will describe about basics of Interfacing LED with 8051 Microcontroller.

- Light Emitting Diodes are the semiconductor light sources. Commonly used LEDs will have a cut-off voltage of 1.7V and current of 10mA. When an LED is applied with its required voltage and current it glows with full intensity.

- The Light Emitting Diode is similar to the normal PN diode but it emits energy in the form of light. The color of light depends on the band gap of the semiconductor. The following figure shows "how an LED glows?"



- Thus, LED is connected to the AT89C51 microcontroller with the help of a current limiting resistor. The value of this resistor is calculated using the following formula.

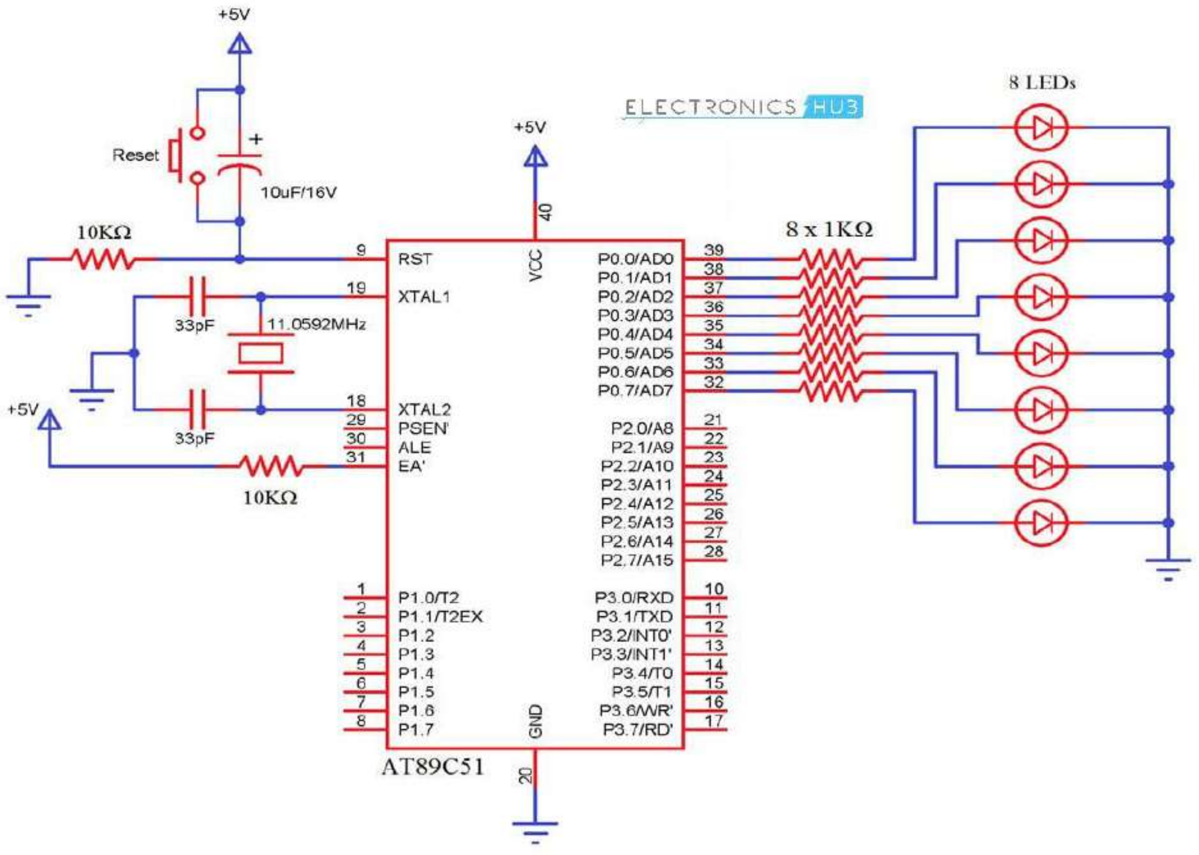
R= (V-1.7)/10mA, where V is the input voltage.

- Generally, microcontrollers output a maximum voltage of 5V. Thus, the value of resistor calculated for this is 330 Ohms. This resistor can be connected to either the cathode or the anode of the LED.

Principle behind Interfacing LED with 8051:

- The main principle of this circuit is to interface LEDs to the 8051 family micro controller. Commonly, used LEDs will have voltage drop of 1.7v and current of 10mA to glow at full intensity. This is applied through the output pin of the micro controller.

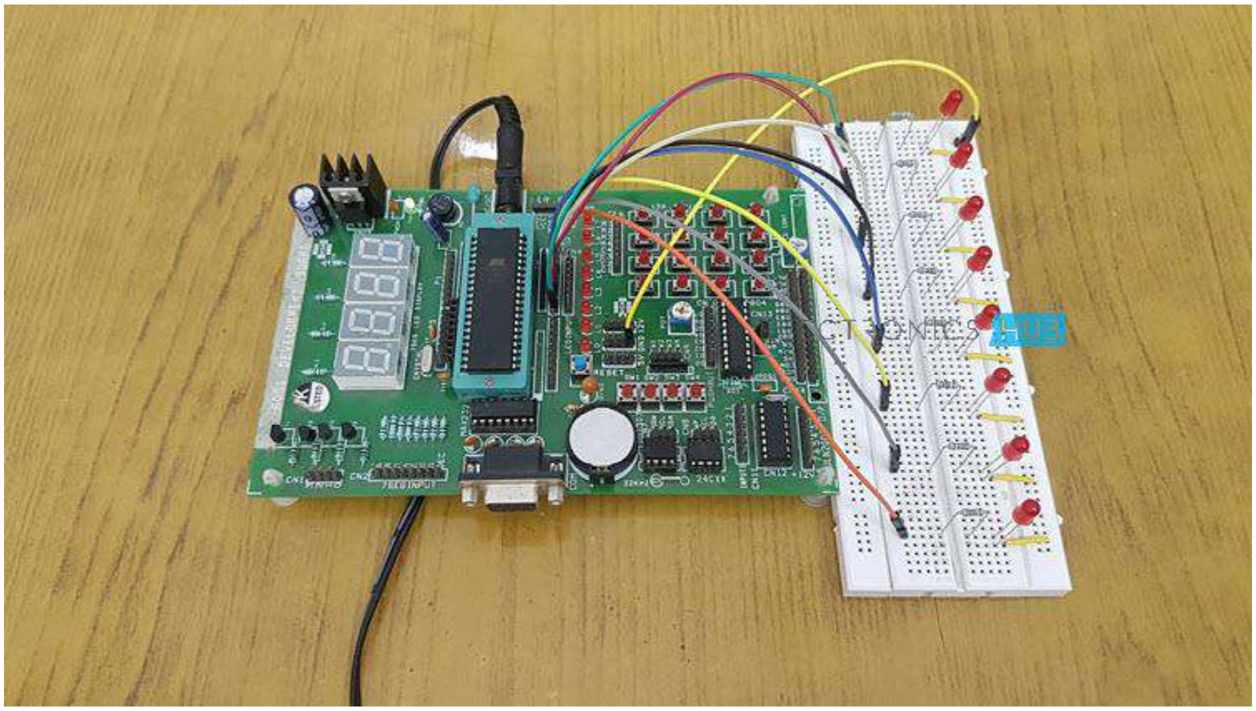
Circuit Diagram



Circuit Design

The circuit mainly consists of AT89C51 microcontroller. AT89C51 belongs to the family of 8051 microcontroller. It is an 8-bit microcontroller. This microcontroller has 4KB of Flash Programmable and Erasable Read Only Memory and 128 bytes of RAM. This can be programmed and erased a maximum of 1000 times.

It has two 16 bit timers/counters. It supports USART communication protocol. It has 40 pins. There are four ports are designated as P0, P1, P2, and P3. Port P0 will not have internal pull- ups, while the other ports have internal pull-ups.



In this circuit, LEDs are connected to the port P0. The controller is connected with external crystal oscillator to pin 18 and 19 pins. Crystal pins are connected to the ground through capacitors of 33pf.

INTERFACING STEPPER MOTOR WITH 8051:

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 mainly used for position control.

• It is used in disk drives, dot matrix printers, plotters and robotics and process control circuits.

Structure

Stepper motors have a permanent magnet called rotor (also called the shaft) surrounded by a stator. The most common stepper motors have four stator windings that are paired with a center-tap. This type of stepper motor is commonly referred to as a four-phase or unipolar stepper motor. The center tap allows a change of current direction in each of two coils when a winding is grounded, thereby resulting in a polarity change of the stator.

Interfacing

Even a small stepper motor require a current of 400 mA for its operation. But the ports of the microcontroller cannot source this much amount of current. If such a motor is directly connected to the microprocessor/microcontroller ports, the motor may draw large current from the ports and damage it. So a suitable driver circuit is used with the microprocessor/microcontroller to operate the motor.

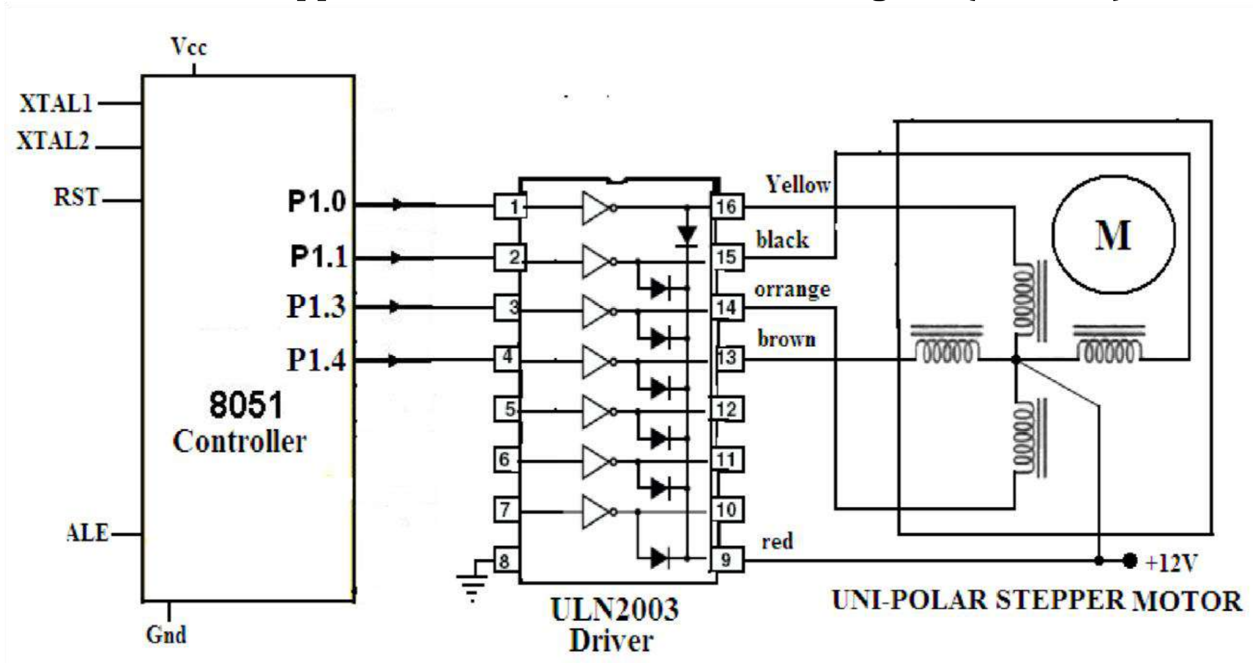
Motor Driver Circuit (ULN2003)

Stepper motor driver circuits are available readily in the form of ICs. ULN2003 is one such driver IC which is a High-Voltage High-Current Darlington transistor array and can give a current of 500mA.This current is sufficient to drive a small stepper motor. Internally, it has protection diodes used to protect the motor from damage due to back e.m.f. and large eddy currents. So, this ULN2003 is used as a driver to interface the stepper motor to the microcontroller.

Operation:

- The important parameter of a stepper motor is the **step angle**.
- It is the minimum angle through which the motor rotates in response to each **excitation pulse**.
- In a four phase motor if there are 200 steps in one complete rotation then then the step angle is 360/200 = 1.8^0.
- So to rotate the stepper motor we have to apply the excitation pulse. For this the controller should send a hexa decimal code through one of its ports.
- **The hex code mainly depends on the construction of the stepper motor**. So, all the stepper motors do not have the same Hex code for their rotation.
- For example, let us consider the hex code for a stepper motor to rotate in clockwise direction is 77H, BBH, DDH and EEH.
- This hex code will be applied to the input terminals of the driver through the assembly language program.
- To rotate the stepper motor in anti-clockwise direction the same code is applied in the reverse order.

Stepper Motor interface- Schematic Diagram (for 8051)



The assembly language program for 8051 is given below.
ASSEMBLY LANGUAGE PROGRAM (8051)

```
Main    : MOV A, # 0FF H                  ;    Initialization of Port 1
             MOV P1, A                     ;
             MOV A, #77 H                  ; Code for the Phase 1
             MOV P1, A                     ;
             ACALL DELAY                   ;   Delay subroutine
             MOV A, # BB H                 ; Code for the Phase II
             MOV P1, A                     ;
             ACALL DELAY                   ;   Delay subroutine.
          MOV A, # DD H                    ; Code for the Phase III
             MOV P1, A                     ;
             ACALL DELAY                   ; Delay subroutine
             MOV A, # EE H                 ;  Code for the Phase 1
             MOV P1, A                     ;
```

```
        ACALL DELAY              ;  Delay subroutine
        SJMP MAIN; Keep the motor rotating continuously.
```

DELAY Subroutine

```
         MOV R4, #0FF H                   ; Load R4 with FF
         MOV R5, # 0FF                    ; Load R5 with FF
LOOP1:  DJNZ R4, LOOP1                   ; Decrement R4 until zero,
tLOOP2:  DJNZ R5, LOOP2                  ; Decrement R5 until
                                          zero, wait
         RET                               ; Return to main
                                          program
```