

UNIT- 1: MICROPROCESSOR (ARCHITECTURE 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:

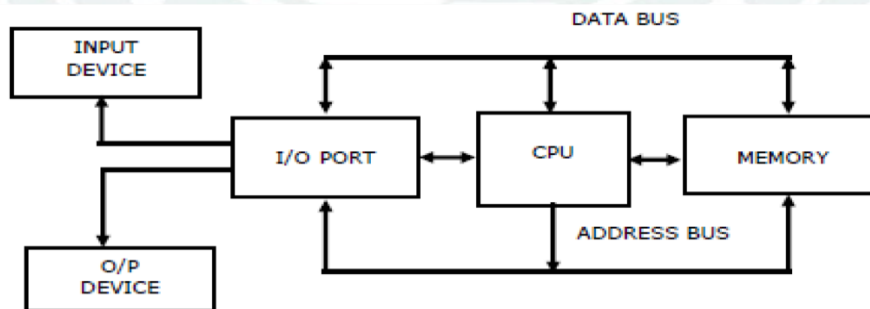


Figure: Block Diagram of a simple Microcomputer

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-

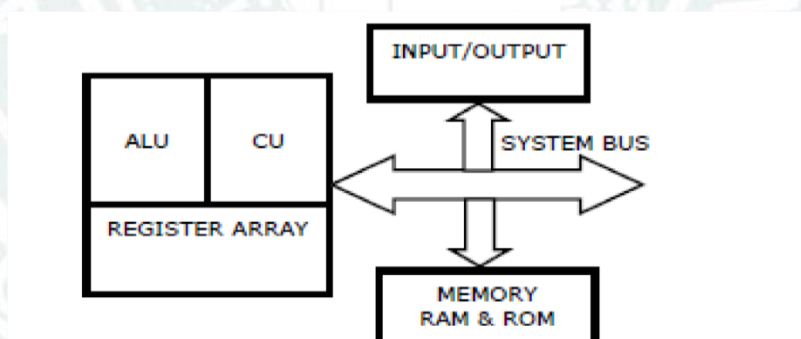


Figure: Microprocessor Based System with Bus Architecture.

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 2^N memory locations i.e. CPU with 16 address lines can address 2^{16} or 65536 memory locations.

DATA BUS:

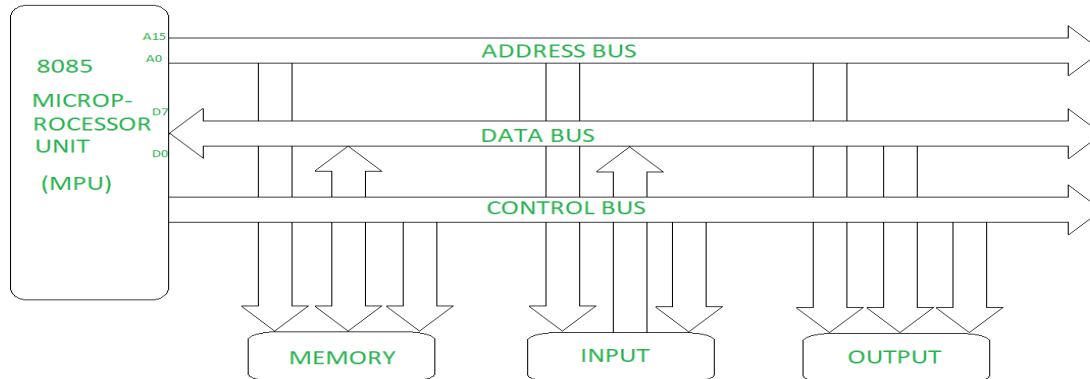
- The data bus consists of 8, 16 or 32 parallel signal lines.
- The data bus lines are bi-directional.
- This means that the CPU can read data in from memory or it can send data out to memory.

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:

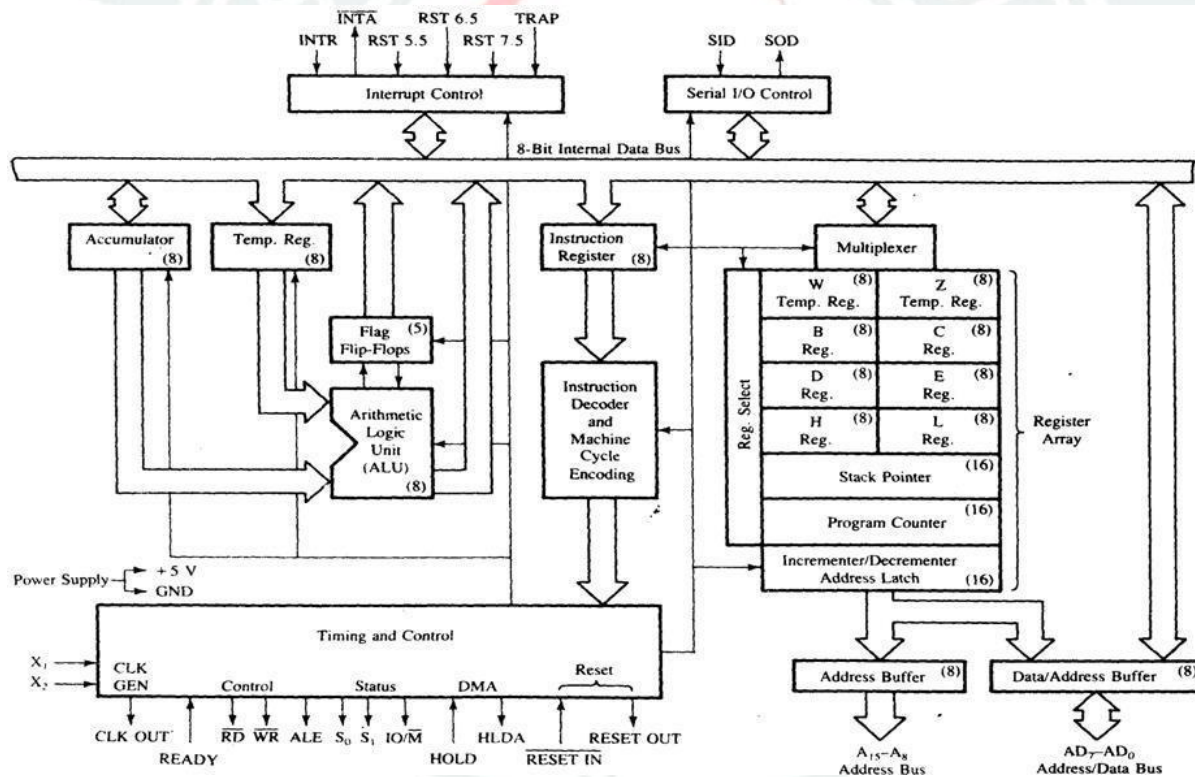
- It is a group of conducting wires which carries Data only.
- Data bus is bidirectional because data flow in both directions, from microprocessor to memory or Input/output devices and from memory or Input/output devices to microprocessor.
- Length of Data Bus of 8085 microprocessor is 8 Bit (That is,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

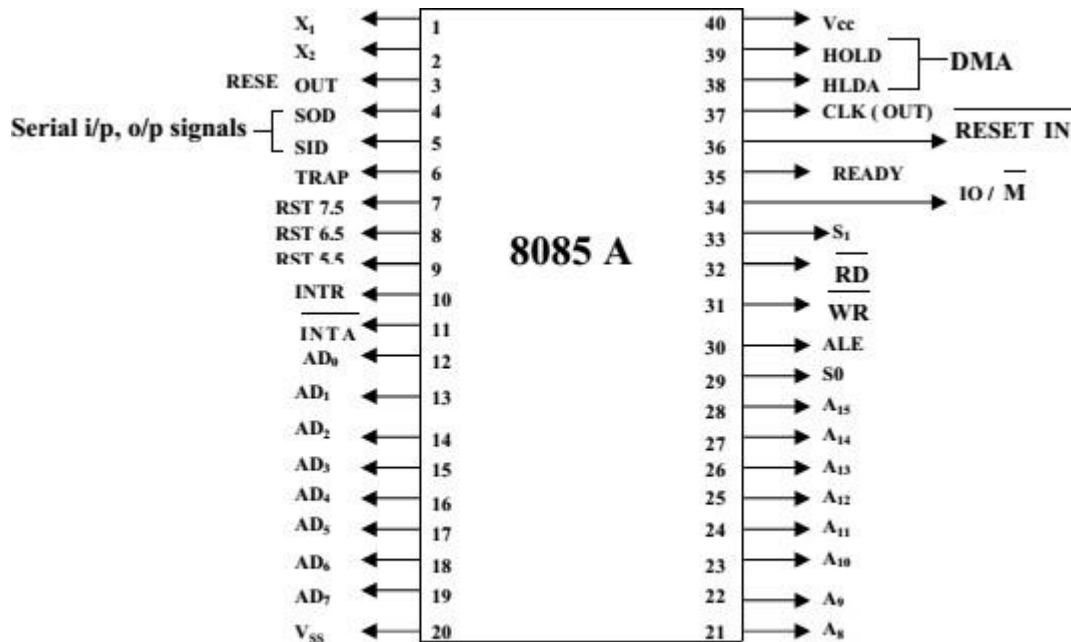
Address buffer and address-data buffer:

- 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} . V_{CC} indicates +5v power supply and V_{SS} 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.

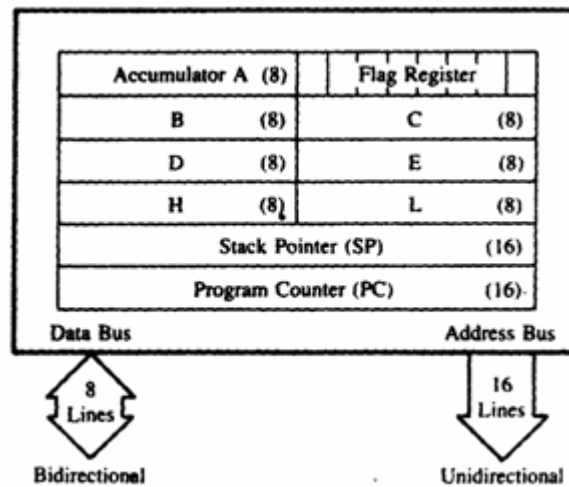
SID (Serial input data line):

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 B and 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 8085 Microprocessor Architecture uses them internally.

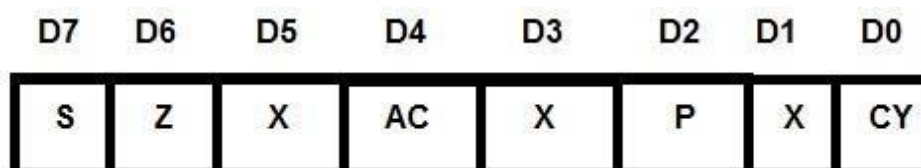
3. Special Purpose Registers:

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 is 1, the number will be viewed as a negative number. If D7 is 0, 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.

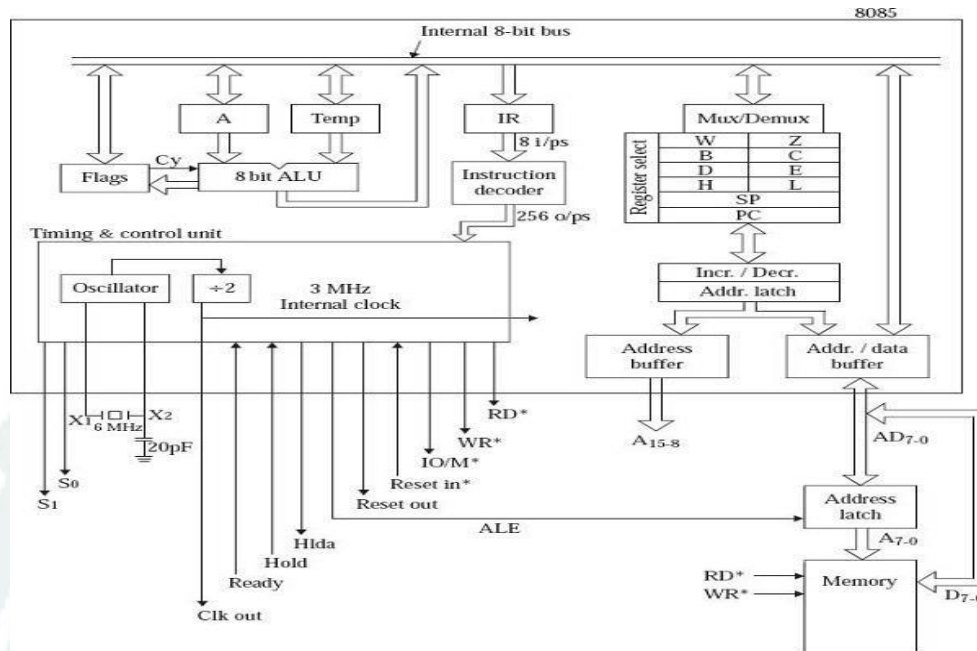
SPR-

- 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:

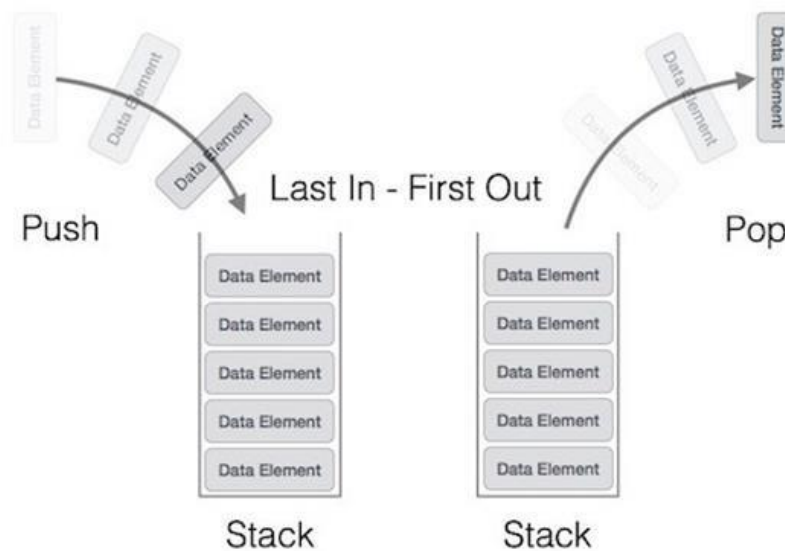
- We use Timing and controlling unit in 8085 for the generation of timing signals and the signals to control.
- All the operations and functions both interior and exterior of a microprocessor are controlled by this unit.
- X2 and CLK output pins: To do or rather perform the operations of timing in the 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

Classification of Interrupts:

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.

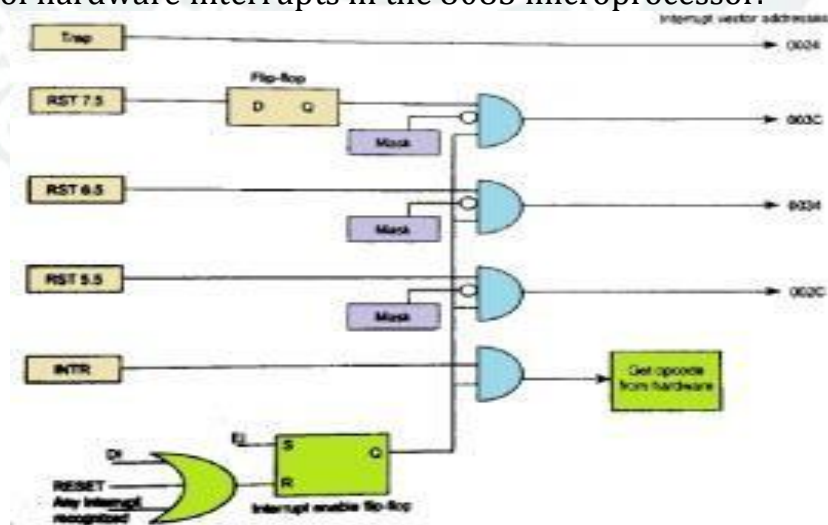
- There are two ways of redirecting the execution to the ISR depending on whether the interrupt is vectored or non-vectored.
- **Vectored:** The address of the subroutine is already known to the Microprocessor.
- **Non Vectored:** The device will have to supply the address of the subroutine to the Microprocessor.
- 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 all automatically vectored and are maskable.
- **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 below 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

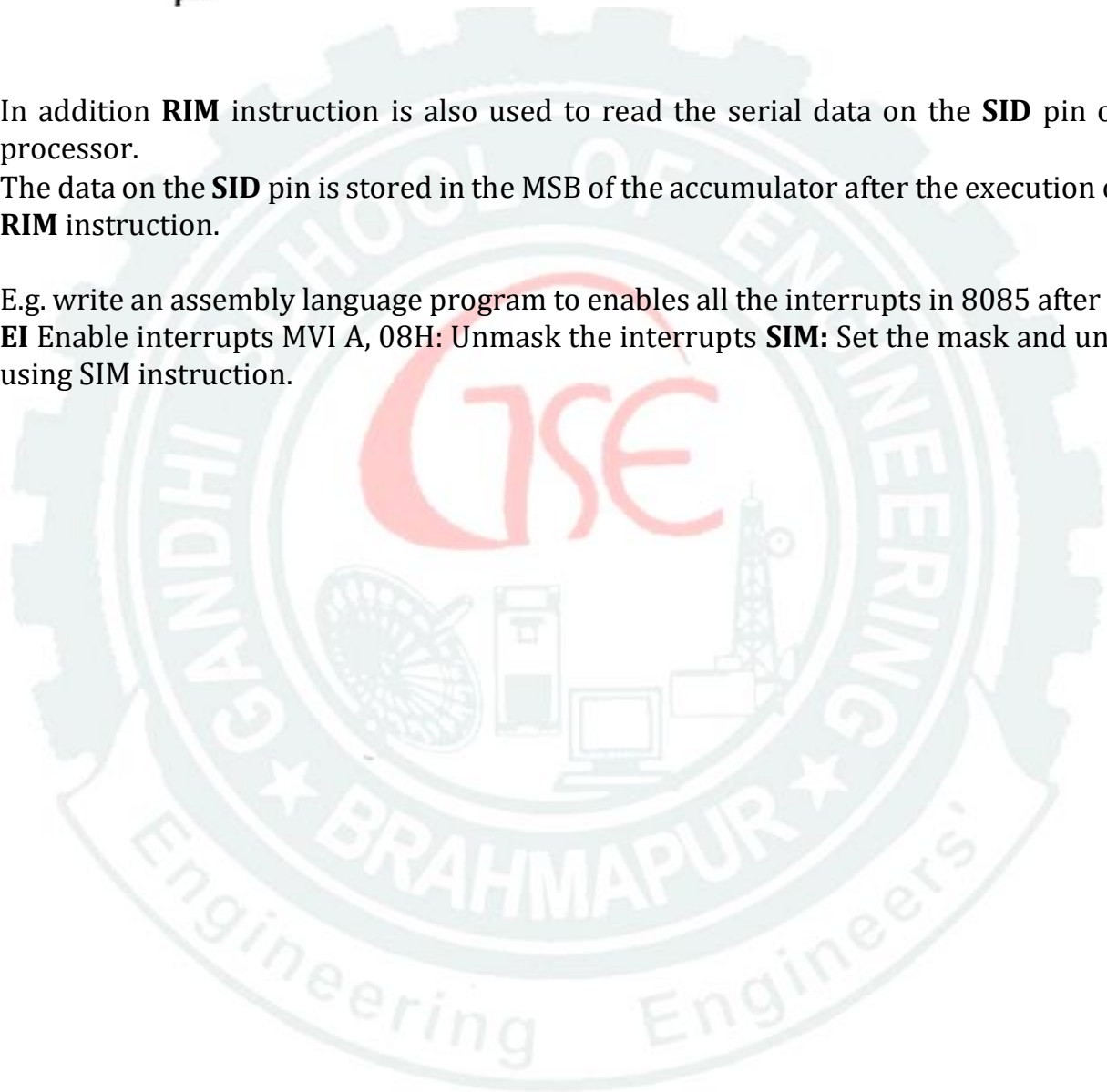
- In addition to masking interrupts, **SIM** instruction can be used to send serial data on the **SOD** line of the processor.
- The data to be send is placed in the MSB bit of the accumulator and the serial data output is enabled by making D6 bit to 1.

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	Description
Copy from source to destination		
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
		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
	M, data	

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 complement, 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

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

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

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 is satisfied.

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

2. Call Instructions:-

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

(a) Unconditional Call Instructions:

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

OPCODE	OPERAND	EXPLANATION	EXAMPLE
CALL	address	Unconditionally calls	CALL 2050

(b) Conditional Call Instructions:

Only if the condition is satisfied, the instructions executes.

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

3. Return Instructions: -

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

(a) Unconditional Return Instruction:

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

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

(b) Conditional Return Instruction:

The program sequence is transferred unconditionally from the subroutine to the calling program only if the condition is satisfied.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
RC	none	Return from the subroutine if carry flag is 1	RC
RNC	none	Return from the subroutine if carry flag is 0	RNC
RZ	none	Return from the subroutine if zero flag is 1	RZ
RNZ	none	Return from the subroutine if zero flag is 0	RNZ
RPE	none	Return from the subroutine if parity flag is 1	RPE
RPO	none	Return from the subroutine if parity flag is 0	RPO
RM	none	Returns from the subroutine if sign flag is 1	RM
RP	none	Returns from the subroutine if sign flag is 0	RP

STACK, I/O & 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.

Examples of assembly language program

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, 08 H will place the given data 08 H in the register A.
- The Hex code for MVI A, 08 H is 3E, 08 H 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 copied 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 fetched 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	1 st number in accumulator.
2004	23	INX	H	Increment content of H-L pair.
2005	86	ADD	M	Add 1 st and 2 nd numbers.
2006	32,03,25	STA	2503H	Store sum in 2503H.
2009	76	HLT		Stop the program.

EXPLANATION:

- The 1st number 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 1st number 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 1 st in H-L pair.
2003	7E	MOV	A,M	1 st number in accumulator.
2004	23	INX	H	Content of H-L pair increases from 2501 to 2502 H
2005	96	SUB	M	1 st number- 2 nd number.
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	$A \leftarrow M[2050]$
2003	ANI 0F	$A \leftarrow A \text{ (AND) } 0F$
2005	MOV B, A	$B \leftarrow A$
2006	LDA 2050	$A \leftarrow M[2050]$
2009	ANI F0	$A \leftarrow A \text{ (AND) } F0$
200B	RLC	Rotate accumulator left by one bit without carry

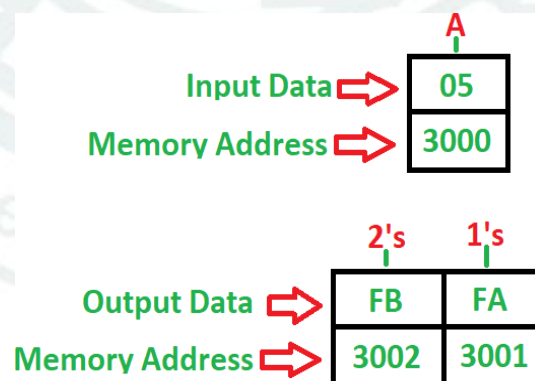
200C	RLC	Rotate accumulator left by one bit without carry
200D	RLC	Rotate accumulator left by one bit without carry
200E	RLC	Rotate accumulator left by one bit without carry
200F	ANA B	$A \leftarrow A \text{ (AND) } B$
2010	STA 3050	$M[3050] \leftarrow A$
2013	HLT	END

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 -

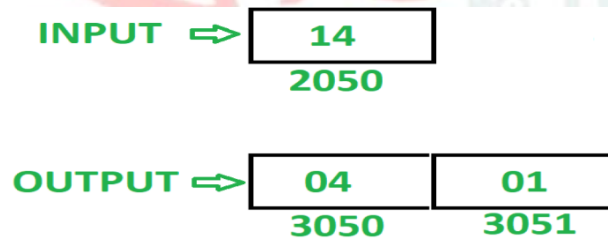
MEMORY ADDRESS	MNEMONICS	OPERANDS	COMMENT
2000	LDA	[3000]	$[A] \leftarrow [3000]$
2003	CMA		$[A] \leftarrow [A^{\wedge}]$
2004	STA	[3001]	1's complement
2007	ADI	01	$[A] \leftarrow [A] + 01$
2009	STA	[3002]	2's complement
200C	HLT		Stop

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	$A \leftarrow M[2050]$
2003	MOV B, A	$B \leftarrow A$
2004	ANI 0F	$A \leftarrow A \text{ (AND) } 0F$

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	7
• LOOP: DCR B	4
• JNZ LOOP	7/10
• RET	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 $3584 * 1/3\mu s = 1194.66\mu s$. 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	10
LOOP: DCX B	6
MOV A,B	4
ORA C	4
JNZ LOOP	10 (For Jump),
RET	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 $1572857 * 1/3\mu s = 0.52428s$. 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.

Looping, counting and indexing (Call/JMP)

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 at a 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
```

```
SPLH
```

```
LXI H, 1234H
```

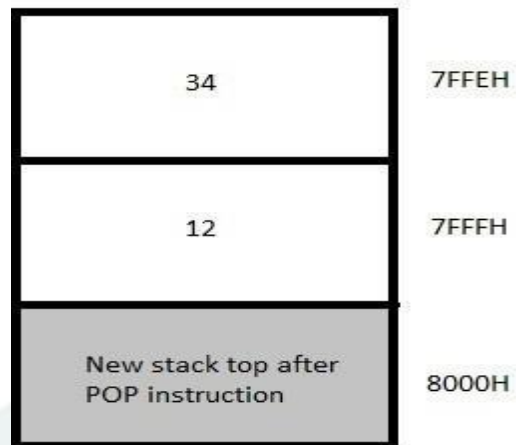
```
PUSH H
```

```
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.
- SPLH – 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 in 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 0AH + 7 = 3CH + 7 = 43H

Source Program:

```

LDA 2200H      : Get the BCD number
MOV B, A       : Save it
ANI 0FH       : Mask most significant four bits
MOV C, A       : Save unpacked BCDI in C register
MOV A, B       : Get BCD again
ANI 0FH       : 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? 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 1 st number in H-L pair.
2003	7E		MOV	A,M	1 st number in accumulator.
2004	23		INX	H	Address of 2 nd number in H-L pair.
2005	BE		CMP	M	Compared 2 nd number with 1 st number. Is the 2 nd number > 1 st ?
2006	D2,0A,20		JNC	AHEAD	No, larger number is in accumulator. Go to AHEAD
2009	7E		MOV	A,M	Yes, get 2 nd 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	1 st number in accumulator
2004	23		INX	H	Address of the 2 nd number in H-L pair.
2005	BE		CMP	M	Compare 2 nd number with 1 st . Is 1 st number < 2 nd number?
2006	DA,0A,20		JNC	AHEAD	Yes, smaller number is in accumulator. Go to AHEAD.
2009	7E		MOV	A,M	No, get 2 nd 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 1 st number in H-L pair.
2005	7E		MOV	A,M	1 st number in accumulator.
2006	0D		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 1 st number in H-L pair.
2005	7E		MOV	A,M	1 st number 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 in-line 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.

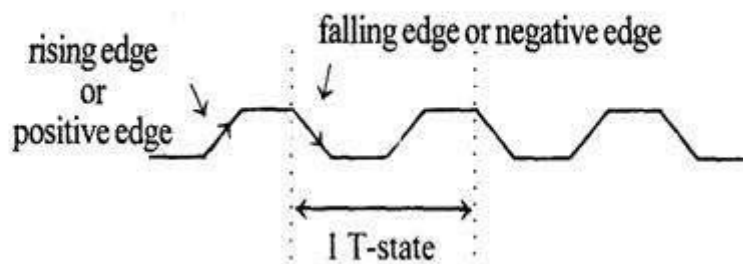
3.2 CONCEPT OF TIMING DIAGRAM:

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)

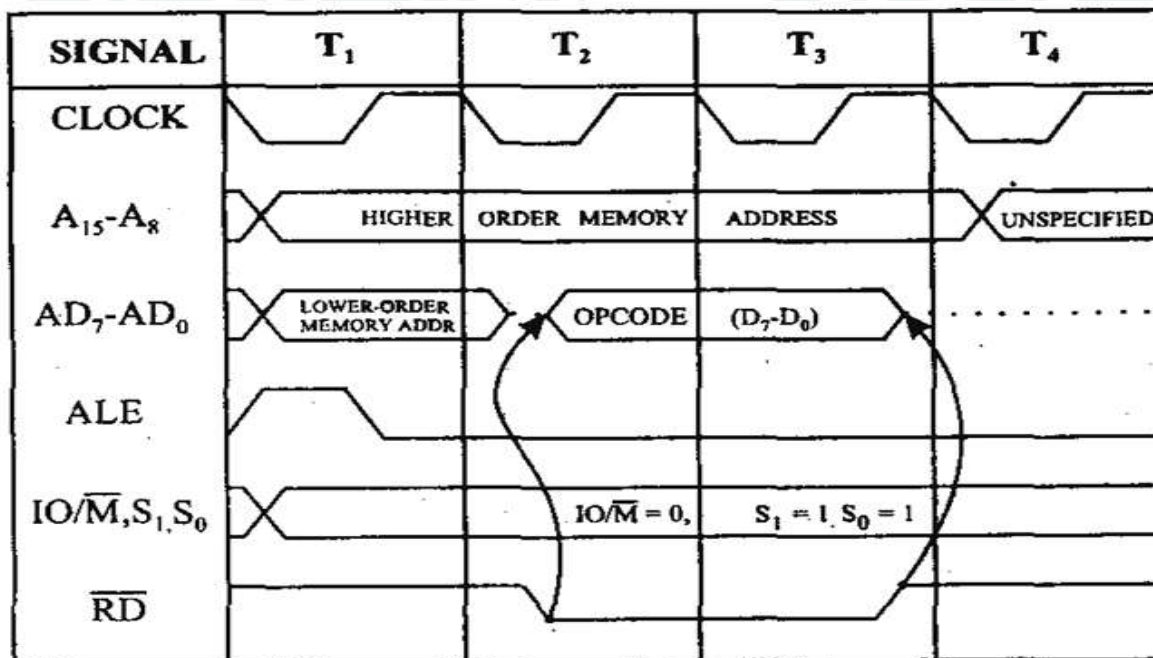
Note : Time period, $T = 1/f$; where $f =$ Internal clock frequency



- 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.

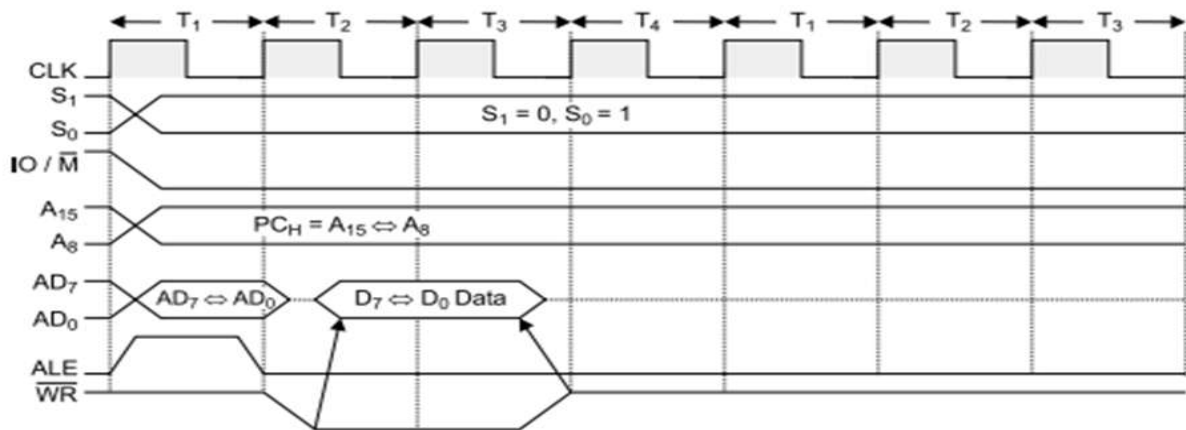
T4 clock cycle:

- 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:

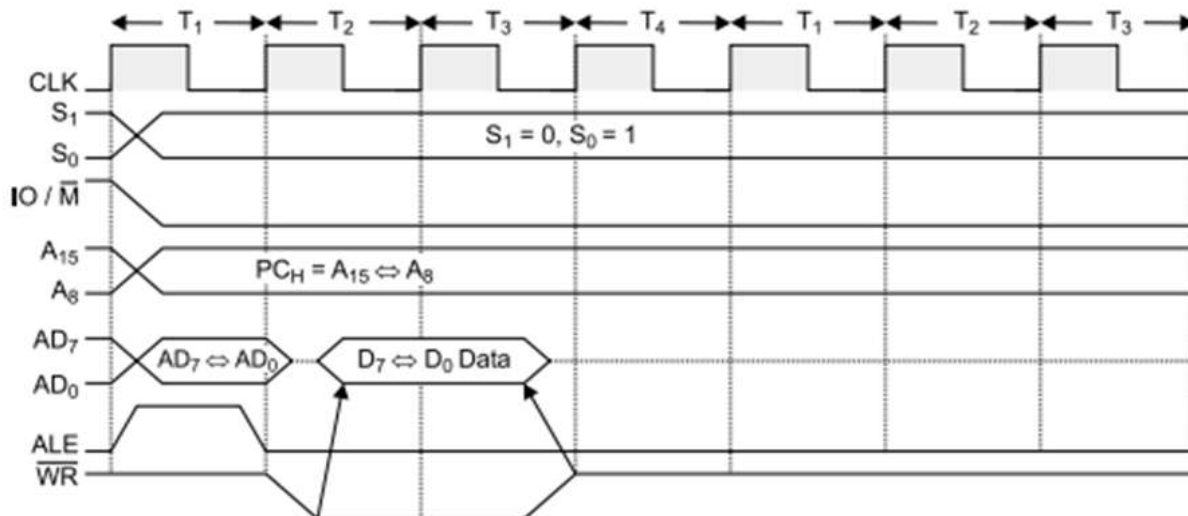
- Selected memory location is placed on the (D0-D7) of the A/D multiplexed bus. RD' goes LOW

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.

Memory Write Machine Cycle:

- 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:

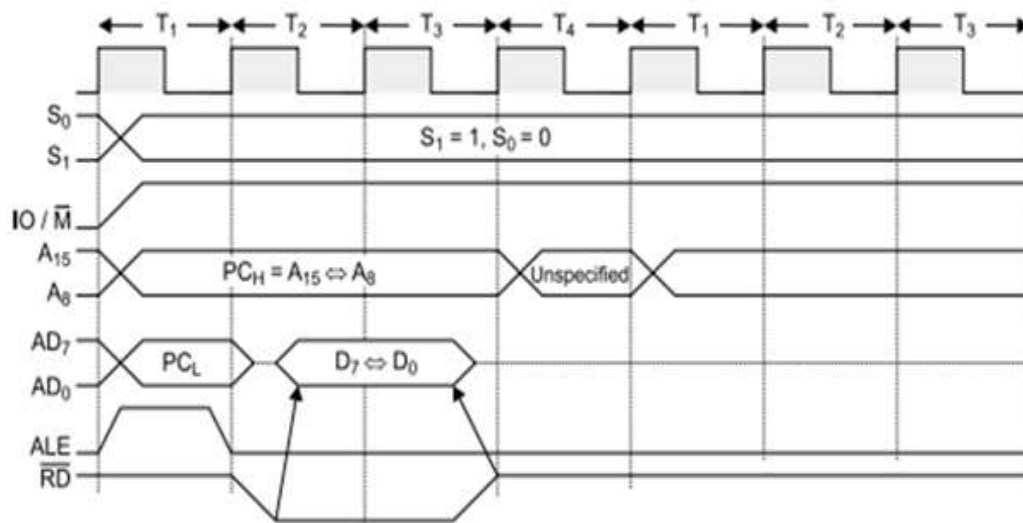
- Selected memory location is placed on the (D0-D7) of the A/D multiplexed bus. WR' goes LOW

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.

I/O Read Cycle:

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.

I/O Write Cycle:

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:

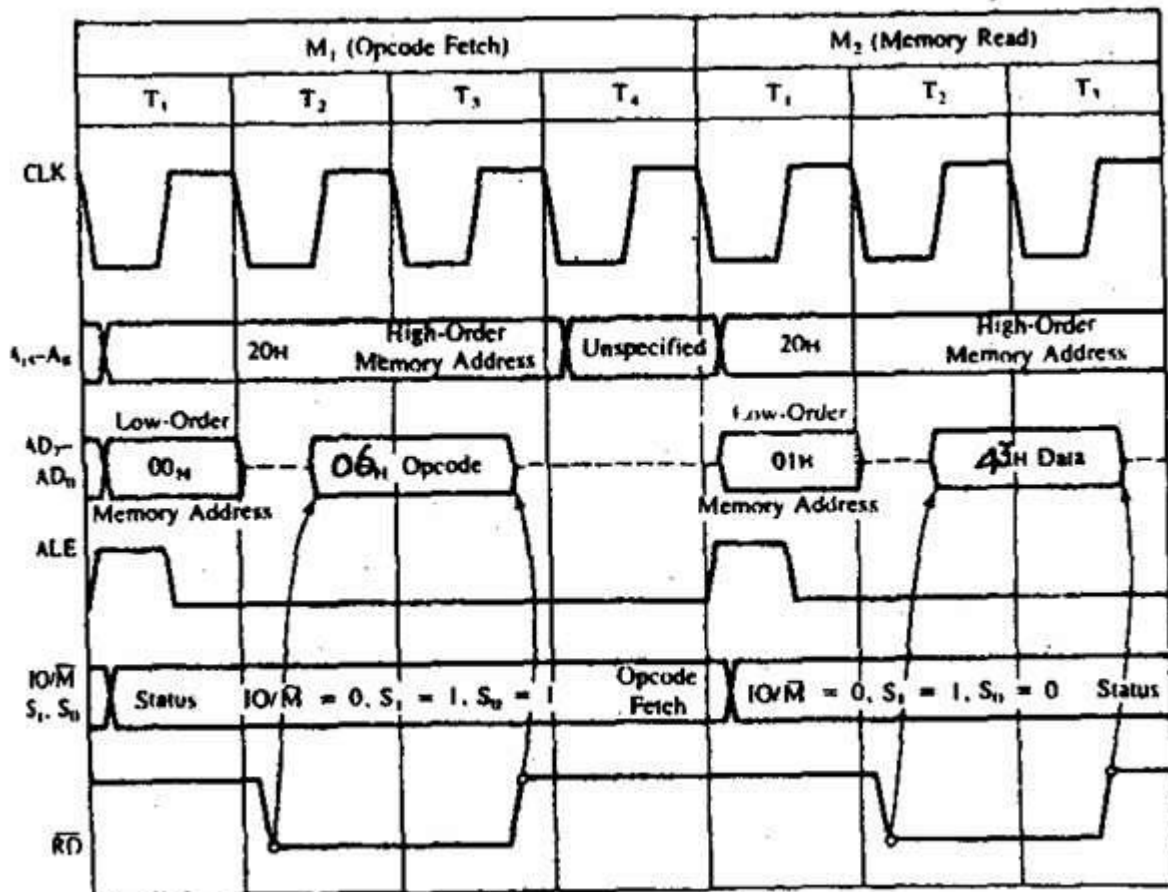
- In the middle of the T3 state WR' goes high and disables the I/O write operation. The data which was obtained from the I/O is then decoded.

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)

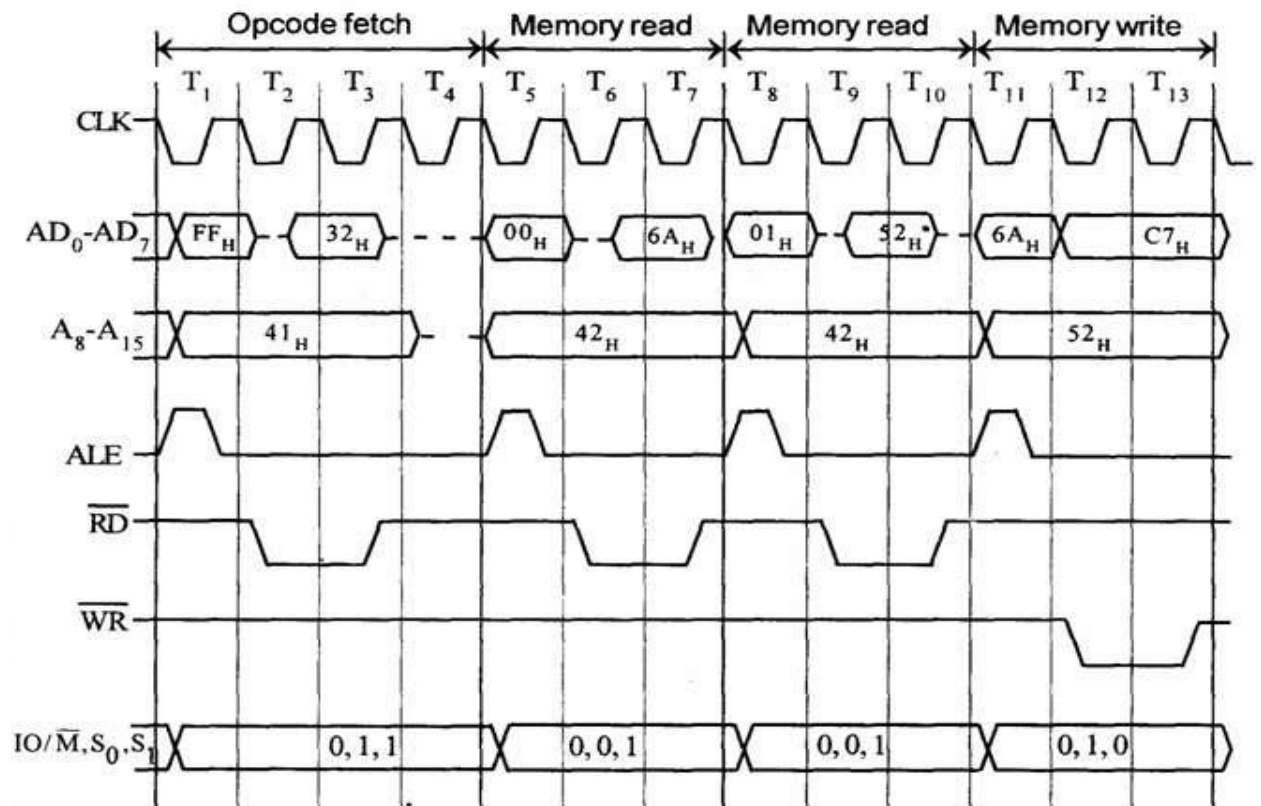
Address	Mnemonics	Op code
2000	MVI B, 43H	06H
2001		43H



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.

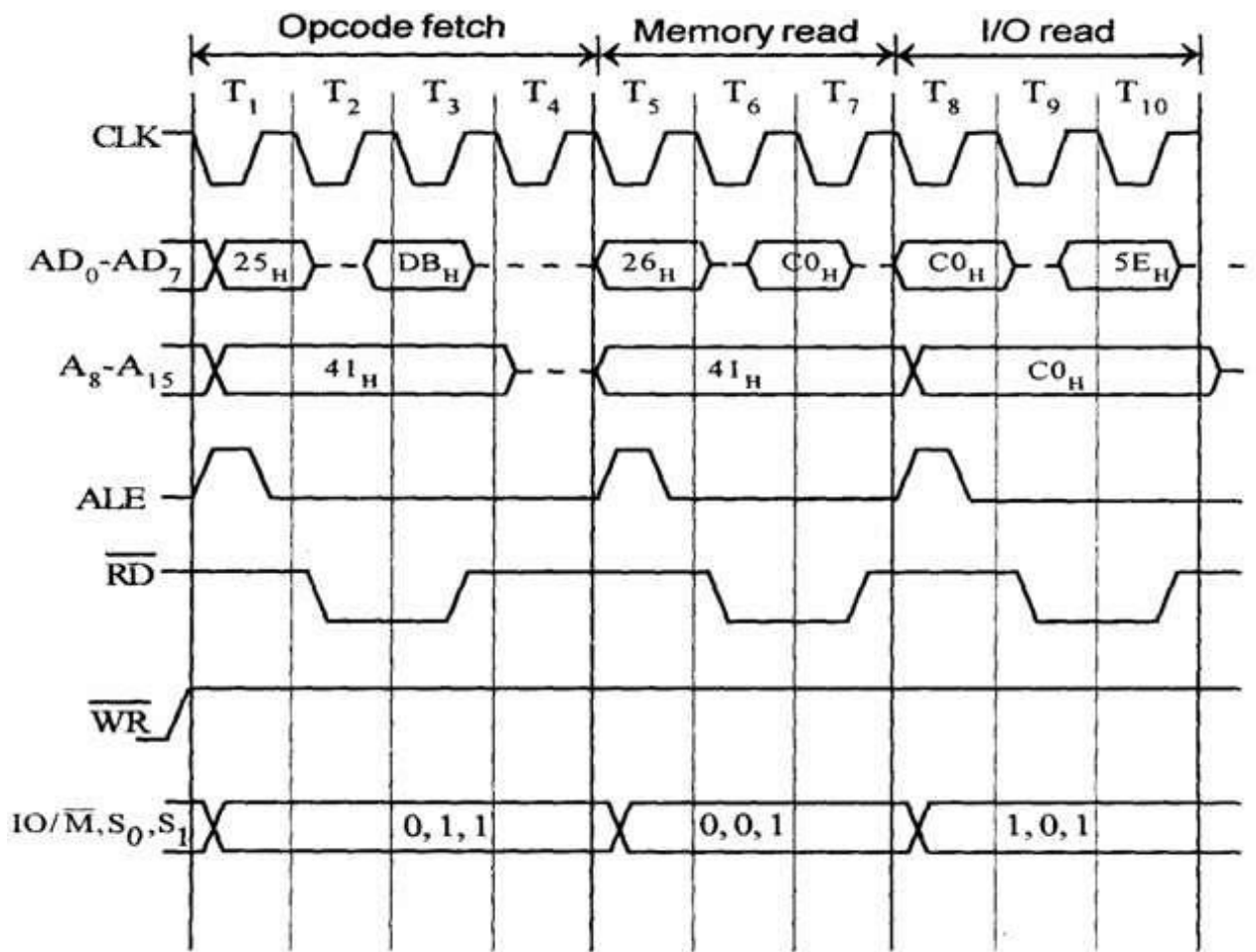
Address	Mnemonics	Op code
41FF	STA 526AH	32H
4200		6AH
4201		52H



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.

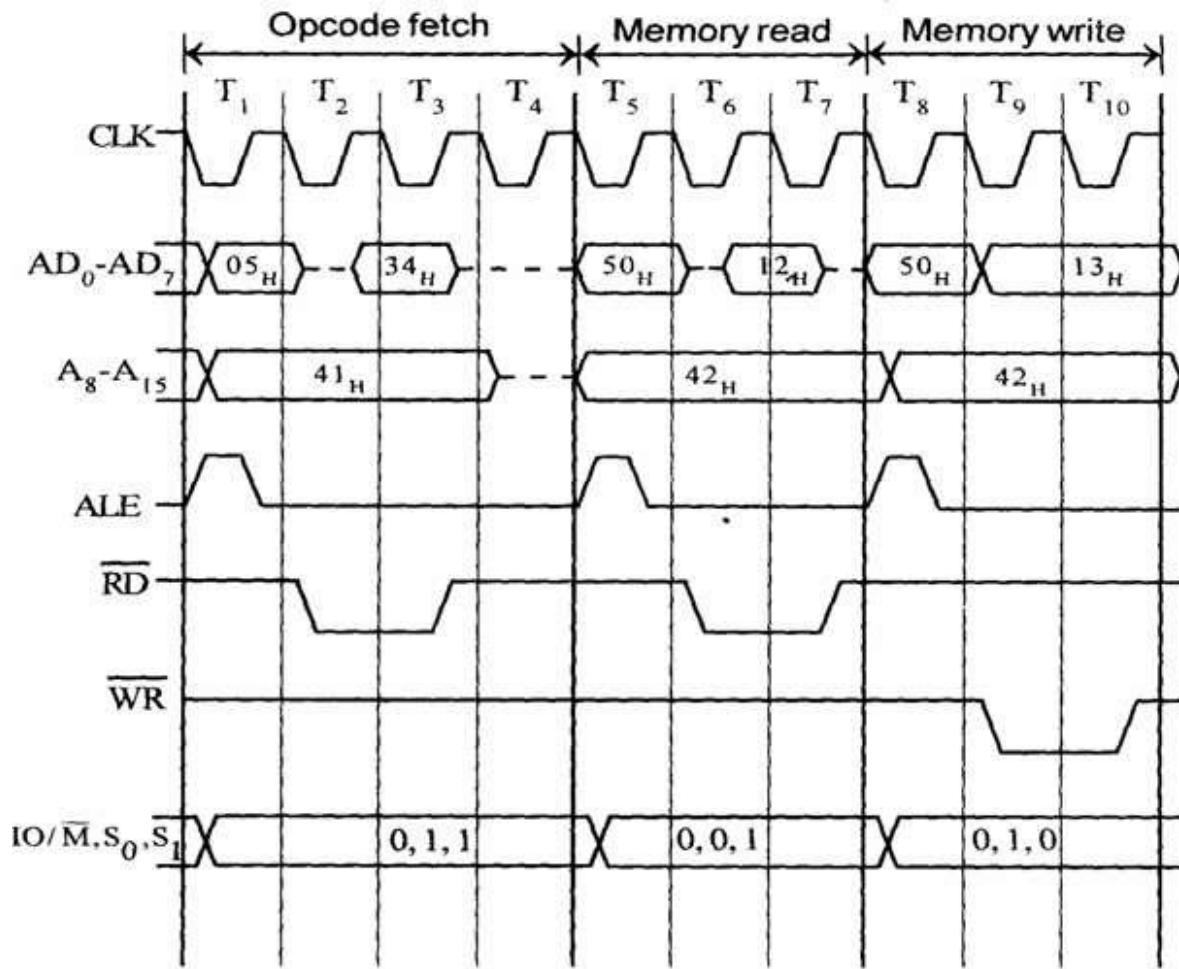
Address	Mnemonics	Op code
4125	IN C0H	DBH
4126		C0H



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	Op code
4105	INR.M	34 _H



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:

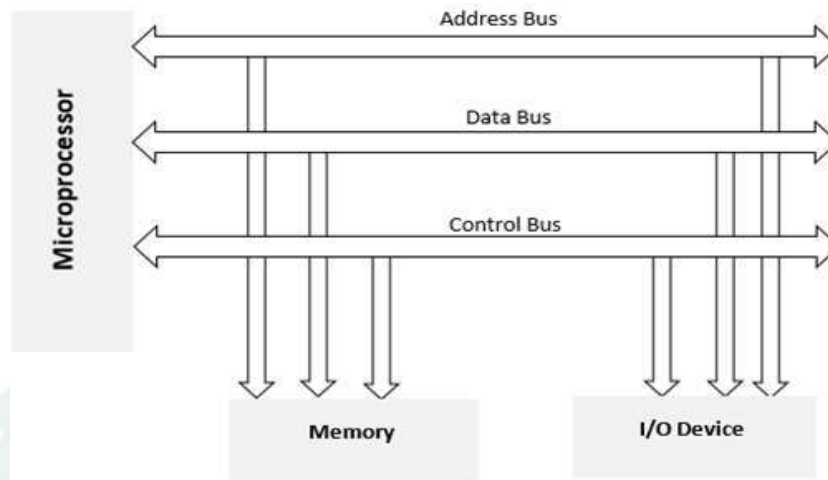
MEMORY MAPPING:

- 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:

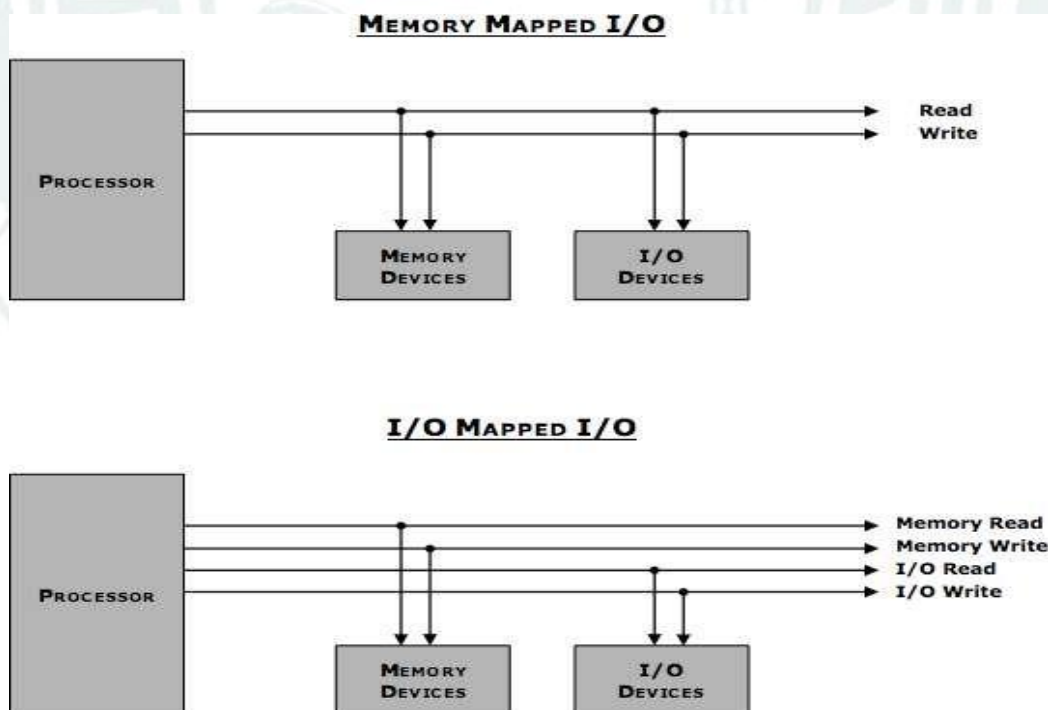
I/O interfacing:

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.

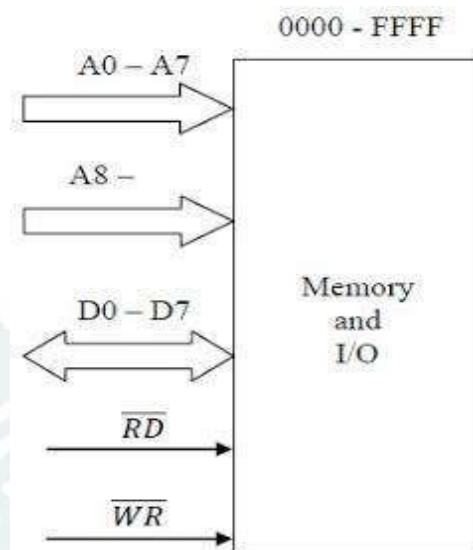


Figure: Memory mapped I/O scheme

- 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.
- \overline{RD} ' is activated for read operation and \overline{WR} ' 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.
- \overline{RD} ' is activated for read operation and \overline{WR} ' 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.

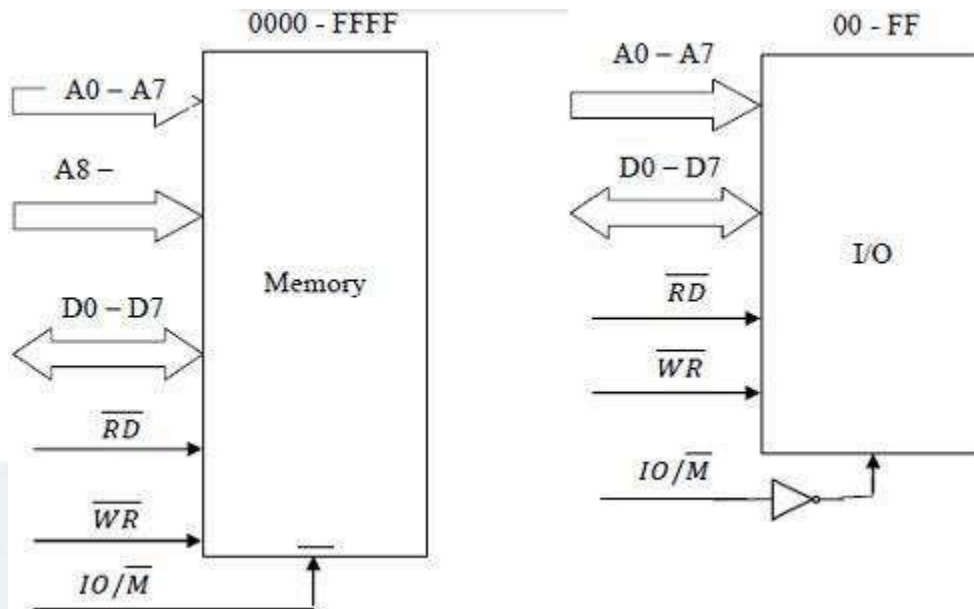


Figure: I/O mapped I/O scheme

- 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/\overline{M} ' signal.
- When IO/\overline{M} ' is low, the memory is selected for reading and writing operations.
- When IO/\overline{M} ' 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/\overline{M} ' line low.
- The microprocessor makes the \overline{RD} ' low for read operation and \overline{WR} ' 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 \overline{OE} ' low for read operation and \overline{WE} ' 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

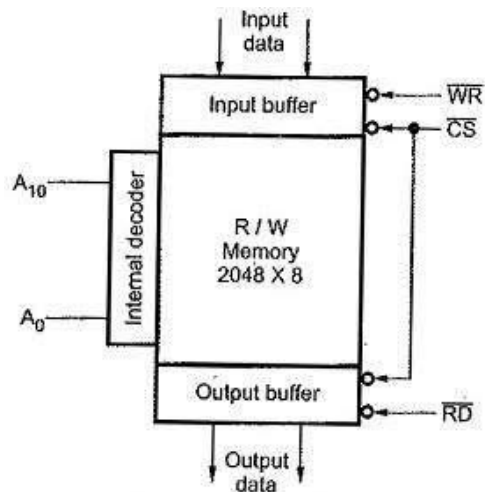
- Memory Interfacing is used when the microprocessor needs to access memory frequently for reading and writing data stored in the memory. It is used when reading/writing to a specific register of a memory chip.

The memory interfacing requires to:

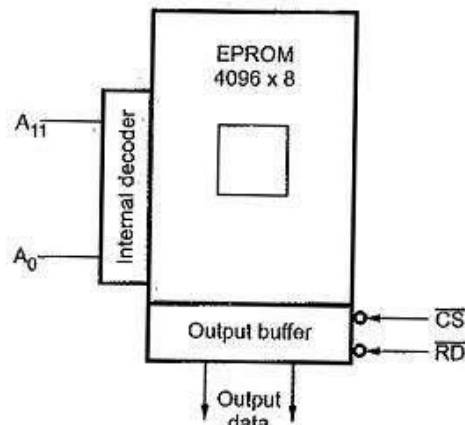
- Select the chip
- Identify the register
- Enable the appropriate buffer.
- Microprocessor system includes memory devices and I/O devices. It is important to note that microprocessor can communicate (read/write) with only one device at a time, since the data, address and control buses are common for 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 $N \times M$, 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 0FFFH, 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

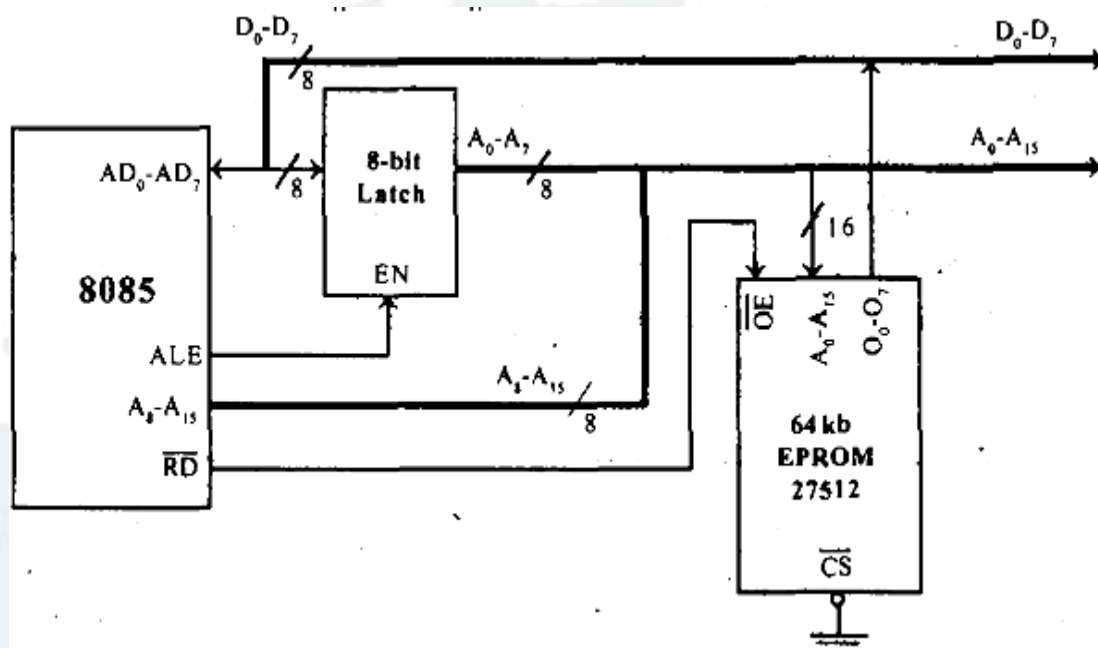
EXAMPLES OF MEMORY INTERFACING

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.
- $2^n = 64 \times 1000$ 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.

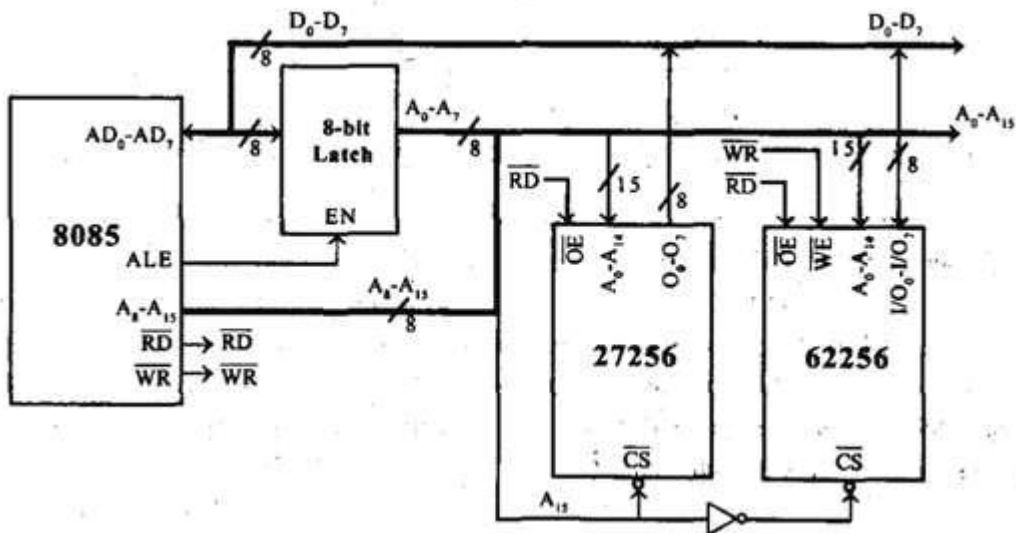


Interfacing 64Kb EPROM with 8085

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.

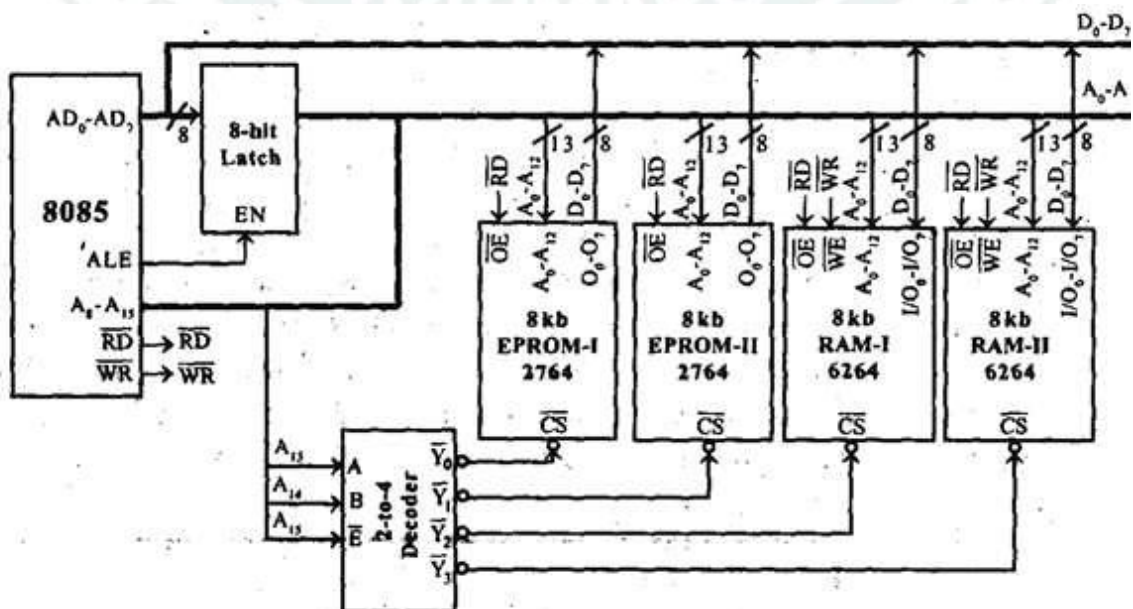


Interfacing 32Kb EPROM and 32Kb RAM with 8085

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

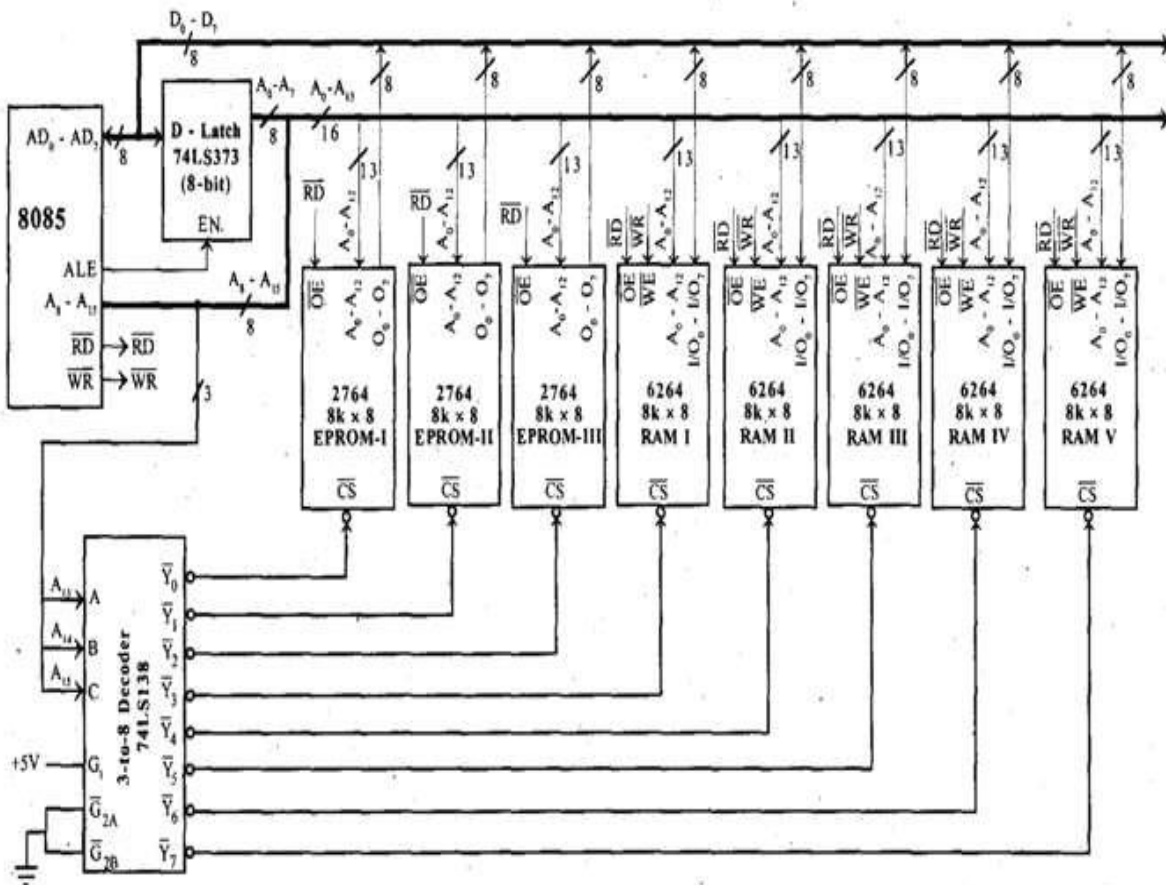
	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	5FFF
8kb RAM - II	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	6000
	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	6001
	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	6002

	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 13 address pins of all the memory ICs.
- 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	1	0	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	1	0	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	1	0	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	1	0	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	1	0	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	1	0	C001
	1	1	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	1	0	0	E001
	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	E002

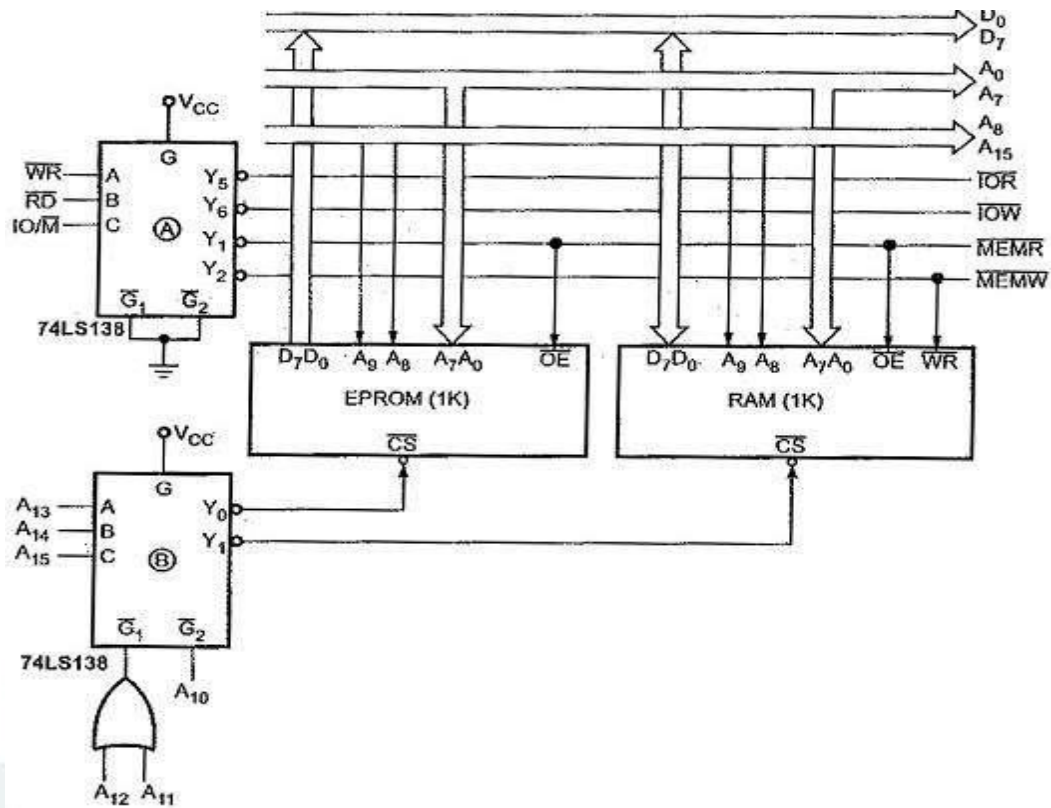
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FFFF

4.4 ADDRESS DECODING TECHNIQUES:

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

Absolute decoding:

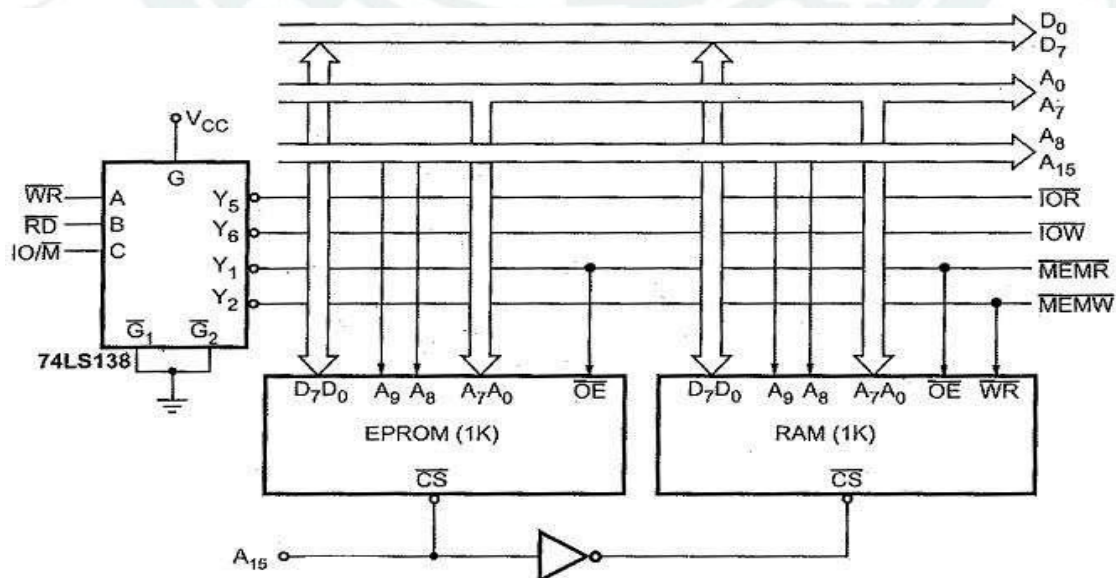
- In absolute decoding technique, all the higher address lines are decoded to select the memory chip, and the memory chip is selected only for the specified logic levels on these high-order address lines; no other logic levels can select the chip.
- This figure shows the Memory Interfacing in 8085 with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

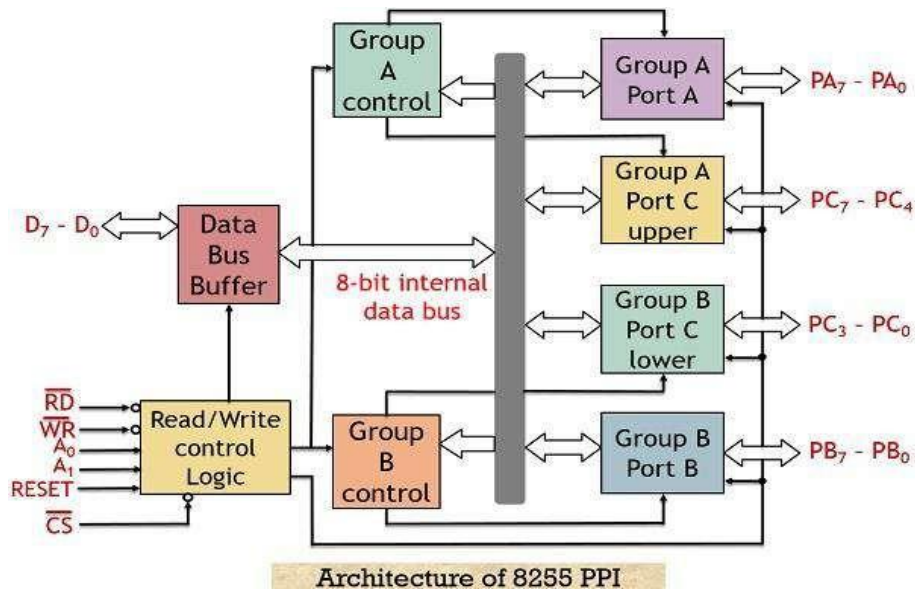
Mode 0: Simple input/output

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:



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:

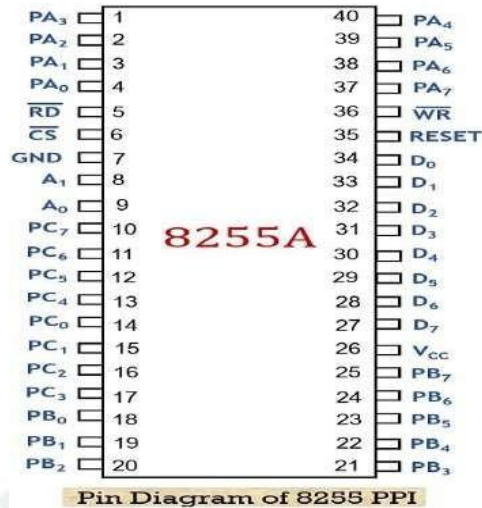
- This unit manages the internal operations of the system. This unit holds the ability to control the transfer of data and control or status words both internally and externally.
- Whenever there exists a need for data fetch then it accepts the address provided by the processor through the bus and immediately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



CS:

- 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.

RD:

- 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.

WR:

- 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.

A₀ and A₁:

- 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:

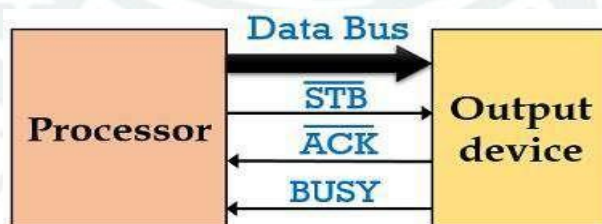
This mode is the simple input output mode of 8255 which allows the programming of each port as either input or output port. The input/output feature of mode 0 includes:

- 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 act as bidirectional data bus and the port C upper (PC₇ – PC₄) 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:

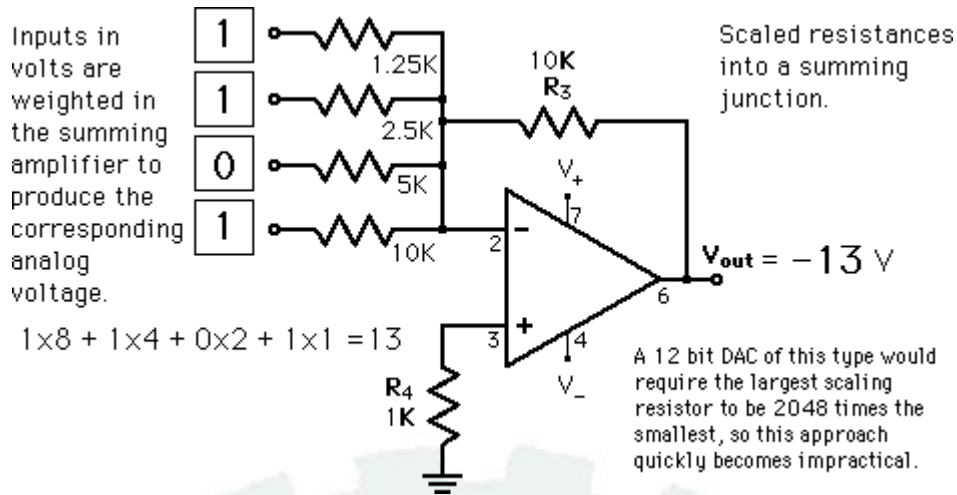
- Digital-to-Analog Conversion or simply DAC, is a device that is used to convert a digital (usually binary) code into an analog signal (current, voltage, or electric charge).
- Digital-to-analog conversion is the primary means by which digital equipment such as computer-based systems are able to translate 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

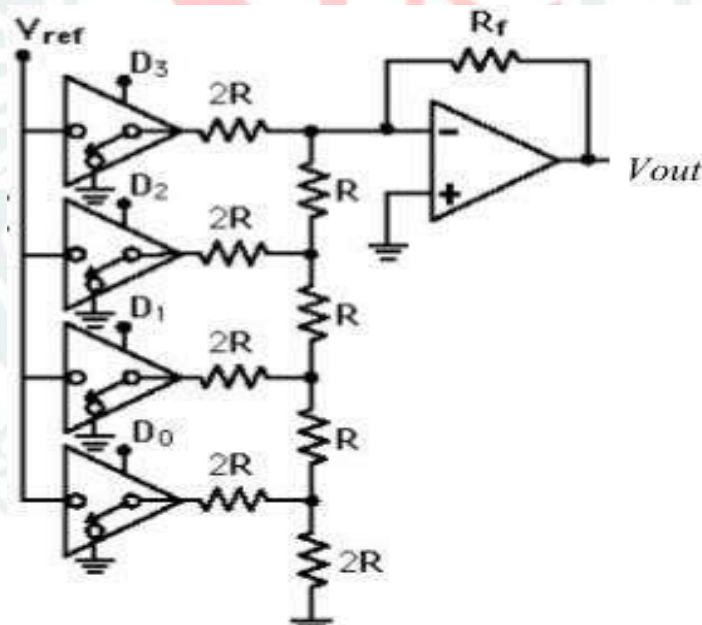
Weighted Sum DAC:

- 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.

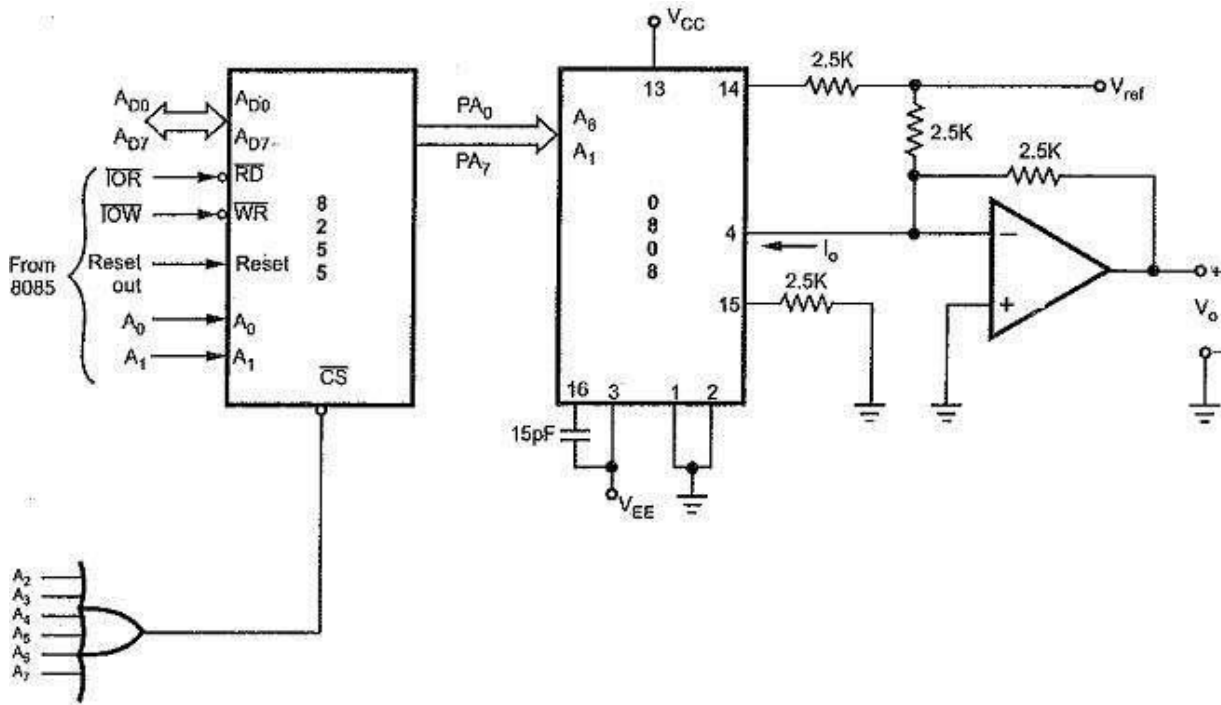


R-2R Ladder DAC:

- 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

Program:

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 portlines 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.

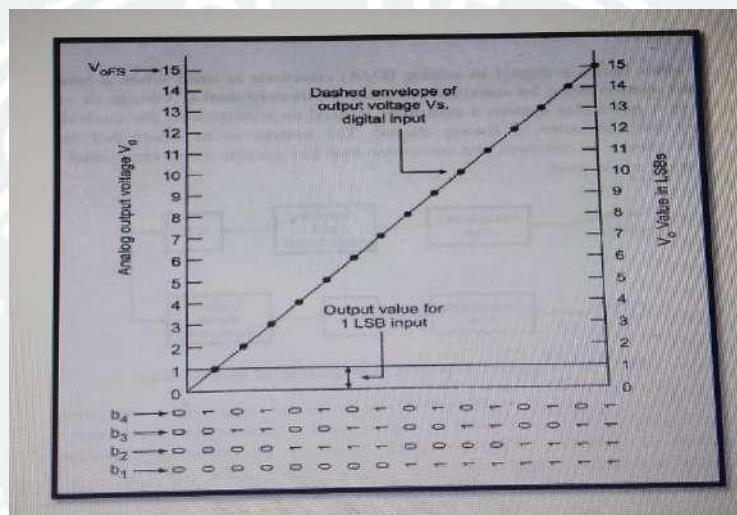


Figure shows analog output voltage v_0 is plotted against all 16 possible digital input words.

Performance Parameters of DAC:

The performance parameters of DAC are:

1. Resolution:

- 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 = $V_0 F_s / 2^n - 1$ (2)**
- Where, $V_0 F_s$ = 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 $\text{Resolution} = \frac{V_o F_s}{2^n - 1} = \frac{10.2}{2^8 - 1} = \frac{10.2}{255} = 40 \text{ mV/LSB}$

- Therefore, we can say that an input change of 1 LSB causes the output to change by 40 mv

2. Accuracy:

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

$$\text{Accuracy} = \frac{V_o F_s}{(2^n - 1)2} = \frac{10.2}{255 \times 2} = 20 \text{ mV}$$

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:

Input voltage range:

- 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)

Output voltage range:

- 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 = $5/2^8 = 19.5 \text{ mv}$

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.

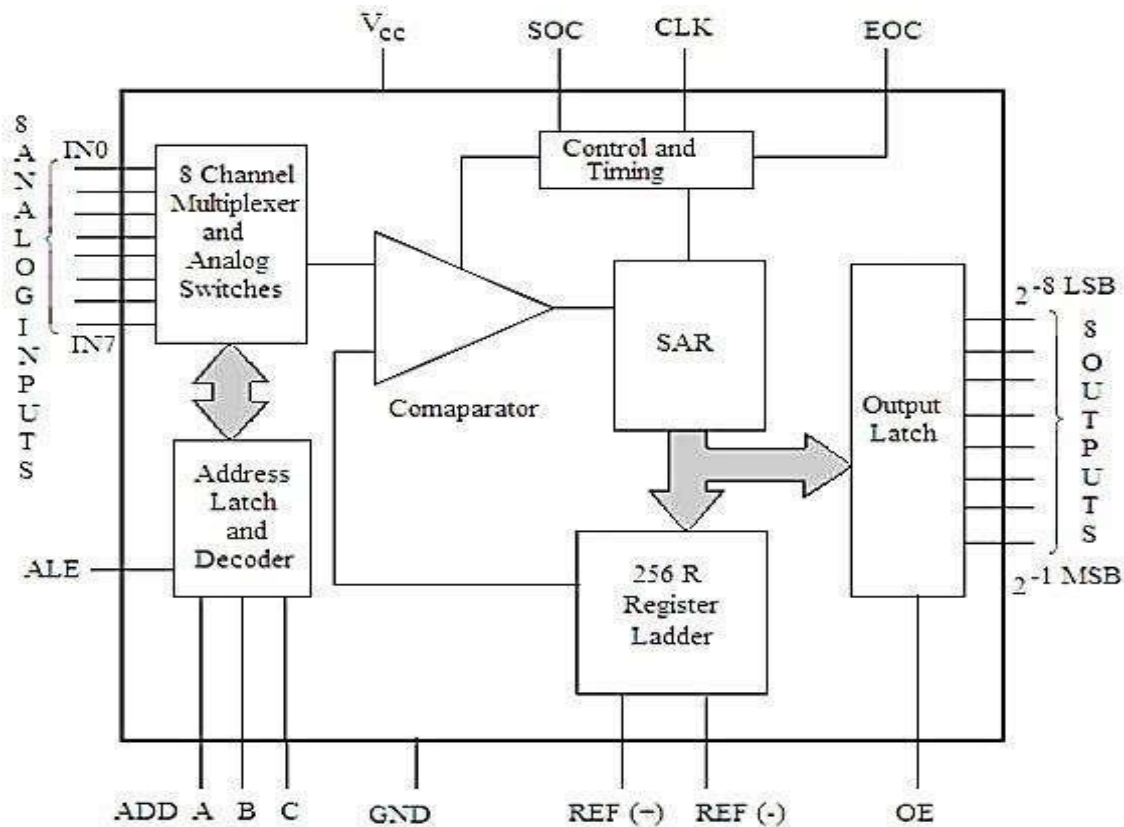
The ADC 0808/0809 Chip

- 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

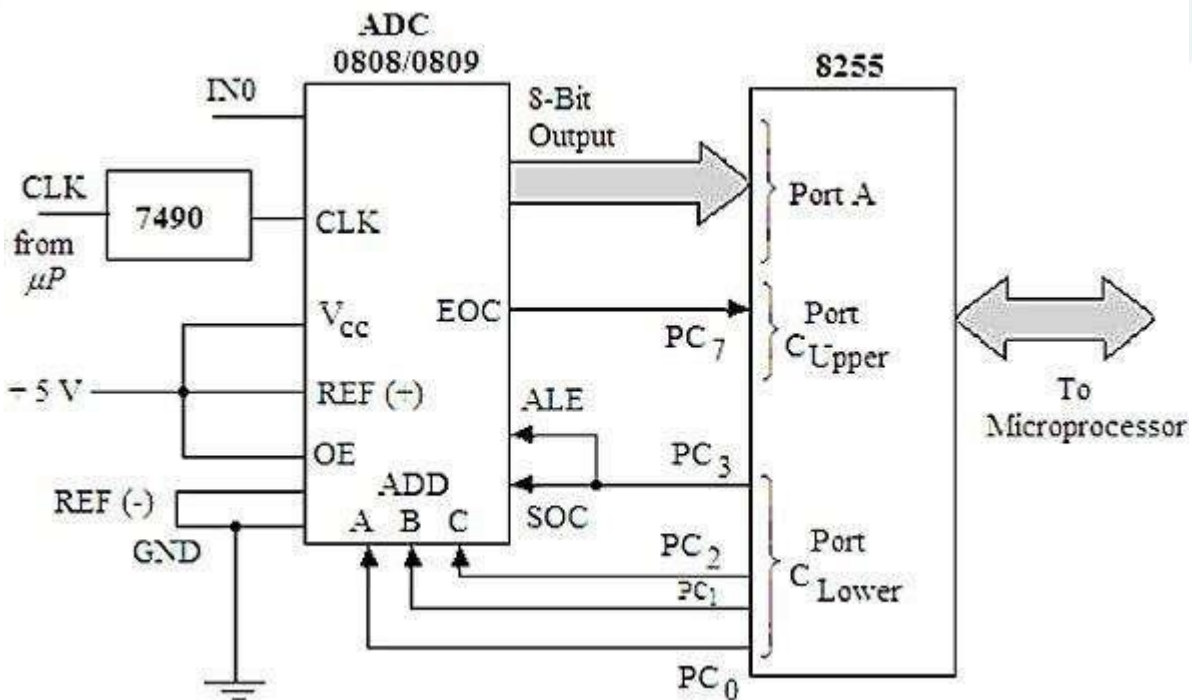
- The conversion speed is much higher
- The accuracy is also high
- It has minimal temperature dependence
- Excellent long term accuracy and repeatability
- Less power consumption

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 PC₇ 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 IN_0 as input pin, so the pin selection value will be 00H.

MVI A, 98H; Set Port A and Copper 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

IN_0

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 (PC_7)

RAL; Rotate left to check C_7 is 1.

JNC READ; if C_7 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

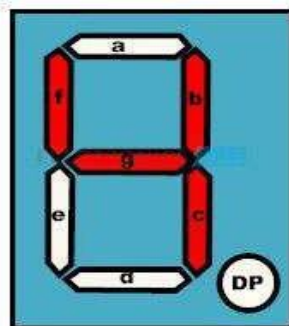
SEVEN SEGMENT DISPLAYS:

- 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:

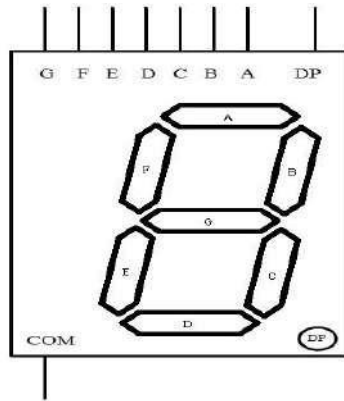
- Seven segment display works, by glowing the required respective LEDs in the numeral. The display is controlled using pins that are left freely. Forward biasing of these pins in a sequence will display the particular numeral or alphabet. Depending on the type of seven segment the segment pins are applied 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:

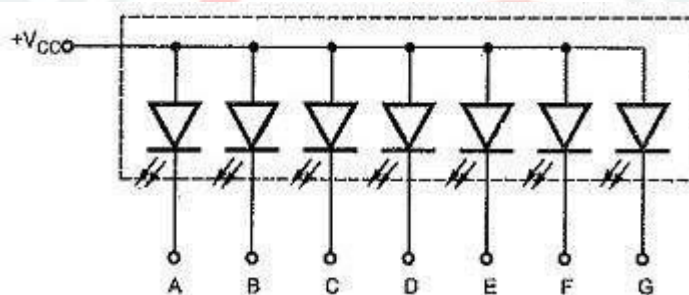
- Seven Segment Display Interfacing are generally used as numerical indicators and consists of a number of LEDs arranged in seven segments as shown in the Figure



- 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.

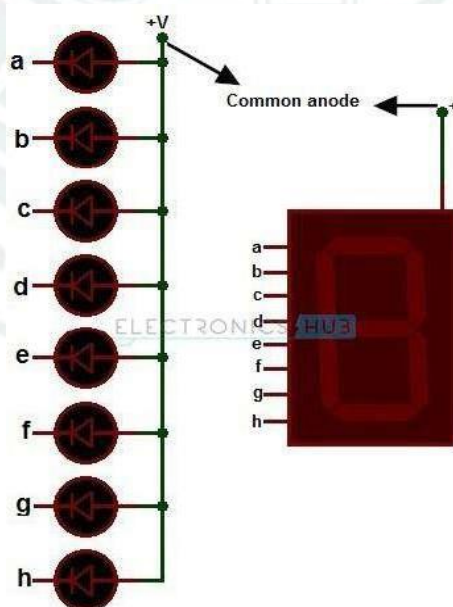
Common Anode type:

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



Common anode type

or



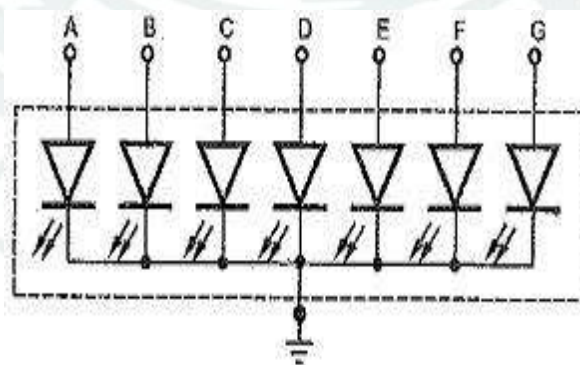
- In common anode type, all the anodes of 8 LED's are connected to the common terminal and cathodes are left free. Thus, in order to glow the LED, these cathodes have to be connected to the logic '0' and anode to the logic '1'.
- Below truth table gives the information required for driving the common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

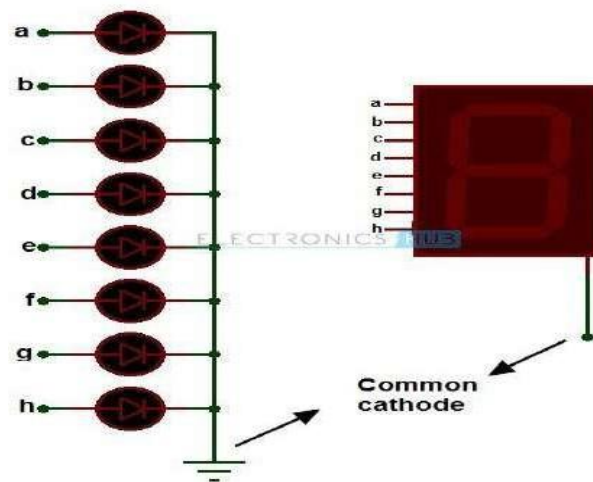
Common cathode type:

- 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



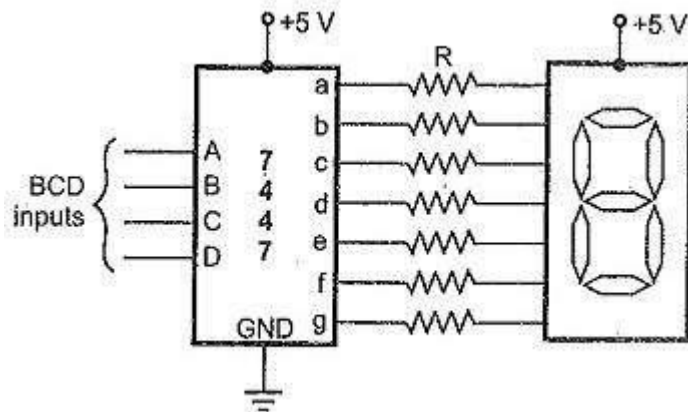
The truth table of seven segment display is shown below.

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:

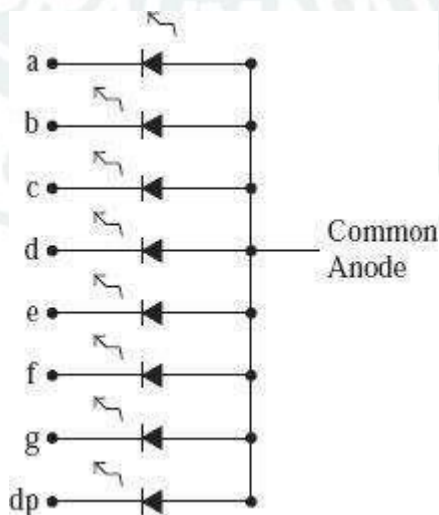
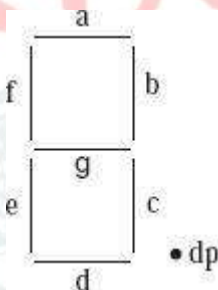
- It shows a circuit to drive a single, Seven Segment Display Interfacing, common anode LED display.
- For common anode, when anode is connected to positive supply, a low voltage is applied to a cathode to turn it on.
- Here, BCD to seven segment decoder, IC 7447 is used to apply low voltages at cathodes 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

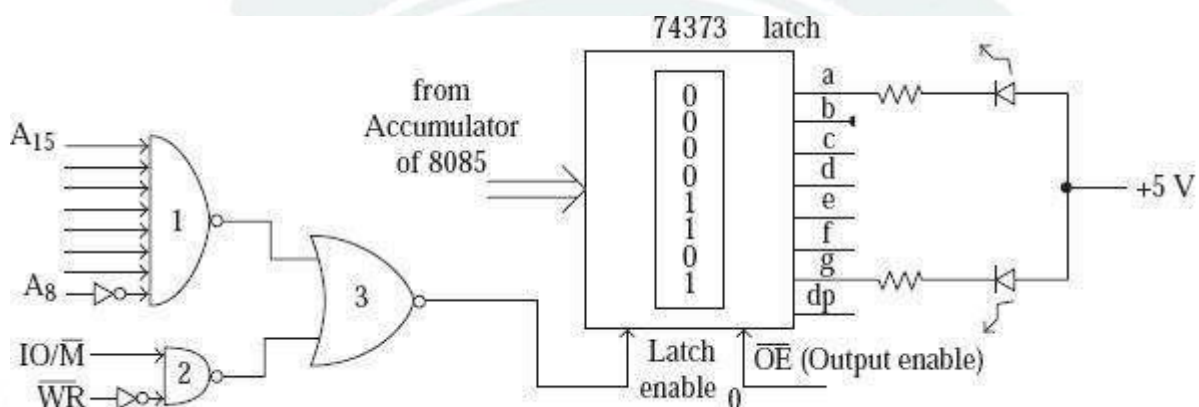
INTERFACING:

- 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.

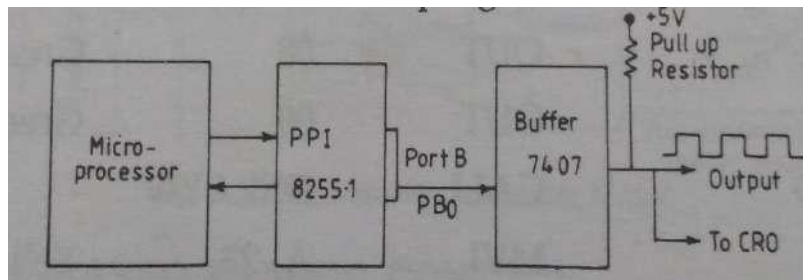
4.8 GENERATE SQUARE WAVES ON ALL LINES OF 8255:

- 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 PB₀ 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 contains three 8-bit ports which can be configured by software means to provide 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	0B	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		

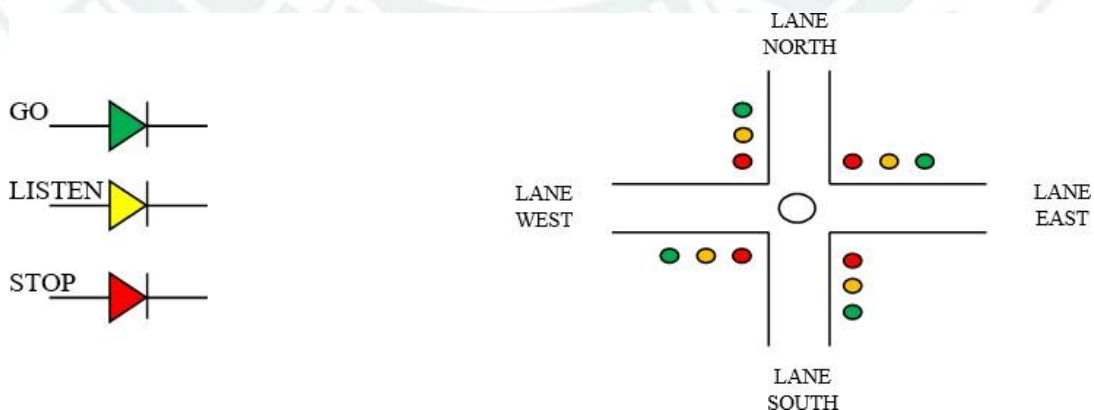
4.9 DESIGN INTERFACE A TRAFFIC LIGHT CONTROL SYSTEM USING 8255:

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

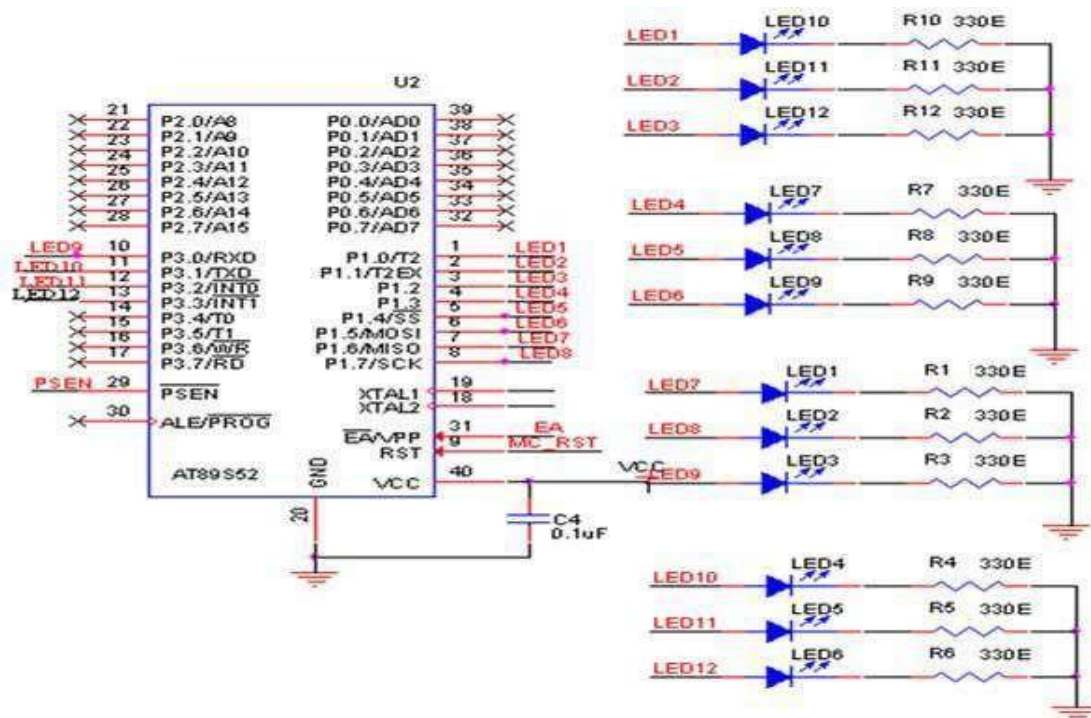
- Traffic lights alternate the right of way of road users by displaying lights of a standard color (red, yellow/amber, and green), using a universal color code (and a precise sequence to enable comprehension by those who are color blind).
- Illumination of the red signal prohibits any traffic from proceeding. 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).



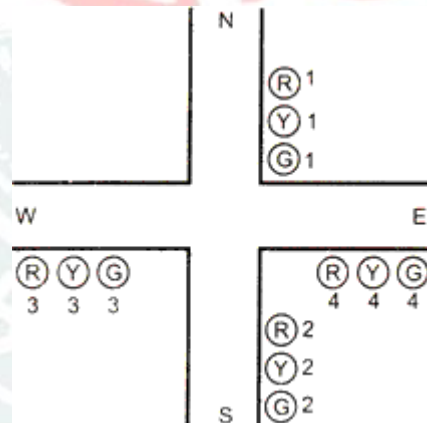
INTERFACING TRAFFIC LIGHT WITH 8085

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



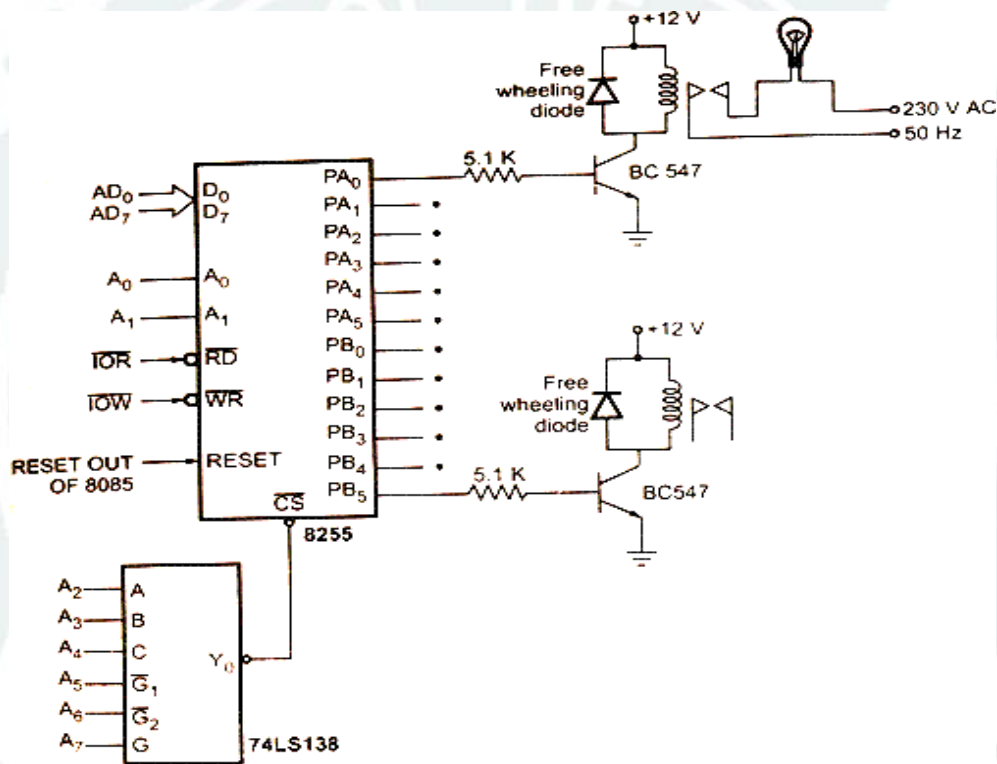
- The interfacing diagram to control 12 electric bulbs. Port A is used to control lights on N-S road and Port B is used to control lights on W-E road. Actual pin connections are listed in Table 1 below.

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								Address
	A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀	
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	= 80H
1	0	0	0	X	0	0	X	

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 (4010) 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 (1010) 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 (4010) 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 (1010) 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)

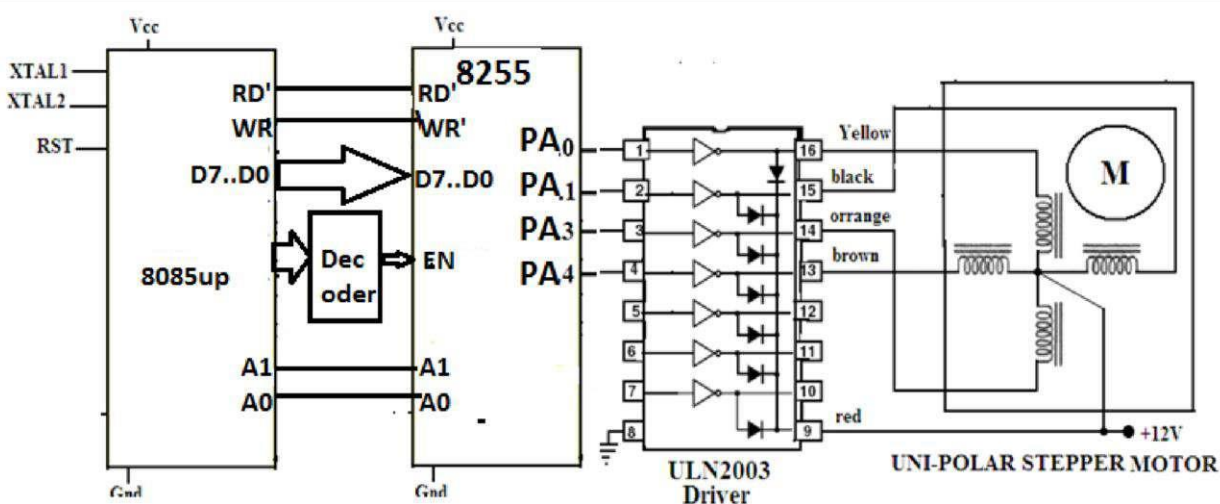
- Stepper motor driver circuits are available readily in the form of ICs.

- 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.

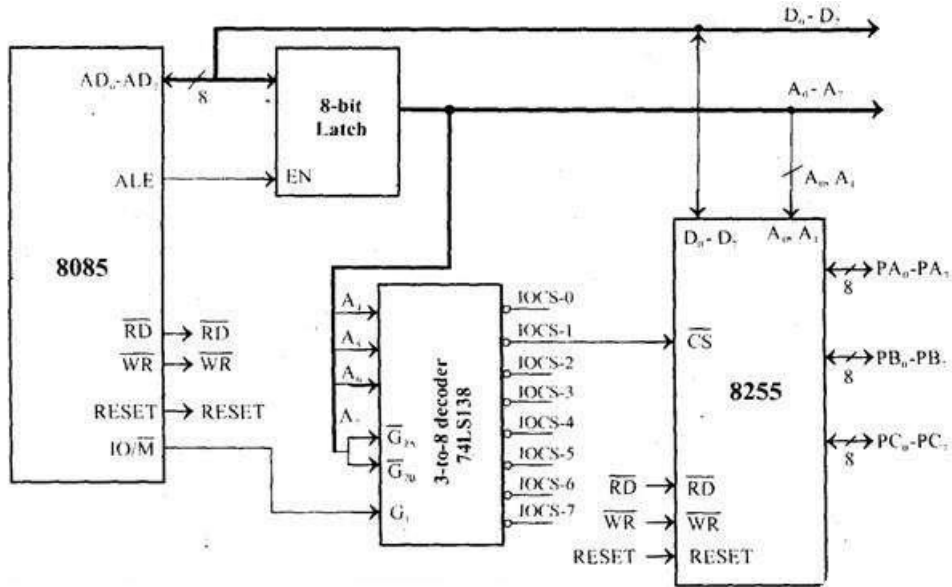
Operation:

- 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 $360/200 = 1.8^\circ$.
- 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                   ; sent to motor
PortA_Address         ; via port A of
                       8255
CALL DELAY            ; Delay
                       subroutine
JMP MAIN              ; Keep the
DELAY                motor rotating
Subroutine           continuously.
MVI C, FF             ; Load C with FF
                       -- Change it for
                       the speed
                       variation
LOOP1: MVI            ; Load D with FF
D,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.

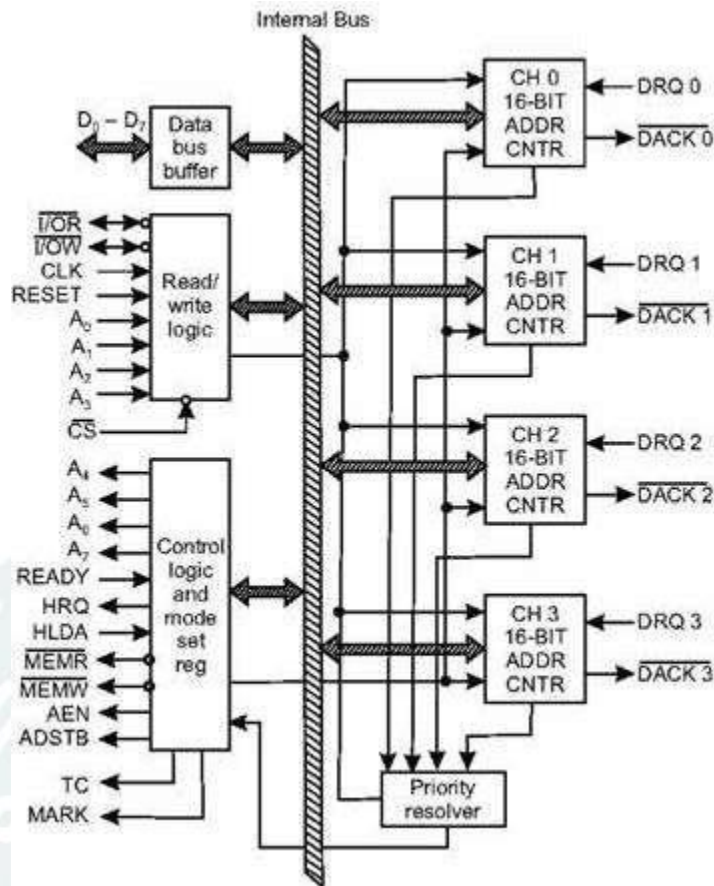
Features of 8257:

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**.

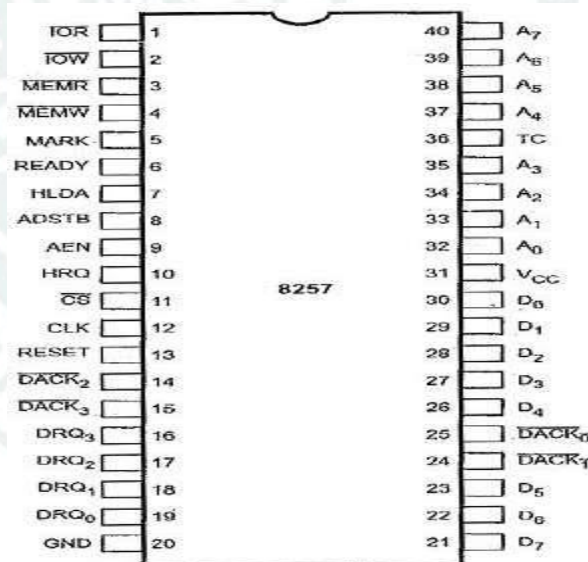
8257 Architecture:

The following image shows the architecture of 8257 –



8257 Pin Description:

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



DRQ₀–DRQ₃

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 DRQ₀ has the highest priority and DRQ₃ has the lowest priority among them.

DACK₀ – DACK₃

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.

D₀ – D₇

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.

A₀ - A₃

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.

A₄ - A₇

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}

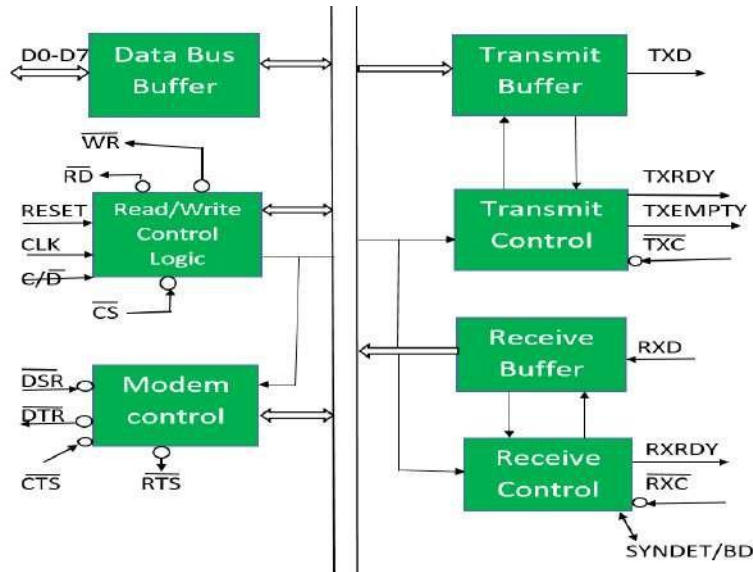
It is the power signal which is required for the operation of the circuit

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.

Modem control (modulator/demodulator):

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.