## 4.4 MICROPROCESSORS AND PERIPHERAL DEVICES

L T P

3 - 3

**RATIONALE**

The study of microprocessors in terms of architecture, software and interfacing techniques leads to the understanding of working of CPU in a microcomputer. The development in microprocessors of 32 bit architecture brings the students face-to-face with mainframe enabling them to get employment in R&D, assembly, repair and maintenance of hardware of microprocessors and computers. Microprocessors find application in process control industry. They also form a part of the electronic switching system between source and destination in long distance telecommunications. Thus the microprocessor is an area of specialization. Students of electronics and related engineering branches often use microprocessors to introduce programmable control in their projects, in industrial training.

**LEARNING OUTCOMES**

After undergoing the subject, the students will be able to:

- Describe all the internal parts and pins of 8085
- Write and Edit assembly language program using mnemonics
- Write, execute and debug assembly language programs for simple applications
- Interface various peripheral devices with microprocessor.
- Use various data transfer techniques in micro computers
- Describe the internal parts and pins of 8086

**DETAILED CONTENTS**

1. Evolution of Microprocessor (03 Periods)

   Typical organization of a microcomputer system and functions of its various blocks. Microprocessor, its evolution, function and impact on modern society

2. Architecture of a Microprocessor (With reference to 8085 microprocessor)

   (09 periods)

   Concept of Bus, bus organization of 8085, Functional block diagram of 8085 and function of each block, Pin details of 8085 and related signals, Demultiplexing of address/data bus generation of read/write control signals, Steps to execute a stored programme

3. Instruction Timing and Cycles (06 periods)

Instruction cycle, machine cycle and T-states, Fetch and execute cycle.

4. Programming (with respect to 8085 microprocessor) (12 periods)

Brief idea of machine and assembly languages, Machines and Mnemonic codes. Instruction format and Addressing mode. Identification of instructions as to which addressing mode they belong. Concept of Instruction set. Explanation of the instructions of the following groups of instruction set. Data transfer group, Arithmetic Group, Logic Group, Stack, I/O and Machine Control Group. Programming exercises in assembly language. (Examples can be taken from the list of experiments).

5. Memories and I/O interfacing (08 periods)

Concept of memory mapping, partitioning of total memory space. Address decoding, concept of peripheral mapped I/O and memory mapped I/O. Interfacing of memory mapped I/O devices.

6. Interrupts (03 periods)

Concept of interrupt, Maskable and non-maskable, Edge triggered and level triggered interrupts, Software interrupt, Restart interrupts and its use, Various hardware interrupts of 8085, Servicing interrupts, extending interrupt system

7. Data Transfer Techniques (03 periods)

Concept of programmed I/O operations, sync data transfer, async data transfer (hand shaking), Interrupt driven data transfer, DMA, Serial output data, Serial input data

8. Peripheral devices (02 periods)

## 8255 PPI, 8253 PIT and 8257 DMA controller

9 Architecture of 8086 Microprocessor (02 periods)

- Block diagram
- Minimum and Maximum mode
- Pin and Signals

**LIST OF PRACTICALS**

1. Familiarization of different keys of 8085 microprocessor kit and its memory map
2. Steps to enter, modify data/program and to execute a programme on 8085 kit
3. Writing and execution of ALP for addition and sub station of two 8 bit numbers
4. Writing and execution of ALP for multiplication and division of two 8 bit numbers
5. Writing and execution of ALP for arranging 10 numbers in ascending/descending order

6. Writing and execution of ALP for 0 to 9 BCD counters (up/down counter according to choice stored in memory)
7. Interfacing exercise on 8255 like LED display control
8. Interfacing exercise on 8253 programmable interval timer
9. Interfacing exercise on 8279 programmable KB/display interface like to display the hex code of key pressed on display
10. Use of 8085 emulator for hardware testing

## INSTRUCTIONAL STRATEGY

The digital systems in microprocessors have significant importance in the area of electronics. Adequate competency needs to be developed by giving sufficient practical knowledge in microprocessors (programming as well as interfacing). Help may be taken in the form of charts, simulation packages to develop clear concepts of the subject. Programming exercises other than the given in the list may be given to the students.

## MEANS OF ASSESSMENT

- Assignments and quiz/class tests, mid-term and end-term written tests
- Actual laboratory and practical work, exercises
- Viva-voce

## RECOMMENDED BOOKS


1. Microprocessor Architecture, Programming and Applications with 8080/8085 by Ramesh S Gaonker, Willey Eastern Ltd. New Delhi
2. Introduction to Microprocessor by Mathur ,Tata McGraw Hill Education Pvt Ltd , New Delhi
3. Microprocessor and Microcontrollers by Dr BP Singh, Galgotia Publications, New Delhi
4. Microprocessor and Applications by Badri Ram: Tata McGraw Hill Education Pvt Ltd , New Delhi
5. Microprocessor and Microcomputers by Refiquzzaman, Prentice Hall of India Ltd., New Delhi.
6. Microprocessor programming & applications.by sudhir Goyal, North Publication.
7. Digital Logic and Computer Design by Mano, M Morris; Prentice Hall of India, New Delhi

8. Digital Electronics by Rajaraman; Prentice Hall of India Ltd., New Delhi

9. e-books/e-tools/relevant software to be used as recommended by AICTE/HSBTE/NITTTR.

**Websites for Reference:**

http://swayam.gov.in

## SUGGESTED DISTRIBUTION OF MARKS

| Topic No. | Time Allotted (Periods) | Marks Allotted (%) |
|---|---|---|
| 1. | 03 | 05 |
| 2. | 09 | 20 |
| 3 | 06 | 15 |
| 4 | 12 | 25 |
| 5. | 08 | 15 |
| 6. | 03 | 05 |
| 7. | 03 | 05 |
| 8. | 02 | 05 |
| 9 | 02 | 05 |
| Total | 48 | 100 |

# UNIT-1 Evolution of Microprocessor

## Definition of the Microprocessor

The microprocessor is a programmable device that takes in numbers, performs on them arithmetic or logical operations according to the program stored in memory and then produces other numbers as a result.

## Evolution of Microprocessors

First Generation (4 - bit Microprocessors) the first generation microprocessors were introduced in the year 1971-1972 by Intel Corporation. It was named Intel 4004 since it was a 4-bit processor. It was a processor on a single chip. It could perform simple arithmetic and logical operations such as addition, subtraction, Boolean OR and Boolean AND.I had a control unit capable of performing control functions like fetching an instruction from storage memory, decoding it, and then generating control pulses to execute it. Second Generation (8 - bit Microprocessor)The second generation microprocessors were introduced in 1973 again by Intel. It was a first 8 - bit microprocessor which could perform arithmetic and logic operations on 8-bit words. It was Intel 8008, and another improved version was Intel 8088.Third Generation (16 - bit Microprocessor) The third generation microprocessors, introduced in 1978 were represented by Intel's 8086, Zilog Z800 and 80286, which were 16 - bit processors with a performance like minicomputers. Fourth Generation (32 - bit Microprocessors) Several different companies introduced the 32-bit microprocessors, but the most popular one is the Intel 80386.Fifth Generation (64 - bit Microprocessors)From 1995 to now we are in the fifth generation. After 80856, Intel came out with a new processor namely Pentium processor followed by Pentium Pro CPU, which allows multiple CPUs in a single system to achieve multiprocessing. Other improved 64-bit processors are Celeron, Dual, Quad, Octa Core processors.

## Basic Terms used in Microprocessor

**Instruction Set** - The group of commands that the microprocessor can understand is called Instruction set. It is an interface between hardware and software.

**Bus** - Set of conductors intended to transmit data, address or control information to different elements in a microprocessor. A microprocessor will have three types of buses, i.e., data bus, address bus, and control bus.

**IPC (Instructions per Cycle)** - It is a measure of how many instructions a CPU is capable of executing in a single clock.

**Clock Speed** - It is the number of operations per second the processor can perform. It can be expressed in megahertz (MHz) or gigahertz (GHz). It is also called the Clock Rate.

**Bandwidth** - The number of bits processed in a single instruction is called Bandwidth.

**Word Length** - The number of bits the processor can process at a time is called the word length of the processor. 8-bit Microprocessor may process 8 -bit data at a time. The range of word length is from 4 bits to 64 bits depending upon the type of the microcomputer.

## Working of Microprocessor

The microprocessor follows a sequence to execute the instruction: Fetch, Decode, and then Execute. Initially, the instructions are stored in the storage memory of the computer in sequential order. The microprocessor fetches those instructions from the stored area (memory), then decodes it and executes those instructions till STOP instruction is met. Then, it sends the result in binary form to the output port. Between these processes, the register stores the temporary data and ALU (Arithmetic and Logic Unit) performs the computing functions.

## Features of Microprocessor

Low Cost - Due to integrated circuit technology microprocessors are available at very low cost. It will reduce the cost of a computer system.

High Speed - Due to the technology involved in it, the microprocessor can work at very high speed. It can execute millions of instructions per second.

Small Size - A microprocessor is fabricated in a very less footprint due to very large scale and ultra large scale integration technology. Because of this, the size of the computer system is reduced.

Versatile - The same chip can be used for several applications, therefore, microprocessors are versatile.

Low Power Consumption - Microprocessors are using metal oxide semiconductor technology, which consumes less power.

Less Heat Generation - Microprocessors uses semiconductor technology which will not emit much heat as compared to vacuum tube devices.

Reliable - Since microprocessors use semiconductor technology, therefore, the failure rate is very less. Hence it is very reliable.

Portable - Due to the small size and low power consumption microprocessors are portable.

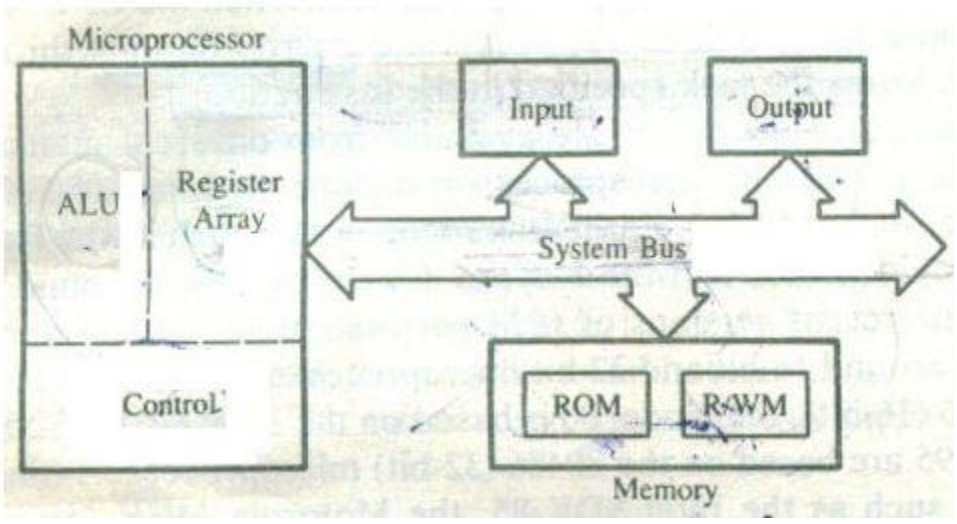
## What is the 8085 Microprocessor?

Generally, the 8085 is an 8-bit microprocessor, and it was launched by the Intel team in the year of 1976 with the help of NMOS technology. This processor is the updated version of the microprocessor. The configurations of 8085 microprocessor mainly include data bus-8-bit, address bus-16 bit, program counter-16-bit, stack pointer-16 bit, registers 8-bit, +5V voltage supply, and operates at 3.2 MHz single segment CLK. The applications of 8085

microprocessor are involved in microwave ovens, washing machines, gadgets, etc. The features of the 8085 microprocessor are as below:

- This microprocessor is an 8-bit device that receives, operates, or outputs 8-bit information in a simultaneous approach.
- The processor consists of 16-bit address and 8 bit data lines and so the capacity of the device is 216 which is 64KB of memory.
- This is constructed of a single NMOS chip device and has 6200 transistors
- A total of 246 operational codes and 80 instructions are present
- As the 8085 microprocessor has 8-bit input/output address lines, it has the ability to address 28 = 256 input and output ports.
- This microprocessor is available in a DIP package of 40 pins
- In order to transfer huge information from I/O to memory and from memory to I/O, the processor shares its bus with the DMA controller.
- It has an approach where it can enhance the interrupt handling mechanism
- An 8085 processor can even be operated as a three-chip microcomputer using the support of IC 8355 and IC 8155 circuits.
- It has an internal clock generator
- It functions on a clock cycle having a duty cycle of 50%

## Basic Organization of Microcomputer




1. **MP**

i. **ALU**

This unit executes all arithmetic and logical operations as specified by instruction set; and produces output.

The results of addition, subtraction, and logical operations (AND, OR, XOR) are stored in the registers or in memory unit or sent to output unit.

**ii. Register unit**

Consists of various registers.

Used for temporary storage of data during execution of data.

**iii. CU**

Controls the operations of different instructions.

Provides necessary timing and control signals to all the operations in the MP and peripherals including memory.

**2. Memory**

Stores binary information such as instruction and data, and provide these information to MP when required.

To execute programs, the MP reads data and instructions from memory and performs the computing operations.

**3. System bus**

The system bus is a communication path between MP and peripherals.

It is used to carry data, address and control signals.

It consists:

Data bus: carries data

Address bus: carries address

Control bus: carries control signals

**4. I/O bus**

Input unit is used to input instruction or data to the MP externally.

Output unit is used to carry out the information from the MP unit.

# UNIT-2 Architecture of a Microprocessor

## Architecture of a Microprocessor

**Definition:** 8085 is an 8-bit microprocessor as it operates on 8 bits at a time and is created with N-MOS technology. This microprocessor exhibits some unique characteristics and this is the reason it still holds popularity among the microprocessors.

Basically, 8085 was the first commercially successful microprocessor by Intel. As some of the architectural drawbacks associated with 8080 was also eliminated by 8085.

The size of the data bus of 8085 is 8 bits while that of the address bus is 16. Therefore, can address 64 KB (i.e., 216) memory. Also, as it can perform 8-bit operation thus the size of ALU is also 8-bit.

It also provides operational advantages, as 8085 needs a single +5V supply with only one clock single of width 320 ns. While 8080 requires 3 power supply lines and 2 clock signals of 500 ns.

**Architecture of 8085 Microprocessor**

The architecture of 8085 microprocessor provides the idea about what are the operations to be executed and how these are performed.

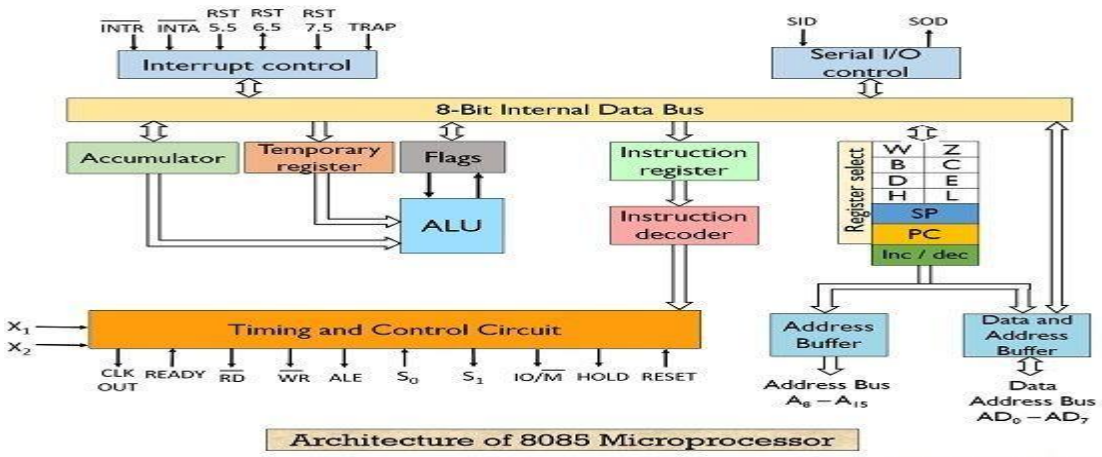
It can perform operations that are given below:

1. Operates on and stores 8-bit data.
2. It executes arithmetic and logic operations.
3. 8085 also sequences the instructions to be executed.
4. Stores data temporarily.

However, in order to perform all such operations, the processor needs a control unit, arithmetic logic unit, registers, buses etc.

**Working of Microprocessor**

The microprocessor follows a sequence to execute the instruction: Fetch, Decode, and then Execute. Initially, the instructions are stored in the storage memory of the computer in sequential order. The microprocessor fetches those instructions from the stored area (memory), then decodes it and executes those instructions till STOP instruction is met. Then, it sends the result in binary form to the output port. Between these processes, the register stores the temporary data and ALU (Arithmetic and Logic Unit) performs the computing functions.




## Functional Units of 8085:

**1. Registers**: These are nothing but set of flip flops. These are basically used to hold (store) the data.

**General purpose registers**– 8085 microprocessors contain 6 general purpose registers that are present inside the microprocessor and stores 8-bit data in order to execute a program.

These general purpose registers are B, C, D, E, H and L. These registers can be combined to form pairs – BC, DE and HL in order to execute the 16-bit operation.

These are programmable registers that means these registers are accessed by the programmer to insert and transfer the data by making use of instructions.

**Temporary registers:** These registers are used by the ALU to store the data on temporary basis and these are not accessed by the programmer. These are of 2 types:

1. Temporary data register – It is an 8-bit register that holds the operand and provides it to the ALU for program execution. Also, the immediate results are stored by the ALU in this register.

2. W and Z register – These registers are also used to hold the temporary values. It is used by the control section of the microprocessor so as to store the data during operations.

**2. Program Counter (PC):** It is basically a special purpose register that is used to store the memory location of the instruction to be performed. As it is clear that in order to fetch an instruction from the memory the microprocessor needs to know about its address.

It is a 16-bit register as it stores address. This register is used by the microprocessor to line up the instructions that are to be executed in a sequential manner.

It functions in such a way that it fetches the opcode from one memory location and simultaneously gets incremented by the next memory location. Thus, it provides sequencing of the program to be executed.

**3. Stack Pointer (SP):** It is also a 16-bit register and is a part of memory. The data is stored in the stack in serial format and stack pointer generally stores the address of the last data element stored in the stack. Thus the stack is based on LIFO.

Whenever a new data is added in the stack, then the stack pointer starts pointing towards the very next memory location.

As against, when a data element is removed from the stack, then the stack pointer points to previous occupied memory location.

**4. Accumulator:** It is an 8-bit register that stores the result of the operation performed by the ALU. It is also known as register A.

**5. Flags:** Flag register basically holds the status of the current result generated by the ALU and not the actually generated result. Thus we can say it is used to test the data conditions.

8085 has 5 flags that shows 5 different data conditions. These are carry, sign, zero, parity and auxiliary carry flags.

However, the mostly used are: sign, carry and zero.

A flag is a flip flop. It indicates some condition produced by the execution of an instruction. The flag register of 8085 microprocessor consists of 5 flags. The flag register is connected to ALU. When an operation is performed by ALU the result is transferred on data bus and status of result will be stored in flip flops. The different flags and their positions in flag register are shown in following fig.

Flag register of 8085 microprocessor

**1) The carry flag (CF):-**

This flag is set whenever there has been a carry out of, or a borrow into, the higher order bit of the result. The flag is used by the instructions that add aNd subtract multi byte numbers.

1-carry out from MSB bit on addition or borrow into MSB bit on subtraction

0-no carry out or borrow into MSB bit

**2) The parity flag (PF):-**

This flag is set whenever the result has even parity, an even number of 1 bits. If parity is odd, PF is cleared.

1-low byte has even number of 1 bits

0-low byte has odd parity

**3) The auxiliary carry flag (AF):-**

This flag is set whenever there has been a carry out of the lower nibble into the higher nibble or a borrow from higher nibble into the lower nibble of an 8 bit quantity, else AF is reset. This flag is used by decimal arithmetic instructions.

1-carry out from bit 3 on addition or borrow into bit 3 on subtraction

0-otherwise

**4) The zero flag (ZF):-**

This flag is set, when the result of operation is zero, else it is reset.

1-zero result

0-non-zero result

**5) The sign flag (SF):-**

This flag is set, when MSB (Most Significant Bit) of the result is 1. Since negative binary numbers are represented in the 8085 CPU in standard two's complement notation, SF indicates sign of the result.

1-MSB is 1 (negative)

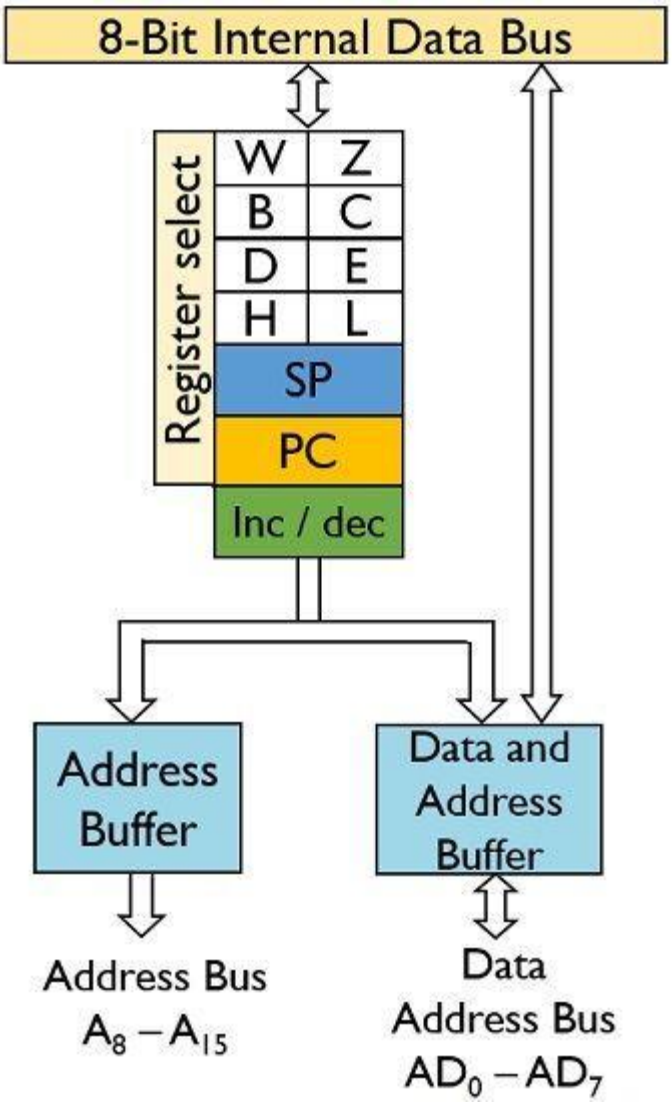
0-MSB is 0 (positive)

**Working of 8085 Microprocessor**

Now after having an idea about the functional units of the 8085 microprocessor, let us proceed further to understand the operation of the 8085 microprocessor.

We already know that the function of a microprocessor is to execute instructions. Also, in order to execute an instruction, it first needs to be fetched then decoded and then executed. And in order to fetch an instruction, firstly the address of the instruction must be known.

The address of the instruction is present in the program counter. This address is then placed on the 16-bit address bus and is then forwarded towards the memory. From the memory, the instruction present at that particular memory location is fetched through the 8-bit data bus.




Here, in the above figure, we can clearly see that we have combine used data bus and address bus so as to reduce the number of lines. As we know that at a particular time the processor will access either the data bus or the address bus.

Further when the instruction is fetched from the memory, then through internal buses the instruction is provided to the instruction register. At this particular point of time fetching the instruction from the memory is over.

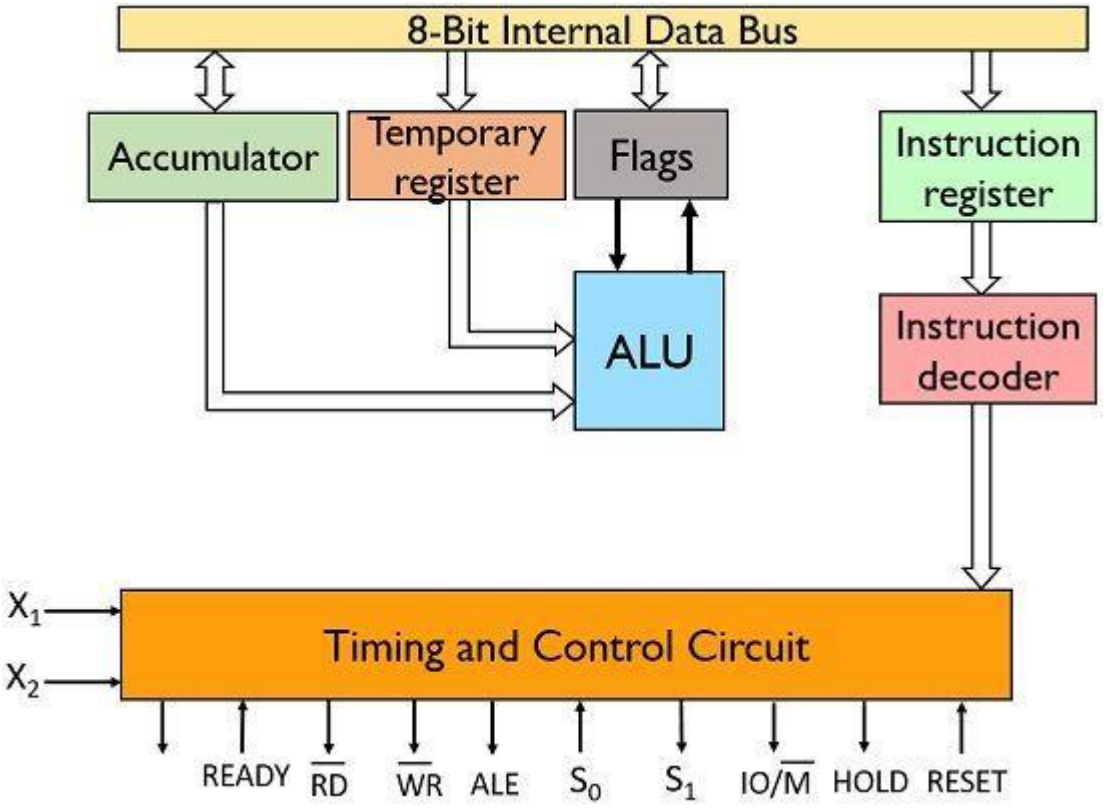
Now, it's time for the processor to decode the instruction. So, it is then fed to the instruction decoder.

We already have the idea that both data and instruction in the memory is stored in the form of an opcode. So, the fetched opcode is analyzed by the instruction decoder present inside the processor in order to execute the instruction.

But, a noteworthy point over here is that, after an instruction is fetched from the memory, then PC increments itself thereby providing the address location of the next instruction. As PC does not play any role in decoding and executing.

However, after execution of first instruction, the next is fetched form the memory.

Now, this is all about fetching and decoding, now what about the execution of the instruction.




In the above figure, we can see the timing and control circuit. This circuit basically sends the control signals to the various units of the microprocessor to execute the instruction.

Suppose, the instruction is ADD A and B. This simply tells the ALU to add the data present in B register with the data present in accumulator i.e., A register. But, in 8085 the decoded instruction is simply ADD B. So, automatically, the ALU adds the value present in the accumulator with the data in register B.

Also, in 8085 the outcome of the operation is stored register A which is nothing but an accumulator.

Further when an instruction ADD C is decoded by the 8085, then data at C register is added to the data present in register A and is stored at register A. Due to such cumulative action, register A is termed as an accumulator.

Basically, the timing and control circuit timely sends the signals to the accumulator to release its value for proceeding execution. Also, this timing and control circuit sends signals to the register select that tells it to choose the particular register.

Once the data is fetched from a particular register then it is stored in the temporary register and it is used by the ALU. It is to be kept in mind that a programmer can only access the general purpose register as the temporary register is used by the processor to hold the 2nd operand of the operation.

Now, once the operation is executed, then the result is fed to the accumulator through the data bus. But a flag register is also present that holds the status of the result present at the accumulator.

As we have also discussed in the functional unit that a flag register holds a sign, carry etc. related information of the generated result.

After every instruction execution performed by the ALU, the status of the flag register gets changed. So, ALU produces the result and its status simultaneously after each operation.

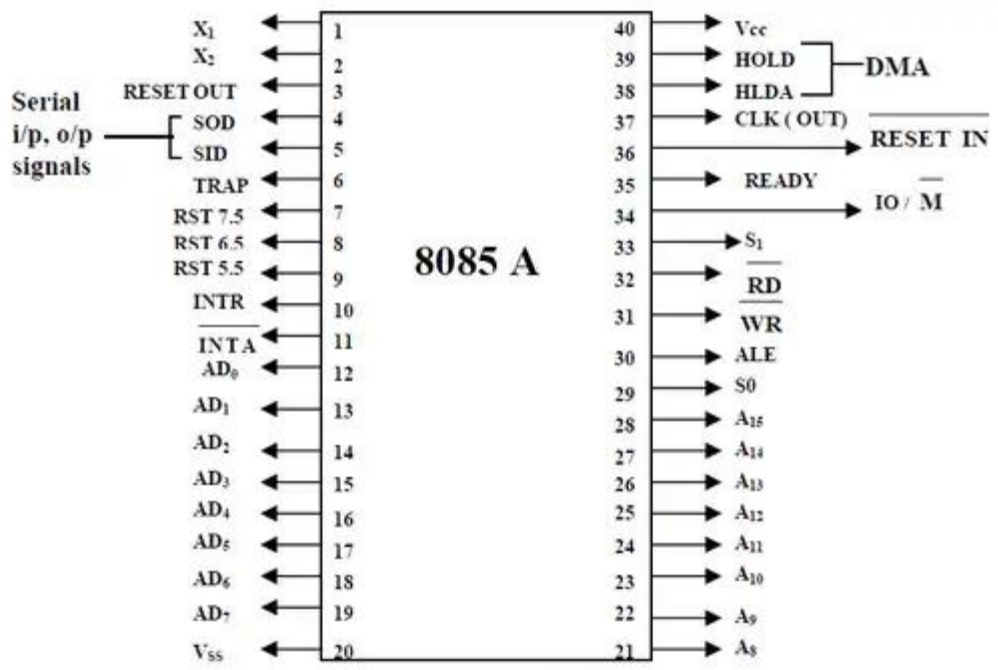
As we have already discussed that W and Z are the temporary registers but these are not accessed by the programmer as both are used by the processor to hold the temporary value stored by it.

So, this is all about the block diagram and working of 8085 microprocessor.

**Applications of 8085 microprocessor**

8085 finds its major applications in programmable calculators as well as in numerical control and environment monitoring systems. These are also used in switching, banking and financial systems.

## Pin Diagram 8085




Pin Diagram of 8085

The signals of this 40 pin IC is grouped into 7 categories, which are given below:

**Power supply and clock signals**

**Data bus**

**Address bus**

**Serial I/O ports**

**Control and status signals**

**Interrupts and externally generated signals**

**Direct memory access**

These are the categories among which the 40 pin configuration of 8085 is divided. So, let us proceed to understand the role of each pin inside the 8085 microprocessor. Pin Description of 8085 Microprocessor1.Power supply and clock signals: In 40 pin configuration, 4 pins are allotted to this particular category.

VCC – Pin number 40 denotes VCC, and an external power supply of + 5 V is provided at this pin.

VSS – Its pin number is 20. This pin shows the grounded connection of the microprocessor.

X1 and X2 – These are represented by pin number 1 and 2 respectively in the pin configuration. These 2 pins are connected with a crystal or LC network to maintain the internal frequency of the clock generator.

CLK (OUT) – It is the 37th pin of the 8085 IC and acts as the system clock that keeps the record of time duration required by each operation to get completed.

2. Address Bus – This category contains 8 pins. The address bus has 16 lines i.e.; it can carry 16 bits at a time. However, out of 16, 8 are multiplexed with the data bus and the leftover 8 are separately shown by pin number 21 to 28 in the pin configuration. These are used to carry the address of data and instruction from the processor to the memory location and is unidirectional in nature. These are denoted by A8 to A15 that represents the 8 MSB of the memory location or input-output address.3.Data Bus with multiplexed address bus – This category also contains 8 pins.The size of the data bus of the 8085 microprocessor is 8 bits. However, to reduce the number of bus lines these 8-bit data bus lines are multiplexed with the 8-bit address bus. These are shown by pin number 12 to 19. The address bus is denoted by A whereas the data bus is denoted by D. The pin configuration denotes the lower order multiplexed address and data bus bits from AD0 to AD7.

We have already discussed that the address bus contains the address of the desired memory location from where the data or instruction is to be fetched. While the data bus contains the data or instruction that is needed to be fetched from the memory.4.Serial I/O ports: It has basically 2 pins.

SID – SID denotes serial input data pin and its pin are numbered as 5. With this pin, data is serially fed to the processor directly through the input devices.

SOD – SOD denotes serial output data pin and its pin number is 4, in the pin configuration of 8085. Once the data is processed in the microprocessor then this pin represents bit by bit results at the output devices.

5. Control and status signals: Basically, 6 pins of the pin configuration are used by control and status signals.

ALE – ALE is an acronym for address latch enable and is pin number 30 in the configuration. We know that 8 lower order bits of the 16-bit address bus are multiplexed with the 8-bit data bus.

This pin gets enabled at the time when the address is present at the multiplexed address and data bus. Otherwise, it gets disabled showing the absence of an address on the bus.

RD – This pin is numbered 32 in the configuration and a low signal in this pin shows the read operation either from I/O devices or from the memory unit. Thereby indicating that the data bus is now in a state or position to accept the data from the memory or I/O devices.

WR – It is the 31st pin in the pin diagram and a low signal in this pin represents the write operation at the memory or I/O devices. This indicates that the data present in the data bus is to be written into the desired memory address or I/O device by the processor.

IO/M – It is pin number 34 and indicates the selection of a memory address or input-output device. This shows whether the read/write operation is to be carried out at the memory location or at the I/O device.

The low signal at this pin shows that operation is performing over memory location. As against, a high signal at this pin represents the operation at I/O device.

S0 and S1 – The pins S0 and S1 represent the status signal at pin number 29 and 33 respectively. These signals show the type of recent operation of the microprocessor. The table below represents the status of the data bus under different conditions:

6. Interrupts and Externally generated signals: Interrupts are the signals that are generated to break the sequence of an ongoing operation. When an interrupt signal is generated then CPU immediately stops its recent task under operation and switches to some other program known as interrupt service routine (ISR).However, after handling ISR, the CPU gets back to its main program for execution.

In the pin configuration, 5 types of interrupts are shown by 5 different pins from pin number 6 to 10. These pins are used to manage the interrupt. Basically, there exist 2 types of interrupts:Maskable Interrupt and Non- maskable interrupt Out of the 5 major interrupts 4 are the maskable interrupts. These are INTR, RST5.5, RST6.5, RST7.5 and are easily manageable interrupts. However, TRAP is a non-maskable interrupt and holds the topmost priority among all interrupts in the 8085 microprocessor.

RESET IN – It is pin number 36 in the pin diagram. An active low signal at this pin resets the PC of the microprocessor to 0. Or we can say, after resetting the PC holds its initial memory address.

RESET OUT – It is the 3rd pin in the pin diagram. This pin generates a signal to provide information about the resetting of the microprocessor. Also, we can say that once a processor is reset then all the connected devices must also be reset.

So, enabling this signal shows the resetting of the interconnected devices.

INTA: It is the 11th pin of the 8085 pin configuration. A signal at this pin acknowledges the generated interrupt.

7. Direct Memory Access (DMA) :We are aware of the fact that memory and I/O devices are connected with each other by the microprocessor. So, the intermediator i.e., CPU manages the data transfer between the input-output device and memory. However, when data in a large amount is to be transferred between I/O devices and memory the CPU gets disabled by tri-stating its buses. And this transfer is manageable by external control circuits. The DMA has 2 pins.

HOLD – This signal is generated at pin number 39. This pin generates a signal to notify the processor that more than one request is present to access the data and address bus.

When this signal gets enabled, the CPU frees the bus after completion of the recent operation. Once the hold signal gets disabled, the processor can access the bus again.

HLDA -This signal is generated at pin number 38. This signal is enabled at the time when the processor gets HOLD signal and it releases HLDA i.e., hold acknowledge signal. In order to show that the multiple requests are kept on hold and will be considered once the bus gets free after the recent operation.

After the disabling of hold request, the HLDA signal becomes low.

READY -This is the 35th numbered pin in the pin diagram that maintains synchronization between the processor and peripherals, memory. It is clear that a microprocessor has a much faster response than peripherals and memory.

So, this pin is enabled when the processor as well as the peripherals and memory both become ready to begin the next operation. In the case when the READY pin is disabled, then the microprocessor is in the WAIT state.

# UNIT-3 Instruction Timing and Cycles

## Instruction Cycle &Machine Cycle in 8085 Microprocessor:

The Program and data which are stored in the memory are used externally to the microprocessor for executing the complete instruction cycle. Thus to execute a complete instruction of the program, the following steps should be performed by the 8085 microprocessor.

- Fetching the opcode from the memory;
- Decoding the opcode to identify the specific set of instructions;
- Fetching the remaining Bytes left for the instruction, if the instruction length is of 2 Bytes or 3 Bytes;
- Executing the complete instruction procedure.

The given steps altogether constitute the complete instruction cycle. The above instructions are assumed by us for being in the memory, at the specified locations allocated for the memory.

The points to be noted as without fetching of the opcode from the memory the complete instruction would remain incomplete. Secondly decoding should be done, thirdly the fetching process should be done depending on the instruction length. Thirdly the complete execution process should be carried out to complete the entire process of execution.

Each instruction in 8085 microprocessor consists of two part- operation code (opcode) and operand. The opcode is a command such as ADD and the operand is an object to be operated on, such as a byte or the content of a register.

**Instruction Cycle:** 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 complete one operation; accessing either the memory or I/O device. A machine cycle consists of three to six T-states.

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

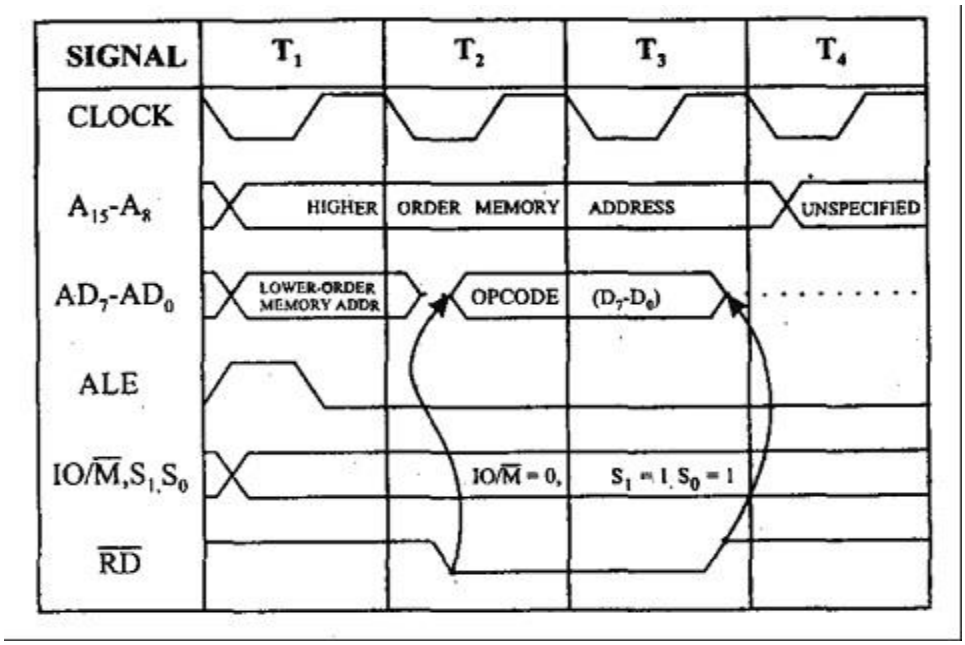
The seven Machine Cycle in 8085 Microprocessor are :

1. **Opcode Fetch Cycle**
2. **Memory Read**
3. **Memory Write**

4. **I/O Read**
5. **I/O Write**
6. **Interrupt Acknowledge**
7. **Bus Idle**

**1. Opcode Fetch Cycle:** Each instruction of the microprocessor has one byte Opcode. The Opcode is stored in memory. In order to fetch the Opcode from memory, processor executes the Opcode Fetch machine cycle. So, every instruction starts with Opcode Fetch machine (OFM) cycle. The time taken by the microprocessor to execute the Opcode Fetch cycle is 4T (T- states). Inorder to fetch the Opcode from memory, the first 3 T-states are used. The remaining T-state is used for internal operations by the microprocessor.

The timing diagram for Opcode Fetch machine cycle is shown in figure.



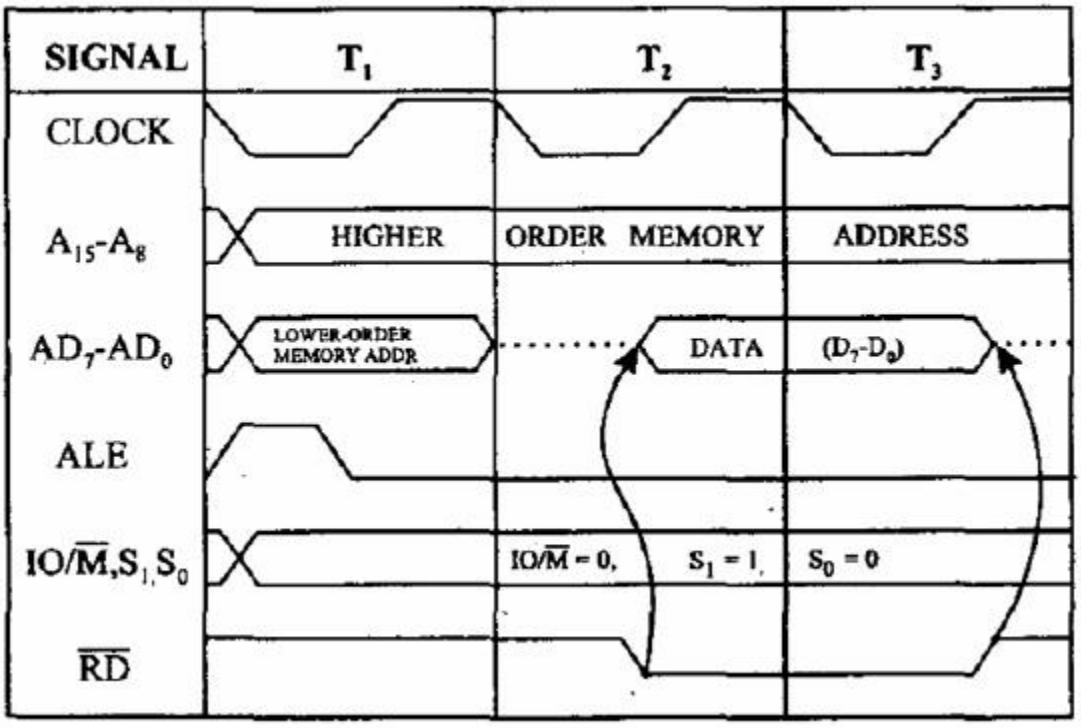

The steps in Opcode Fetch machine cycle are given in table.

| S. No | T state | Operation |
|---|---|---|
| 1 | $T_1$ | The microprocessor places the higher order 8-bits of the memory address on A15 – A8 address bus and the lower order 8-bits of the memory address on AD7 – AD0 address / data bus. |
| 2 | | The microprocessor makes the ALE signal HIGH and at the middle of T1 state, ALE signal goes LOW. |
| 3 | | The status signals are changed as IO/*M*' = 0, S1 =1 and S0 = 1. These status signals do not change throughout the OF machine cycle. |

| | | |
|---|---|---|
| 4 | $T_2$ | The microprocessor makes the RD' line LOW to enable memory read and increments the Program Counter. |
| 5 | | The contents on D7 – D0 (i.e. the Opcode) are placed on the address / data bus. |
| 6 | $T_3$ | The microprocessor transfers the Opcode on the address / data bus to Instruction Register (IR). |
| 7 | | The microprocessor makes the RD' line HIGH to disable memory read. |
| 8 | $T_4$ | The microprocessor decodes the instruction. |

**2. Memory Read Cycle: S**ingle byte instructions require only Opcode Fetch machine cycles. But, 2-byte and 3-byte instructions require additional machine cycles to read the operands from memory. The additional machine cycle is called Memory Read machine cycle. For example, the instruction MVI A, 50H requires one OF machine cycle to fetch the operand from memory and one MR machine cycle to read the operand (50H) from memory. The MR machine cycle takes 3 T-states. The timing diagram for Memory Read machine cycle is shown in figure.

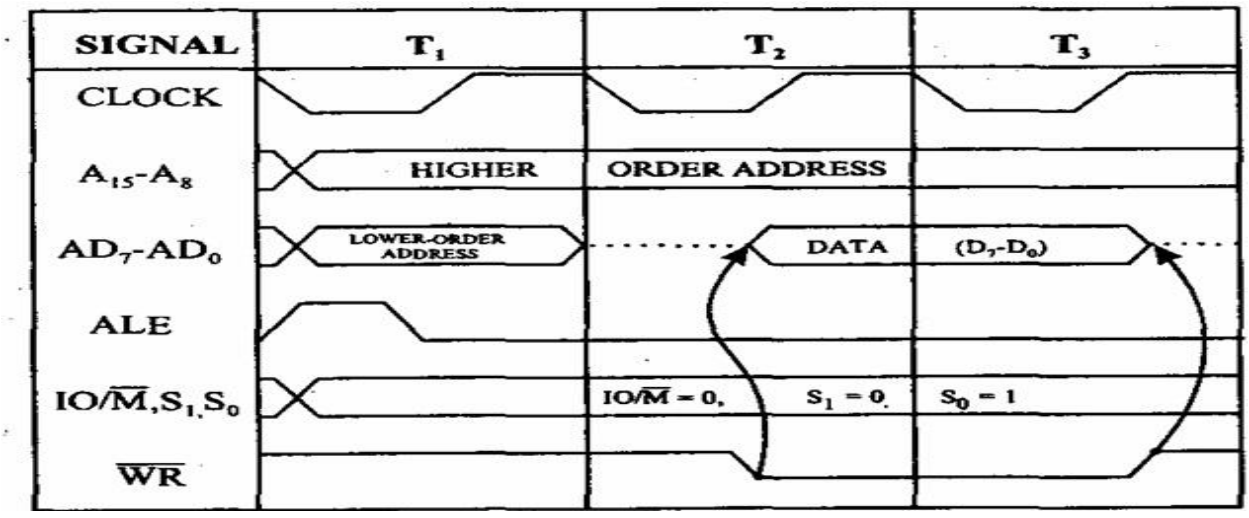
Timing Diagram for Memory Read Machine Cycle




The steps in Memory Read machine cycle are given in table.

| S. No | T state | Operation |
|---|---|---|
| 1 | $T_1$ | The microprocessor places the higher order 8-bits of the memory address on A15 – A8 address bus and the lower order 8-bits of the memory address on AD7 – AD0 address / data bus. |
| 2 | | The microprocessor makes the ALE signal HIGH and at the middle of T1 state, ALE signal goes LOW. |
| 3 | | The status signals are changed as IO/*M*' = 0, S1 =1 and S0 = 0. These status signals do not change throughout the memory read machine cycle. |
| 4 | $T_2$ | The microprocessor makes the RD' line LOW to enable memory read and increments the Program Counter. |
| 5 | | The contents on D7 – D0 (i.e. the data) are placed on the address / data bus. |
| 6 | $T_3$ | The data loaded on the address / data bus is moved to the microprocessor. |
| 7 | | The microprocessor makes the RD' line HIGH to disable the memory read operation. |

3. **Memory Write Cycle:** Microprocessor uses the Memory Write machine cycle for sending the data in one of the registers to memory. For example, the instruction STA 5000H writes the data in accumulator to the memory location 5000H. The MW machine cycle takes 3 T-states. The timing diagram for Memory Write machine cycle is shown in figure.




Timing Diagram for Memory Write Machine Cycle

The steps to disable the memory write machine cycle are given in table.

| S. No | T state | Operation |
|---|---|---|
| 1 | $T_1$ | The microprocessor places the higher order 8-bits of the memory address on A15 – A8 address bus and the lower order 8-bits of the memory address on AD7 – AD0 address / data bus. |
| 2 | | The microprocessor makes the ALE signal HIGH and at the middle of T1 state, ALE signal goes LOW. |
| 3 | | The status signals are changed as IO/$\overline{M}$ = 0, S1 =0 and S0 = 1. These status signals do not change throughout the memory write machine cycle. |
| 4 | $T_2$ | The microprocessor makes the $\overline{WR}$ line LOW to enable memory write. |
| 5 | | The contents of the specified register are placed on the address / data bus. |
| 6 | $T_3$ | The data placed on the address / data bus is transferred to the specified memory location. |
| 7 | | The microprocessor makes the $\overline{WR}$ line HIGH to disable the memory write operation. |

## UNIT-4 Programming (with respect to 8085 microprocessor)

## Computer Languages

Over the years, computer languages have been evolved from Low-Level to High-Level Languages. In the earliest days of computers, only Binary Language was used to write programs. The computer languages are classified as follows:

**Machine Language (low level language)**

Low-Level language is the only language which can be understood by the computer. Low-level language is also known as Machine Language. The machine language contains only two symbols 1 & 0. All the instructions of machine language are written in the form of binary numbers 1's & 0's. A computer can directly understand the machine language.

**Assembly Language (middle level language)**

Middle-level language is a computer language in which the instructions are created using symbols such as letters, digits and special characters. Assembly language is an example of middle-level language. In assembly language, we use predefined words called mnemonics. Binary code instructions in low-level language are replaced with mnemonics and operands in middle-level language. But the computer cannot understand mnemonics, so we use a translator called Assembler to translate mnemonics into machine language.

**Assembler** is a translator which takes assembly code as input and produces machine code as output. That means, the computer cannot understand middle-level language, so it needs to be translated into a low-level language to make it understandable by the computer. Assembler is used to translate middle-level language into low-level language.

**High Level Language**

High-level language is a computer language which can be understood by the users. The high-level language is very similar to human languages and has a set of grammar rules that are used to make instructions more easily. Every high-level language has a set of predefined words known as Keywords and a set of rules known as Syntax to create instructions. The high-level language is easier to understand for the users but the computer cannot understand it. High-level language needs to be converted into the low-level language to make it understandable by the computer. We use Compiler or interpreter to convert high-level language to low-level language.

Languages like FORTRAN,C, C++, JAVA, Python, etc., are examples of high-level languages. All these programming languages use human-understandable language like English to write program instructions. These instructions are converted to low-level language by the compiler or interperter so that it can be understood by the computer.

**MNEMONIC:** English word MNEMONIC means "A device such as a pattern of letters, ideas, or associations that assists in remembering something." So, it's usually used by assembly language programmers to remember the "OPERATIONS" a machine can do, like "ADD" and "MUL" and "MOV" etc. This is assembler specific.

**MACHINE CODE:** It is the sequence of numbers that flip the switches in the computer on and off (means 1 and 0) to perform a certain job of work - such as addition of numbers, branching, multiplication, etc etc. This is purely machine specific and well documented by the implementers of the processor.

## Instruction formats: 8085

An instruction (instruction format) is a command to the microprocessor to perform a given task on a particular data. Each instruction (instruction format) is of two parts. One is to be performed, called the operation code or opcode and the second one is the data to be operated on, called the operand. Operands or data can be specified in different ways. It may include an 8-bit or 16-bit data, an internal register. a memory location, or it or 16-bit address. In some instructions, the operand is implicit.

•The format of a typical instruction is composed of two parts: an operation code or op-code and an operand.

•Every instruction needs an op-code to specify what the operation of the instruction is and then an operand that gives the appropriate data needed for that particular operation code.

•According to the word or byte size the 8085 instructions are classified into three types. They are

(a) One byte (single) instructions.

(b)Two byte instructions.

(c) Three byte instructions.

**One–byte instructions:** An instruction with only opcode and do not require any data or address is called a one byte instruction.

Ex: 1. MOV C, A Hex code = 4FH (one byte)

2. ADD B Hex code = 80H (one byte)

3. CMA Hex code = 2FH (one byte)

**Two–byte instructions:** A two byte instruction is one which contains an 8-bit op-code and 8-bit operand (Data).

Ex: 1. MVI A, 09 Hex code = 3E, 09 (two bytes)

2. ADD B, 07 Hex code = 80, 07 (two bytes)

3. SUB A, 05 Hex code = 97, 05 (two bytes)

**Three–byte instructions:** A three byte instruction contains an opcode plus a 16 – bit address.

Ex: 1.LXI H, 8509 Hex code = 21, 09, 85 (Three bytes)

2 .LDA 8509 Hex code = 3A, 09, 85 (Three bytes)

3. JMP 9567 Hex code = C3, 67, 95 (Three bytes)

4. STA 3525 Hex code = 32, 35, 25 (Three bytes)

## Instruction Set:

The various techniques to specify data for instructions are:

1. 8-bit or 16-bit data may be directly given in the instruction itself.
2. The address of the memory location, I/O port or I/O device, where data resides, may be given in the instruction itself.
3. In some instructions, only one register is specified. The content of the specified register is one of the operands.
4. Some instructions specify two registers. The contents of the registers are the required data.
5. In some instructions, data is implied. The most instructions of this type operate on the content of the accumulator.

Due to different ways of specifying data for instructions, the machine codes of all instructions are not of the same length. It may 1-byte, 2-byte or 3-byte instruction.

## Addressing Modes:

Each instruction requires some data on which it has to operate. There are different techniques to specify data for instructions. These techniques are called addressing modes. Intel 8085 uses the following addressing modes:

**Direct Addressing**

In this addressing mode, the address of the operand (data) is given in the instruction itself.

Example

STA 2400H: It stores the content of the accumulator in the memory location 2400H.

In this instruction, 2400H is the memory address where data is to be stored. It is given in the instruction itself. The 2nd and 3rd bytes of the instruction specify the address of the memory location. Here, it is understood that the source of the data is accumulator.

**Register Addressing**

In register addressing mode, the operand is in one of the general purpose registers. The opcode specifies the address of the register(s) in addition to the operation to be performed.

Example:

MOV A, B: Move the content of B register to register A.

**Register Indirect Addressing**

In Register Indirect mode of addressing, the address of the operand is specified by a register pair.

Example

LXI H, 2500 H - Load H-L pair with 2500H.

MOV A, M - Move the content of the memory location, whose address is in H-L pair (i.e. 2500 H) to the accumulator.

HLT - Halt.

In the above program the instruction MOV A, M is an example of register indirect addressing. For this instruction, the operand is in the memory. The address of the memory is not directly given in the instruction. The address of the memory resides in H-L pair and this has already been specified by an earlier instruction in the program, i.e. LXI H, 2500 H.

**Immediate Addressing**

In this addressing mode, the operand is specified within the instruction itself.

Example

LXI H, 2500 is an example of immediate addressing. 2500 is 16-bit data which is given in the instruction itself. It is to be loaded into H-L pair.

**Implicit Addressing**

There are certain instructions which operate on the content of the accumulator. Such instructions do not require the address of the operand.

**Example**

CMA, RAL, RAR, etc.

## Symbols and Abbreviations

The symbol and abbreviations which have been used while explaining Intel 8085 instructions are as follows:

| Symbol/Abbreviations | Meaning |
|---|---|
| Addr | 16-bit address of the memory location. |
| Data | 8-bit data |
| data 16 | 16-bit data |
| r, r1, r2 | One of the registers A, B, C, D, E, H or L |
| A, B, C, D, H, L | 8-bit register |
| A | Accumulator |
| H-L | Register pair H-L |
| B-C | Register pair B-C |
| D-E | Register pair D-E |
| PSW | Program Status Word |
| M | Memory whose address is in H-L pair |
| H | Appearing at the end of the group of digits specifies hexadecimal, e.g. 2500H |
| Rp | One of the register pairs. |
| Rh | The high order register of a register pair |
| Rl | The low order register of a register pair |
| PC | 16 bit program counter, PCH is high order 8 bits and PCL low order 8 bits of register PC. |
| CS | Carry Status |
| [] | The contents of the register identified within bracket |
| [ [] ] | The content of the memory location whose address is in the register pair identified within brackets |
| ^ | AND operation |
| ∨ | OR operation |
| ⊕ or ∀ | Exclusive OR |

| | |
|---|---|
| ← | Move data in the direction of arrow |
| ⇔ | Exchange contents |

## Intel 8085 Instructions

An **instruction** of a computer is a command given to the computer to perform a specified operation on given data. In microprocessor, the **instruction** set is the collection of the instructions that the microprocessor is designed to execute.

The programmer writes a program in assembly language using these instructions. These instructions have been classified into the following groups:

### Data Transfer Group

Instructions which are used to transfer the data from a register to another register from memory to register or register to memory come under this group.

| Instruction Set | Explanation | States | Flags | Addre-ssing | Machine Cycles | Example |
|---|---|---|---|---|---|---|
| MOV $r_1$, $r_2$ [r1] ← [r2] | Move the content of the one register to another | 4 | none | Register | 1 | MOV A, B |
| MOV r, M [r]←[[H-L]] | Move the content of memory to register | 7 | none | Register Indirect | 2 | MOV B, M |
| MOV M, r [[H-L]]←[r] | Move the content of register to memory | 7 | none | Register Indirect | 2 | MOV M, C |
| MVI r, data [r] ←data | Move immediate data to register | 7 | None | Immediate Register | 3 | MVI M, 08 |
| LXI rp, data 16 [rp] ←data 16 bits, [rh] ←8 MSBs, [rl] ←8 LSBs of data | Load Register pair immediate | 10 | None | Immediate | 3 | LXI H, 2500H |
| LDA addr [A] ←[addr] | Load Accumulator direct | 13 | None | Direct | 4 | LDA 2400 H |
| STA Addr | Store accumulator | 13 | None | Direct | 4 | STA |

| | | | | | | |
|---|---|---|---|---|---|---|
| [addr] ←[A] | direct | | | | | 2000H |
| LHLD addr [L] ←[addr], [H] ← [addr + 1 ] | Load H-L pair direct | 16 | None | Direct | 5 | LHLD 2500H |
| SHLD addr [addr] ←[L], [addr +1] ← [H] | Store H-L pair direct | 16 | None | Direct | 5 | SHLD 2500 H |
| LDAX rp [A] ←[[rp]] | Load accumulator indirect | 7 | None | Register Indirect | 2 | LDAX B |
| STAX rp [[rp]] ←[A] | Store accumulator indirect | 7 | None | Register Indirect | 2 | STAX D |
| XCHG [H-L] ↔[D-E] | Change the contents of H-L with D-E pair | 4 | None | Register | 1 | |

**Arithmetic Group**

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

| Instruction Set | Explanation | States | Flags | Addre-ssing | Machine Cycles | Example |
|---|---|---|---|---|---|---|
| ADD r [A] ←[A]+[r] | Add register to accumulator | 4 | All | Register | 1 | ADD K |
| ADD M [A] ← [A] + [[H-L]] | Add memory to accumulator | 7 | All | Register indirect | 2 | ADD K |
| ACC r [A] ← [A] + [r] + [CS] | Add register with carry to accumulator | 4 | All | Register | 1 | ACC K |
| ADC M [A] ← [A] + | Add memory with carry to accumulator | 7 | All | Register indirect | 2 | ADC K |

| | | | | | | |
|---|---|---|---|---|---|---|
| [[H-L]] [CS] | | | | | | |
| ADI data [A] ← [A] + data | Add immediate data to accumulator | 7 | All | Immediate | 2 | ADI 55K |
| ACI data [A] ← [A] + data + [CS] | Add with carry immediate data to accumulator | 7 | All | Immediate | 2 | ACI 55K |
| DAD rp [H-L] ←[H-L] + [rp] | Add register paid to H-L pair | 10 | CS | Register | 3 | DAD K |
| SUB r [A] ←[A]-[r] | Subtract register from accumulator | 4 | All | Register | 1 | SUB K |
| SUB M [A] ← [A] - [[H-L]] | Subtract memory from accumulator | 7 | ALL | Register indirect | 2 | SUB K |
| SBB r [A] ←[A]-[H-L]] - [CS] | Subtract memory from accumulator with borrow | 7 | All | Register indirect | 2 | SBB K |
| SUI data [A] ←[A]-data | Subtract immediate data from accumulator | 7 | All | Immediate | 2 | SUI 55K |
| SBI data [A] ←[A]-data-[CS] | Subtract immediate data from accumulator with borrow | 7 | All | Immediate | 2 | XCHG |
| INR r [r] ←[r]+1 | Increment register content | 4 | All except carry flag | Register | 1 | INR K |
| INR M [[H-L]] ←[[H-L]]+1 | Increment memory content | 10 | All except carry flag | Register indirect | 3 | INR K |
| DCR r [r] ←[r] -1 | Decrement register content | 4 | All except carry flag | Register | 1 | DCR K |

| | | | | | | |
|---|---|---|---|---|---|---|
| DCR M [[H-L]] ← [[H-L]]-1 | Decrement memory content | 10 | All except carry flag | Register indirect | 3 | DCR K |
| INX rp [rp] ←[rp]+1 | Increment memory content | 6 | None | Register | 1 | INX K |
| DCX rp [rp] ←[rp]-1 | Decrement register pair | 6 | None | Register | 1 | DCX K |
| DAA | Decimal adjust accumulator | 4 | | | 1 | DAA |

**Logical Group**

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

| Instruction Set | Explanation | States | Flags | Addressing | Machine Cycles |
|---|---|---|---|---|---|
| ANA r [A] ←[A]∧[r] | AND register with accumulator | 4 | All | Register | 1 |
| ANA M [A] ←[A]∧[[H-]] | AND memory with accumulator | 4 | All | Register indirect | 2 |
| ANI data [A] ← [A] ∧ [data] | AND immediate data with accumulator | 7 | All | Immediate | 2 |
| ORA r [A] ←[A]∨[r] | OR-register with accumulator | 4 | All | Register | 1 |
| ORA M [A] ←[A]∨[[H-L]] | OR-memory with accumulator | 7 | All | Register indirect | 2 |
| ORI data [A] ← [A] ∨ [data] | OR -immediate data with accumulator | 7 | All | Immediate | 2 |
| XRA r [A] ← [A]∀[r] | XOR register with | 4 | All | Register | 1 |

| | | | | | |
|---|---|---|---|---|---|
| | accumulator | | | | |
| XRA M [A] ← [A] ∀ [[H-L]] | XOR memory with accumulator | 7 | All | Register indirect | 2 |
| XRI data [A] ←[A] ∀ [data] | XOR immediate data with accumulator | 7 | All | Immediate | 2 |
| CMA [A] ←[A] | Complement the accumulator | 4 | None | Implicit | 1 |
| CMC<br>[CS] ←[CS] | Complement the carry status | 4 | CS | | 1 |
| STC<br>[CS] ← 1 | Set carry status | 4 | CS | | 1 |
| CMP r<br>[A]-[r] | Compare register with accumulator | 4 | All | Register | 1 |
| CMP M<br>[A] - [[H-L]] | Compare memory with accumulator | 7 | All | Register indirect | 2 |
| CPI data<br>[A] - data | Compare immediate data with accumulator | 7 | All | Immediate | 2 |
| RLC<br>$[A_{n+1}] \leftarrow [A^n]$, $[A^0] \leftarrow [A^7]$, [CS] $\leftarrow [A^7]$ | Rotate accumulator left | 4 | Cs | Implicit | 1 |
| RRC<br>$[A^7] \leftarrow [A^0]$, [CS] $\leftarrow [A^0]$, $[A^n] \leftarrow [A^{n+1}]$ | Rotate accumulator right | | CS | Implicit | 1 |
| RAL<br>$[A^{n+1}] \leftarrow [A^n]$, [CS] $\leftarrow [A^7]$, $[A^0] \leftarrow$ [CS] | Rotate accumulator left through carry | | CS | Implicit | 1 |
| RAR<br>$[A^n] \leftarrow [A^{n+1}]$, [CS] | Rotate accumulator right through carry | | CS | Implicit | 1 |

| | | | | | |
|---|---|---|---|---|---|
| ←[$A^0$], [$A^7$] ←[CS] | | | | | |

**Branch Control Group**

This group contains the instructions for conditional and unconditional jump, subroutine call and return, and restart.

**Unconditional Jump**

| Instruction Set | Explanation | States | Flags | Addressing | Machine Cycles |
|---|---|---|---|---|---|
| JMP addr(label)<br>[PC] ← Label | Unconditional jump: jump to the instruction specified by the address | 10 | None | Immediate | 3 |

**Conditional Jump**

| Instruction Set | Explanation | States | Machine Cycles |
|---|---|---|---|
| Jump addr (label)<br>[PC] ← Label | Conditional jump: jump to the instruction specified by the address if the specified condition is fulfilled | 10, if true and<br>7, if not true | 3, if true and 2, if not true |

| Instruction Set | Explanation | Status | States | Flags | Addressing | Machine Cycles |
|---|---|---|---|---|---|---|
| JZ addr (label) [PC] ← address (label) | Jump, if the result is zero | Jump if Z=1 | 7/10 | None | Immediate | 2/3 |
| JNZ addr (label) [PC] ← address (label) | Jump if the result is not zero | Jump if Z=0 | 7/10 | None | Immediate | 2/3 |
| JC addr (label) [PC] ← address (label) | Jump if there is a carry | Jump if CS =1 | 7/10 | None | Immediate | 2/3 |
| JNC addr (label) [PC] ← address (label) | Jump if there is no carry | Jump if CS =0 | 7/10 | None | Immediate | 2/3 |

| | | | | | | |
|---|---|---|---|---|---|---|
| JP addr (label) [PC] ← address (label) | Jump if result is plus | Jump if S=0 | 7/10 | None | Immediate | 2/3 |
| JM addr (label) [PC] ← address (label) | Jump if result is minus | Jump if S=1 | 7/10 | None | Immediate | 2/3 |
| JPE addr (label) [PC] ← address (label) | Jump if even parity | The parity status P =1 | 7/10 | None | Immediate | 2/3 |
| JPO addr (label) [PC] ← address (label) | Jump if odd parity | The parity status P =0 | 7/10 | None | Immediate | 2/3 |

**Unconditional CALL**

| Instruction Set | Explanation | States | Flags | Addressing | Machine Cycles |
|---|---|---|---|---|---|
| CALL addr (label) [SP]-1] ← [PCH] ,[[SP-2] ← [PCL], [SP] ← [SP]-2, [PC] ← addr(label) | Unconditional CALL: Call the subroutine identified by the address | 18 | None | Immediate /register | 5 |

**Conditional CALL**

| Instruction Set | Explanation | States | Machine Cycles |
|---|---|---|---|
| CALL addr (label) [SP]-1] ← [PCH] , [[SP-2] ← [PCL], [PC] ← addr (label), [SP] ← [SP]-2 | Unconditional CALL: Call the subroutine identified by the address if the specified condition is fulfilled | 18, if true and 9, if not true | 5, if true and 2, if not true |

| Instruction Set | Explanation | Status | States | Flags | Addressing | Machine Cycles |
|---|---|---|---|---|---|---|
| CC addr(label) | Call subroutine if carry status CS=1 | CS =1 | 9/18 | None | Immediate /register | 2/5 |

| | | | | | | |
|---|---|---|---|---|---|---|
| CNC addr (label) | Call subroutine if carry status CS=0 | CS =0 | 9/18 | None | Immediate /register | 2/5 |
| CZ addr (label) | Call Subroutine if the result is zero | Zero status Z=1 | 9/18 | None | Immediate /register | 2/5 |
| CNZ addr (label) | Call Subroutine if the result is not zero | Zero status Z=0 | 9/18 | None | Immediate /register | 2/5 |
| CP addr (label) | Call Subroutine if the result is plus | Sign status S=0 | 9/18 | None | Immediate /register | 2/5 |
| CM addr (label) | Call Subroutine if the result is minus | Sign status S= 1 | 9/18 | None | Immediate /register | 2/5 |
| CPE addr(label) | Call subroutine if even parity | Parity Status P=1 | 9/18 | None | Immediate /register | 2/5 |
| CPO addr(label) | Call subroutine if odd parity | Parity Status P= 0 | 9/18 | None | Immediate /register | 2/5 |

**Unconditional Return**

| Instruction Set | Explanation | States | Flags | Addressing | Machine Cycles |
|---|---|---|---|---|---|
| RET<br>[PCL] ← [[SP]], [PCH] ← [[SP] + 1], [SP] ← [SP] + 2 | Unconditional RET: Return from subroutine | 10 | None | Indirect | 3 |

**Conditional Return**

| Instruction Set | Explanation | States | Machine Cycles |
|---|---|---|---|
| RET<br>[PCL] ← [[SP]],<br>[PCH] ← [[SP] + 1],<br>[SP] ← [SP] + 2 | Conditional RET: Return from subroutine | 12, if true and 6, if not true | 3, if true and 1, if not true |

| Instruction Set | Explanation | Status | States | Flags | Addressing | Machine Cycles |
|---|---|---|---|---|---|---|
| RC | Return from subroutine if carry status is zero. | CS =1 | 6/12 | None | Register indirect | 1/3 |
| RNC | Return from subroutine if carry status is not zero. | CS = 0 | 6/12 | None | Register indirect | 1/3 |
| RZ | Return from subroutine if result is zero. | Zero status Z=1 | 6/12 | None | Register indirect | 1/3 |
| RNZ | Return from subroutine if result is not zero. | Zero status Z= 0 | 6/12 | None | Register indirect | 1/3 |
| RP | Return from subroutine if result is not plus. | Sign Status S= 0 | 6/12 | None | Register indirect | 1/3 |
| RM | Return from subroutine if result is not minus. | Sign Status S= 0 | 6/12 | None | Register indirect | 1/3 |
| RPE | Return from subroutine if even parity. | Parity Status P= 1 | 6/12 | None | Register indirect | 1/3 |
| RPO | Return from subroutine if odd parity. | Parity Status P= 1 | 6/12 | None | Register indirect | 1/3 |

**Restart**

| Instruction Set | Explanation | States | Flags | Addressing | Machine Cycles |
|---|---|---|---|---|---|
| RST [[SP]-1] ← [PCH], [[SP]-2] ← [PCL], [SP] ← [SP] - 2, [PC] ← 8 times n | Restart is a one word CALL instruction. | 12 | None | Register Indirect | 3 |

The restart instructions and locations are as follows:

| Instruction | Opcode | Restart Locations |
|---|---|---|
| RST 0 | C7 | 0000 |
| RST 1 | CF | 0008 |
| RST 2 | D7 | 0010 |
| RST 3 | DF | 0018 |
| RST 4 | E7 | 0020 |
| RST 5 | EF | 0028 |
| RST 6 | F7 | 0030 |
| RST 7 | FF | 0038 |

PCHL

| Instruction Set | Explanation | States | Flags | Addressing | Machine Cycles |
|---|---|---|---|---|---|
| PCHL [PC] ← [H-L], [PCH] ←[H], **[PCL] ←[L]** | Jump address specified by H-L pair | 6 | None | Register | 1 |

**Stack, I/O and Machine Control Group**

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

| Instruction Set | Explanation | States | Flags | Addressing | Machine Cycles |
|---|---|---|---|---|---|
| IN port - address [A] ← [Port] | Input to accumulator from I/O port | 10 | None | Direct | 3 |

| | | | | | |
|---|---|---|---|---|---|
| OUT port-address<br>[Port] ← [A] | Output from accumulator to I/O port | 10 | None | Direct | 3 |
| PUSH rp<br>[[SP] - 1] ← [rh],<br>[[SP] - 2] ← [rh],<br>[SP] ← [SP] - 2 | Push the content of register pair to stack | 12 | None | Register(source)/register Indirect(destination) | 3 |
| PUSH PSW<br>[SP]-1] ← [A],<br>[[SP] -2] ← PSW,<br>[SP] ← [SP] - 2 | Push processor word | 12 | None | Register(source)/register Indirect(destination) | 3 |
| POP rp<br>[rl] ← [ [ SP ] ],<br>[rh] ← [[SP]+1],<br>[SP] ← [SP] + 2 | Pop the content of register pair, which was saved, from the stack | 10 | None | Register(source)/register Indirect(destination) | 3 |
| HLT | Halt | 5 | None | | 1 |
| XTHL<br>[L] ↔ [[SP]],<br>[H] ↔ [[SP] + 1] | Exchange top stack with H-L | 16 | None | Register indirect | 5 |
| SPHL<br>[H-L] → [SP] | Moves the contents of H-L pair to stack pointer | 6 | None | Register | 1 |
| EI | Enable Interrupts | 4 | None | | 1 |
| SIM | Set Interrupts Masks | 4 | None | | 1 |

| | | | | | |
|---|---|---|---|---|---|
| RIM | Read Interrupts Masks | 4 | None | | 1 |
| NOP | No Operation | 4 | None | | 1 |

## UNIT-5 Memories and I/O interfacing

### Memory mapped I/O and Isolated I/O

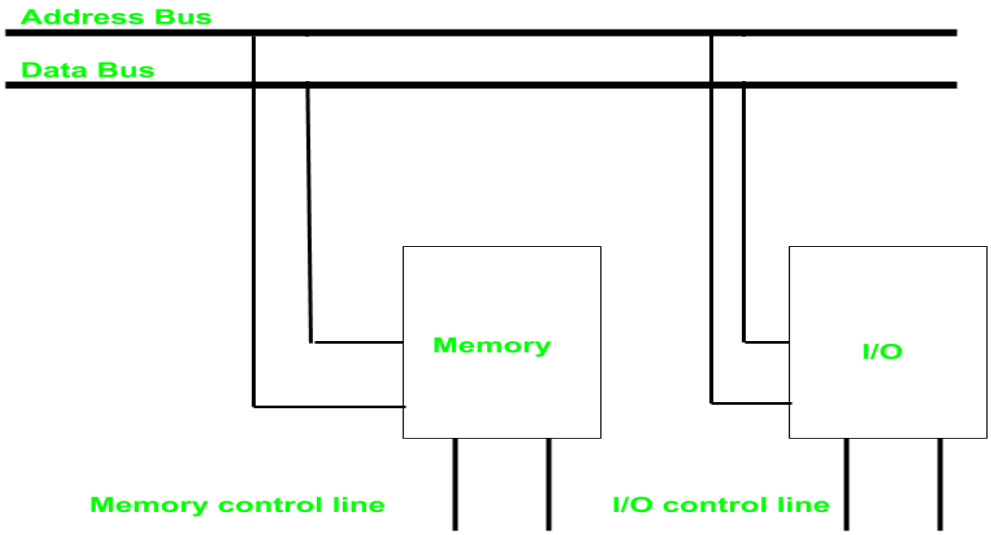
As a CPU needs to communicate with the various memory and input-output devices (I/O) as we know data between the processor and these devices flow with the help of the system bus. There are three ways in which system bus can be allotted to them:

1. Separate set of address, control and data bus to I/O and memory.
2. Have common bus (data and address) for I/O and memory but separate control lines.
3. Have common bus (data, address, and control) for I/O and memory.

In first case it is simple because both have different set of address space and instruction but require more buses.

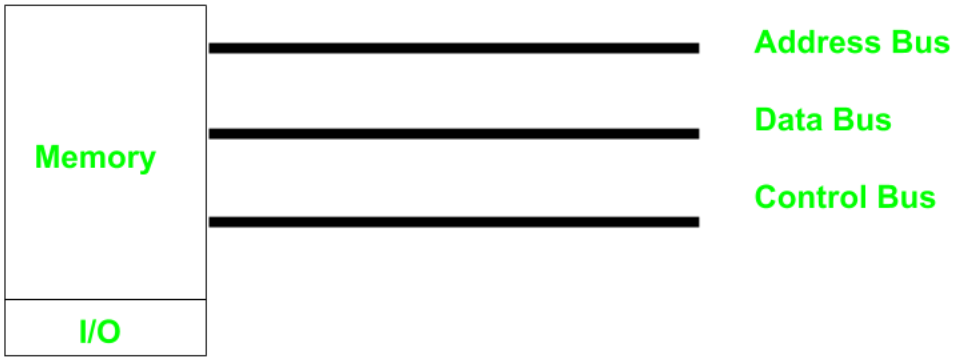
**Isolated I/O –**

Then we have Isolated I/O in which we Have common bus(data and address) for I/O and memory but separate read and write control lines for I/O. So when CPU decode instruction then if data is for I/O then it places the address on the address line and set I/O read or write control line on due to which data transfer occurs between CPU and I/O. As the address space of memory and I/O is isolated and the name is so. The address for I/O here is called ports. Here we have different read-write instruction for both I/O and memory.




**Memory Mapped I/O –**

In this case every bus in common due to which the same set of instructions work for memory and I/O. Hence we manipulate I/O same as memory and both have same address space, due to which addressing capability of memory become less because some part is occupied by the I/O.




**Differences between memory mapped I/O and isolated I/O –**

| Isolated I/O | Memory Mapped I/O |
|---|---|
| Memory and I/O have separate address space | Both have same address space |
| All address can be used by the memory | Due to addition of I/O addressable memory become less for memory |
| Separate instruction control read and write operation in I/O and Memory | Same instructions can control both I/O and Memory |
| In this I/O address are called ports. | Normal memory address are for both |
| More efficient due to separate buses | Lesser efficient |
| Larger in size due to more buses | Smaller in size |
| It is complex due to separate logic is used to control both. | Simpler logic is used as I/O is also treated as memory only. |

## UNIT-6 Interrupts

# Interrupts in 8085

Interrupts are the signals generated by the 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.

Interrupt are classified into following groups based on their parameter −

- **Vector interrupt** − In this type of interrupt, the interrupt address is known to the processor. **For example:** RST 7.5, RST6.5, RST5.5, and TRAP.
- **Non-Vector interrupt** − in this type of interrupt, the interrupt address is not known to the processor so; the interrupt address needs to be sent externally by the device to perform interrupts. **For example:** INTR.
- **Maskable interrupt** − in this type of interrupt, we can disable the interrupt by writing some instructions into the program. **For example:** RST7.5, RST6.5, and RST5.5.
- **Non-Maskable interrupt** − In this type of interrupt, we cannot disable the interrupt by writing some instructions into the program. **For example:** TRAP.
- **Software interrupt** − In this type of interrupt, the programmer has to add the instructions into the program to execute the interrupt. There are 8 software interrupts in 8085, i.e. RST0, RST1, RST2, RST3, RST4, RST5, RST6, and RST7.
- **Hardware interrupt** − There are 5 interrupt pins in 8085 used as hardware interrupts, i.e. TRAP, RST 7.5, RST6.5, RST5.5, INTA.

**Note** − INTA is not an interrupt; it is used by the microprocessor for sending acknowledgement. TRAP has the highest priority, then RST7.5 and so on.

**Interrupt Service Routine (ISR)**

A small program or a routine that when executed, services the corresponding interrupting source is called an ISR.

**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.

**RST 7.5**

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.

**Interrupts can be classified into various categories based on different parameters:**

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

**3. Vectored and Non-Vectored Interrupts –**

*Vectored Interrupts* are those which have fixed vector address (starting address of sub-routine) and after

executing these, program control is transferred to that address.

**Non-Vectored Interrupts** are those in which vector address is not predefined. The interrupting device gives the address of sub-routine for these interrupts. *INTR* is the only non-vectored interrupt in 8085 microprocessor.

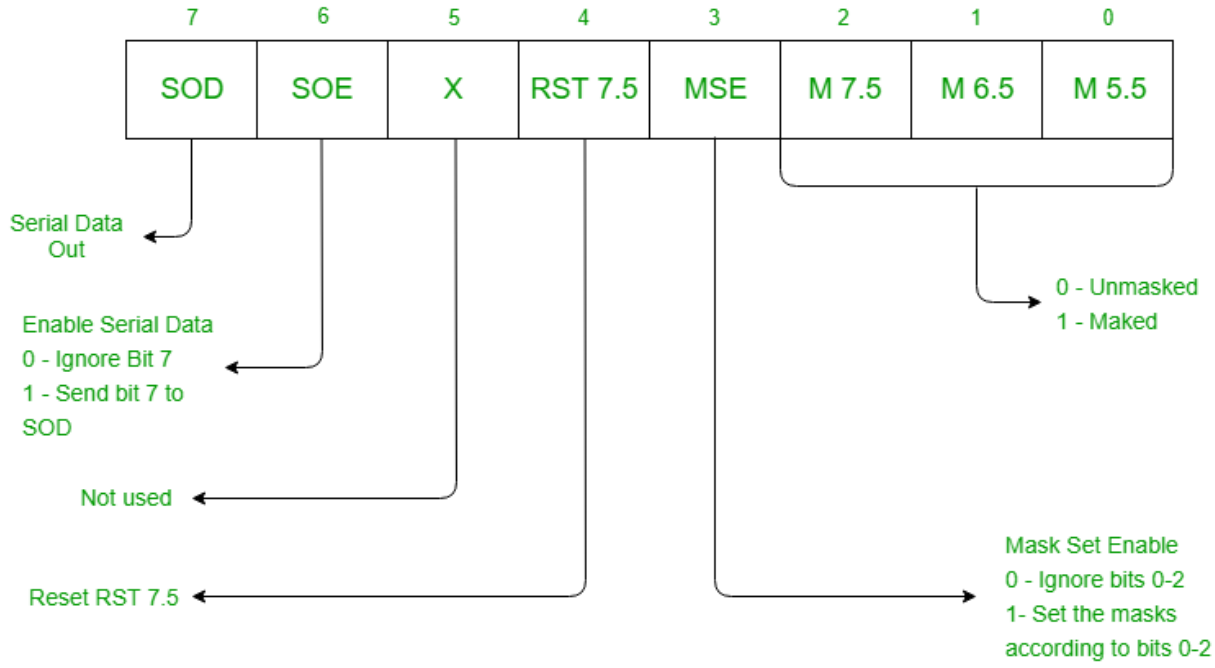
1. **Maskable and Non-Maskable Interrupts –**

   *Maskable Interrupts* are those which can be disabled or ignored by the microprocessor. These interrupts are either edge-triggered or level-triggered, so they can be disabled. *INTR, RST 7.5, RST 6.5, RST 5.5* are maskable interrupts in 8085 microprocessor.
2. Non-Maskable Interrupts are those which cannot be disabled or ignored by microprocessor. *TRAP* is a non-maskable interrupt. It consists of both level as well as edge triggering and is used in critical power failure conditions.

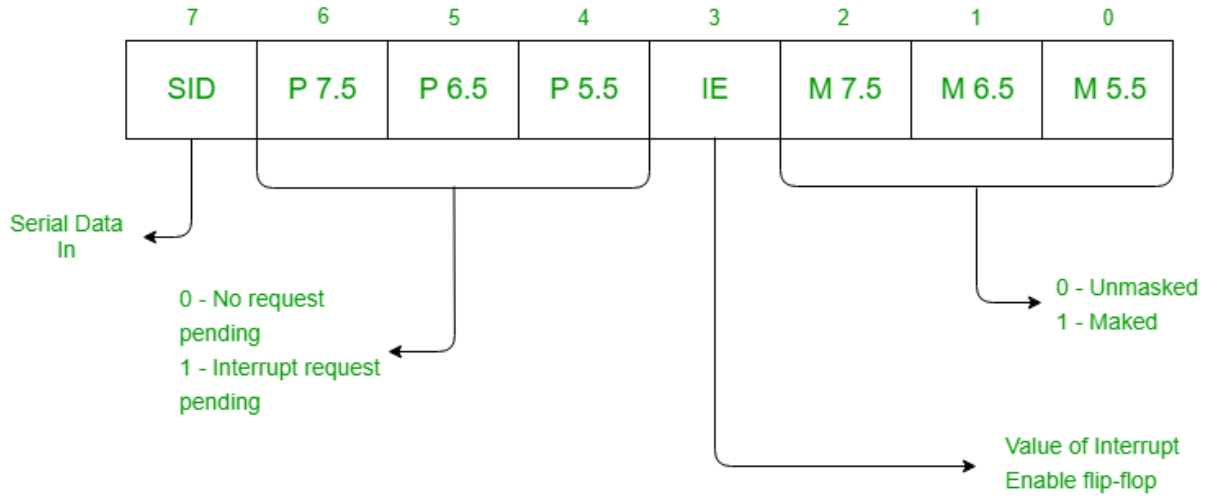
**Priority of Interrupts –**
When microprocessor receives multiple interrupt requests simultaneously, it will execute the interrupt service request (ISR) according to the priority of the interrupts.


**Instruction for Interrupts –**

1. **Enable Interrupt (EI) –** The interrupt enable flip-flop is set and all interrupts are enabled following the execution of next instruction followed by EI. No flags are affected. After a system reset, the interrupt enable flip-flop is reset, thus disabling the interrupts. This instruction is necessary to enable the interrupts again (except TRAP).
2. **Disable Interrupt (DI) –** This instruction is used to reset the value of enable flip-flop hence disabling all the interrupts. No flags are affected by this instruction.
3. **Set Interrupt Mask (SIM) –** It is used to implement the hardware interrupts (RST 7.5, RST 6.5, RST 5.5) by setting various bits to form masks or generate output data via the Serial Output Data (SOD) line. First the required value is loaded in accumulator then SIM will take the bit pattern from it.




**Read Interrupt Mask (RIM) –** This instruction is used to read the status of the hardware interrupts (RST 7.5, RST 6.5, RST 5.5) by loading into the A register a byte which defines the condition of the mask bits for the interrupts. It also reads the condition of SID (Serial Input Data) bit on the microprocessor.




**Level triggered interrupt** is an indication that a device needs attention. ... **Edge triggered interrupt** is an event notification. When some particular thing happens, the device generates an active **edge** on the **interrupt** line.

UNIT-7 Data Transfer Techniques

## Why is Data Transfer needed?

We can connect several I/O devices and memory peripherals to a microprocessor. However, since different technologies are involved, there will be differences in the speed of operation and of data transfer.

Usually, when memory is connected with the microprocessor, there isn't a stark difference in the processing speed since semiconductor memories are generally easily compatible with microprocessors. However, this might not always be the case.

But the problem often arises when external peripherals are connected as I/O devices. A slow I/O device won't be able to transfer data at a satisfactory rate whenever the microprocessor requests for data transfer. And usually, the peripheral devices do have slower transfer rates than the processor. Maybe the processor might send two units of information per second, but the external device might only accept 1 unit of information per second. Or conversely, perhaps the external device expects quicker transfers, but our microprocessor might be sending information a bit slower.

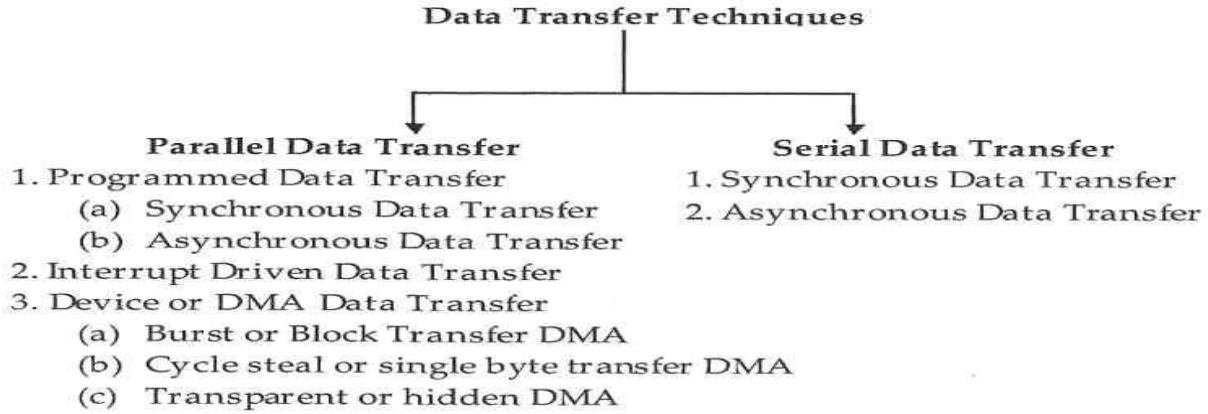
This might lead to severe data losses, or the devices might get damaged, or there might even be chances for the system to slow down all over, thus affecting the overall efficiency of the system.

To avoid this problem, a number of data transfer techniques have been devised.

Since constant communication is required between the external device and the processor, these data transfer techniques play a crucial role in the efficient functioning of the system with the externally connected devices.

**Classification of Data Transfer Schemes**

We can broadly classify the data transfer schemes into two modes – Serial Data Transfer and Parallel Data Transfer.




**Classification of the data transfer techniques in 8085**

Our device, the Intel 8085 Microprocessor, is a parallel device. Thus, it transfers 8 bits of information simultaneously over 8 data lines in the parallel I/O mode. But sometimes, there are instances where using this data transfer technique might be just theoretical or impossible to apply. Using parallel data transfer can be expensive if communication is to take place over vast distances. Also, if the device, on the other hand, follows the protocol of serial communication only, then it is impossible to use parallel mode.

**Difference between Parallel Data Transfer and Serial Data Transfer**

As discussed earlier, we can broadly classify data transfer schemes into parallel data transfer techniques and serial data transfer techniques. Let us understand the differences between them.

| SERIAL DATA TRANSFER TECHNIQUES | PARALLEL DATA TRANSFER TECHNIQUES |
|---|---|
| Under this scheme, the data is transferred one bit at a time. | Under this scheme, the data is transferred several bits at the same time. |
| It is a slower mode of data transfer. | Data is transferred much quicker. |
| Serial data transfer is preferred when data is to be sent over a long distance and the cost of cables would be too expensive. | Parallel data transfer is the preferred technique for short-distance communication. |
| For this mode, the transmitter first performs parallel – to – serial conversion and the serial – to – parallel conversion at the receiver. | No such conversions are required at both the transmission and reception endpoints. |
| This mode requires a single line to transfer information. | This mode requires multiple lines for data transfer. |
| Noise and errors are much lesser. | As multiple bits are transmitted at the same time, there is scope for more error and noise. |
| Cables used for serial communication are much thinner, longer, and very economical. | Here, the cables are much shorter, and thicker compared to the Serial communication cables. |

**Parallel Data Transfer Techniques**

We know that under the parallel data transfer scheme, multiple data bits can be transmitted at the same time. Thus, for the Intel 8085, 8 bits of data are sent all together using eight parallel lines. Let us go through the different types of parallel data transfer schemes. We have:

1. Programmed I/O Data Transfer
2. Interrupt Driven I/O Data Transfer
3. Device or Direct Memory Access (DMA) Data Transfer

Let us study each of these transfer schemes in detail.

**Programmed I/O Data Transfer**

This is a straightforward scheme under parallel data transfer mechanisms. This mode is generally preferred for simple, small microprocessor systems where speed is critical.

This method can work under the synchronous and asynchronous mode, depending on the speed and architecture of the I/O devices.

We also prefer this method when there is a small amount of information to be exchanged between the microprocessor and other devices that are placed near to the microprocessor. Example: Computer, printer, etc.

Using the IN and OUT instructions, data transfer is carried out between the microprocessor and I/O devices.

The processor reads the data from an input port or input device using the IN command. The processor sends data out from the CPU to the output port or the output device using the OUT instruction.

Thus, as the speeds of the processor and external device match, the data transferring process is carried out using the IN and OUT instructions.

**Synchronous Data Transfer Method**

The word 'Synchronous'means 'taking place at the same time.'

Thus, to establish communication between our processor and the device, we need to set a common clock pulse. This common pulse synchronizes the peripheral device with the 8085 microprocessor.

This method is used when the speed of the microprocessor, Intel 8085, in this case, and the external peripheral device match with each other.

If the device is ready to send data, it can indicate via the READY pin of 8085. Once the speeds match, the data transfer immediately begins, once a signal is issued by the microprocessor to begin transferring. The microprocessor need not wait for an extended period because of the matching speeds.

This technique of data transfer is seldom used to communicate with I/O devices though. Because I/O devices compatible with the microprocessor's speed are usually not found.

Hence, this method of data transfer is most commonly employed for communicating with compatible memory devices.

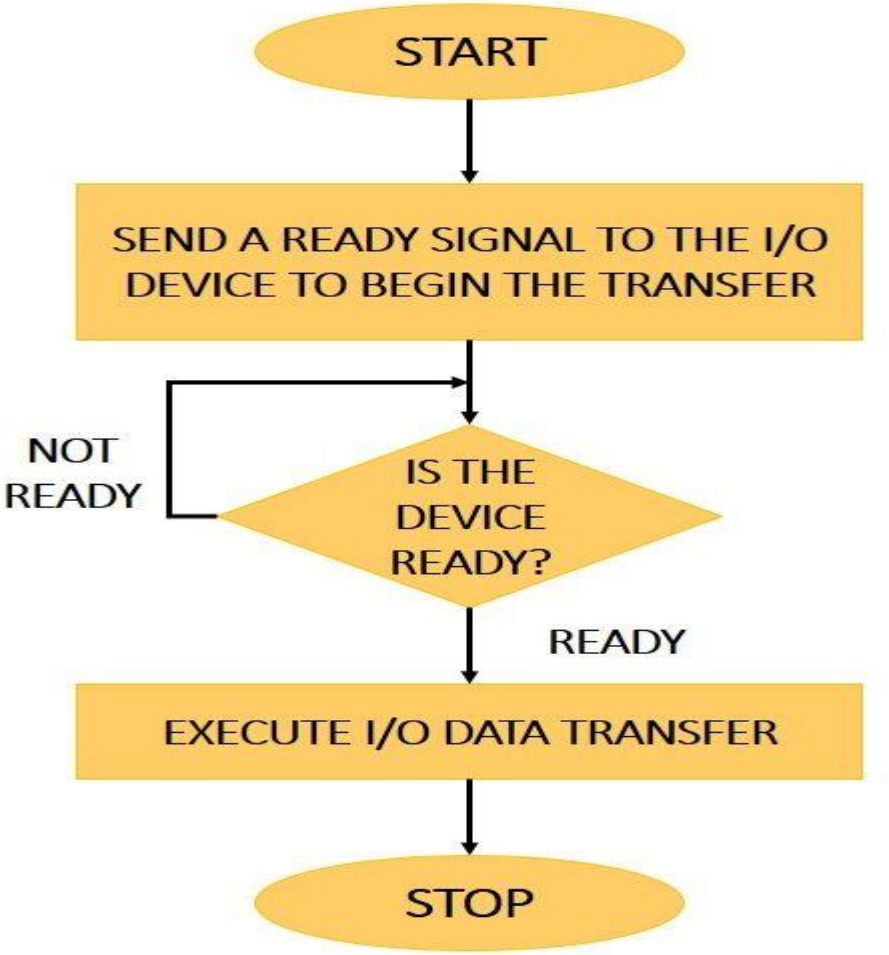
**Asynchronous Data Transfer Method**

When the speed of the I/O device is slower than that of the microprocessor, we prefer the Asynchronous Data Transfer Method. As the speeds of both the devices differ, the I/O device's internal timing is entirely independent of the microprocessor.

Thus, they are termed to be 'asynchronous'from each other. The term asynchronous means 'at irregular intervals.'

We implement the Asynchronous Data Transfer Method using the handshaking policy. But how is this method applied? And what happens next?

Under the handshaking method, the microprocessor and I/O device exchange a few signals before beginning the transfer of data between them.

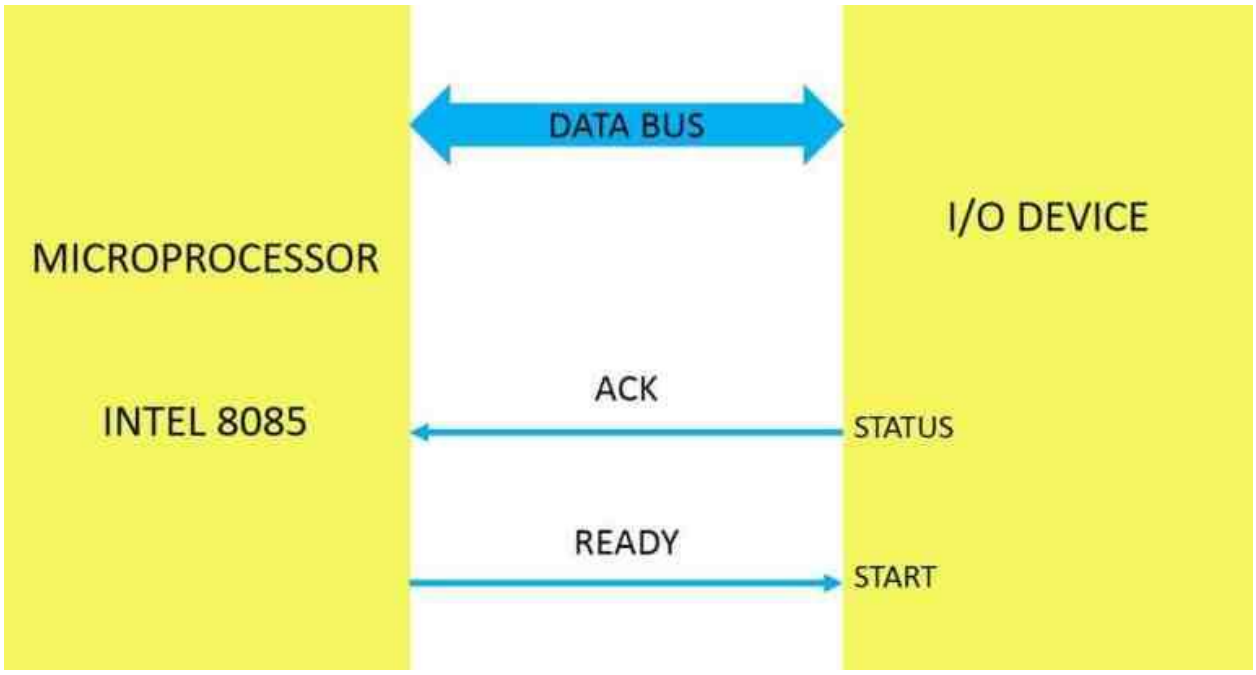



**Handshaking Protocol**

Let us understand the flowchart of the handshake protocol.

- First, the microprocessor raises a ready signal and sends it to the I/O device, which is connected to it.
- The status of the I/O device is continually checked to see it is prepared for data transferring.
- If the device is not ready, it is checked again and again until the device signals it is ready.
- Once the device is ready, the data transfer begins from the microprocessor to the I/O device.

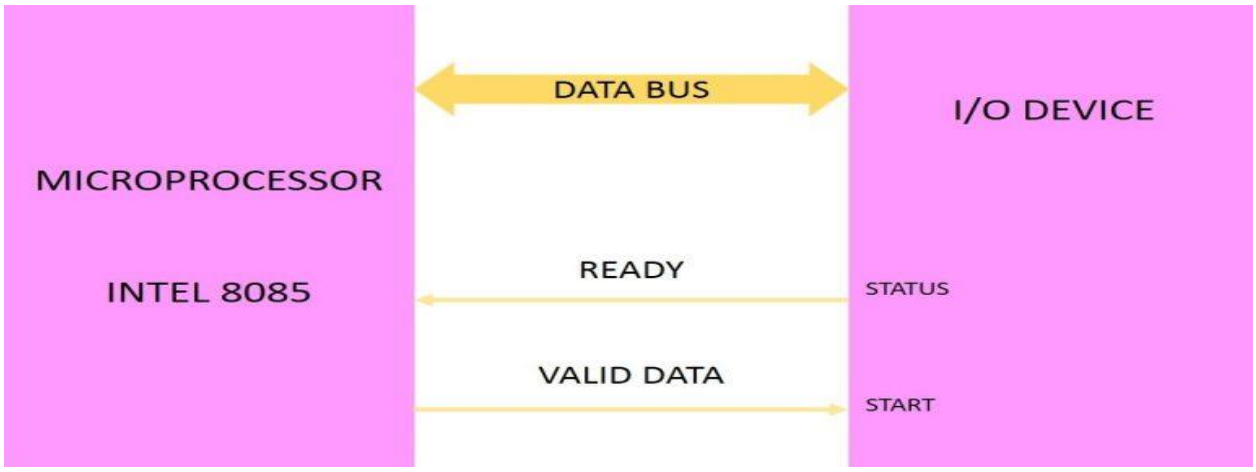
But how does the I/O device inform the microprocessor that it is ready for data transfer? This is done by the 'ACK' (Acknowledge) Signal. Also known as the handshake signal.




The microprocessor first checks the readiness of the I/O device continually. Once the I/O device is ready to accept data from the microprocessor, it sends an ACK signal to the microprocessor. This indicates the microprocessor that the device is all set for the reception of data. Now the data transmission can finally take place.

What happens if the opposite is supposed to happen? What if the I/O device has to submit information to the microprocessor?

In such a situation, the I/O device issues the ready signal back to 8085, informing it that it is ready to send data to the processor. In response to this ready signal, a valid data signal is sent by the 8085 to the I/O device, and then the valid information is put on the common data bus for the data transfer.




We can conclude by saying that under the Programmed I/O Data Transfer method, the microprocessor is always busy checking the status of the slower I/O device continuously for sharing data. Thus, some amount of time is

wasted on the microprocessor's behalf. Additionally, this method of data transfer is used by I/O devices and slow external memory peripherals.

**Interrupt Driven I/O Data Transfer**

The Programmed I/O Data Transfer, unfortunately, wastes a lot of time. Thus, the Interrupt Driven I/O Data Transfer is a much better option. Here, no extra time of the microprocessor is wasted in waiting for a response/signal from the external device. Under this method, the I/O device informs the microprocessor about its readiness, only when it is ready. This is accomplished by interrupting the Intel 8085.


To begin with, the microprocessor initiates the transfer of data by sending a request to the external device to be 'get ready.'

Following this, the processor continues working normally, executing the original program rather than wasting its cycles by keeping a check on the status of the external I/O device.

Once the I/O device is all set to transfer information, it sends a control signal to the 8085 to inform its readiness. This control signal is called the Interrupt (INTR) signal.

Once the INTR signal is received by 8085, it responds to it by submitting an INTR acknowledgment signal back to the external I/O device.

When the I/O device gets the acknowledgment, both the devices are now prepared for the data transfer. The microprocessor completes the execution of the current instruction, then suspends the job. It saves all the contents and the status of the PC (Program Counter) into the stack memory and then proceeds with the subroutine program.

This subroutine program for the raised interrupt is called the Interrupt Service Subroutine (ISR) program.

This ISR first saves the status of our processor into the stack.

Once the data transfer is completed by executing the necessary instructions, the ISR restores the previous processor status and continues with the normal execution of the main program.

There may be different types of interrupt requesting configurations that might arise when the I/O devices are interfaced with the Intel 8085. They are:

1. Single Interrupt System
2. Multi Interrupt System

Under the Single Interrupt System, only one interrupt wire/line is available with the processor to which multiple I/O devices are to be connected to.

**Multiple Interrupt Syste**m

If the microprocessor has multiple interrupt terminals and to each interrupt terminal, one I/O device is to be connected, it is known as the multiple interrupt system.

Here, the total number of external devices to be linked with the interrupt wires/lines should be either equal to or lesser than the total number of interrupt terminals. Thus, one device is connected at a time to each level of the interrupt.

When a device interrupts the microprocessor, the processor immediately recognizes which device has raised the interrupt signal and identifies the location in the memory where the required subroutine is stored. This kind of interrupt scheme is referred to as a Vectored Interrupt.

**Device or Direct Memory Access (DMA) Data Transfer**

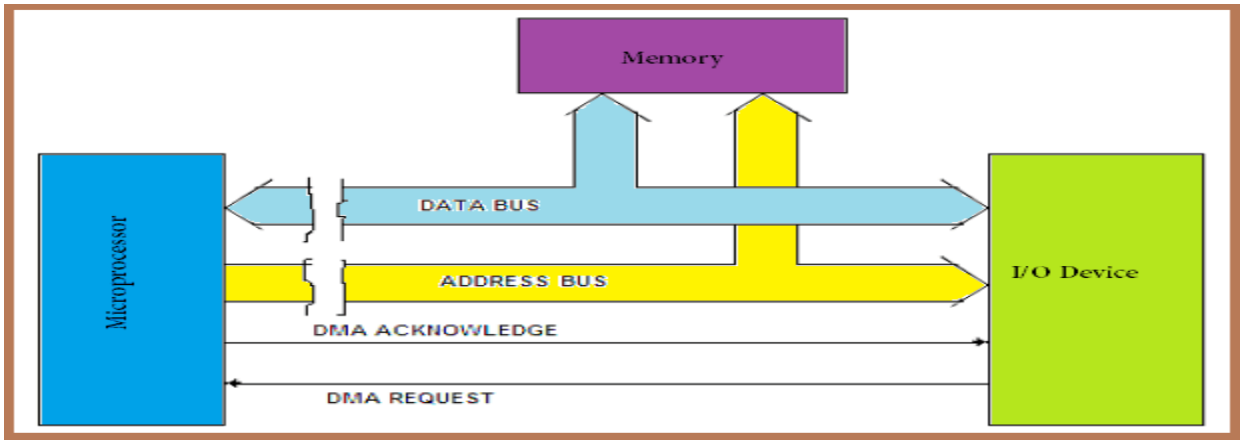
In the two schemes discussed earlier, the transfer of information between the I/O device and the mass storage device/memory is via the Accumulator register. But if there is a bulk amount of data to be transferred between the I/O device and the microprocessor, then these methods are unworthy of the investment being put into them, and even time – consuming.

To overcome such circumstances, the Direct Memory Access Data Transfer is put into use. This is an ideal method used for data transfer of a considerable amount between the microprocessor and the I/O device. It is also considered the fastest data transfer scheme.

But what makes it different from the previous two methods?

In the DMA scheme, the data is transferred between the I/O device (or DMA controller, a special IC) and the external memory directly without having any interference by the accumulator register. The processor gives up its control over the address bus as well as the data bus to the I/O device or DMA controller, thus aiding the exchange of information between the source and the destination directly.

We will now understand the working principle of the Direct Memory Access Data Transfer method.




**Direct Memory Access Data Transfer**

First of all, we need to inform the microprocessor before beginning the process. For this, an I/O device first sends a request to a DMA controller using the DMARQ signal, which in turn forwards it to the processor in the form of a HOLD signal.

Once such a request is received by the microprocessor, it ceases its control over the address bus and data bus. It then informs the DMA Controller of the situation by sending an acknowledgment signal using the HLDA command.

The controller then informs the external peripheral by sending a DMACK signal.

The DMA controller then gains control over the two buses and monitors the transfer of information between the two locations (Memory and I/O).

Once the entire data transfer between the I/O device and the external memory gets over, the DMA Controller withdraws its request for using the data and address bus by disabling the HOLD and DMACK signals, and thus the microprocessor gains control over them again.

There are three types of techniques under the Direct Memory Access Data Transfer:

1. Burst or block transfer DMA
2. Cycle steal or the single-byte transfer DMA
3. Transparent or hidden DMA

**Burst or Block Transfer DMA**

This is the fastest DMA mode of data transfer.

In the Burst mode, at least two or more bytes of data are transferred continuously i.e., the entire block of data is transferred in a straight sequence. Here, the microprocessor is disconnected from the main system during the data transfer, and thus, the processor is unable to execute any program on its own during the information transfer.

In fact, in this mode, the DMA controller acts as a Master.

If there are about N number of bytes to be transferred, then N number of machine cycles will be adopted into the working of the processor.

The DMA controller first sends a HOLD signal to the microprocessor to request access for the system's buses, and in turn, wait for the HLDA signal.

One the HLDA signal is received; the DMA controller gets access over the system bus and sends one byte of information.

Once a single byte is sent, the memory address is incremented, the counter is decremented, and then the next byte is sent.

Thus, following this technique, all data bytes are transferred between memory and I/O devices. Once all the information is sent, the DMA controller disables the HOLD signal.

It then enters into the slave mode.

This method is generally used for loading data files or important programs into the memory. However, it keeps the CPU inactive for relatively long periods.

**Cycle steal or Single-Byte transfer DMA**

In this mode, only a single byte is transferred at a time.

This is thus much slower than burst DMA.

In the cycle-steal DMA, The DMA controller sends a HOLD signal to the microprocessor.

It then waits for the HLDA signal in return.

Once the HLDA signal is received, it gets access over the system buses and executes one DMA cycle only.

After this transfer, the HOLD signal is disabled, and it enters into the slave mode.

The processor thus gets back its control over the address and data bus and continues executing the following machine cycle.

However, if the counter has not touched down to zero, and there is still data to be exchanged, then the DMA controller sends a HOLD signal again to the processor, and sends the next byte of the information block.

Thus, only one DMA cycle takes place between every two machine cycles of the processor, and the execution speed of the instructions in the microprocessor falls back a bit.

The DMA Controller obtains the access for the system buses by repeatedly issuing the requests using the Bus Request (BR) and Bus Grant (BG) signals until the entire data block has been transferred.

Though the information bytes are not transferred as fast as in the Burst mode, the only advantage of the cycle-steal mode is that the CPU does not remain idle for long durations of time, as in the burst DMA mode. This method is mainly used in controllers which are used in monitoring data in real-time.

**Transparent or Hidden DMA transfer**

The Hidden DMA Transfer method is considered the slowest among all the other DMA transfers.

Here, the microprocessor executes some of its states during which it floats the data bus and the address bus.

In these states, the microprocessor gets isolated from the main system bus.

In this isolation, the DMA controller transfers the information byte between the external peripherals and the memory. This, thus, becomes transparent to the microprocessor.

The instruction execution speed of the microprocessor does not get reduced. However, this DMA mode requires extra logic to sense the states in which the processor is floating the buses.

This mode is favored at the time when we do not want the CPU to stop executing its main program, as the data transfer is performed only when the CPU does not need the system bus.

Nevertheless, the hardware needed to check such states can be pretty complex and a little too expensive as well.

**Serial Data Transfer Techniques**

Under the serial data transfer mode, a single bit of information/data is transmitted on a single line at a time. But how would the 8085 accomplish this?

An 8-bit parallel word is first converted into a stream of 8 serial bits, with the help of a parallel – to – serial converter. In the same way, during the serial reception of information bits, the microprocessor receives a stream of separate 8 bits, one by one, which are later converted to an 8-bit parallel word with the help of a serial – to – parallel converter.

**Serial data communication can be categorized on the basis of how data transmission occurs. These are:**

**Simplex:** In simplex communication, the hardware is set such that the data exchange takes place in only one direction. Example: Computer to Printer communication.

**Half Duplex:** The half-duplex communication allows the data exchange in both directions, but not at the same time. Example: Walkie talkie.

**Full Duplex:** It permits the information transfer in both directions at the same time. Example: Telephone lines.

**The information in serial communication can be transmitted in two ways. They are:**

1. Synchronous Serial Data Transfer
2. Asynchronous Serial Data Transfer

Let us study each of these transfer schemes in detail.

**Asynchronous Serial Data Transfer**

The Asynchronous formats are more character-oriented. Here, the bits/character/data word are sent together at a constant rate, but the characters can arrive at any rate i.e., asynchronously, provided that these do not overlap. If no characters are being transmitted over the line, the line stays HIGH at Logic 1 called MARK. Logic 0 is called SPACE.

**At the transmitter end**

When a character (data bits) is to be sent, it is indicated by a start bit, which is always a logic 0 signal (coming down from a logic 1).

This signal then synchronizes the transmitter and the receiver.

Following this bit, the information bits are sent one bit at a time, with the Least Significant Bit sent first (D0-D7).

The size of the character (number of bits) can vary and depends on the system requirement.

Once all information is sent, a parity bit might be sent at the end.

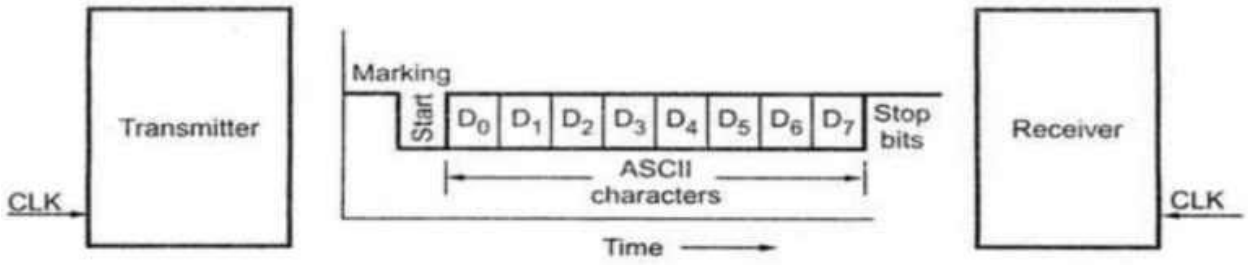
After that, the stop bit(s) is sent, which is a high signal.

This stop bit indicates the end of transmission of the information bits.

The start and stop bit carry no information but are just indications for the commencement and end of data exchange.

The combination of the start bit, the character bits, and stop bits is known as a frame.

This mode is used in the low-speed transmission of information, where speed might be less than 20kbps.




**At the receiver end**

The output of the transmitter is the input to the receiver.

The receiver is always on the lookout for the high signal transitioning into a low signal.

Once it detects a high signal that transitions into a low, it considers the low bit to be the start bit and only then starts accepting data.

Once all the data is transferred, the receiver waits for the stop bit to restart looking for the next start bit.

**Synchronous Serial Data Transfer**

In the asynchronous data transmission method, the start and stop bits included in every frame are the wasted overhead bytes, which reduces the overall character rate.

We can avoid using the start and stop bits for every frame of data by synchronizing the transmitter and the receiver instead. Once synchronized, any size of data can be transferred serially.

This synchronization can be done by placing synchronous bits in the place of the start and stop bits.

Such a protocol is known as synchronous serial communication.

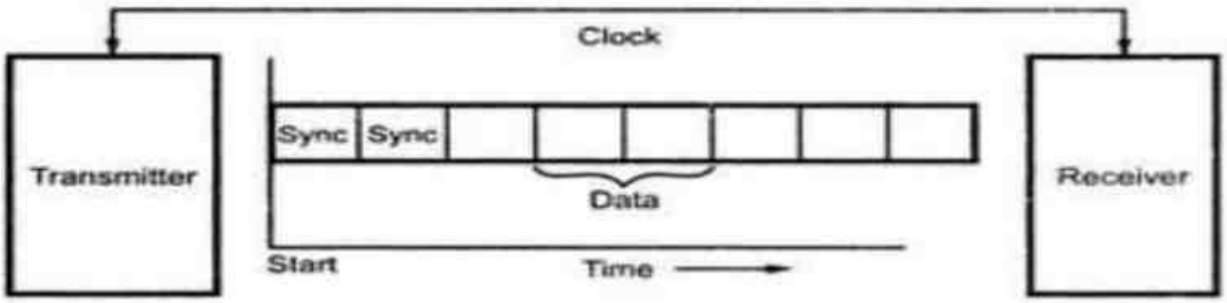
These synchronous bits are transferred by the transmitter. And at the receiver end, the receiver waits to detect these two or three sync bits.

Once detected, the transmitter and receiver are said to be in sync and data transmission begins.

Characters are received at a constant rate here, and the data transfer takes place in blocks.

Synchronous serial data transfer is ideal for high-speed transmission of information.

If no data bits exist, the transmitter still keeps transmitting the sync bits to keep the synchronization intact.




**Synchronous transmission format**

UNIT-8 Peripheral devices

## 8255 PPI

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 have 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., PC3 – PC0 and port C upper i.e., PC7 – PC4. 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:

**Bit set/reset mode**

**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.

**Modes of Operation**

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

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

***Mode 0***: Input/output mode

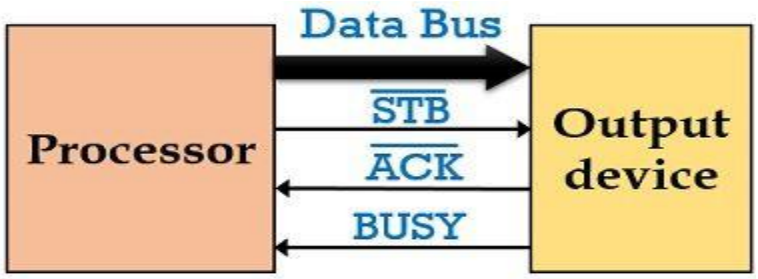
This mode is the simple input output mode of 8255 which allows the programming of each port as either input or output port. The input/output feature of mode 0 includes:

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

***Mode 1***: Input/output with handshaking

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

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




Data Transfer using handshaking signals

- Here STB signal is used to inform the output device that data is available on the data bus by the processor.

- Here port A and port B can be separately configured as either input or output port.
- Both the port utilizes 3-3 lines of port C for handshaking signals. The rest two lines operates as input/output port.
- It supports interrupt logic.
- The data at the input or output ports are latched.
-

***Mode 2***: Bidirectional I/O port with handshaking

In this mode, the ports can be utilized for the bidirectional flow of information by handshaking signals. The pins of group A can be programmed to acts as bidirectional data bus and the port C upper ($PC_7$ – $PC_4$) are used by the handshaking signal. The rest 4 lower port C bits are utilized for I/O operations.

As the data bus exhibits bidirectional nature thus when the peripheral device request for a data input only then the processor load the data in the data bus. Port B can be programmed in mode 0 and 1. And in mode 1 the lower bits of port C of group B are used for handshaking signals.

So, from the above discussion it is clear that how interfacing of peripheral devices is performed with the processor.

---

## What is Programmable Interval Timer?

The programmable Interval Timers are specially designed by Intel called as 8253 constructed for microprocessors to perform timing and counting functions by using three 16-bit registers. Each counter has 2 input pins, i.e. Clock & Gate, and 1 pin is for "OUT" output. To perform a counter, a 16-bit count is loaded in its register. On giving command, it begins to decrease the count until it reaches 0, then it produces a pulse that can be used to interrupt the CPU.

- Its operating frequency is 0 - 2.6 MHz.
- It uses N-MOS technology.
- Read-Back command is not available.
- Reads and writes of the same counter cannot be interleaved.

**Explain the operational modes of 8253?**

8253 consists of 6 different modes which perform specific operations. In this chapter, we will discuss these operational modes.

**Mode 0 ─ Interrupt on Terminal Count**

It helps in generating an interrupt to the microprocessor after a specific interval.

Initially the output becomes low after the mode is set. The output becomes LOW when the count value is loaded into the counter.

The output becomes low as the loaded count value decreases per every cycle. This process continues till the terminal count is reached to zero and the output goes HIGH and is considered as an interrupt. It will remain high until a new count is reloaded.

The GATE signal would be high for normal counting. When GATE goes low, counting gets terminated and the current count will be latched till the GATE goes high again.

**Mode 1 – Programmable One Shot**

It can be used as a mono stable multi-vibrator.

The gate input is used as a trigger input and the output becomes high in this mode.

The output remains high at the end of the count that is loaded and a trigger is applied.

**Mode 2 – Rate Generator**

The output remains normally high after initialization.

Whenever the count comes to zero, generation of low pulse occurs at the output and reloading of the counter will take place.

**Mode 3 – Square Wave Generator**

This mode acts similar to that of mode2 except the output remains low for half of the timer period and other half of the period it should be high.

**Mode 4 − Software Triggered Mode**

In this mode, the output would remain high until the timer has to be counted to zero, at certain point the output would pulse low and then would go high again.

The count will become latched when the GATE signal goes LOW.

At the end of the count, the output will go low for one clock cycle then it would go HIGH. This low pulse can be used as a strobe.

**Mode 5 – Hardware Triggered Mode**

This mode produces a strobe with response to an externally generated signal.

This mode acts similar to that of mode 4 except that the counting was initiated by a signal at the gate input, which implies that it is hardware triggered instead of software triggered.

After it gets initialized, the output goes high.

When the count was reached at the end, the output would go low for one clock cycle.

## Microprocessor - 8257 DMA Controller

DMA stands for Direct Memory Access. It is designed by Intel to transfer data at the fastest rate. It allows the device to transfer the data directly to/from memory without any interference of the 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.

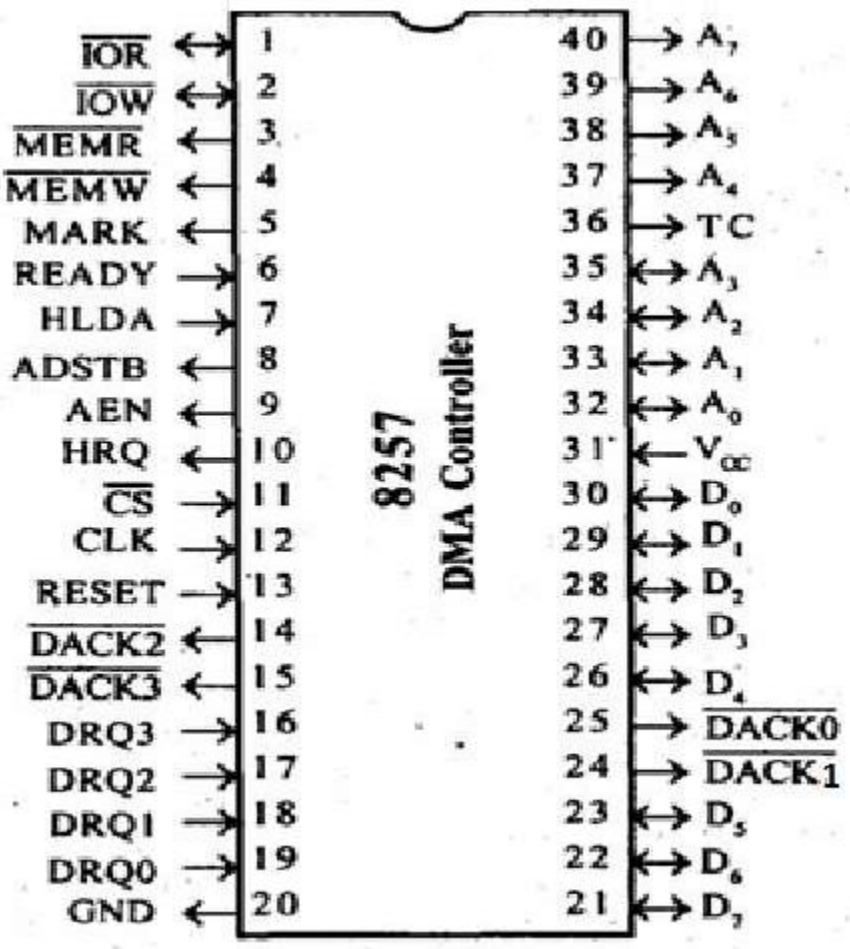
**Features of 8257**

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 Pin Description**

The following image shows the pin diagram of a 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.

DACKo − 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.

Do − 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.

Ao - 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.

Vcc

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

**Architecture of 8086**

Unlike microcontrollers, microprocessors do not have inbuilt memory. Mostly Princeton architecture is used for microprocessors where data and program memory are combined in a single memory interface. Since a microprocessor does not have any inbuilt peripheral, the circuit is purely digital and the clock speed can be anywhere from a few MHZ to a few hundred MHZ or even GHZ. This increased clock speed facilitates intensive computation that a microprocessor is supposed to do.

We will discuss the basic architecture of Intel 8086 before discussing more advanced microprocessor architectures.

**Internal architecture of Intel 8086:**

Intel 8086 is a 16 bit integer processor. It has 16-bit data bus and 20-bit address bus. The lower 16-bit address lines and 16-bit data lines are multiplexed (AD0-AD15). Since 20-bit address lines are available, 8086 can access up to 2 20 or 1 Giga byte of physical memory.

The basic architecture of 8086 is shown below.

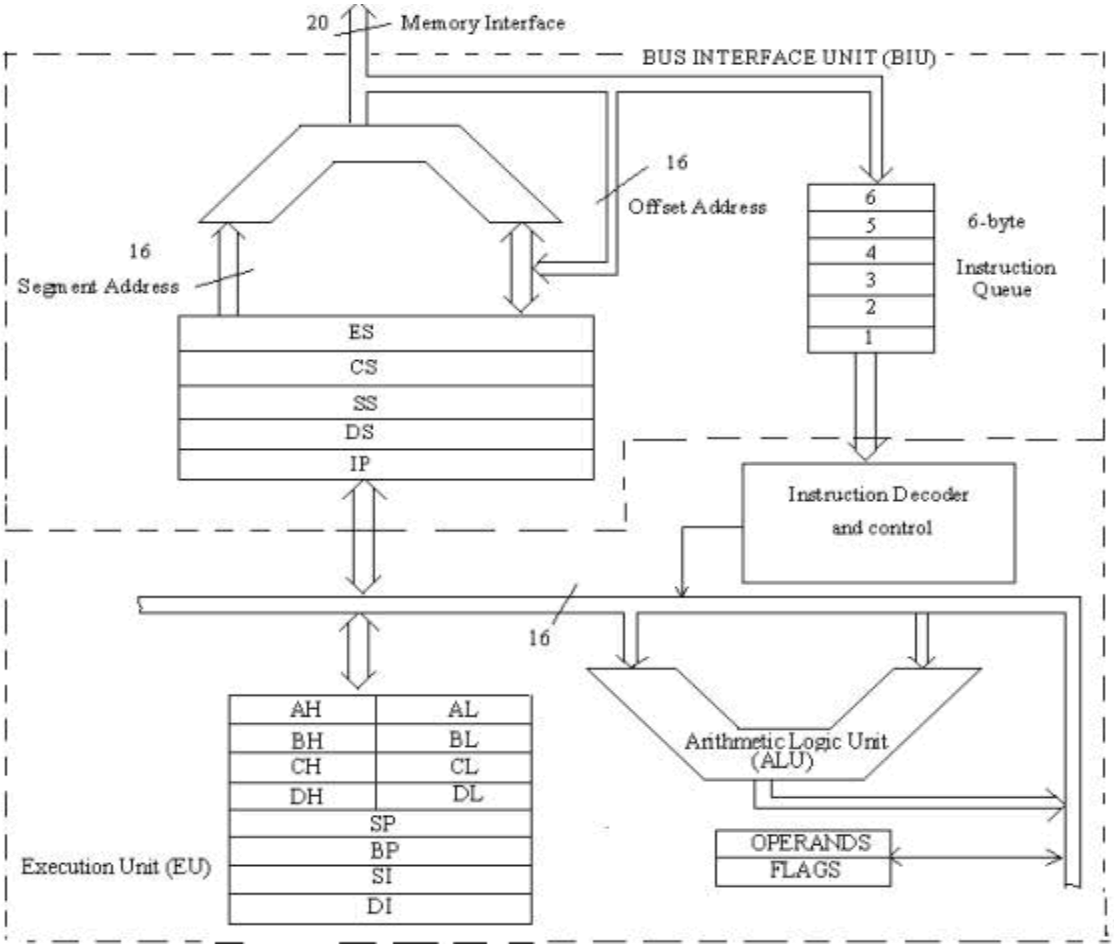



**Fig Basic Architecture of 8086 Microprocessor**

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

**Bus Interface Unit (BIU )**

The Bus Interface Unit (BIU) generates the 20-bit physical memory address and provides the interface with external memory (ROM/RAM). As mentioned earlier, 8086 has a single memory interface. To speed up the execution, 6-bytes of instruction are fetched in advance and kept in a 6-byte Instruction Queue while other instructions are being executed in the Execution Unit (EU). Hence after the execution of an instruction, the next instruction is directly fetched from the instruction queue without having to wait for the external memory to send the instruction. This is called pipe-lining and is helpful for speeding up the overall execution process.

8086's BIU produces the 20-bit physical memory address by combining a 16-bit segment address with a 16-bit offset address. There are four 16-bit segment registers, viz., the code segment (CS), the stack segment (SS), the extra segment (ES), and the data segment (DS). These segment registers hold the corresponding 16-bit segment addresses. A segment address is the upper 16-bits of the starting address of that segment. The lower 4-bits of the starting address of a segment is always zero. The offset address is held by another 16-bit register. The physical 20-bit address is calculated by shifting the segment address 4-bit left and then adding that to the offset address.

For Example:

Code segment Register CS holds the segment address which is 4569 H

Instruction pointer IP holds the offset address which is 10A0 H

The physical 20-bit address is calculated as follows.

| | | | |
|---|---|---|---|
| Segment address | : | 45690 | H |
| Offset address | :+ | 10A0 | H |

Physical address : 46730 H

## Execution Unit [EU]:

The execution unit of 8086 Internal Architecture tells the BIU from where to fetch instructions or data, decodes instructions and executes instructions. It contains

- Control Circuitry
- Instruction Decoder
- Arithmetic Logic Unit (ALU)
- Flag Register
- General Purpose Registers
- Pointers and Index Registers

The Execution unit is responsible for decoding and executing all instructions. • The EU extracts instructions from the top of the queue in the BIU, decodes them, generates operands if necessary, passes them to the BIU and requests it to perform the read or write bys cycles to memory or I/O and perform the

operation specified by the instruction on the operands. • During the execution of the instruction, the EU tests the status and control flags and updates them based on the results of executing the instruction.

## Minimum and Maximum mode

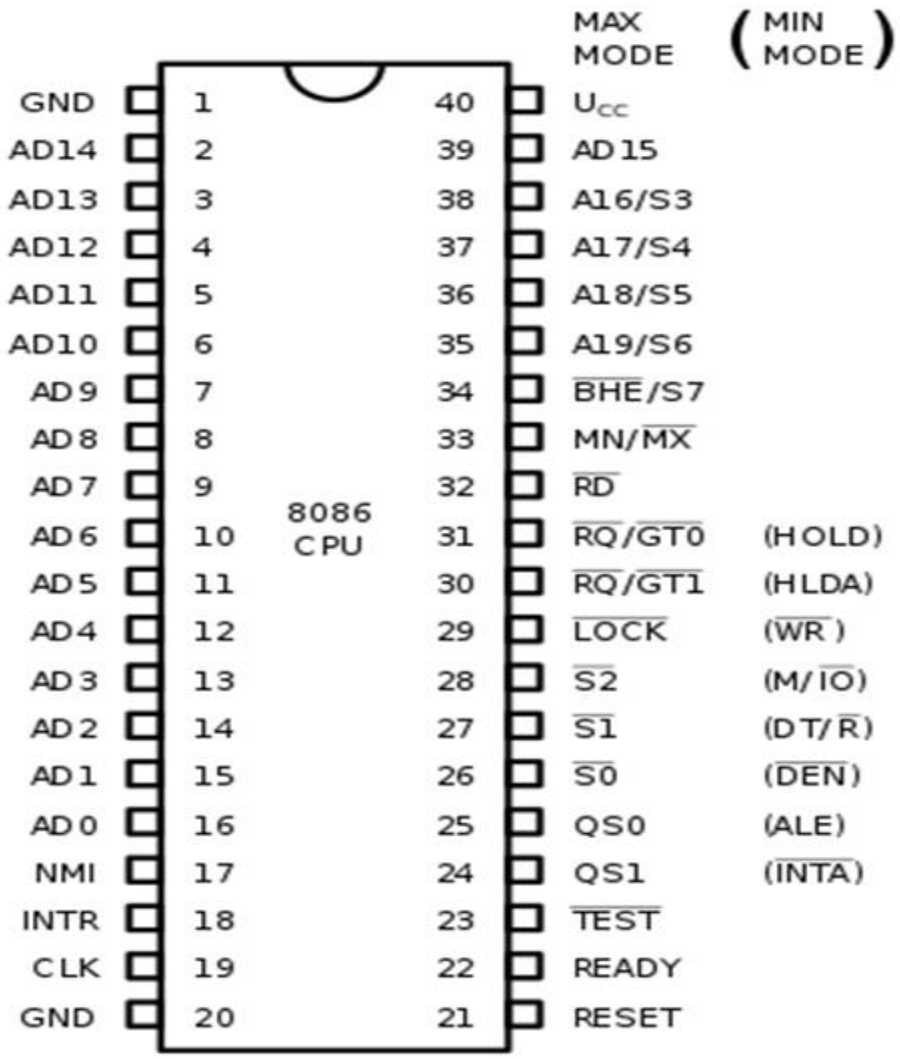
Minimum and Maximum Modes: • the minimum mode is selected by applying logic 1 to the MN / MX# input pin. This is a single microprocessor configuration.

The maximum mode is selected by applying logic 0 to the MN / MX# input pin. This is a multi micro processors configuration.

| **Minimum mode** | **Maximum mode** |
|---|---|
| In minimum mode there can be only one processor i.e. 8086. | In maximum mode there can be multiple processors with 8086, like 8087 and 8089. |
| MN/$\overline{MX}$ MN/$\overline{MX}$ is 1 to indicate minimum mode. | MN/$\overline{MX}$ MN/$\overline{MX}$ is 0 to indicate maximum mode. |
| ALE for the latch is given by 8086 as it is the only processor in the circuit. | ALE for the latch is given by 8288 bus controller as there can be multiple processors in the circuit. |
| $\overline{DEN}$ and DT/$\overline{R}$ for the trans-receivers are given by 8086 itself. | and DT/$\overline{R}$ for the trans-receivers are given by 8288 bus controller. |
| Direct control signals M/$\overline{IO}$, $\overline{RD}$ and $\overline{WR}$ are given by 8086. | Instead of control signals, each processor generates status signals called $\overline{S2}$, $\overline{S1}$ and $\overline{S0}$. |
| Control signals M/$\overline{IO}$, $\overline{RD}$ and $\overline{WR}$ are decoded by a 3:8 decoder like 74138. | Status signals $\overline{S2}$, $\overline{S1}$ and $\overline{S0}$are decoded by a bus controller like 8288 to produce control signals. |
| $\overline{INTA}$ is given by 8086 in response to an interrupt on INTR line. | $\overline{INTA}$ is given by 8288 bus controller in response to an interrupt on INTR line. |
| HOLD and HLDA signals are used for bus request with a DMA controller like 8237. | $\overline{RQ}$/$\overline{GT}$,lines are used for bus requests by other processors like 8087 or 8089. |

| Minimum mode | Maximum mode |
|---|---|
| The circuit is simpler. | The circuit is more complex. |
| Multiprocessing cannot be performed hence performance is lower. | As multiprocessing can be performed, it can give very high performance. |

Pin Diagram 8086 :



Pin description for 8086 Microprocessor

**AD0 to AD15**

These lines are multiplexed unidirectional address and status bus. What this means is that at time T1, they carry status signals and in remaining cycles, they carry status signals.

- S6 is used as bus master, which handles the internal bus control.
- S5 is used as interrupt flag.
- S4 and S3 are used to select the segment out of the four segments.

| S4 | S3 | Segment selected |
|---|---|---|
| 0 | 0 | ES |
| 0 | 1 | SS |
| 1 | 0 | CS |
| 1 | 1 | DS |

**BHE' / S7**

BHE stands for Bus High Enable. It is an active low signal, i.e. it is active when it is low. It is used to indicate the transfer of data over the higher order data bus (D8 to D15).

BHE' decides whether the data bus will carry 16-bit data or 8-bit data. When BHE' is enabled (i.e. 0), then the bus will carry 16-bit data, else only 8-bit data through the lower order data bus lines. It is multiplexed with status pin S7.

**RD'**

It is a read signal used for read operation. It is also an active low signal.

**READY**

This is an acknowledgment signal from the slower I/O devices or memory. When high, it indicates that the device is ready to transfer data, else the microprocessor is in the wait state.

**RESET**

By using this pin, the program control returns to $FFFF0_H$.

**INTR**

This pin is used to receive an interrupt request signal. It is a type of maskable interrupt.

**NMI**

This is used for Non-Maskable Interrupt Request.

**GND**

There are two ground pins in the 8086, pin 1 and pin 20.

**VCC**

The pin 40 is for voltage input.

**MN / MX'**

This pin is used for minimum or maximum mode of the microprocessor. When this pin is 1, the microprocessor works in minimum mode, and when the pin is at 0, the maximum mode is followed.

**Minimum and maximum Mode Pins**

Total 8 pins, from Pin 24 to pin 31 work differently for different modes (maximum or minimum).

**TEST'**

This is also an active low signal. This pin is used for wait instruction when the 8086 is connected with the 8087 microprocessor.

**CLK**

This pin tells about the clock pulse.

**Short Question /Answer**

**1. What is Microprocessor? Give the power supply & clock frequency of 8085?**

**Ans:** A microprocessor is a multipurpose, programmable logic device that reads binary instructions from a storage device called memory accepts binary data as input and processes data according to those instructions and provides result as output. The power supply of 8085 is +5V and clock frequency in 3MHz.

**2. List few applications of microprocessor-based system.**

**Ans:** It is used:

i. For measurements, display and control of current, voltage, temperature, pressure, etc.

ii. For traffic control and industrial tool control.

iii. For speed control of machines.

**3. What are the functions of an accumulator?**

**Ans:** The accumulator is the register associated with the ALU operations and sometimes I/O operations. It is an integral part of ALU. It holds one of data to be processed by ALU. It also temporarily stores the result of the operation performed by the ALU.

**4. List the 16 – bit registers of 8085microprocessor.**

**Ans:** Stack pointer (SP) and Program counter (PC).

**5. List the allowed register pairs of8085.**

**Ans:**

- B-C register pair
- D-E register pair
- H-L register pair

**6 . What is an Opcode?**

**Ans:** The part of the instruction that specifies the operation to be performed is called the operation code or opcode.

**7. What is the function of IO/M signal in the 8085?**

**Ans:** It is a status signal. It is used to differentiate between memory locations and I/O operations. When this signal is low (IO/M = 0) it denotes the memory related operations. When this signal is high (IO/M = 1) it denotes an I/O operation.

**8. What is an Operand?**

**Ans:** The data on which the operation is to be performed is called as an Operand.

**9. Define instruction cycle, machine cycle and T-state**

**Ans:-**Instruction cycle is defined, as the time required completing the execution of an instruction. Machine cycle is defined as the time required completing one operation of accessing memory, I/O or acknowledging an external request. T cycle is defined as one subdivision of the operation performed in one clock period

**10. What is an instruction?**

**Ans:-**An instruction is a binary pattern entered through an input device to command the microprocessor to perform that specific function

**11. What is the use of ALE**

**Ans:-**The ALE is used to latch the lower order address so that it can be available in T2 and T3 and used for identifying the memory address. During T1 the ALE goes high, the latch is transparent ie, the output changes according to the input data, so the output of the latch is the lower order address. When ALE goes low the lower order address is latched until the next ALE.

**12. How many machine cycles does 8085 have, mention them**

**Ans:** The 8085 have seven machine cycles. They are

- Opcode fetch
- Memory read
- Memory write
- I/O read
- I/O write
- Interrupt acknowledge
- Bus idle

**13. Explain the signals HOLD, READY and SID**

**Ans:** HOLD indicates that a peripheral such as DMA controller is requesting the use of address bus, data bus and control bus. READY is used to delay the microprocessor read or write cycles until a slow responding peripheral is ready to send or accept data.SID is used to accept serial data bit by bit

**14. What is Program counter?**

Program counter holds the address of either the first byte of the next instruction to be fetched for execution or the address of the next byte of a multi byte instruction, which has not been completely fetched. In both the cases it gets incremented automatically one by one as the instruction bytes get fetched. Also Program register keeps the address of the next instruction.

**15 .What is clock frequency for 8085?**

3 MHz is the maximum clock frequency for 8085.

**16. What is Stack Pointer?**

- Stack pointer is a special purpose 16-bit register in the Microprocessor, which holds the address of the top of the stack.

**17. What is Program counter?**

- Program counter holds the address of either the first byte of the next instruction to be fetched for execution or the address of the next byte of a multi byte instruction, which has not been completely fetched. In both the cases it gets incremented automatically one by one as the instruction bytes get fetched. Also Program register keeps the address of the next instruction.

**18. Which Stack is used in 8085?**

- LIFO (Last In First Out) stack is used in 8085.In this type of Stack the last stored information can be retrieved first.

19. **What are most common registers present in a microprocessor?**

Accumulator registers, Data registers, Temporary registers, Instruction registers, Stack Pointer, Program Counter and Condition Code Register.

20. **Why is address bus unidirectional?**

The address bus is unidirectional because the address is always given by the microprocessor, to address a memory location of an I/O device. Only microprocessor can write a value onto address bus, I/O devices can only read address bus.

21. **Why is data bus bidirectional?**

The data bus is bidirectional because it is used by microprocessor, memory units, and I/O devices for both to transfer and receive data.

## 4th Sem. / Computer Engg.

## Subject: Microprocessors & Peripherals Devices/ Microp. & App.

(1) 170844/120844/31045/30834

**SECTION-A**

**Note:** Objectives questions. All questions are compulsory (10x1=10)

Q.1 Word size of 8085 is _____ bits.

Q.2 The size of PC is _____ bits.

Q.3 Name the format of instruction DAD D.

Q.4 Instruction cycle = fetch cycle + _____

Q.5 Write two instructions used for subroutine operation.

Q.6 Write the I/O address Space of 8085.

Q.7 Name non-maskable interrupt found in 8085.

Q.8 Name the Data transfer technique in which handshaking is used.

Q.9 Expand PPI.

Q.10 Name two functional units of 8086.

**SECTION-B**

**Note:** Very Short answer type questions. Attempt any ten parts 10x2=20

Q.11 List four functions of ALU.

Q.12 Name the general purpose registers of 8085.

Q.13 Name the interrupt pins of 8085.

Q.14 Define instruction cycle.

Q.15 Identify the machine cycles of instruction MOV A, M.

Q.16 List the formats of instruction.

Q.17 Write the arithmetic equation of the instruction ADD D.

Q.18 Write down the four differences between Edge triggered and level triggered interrupts.

Q.19 Write four differences between peripheral I/O and memory mapped I/O.

Q.20 Define DMA operation.

Q.21 Write the main advantage of interrupt driven data transfer technique.

Q.22 Define the function of BIU and EU in 8086.

**SECTION-C**

**Note:** Short answer type questions. Attempt any eight questions. 8x5=40

Q.23 Explain how control signals are generated for memory and I/O with suitable logic diagram.

Q.24 Describe the evolution of microprocessor and its impacts on society.

Q.25 Explain memory read machine cycle with suitable timing diagram.

Q.26 Illustrate arithmetic group of instructions with suitable examples referring to 8085.

Q.27 Why decoding of memory address is required in memory accessing? Explain the working of 3-to- 8 line decoder with its diagram.

Q.28 Classify the interrupts of 8085. Explain the steps to process the interrupt generated in 8085.

Q.29 Explain the control word format of 8255 and define the purpose of each bit.

Q.30 Differentiate asynchronous (handshake) mode of data transfer and interrupt driven data transfer.

Q.31 Write a program in assembly language to find largest of ten nos. stored at some memory locations.

Q.32 Differentiate minimum and maximum mode of configuration of 8086.

**SECTION-D**

**Note:** Long answer type questions. Attempt any three questions. **10x3=30**

Q.33 Draw the pin diagram of 8085 and define the function of each pin.

Q.34 Explain programmed data transfer techniques with suitable diagrams.

Q.36 (a) Differentiate memory mapped I/O and peripheral I/O interfacing schemes.

(b) Draw the block diagram of 8086.

## 4th Sem. / Computer Engg.

## Subject : Microprocessors and Peripherals Devices/ Microp.& App

## SECTION-A

**Note:** Objective questions. All questions are compulsory. (10x1=10)

Q.1 Word size of 8085 is_____bits.

Q.2 Name the format of instruction DADD.

Q.3 What is the function of address bus?

Q.4 Write two instructions used for subroutine operation.

Q.5 Name the data transfer techniques.

Q.6 RST 7.5 is mask able interrupt (T/F).

Q.7 Name two functional units of 8086.

Q.8 Define stack.

Q.9 Define ALE.

Q.10 what is S/W interrupt.

**Note:** Very Short answer type questions. Attempt any ten questions. 10x2=20

Q.11 Name the interrupt pins of 8085.

Q.12 List four function of ALU.

Q.13 Write the arithmetic equation of the instruction ADD D.

Q.14 Explain peripheral I/O &memory mapped I/O.

Q.15 Define DMA operation.

Q.16 What is NOP?

Q.17 What is handshaking

Q.18 What is the function of Accumulator?

Q.19 Explain DMA.

Q.20 What is a buffer?

Q.21 What is assembler?.

Q.22 What is the function of O/P devices?

**SECTION-C**

**Note:** Short answer type questions. Attempt anyeight questions out of ten questions. 8x5=40

Q.23 Explain the evolution of microprocessor & its impacts on society.

Q.24 Classify the interrupts of 8085 in detail

Q.25 Differentiate minimum & maximum mode of configuration of 8086.

Q.26 Discuss various flags of 8085.

Q.27 Describe arithmetic group of instruction with suitable example referring to 8085.

Q.28 Write the various application of microprocessor.

Q.29 Write down various addressing modes of 8085.

Q.30 What is memory interfacing?

Q.31 Draw timing diagram of memory read cycle.

Q.32 Differentiate between Hardware & Software interrupt.

**SECTION-D**

**Note:** Long answer type questions. Attempt any three questions out of four questions. 3x10=30

Q.33 Draw and discuss the pin diagram of 8085 in details.

Q.34 Explain programmed data transfer techniques with suitable diagrams.

Q.35 Discuss and draw the block diagram of 8086 in details.

Q.36 Write short note on the following:-

a) stack

b) Non-mask able interrupt.