



# SRI G.C.S.R COLLEGE

(Affiliated to Dr. B. R. Ambedkar University, Srikakulam)

GMR Nagar, Rajam – 532127, Srikakulam (Dist.), A.P

T: +91 8941-251336, M: +91 89785 23866, F: +91 8941-251591,

www.srigcsrcollege.org



*Department of ELECTRONICS*

**II B.Sc. IV – Semester**  
**Electronics Paper-4**

## **Microprocessors** **&** **Interfacing** (Study Material)

Name of the Student : \_\_\_\_\_

Roll Number : \_\_\_\_\_

Group : \_\_\_\_\_

Year/ Semester : \_\_\_\_\_

**Prepared by:**

**K. Venugopala Rao** M.Sc.(Tech), M. Tech, (PhD)

Lecturer in Electronics

SRI GCSR COLLEGE, GMR Nagar, Rajam

venugopalarao.k@sgsrc.edu.in

+91-9603803168, +91-9182061933

**Department of ELECTRONICS**



**Dr. B. R. AMBEDKAR UNIVERSITY-SRIKAKULAM**  
**B.Sc. ELECTRONICS SYLLABUS**  
**STRUCTURE UNDER CHOICE BASED CREDITS SYSTEM**  
**REVIEWED SYLLABUS w.e.f. 2021-22**  
**II B.Sc. Semester – IV**  
**Paper – IV: Microprocessors & Interfacing**

**UNIT-I (12Hrs): CPU ARCHITECTURE**

Introduction to Microprocessor, INTEL -8085 Architecture, CPU, ALU unit, Register organization, Address, data and control Buses. Pin configuration of 8085. Addressing modes.

**UNIT-II (12Hrs): 8085 INSTRUCTION SET & PROGRAMMING USING 8085**

Data transfer Instruction, Logical Instructions, Arithmetic Instructions, Branch Instructions, Machine Control instructions. Programs for Addition, Subtraction, Multiplication, Division, largest and smallest number in an array, Ascending and Descending order in an Array.

**UNIT-III (12Hrs): 8086 MICROPROCESSORS**

8086 Microprocessor: Architecture, Pinout diagram, Addressing modes, Basic 8086 Configurations, Minimum mode and Maximum Mode, Interrupt Priority Management I/O Interfaces: Serial Communication interfaces, Parallel Communication

**UNIT-IV (12Hrs): INTERFACING**

Serial communication interface (8251 – USART), Programmable peripheral interface (8255 PPI), Programmable interval timer (8253), DMA Controller (8257/8237), Keyboard and Display interface (8279).

**UNIT-V (12Hrs): ARM PROCESSOR**

Introduction to 16/32-bit processors, Arm architecture & organization, ARM based MCUs, Instruction set.

## UNIT-I

### CPU ARCHITECTURE

#### 1.1 INTRODUCTION TO MICROPROCESSORS

##### 1.1(a) Terms Used in Microprocessor Literature

- Bit** : A digit of the binary number or code is called a bit.
- Nibble** : The 4-bit (4-digit) binary number or code is called a nibble.
- Byte** : The 8-bit (8-digit) binary number or code is called a byte.
- Word** : The 16-bit (16-digit) binary number or code is called a word.
- Double Word** : The 32-bit (32-digit) binary number or code is called a double word.
- Multiple Word** : The 64, 128, ... bit/digit binary numbers or codes are called multiple words.
- Data** : The quantity (binary number/code) operated by an instruction of a program is called data. The size of data is specified as bit, byte, word, etc.
- Address** : Address is an identification number in binary for memory locations.  
The 8085 processor uses a 16-bit address for memory.
- Memory Word Size (Or Addressability)** : Memory word size or addressability is the size of the binary information that can be stored in a memory location. The memory word size for 8085 and 8086 processor-based systems is 8-bit.

**Note:** The address and program codes in a microprocessor system are given in binary (i.e., as a combination of "0" and "1"). With an n-bit binary we can generate  $2^n$  different binary codes or addresses.

- Microprocessor** : A microprocessor is a program-controlled semiconductor device (or IC), which fetches (from memory), decodes and executes instructions. It is used as a CPU (Central Processing Unit) in computers. The basic functional blocks of a microprocessor are ALU (Arithmetic Logic Unit), an array of registers and a control unit. The microprocessor is identified with the size of the data the ALU of the processor can work with at a time. The 8085 processor has an 8-bit ALU, hence it is called a 8-bit processor. The 8086 processor has a 16-bit ALU, hence it is called a 16-bit processor.

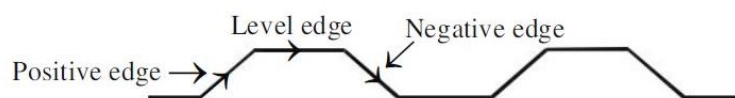
**Bus** : A bus is a group of conducting lines that carries data, address and control signals. Buses can be classified into data bus, address bus and control bus. The group of conducting lines that carries data is called a data bus. The group of conducting lines that carries address is called an address bus. The group of conducting lines that carries control signals is called a control bus.

**CPU Bus** : The group of conducting lines that are directly connected to a microprocessor is called a CPU bus. In a CPU bus the signals are multiplexed, i.e., more than one signal is passed through the same line but at different timings.

**System Bus** : The group of conducting lines that carries data, address and control signals in a microcomputer system is called a system bus. Multiplexing is not allowed in a system bus.

[In microprocessor-based systems each bit of information (data /address /control signal) is sent through a separate conducting line. Due to practical limitations the manufacturers of microprocessors may provide multiplexed pins, i.e., one pin is used for more than one purpose. This leads to multiplexed CPU buses. For example, in an 8085 processor the address and data are sent through the same pins but at different timings. But when the system is formed the multiplexed bus lines should be de-multiplexed by using latches, ports, transceivers, etc. The de-multiplexed bus lines is called a system bus. In a system, separate conducting line will be provided for each bit of data, address and control signals.]

**Clock** : A clock is a square wave which is used to synchronize various devices in the microprocessor and in the system. Every microprocessor system requires a clock for its functioning. The time taken for the microprocessor and the system to execute an instruction or program is measured only in terms of the time period of its clock.



A clock has three edges: Rising edge (positive edge), level edge and falling edge (negative edge). The devices are made sensitive to any one of the edges for better functioning (it means that the device will recognize the clock only when the edge is asserted or arrived).

**Tristate Logic** : Almost all the devices used in a microprocessor-based system use tristate logic. In devices with tristate logic, three logic levels will be available. These are high state, low state and high impedance state.

The high and low are normal logic levels for data, address or control signals. The high impedance state is an electrical open circuit condition. It is provided to keep the device electrically isolated from the system. Tristate devices will normally remain in high impedance state and their pins are physically connected in the system bus but electrically isolated. In high impedance state they cannot receive or send any signal or information. These devices are provided with chip enable or chip select pins. When the signal at this pin is asserted to the right level, they come out from the high impedance state to normal levels.

### **1.1(b) Evolution of Microprocessors**

In 1971, Intel Corporation released the world's first microprocessor - the INTEL 4004, a 4-bit microprocessor. It addresses 4096 memory locations of word size 4-bit. The instruction set consists of 45 different instructions. It is a monolithic IC employing large-scale integration with PMOS technology. The INTEL 4004 was soon followed by a variety of microprocessors, with most of the major semiconductor manufacturers producing one or more types.

#### **First Generation Microprocessors:**

The microprocessors introduced between 1971 and 1973 were the first-generation processors. They were designed using PMOS technology. This technology provided low cost, slow speed and low output currents and was not compatible with TTL (Transistor Transistor Logic) levels. The first-generation processors require a lot of additional support ICs to form a system. It may require as high as 30 ICs to form a system. The 4-bit processors are provided with only 16 pins, but 8-bit and 16-bit processors are provided with 40 pins. Due to the limitations of pins, the signals are multiplexed. A list of first-generation microprocessors is given below:

- INTEL 4004
  - INTEL 4040
  - FAIR CHILD PPS - 25
  - NATIONAL IMP - 4
  - ROCKWELL PPP - 4
  - MICRO SYSTEMS INTL. MC - 1
- } 4-bit processors

- INTEL 8008
  - NATIONAL IMP - 8
  - ROCKWELL PPS - 8
  - AMI 7200
  - MOSTEK 5065
  - NATIONAL IMP/16
  - NATIONAL PACE
- } 8-bit processors
- } 16-bit processors

### **Second Generation Microprocessors:**

The second-generation microprocessors appeared in 1973 and were manufactured with NMOS technology. The NMOS technology offers faster speed and higher density than PMOS and it is TTL-compatible. Some of the second-generation processors are given below:

- INTEL 8080
  - INTEL 8085
  - FAIRCHILD F - 8
- } 8-bit processors
- MOTOROLA M6800
  - MOTOROLA M6809
  - NATIONAL CMP - 8
  - RCA COSMAC
  - MOS TECH. 6500
  - SIGNETICS 2650
  - ZILOG Z80
- } 8-bit processors
- INTERSIL 6100
  - TOSHIBA TLCS - 12
- } 12-bit processors
- TI TMS 9900
  - DEC - W.D. MCP - 1600
  - GENERAL INSTRUMENT CP 1600
  - DATA GENERAL  $\mu$ N601
- } 16-bit processors

### **Characteristics of second-generation microprocessors**

- Larger chip size (170 × 200 mil). [1mil = 10<sup>-3</sup> inch]
- 40 pins.
- More number of on-chip decoded timing signals.
- The ability to address large memory spaces.
- The ability to address more IO ports.
- Faster operation.
- More powerful instruction set.
- A greater number of levels of subroutine nesting.
- Better interrupt handling capabilities.

### **Third Generation Microprocessors:**

After 1978, the third-generation microprocessors were introduced. These are 16 – bit processors and designed using HMOS (High Density MOS) technology. Names of some of the third-generation microprocessors are given below:

- INTEL 8086
- INTEL 8088
- INTEL 80186
- INTEL 80286
- MOTOROLA 68000
- MOTOROLA 68010
- ZILOG Z8000
- NATIONAL NS 16016
- TEXAS INSTRUMENTS TMS 99000

The HMOS technology offers better Speed Power Product (SPP) and higher packing density than NMOS.

$$\begin{aligned} \text{Speed Power Product} &= \text{Speed} \times \text{Power} \\ &= \text{Nanosecond} \times \text{Milli watt} \\ &= \text{Pico joules} \end{aligned}$$

- Speed Power Product of HMOS is four times better than NMOS.  
SPP of NMOS = 4 picojoules (pJ)  
SPP of HMOS = 1 picojoules (pJ)
- Circuit densities provided by HMOS are approximately twice those of NMOS.  
Packing density of NMOS = 1852.5 gates/mm<sup>2</sup>  
Packing density of HMOS = 4128 gates/mm<sup>2</sup> (1 mm = 10<sup>-6</sup> meter)

### **Characteristics of third generation microprocessors**

- Provided with 40/48/64 pins.
- High speed and very strong processing capability.
- Easier to program.
- Allow dynamically relocatable programs.
- Size of internal registers are 8/16/32 bits.
- The processor has multiply/divide arithmetic hardware.
- Physical memory space is from 1 to 16 Megabytes.
- The processor has segmented addresses and virtual memory features.
- More powerful interrupt handling capabilities.
- Flexible IO port addressing.
- Different modes of operations (e.g., user and supervisor modes of M68000).

### **Fourth Generation Microprocessors:**

The fourth-generation microprocessors were introduced in the year 1980. The fourth-generation processors are 32-bit processors and are fabricated using the low-power version of the HMOS technology called HCMOS. These 32-bit microprocessors have increased sophistication that compete strongly with mainframes. Some of the fourth-generation microprocessors are given below:

- INTEL 80386
- INTEL 80486
- NATIONAL NS16032
- MOTOROLA M68020
- BELLMAC - 32
- MOTOROLA M68030
- MOTOROLA MC88100

### Characteristics of fourth generation microprocessors

- Physical memory space of 224 bytes = 16 Mb (Megabytes).
- Virtual memory space of 240 bytes = 1 Tb (Terabytes).
- Floating-point hardware is incorporated.
- Supports increased number of addressing modes.

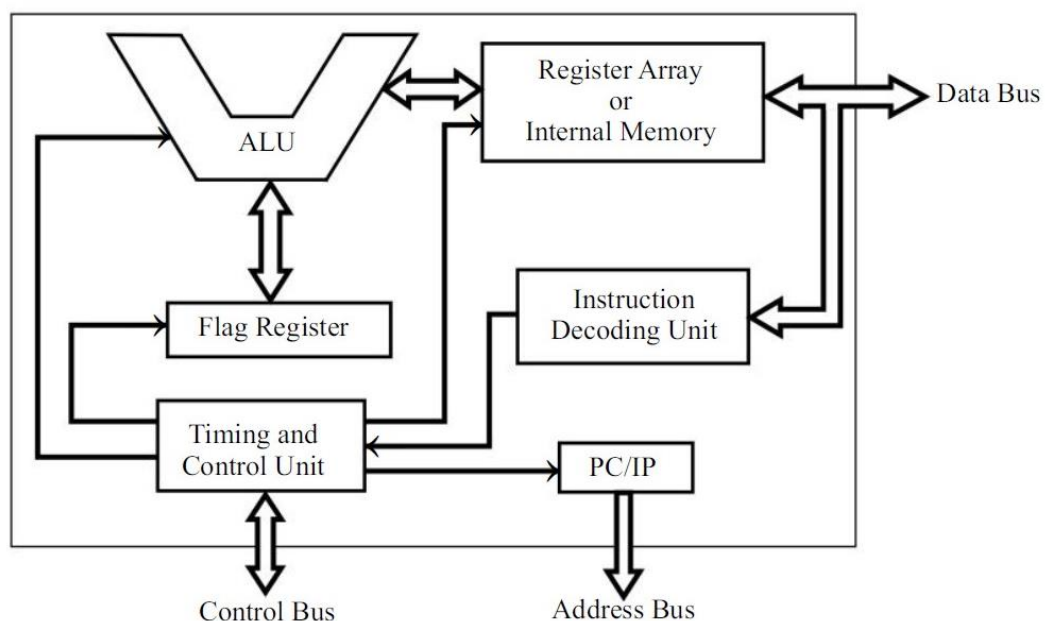
### Fifth Generation Microprocessors:

In microprocessor technology, INTEL has taken a leading edge and is developing more and more new processors. The INTEL Pentium processor released in the year 1993 is considered as a fifth-generation processor. The Pentium is a 32-bit processor with a 64-bit data bus and is available in wide range of clock speeds from 60 MHz to 3.2 GHz. With improvement in semiconductor technology, the processing speed of microprocessors have been increased tremendously. The 8085 released in the year 1976 executes 0.5 million Instructions Per Second (0.5 MIPS). The 80486 executes 54 million Instructions Per Second. The Pentium is optimized to execute two instructions in one clock period. Therefore, a Pentium processor working at 1GHz clock can execute 2000 MIPS.

### 1.1(c) Basic Functional Blocks of a Microprocessor

A microprocessor is a programmable IC which is capable of performing arithmetic and logical operations. The basic functional block diagram of a microprocessor is shown in Fig. 1.1.

The basic functional blocks of a microprocessor are ALU, Flag register, register array, Program Counter (PC)/Instruction Pointer (IP), Instruction decoding unit and Timing and Control unit.



**Fig. 1.1: Block diagram showing basic functional blocks of a microprocessor.**



ALU is the computational unit of the microprocessor which performs arithmetic and logical operations on binary data. The various conditions of the result are stored as status bits called flags in the flag register. For example, consider the sign flag: one of the bit positions of the flag register is called the sign flag and it is used to store the status of the sign of the result of the ALU operation (output data of ALU). If the result is negative, then "1" is stored in the sign flag and if the result is positive then "0" is stored in the sign flag.

The register array is the internal storage device and so it is also called the internal memory. The input data for ALU, the output data of ALU (result of computations) and any other binary information needed for processing are stored in the register array.

For any microprocessor, there will be a set of instructions given by the manufacturer of the microprocessor. For doing any useful work with the microprocessor, we have to write a program using these instructions and store them in a memory device external to the microprocessor.

The instruction pointer generates the address of the instructions to be fetched from the memory and send through the address bus to the memory. The memory will send the instruction codes and data through the data bus. The instruction codes are decoded by the decoding unit which sends the information to the timing and control unit. The data is stored in the register array for processing by the ALU.

The control unit will generate the necessary control signals for the internal and external operations of the microprocessor.

#### **1.1(d) Microprocessor Based Systems** **(Organization of a microcomputer)**

A microprocessor is a semiconductor device (or integrated circuit) manufactured by using the VLSI (Very Large-Scale Integration) technique. It includes the ALU, the register arrays and the control circuit on a single chip. To perform a function or useful task we have to form a system by using the microprocessor as a CPU (Central Processing Unit) and interfacing the memory, input and output devices to it. A system designed by using a microprocessor as its CPU is called a microcomputer or a single board microcomputer. A microprocessor-based system consists of a microprocessor as the CPU, semiconductor memories like EPROM and RAM, an input device, an output device and interfacing devices. The memories, input device, output device and interfacing devices are called peripherals.

The commonly used EPROM and static RAM in microcomputers are given below:

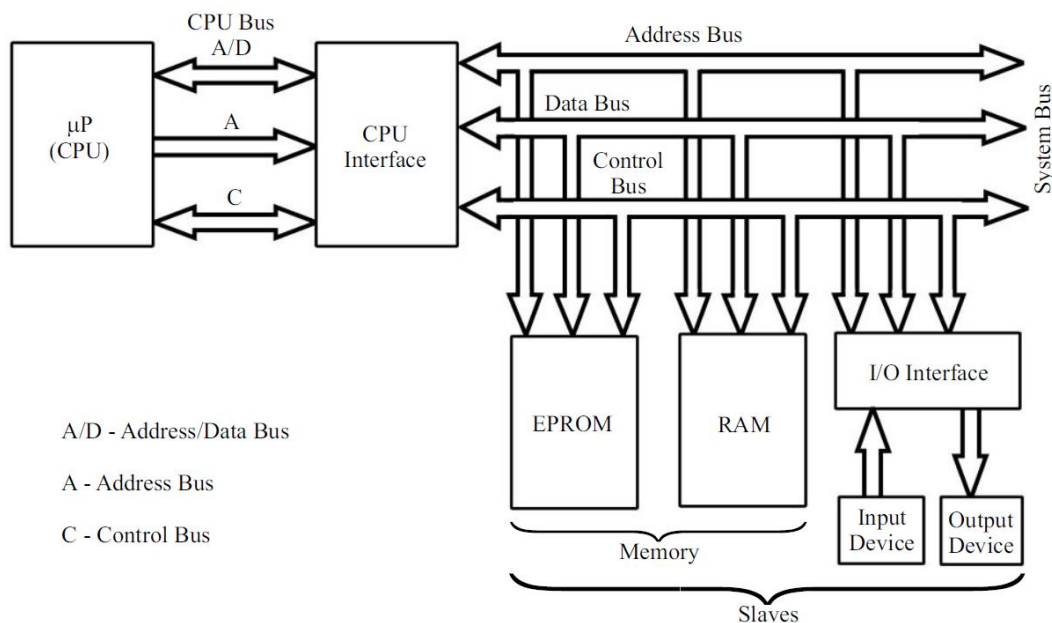
INTEL 2708 (1 kb)  
 INTEL 2716 (2 kb)  
 INTEL 2732 (4 kb)  
 INTEL 2764 (8 kb)

MOTOROLA 6208 (1 kb)  
 MOTOROLA 6216 (2 kb)  
 MOTOROLA 6232 (4 kb)  
 MOTOROLA 6264 (8 kb)

*Note : kb refers to Kilobytes.*

Popular input devices are keyboard, floppy disk, etc., and output devices are printer, LED and LCD displays, CRT monitor, etc.

The block diagram of a microprocessor-based system (or organization of microcomputer) is shown in Fig. 1.2. In this system the microprocessor is the master and all other peripherals are slaves. The master controls all the peripherals and initiates all operations.



**Fig. 1.2: Microprocessor-based system (organization of microcomputer)**

Buses are groups of lines that carry data, addresses or control signals. The CPU bus has multiplexed lines, i.e., the same line is used to carry different signals. The CPU interface is provided to de-multiplex the multiplexed lines to generate chip select signals and additional control signals. The system bus has separate lines for each signal.

All the slaves in the system are connected to the same system bus. At any one-time communication takes place between the master and one of the slaves. All the slaves have tristate logic and hence normally remain in high impedance state. The processor selects a slave by sending an address. When a slave is selected, it comes to the normal logic and communicates with the processor.

The EPROM memory is used to store permanent programs and data. The RAM memory is used to store temporary programs and data. The input device is used to enter the program, data and to operate the system. The output device is used for examining the results. Since the speed of IO devices does not match with the speed of the microprocessor, an interface device is provided between the system bus and the IO devices. Generally, IO devices are slow devices.

The work done by the processor can be classified into the following three groups:

1. Work done internal to the processor.
2. Work done external to the processor.
3. Operations initiated by the slaves or peripherals.

The work done internal to the processor are additions, subtractions, logical operations, data transfer within registers, etc. The work done external to the processor are reading / writing the memory and reading /writing the IO devices or the peripherals. If the peripheral requires the attention of the master, then it can interrupt the master and initiate an operation.

The microprocessor is the master which controls all the activities of the system. To perform a specific job or task, the microprocessor has to execute a program stored in the memory. The program consists of a set of instructions stored in consecutive memory locations. In order to execute the program, the microprocessor issues address and control signals to fetch the instructions and data from the memory one by one. After fetching each instruction, it decodes the instructions and performs the task specified by the instruction.

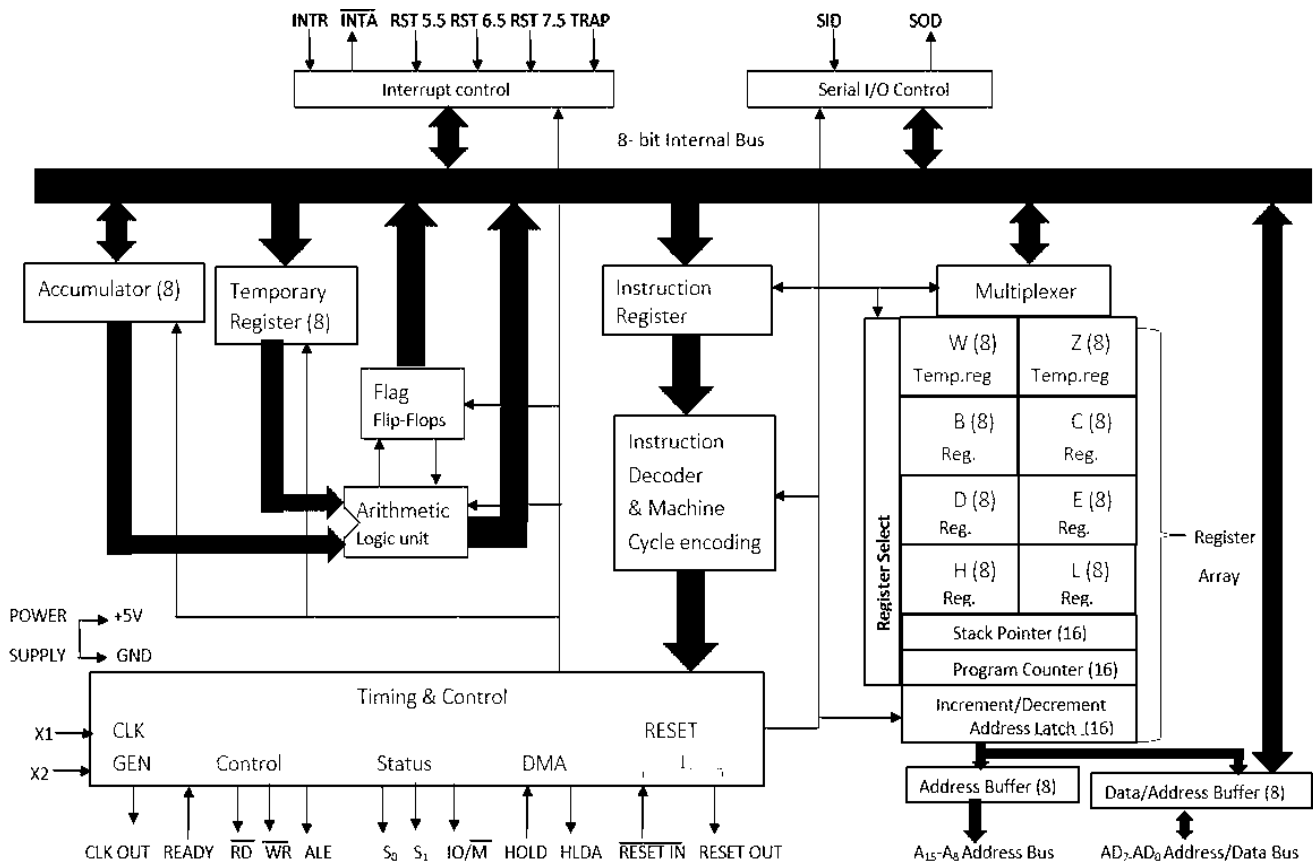
## **1.2 INTEL 8085 MICROPROCESSOR ARCHITECTURE**

### **Functional Block diagram of INTEL-8085 Microprocessor**

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

1. 8-bit data bus.
2. 16-bit address bus, which can address up to 64KB.
3. A 16-bit program counter and a 16-bit stack pointer.
4. Six 8-bit registers arranged in pairs: BC, DE, HL.
5. Requires +5V supply to operate at 3.2 MHZ single phase clock.

It is used in washing machines, microwave ovens, mobile phones, etc.,



**Fig:1.3: Intel-8085 Microprocessor Architecture**

### **8085 consists of the following functional units:**

#### **Accumulator**

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

#### **Arithmetic and logic unit**

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

#### **General purpose register**

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

#### **Program counter (PC)**

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

**Stack pointer (SP)**

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

**Temporary register**

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

**Flag register**

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

These are the set of 5 flip-flops –

Sign (S)

Zero (Z)

Auxiliary Carry (AC)

Parity (P)

Carry (C)

Its bit position is shown in the following table –

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

**Instruction register and decoder**

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

**Timing and control unit**

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

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

Status Signals: S0, S1, IO/M'

DMA Signals: HOLD, HLDA

RESET Signals: RESET IN, RESET OUT

**Interrupt control**

As the name suggests it controls the interrupts during a process. When a microprocessor is executing a main program and whenever an interrupt occurs, the microprocessor shifts the

control from the main program to process the incoming request. After the request is completed, the control goes back to the main program.

There are 5 interrupt signals in 8085 microprocessors: INTR, RST 7.5, RST 6.5, RST 5.5, TRAP.

### **Serial Input/output control**

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

### **Address buffer and address-data buffer**

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

### **Address bus and data bus**

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

## **1.3 8085 REGISTER STRUCTURE**

### **(8085 Register Organization) (8085 Registers)**

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

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

### **1. General Purpose Registers**

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

- Move to A register the contents of B register.

- Then add A and E registers. The result will be in A.
- Move this result from A register to B register.

It is possible to use these registers as pairs to store 16-bit information. Only BC, DE, and HL can form register pairs. When they are used as register pairs in an instruction, the left register is understood to have the MS byte and the right register the LS byte. For example, in DE register pair, the content of the D register is treated as the MS byte, and the content of E register is treated as the LS byte.

## 2. Temporary Registers

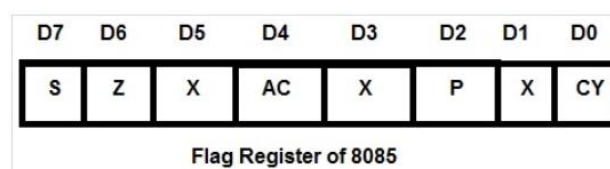
**(a) Temporary Data Register** - The ALU has two inputs. One input is supplied by the accumulator and other from the temporary data register. The programmer cannot access this temporary data register. However, it is internally used for execution of most of the arithmetic and logical instructions.

**(b) W and Z registers** - W and Z registers are temporary registers. These registers are used to hold 8-bit data during the execution of some instructions. These registers are not available for the programmer since 8085 Microprocessor Architecture uses them internally.

## 3. Special Purpose Registers

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

**(b) Flag Register** - It is a 8-bit register, in which five of the bits carry significant information in the form of flags: S (Sign flag), Z (Zero flag), AC (Auxiliary carry flag), P (Parity flag), and CY (carry flag); as shown in below Fig.



- **S-Sign flag** - After the execution of arithmetic or logical operations, if bit D7 of the result is 1, the sign flag is set. In a given byte if D7 is 1, the number will be viewed as a negative number. If D7 is 0, the number will be considered as a positive number.
- **Z-Zero flag** -The zero flag sets if the result of the operation in ALU is zero and flag resets if the result is non zero. The zero flags are also set if a certain register content becomes zero following an increment or decrement operation of that register.
- **AC-auxiliary Carry flag** - This flag is set if there is an overflow out of bit 3 i.e. carry from lower nibble to higher nibble (D3 bit to D4 bit). This flag is used for BCD operations and it is not available for the programmer.
- **P-Parity flag** - Parity is defined by the number of ones present in the accumulator. After arithmetic or logical operation, if the result has an even number of ones, i.e., even parity, the flag is set. If the parity is odd, the flag is reset.
- **CY-Carry flag** - This flag is set if there is an overflow out of bit 7. The carry flag also serves as a borrow flag for subtraction. In both the examples shown below, the carry flag is set.

Addition		
9BH	----->	1001 1011
+75 H	-----> +	0111 0101
-----		-----
Carry 1 10 H	----->	1 0001 0000
Subtraction		
89H	----->	1000 1001
-AB H	-----> -	1010 1011
-----		-----
Borrow1 10 H	----->	1 1101 1110



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

#### **4.SixteenBit Registers**

**a) Program counter (PC)** - Program is a sequence of instructions. As mentioned earlier, microprocessor fetches these instructions from the memory and executes them the program counter is a special purpose register which, at a given time, stores the address of the next instruction to be fetched. Program Counter acts as a pointer to the next instruction. How processor increments program counter depends on the nature of the instruction; for one-byte instruction it increments program counter by one, for two-byte instruction it increments program counter by two and for three-byte instruction it increments program counter by three such that program counter always points to the address of the next instruction.

In case of JUMP and CALL instructions, address followed by JUMP and CALL instructions is placed in the program counter. The processor then fetches the next instruction from the new address specified by JUMP or CALL instruction. In conditional JUMP and conditional CALL instructions, if the condition is not satisfied, the processor increments program counter by three so that it points the instruction followed by conditional JUMP or CALL instruction; otherwise, processor fetches the next instruction from the new address specified by JUMP or CALL instruction.

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

#### **1.4 8085 PIN DESCRIPTION**

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

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

The following image depicts the pin diagram of 8085 Microprocessor –

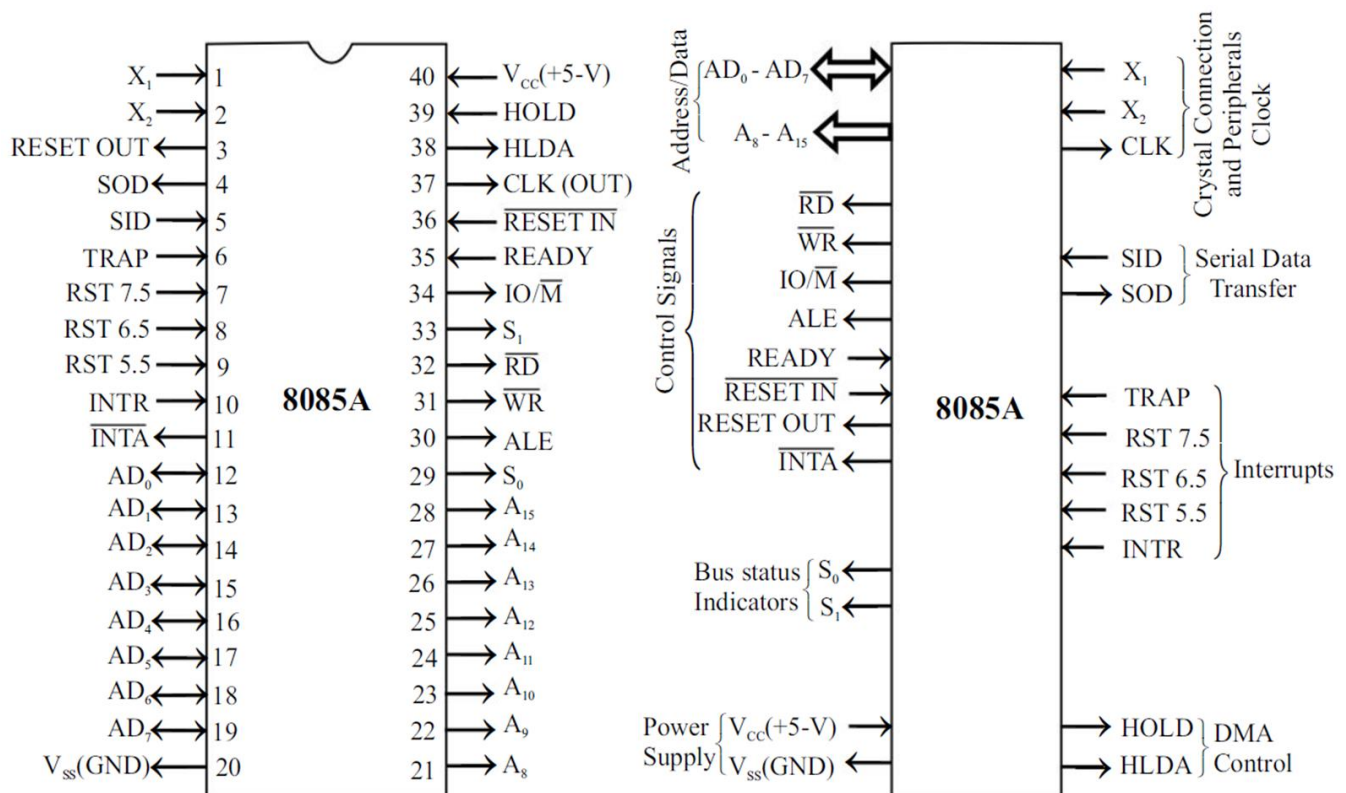


Fig. 1.4: 8085 pin Configuration

### 1. Address bus

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

### 2. Data bus

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

### 3. Control and status signals

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

(a) Three control signals are RD, WR & ALE.

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

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

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

(b) Three status signals are IO/M, S0 & S1.

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

**S1 & S0** – These signals are used to identify the type of current operation.

S. No	IO/M'	S1	S0	Status
1	0	0	0	Memory Write
2	0	1	0	Memory Read
3	1	0	1	I/O Write
4	1	1	0	I/O Read
5	1/0	1	1	Opcode fetch
6	1	1	1	Interrupt Acknowledge

#### **4. Power supply**

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

#### **5. Clock Signals**

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

**X1, X2** – A crystal (RC, LC N/W) is connected at these two pins and is used to set frequency of the internal clock generator. This frequency is internally divided by 2.

**CLK OUT** – This signal is used as the system clock for devices connected with the microprocessor.

#### **6. Interrupts & externally initiated signals**

Interrupts are the signals generated by external devices to request the microprocessor to perform a task. There are 5 interrupt signals, i.e., TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR.

**INTA'** – It is an interrupt acknowledgment signal.

**RESET IN'** – This signal is used to reset the microprocessor by setting the program counter to zero.

**RESET OUT** – This signal is used to reset all the connected devices when the microprocessor is reset.

**READY** – This signal indicates that the device is ready to send or receive data. If READY is low, then the CPU has to wait for READY to go high.

**HOLD** – This signal indicates that another master is requesting the use of the address and data buses.

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

### 7. Serial I/O signals

SID is a data line for serial input whereas SOD is a data line for serial output.

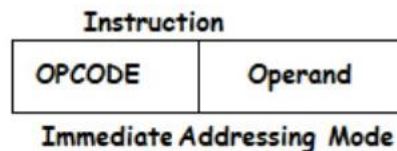
## 1.5 ADDRESSING MODES

Every instruction of a program has to operate on a data. The method of specifying the data to be operated by the instruction is called Addressing. The 8085 has the following 5 different types of addressing.

1. Immediate Addressing
2. Direct Addressing
3. Register Addressing
4. Implied Addressing
5. Register Indirect Addressing

### 1. Immediate Addressing:

#### Format:



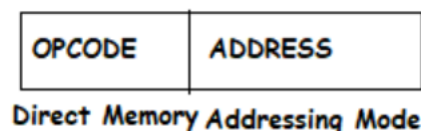
In immediate addressing mode, the data is specified in the instruction itself. The data will be a part of the program instruction.

**EX.** MVI B, 3EH - Move the data 3EH given in the instruction to B register;

LXI SP, 2700H.

### 2. Direct Addressing:

#### Format:



In direct addressing mode, the address of the data is specified in the instruction. The data will be in memory. In this addressing mode, the program instructions and data can be stored in different memory.

**EX.** LDA 1050H - Load the data available in memory location 1050H in to accumulator;  
SHLD 3000H

### **3. Register Addressing:**

#### **Format:**

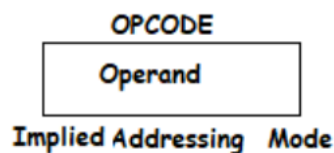


In register addressing mode, the instruction specifies the name of the register in which the data is available.

**EX.** MOV A, B - Move the content of B register to A register;  
SPHL; ADD C.

### **4. Implied Addressing:**

#### **Format:**

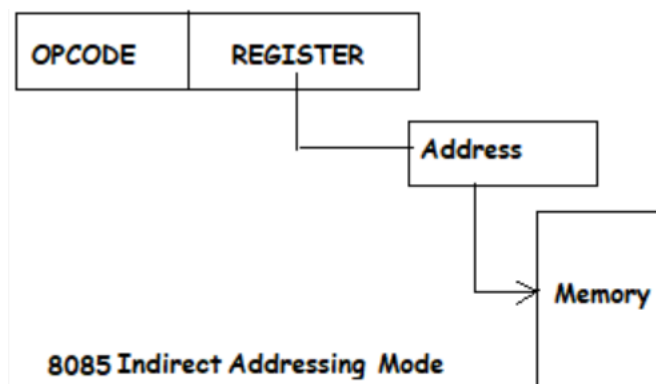


In implied addressing mode, the instruction itself specifies the data to be operated.

**EX.** CMA - Complement the content of accumulator;  
RAL

### **5. Register Indirect Addressing:**

#### **Format:**



In register indirect addressing mode, the instruction specifies the name of the register in which the address of the data is available. Here the data will be in memory and the address will be in the register pair.

**EX.** MOV A, M - The memory data addressed by H L pair is moved to A register.

**UNIT-II****8085 INSTRUCTION SET & PROGRAMMING USING 8085****2.1 INSTRUCTION CYCLE**

The time a microprocessor needs to fetch and execute one entire instruction is known as an instruction cycle. There are typically four stages of an instruction cycle that the CPU carries out.

1. **Fetching the instruction:** The next instruction is fetched from the memory address that is currently stored in the program counter (PC) and stored in the instruction register (IR). At the end of the fetch operation, the PC points to the next instruction that will be read at the next cycle.
2. **Decode the instruction:** During this cycle the encoded instruction present in the IR (instruction register) is interpreted by the decoder.
3. **Read the effective address:** In case of a memory instruction (direct or indirect) the execution phase will be in the next clock pulse. If the instruction has an indirect address, the effective address is read from main memory and any required data is fetched from main memory to be processed and then placed into data registers.
4. **Execute the instruction:** The control unit of the CPU passes the decoded information as a sequence of control signals to the relevant function units of the CPU to perform the actions required by the instruction such as reading values from registers, passing them to the ALU to perform mathematical or logic functions on them and writing the result back to a register.

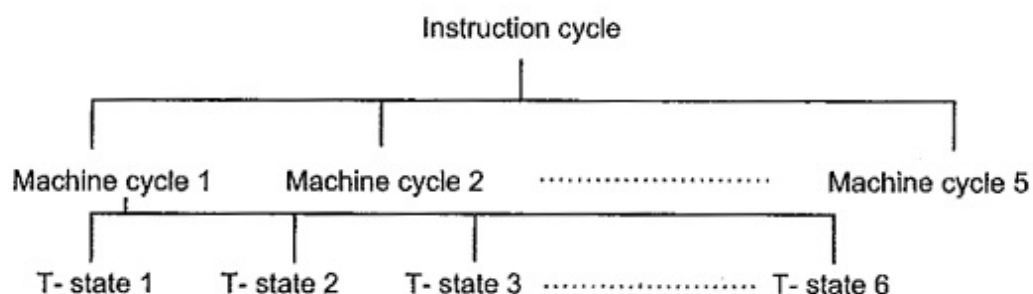


Fig. 2.1 Relation between instruction cycle, machine cycle and T-states

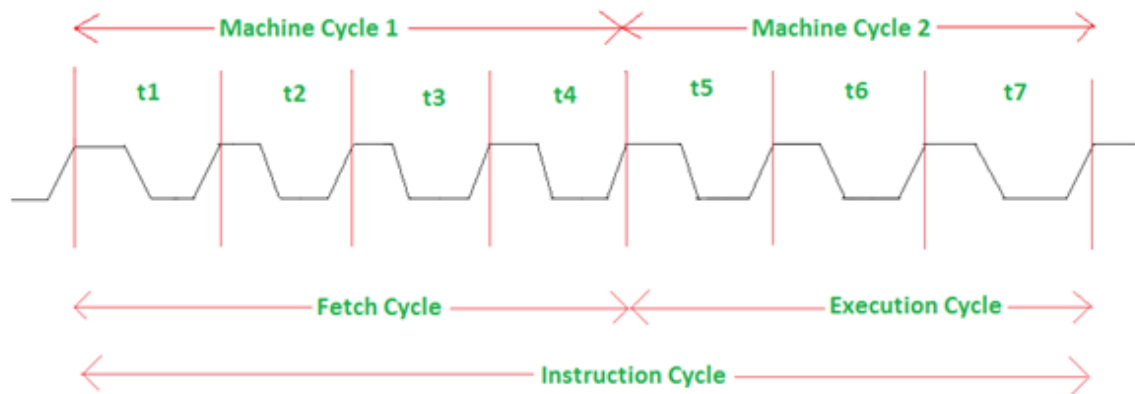


Fig. 2.2 8085 instruction cycle

## 2.2 MACHINE CYCLE

The machine cycles are the basic operations performed by the processor. To execute an instruction, the processor executes one or more machine cycles in a particular sequence. The machine cycles of a processor are also called Processor Cycles. The time required to access the memory or input/output devices is called machine cycle.

The 8085 microprocessor has seven basic machine cycles. These are:

1. Opcode fetch cycle (4T or 6T)
2. Memory read cycle (3T)
3. Memory write cycle (3T)
4. IO read cycle (3T)
5. IO write cycle (3T)
6. Interrupt acknowledge cycle (6T or 12T)
7. Bus idle cycle (2T or 3T)

Each instruction of the 8085 processor consists of one to five machine cycles, i.e., when the 8085 processor executes an instruction; it will execute some of the machine cycles in a specific order.

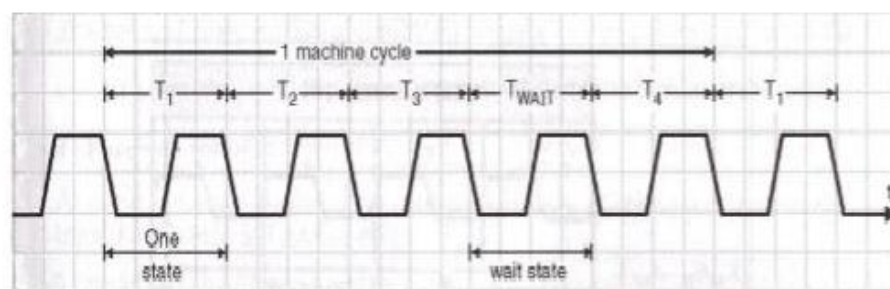
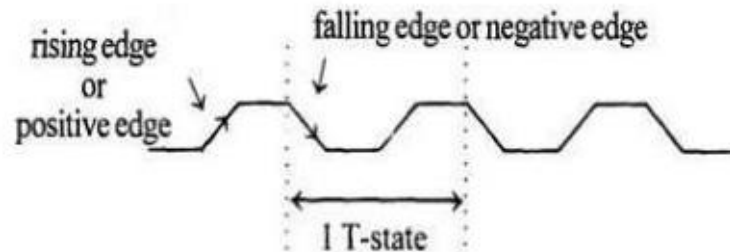


Fig. 2.3 Clock, T-states and machine cycle

### 2.3 T-STATE

One complete cycle of clock is called as T-state as shown in the above figure. The time intervals T1 or T2 are the examples of T-state. A T-state is measured from the falling edge of one clock pulse to the falling edge of the next clock pulse.

*Time period,  $T = 1/f$ ; where  $f = \text{Internal clock frequency}$*

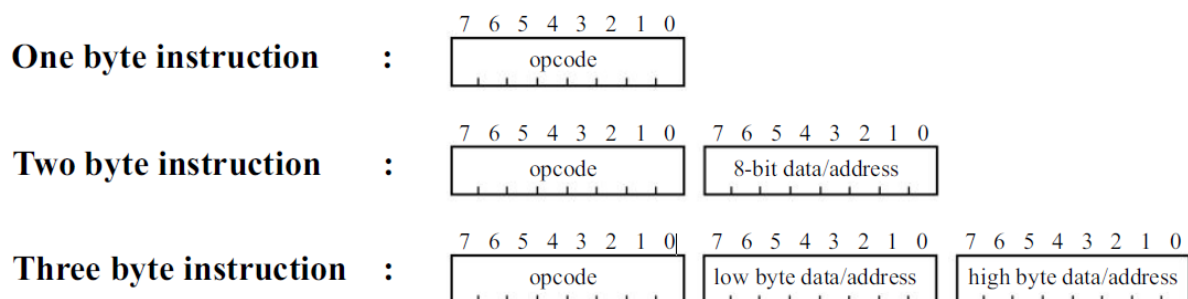


**Fig 2.4 Clock Signal**

### 2.4 8085 INSTRUCTION FORMAT

The 8085 has 74 basic instructions and 246 total instructions. The instruction set of 8085 is defined by the manufacturer INTEL Corporation. Each instruction of 8085 has one-byte opcode. With 8-bit binary code, we can generate 256 different binary codes. In this, 246 codes have been used for opcodes of 8085 instructions.

The size of 8085 instructions can be one byte, two byte or three bytes. The one-byte instruction has an opcode alone and the two-byte instruction has an opcode followed by an eight-bit address or data. The three-byte instruction has an opcode followed by a 16-bit address or data. While storing the three-byte instruction in the memory, the sequence of storage is, opcode first followed by low byte of address or data and then high byte of address or data. The format of 8085 instructions are shown in Fig. 2.5.



**Fig 2.5 Format of 8085 instructions.**



## 2.5 INSTRUCTION SET OF 8085

The 8085-instruction set can be classified into the following five functional headings.

1. **Data Transfer Instructions:** Includes the instructions that moves (copies) data between registers or between memory locations and registers. In all data transfer operations, the content of source register is not altered. Hence the data transfer is copying operation.
2. **Arithmetic Instructions:** Includes the instructions, which performs the addition, subtraction, increment or decrement operations. The flag conditions are altered after execution of an instruction in this group.
3. **Logical Instructions:** The instructions which performs the logical operations like AND, OR, EXCLUSIVE-OR, complement, compare and rotate instructions are grouped under this heading. The flag conditions are altered after execution of an instruction in this group.
4. **Branching Instructions:** The instructions that are used to transfer the program control from one memory location to another memory location are grouped under this heading.
5. **Machine Control Instructions:** Includes the instructions related to interrupts and the instruction used to halt program execution.

### 2.5.1 DATA TRANSFER INSTRUCTIONS

Opcode	Operand	Description
MOV	Rd, Rs M, Rs Rd, M	Copy from source to destination. This instruction copies the contents of the source register into the destination register; the contents of the source register are not altered. If one of the operands is a memory location, its location is specified by the contents of the HL registers. Example: MOV B, C or MOV B, M
MVI	Rd, data M, data	Move immediate 8-bit. The 8-bit data is stored in the destination register or memory. If the operand is a memory location, its location is specified by the contents of the HL registers. Example: MVI B, 57 or MVI M, 57
LDA	16-bit address	Load accumulator. The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator. The contents of the source are not altered. Example: LDA 2034 or LDA XYZ

Opcode	Operand	Description
LDAX	B/D Reg. pair	Load accumulator indirect. The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator. The contents of either the register pair or the memory location are not altered. Example: LDAX B
LXI	Reg. pair, 16-bit data	Load register pair immediate. The instruction loads 16-bit data in the register pair designated in the operand. Example: LXI H, 2034
LHLD	16-bit address	Load H and L registers direct. The instruction copies the contents of the memory location pointed out by the 16-bit address into register L and copies the contents of the next memory location into register H. The contents of source memory locations are not altered. Example: LHLD 2040
STA	16-bit address	Store accumulator direct. The contents of the accumulator are copied into the memory location specified by the operand. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. Example: STA 4350 or STA XYZ
STAX	Reg. pair	Store accumulator indirect. The contents of the accumulator are copied into the memory location specified by the contents of the operand (register pair). The contents of the accumulator are not altered. Example: STAX B
SHLD	16-bit address	Store H and L registers direct. The contents of register L are stored into the memory location specified by the 16-bit address in the operand and the contents of H register are stored into the next memory location by incrementing the operand. The contents of registers HL are not altered. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. Example: SHLD 2470
XCHG	none	Exchange H and L with D and E. The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E. Example: XCHG

Opcode	Operand	Description
SPHL	none	Copy H and L registers to the stack pointer. The instruction loads the contents of the H and L registers into the stack pointer register, the contents of the H register provide the high-order address and the contents of the L register provide the low-order address. The contents of the H and L registers are not altered. Example: SPHL
PUSH	Reg. pair	Push register pair onto stack. The contents of the register pair designated in the operand are copied onto the stack in the following sequence. The stack pointer register is decremented and the contents of the high order register (B, D, H, A) are copied into that location. The stack pointer register is decremented again and the contents of the low-order register (C, E, L, flags) are copied to that location. Example: PUSH B or PUSH A
POP	Reg. pair	Pop off stack to register pair. The contents of the memory location pointed out by the stack pointer register are copied to the low-order register (C, E, L, status flags) of the operand. The stack pointer is incremented by 1 and the contents of that memory location are copied to the high-order register (B, D, H, A) of the operand. The stack pointer register is again incremented by 1. Example: POP H or POP A
IN	8-bit port address	Input data to accumulator from a port with 8-bit address. The contents of the input port designated in the operand are read and loaded into the accumulator. Example: IN 82
OUT	8-bit port address	Output data from accumulator to a port with 8-bit address. The contents of the accumulator are copied into the I/O port specified by the operand. Example: OUT 87

**PUSH rp**  $(SP) \leftarrow (SP) - 1; ((SP)) \leftarrow (rp)H$   
 $(SP) \leftarrow (SP) - 1; ((SP)) \leftarrow (rp)L$

The content of the register pair (rp) is pushed to the stack. After execution of this instruction, the content of the Stack Pointer (SP) will be 02 less than the earlier value. The register pairs can be BC, DE, HL and PSW. No flags are affected.

[PSW (Program Status Word): Accumulator and Flag register together called PSW. Accumulator is high order register and Flag register is low order register.]

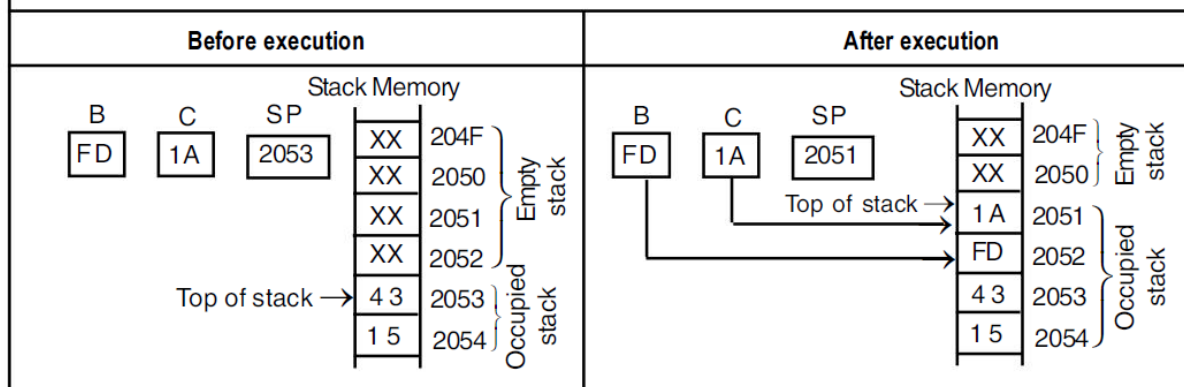
The instruction is executed as follows:

- (i) The content of the SP is decremented by one.
- (ii) The content of the high order register is moved to memory addressed by SP.
- (iii) The content of the SP is decremented by one.
- (iv) The content of the low order register is moved to memory addressed by SP

**PUSH PSW      PUSH B      PUSH D      PUSH H**

**Example : PUSH B**  $(SP) \leftarrow (SP) - 01$   
 $((SP)) \leftarrow (B)$   
 $(SP) \leftarrow (SP) - 01$   
 $((SP)) \leftarrow (C)$

- (i) The content of the SP is decremented by one.
- (ii) The content of the B-register is moved to the memory addressed by the Stack Pointer (SP).
- (iii) Again the content of SP is decremented by one.
- (iv) The content of the C-register is moved to the memory addressed by SP.



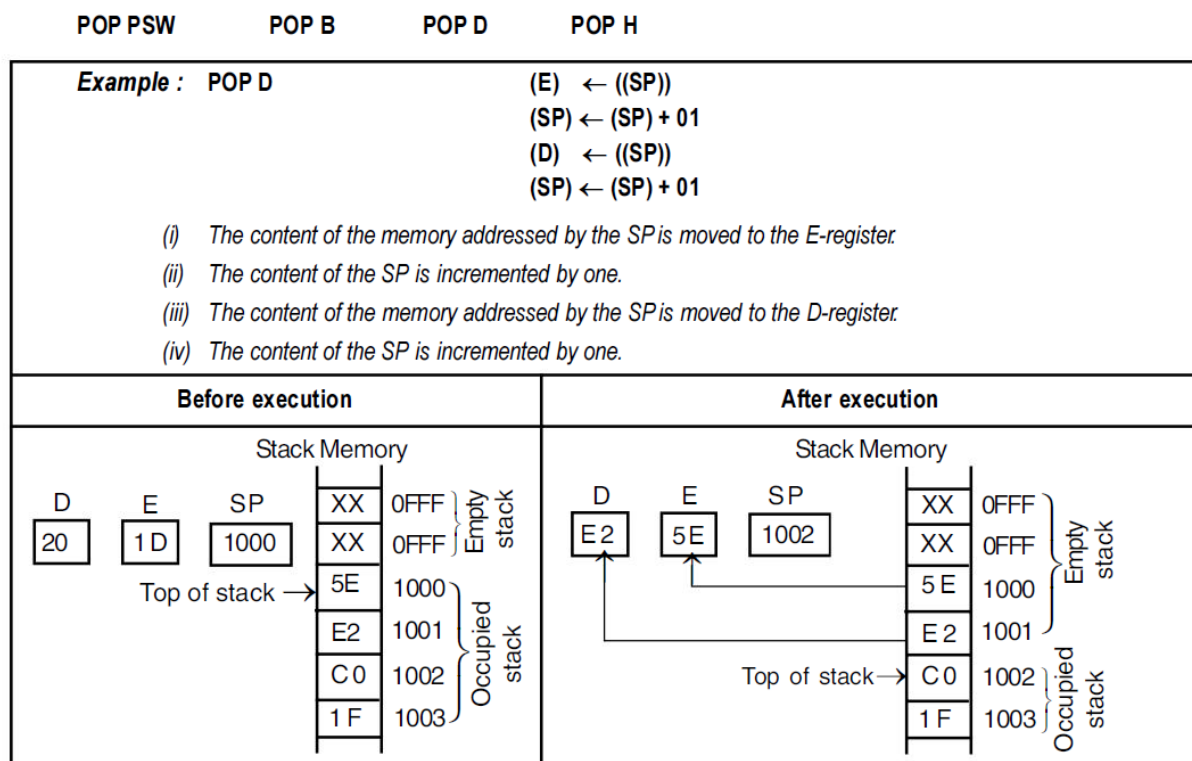
**POP r p** $(rp)L \leftarrow ((SP)); (SP) \leftarrow (SP) + 1$  $(rp)H \leftarrow ((SP)); (SP) \leftarrow (SP) + 1$ 

The content of top of stack memory is moved to the register pair. After execution of this instruction the content of the Stack Pointer (SP) will be 02 greater than the earlier value. The register pairs can be BC, DE, HL and PSW. No flags are affected. [PSW (Program Status Word): Accumulator and Flag register are together called PSW. The accumulator is a high order register and the flag

register is a low order register.]

The pop instruction is executed as follows:

- (i) The content of the memory addressed by the SP is moved to the low order register.
- (ii) The content of the SP is incremented by one.
- (iii) The content of the memory addressed by the SP is moved to the high order register.
- (iv) The content of the SP is incremented by one.



**2.5.2 ARITHMETIC INSTRUCTIONS**

Opcode	Operand	Description
ADD	R M	Add register or memory to accumulator. The contents of the operand (register or memory) are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. Example: ADD B or ADD M
ADC	R M	Add register to accumulator with carry. The contents of the operand (register or memory) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. Example: ADC B or ADC M
ADI	8-bit data	Add immediate to accumulator. The 8-bit data (operand) is added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. Example: ADI 45
ACI	8-bit data	Add immediate to accumulator with carry. The 8-bit data (operand) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. Example: ACI 45
DAD	Reg. pair	Add register pair to H and L registers. The 16-bit contents of the specified register pair are added to the contents of the HL register and the sum is stored in the HL register. The contents of the source register pair are not altered. If the result is larger than 16 bits, the CY flag is set. No other flags are affected. Example: DAD H
SUB	R M	Subtract register or memory from accumulator. The contents of the operand (register or memory) are subtracted from the contents of the accumulator, and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction. Example: SUB B or SUB M

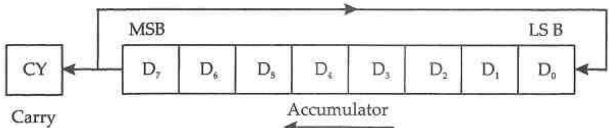
Opcode	Operand	Description
SBB	R M	Subtract source and borrow from accumulator. The contents of the operand (register or memory) and the Borrow flag are subtracted from the contents of the accumulator and the result is placed in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction. Example: SBB B or SBB M
SUI	8-bit data	Subtract immediate from accumulator. The 8-bit data (operand) is subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtraction. Example: SUI 45
SBI	8-bit data	Subtract immediate from accumulator with borrow. The 8-bit data (operand) and the Borrow flag are subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtraction. Example: SBI 45
INR	R M	Increment register or memory by 1. The contents of the designated register or memory) are incremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers. Example: INR B or INR M
INX	R	Increment register pair by 1. The contents of the designated register pair are incremented by 1 and the result is stored in the same place. Example: INX H
DCR	R M	Decrement register or memory by 1. The contents of the designated register or memory are decremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers. Example: DCR B or DCR M
DCX	R	Decrement register pair by 1. The contents of the designated register pair are decremented by 1 and the result is stored in the same place. Example: DCX H

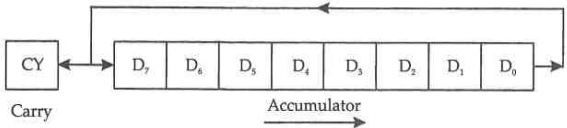
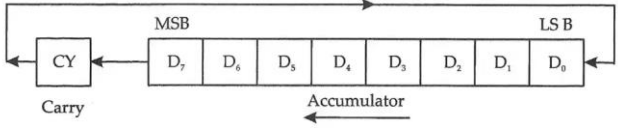
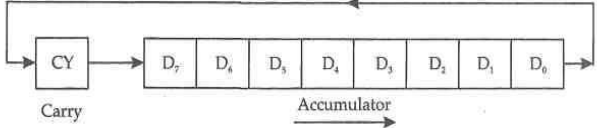
DAA	None	<p>Decimal adjust accumulator.</p> <p>The contents of the accumulator are changed from a binary value to two 4-bit binary coded decimal (BCD) digits. This is the only instruction that uses the auxiliary flag to perform the binary to BCD conversion, and the conversion procedure is described below. S, Z, AC, P, CY flags are altered to reflect the results of the operation.</p> <p>If the value of the low-order 4-bits in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low-order four bits.</p> <p>If the value of the high-order 4-bits in the accumulator is greater than 9 or if the Carry flag is set, the instruction adds to the high-order four bits.</p> <p>Example: DAA</p>
-----	------	---

### 2.5.3 LOGICAL INSTRUCTIONS

Opcode	Operand	Description
CMP	R M	<p>Compare register or memory with accumulator.</p> <p>The contents of the operand (register or memory) are compared with the contents of the accumulator. Both contents are preserved. The result of the comparison is shown by setting the flags of the PSW as follows:</p> <p>if (A) &lt; (reg/mem): carry flag is set, s=1            if (A) = (reg/mem): zero flag is set, s=0            if (A) &gt; (reg/mem): carry and zero flags are reset, s=0</p> <p>Example: CMP B or CMP M</p>
CPI	8-bit data	<p>Compare immediate with accumulator.</p> <p>The second byte (8-bit data) is compared with the contents of the accumulator. The values being compared remain unchanged. The result of the comparison is shown by setting the flags of the PSW as follows:</p> <p>if (A) &lt; data: carry flag is set, s=1            if (A) = data: zero flag is set, s=0            if (A) &gt; data: carry and zero flags are reset, s=0</p> <p>Example: CPI 89</p>
ANA	R M	<p>Logical AND register or memory with accumulator.</p> <p>The contents of the accumulator are logically ANDed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set. Example: ANA B or ANA M</p>



Opcode	Operand	Description
ANI	8-bit data	Logical AND immediate with accumulator. The contents of the accumulator are logically ANDed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set. Example: ANI 86
XRA	R M	Exclusive OR register or memory with accumulator. The contents of the accumulator are Exclusive ORed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. Example: XRA B or XRA M
XRI	8-bit data	Exclusive OR immediate with accumulator. The contents of the accumulator are Exclusive ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. Example: XRI 86
ORA	R M	Logical OR register or memory with accumulator. The contents of the accumulator are logically ORed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. Example: ORA B or ORA M
ORI	8-bit data	Logical OR immediate with accumulator. The contents of the accumulator are logically ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. Example: ORI 86
RLC	None	Rotate accumulator left. Each binary bit of the accumulator is rotated left by one position. Bit D <sub>7</sub> is placed in the position of D <sub>0</sub> as well as in the Carry flag. CY is modified according to bit D <sub>7</sub> . S, Z, P, AC are not affected. Example: RLC 

Opcode	Operand	Description
RRC	None	<p>Rotate accumulator right. Each binary bit of the accumulator is rotated right by one position. Bit D0 is placed in the position of D7 as well as in the Carry flag. CY is modified according to bit D0. S, Z, P, AC are not affected. Example: RRC</p> 
RAL	None	<p>Rotate accumulator left through carry. Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0. CY is modified according to bit D7. S, Z, P, AC are not affected. Example: RAL</p> 
RAR	None	<p>Rotate accumulator right through carry. Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7. CY is modified according to bit D0. S, Z, P, AC are not affected. Example: RAR</p> 
CMA	None	<p>Complement accumulator. The contents of the accumulator are complemented. No flags are affected. Example: CMA</p>
CMC	None	<p>Complement carry. The Carry flag is complemented. No other flags are affected. Example: CMC</p>
STC	None	<p>Set Carry. The Carry flag is set to 1. No other flags are affected. Example: STC</p>

**2.5.4 BRANCHING INSTRUCTIONS**

Opcode	Operand	Description	
JMP	16-bit address	Jump unconditionally. The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. Example: JMP 2034 or JMP XYZ	
Opcode:	16-bit address	Jump conditionally. The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW as described below. Example: JZ 2034 or JZ XYZ	
	Opcode	Description	Flag Status
	JC	Jump on Carry	CY = 1
	JNC	Jump on no Carry	CY = 0
	JP	Jump on positive	S = 0
	JM	Jump on minus	S = 1
	JZ	Jump on zero	Z = 1
	JNZ	Jump on no zero	Z = 0
	JPE	Jump on parity even	P = 1
	JPO	Jump on parity odd	P = 0
CALL	16-bit address	Unconditional subroutine call. The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. Before the transfer, the address of the next instruction after CALL (the contents of the program counter) is pushed onto the stack. Example: CALL 2034 or CALL XYZ	
Opcode:	16-bit address	The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW as described below. Before the transfer, the address of the next instruction after the call (the contents of the program counter) is pushed onto the stack. Example: CZ 2034 or CZ XYZ	
	Opcode	Description	Flag Status
	CC	Call on Carry	CY = 1
	CNC	Call on no Carry	CY = 0
	CP	Call on positive	S = 0
	CM	Call on minus	S = 1
	CZ	Call on zero	Z = 1
	CNZ	Call on no zero	Z = 0
	CPE	Call on parity even	P = 1
	CPO	Call on parity odd	P = 0

Opcode	Operand	Description																												
RET	None	Return from subroutine unconditionally. The program sequence is transferred from the subroutine to the calling program. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address. Example: RET																												
Opcode:	None	Return from subroutine conditionally. The program sequence is transferred from the subroutine to the calling program based on the specified flag of the PSW as described below. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address. Example: RZ																												
	Opcode	Description                      Flag Status																												
	RC	Return on Carry                      CY = 1																												
	RNC	Return on no Carry                      CY = 0																												
	RP	Return on positive                      S = 0																												
	RM	Return on minus                      S = 1																												
	RZ	Return on zero                      Z = 1																												
	RNZ	Return on no zero                      Z = 0																												
	RPE	Return on parity even                      P = 1																												
	RPO	Return on parity odd                      P = 0																												
RST	0-7	Restart. The RST instruction is equivalent to a 1-byte call instruction to one of eight memory locations depending upon the number. The instructions are generally used in conjunction with interrupts and inserted using external hardware. However, these can be used as software instructions in a program to transfer program execution to one of the eight locations. The addresses are: <table border="0" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Instruction</th> <th>Restart Address</th> </tr> </thead> <tbody> <tr><td>RST 0</td><td>0000H</td></tr> <tr><td>RST 1</td><td>0008H</td></tr> <tr><td>RST 2</td><td>0010H</td></tr> <tr><td>RST 3</td><td>0018H</td></tr> <tr><td>RST 4</td><td>0020H</td></tr> <tr><td>RST 5</td><td>0028H</td></tr> <tr><td>RST 6</td><td>0030H</td></tr> <tr><td>RST 7</td><td>0038H</td></tr> </tbody> </table> The 8085 has four additional interrupts and these interrupts generate RST instructions internally and thus do not require any external hardware. These instructions and their Restart addresses are: <table border="0" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Interrupt</th> <th>Restart Address</th> </tr> </thead> <tbody> <tr><td>TRAP</td><td>0024H</td></tr> <tr><td>RST 5.5</td><td>002CH</td></tr> <tr><td>RST 6.5</td><td>0034H</td></tr> <tr><td>RST 7.5</td><td>003CH</td></tr> </tbody> </table>	Instruction	Restart Address	RST 0	0000H	RST 1	0008H	RST 2	0010H	RST 3	0018H	RST 4	0020H	RST 5	0028H	RST 6	0030H	RST 7	0038H	Interrupt	Restart Address	TRAP	0024H	RST 5.5	002CH	RST 6.5	0034H	RST 7.5	003CH
Instruction	Restart Address																													
RST 0	0000H																													
RST 1	0008H																													
RST 2	0010H																													
RST 3	0018H																													
RST 4	0020H																													
RST 5	0028H																													
RST 6	0030H																													
RST 7	0038H																													
Interrupt	Restart Address																													
TRAP	0024H																													
RST 5.5	002CH																													
RST 6.5	0034H																													
RST 7.5	003CH																													

**2.5.5 MACHINE CONTROL INSTRUCTIONS**

<b>Opcode</b>	<b>Operand</b>	<b>Description</b>
NOP	none	No operation. No operation is performed. The instruction is fetched and decoded. However, no operation is executed. Example: NOP
HLT	none	Halt and enter wait state. The CPU finishes executing the current instruction and halts any further execution. An interrupt or reset is necessary to exit from the halt state. Example: HLT
DI	none	Disable interrupts. The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled. No flags are affected. Example: DI
EI	none	Enable interrupts. The interrupt enable flip-flop is set and all interrupts are enabled. No flags are affected. After a system reset or the acknowledgement of an interrupt, the interrupt enable flip-flop is reset, thus disabling the interrupts. This instruction is necessary to re-enable the interrupts (except TRAP). Example: EI

**Program-1: Addition of two 8-bit numbers.**

Address	Label	Mnemonics		Comments
		Opcode	Operands	
8100		LXI	H,8200H	Set pointer for data.
8103		MVI	C,00H	Clear C-register to account for carry.
8105		MOV	A,M	Get 1st data in A-register.
8106		INX	H	Increment memory to point 2 <sup>nd</sup> data
8107		ADD	M	Add 2nd data which is available in memory to A
8108		JNC	AHEAD	If CF=0, go to AHEAD. If CF=1, increment C-register.
810B		INR	C	Increment register C by one
810C	AHEAD	INX	H	Increment memory location by one
810D		MOV	M, A	Save the sum in memory.
810E		INX	H	Increment memory location by one
810F		MOV	M, C	Save the carry in memory.
8110		HLT		Halt program execution.

Input		Output	
Address	Data	Address	Data
8200H	E2H	8202H	27H
8201H	45H	8203H	01H

**Program-2: Subtraction of two 8-bit numbers.**

Address	Label	Mnemonics		Comments
		Opcode	Operands	
8100		LDA	8201H	Get the subtrahend in B-register.
8103		MOV	B, A	
8104		LDA	8200H	Get the minuend in A-register.
8107		MVI	C, 00H	Clear C-register to account for sign.
8109		SUB	B	Get the difference in A-register.
810A		JNC	AHEAD	If CF =0, then go to AHEAD. If CF=1, then increment C-register.
810D		INR	C	
810E		CMA		Get 2's complement of difference (result) in A-register.
810F		ADI	01H	
8111	AHEAD	STA	8202H	Store the result in memory.
8114		MOV	A, C	Store the sign bit in memory.
8115		STA	8203H	
8118		HLT		Halt the program execution.

Input		Output	
Address	Data	Address	Data
8200H	5EH	8202H	2AH
8201H	34H	8203H	00H

**Program-3: Multiplication of two 8-bit numbers.**

Address	Label	Mnemonics		Comments
		Opcode	Operands	
8000		LXI	H, 8500H	Load HL pair with immediate data 8500h
8003		MVI	C,00H	Initialize register C = 00h
8005		XRA	A	Clear accumulator
8006		MOV	B, M	Get the first number into register B
8007		INX	H	Increment Memory address (HL pair)
8008		MOV	D, M	Get the second number into register D
8009	REPT	ADD	D	Add the content of register D with A
800A		JNC	AHEAD	If carry = 0, go to AHEAD
800D		INR	C	If carry =1, increment register C
800E	AHEAD	DCR	B	Decrement register B
800F		JNZ	REPT	Repeat addition until register B is zero
8012		INX	H	Increment memory address
8013		MOV	M, A	Store accumulator into memory address
8014		INX	H	Increment memory address
8015		MOV	M, C	Store register C into memory address
8016		HLT		Stop execution

Input		Output	
Address	Data	Address	Data
8500	02H	8502	0AH
8501	05H	8503	00H



**Program-4: Division of two 8-bit numbers.**

Address	Label	Mnemonics		Comments
		Opcode	Operands	
8000		LDA	8501H	Load accumulator with the data in 8501
8003		MOV	B, A	Move divisor into register B
8004		LDA	8500H	Load accumulator with the data in 8502 (Dividend)
8007		MVI	C, 00H	Clear register C for quotient
8009	AGAIN	CMP	B	Compare register B with A
800A		JC	STORE	If divisor is greater than dividend, go to STORE
800D		SUB	B	Subtract divisor from dividend
800E		INR	C	Increment register C
800F		JMP	AGAIN	Move to AGAIN
8012	STORE	STA	8503H	Store the content of the accumulator (remainder) into 8503 location
8015		MOV	A, C	Move content of register C to A
8016		STA	8502	Store accumulator (quotient) into location 8502
8019		HLT		Stop execution

Input		Output	
Address	Data	Address	Data
8500	C9 H (Dividend)	8502	14 H (Quotient)
8501	0A H (Divisor)	8503	01 H (Remainder)

**Program-5: Smallest number in an array.**

Address	Label	Mnemonics		Comments
		Opcode	Operands	
8000		LXI	H,8500	Load HL pair with 8500h
8003		MOV	B, M	Set count for number of elements in array
8004		INX	H	Increment memory address
8005		MOV	A, M	Select 1 <sup>st</sup> element of array as smallest number
8006		DCR	B	Decrement count
8007	LOOP	INX	H	Increment memory address
8008		CMP	M	Compare element of array with current smallest number
8009		JC	AHEAD	If carry = 1, go to AHEAD
800C		MOV	A, M	If carry = 0, the content of memory is smaller than A, hence make memory element as smallest number
800D	AHEAD	DCR	B	Decrement count
800E		JNZ	LOOP	If it is not zero, go to LOOP
8011		STA	8600H	Store accumulator in 8600 location
8014		HLT		Stop execution

Input		Output	
Address	Data	Address	Data
8500	07h (Count)	8600	03h (Smallest number)
8501	12h		
8502	22h		
8503	42h		
8504	03h		
8505	A1h		
8506	2Bh		
8507	FF h		

**Program-6: Largest number in an array.**

Address	Label	Mnemonics		Comments
		Opcode	Operands	
8000		LXI	H,8500	Load HL pair with 8500h
8003		MOV	B, M	Set count for number of elements in array
8004		INX	H	Increment memory address
8005		MOV	A, M	Select 1 <sup>st</sup> element of array as smallest number
8006		DCR	B	Decrement count
8007	LOOP	INX	H	Increment memory address
8008		CMP	M	Compare element of array with current smallest number
8009		JNC	AHEAD	If carry = 0, go to AHEAD
800C		MOV	A, M	If carry = 1, the content of memory is greater than A, hence make memory element as largest number
800D	AHEAD	DCR	B	Decrement count
800E		JNZ	LOOP	If it is not zero, go to LOOP
8011		STA	8600H	Store accumulator in 8600 location
8014		HLT		Stop execution

Input		Output	
Address	Data	Address	Data
8500	07h (Count)	8600	FF h (Largest number)
8501	12h		
8502	22h		
8503	42h		
8504	03h		
8505	A1h		
8506	2Bh		
8507	FF h		

**Program-7: Sorting an Array in Ascending Order**

Address	Label	Mnemonics		Comments
		Opcode	Operands	
8100		LDA	8200H	Load the count value in A-register.
8103		MOV	B, A	Move count value to Reg. B
8104		DCR	B	Set count for N-1 repetitions of N-1 comparisons.
8105	LOOP2	LXI	H,8200H	Set pointer for array.
8108		MOV	C, M	Set count for N-1 comparisons.
8109		DCR	C	Decrement Reg. C content by one.
810A		INX	H	Increment pointer.
810B	LOOP1	MOV	A, M	Get one data of array in A.
810C		INX	H	Increment pointer.
810D		CMP	M	Compare next data with A-register.
810E		JC	AHEAD	If content of A is less than memory then go to AHEAD.
8111		MOV	D,M	If the content of A is greater than the content of memory then exchange the content of memory pointed by HL and previous location.
8112		MOV	M,A	
8113		DCX	H	
8114		MOV	M, D	
8115		INX	H	
8116	AHEAD	DCR	C	Repeat comparisons until C count is zero.
8117		JNZ	LOOP1	
811A		DCR	B	Repeat N-1 comparisons until B count is zero.
811B		JNZ	LOOP2	
811E		HLT		Halt program execution.

Input		Output	
Address	Data	Address	Data
8200H	04H	8200H	04H
8201H	92H	8201H	02H
8202H	2AH	8202H	12H
8203H	12H	8203H	2AH
8204H	02H	8204H	92H

**Program-8: Sorting an Array in Descending Order**

Address	Label	Mnemonics		Comments
		Opcode	Operands	
8100		LDA	8200H	Load the count value in A-register.
8103		MOV	B, A	Move count value to Reg. B
8104		DCR	B	Set count for N-1 repetitions of N-1 comparisons.
8105	LOOP2	LXI	H,8200H	Set pointer for array.
8108		MOV	C, M	Set count for N-1 comparisons.
8109		DCR	C	Decrement Reg. C content by one.
810A		INX	H	Increment pointer.
810B	LOOP1	MOV	A, M	Get one data of array in A.
810C		INX	H	Increment pointer.
810D		CMP	M	Compare next data with A-register.
810E		JC	AHEAD	If content of A is greater than memory then go to AHEAD.
8111		MOV	D,M	If the content of A is less than the content of memory then exchange the content of memory pointed by HL and previous location.
8112		MOV	M,A	
8113		DCX	H	
8114		MOV	M, D	
8115		INX	H	
8116	AHEAD	DCR	C	Repeat comparisons until C count is zero.
8117		JNZ	LOOP1	
811A		DCR	B	Repeat N-1 comparisons until B count is zero.
811B		JNZ	LOOP2	
811E		HLT		Halt program execution.

Input		Output	
Address	Data	Address	Data
8200H	04H	8200H	04H
8201H	21H	8201H	5FH
8202H	3DH	8202H	3DH
8203H	02H	8203H	21H
8204H	5FH	8204H	02H

## UNIT-III

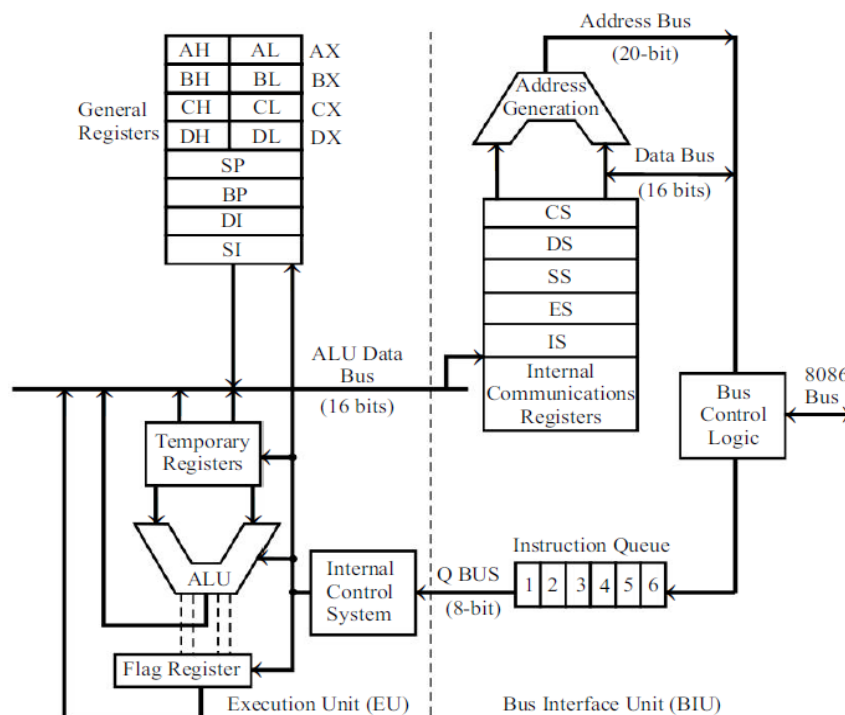
### 8086 MICROPROCESSORS

#### 3.1 8086 MICROPROCESSOR ARCHITECTURE

##### Features:

1. The INTEL 8086 is the first 16-bit processor released by INTEL in the year 1978. The 8086 is designed using the HMOS technology and contains approximately 29,000 transistors. The 8086 is packed in a 40-pin DIP and requires a single 5-V supply.
2. Its operating frequency is 5 MHz.
3. It has 16-bit data bus and 20-bit address bus. It has fourteen 16-bit registers.
4. The total memory addressing capacity is 1MB (external).
5. The 8086 has a pipelined architecture. Pipelining improves the performance of the processor so that operation is faster. 8086 uses two stage of pipelining. First is Fetch Stage and the second is Execute Stage.
6. Fetch stage can pre-fetch up to 6 bytes of instructions and stores them in the queue.
7. It supports two modes of operation, i.e. Maximum mode and Minimum mode. Maximum mode is suitable for system having multiple processors and Minimum mode is suitable for system having a single processor.

##### Architecture:



**Fig. 3.1 Internal architecture of 8086**

1. The 8086 has a pipelined architecture. In pipelined architecture the processor will have a number of functional units and the execution time of the functional units are overlapped. Each functional unit works independently most of the time.
2. The simplified block diagram of the internal architecture of an 8086 is shown in Fig. 3.1. The architecture of the 8086 can be internally divided into two separate functional units: Bus Interface Unit (BIU) and Execution Unit (EU).
3. The BIU fetches instructions, reads data from memory and IO ports and writes data to memory and IO ports. The EU executes instructions that have already been fetched by the BIU. The BIU and EU function independently.
4. **Memory segmentation:** To increase execution speed and fetching speed, 8086 segments the memory. Its 20-bit address bus can address 1MB of memory, it segments it into 16 64kB segments. 8086 works only with four 64KB segments within the whole 1MB memory.

### **The Bus Interface Unit (BIU):**

BIU performs the following functions-

1. It generates the 20-bit physical address for memory access.
2. It fetches instructions from the memory.
3. It transfers data to and from the memory and I/O.
4. Maintains the 6-byte prefetch instruction queue (supports pipelining).

The BIU contains segment registers, an instruction pointer, an instruction queue, an address generation unit and a bus control unit.

1. The **instruction queue** is a FIFO (First-In-First-Out) group of registers. The size of the queue is 6 bytes. The BIU fetches the instruction code from memory and stores it in queue. The EU fetches the instruction codes from the queue.
2. **Segment registers:** The BIU has four 16-bit segment registers: Code Segment (CS) register, Data Segment (DS) register, Stack Segment (SS) register and Extra Segment (ES) register. The 4-segment registers are used to hold four segment base addresses.
3. The dedicated **address generation unit** generates a 20-bit physical address from the segment base and an offset or effective address. The segment base address is logically shifted left four times and added to the offset. [logically shifting left four times is equal to multiplying it by  $16_{10}$ .]

4. The address for fetching instruction codes is generated by logically shifting the content of the CS to the left four times and then adding it to the content of the IP (Instruction Pointer). The IP holds the offset address of the program codes.
5. The address of the next instruction is calculated as  $CS \times 10H + IP$ .

**Example:**

CS = 4321H, IP = 1000H then

$CS \times 10H = 43210H + \text{offset} = 44210H$  (Here Offset = Instruction Pointer (IP))

6. The data address is computed by using the content of the DS or ES as the base address and an offset or effective address specified by the instruction. The stack address is computed by using the content of the SS as the base address and the content of the SP (Stack Pointer) as the offset address or effective address.
7. The **bus control logic** of the BIU generates all the bus control signals such as read and write signals for memory and IO.

**The Execution Unit (EU):**

The EU consists of the ALU, the flag register and the general purpose registers. The EU decodes and executes the instructions. A decoder in the EU control system translates the instructions.

The EU has a 16-bit ALU to perform arithmetic and logical operations. The EU has eight numbers of 16-bit general purpose registers. They are AX, BX, CX, DX, SP, BP, SI and DI. Some of the 16-bit registers can also be used as two numbers of 8-bit registers as given below:

AX - can be used as AH and AL;      CX - can be used as CH and CL

BX - can be used as BH and BL;      DX - can be used as DH and DL

The general purpose registers can be used for data storage when they are not involved in any special functions assigned to them. These registers are named after special functions carried out by each one of them as given in the following Table.

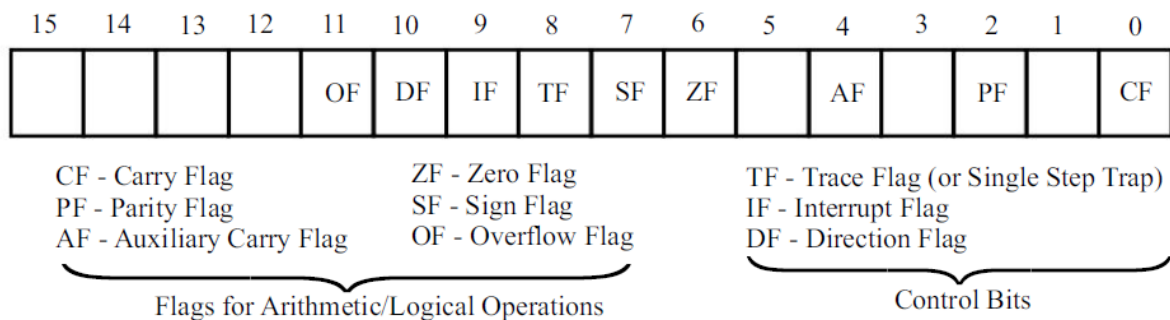
Register	Name of the register	Special function
AX	16-bit accumulator	Stores the 16-bit results of certain arithmetic and logical operations.
AL	8-bit accumulator	Stores the 8-bit results of certain arithmetic and logical operations.
BX	Base register	Used to hold the base value in base addressing mode to access memory data.
CX	Count register	Used to hold the count value in SHIFT, ROTATE and LOOP instructions.



Register	Name of the register	Special function
DX	Data register	Used to hold data for multiplication and division operations.
SP	Stack pointer	Used to hold the offset address of top of the stack memory.
BP	Base pointer	Used to hold the base value in base addressing using the stack segment register to access data from the stack memory.
SI	Source index	Used to hold the index value of the source operand (data) for string instructions.
DI	Destination index	Used to hold the index value of the destination operand (data) for string instructions.

### 8086 Flag Register

The size of an 8086 flag register is 16 bits and in this nine bits are defined as flags. The six flags (status flags) are used to indicate the status of the result of the arithmetic or logical operations. Three flags are used to control the processor operation and so they are also called control bits (control flags). The various flags of an 8086 processor and their bit position in flag register are shown in Fig. 3.2.



**Fig. 3.2 Bit positions of various flags in the flag register of 8086**

1. The Carry Flag (CF) is set if there is a carry from the addition or borrow from the subtraction.
2. Auxiliary carry Flag (AF) is set if there is a carry from low nibble to high nibble of the low order 8-bit of a 16-bit number.
3. The Overflow Flag (OF) is set to one if there is an arithmetic overflow, that is, if the size of the result exceeds the capacity of the destination location.
4. Sign Flag (SF) is set to one if the most significant bit of the result is one and SF is cleared to zero for non-negative result.

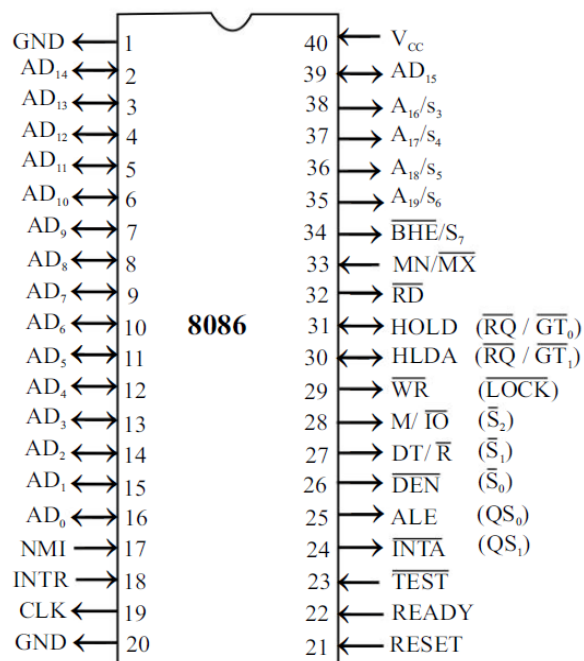
5. The Parity Flag (PF) is set to one if the result has even parity and PF is cleared to zero for odd parity of the result.
6. The Zero Flag (ZF) is set to one if the result is zero and ZF is cleared to zero for a non-zero result.

The three control bits in the flag register can be set or reset by the programmer.

1. The Direction Flag (DF) is set to one for auto-decrement and reset to zero for auto-increment of the SI and DI registers during string data accessing.
2. Setting Interrupt Flag (IF) to one causes the 8086 to recognize the external maskable interrupts, and clearing IF to zero disables the interrupts.
3. Setting Trace Flag (TF) to one, places the 8086 in the single step mode. In this mode the 8086 generates an internal interrupt after execution of each instruction. The single stepping is used for debugging a program.

### 3.2 8086 MICROPROCESSOR PIN DIAGRAM

The 8086 pins and signals are shown in Fig. 3.3. The 8086 is a 40-pin IC and all the 8086 pins are TTL compatible. The signal assigned to pins 24 to 31 is different for minimum and maximum mode of operation. The signal assigned to all the other pins are common for minimum and maximum mode of operation



Note : Signals shown in parenthesis are maximum mode signals.

Fig. 3.3: 8086 pin assignments

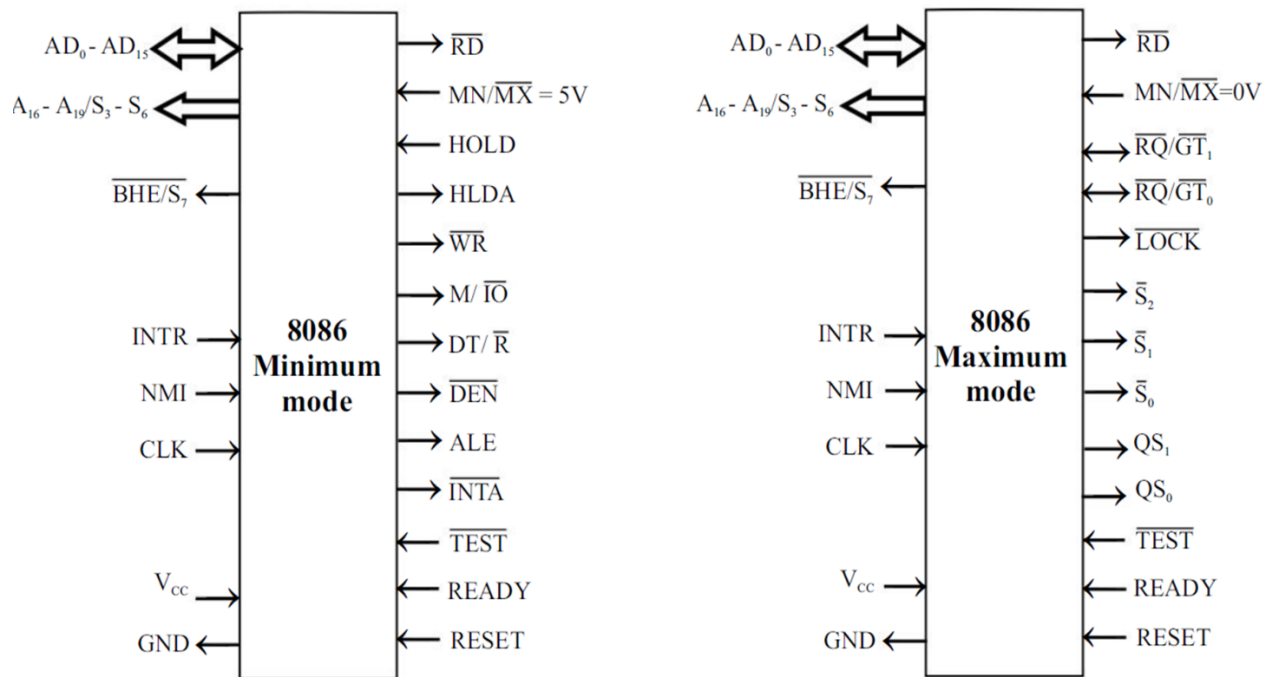


Fig. 3.4: 8086 Minimum and maximum mode signals

**Common Signals:**

Name	Description/Function	Type
$AD_{15} - AD_0$	Address/Data	Bidirectional, Tristate
$A_{19}/S_6 - A_{16}/S_3$	Address/Status	Output, Tristate
$\overline{BHE}/S_7$	Bus high enable/Status	Output, Tristate
$MN/\overline{MX}$	Minimum/Maximum mode control	Input
$\overline{RD}$	Read control	Output, Tristate
$\overline{TEST}$	Wait on test control	Input
READY	Wait state control	Input
RESET	System reset	Input
NMI	Non-maskable interrupt request	Input
INTR	Interrupt request	Input
CLK	System clock	Input
$V_{cc}$	+5-V	Power supply input
GND	Ground	Power supply ground

**AD<sub>15</sub> - AD<sub>0</sub>**

Bidirectional address/data lines. These are low order address bus. They are multiplexed with data. When these lines are used to transmit memory address, the symbol A is used instead of AD, for example,  $A_0 - A_{15}$ .

**A<sub>19</sub> /S<sub>6</sub> - A<sub>16</sub> /S<sub>3</sub>**

High order address bus. These are multiplexed with status signals. The status signals on S<sub>3</sub> and S<sub>4</sub> specifies the segment register used for calculating the physical address.

Status signal		Segment register
S <sub>4</sub>	S <sub>3</sub>	
0	0	Extra segment
0	1	Stack segment
1	0	Code or no segment
1	1	Data segment

**BHE /S<sub>7</sub>**

Bus High Enable/Status. During T1, it is low. It enables the data onto the most significant half of data bus, D8-D15. 8-bit device connected to upper half of the data bus use BHE signal. It is multiplexed with status signal S<sub>7</sub>. S<sub>7</sub> signal is available during T3 and T4.

 **$\overline{\text{RD}}$** 

When the processor reads from memory or an IO location it asserts RD low.

**MN/  $\overline{\text{MX}}$** 

Minimum / Maximum mode control signal, use to set the mode of operation of 8086 microprocessors. 8086 will operate in minimum mode when this signal is at HIGH (V<sub>cc</sub>), it is operated in maximum mode when this signal is at LOW (GND).

**TEST**

The TEST input is tested by the WAIT instruction. The 8086 will enter a wait state after execution of the WAIT instruction, and it will resume execution only when TEST is made low by an external hardware.

**INTR**

INTR is the maskable interrupt and INTR must be held high until it is recognized to generate an interrupt signal.

**NMI**

NMI is the non-maskable interrupt input activated by a leading edge signal.

**RESET**

RESET is the system reset input signal. For power-ON reset it is held high for 50 microseconds. For reset while working, it is held high for at least four clock cycles. When the processor is resetted, the DS, SS, ES, IP and flag register are cleared, Code Segment (CS) register is initialized to FFFF<sub>H</sub> and queue is emptied. After reset the processor will start fetching instruction from 20-bit physical address FFFF0<sub>H</sub>.

**READY**

READY is an input signal to the processor, used by the memory or IO devices to get extra time for data transfer or to introduce wait states in the bus cycles. Normally READY is tied high.

If the READY is tied low, the 8086 introduces wait states after second T-state of a bus cycle and it will complete the bus cycle only when READY is made high again.

**Minimum Mode Signals [MN / MX = Vcc (logic high)]:**

Name	Description / Function	Type
HOLD	Hold request	Input
HLDA	Hold acknowledge	Output
$\overline{WR}$	Write control	Output, Tristate
$M/\overline{IO}$	Memory/IO control	Output, Tristate
$DT/\overline{R}$	Data transmit/Receive	Output, Tristate
$\overline{DEN}$	Data enable	Output, Tristate
ALE	Address latch enable	Output
$\overline{INTA}$	Interrupt acknowledge	Output

**$DT/\overline{R}$  - [Data Transmit / Receive]** It is an output signal from the processor to control the direction of data flow through the data transceivers.

**$\overline{DEN}$  - (Data Enable)** - It is an output signal from the processor used as output enable for the data transceivers.

**ALE - (Address Latch Enable)** - It is used to de-multiplex the address and data lines using external latches.

**$\overline{WR}$  (Write Control)** – Write control signal used to indicate write operation.

**$M/\overline{IO}$**  - It is used to differentiate memory access and IO access. For IN and OUT instructions it is asserted low. For memory reference instructions it is asserted high.

**INTA** - (Interrupt Acknowledge) - The 8086 output is asserted low on this line to acknowledge when the interrupt request is accepted by the processor.

**HOLD** - It is an input signal to the processor from other bus masters as a request to grant control of the bus. It is usually used by the DMA controller to get control of the bus.

**HLDA** - (Hold Acknowledge) - It is an acknowledge signal by the processor to the master requesting the control of the bus through HOLD. The acknowledge is asserted high when the processor accepts the HOLD. [On accepting the hold the processor drives all the tristate pins to high impedance state and sends an acknowledgement to the device which requested HOLD. On receiving the acknowledgement, the other master will take control of the bus.]

**Maximum Mode Signals [MN / MX = GND (logic Low)]:**

Name	Description/Function	Type
$\overline{RQ/GT}_1, \overline{RQ/GT}_0$	Request/Grant bus access control	Bidirectional
$\overline{LOCK}$	Bus priority lock control	Output, Tristate
$\overline{S}_2, \overline{S}_1, \overline{S}_0$	Bus cycle status	Output, Tristate
$QS_1, QS_0$	Instruction queue status	Output

**$\overline{S}_2, \overline{S}_1, \overline{S}_0$**

These are status signals and they are used by the 8288 bus controller to generate bus timing and control signals. The status signals are decoded as shown in Table.

Status Signal			Machine Cycle
$\overline{S}_2$	$\overline{S}_1$	$\overline{S}_0$	
0	0	0	Interrupt acknowledge
0	0	1	Read IO port
0	1	0	Write IO port
0	1	1	Halt
1	0	0	Code access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive/Inactive

**$\overline{RQ/GT}_0, \overline{RQ/GT}_1$**  – These are the Request/Grant signals used by the other processors requesting the CPU to release the system bus. When the signal is received by CPU, then it sends acknowledgment.  $\overline{RQ/GT}_0$  has a higher priority than  $\overline{RQ/GT}_1$ .

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

**QS1, QS0 - (Queue Status)** - The processor provides the status of queue on these lines. The queue status can be used by the external device to track the internal status of the queue in an 8086.

Queue status		Queue operation
QS <sub>1</sub>	QS <sub>0</sub>	
0	0	No operation
0	1	First byte of an opcode from the queue
1	0	Empty the queue
1	1	Subsequent byte from the queue

### **3.3 ADDRESSING MODES OF 8086**

Every instruction of a program has to operate on a data. The method of specifying the data to be operated by the instruction is called Addressing. The 8086 has 8 addressing modes and they can be classified into following groups.

1. Register addressing
2. Immediate addressing
3. Direct addressing
4. Register indirect addressing
5. Implied addressing
6. Based addressing
7. Indexed addressing
8. Based index addressing

#### **1. Register addressing**

In register addressing, the instruction will specify the name of the register which holds the data to be operated by the instruction.

##### **Examples :**

a) MOV CL,DH                      (CL) ← (DH)

*The content of an 8-bit register DH is moved to another 8-bit register CL.*

b) MOV BX,DX                      (BX) ← (DX)

*The content of an 16-bit register DX is moved to another 16-bit register BX.*

#### **2. Immediate addressing**

In immediate addressing mode, an 8 - bit or 16 - bit data is specified as part of the instruction.





override prefix is employed, the content of the segment register specified in the override prefix will be used for base address calculation instead of DS-register.

The base address is obtained by multiplying the content of the segment register by  $16_{10}$ . The 20-bit physical address of memory is computed by adding the effective address to the base address.

**Examples :**

**a) MOV CX, [BX]**

$$\begin{aligned} \text{EA} &= (\text{BX}) \quad ; \quad \text{BA} = (\text{DS}) \times 16_{10} \quad ; \quad \text{MA} = \text{BA} + \text{EA} \\ (\text{CX}) &\leftarrow (\text{MA}) \quad \text{or} \quad (\text{CL}) \leftarrow (\text{MA}) \\ & \quad \quad \quad (\text{CH}) \leftarrow (\text{MA}+1) \end{aligned}$$

*The content of BX is the Effective Address(EA). The segment Base Address(BA) is computed by multiplying the content of DS by  $16_{10}$ . The Memory Address(MA) is obtained by adding BA and EA.*

*The content of the memory whose address is calculated as explained above is moved to the CL-register, and the content of next memory location is moved to the CH-register.*

**b) MOV AX,[SI]**

$$\begin{aligned} \text{EA} &= (\text{SI}) \quad ; \quad \text{BA} = (\text{DS}) \times 16_{10} \quad ; \quad \text{MA} = \text{BA} + \text{EA} \\ (\text{AX}) &\leftarrow (\text{MA}) \quad \text{or} \quad (\text{AL}) \leftarrow (\text{MA}) \\ & \quad \quad \quad (\text{AH}) \leftarrow (\text{MA} + 1) \end{aligned}$$

*The content of SI is the Effective Address (EA). The segment Base Address(BA) is computed by multiplying the content of DS by  $16_{10}$ . The memory address is obtained by adding BA and EA.*

*The content of memory whose address is calculated as explained above is moved to the AL-register, and the content of the next memory location is moved to the AH-register.*

## **5. Implied addressing**

In implied addressing mode, the instruction itself will specify the data to be operated by the instruction.

**Example :**

**CLC - Clear carry ;  $\text{CF} \leftarrow 0$**

*Execution of this instruction will clear the Carry Flag(CF).*

## **6. Based addressing**

In this addressing mode, BX or BP-register is used to hold a base value for the effective address and a signed 8-bit or unsigned 16-bit displacement will be specified in the instruction. The displacement is added to the base value in BX or BP to obtain the Effective Address (EA). In case of 8-bit displacement, it is sign extended to 16-bit before adding to the base value.

When BX is used to hold the base value for EA, the 20-bit physical address of the memory is calculated by multiplying the content of DS by  $16_{10}$  adding to EA. When BP is used to hold the base value for EA, the 20-bit physical address of memory is calculated by multiplying the content of SS by  $16_{10}$  and adding to EA.

**Example :**

MOV AX, [BX+08H]

$0008_H$        $08_H$  ; EA = (BX) +  $0008_H$

BA = (DS)  $\times$   $16_{10}$  ; MA = BA + EA

(AX)  $\leftarrow$  (MA)    or    (AL)  $\leftarrow$  (MA)

(AH)  $\leftarrow$  (MA+1)

*The effective address is calculated by sign extending the 8-bit displacement given in the instruction to 16-bit and adding to the content of the BX-register. The Base Address(BA) is obtained by multiplying the content of DS by  $16_{10}$ . The Memory Address(MA) is obtained by adding BA and EA.*

*The content of the memory whose address is calculated as explained above is moved to the AL-register, and the content of the next memory is moved to the AH-register.*

**7. Indexed addressing**

In this addressing mode, an SI or DI-register is used to hold an index value for memory data and a signed 8 -bit displacement or unsigned 16 - bit displacement will be specified in the instruction. The displacement is added to the index value in the SI or DI-register to obtain the Effective Address (EA). In case of 8 - bit displacement it is sign extended to 16 - bit before adding to the index value. A 20 - bit memory address is calculated by multiplying the content of the data segment (DS) by  $16_{10}$  and adding to EA.

**Example :**

MOV CX, [SI+0A2H]

$FFA2_H$   $\xleftarrow{\text{sign extend}}$   $A2_H$  ; EA = (SI) +  $FFA2_H$

BA = (DS)  $\times$   $16_{10}$  ; MA = BA + EA

(CX)  $\leftarrow$  (MA)    or    (CL)  $\leftarrow$  (MA)

(CH)  $\leftarrow$  (MA + 1)

*The Effective Address (EA) is calculated by sign extending the 8-bit displacement given in the instruction to 16-bit and adding to the content of the SI-register. The Base Address (BA) is obtained by multiplying the content of DS by  $16_{10}$ . The Memory Address (MA) is obtained by adding BA and EA.*

*The content of memory whose address is calculated as explained above is moved to CL-register and the content of next memory is moved to CH-register.*

### 8. Based index addressing

In this addressing mode, the effective address is given by the sum of the base value, the index value and an 8-bit or 16-bit displacement specified in the instruction. The base value is stored in the BX or BP-register. The index value is stored in the SI or DI-register. In case of 8-bit displacement, it is sign extended to 16-bit before adding to the base value. This type of addressing will be useful in addressing two dimensional arrays where we require two modifiers.

When BX is used to hold the base value for EA, the 20-bit physical address of the memory is calculated by multiplying the content of DS by  $16_{10}$  and adding it to EA.

When BP is used to hold the base value for EA, the 20-bit physical address of the memory is obtained by multiplying the content of the SS-register by  $16_{10}$  and adding it to EA.

#### **Example :**

**MOV DX, [BX+SI+0AH]**

$000A_H \xleftarrow{\text{sign extend}} 0A_H$  ;  $EA = (BX) + (SI) + 000A_H$

$BA = (DS) \times 16_{10}$  ;  $MA = BA + EA$

$(DX) \leftarrow (MA)$  or  $(DL) \leftarrow (MA)$

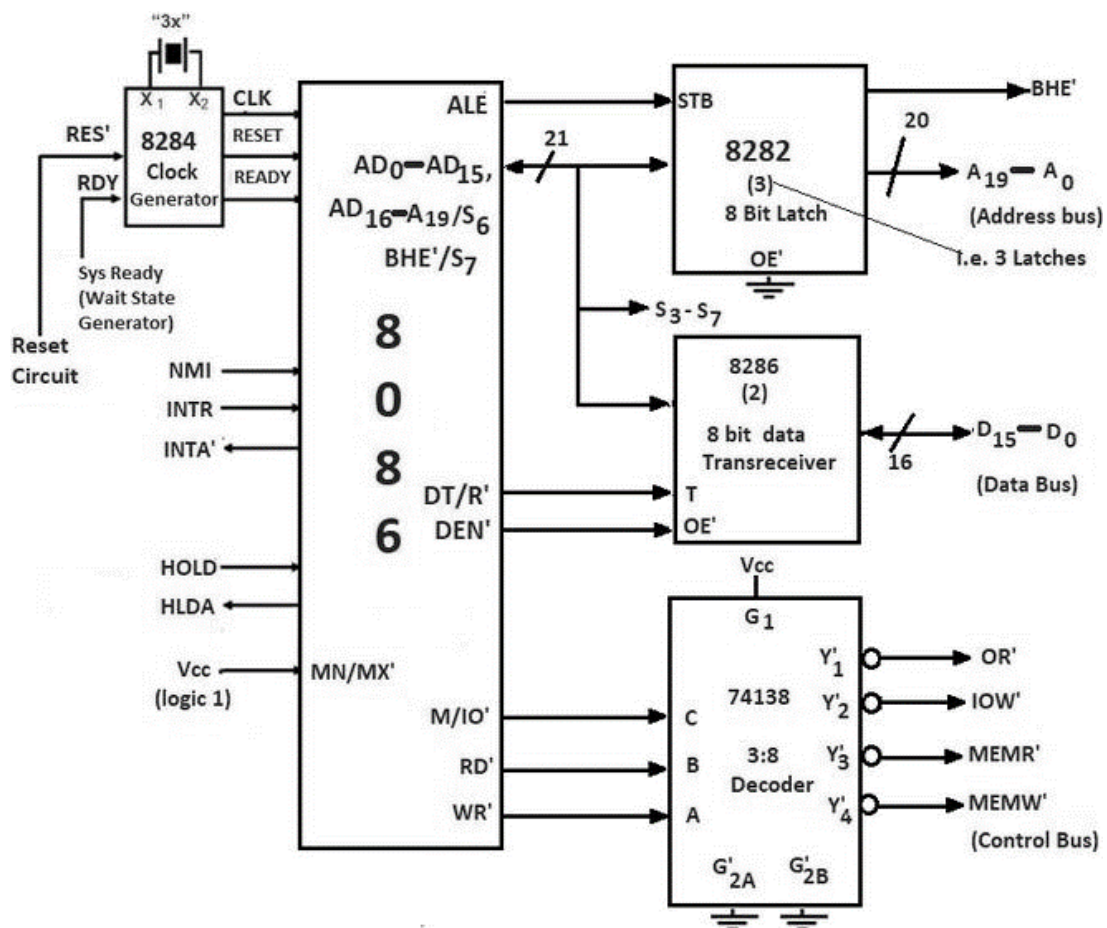
$(DH) \leftarrow (MA + 1)$

*The Effective Address (EA) is calculated by sign extending the 8-bit displacement given in the instruction to 16-bit and adding to the content of BX and SI-register. The Base Address (BA) is obtained by multiplying the content of DS by  $16_{10}$ . The 20-bit Memory Address (MA) is obtained by adding BA and EA.*

*The content of the memory whose address is calculated as explained above is moved to the DL-register and the content of the next memory location is moved to the DH-register.*

### 3.4 8086 MODES OF OPERATION

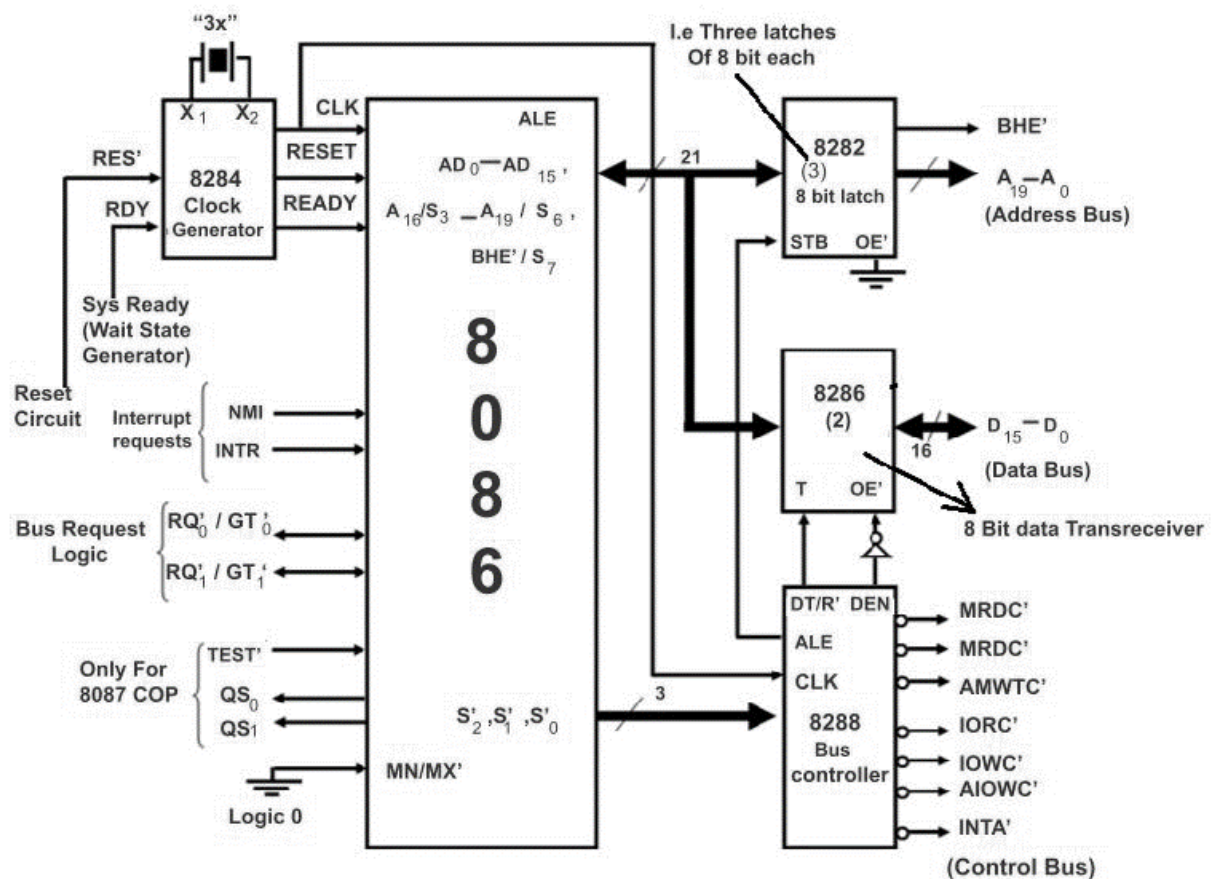
The 8086 can operate in two modes: minimum mode and maximum mode. The mode is decided by a signal at  $\overline{MN/MX}$  pin. When the  $\overline{MN/MX}$  is tied high it works in minimum mode and the system is called a uniprocessor system. When  $\overline{MN/MX}$  is tied low it works in maximum mode and the system is called a multiprocessor system. Usually the pin  $\overline{MN/MX}$  is permanently tied to low or high so that the 8086 system can work in any one of the two modes. The 8086 can work with an 8087 coprocessor in maximum mode. In this mode an external bus controller 8288 is required to generate bus control signals.

**Minimum Mode:****Fig. 3.5: 8086 Minimum mode configuration**

1. 8086 works in Minimum Mode, when value of MN/ MX' = 1.
2. Minimum mode is comparatively smaller than maximum mode and contain single processor.
3. Clock is provided by the 8284 clock generator, it provides CLK, RESET and READY input to 8086.
4. Address from the address bus is latched into 8282 8-bit latch. Three such latches are needed, as address bus is 20-bit. The ALE of 8086 is connected to STB of the latch. The ALE for this latch is given by 8086 itself.
5. The data bus is driven through 8286 8-bit trans-receiver. Two such trans-receivers are needed, as the data bus is 16-bit.
6. Latches: These are buffered output D-Type flip flop, which are used to separate the valid address from the multiplexed address/data bus.

7. Trans-receivers (Data Amplifiers): These are bidirectional buffers, which are used to separate the valid data from the time multiplexed address/data bus.
8. DEN (Data Enable): As the name suggests, this signal indicates whether there is data(valid) over the address/bus lines.
9. DT/R' (Data Transmit/Receiver): This signal indicates the transmission of data i.e., from or to the processor.

### Maximum Mode:



**Fig. 3.6: 8086 Maximum mode configuration**

1. Maximum Mode is operated when the value of MN/MX' is 0.
2. The processor derives the status signal S2, S1, S0 and bus controller derives the controller signal using this status information.
3. Maximum mode there may be more than one microprocessor.
4. Basic function of the bus controller chip is to derive control signals like RD and WR, DEN, DT/R', ALE by the status lines.

5. Bus controller chip has input lines S2', S1', S0' and CLK (these inputs to 8288 are driven by the CPU).
6. Outputs being ALE (Address Latch Enable), DEN (Data Enable), DT/R' (Data Transmit/Read), MRDC (Memory Read Command), MWTC (Memory Write Command Signal), AMWC (Advance memory Write Command), IORC (I/O Read Command), IOWC (I/O Write Command) and AIOWC (Advance I/O Write Command).
7. All these signals instruct the memory to accept or send data from or to the bus.
8. INTA' (Interrupt Acknowledge) used to pulse the interrupt controller or to an interrupting device.

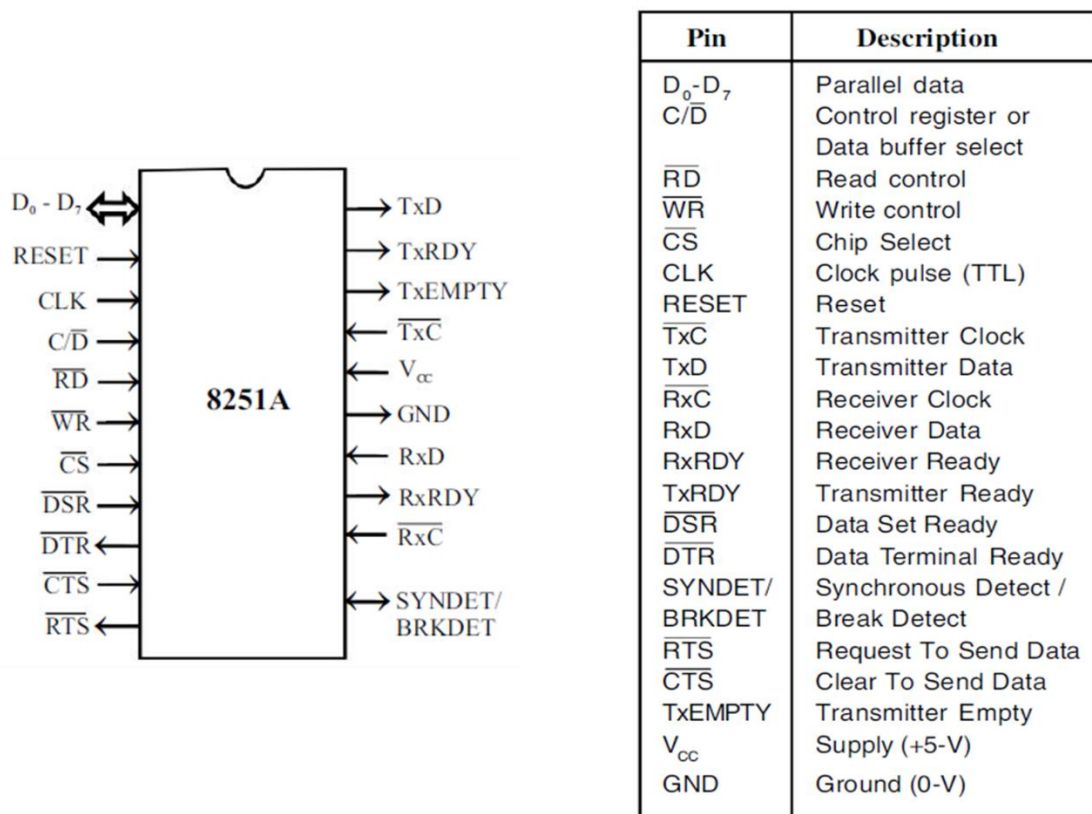
## UNIT-IV

### INTERFACING

#### 4.1 SERIAL COMMUNICATION INTERFACE (8251 – USART)

8251 Universal Synchronous Asynchronous Receiver Transmitter (USART) acts as a mediator between microprocessor and peripheral to convert serial data into parallel form and vice versa.

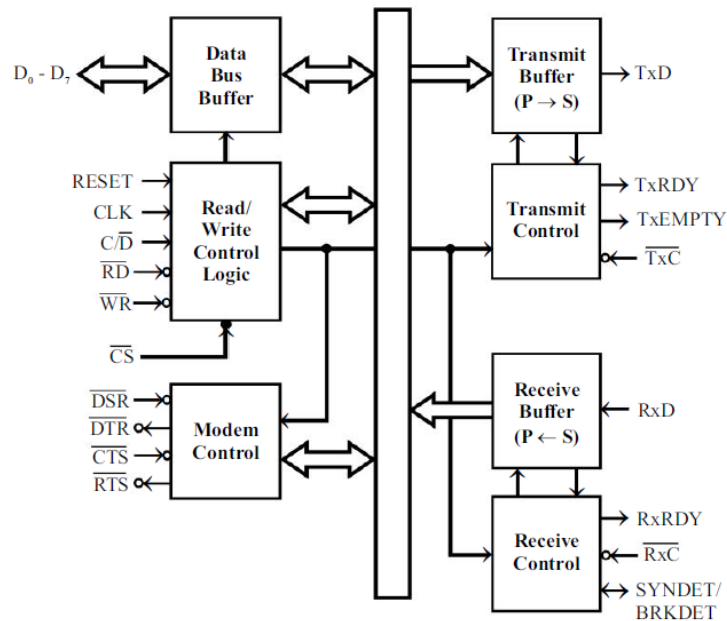
1. It takes data serially from peripheral (outside devices) and converts into parallel data.
2. After converting the data into parallel form, it transmits it to the CPU.
3. Similarly, it receives parallel data from microprocessor and converts it into serial form.
4. After converting data into serial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



**Fig. 4.1: 8251 pin diagram & description**

The functional block diagram of an 8251A is shown in Fig. 4.2. The block diagram shows five sections, they are

1. Read/Write control logic
2. Transmitter
3. Receiver
4. Data bus buffer
5. Modem control



**Fig. 4.2: 8251 Functional Block Diagram**

### Read/Write Control Logic

1. The Read/Write Control logic interfaces the 8251 with CPU, determines the functions of the 8251A according to the control word written into its control register and monitors the data flow.
2. This section has three registers. These are: control register, status register and data buffer. The signals RD', WR', C/D' and CS' are used for read/write operations with these registers.
3. When C/D' is high, the control register is selected for writing control word or reading status word. When C/D' is low, the data buffer is selected for read/write operation.
4. A high on the reset input forces 8251A into the idle mode.
5. The clock input is necessary for 8251A for communication with CPU and this clock does not control either the serial transmission or the reception rate.
6. It is a control block for overall device. It controls the overall working by selecting the operation to be done. The operation selection depends upon input signals as:

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



### **Transmitter Section**

1. The transmitter section accepts parallel data from CPU and converts them into serial data. The transmitter section is double buffered, i.e., it has a buffer register to hold an 8-bit parallel data and another register called the output register to convert the parallel data into a stream of serial bits.
2. The processor loads a data into the buffer register. When the output register is empty, the data is transferred from the buffer to the output register.
3. If the buffer register is ready to transmit a character, then TxRDY is asserted high and if the output register is empty then TxEMPTY is asserted high.
4. The clock signal, TxC' controls the rate at which the bits are transmitted by the USART.

### **Receiver Section**

1. The receiver section accepts the serial data and converts them into parallel data. The receiver section is double buffered, i.e., it has an input register to receive serial data and convert to parallel, and a buffer register to hold the previous converted data.
2. Normally RxD line is high. When the RxD line goes low, the control logic assumes it as a START bit, then the input register accepts the following bits, forms a character and loads it into the buffer register. The CPU reads the parallel data from the buffer register.
3. When the input register loads a parallel data to the buffer register, the RxRDY line goes high.
4. The clock signal RxC controls the rate at which bits are received by the USART.
5. During asynchronous mode, the signal SYNDET/BRKDET will indicate the intentional break in the data transmission, this signal is asserted high to indicate the break in the transmission.
6. During synchronous mode, the signal SYNDET/BRKDET will indicate the reception of the synchronous character.

### **MODEM Control**

The MODEM control unit allows to interface a MODEM to 8251A and to establish data communication through MODEM over telephone lines. This unit takes care of handshake signals for MODEM interface.

### **Data bus buffer**

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

## 4.2 PROGRAMMABLE INTERVAL TIMER – 8253

When the processor has to perform time-based activities, there are two methods to maintain the timings of the operations.

1. In one method, the processor can execute a delay subroutine.

In this method, the delay subroutine will load a count value in one of the register of the processor and start decrementing the count value. After every decrement operation, the zero flag is checked to verify whether the count has reached zero or not. If the count has reached zero, the delay subroutine is terminated. Now the desired time will be elapsed and the processor can perform the desired time based task. In this method, the time is estimated in terms of processor clock periods needed to execute the delay subroutine.

2. In the second method, an external timer can maintain the timings and interrupt the processor at periodic intervals.

In the first method, the processor time is wasted by simply decrementing a register. But in the second method, the processor time can be efficiently utilized, because the processor can perform other tasks in between timer interrupts.

One of the programmable external timer device is the 8253 developed by INTEL. The INTEL 8253 timer has three independent counters. In each counter, a count value can be loaded and the count value can be decremented by applying a clock signal. At the end of count, each counter will generate an output which can be used as interrupt to processor to initiate the time-based activity.

The 8253 is a 24-pin IC packed in DIP and requires a single +5-V supply. The pin configuration of 8254 is shown in Fig. 4.3. The functional block diagram of an 8253 is shown in Fig. 4.4.

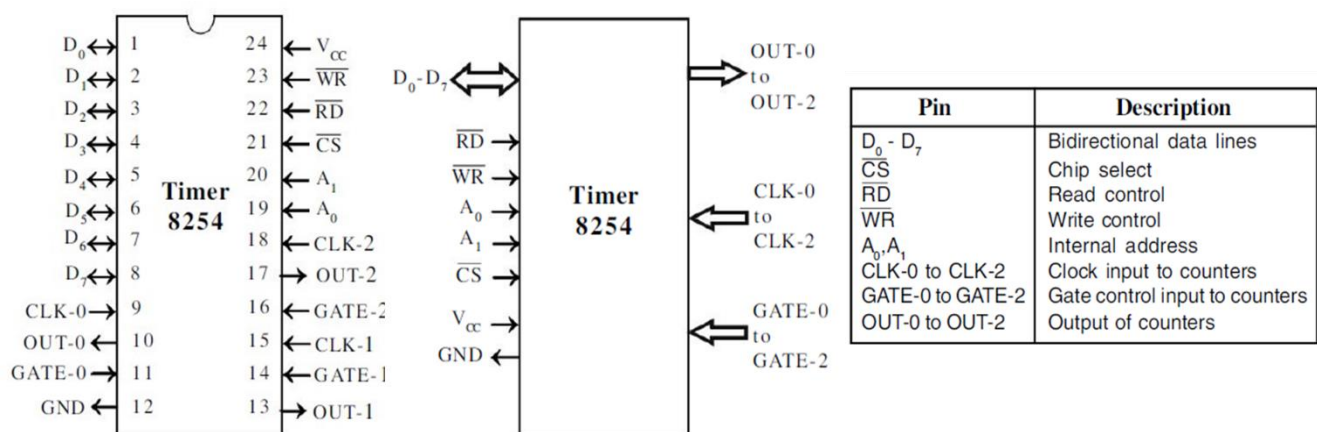
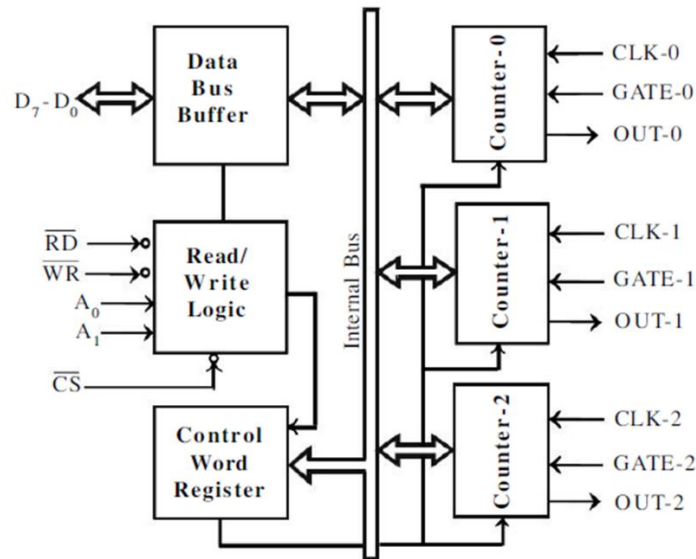


Fig. 4.3: 8253/8254 Pin Diagram



**Fig. 4.4: 8253/8254 Functional Block Diagram**

1. The 8253 has three independent 16-bit counters which can be programmed to work in any one of the possible six modes.
2. Each counter has a clock input, gate input and counter output.
3. To operate a counter, a count value has to be loaded in count register, gate should be tied high and a clock signal should be applied through clock input.
4. The counter counts by decrementing the count value by one in each cycle of clock signal and generates an output depending on the mode of operation. The maximum input clock frequency for 8254 is 10 MHz.
5. The 8254 has eight data lines which can be used for communication with the processor. The control words and count values are written into the 8254 registers through the data bus buffer.
6. The CS' is used to select the chip.
7. The address lines A<sub>0</sub> and A<sub>1</sub> are used to select any one of the four internal devices as shown in Table-4.1.

**Table 4.1. Internal Address of 8254**

Internal address		Device selected
A <sub>1</sub>	A <sub>0</sub>	
0	0	Counter-0
0	1	Counter-1
1	0	Counter-2
1	1	Control Register

8. The control signals RD' and WR' are used by the processor to perform read/write operation. The processor can read the count value in the count register with/without stopping the counter at any time.

*Note : Another timer released by INTEL is the 8253 which is a low clock version of the 8254. The maximum input clock frequency to the 8253 is 2.6 MHz. The 8253 and 8254 are pin to pin compatible and functionally same except the clock frequency.*