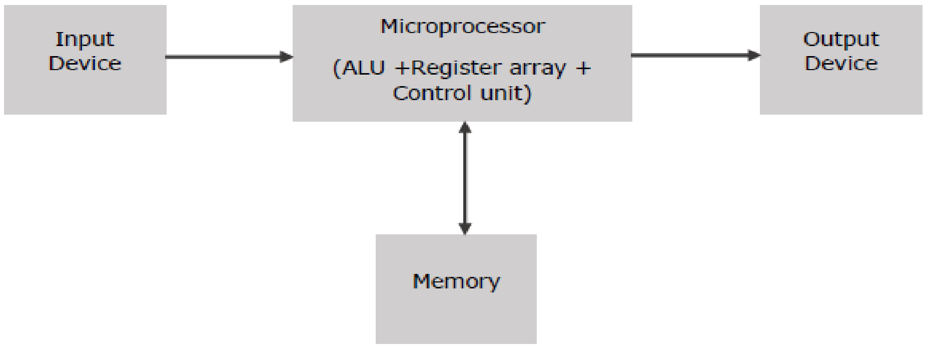
Unit -1 Introduction to Microprocessors [4 Hrs.]

Microprocessor and its Application

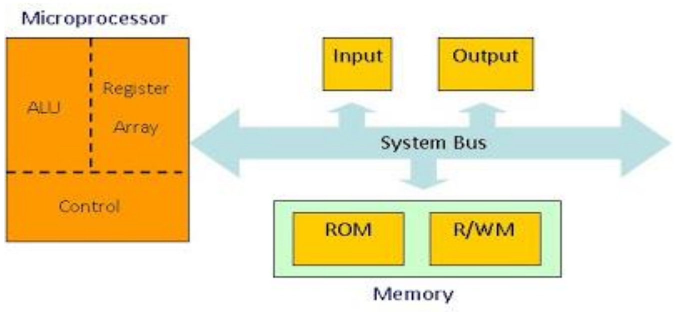


- Microprocessor is a controlling unit of a micro-computer, fabricated on a small chip capable of performing ALU (Arithmetic Logical Unit) operations and communicating with the other devices connected to it.
- Microprocessor consists of an ALU, register array, and a control unit.
- **ALU** performs arithmetical and logical operations on the data received from the memory or an input device.
- **Register array** consists of registers identified by letters like B, C, D, E, H, L and accumulator (register in which intermediate arithmetic and logic results are stored.).
- The **control unit** controls the flow of data and instructions within the computer.

Features of a Microprocessor

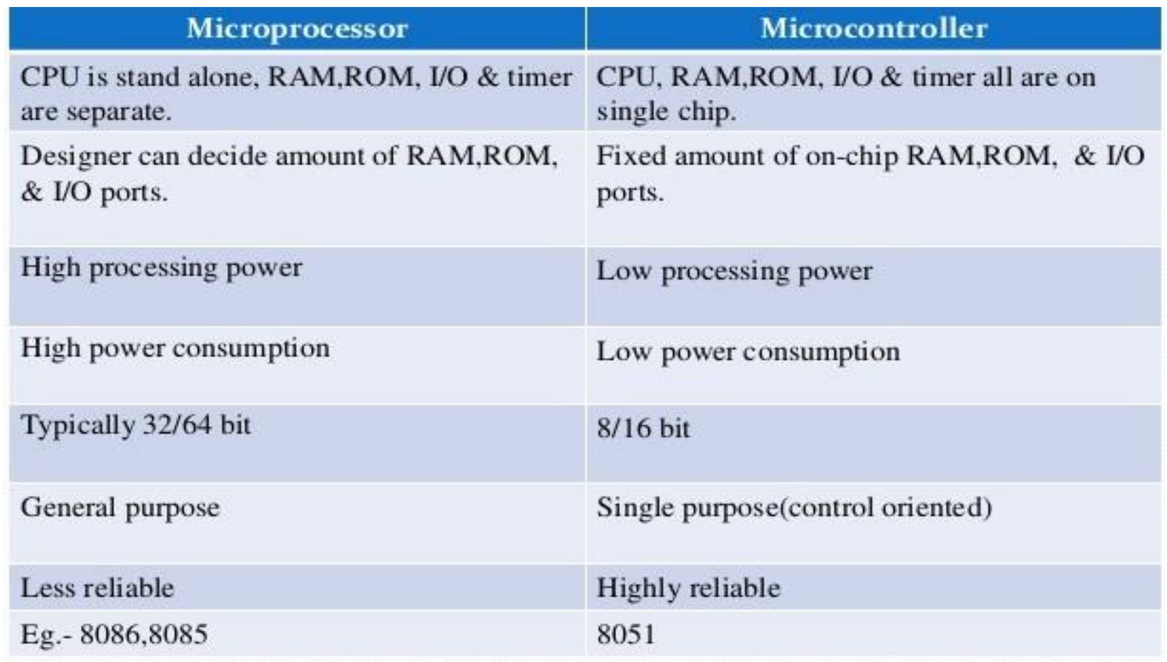
- **Cost-effective** − The microprocessor chips are available at low prices and results its low cost.
- **Size** − The microprocessor is of small size chip, hence is portable.
- **Low Power Consumption** − Microprocessors are manufactured by using metaloxide semiconductor technology, which has low power consumption.
- **Versatility** − The microprocessors are versatile as we can use the same chip in a number of applications by configuring the software program.
- **Reliability** − The failure rate of microprocessors is very low; hence it is reliable.

Components of Microprocessor



- **ALU (Arithmetic/Logic Unit)** – It performs such arithmetic operations as addition and subtraction, and such logic operations as AND, OR, and XOR. Results are stored either in registers or in memory.
- **Register Array** – It consists of various registers identified by letter such as B, C, D, E, H, L, IX, and IY. These registers are used to store data and addresses temporarily during the execution of a program.
- **Control Unit** – The control unit provides the necessary timing and control signals to all the operations in the microcomputer. It controls the flow of data between the microprocessor and memory and peripherals.
- **Input** – The input section transfers data and instructions in binary from the outside world to the microprocessor. It includes such devices as a keyboard, switches, a scanner, and an analog-to-digital converter.
- **Output** – The output section transfers data from the microprocessor to such output devices as LED, CRT, printer, magnetic tape, or another computer.
- **Memory** – It stores such binary information as instructions and data, and provides that information to the microprocessor. To execute programs, the microprocessor reads instructions and data from memory and performs the computing operations in its ALU section. Results are either transferred to the output section for display or stored in memory for later use.
- **System bus** – It is a communication path between the microprocessor and peripherals. The microprocessor communicates with only one peripheral at a time. The timing is provided by the control unit of the microprocessor.

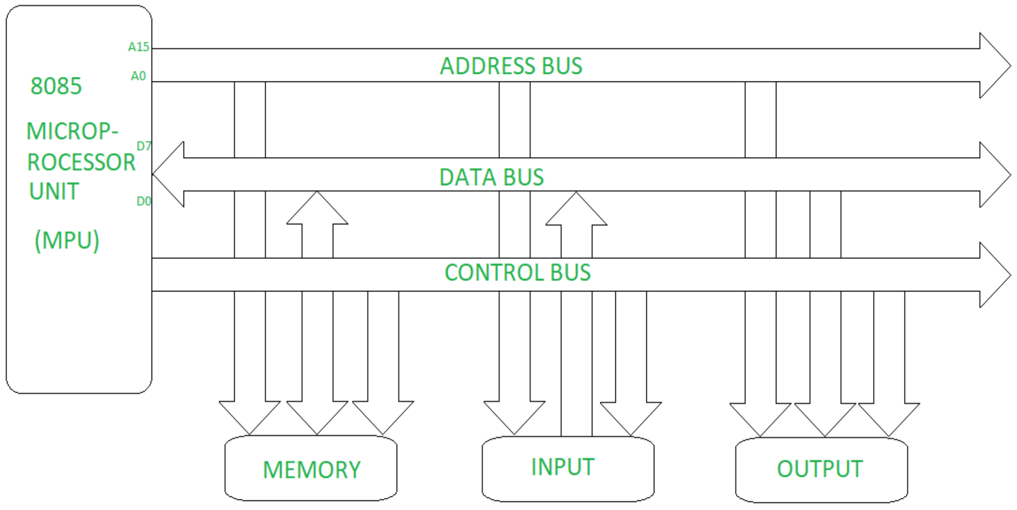
Microprocessor Vs Microcontroller



Microprocessor	Microcontroller
CPU is stand alone, RAM,ROM, I/O & timer are separate.	CPU, RAM,ROM, I/O & timer all are on single chip.
Designer can decide amount of RAM,ROM, & I/O ports.	Fixed amount of on-chip RAM,ROM, & I/O ports.
High processing power	Low processing power
High power consumption	Low power consumption
Typically 32/64 bit	8/16 bit
General purpose	Single purpose(control oriented)
Less reliable	Highly reliable
Eg.- 8086,8085	8051

Bus organization of Microprocessor

Bus is a group of conducting wires which carries information, all the peripherals are connected to microprocessor through Bus.



Bus organization system of 8085 Microprocessor

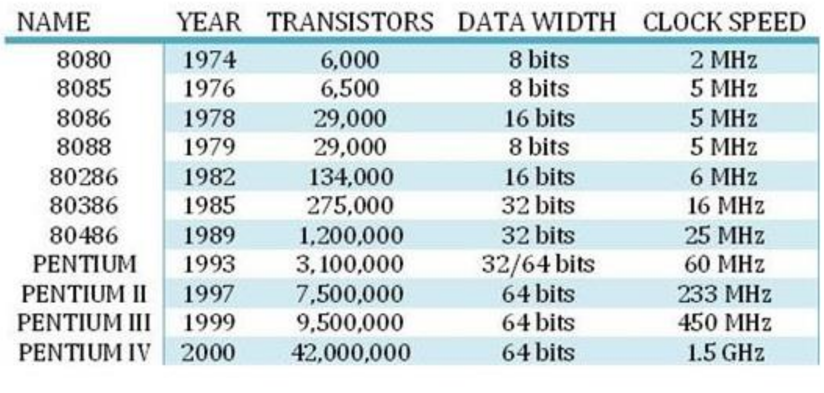
There are three types of buses.

- **Address bus** – It is a group of conducting wires which carries address only. Address bus is unidirectional because data flow in one direction, from microprocessor to memory or from microprocessor to Input/output devices (That is, Out of Microprocessor).
- **Data bus –** It is a group of conducting wires which carries Data only. Data bus is bidirectional because data flow in both directions, from microprocessor to memory or Input/output devices and from memory or Input/output devices to microprocessor.
- **Control bus –**It is a group of wires, which is used to generate timing and control signals to control all the associated peripherals, microprocessor uses control bus to process data, that is what to do with selected memory location. **Some control signals are: Memory read, Memory write, I/O read, I/O Write.**

Evolution of Microprocessor

- The first microprocessor was introduced in the year 1971. It was introduced by Intel and was named Intel 4004.
- Intel 4004 is a 4-bit microprocessor and it was not a powerful microprocessor. It can perform addition and subtraction operation on 4 bits at a time.
- However, it was Intel's 8080 was the first microprocessor to make it to Home computers. It was introduced during the year 1974 and it can perform 8 bit operations.
- Then during the year 1976, Intel introduced 8085 processors which is nothing but an update of 8080 processors.

- Intel introduced 8086 pins during the year 1976. The major difference between 8085 and 8086 processors is that 8085 is an 8-bit processor, but 8086 processors is a 16-bit processor.
- Intel later introduced 8087 processors which was the first math co-processor and later the 8088 processor which was incorporated into IBM personal computers.
- As the years progressed lots of processors from 8088, 80286, 80386, 80486, Pentium II, Pentium III, Pentium IV and now Core2Duo, Dual Core and Quad core processors are the latest in the market.
- Apart from Intel, there are some other manufacturers. Such manufacturers are called second source manufacturers.
- The second source manufacturers include: AMD, Mitsubishi, NEC, OKI, Toshiba, Siemens, Motorola & many more.



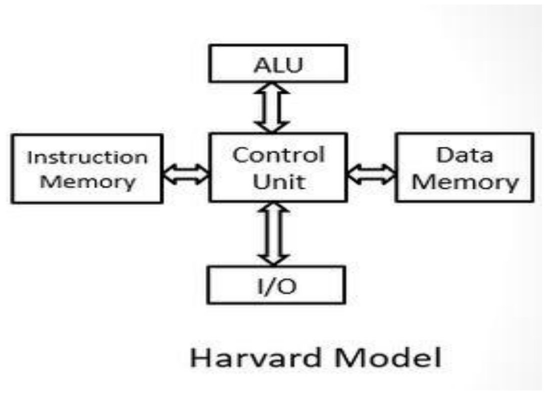
NAME	YEAR	TRANSISTORS	DATA WIDTH	CLOCK SPEED
8080	1974	6,000	8 bits	2 MHz
8085	1976	6,500	8 bits	5 MHz
8086	1978	29,000	16 bits	5 MHz
8088	1979	29,000	8 bits	5 MHz
80286	1982	134,000	16 bits	6 MHz
80386	1985	275,000	32 bits	16 MHz
80486	1989	1,200,000	32 bits	25 MHz
PENTIUM	1993	3,100,000	32/64 bits	60 MHz
PENTIUM II	1997	7,500,000	64 bits	233 MHz
PENTIUM III	1999	9,500,000	64 bits	450 MHz
PENTIUM IV	2000	42,000,000	64 bits	1.5 GHz

<u>Computer Architecture</u>

- Computer architecture is a specification detailing how a set of software and hardware technology standards interact to form a computer system or platform. In short, computer architecture refers to how a computer system is designed and what technologies it is compatible with**.**
- A very good example of computer architecture is von Neumann architecture, which is still used by most types of computers today.
- This was proposed by the mathematician John von Neumann in 1945. It describes the design of an electronic computer with its CPU, which includes the arithmetic logic unit, control unit, registers, memory for data and instructions, an input/output interface and external storage functions.

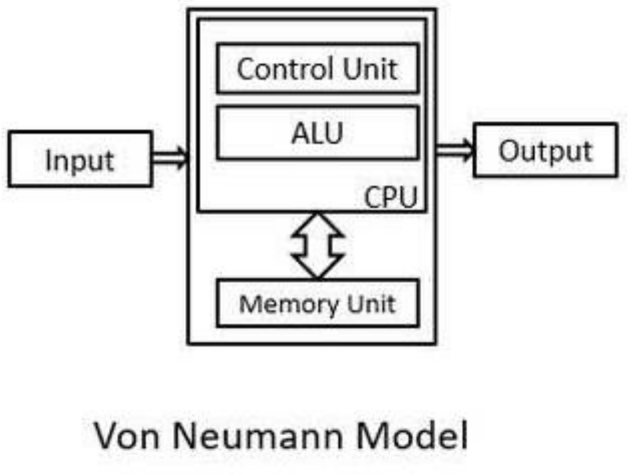
<u>Harvard Architecture</u>

- The Harvard architecture is a computer architecture with physically separate storage and signal pathways for instructions and data. The term originated from the Harvard Mark I, which stored instructions on punched tape and data in electro-mechanical counters.
- These early machines had data storage entirely contained within the central processing unit, and provided no access to the instruction storage as data. It required two memories for their instruction and data. Harvard architecture requires separate bus for instruction and data.



Von Neumann Architecture

- Von Neumann architecture was first published by John von Neumann in 1945.
- His computer architecture design consists of a Control Unit, Arithmetic and Logic Unit (ALU), Memory Unit, Registers and Inputs/Outputs.
- Von Neumann architecture is based on the stored-program computer concept, where instruction data and program data are stored in the same memory. This design is still used in most computers produced today.
- The modern computers are based on a stored-program concept introduced by John von Neumann. In this stored-program concept, programs and data are stored in a separate storage unit called memories and are treated the same. Von Neumann architecture requires only one bus for instruction and data.



UNIT II & IV – 8085 Theory & Instruction Set

Introduction to 8085 Microprocessor

- 8085 is pronounced as "eighty-eighty-five" microprocessor. It is an 8-bit microprocessor designed by Intel in 1977.
- It has the following configuration –
  - 8-bit data bus
  - 16-bit address bus, which can address up to 64KB
  - A 16-bit program counter
  - A 16-bit stack pointer
  - Six 8-bit registers arranged in pairs: BC, DE, HL
  - Requires +5V supply to operate at 3.2 MHZ single phase clock
  - It is used in washing machines, microwave ovens, mobile phones, etc.

8085 Microprocessor Architecture & Functional Units

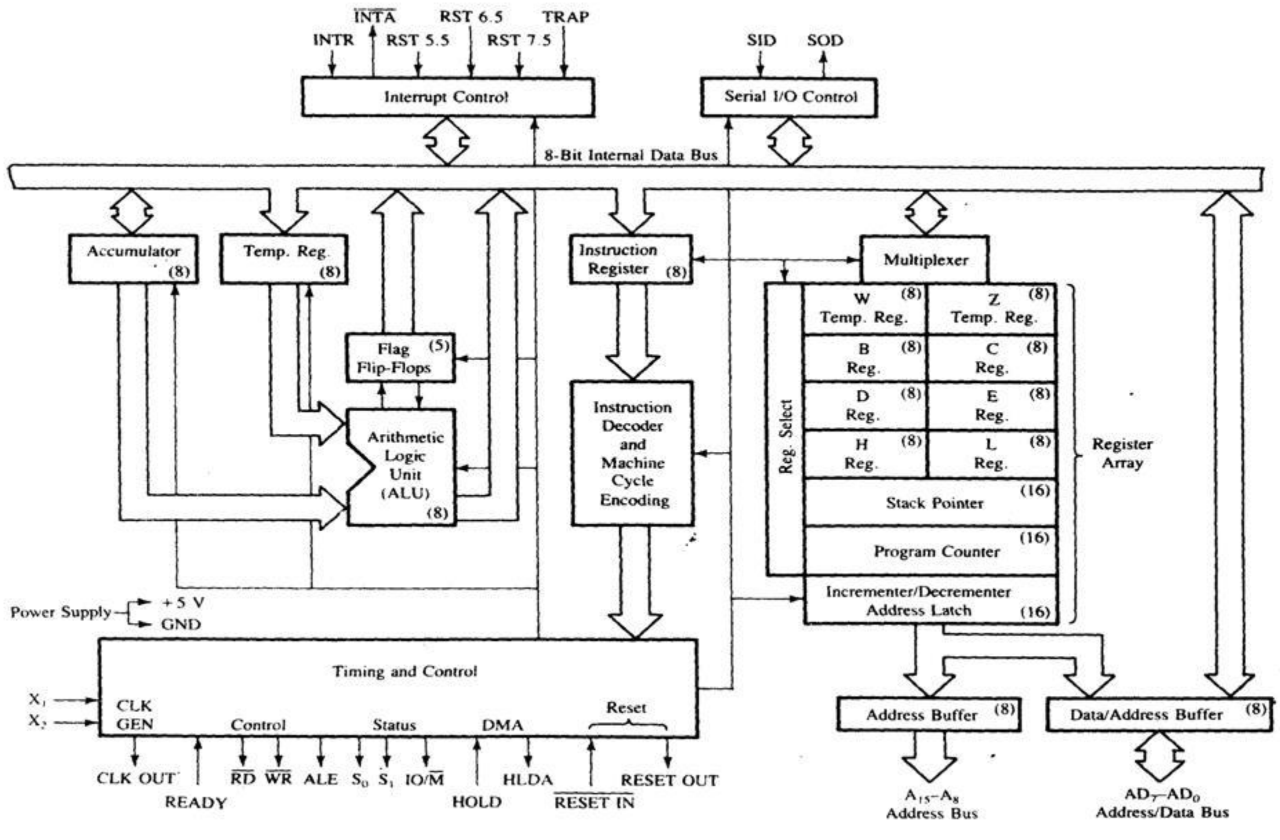


Fig: Block Diagram of 8085

8085 consists of the following functional units −

1) 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.

2) Arithmetic and logic unit

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

3) 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.

4) Program counter

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

5) Stack pointer

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

6) Temporary register

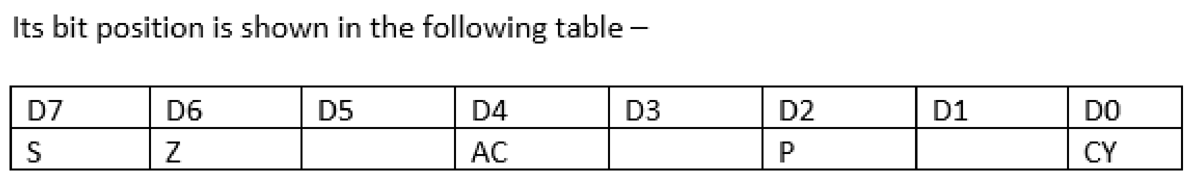
It is an 8-bit register, which holds the temporary data of arithmetic and logical operations.

7) 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

a) **Sign Flag (S) –** After any operation if result is negative sign flag becomes set, i.e. If result is positive sign flag becomes reset i.e. 0.

Example:

– MVI A 30 (load 30H in register A)
  MVI B 40 (load 40H in register B)

SUB B (A = A – B)
These set of instructions will set the sign flag to 1 as 30 – 40 is a negative number.

- MVI A 40 (load 40H in register A)
  MVI B 30 (load 30H in register B)
  SUB B (A = A – B)
  These set of instructions will reset the sign flag to 0 as 40 – 30 is a positive number.

b) Zero Flag (Z) – After any arithmetical or logical operation if the result is 0 (00)H, the zero flag becomes set i.e. 1, otherwise it becomes reset i.e. 0.

Example:

MVI A 10 (load 10H in register A)
SUB A (A = A – A)
These set of instructions will set the zero flag to 1 as 10H – 10H is 00H

c) Auxiliary Carry Flag (AC) – If intermediate carry is generated this flag is set to 1, otherwise it is reset to 0.

Example:

MOV A 2B (load 2BH in register A)
MOV B 39 (load 39H in register B)
ADD B (A = A + B)
These set of instructions will set the auxiliary carry flag to 1, as on adding 2B and 39, addition of lower order nibbles B and 9 will generate a carry.

d) Parity Flag (P) – If after any arithmetic or logical operation the result has even parity, an even number of 1 bits, the parity register becomes set i.e. 1, otherwise it becomes reset.

1-accumulator has even number of 1 bits
0-accumulator has odd parity

e) Carry Flag (CY) – Carry is generated when performing n bit operations and the result is more than n bits, then this flag becomes set i.e. 1, otherwise it becomes reset i.e. 0.

During subtraction (A-B), if A>B it becomes reset and if (A<B) it becomes set.
Carry flag is also called borrow flag.

8) 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.

9) Timing and control unit

It provides timing and control signal to the microprocessor to perform operations. Following are the timing and control signals-
Control Signals: READY, RD’, WR’, ALE
Status Signals: S0, S1, IO/M’
DMA Signals: HOLD, HLDA
RESET Signals: RESET IN, RESET OUT

10) 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.

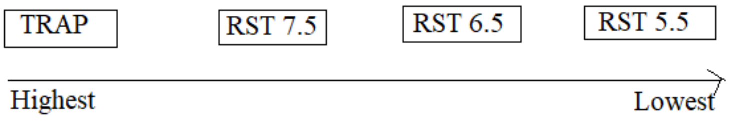
There are 5 interrupt signals in 8085 microprocessors: INTR, RST 7.5, RST 6.5, RST 5.5, TRAP. When microprocessor receives interrupt signals, it sends an acknowledgement (INTA) to the peripheral which is requesting for its service.

Maskable and Non-Maskable Interrupts

- ***Maskable*** *Interrupts* are those which can be disabled or ignored by the microprocessor. *INTR, RST 7.5, RST 6.5, RST 5.5* are maskable interrupts in 8085 microprocessor.
- **Non-Maskable** Interrupts are those which cannot be disabled or ignored by microprocessor. *TRAP* is a non-maskable interrupt.

Priority of Interrupts

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



11) Serial Input/output control

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

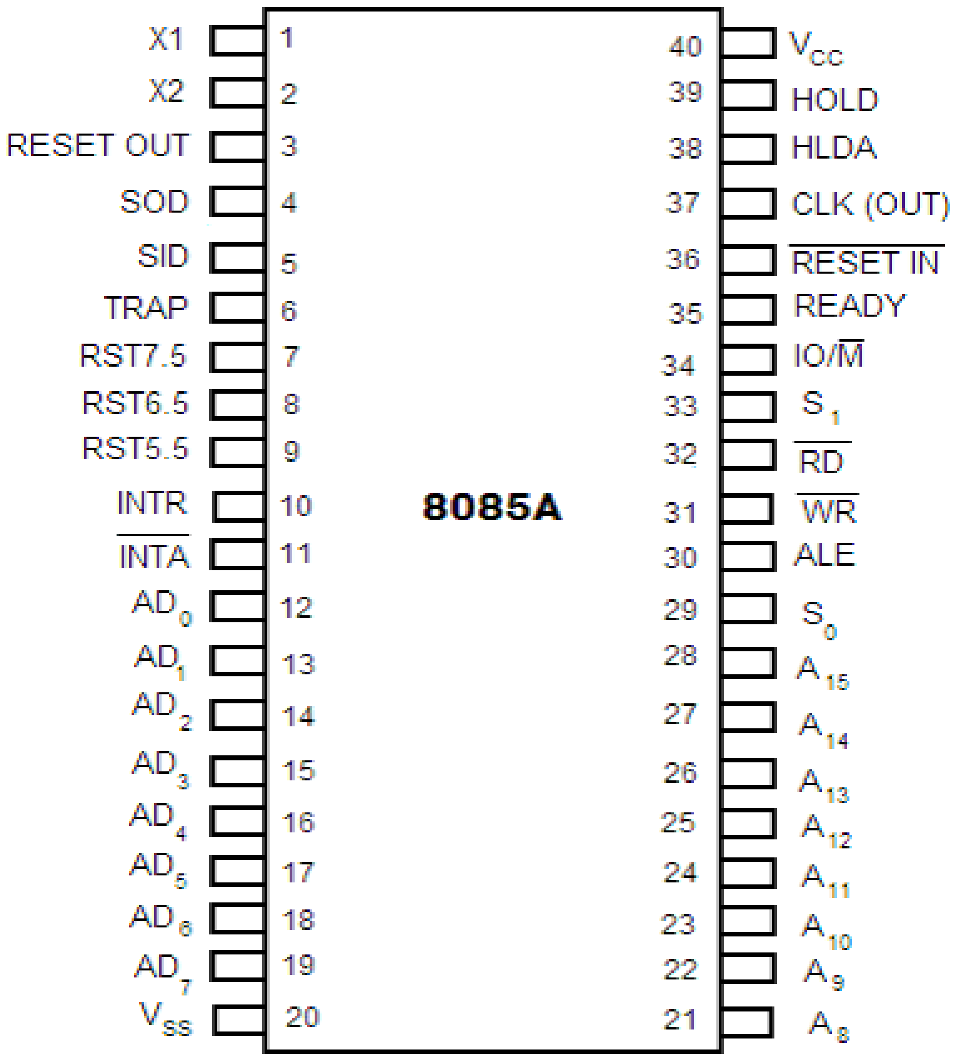
12) Address buffer and address-data buffer

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.

13) Address bus and data bus

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

8085 Pin Configuration



The pins of a 8085 microprocessor can be classified into seven groups −

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. Three control signals are RD, WR & ALE.

a) 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.

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

c) 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.

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.

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.

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.

7) Serial I/O signals

There are 2 serial signals, i.e. SID (Serial input data line) and SOD (Serial output data line) and these signals are used for serial communication.

Addressing Modes in 8085

To perform any operation, we have to give the corresponding instructions to the microprocessor. In each instruction, programmer has to specify 3 things:

1. Operation to be performed.
2. Address of source of data.
3. Address of destination of result.

The method by which the address of source of data or the address of destination of result is given in the instruction is called Addressing Modes. The term addressing mode refers to the way in which the operand of the instruction is specified.

Intel 8085 uses the following addressing modes:

1) Direct Addressing Mode
2) Register Addressing Mode
3) Register Indirect Addressing Mode
4) Immediate Addressing Mode
5) Implicit Addressing Mode

1) Direct Addressing Mode

- In direct addressing mode, the data to be operated is available inside a memory location and that memory location is directly specified as an operand. The operand is directly available in the instruction itself.
  Example:
  LDA 2050 (load the contents of memory location into accumulator A)

2) Register Addressing Mode

- In register addressing mode, the data to be operated is available inside the register(s) and register(s) is(are) operands. Therefore, the operation is performed within various registers of the microprocessor.
  Examples:
  MOV A, B (move the contents of register B to register A)
  ADD B (add contents of registers A and B and store the result in register A)
  INR A (increment the contents of register A by one)

3) Register Indirect Addressing Mode

In register indirect addressing mode, the data to be operated is available inside a memory location and that memory location is indirectly specified by a register pair.

Example:
MOV A, M (move the contents of the memory location pointed by the H-L pair to the accumulator)

4) Immediate Addressing Mode

In immediate addressing mode the source operand is always data. If the data is 8-bit, then the instruction will be of 2 bytes, if the data is of 16-bit then the instruction will be of 3 bytes.

Example:
MVI B 45 (move the data 45H immediately to register B)

5) Implied/Implicit Addressing Mode

In implied/implicit addressing mode the operand is hidden and the data to be operated is available in the instruction itself.

Examples:
RRC (rotate accumulator A right by one bit)
RLC (rotate accumulator A left by one bit)

Instruction Set of 8085

An **instruction** is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions that a microprocessor supports is called **Instruction Set**.

8085 has **246** instructions. Each instruction is represented by an 8-bit binary value. These 8-bits of binary value is called **Op-Code** or **Instruction Byte**.

Following are the classification of instructions:

a) Data Transfer Instruction
b) Arithmetic Instructions
c) Logical Instructions
d) Branching Instructions
e) Control Instructions

a) Data Transfer Instruction

These instructions move data between registers, or between memory and registers. These instructions copy data from source to destination. While copying, the contents of source are not modified.
Example: MOV, MVI

b) Arithmetic Instructions

These instructions perform the operations like addition, subtraction, increment and decrement.
Example: ADD, SUB, INR, DCR

c) Logical Instructions

These instructions perform logical operations on data stored in registers and memory. The logical operations are: AND, OR, XOR, Rotate, Compare and Complement.
Example: ANA, ORA, RAR, RAL, CMP, CMA

d) Branching Instructions

Branching instructions refer to the act of switching execution to a different instruction sequence as a result of executing a branch instruction. The three types of branching instructions are: Jump, Call and Return.

e) Control Instructions

The control instructions control the operation of microprocessor. Examples: HLT, NOP, EI (Enable Interrupt), DI (Disable Interrupt).

Data Transfer Instructions

Instruction	**Opcode**	**Addressing Mode**	**Bytes**	**Description**
LDA address	3A	Direct	3	Transfers contents stored in memory location to accumulator.
STA address	32	Direct	3	Transfers contents stored in accumulator to memory address.
MOV A, A	7F	Register	1	Transfers the contents from one register to another

MOV A, B	78	Register	1	
MOV A, C	79	Register	1	
MOV A, D	7A	Register	1	
MOV A, E	7B	Register	1	
MOV A, H	7C	Register	1	
MOV A, L	7D	Register	1	
MOV B, A	47	Register	1	
MOV B, B	40	Register	1	
MOV B, C	41	Register	1	
MOV B, D	42	Register	1	
MOV B, E	43	Register	1	
MOV B, H	44	Register	1	
MOV B, L	45	Register	1	
MOV C, A	4F	Register	1	
MOV C, B	48	Register	1	
MOV C, C	49	Register	1	
MOV C, D	4A	Register	1	
MOV C, E	4B	Register	1	
MOV C, H	4C	Register	1	
MOV C, L	4D	Register	1	
MOV D, A	57	Register	1	
MOV D, B	50	Register	1	
MOV D, C	51	Register	1	
MOV D, D	52	Register	1	

MOV D, E	53	Register	1	
MOV D, H	54	Register	1	
MOV D, L	55	Register	1	
MOV E, A	5F	Register	1	
MOV E, B	58	Register	1	
MOV E, C	59	Register	1	
MOV E, D	5A	Register	1	
MOV E, E	5B	Register	1	
MOV E, H	5C	Register	1	
MOV E, L	5D	Register	1	
MOV H, A	67	Register	1	
MOV H, B	60	Register	1	
MOV H, C	61	Register	1	
MOV H, D	62	Register	1	
MOV H, E	63	Register	1	
MOV H, H	64	Register	1	
MOV H, L	65	Register	1	
MOV L, A	6F	Register	1	
MOV L, B	68	Register	1	
MOV L, C	69	Register	1	
MOV L, D	6A	Register	1	
MOV L, E	6B	Register	1	
MOV L, H	6C	Register	1	
MOV L, L	6D	Register	1	

MOV A, M	7E	Register Indirect	1	It moves / copies the data stored in memory location whose address is given in H-L register pair, to the given register.
MOV B, M	46	Register Indirect	1	
MOV C, M	4E	Register Indirect	1	
MOV D, M	56	Register Indirect	1	
MOV E, M	5E	Register Indirect	1	
MOV H, M	66	Register Indirect	1	
MOV L, M	6E	Register Indirect	1	
MOV M, A	77	Register Indirect	1	This instruction moves / copies the data in the given register to the memory location addressed by H-L register pair.
MOV M, B	70	Register Indirect	1	
MOV M, C	71	Register Indirect	1	
MOV M, D	72	Register Indirect	1	
MOV M, E	73	Register Indirect	1	
MOV M, H	74	Register Indirect	1	
MOV M, L	75	Register Indirect	1	
MVI A, data	3E	Immediate	2	This instruction transfers the given data immediately to the register.
MVI B, data	06	Immediate	2	
MVI C, data	0E	Immediate	2	
MVI D, data	16	Immediate	2	
MVI E, data	1E	Immediate	2	
MVI H, data	26	Immediate	2	
MVI L, data	2E	Immediate	2	

IN port-address (8-bit)	DB	Immediate	2	Load data from input port to accumulator.
OUT port-address (8-bit)	D3	Immediate	2	Transfer data to output port from accumulator.
LXI H, 16-bit data	21	Immediate	3	Transfer 16-bit data to H-L pair.
LXI D, 16-bit data	11	Immediate		Transfer 16-bit data to D-E pair.
LXI B, 16-bit data	01	Immediate	3	Transfer 16-bit data to B-C pair.
XCHG	EB	Implied/Implicit	1	Exchange contents of H-L pair and D-E pair.
LHLD address	2A	Direct	3	Load 16-bit contents of memory address to H-L pair.
SHLD address	22	Direct	3	Load 16-bit contents of H-L pair to memory address.
LDAX B	0A	Register Indirect	1	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.
LDAX D	1A	Register Indirect	1	
STAX B	02	Register Indirect	1	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.
STAX D	12	Register Indirect	1	
				Total = 84

Arithmetic Instructions

Instruction	Opcode	Addressing Mode	Bytes	Description
ADD A	87	Register	1	It adds the content stored in given register with the accumulator. The result of this addition is stored in accumulator.
ADD B	80	Register	1	
ADD C	81	Register	1	
ADD D	82	Register	1	
ADD E	83	Register	1	
ADD H	84	Register	1	
ADD L	85	Register	1	
ADD M	86	Register Indirect	1	adds the content of memory location whose address is given in H-L register pair with the accumulator and the answer is stored in accumulator.
ADI data	C6	Immediate	2	It immediately adds the given data with the accumulator and the answer will be stored in Accumulator.
ADC A	C6	Register	1	A=A+A+CY (Add with carry)
ADC B	8F	Register	1	A=A+B+CY
ADC C	88	Register	1	
ADC D	89	Register	1	
ADC E	8B	Register	1	
ADC H	8C	Register	1	
ADC L	8D	Register	1	
ADC M	8E	Register Indirect	1	

ACI data	CE	Immediate	2	
DAD B	09	Register	1	HL=HL+BC
DAD D	19	Register	1	HL=HL+DE
DAD H	29	Register	1	HL=HL+HL
SUB A	97	Register	1	A=A-A
SUB B	90	Register	1	A=A-B
SUB C	91	Register	1	
SUB D	92	Register	1	
SUB E	93	Register	1	
SUB H	94	Register	1	
SUB L	95	Register	1	
SUB M	96	Register Indirect	1	
SUI data	D6	Immediate	2	
SBB A	9F	Register	1	A=A-A-CY
SBB B	98	Register	1	A=A-B-CY
SBB C	99	Register	1	
SBB D	9A	Register	1	
SBB E	9B	Register	1	
SBB H	9C	Register	1	
SBB L	9D	Register	1	
SBB M	9E	Register Indirect	1	
SBI data	DE	Immediate	2	
INR A	3C	Register	1	A=A+1
INR B	04	Register	1	B=B+1
INR C	0C	Register	1	

INR D	14	Register	1	
INR E	1C	Register	1	
INR H	24	Register	1	
INR L	2C	Register	1	
INR M	34	Register Indirect	1	M=M+1 (pointed by HL)
INX B	03	Register	1	BC=BC+1
INX D	13	Register	1	DE=DE+1
INX H	23	Register	1	HL=HL+1
DCR A	3D	Register	1	A=A-1
DCR B	05	Register	1	
DCR C	0D	Register	1	
DCR D	15	Register	1	
DCR E	1D	Register	1	
DCR H	25	Register	1	
DCR L	2D	Register	1	
DCR M	35	Register Indirect	1	M=M-1
DCX B	0B	Register	1	BC=BC-1
DCX D	1B	Register	1	DE=DE-1
DCX H	2B	Register	1	HL=HL-1
RAL	17	Implied/Implicit	1	Rotate accumulator left
RAR	1F	Implied/Implicit	1	Rotate accumulator right
RLC	07	Implied/Implicit	1	Rotate accumulator left through carry
RRC	0F	Implied/Implicit	1	Rotate accumulator right through carry
				Total=65

Logical Instructions

Instruction	Opcode	Addressing Mode	Bytes	Description
ANA A	A7	Register	1	A=A AND A
ANA B	A0	Register	1	A=A AND B
ANA C	A1	Register	1	
ANA D	A2	Register	1	
ANA E	A3	Register	1	
ANA H	A4	Register	1	
ANA L	A5	Register	1	
ANA M	A6	Register Indirect	1	
ANI data	E6	Immediate	2	A=A AND data
ORA A	B7	Register	1	A=A OR A
ORA B	B0	Register	1	A=A OR B
ORA C	B1	Register	1	
ORA D	B2	Register	1	
ORA E	B3	Register	1	
ORA H	B4	Register	1	
ORA L	B5	Register	1	
ORA M	B6	Register Indirect	1	
ORI data	F6	Immediate	2	
XRA A	AF	Register	1	A=A XOR A
XRA B	A8	Register	1	A=A XOR B
XRA C	A9	Register	1	
XRA D	AA	Register	1	

XRA E	AB	Register	1	
XRA H	AC	Register	1	
XRA L	AD	Register	1	
XRA M	AE	Register Indirect	1	
XRI data	EE	Immediate	2	
CMP A	BF	Register	1	A=A-A (Accumulator remain unchanged)
CMP B	B8	Register	1	A=A-B
CMP C	B9	Register	1	
CMP D	BA	Register	1	
CMP E	BB	Register	1	
CMP H	BC	Register	1	
CMP L	BD	Register	1	
CMP M	BE	Register Indirect	1	
CPI data	FE	Immediate	2	A=A-data (Acc. Remain unchanged)
CMA	2F	Implied/implicit	1	Complement Accumulator contents
CMC	3F	Implied/implicit	1	Complement Carry flag
STC	37	Implied/implicit	1	CY=1 (It sets carry flag)
				Total=39

Branching Instructions

Instruction	Opcode	Addressing Mode	Bytes	Description
JMP address	C3	Immediate	3	Unconditional Jump
JC address	DA	Immediate	3	Jump if CY=1
JNC address	D2	Immediate	3	Jump if CY=0
JZ address	CA	Immediate	3	Jump if Z=1
JNZ address	C2	Immediate	3	Jump if Z=0
JM address	FA	Immediate	3	Jump if S=1
JP address	F2	Immediate	3	Jump if S=0
JPE address	EA	Immediate	3	Jump if P=1
JPO address	E2	Immediate	3	Jump if P=0
CALL address	CD	Immediate	3	Unconditional Call
CC address	DC	Immediate	3	Call Subroutine if CY=1
CNC address	D4	Immediate	3	Call Subroutine if CY=0
CZ address	CC	Immediate	3	Call Subroutine if Z=1
CNZ address	C4	Immediate	3	Call Subroutine if Z=0
CM address	FC	Immediate	3	Call Subroutine if S=1
CP address	F4	Immediate	3	Call Subroutine if S=0
CPE address	EC	Immediate	3	Call Subroutine if P=1
CPO address	FE	Immediate	3	Call Subroutine if P=1
RNZ	C0	Register Indirect	1	Return to main program if Z=0
RZ	C8	Register Indirect	1	Return to main program if Z=1
RNC	D0	Register Indirect	1	Return to main program if C=0
RC	D8	Register Indirect	1	Return to main program if C=1

RM	F8	Register Indirect	1	Return to main program if S=1
RP	F0	Register Indirect	1	Return to main program if S=0
RPO	E0	Register Indirect	1	Return to main program if P=0
RPE	E8	Register Indirect	1	Return to main program if P=1
				Total=26

Program Control & Stack Instructions

Instruction	**Opcode**	**Addressing Mode**	**Bytes**	**Description**
PUSH B	C5	Register Indirect	1	PUSH data from data from stack on the basis of address pointed by BC pair.
PUSH D	D5	Register Indirect	1	
PUSH H	E5	Register Indirect	1	
POP B	C1	Register Indirect	1	POP data from data from stack on the basis of address pointed by BC pair.
POP D	D1	Register Indirect	1	
POP H	E1	Register Indirect	1	
NOP	00	Implied/Implicit	1	No operation is performed
HLT	76	Implied/Implicit	1	Terminate program
DI	F3	Implied/Implicit	1	
EI	7B	Implied/Implicit	1	
RIM	20	Implied/Implicit	1	Read interrupt mask (read status of interrupt)
SIM	30	Implied/Implicit	1	Set interrupt mask (used to implement interrupt)
				Total=12 Grand Total=226

UNIT II & IV – 8086 Theory & Instruction Set

Introduction to 8086 Microprocessor

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

Features 8086 Microprocessor

- It has an **instruction queue**, which is capable of **storing six instruction bytes** from the memory resulting in faster processing.
- It was the first 16-bit processor having 16-bit ALU, 16-bit registers, internal data bus, and 16-bit external data bus resulting in faster processing.
- It uses **two stages of pipelining**, i.e. **Fetch Stage and Execute Stage**, which improves performance. **Fetch stage** can prefetched up to 6 bytes of instructions and stores them in the queue. **Execute stage** executes these instructions.
- It consists of **29,000 transistors**.
- It can address upto **1 MB** of memory.

Comparison between 8085 & 8086

SN	8085 Microprocessor	8086 Microprocessor
1	It is an 8 bit microprocessor.	It is a 16 bit microprocessor.
2	It has 16 bit address line.	It has 20 bit address line.
3	It has 8- bit data bus.	It has 16- bit data bus.
4	The memory capacity is 64 KB.	The memory capacity is 1 MB.
5	Clock speed of this microprocessor is 3 MHz.	Clock speed of this microprocessor varies between 5, 8 and 10 MHz for different versions.
6	It has 5 flags.	It has 9 flags.
7	8085 microprocessor does not support memory segmentation.	8086 microprocessor supports memory segmentation.
8	It does not support pipelining.	It supports pipelining.
9	It is accumulator based processor.	It is general purpose register based processor.

10	It has no minimum or maximum mode.	It has minimum and maximum modes.
11	In 8085, only one processor is used.	In 8086, more than one processor is used. Additional external processor can also be employed.
12	It contains less number of transistors compare to 8086 microprocessor. It contains about 6500 transistor.	It contains more number of transistors compare to 8085 microprocessor. It contains about 29000 in size.
13	The cost of 8085 is low.	The cost of 8086 is high.

Internal Architecture of 8086

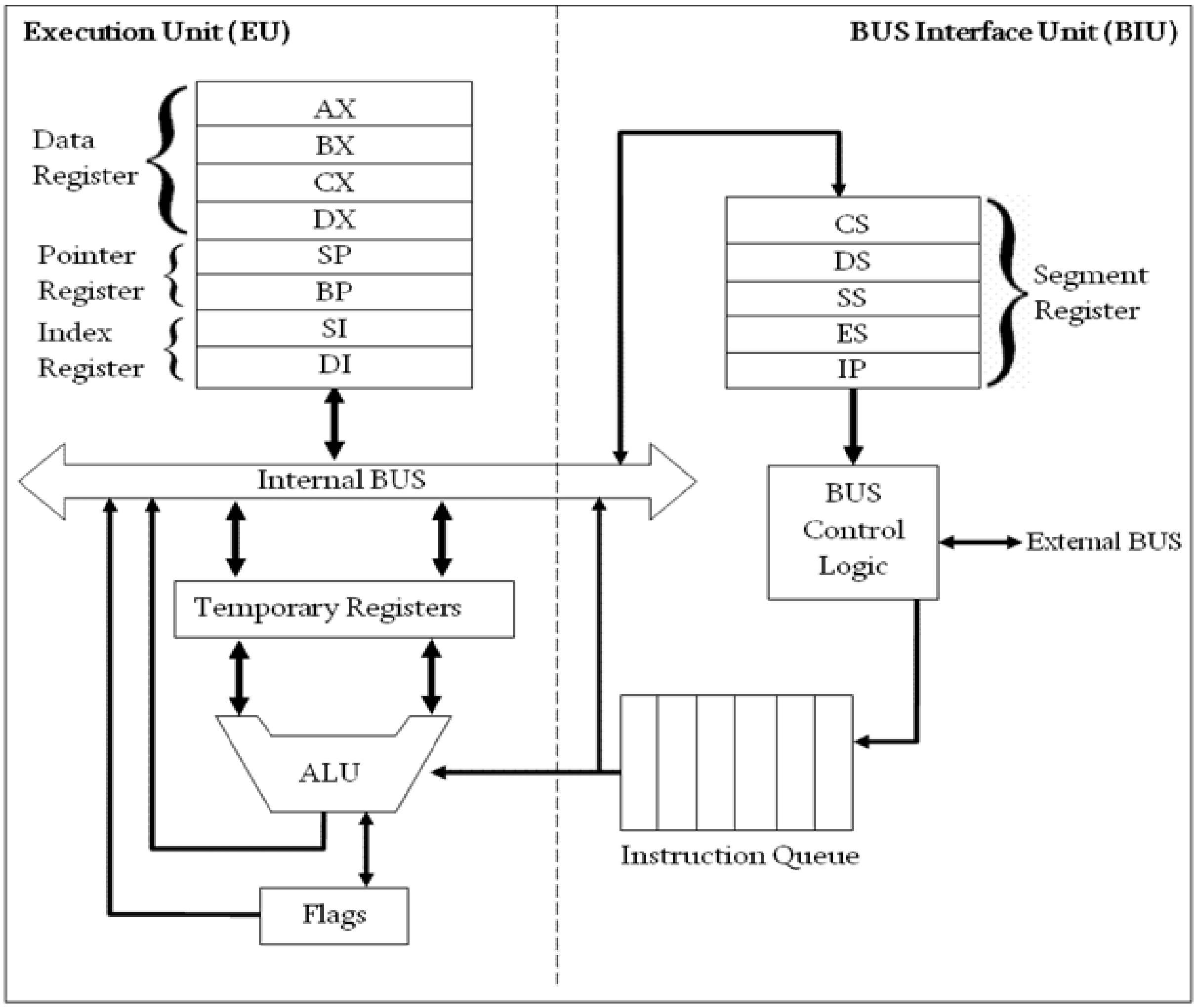


Fig: Block Diagram of 8086

8086 contains two independent functional units: a **Bus Interface Unit (BIU)** and **an Execution Unit (EU).**

Bus Interface Unit (BIU)

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

- **The BIU:**
- ✓ Handles transfer of data and addresses,
- ✓ Fetches instruction codes, stores fetched instruction codes in first-in-first-out register set called a queue,
- ✓ Reads data from memory and I/O devices,
- ✓ Writes data to memory and I/O devices

- It has the following functional parts:
- ✓ **Instruction Queue:** When EU executes instructions, the BIU gets 6-bytes of the next instruction and stores them in the instruction queue and this process is known as instruction pre fetch. This process increases the speed of the processor.

- ✓ **Segment Registers:** A segment register contains the addresses of instructions and data in memory which are used by the processor to access memory locations.

- ✓ **Instruction Pointer (IP):** The instruction pointer in the 8086 microprocessor acts as a program counter. It indicates to the address of the next instruction to be executed.

Memory Segmentation

- ✓ To increase execution speed and fetching speed, 8086 segments the memory.
- ✓ It's 20 bit address bus can address 1MB of memory, it segments it into 4 64kB segments.
- ✓ 8086 works only with four 64KB segments within the whole 1MB memory.

- ✓ There are 4 segment registers in 8086 as given below:
- **Code Segment Register (CS):** Code segment of the memory holds instruction codes of a program.
- **Data Segment Register (DS):** The data, variables and constants given in the program are held in the data segment of the memory.
- **Stack Segment Register (SS):** Stack segment holds addresses and data of subroutines. It also holds the contents of registers and memory locations given in PUSH instruction.
- **Extra Segment Register (ES):** Extra segment holds the destination addresses of some data of certain string instructions.

Execution Unit(EU)

- ✓ The EU receives opcode of an instruction from the queue, decodes it and then executes it. While Execution, unit decodes or executes an instruction, then the BIU fetches instruction codes from the memory and stores them in the queue.

- ✓ It consists of following:
- **General Purpose Registers:** There are four 16-bit general purpose registers: AX (Accumulator Register), BX (Base Register), CX (Counter) and DX.
- **Index Register:** The following four registers are in the group of pointer and index registers: Stack Pointer (SP), Base Pointer (BP), Source Index (SI), Destination Index (DI).

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

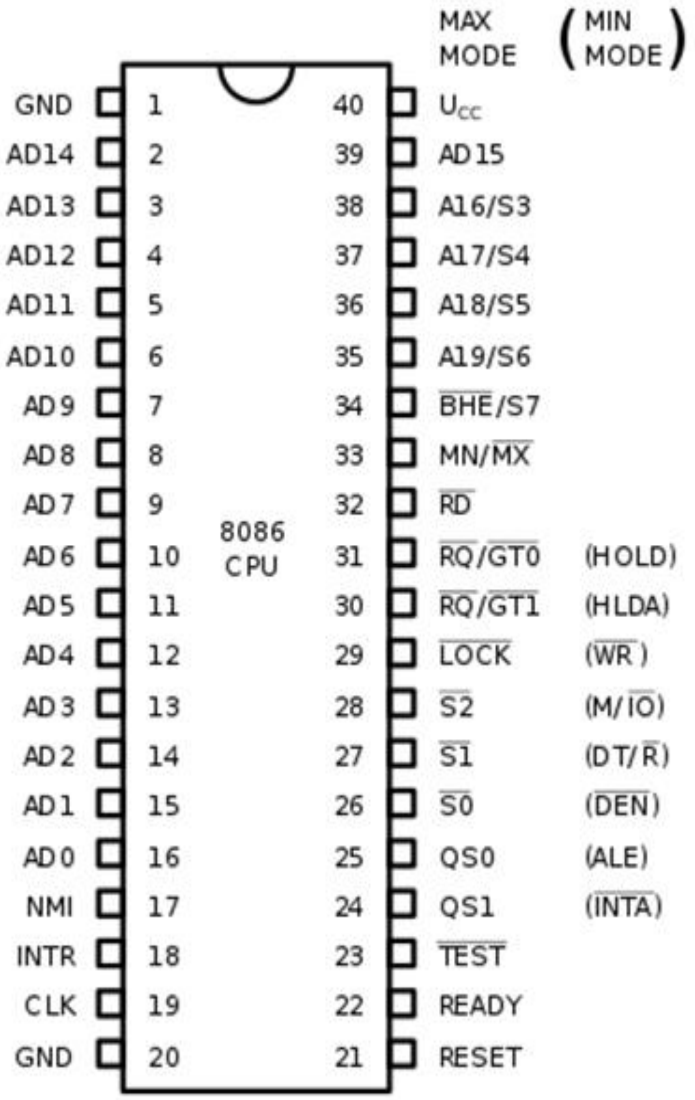
- **Flag Register:** It is a 16 bit register which exactly behaves like a flip-flop, means it changes states according to the result stored in the accumulator. It has **9 flags** and they are divided into **2 groups i.e. conditional and control flags**.

- ✓ **Conditional Flags:** This flag represents the result of the last arithmetic or logical instruction executed. Conditional flags are:
  - Carry Flag
  - Auxiliary Flag
  - Parity Flag
  - Zero Flag
  - Sign Flag
  - Overflow Flag

- ✓ **Control Flags:** It controls the operations of the execution unit. Control flags are:
  - Trap Flag
  - Interrupt Flag
  - Direction Flag

- ✓ **Interrupts:** The Intel 8086 has two hardware interrupt pins:
  - NMI (Non-Maskbale Interrupt)
  - INTR (Interrupt Request) Maskable Interrupt.

Pin Configuration of 8086



- **AD0-AD15 (Address Data Bus):** Bidirectional address/data lines. These are low order address bus. When these lines are used to transmit memory address the symbol A is used instead of AD for example A0- A15.
- **A16 - A19 (Output):** High order address lines. These are multiplexed with status signals.
- **A16/S3, A17/S4:** A16 and A17 are multiplexed with segment identifier signals S3 and S4.
- **A18/S5:** A18 is multiplexed with interrupt status S5.
- **A19/S6:** A19 is multiplexed with status signal S6.
- **BHE/S7 (Output):** 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 S7. S7 signal is available during T3 and T4.
- **RD (Read):** For read operation. It is an output signal. It is active when LOW.
- **Ready (Input):** The addressed memory or I/O sends acknowledgement through this pin. When HIGH it denotes that the peripheral is ready to transfer data.

- **RESET (Input):** System reset.
- **CLK (input):** Clock 5, 8 or 10 MHz.
- **INTR:** Interrupt Request.
- **NMI (Input):** Non-maskable interrupt request.
- **TEST (Input):** Wait for test control. When LOW the microprocessor continues execution otherwise waits.
- **VCC:** Power supply +5V dc. **GND:** Ground.

8086 Addressing Modes

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

Types of addressing modes:

a) Register Addressing Mode (**Eg. MOV AX,BX**)
b) Direct Addressing Mode
c) Immediate Addressing Mode
d) Implied/Implicit Addressing Mode
e) **Register Indirect mode** - In this addressing mode the effective address is in SI, DI or BX. Example:

```
MOV AX, [DI]
ADD AL, [BX]
MOV AX, [SI]
```

f) **Based Indexed Mode -** In this the effective address is sum of base register and index register. Example,
Base register: BX, BP
Index register: SI, DI
The physical memory address is calculated according to the base register.
MOV AL, [BP+SI]
MOV AX, [BX+DI]

g) **Indexed mode –** In this type of addressing mode the effective address is sum of index register and displacement. Example,
MOV AX, [SI+2000]
MOV AL, [DI+3000]

h) **Based mode –** In this the effective address is the sum of base register and displacement. Example,
MOV AL, [BP+ 0100]

i) **Based indexed displacement mode –** In this type of addressing mode the effective address is the sum of index register, base register and displacement.
MOV AL, [SI+BP+2000]

j) **Input/Output mode –** This addressing mode is related with input output operations.

k) **Relative mode –** In this the effective address is calculated with reference to instruction pointer.
JNZ 8 bit address
IP=IP+8 bit address

8086 Instruction Set

LEA Instruction

- ✓ Used to load the effective address of operand into the provided register.
- ✓ Effective address is used to fetch operand.
- ✓ Format - LEA Register, Source
- ✓ Example:
  LEA BX, [DI] (This instruction load the contents of DI to BX register)

MUL Instruction

- ✓ Used to multiply unsigned byte by byte/word by word.
- ✓ It is always performed with accumulator.
- ✓ Only overflow and carry flag are affected.
- ✓ Format – MUL reg / MUL mem
- ✓ Example:
  MUL BX (AX=AX*BX)

IMUL Instruction

- ✓ Used to multiply signed byte by byte/word by word.
- ✓ It is always performed with accumulator.
- ✓ Format – IMUL reg / IMUL mem
- ✓ Example:
  I**MUL BX (AX=AX*BX)**

DIV Instruction

- ✓ Used to divide unsigned byte by byte/word by word.
- ✓ It is always performed with accumulator.
- ✓ Format – DIV reg / DIV mem
- ✓ Example:
  DIV BX (AX=AX/BX)

IDIV Insturction – Used to divide signed number**.**

LOOP Instruction

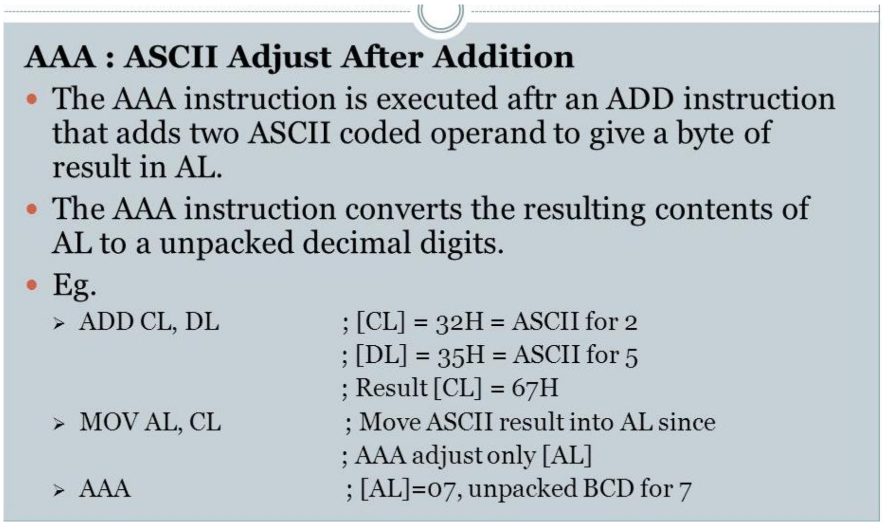
- ✓ Used to loop a group of instructions until the condition satisfies, i.e., CX = 0
- ✓ This intruction decremets CX by 1 and transfers control to the target location if CX is not 0; otherwise, the instruction following LOOP is executed.
- ✓ Format – *LOOP label*

AAA Instruction

- ✓ Used to adjust ASCII after addition.

AAS Instruction

- ✓ Used to adjust ASCII after subtraction.



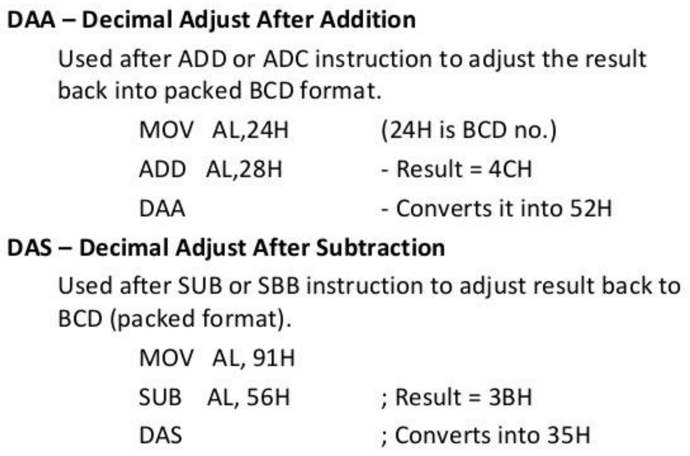
AAA : ASCII Adjust After Addition

- The AAA instruction is executed aftr an ADD instruction that adds two ASCII coded operand to give a byte of result in AL.
- The AAA instruction converts the resulting contents of AL to a unpacked decimal digits.
- Eg.

```
➢ ADD CL, DL          ; [CL] = 32H = ASCII for 2
                      ; [DL] = 35H = ASCII for 5
                      ; Result [CL] = 67H
➢ MOV AL, CL          ; Move ASCII result into AL since
                      ; AAA adjust only [AL]
➢ AAA                 ; [AL]=07, unpacked BCD for 7
```

DAA Instruction

- ✓ Used to adjust the decimal after the addition/subtraction operation.



DAA – Decimal Adjust After Addition

Used after ADD or ADC instruction to adjust the result back into packed BCD format.

```
MOV   AL,24H        (24H is BCD no.)
ADD   AL,28H        - Result = 4CH
DAA                 - Converts it into 52H
```

DAS – Decimal Adjust After Subtraction

Used after SUB or SBB instruction to adjust result back to BCD (packed format).

```
MOV   AL, 91H
SUB   AL, 56H       ; Result = 3BH
DAS                 ; Converts into 35H
```

<u>INT 21H functions</u>

INT 21h is the assembly language mnemonic for the Intel CPU command used to perform a System call to the operating system MS-DOS.
MS-DOS actually used quite a few different software interrupt vectors for system calls, but INT 21h was the main one.

Function number	**Description**
01h e.g. mov ah,01h int 21h	Keyboard input with echo: This operation accepts a character from the keyboard buffer. If none is present, waits for keyboard entry. It returns the character in AL.
02h e.g. mov ah,02h int 21h	Display character: Send the character in DL to the standard output device console.
09h e.g. mov ah,09h int 21h	String output: Send a string of characters to the standard output. DX contains the offset address of string. The string must be terminated with a '$' sign.
0Ah	String input
4Ch e.g. mov ax,4C00h int 21h	Terminate the current program (mov ah,4Ch int 21h is also used.)

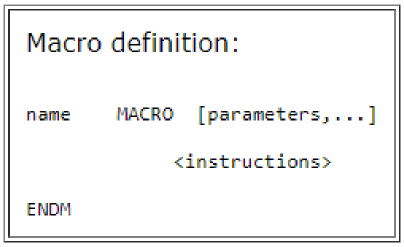
<u>INT 10H functions</u>

It is a video display vector interrupt used for Video display services (BIOS services).

Function number	**Description**
00h	Set video mode
01h	Set cursor size
02h	Set cursor position
06h	Scroll window up
07h	Scroll window down
08h	Read character and attribute of cursor
09h	Display character and attribute at cursor
0Ah	Display character at cursor

Macro Assemblers

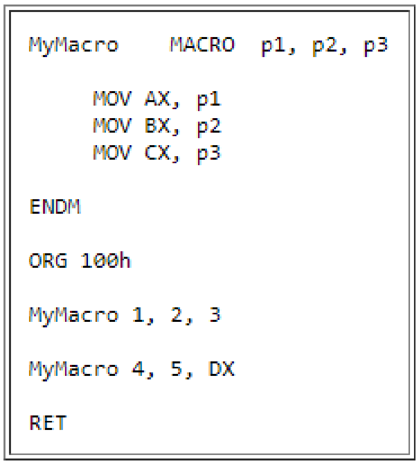
Macros are just like procedures, but not really. Macros look like procedures, but they exist only until your code is compiled, after compilation all macros are replaced with real instructions.
If you declared a macro and never used it in your code, compiler will simply ignore it.



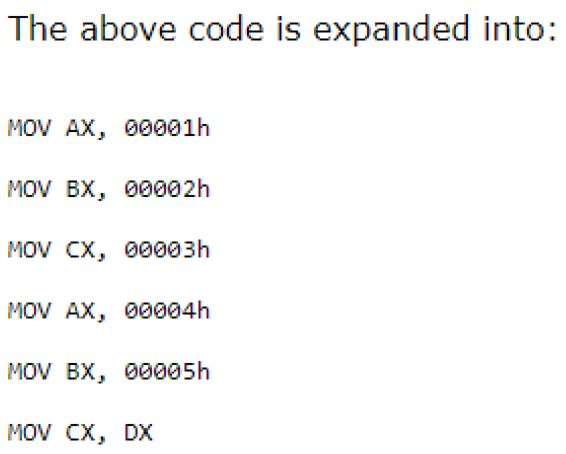
Macro definition:

```
name    MACRO  [parameters,...]

             <instructions>

ENDM
```

Unlike procedures, macros should be defined above the code that uses it, for example:



```
MyMacro    MACRO  p1, p2, p3

     MOV AX, p1
     MOV BX, p2
     MOV CX, p3

ENDM

ORG 100h

MyMacro 1, 2, 3

MyMacro 4, 5, DX

RET
```



The above code is expanded into:

```
MOV AX, 00001h

MOV BX, 00002h

MOV CX, 00003h

MOV AX, 00004h

MOV BX, 00005h

MOV CX, DX
```

Assembler Directives

An assembler directive is a message to the assembler that tells the assembler something it needs to know in order to carry out the assembly process; for example, an assemble directive tells the assembler where a program is to be located in memory.
Following are 8086 assembler directives:

(a) The DB directive

(b) The DW directive

(c) The DD directive

(d) The STRUCT (or STRUC) and ENDS directives (counted as one)

(e)The EQU Directive

(f)The COMMENT directive

(g)ASSUME

(h) EXTERN

(i) GLOBAL

(j) SEGMENT

(k)OFFSET

(l) PROC

(m)GROUP

(n) INCLUDE

1. **DB** – The DB directive is used to declare a BYTE – A BYTE is made up of 8 bits.

2. **DW** – The DW directive is used to declare a WORD type variable – A WORD occupies 16 bits or (2 BYTE).

3. **DD** – The DD directive is used to declare a DWORD – A DWORD double word is made up of 32 bits =2 Word's or 4 BYTE.

4. **STRUCT** and **ENDS** directives to define a structure template for grouping data items.

5. The **EQU** directive - is used to give name to some value or symbol. Each time the assembler finds the given names in the program, it will replace the name with the value or a symbol. The value can be in the range 0 through 65535.

6. **Extern** - It is used to tell the assembler that the name or label following the directive are some other assembly module.

7. **GLOBAL** - The GLOBAL directive can be used in place of PUBLIC directive .for a name defined in the current assembly module; the GLOBAL directive is used to make the symbol available to the other modules.

8. **SEGMENT** - It is used to indicate the start of a logical segment. It is the name given to the segment. Example: the code segment is used to indicate to the assembler the start of logical segment.

9. **PROC: (PROCEDURE)** - It is used to identify the start of a procedure. It follows a name we give the procedure.

10. **NAME** - It is used to give a specific name to each assembly module when program consists of several modules.

11. **INCLUDE** - It is used to tell the assembler to insert a block of source code from the named file into the current source module. This shortens the source module.

12. **OFFSET** - It is an operator which tells the assembler to determine the offset or displacement of a named data item from the start of the segment which contains it. It is used to load the offset of a variable into a register so that variable can be accessed with one of the addressed modes.

13. **GROUP** - It can be used to tell the assembler to group the logical segments named after the directive into one logical group. This allows the contents of all he segments to be accessed from the same group.

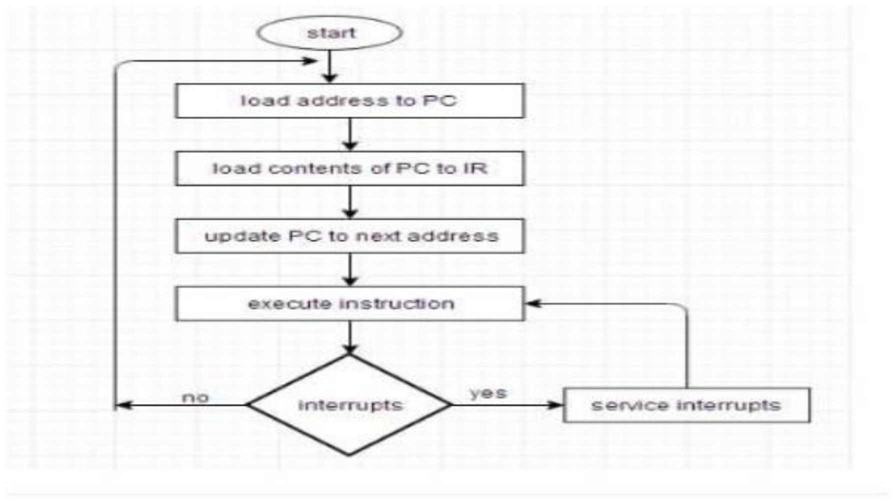
Unit III – Instruction Cycle

Instruction cycle in 8085 microprocessor

Time required to execute and fetch an entire instruction is called ***instruction cycle***. It consists:

- **Fetch cycle –** The next instruction is fetched by the address stored in program counter (PC) and then stored in the instruction register.
- **Decode instruction –** Decoder interprets the encoded instruction from instruction register.
- **Reading effective address –** The address given in instruction is read from main memory and required data is fetched. The effective address depends on direct addressing mode or indirect addressing mode.
- **Execution cycle –** consists memory read (MR), memory write (MW), input output read (IOR) and input output write (IOW)

General Flowchart for Instruction Cycle



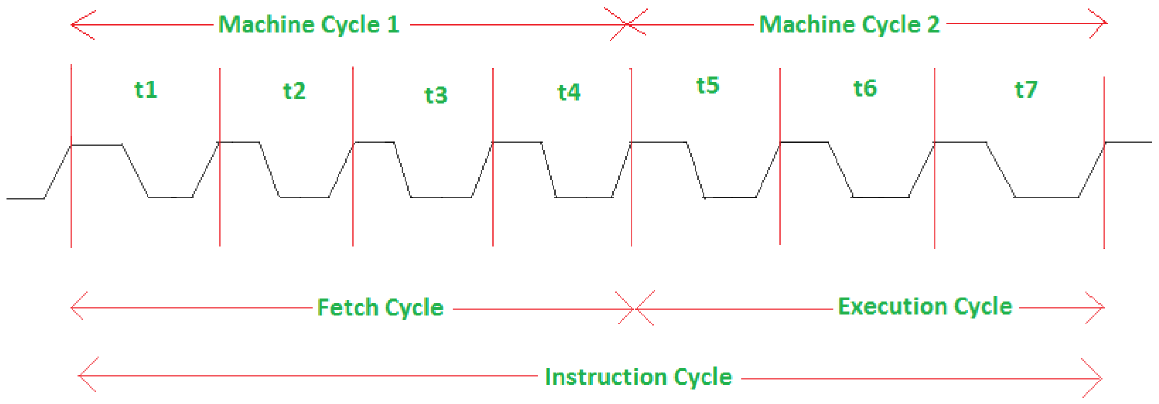
Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction. This process continues indefinitely unless a HALT instruction is encountered.

The time required by the microprocessor to complete an operation of accessing memory or input/output devices is called **machine cycle**. Following are the different types of machine cycle:

- **Opcode fetch**, which takes 4 t-states
- **Memory Read**, which takes 3 t-states
- **Memory Write**, which takes 3 t-states
- **I/O Read**, which takes 3 t-states
- **I/O Write**, which takes 3 t-states

One-time period of frequency of microprocessor is called **t-state**. A t-state is measured from the falling edge of one clock pulse to the falling edge of the next clock pulse.

Fetch cycle takes four t-states and execution cycle takes three t-states. It is shown below:



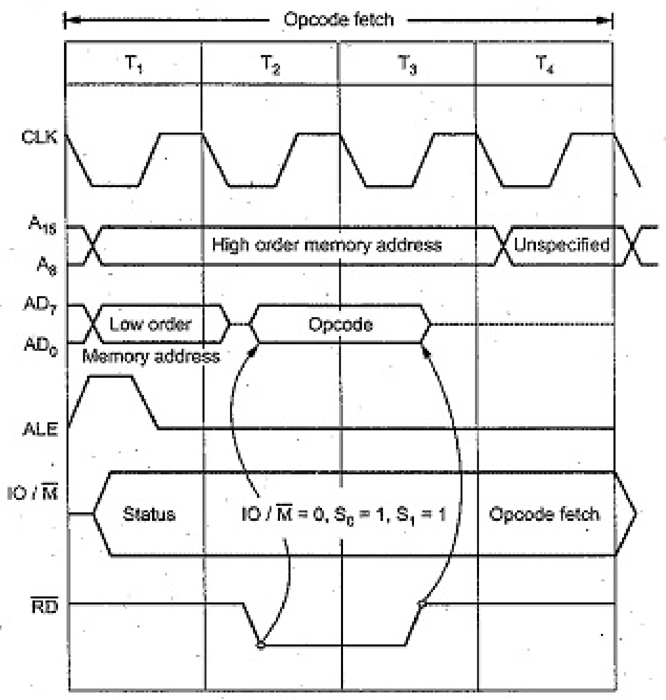
Instruction cycle in 8085 microprocessor

Different types of Machine Cycles:

Opcode Fetch Cycle

The first machine cycle of every instruction is **opcode fetch cycle** in which the 8085 finds the nature of the instruction to be executed. In this machine cycle, processor places the contents of the Program Counter on the address lines, and through the read process, reads the opcode of the instruction. The length of this cycle is not fixed. It varies from 4T states to 6T states as per the instruction.

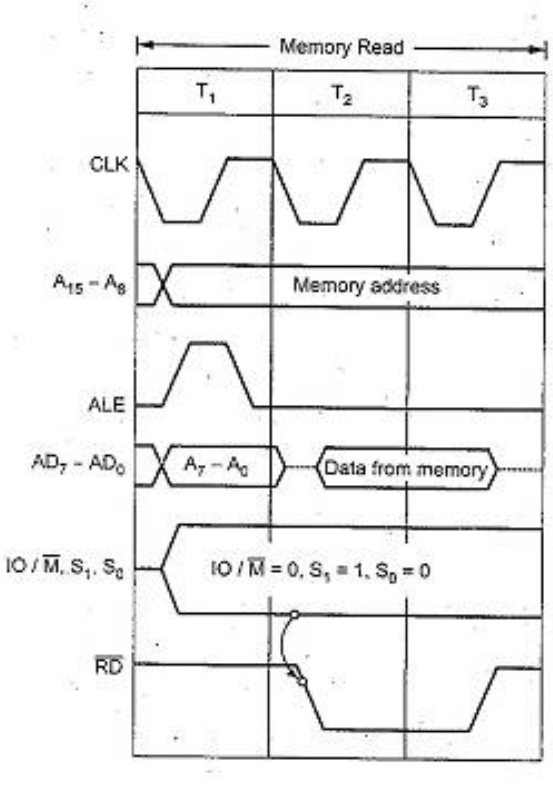
Below figure shows timing diagram of opcode fetch:



Memory Read Cycle

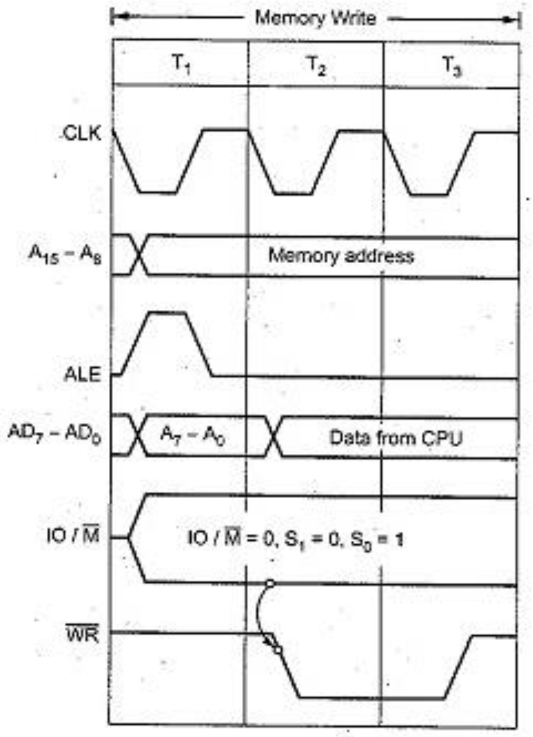
The 8085 executes the memory read cycle to read the contents of R/W memory or ROM. The length of this machine cycle is 3-T states (T1 – T3). In this machine cycle, processor places the address on the address lines from the stack pointer, general purpose register pair or program counter, and through the read process, reads the data from the addressed memory location.

Timing diagram is shown below:



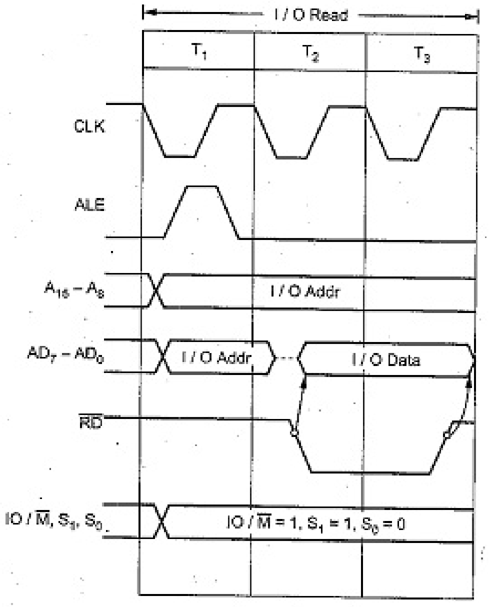
Memory Write Cycle

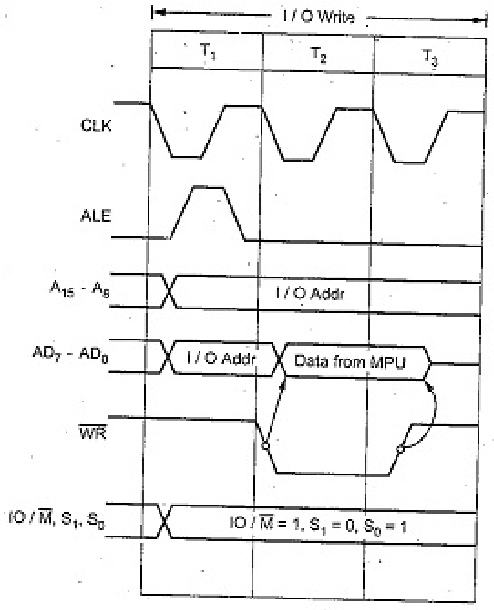
The 8085 executes the memory write cycle to store the data into data memory or stack memory. The length of this machine cycle is 3T states. (T1 — T3). In this machine cycle, processor places the address on the address lines from the stack pointer or general purpose register pair and through the write process, stores the data into the addressed memory location.



I/O Read and I/O Write cycles

The I/O read and I/O write machine cycles are similar to the memory read and memory write machine cycles, respectively, except that the I0/M signal is high for I/O read and I/O write machine cycles. High IO/M signal indicates that it is an I/O operation.





Timing Diagrams

Draw timing diagrams for following instructions:

a) MOV
b) MVI
c) IN
d) OUT
e) LDA
f) STA

Memory Interfacing

As we know that any system which process digital data needs the facility for storing the data. Interfacing is a technique to be used for connecting the Microprocessor to Memory.

Now a days Semiconductor memories are used for storing purpose. There are some of the advantages of the semiconductor memory.

- Small size
- High speed
- Better reliability
- Low cost

Generally, RAM or ROM is used for memory interfacing.

Memory: -A memory is a digital IC which stores the data in binary form.

Memory Size: - The number of location and number of bits per word will vary from memory to memory. For example, if a particular memory chip is capable of storing M words with each word having N-bits. Then the size of the memory will be M× N.

Interfacing a ROM memory of 4096*8 with 8085 Microprocessor: -

Given memory size = 4096 * 8   (4k*8)
4096 =2^12.
So 12 lines will be used for interfacing. A0 to A11

In this system A0 to A11 lines of Microprocessor will be connected to the address lines of the memory. and D0 to D7 of the 8085 microprocessor will be connected to the data bus of the memory.
As we know that the it is EPROM, so only RD pin is connected to the microprocessor. There is not the facility for writing data.

In case if you are using RAM then you have to connect one more pin for writing operation.

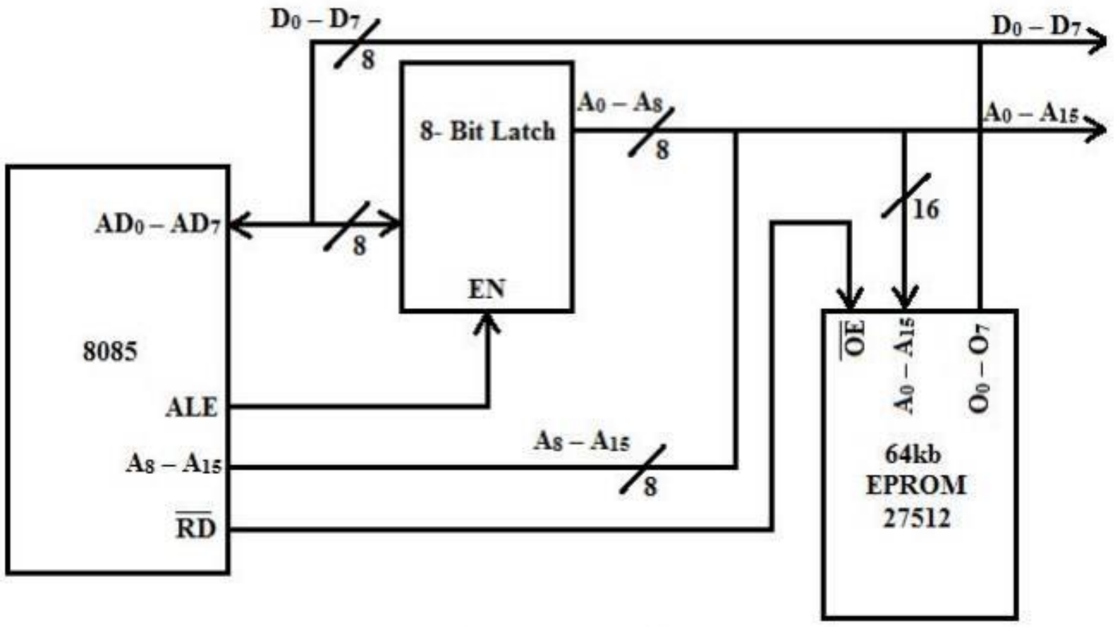


Fig 1.6 Memory Interfacing

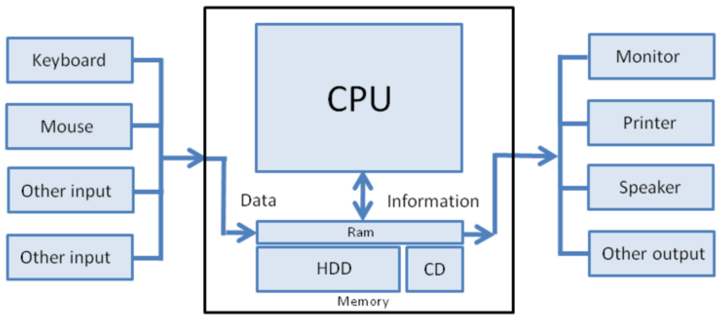
As we can see that there is a pin named CS. Generally, this pin is used for Selection for the chip in case of two or more than memory chip.
Latch has been used to separate the data and address bus.

UNIT V – Basic I/O, Memory R/W & Interrupt Operations

I/O Operations

CPU uses two methods to perform input/output operations between the CPU and peripheral devices in the computer. These two methods are called **memory mapped IO** and **IO mapped IO**.

- **Memory-mapped IO** uses the same address space to address both memory and I/O devices.
- On the other hand, **IO mapped IO** uses separate address spaces to address memory and IO devices.



Memory Mapped I/O

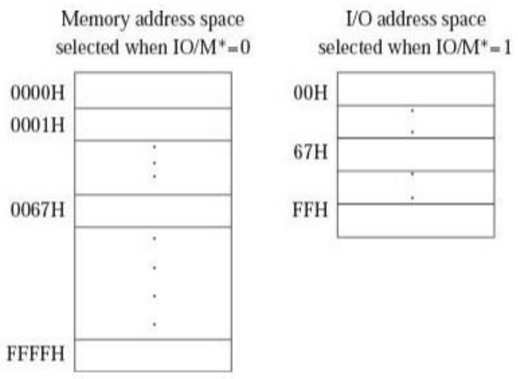
Memory mapped IO uses one address space for memory and input and output devices. In other words, some addresses are assigned to memory while others are assigned to store the addresses of IO devices.

There is one set of read and write instruction lines. The same set of instructions work for both memory and IO operations. Therefore, the instructions used to manipulate memory can be used for IO devices too. Hence, it can lessen the addressing capability of memory because some are occupied by the IO.

I/O Mapped I/O

IO mapped IO uses two separate address spaces for memory locations and for IO devices. There are two separate control lines for both memory and IO transfer. In other words, there are different read-write instruction for both IO and memory.

IO read and IO write are for IO transfer whereas memory read and memory write are for memory transfer. IO mapped IO is also called port-mapped IO or isolated IO.


Memory address space selected when IO/M*=0
I/O address space selected when IO/M*=1
0000H
0001H
0067H
FFFFH
00H
67H
FFH

Memory Mapped Vs I/O Mapped I/O

1) Address Spaces

- The main difference between memory mapped IO and IO mapped IO is that the memory mapped IO uses the same address space for both memory and IO devices. IO mapped IO uses two separate address spaces for memory and IO device.

2) Addresses for Memory

- Branching from the above, there is another difference between memory mapped IO and IO mapped IO. As the memory mapped IO uses one address space for both IO and memory, the available addresses for memory are minimum due to the additional addresses for IO. In IO mapped IO, all the addresses can be used by the memory.

3) Instructions

- While memory mapped IO uses the same instructions for both IO and memory operations, IO mapped IO uses separate instructions for read and write operations in IO and memory. We can say this as one other difference between memory mapped IO and IO mapped IO.

4) Efficiency

- Moreover, memory mapped IO is less efficient while IO mapped IO is more efficient.

Hybrid I/O

- Combination of both previous I/O.
- Uses same address in some cases and two address space in some cases.

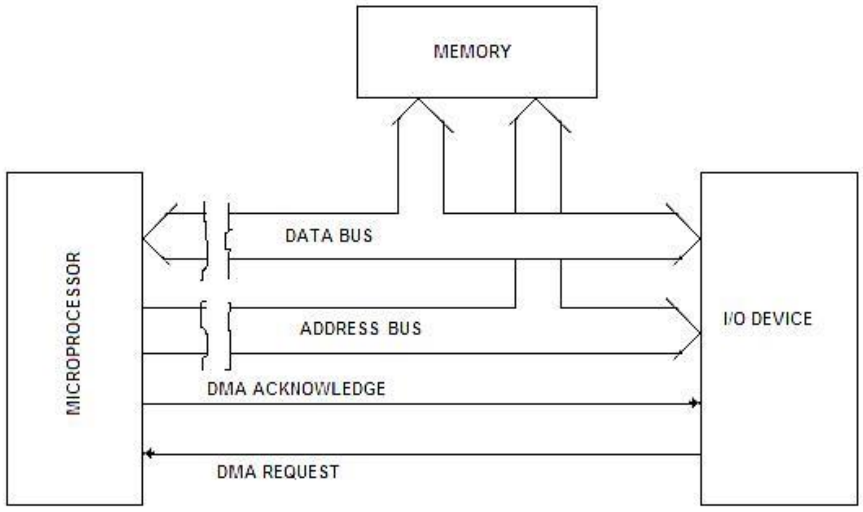
Conclusion

- Memory mapped IO and IO mapped IO are two methods to perform input/output operations between the CPU and peripheral devices in the computer.
- The basic difference between memory mapped IO and IO mapped IO is that memory mapped IO uses the same address space for both memory and IO device while IO mapped IO uses two separate address spaces for memory and IO device.

Direct Memory Access (DMA)

- **Direct memory access (DMA)** is a method that allows an input/output (I/O) device to send or receive data directly to or from the main memory, bypassing the CPU to speed up memory operations.
- The process is managed by a chip known as a **DMA controller (DMAC).**
- It moves the data between memory and peripheral without bothering the CPU. This means it frees up both CPU power and bandwidth between CPU and main memory.
- For a CPU based copy, the data has to go into and out of the CPU, which is often a slow process and blocks both the core doing the copy and the other cores. And if the data flow has gaps, as it will in the case of HDD and network, the CPU can sit there wasting time until more data becomes available.
- If the data rate is very low, DMA is overkill. But if the data is either fast or large, DMA offloads the CPU and lets it get on with useful work.
- The problem of slow data transfer between input-output port and memory or between two memories is avoided by implementing Direct Memory Access (DMA) technique.

- This is faster as the microprocessor/computer is bypassed and the control of address bus and data bus is given to the DMA controller.



Advantages & Disadvantages of DMA

Advantages

- DMA allows a peripheral device to read from/write to memory without going through the CPU.
- DMA allows faster processing since the processor can be working on something else while the peripheral can be populating memory.
- DMA enables more efficient use of interrupts.
- High transfer rates.
- DMA capable device can communicate directly with memory.

Disadvantages

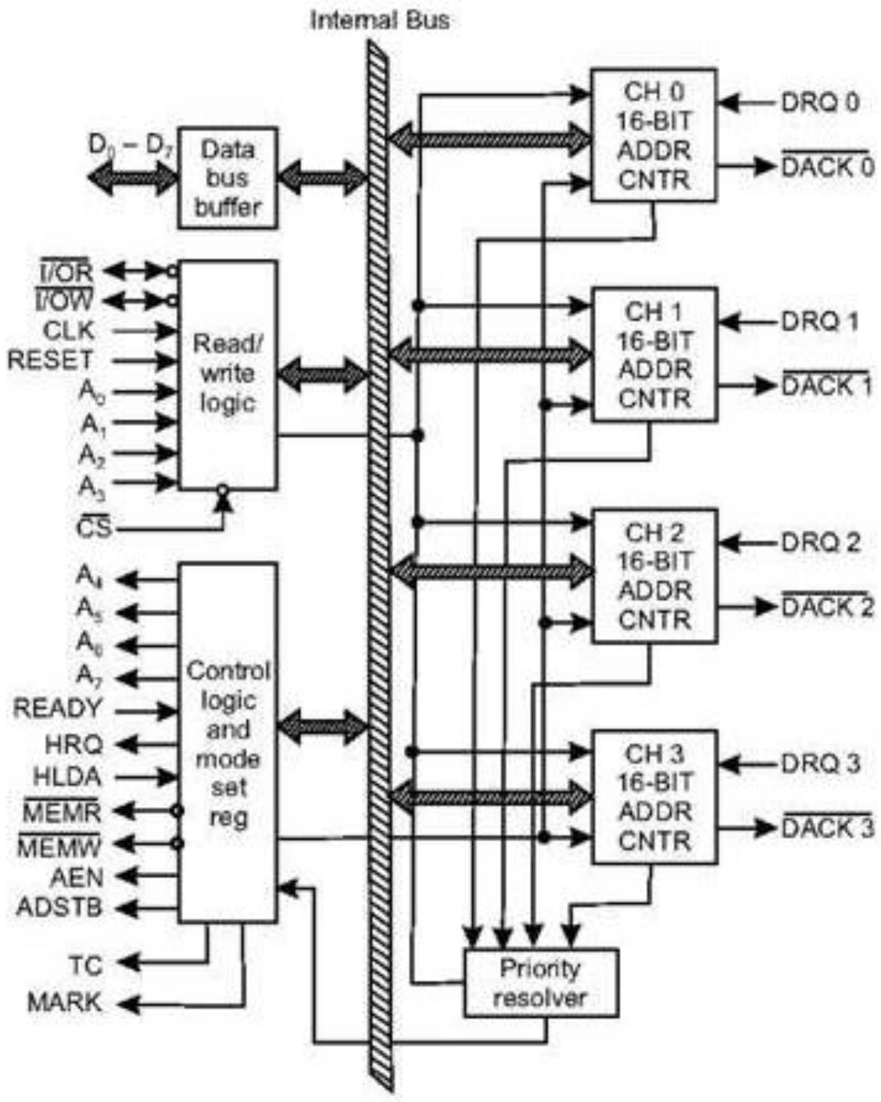
- Cost of DMA hardware.
- Data has to be stored in continuous memory locations.
- DMA controller is slow in comparison to CPU.

Application of DMA

- DMA has been a built-in feature of PC architecture since the introduction of the original IBM PC.
- PC-based DMA was used for floppy disk I/O in the original PC and for hard disk I/O in later versions.
- PC-based DMA technology, along with high speed bus technology, is driven by data storage, communications, and graphics needs-all of which require the highest rates of data transfer between system memory and I/O devices.
- Applications areas are: **cinemas, theatres, hotels, railway stations, shopping centres, trade shows, museums & many more.**

8237 DMA Controller

- A DMA controller interfaces with several peripherals that may request DMA.
- The controller decides the priority of simultaneous DMA requests communicates with the peripheral and the CPU, and provides memory addresses for data transfer.
- DMA controller commonly used with 8088/8085 is the 8237 programmable device.
- The 8237 is in fact a special-purpose microprocessor.
- Normally it appears as part of the system controller chip-sets.
- The 8237 is a 4-channel device.
- Each channel is dedicated to a specific peripheral device and capable of addressing 64 K bytes' section of memory.



How DMA Operations Are Performed?

Following are the sequence of operations performed by a DMA −

- Initially, when any device has to send data between the device and the memory, the device has to send DMA request (DRQ) to DMA controller.
- The DMA controller sends Hold request (HRQ) to the CPU and waits for the CPU to assert the HLDA.
- Then the microprocessor tri-states all the data bus, address bus, and control bus. The CPU leaves the control over bus and acknowledges the HOLD request through HLDA signal.
- Now the CPU is in HOLD state and the DMA controller has to manage the operations over buses between the CPU, memory, and I/O devices.

Interrupt

- 8085 Interrupt pins & priority. (Discussed already)
- Maskable and Non-maskable interrupt (Discussed already)
- **RST Instructions –**

Instruction	**Op Code**	**Binary equivalent**
RST 0	C7	1100 0111
RST 1	CF	1100 1111
RST 2	D7	1101 0111
RST 3	DF	1101 1111
RST 4	E7	1110 0111
RST 5	EF	1110 1111
RST 6	F7	1111 0111
RST 7	FF	1111 1111

Vector Interrupt

- In a computer, a vectored interrupt is an I/O interrupt that tells the part of the computer that handles I/O interrupts at the hardware level that a request for attention from an I/O device has been received and also identifies the device that sent the request.
- Vectored Interrupts are those which have fixed vector address (starting address of sub-routine) and after executing these, program control is transferred to that address.
- Vector Addresses are calculated by the formula 8 * TYPE

INTERRUPT	VECTOR ADDRESS
TRAP (RST 4.5)	24 H
RST 5.5	2C H
RST 6.5	34 H
RST 7.5	3C H

Polled Interrupt

- A polled interrupt is a certain kind of input/output (I/O) interrupt that sends a message to the part of the computer that houses the I/O interface.
- The message states that a device is ready to be accessed without an identifying device.
- In a computer, a polled interrupt is a specific type of I/O interrupt that notifies the part of the computer containing the I/O interface that a device is ready to be read or otherwise handled but does not indicate which device.
- The interrupt controller must poll (send a signal out to) each device to determine which one made the request.

8259 Interrupt Controller

- **Intel 8259** is designed for Intel 8085 and Intel 8086 microprocessor.
- 8259 is defined as **Programmable Interrupt Controller (PIC)**.
- There are 5 hardware interrupts and 2 hardware interrupts in 8085 and 8086 respectively. But by connecting 8259 with CPU, we can increase the interrupt handling capability. 8259 combines the multi interrupt input sources into a single interrupt output.
- Interfacing of single PIC provides 8 interrupts inputs from IR0-IR7.
- For example, **Interfacing of 8085 and 8259** increases the interrupt handling capability of 8085 microprocessor from **5 to 8** interrupt levels.

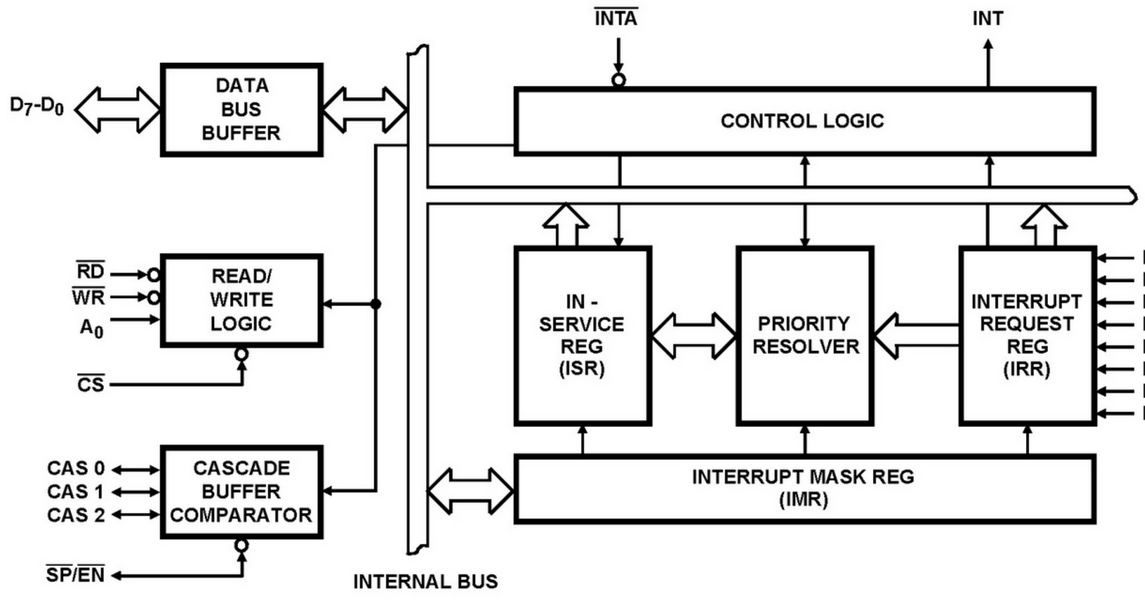


Fig: 8259 Interrupt Controller block diagram

The Block Diagram consists of 8 blocks which are – Data Bus Buffer, Read/Write Logic, Cascade Buffer Comparator, Control Logic, Priority Resolver and 3 registers- ISR, IRR, IMR.

1) Data bus buffer –

- This Block is used as a mediator between 8259 and 8085/8086 microprocessor by acting as a buffer. The data bus buffer consists of 8 bits represented as D0-D7 in the block diagram. Thus, shows that a maximum of 8 bits data can be transferred at a time.

2) Read/Write logic –

- This block is responsible for the flow of data depending upon the inputs of RD and WR. These two pins are active low pins used for read and write operations.

3) Control logic –

- It is the centre of the microprocessor and controls the functioning of every block. It has pin INTR which is connected with other microprocessor for taking interrupt request and pin INT for giving the output.

4) Interrupt request register (IRR) –

- It stores all the interrupt level which are requesting for Interrupt services.

5) Interrupt service register (ISR) –

- It stores the interrupt level which are currently being executed.

6) Interrupt mask register (IMR) –

- It stores the interrupt level which have to be masked by storing the masking bits of the interrupt level.

7) Priority resolver –

- It examines all the three registers and set the priority of interrupts and according to the priority of the interrupts, interrupt with highest priority is set in ISR register. Also, it reset the interrupt level which is already been serviced in IRR.

8) Cascade buffer –

- To increase the Interrupt handling capability, we can further cascade more number of pins by using cascade buffer. So, during increment of interrupt capability, CSA lines are used to control multiple interrupt structure.

8259 Pin Diagram

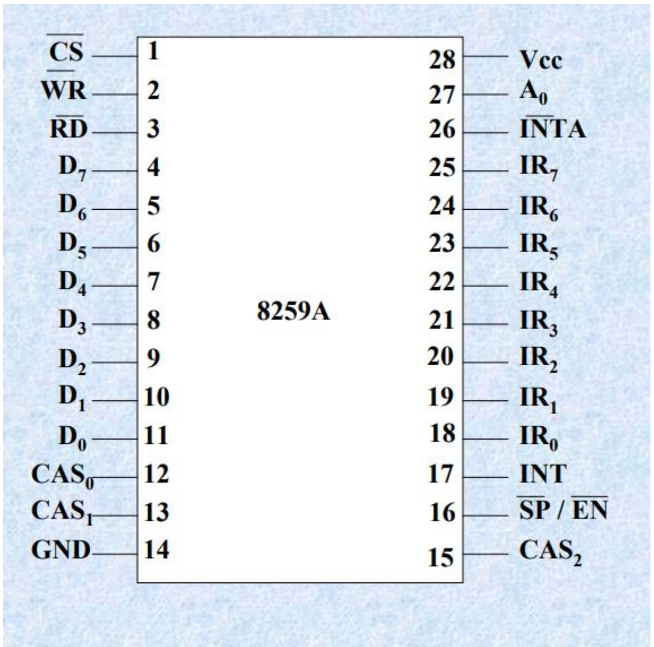


Fig : 8259 Pin Diagram

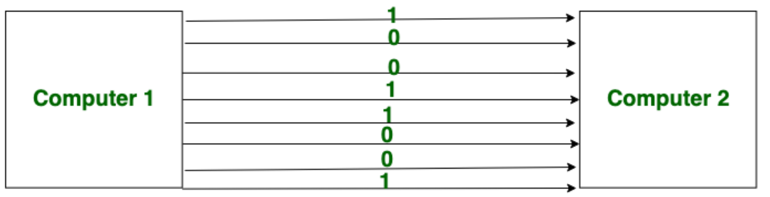
Priority Modes of 8259

- **Priority Resolver** determines the priorities of the interrupt requests appearing simultaneously.
- The **highest priority** is selected and stored into the corresponding bit of **ISR** during **INTA** pulse.
- The **IR 0** has the highest priority while the **IR 7** has the lowest one, normally in **fixed priority** mode.
- The priorities however may be **altered** by **programming the 8259A** in **rotating priority mode.**

UNIT VI – I/O Interfaces

Parallel Communication

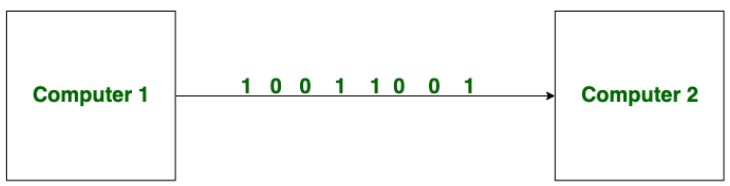
- In data transmission, parallel communication is a method of **conveying multiple binary digits (bits)** simultaneously. It contrasts with **serial communication**, which conveys only a **single bit** at a time.
- In Parallel Transmission, many bits are flow together simultaneously from one computer to another computer.
- Parallel Transmission is faster than serial transmission to transmit the bits. Parallel transmission is used for long distance.



Parallel Transmission

Serial Communication

- **Serial communication** is the process of sending data **one bit at a time**, **sequentially**, over a communication channel or computer bus.
- In **Serial Transmission**, data-bit flows from one computer to another computer in bi-direction. In this transmission **one bit flows at one clock pulse**.
- In Serial Transmission, 8 bits are transferred at a time having a start and stop bit.



Serial Transmission

Serial Vs Parallel Communication

S.NO	SERIAL TRANSMISSION	PARALLEL TRANSMISSION
1.	In serial transmission, data(bit) flows in bi-direction.	In Parallel Transmission, data flows in multiple lines.
2.	Serial Transmission is cost efficient.	Parallel Transmission is not cost efficient.
3.	In serial transmission, one bit transferred at one clock pulse.	In Parallel Transmission, eight bits transferred at one clock pulse.
4.	Serial Transmission is slow in comparison of Parallel Transmission.	Parallel Transmission is fast in comparison of Serial Transmission.
5.	Generally, Serial Transmission is used for short distance.	Generally, Parallel Transmission is used for long distance.

Programmable Communication Interface 8251

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

Block Diagram of 8251 PCI

The block diagram includes five section:

1. Read/write Control Logic
2. Transmitter
3. Receiver
4. Data Bus Buffer and
5. Modem Control

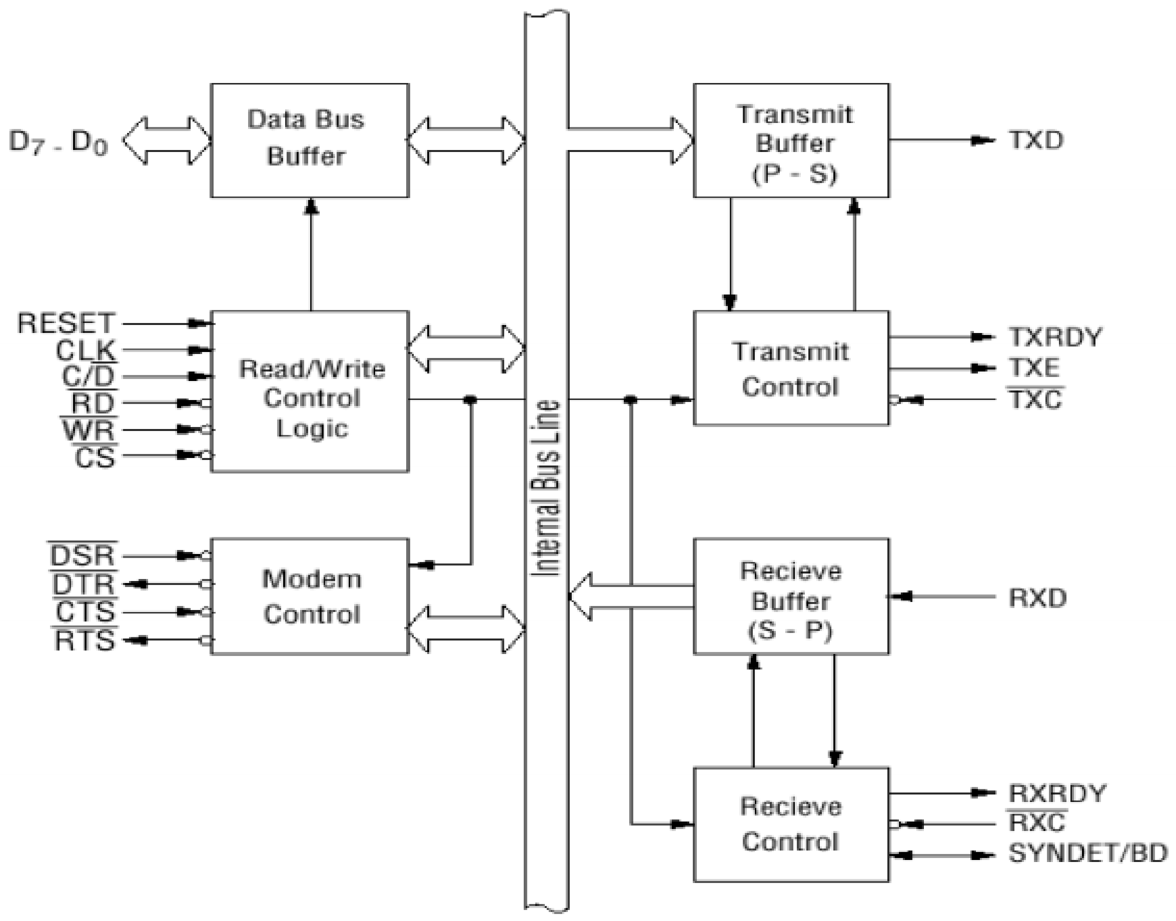


Fig: Block Diagram of 8251 PCI

1. Read/Write Control Logic

- The control logic interfaces the chip with the processor, and monitors the data flow. It controls the overall working by selecting the operation to be done.

2. 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.

3. Transmit Buffer

- This block is used for parallel to serial converter that receives a parallel byte for conversion into serial signal and further transmission onto the common channel.
  - **TXD**: It is an output signal, if its value is one, means transmitter will transmit the data.

4. Transmit control

- This block is used to control the data transmission with the help of following pins:
  - **TXRDY**: It means transmitter is ready to transmit data character.
  - **TXEMPTY**: An output signal which indicates that TXEMPTY pin has transmitted all the data characters and transmitter is empty now.

  - **TXC**: An active-low input pin which controls the data transmission rate of transmitted data.

5. Receive buffer

- This block acts as a buffer for the received data.
  - **RXD**: An input signal which receives the data.

6. Receive control

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

7. Modem control (modulator/demodulator)

- A device converts analog signals to digital signals and vice-versa and helps the computers to communicate over telephone lines or cable wires. The following are active-low pins of Modem.
  - **DSR**: Data Set Ready signal is an input signal.
  - **DTR**: Data terminal Ready is an output signal.
  - **CTS**: It is an input signal which controls the data transmit circuit.
  - **RTS**: It is an output signal which is used to set the status RTS.

Modes of Serial Communication

- As we know in Serial Transmission data is sent bit by bit, in such a way that each bit follows another. It is of two types namely, **Synchronous and Asynchronous Transmission**.
- One of the major differences is that in **Synchronous Transmission**, the **sender and receiver** should have **synchronized clocks** before data transmission.
- Whereas **Asynchronous Transmission** does not require a clock, but it adds a parity bit to the data before transmission.
- **Asynchronous** is simple, economical and used for transmitting a small amount of data.
- Conversely, **Synchronous Transmission** is used for transferring the bulk of data as it is efficient and has less overhead.

Synchronous Vs Asynchronous Mode

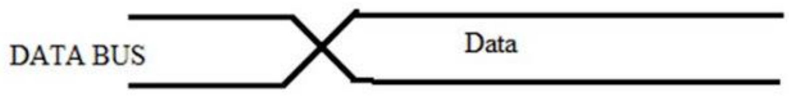
- In Synchronous Transmission, data is transferred in the form of frames. On the other hand, in Asynchronous Transmission data is transmitted 1 byte at a time.
- Synchronous Transmission requires a clock signal between the sender and receiver so as to inform the receiver about the new byte. In contrast, in Asynchronous Transmission sender and receiver does not require a clock signal as the data sent here has a parity bit attached to it which indicates the start of the new byte.
- Data transfer rate of Asynchronous Transmission is slower than that of Synchronous Transmission.

- Asynchronous Transmission is simple and economical, whereas Synchronous Transmission is complicated and expensive.
- Synchronous Transmission is efficient and has lower overhead as compared to the Asynchronous Transmission.

Methods of parallel data transfer

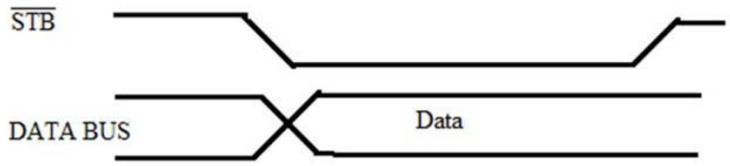
1) Simple I/O

- When we need to get digital data from input device, into microprocessor, all we have to do is connect the switch to an I/O port line and read the port.
- Likewise, when we need to output data to simple display device, such as LED, all we have to do is connect the input of the LED buffer on an output port pin.



2) Strobe I/O

- In many applications, valid data is present on an external device only at a certain time, so it must be read in at that time.
- E.g. the ASCII-encoded keyboard. When a key is pressed, circuitry on the keyboard sends out the ASCII code for the pressed key on eight parallel data lines, and then sends out a strobe signal on another line to indicate that valid data is present on the eight data lines.



8255A Programmable Peripheral Interface

- **PPI 8255** is a general purpose programmable I/O device designed to interface the CPU with its outside world such as keyboard.
- We can program it according to the given condition. It can be used with almost any microprocessor.
- It consists of **three 8-bit bidirectional I/O ports i.e. PORT A, PORT B and PORT C**. We can assign different ports as input or output functions.
- 8255A has three ports, i.e., PORT A, PORT B, and PORT C.
  - **Port A** contains one 8-bit output latch/buffer and one 8-bit input buffer.
  - **Port B** is similar to PORT A.
  - **Port C** can be split into two parts, i.e. PORT C lower (PC0-PC3) and PORT C upper (PC7-PC4) by the control word.

8255A Block Diagram

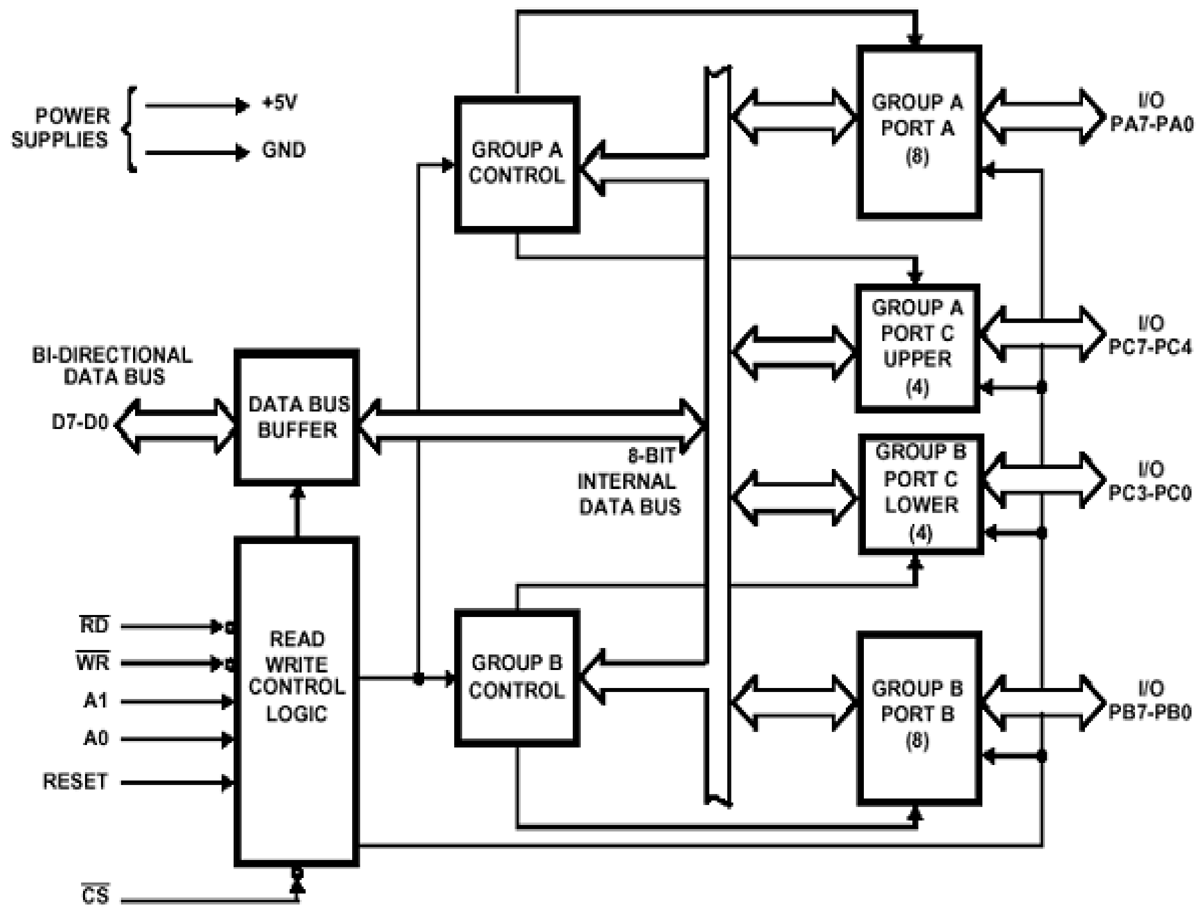


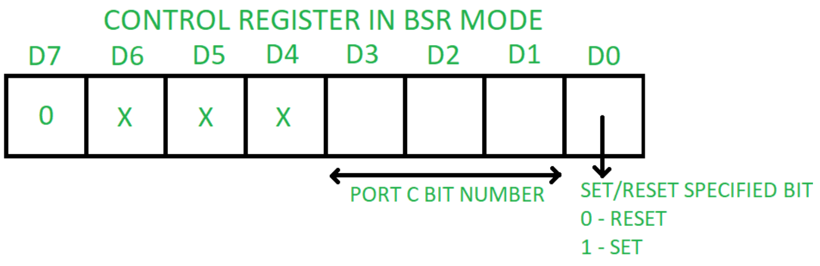
Fig: Block Diagram of 8255A PPI

- It consists of 40 pins and operates in +5V regulated power supply.
- Port C is further divided into two 4-bit ports i.e. port C lower and port C upper and port C can work in either BSR (bit set rest) mode or in mode 0 of input-output mode of 8255.
- Port B can work in either mode or in mode 1 of input-output mode.
- Port A can work either in mode 0, mode 1 or mode 2 of input-output mode.
- It has two control groups, control group A and control group B. Control group A consist of port A and port C upper. Control group B consists of port C lower and port B.

Operating modes of 8255A

1) Bit set reset (BSR) mode –

- If MSB of control word (D7) is 0, PPI works in BSR mode. In this mode only port C bits are used for set or reset.
- The contents of the **control register** are called the **control word** that specifies the input/ output functions of each port.

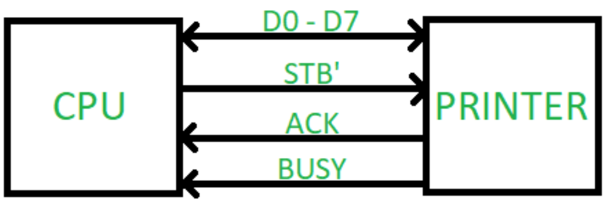


2) Input/output Mode

There are three types of the input/output mode. They are as follows:

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

- **Mode 1**
- In this mode either port A or port B can work as simple input port or simple output port, and port C bits are used for handshake signals before actual data transmission. It has interrupt handling capacity.
- Example: A CPU wants to transfer data to a printer. In this case since speed of processor is very fast as compared to relatively slow printer, so before actual data transfer it will send handshake signals to the printer for synchronization of the speed of the CPU and the peripherals.



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

RS-232

- RS232 is a standard protocol used for serial communication, it is used for connecting computer and its peripheral devices to allow serial data exchange between them.

- In simple terms RS232 defines the voltage for the path used for data exchange between the devices. It specifies common voltage and signal level, common pin wire configuration and minimum, amount of control signals.

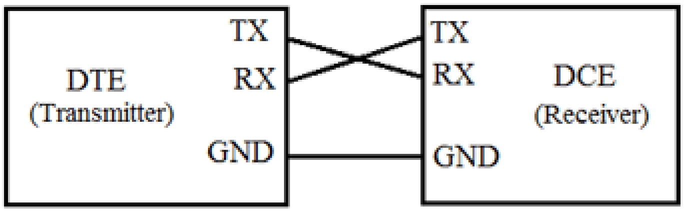
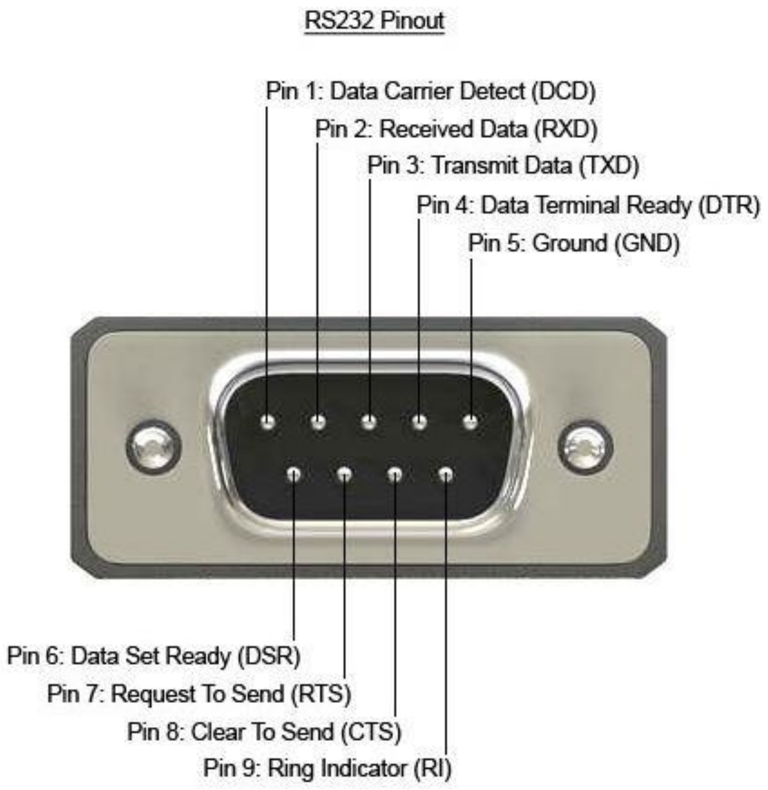


Fig: RS-232

- **DTE** - A DTE stands for data terminal equipment is an end instrument that convert user information into signals or reconverts the receive signal. A male connector is used in DTE and has pin out configuration.

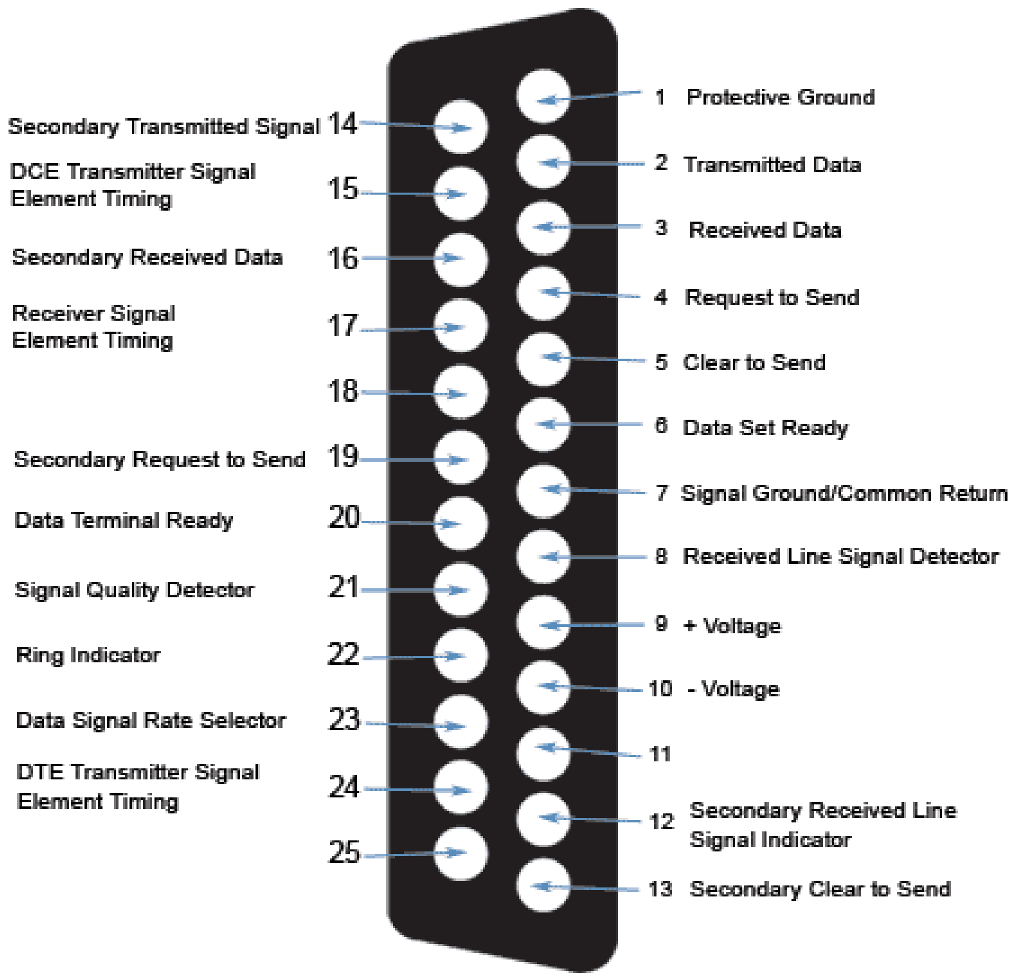
- **DCE** - A DCE stands for data communication equipment's. It sits between the DTE and data transmission circuit for example modem. A DCE device uses a female connector which has holes on the surface to hold male connector.

RS-232 Pin Configuration – 9 pin MALE Connector


RS232 Pinout
Pin 1: Data Carrier Detect (DCD)
Pin 2: Received Data (RXD)
Pin 3: Transmit Data (TXD)
Pin 4: Data Terminal Ready (DTR)
Pin 5: Ground (GND)
Pin 6: Data Set Ready (DSR)
Pin 7: Request To Send (RTS)
Pin 8: Clear To Send (CTS)
Pin 9: Ring Indicator (RI)

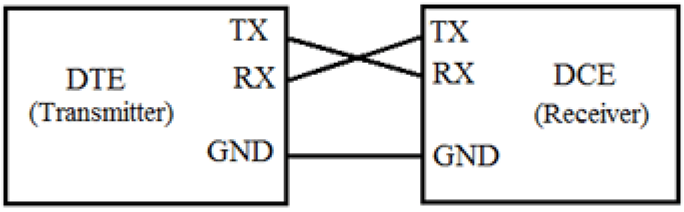
PIN No.	Pin Name	Pin Description
1	CD (Carrier Detect)	Incoming signal from DCE
2	RD (Receive Data)	Receives incoming data from DTE
3	TD (Transmit Data)	Send outgoing data to DCE
4	DTR (Data Terminal Ready)	Outgoing handshaking signal
5	GND (Signal ground)	Common reference voltage
6	DSR (Data Set Ready)	Incoming handshaking signal
7	RTS (Request to Send)	Outgoing signal for controlling flow
8	CTS (Clear to Send)	Incoming signal for controlling flow
9	RI (Ring Indicator)	Incoming signal from DCE

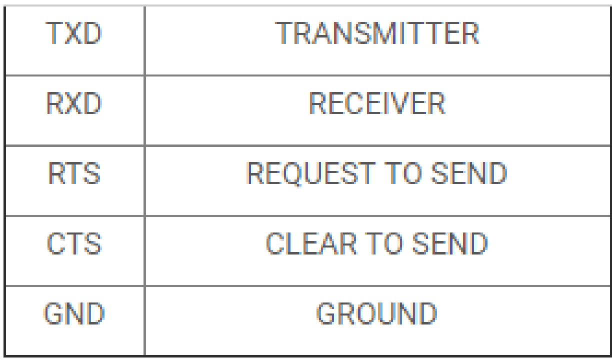
RS-232 Pin Configuration – 25 pin


1 Protective Ground
2 Transmitted Data
3 Received Data
4 Request to Send
5 Clear to Send
6 Data Set Ready
7 Signal Ground/Common Return
8 Received Line Signal Detector
9 + Voltage
10 - Voltage
11
12 Secondary Received Line Signal Indicator
13 Secondary Clear to Send
Secondary Transmitted Signal 14
DCE Transmitter Signal Element Timing 15
Secondary Received Data 16
Receiver Signal Element Timing 17
18
Secondary Request to Send 19
Data Terminal Ready 20
Signal Quality Detector 21
Ring Indicator 22
Data Signal Rate Selector 23
DTE Transmitter Signal Element Timing 24
25

How RS232 Works? – Interconnection between DTE-DCE

- **RS232** works on the **two-way communication** that exchanges data to one another. There are two devices connected to each other, (**DTE) Data Transmission Equipment& (DCE) Data Communication Equipment** which has the pins like **TXD, RXD, and RTS & CTS**.
- Now, from **DTE source, the RTS generates** the request to send the data. Then from the other side DCE, the CTS, **clears the path for receiving the data**.
- After clearing a path, it will give a signal to **RTS of the DTE** source to send the signal. Then the bits are transmitted from **DTE to DCE**.
- Now again from **DCE source**, the request can be generated by **RTS and CTS of DTE** sources clears the path for receiving the data and gives a signal to send the data.
- This is the whole process through which data transmission takes place.





TXD	TRANSMITTER
RXD	RECEIVER
RTS	REQUEST TO SEND
CTS	CLEAR TO SEND
GND	GROUND

Application of RS-232

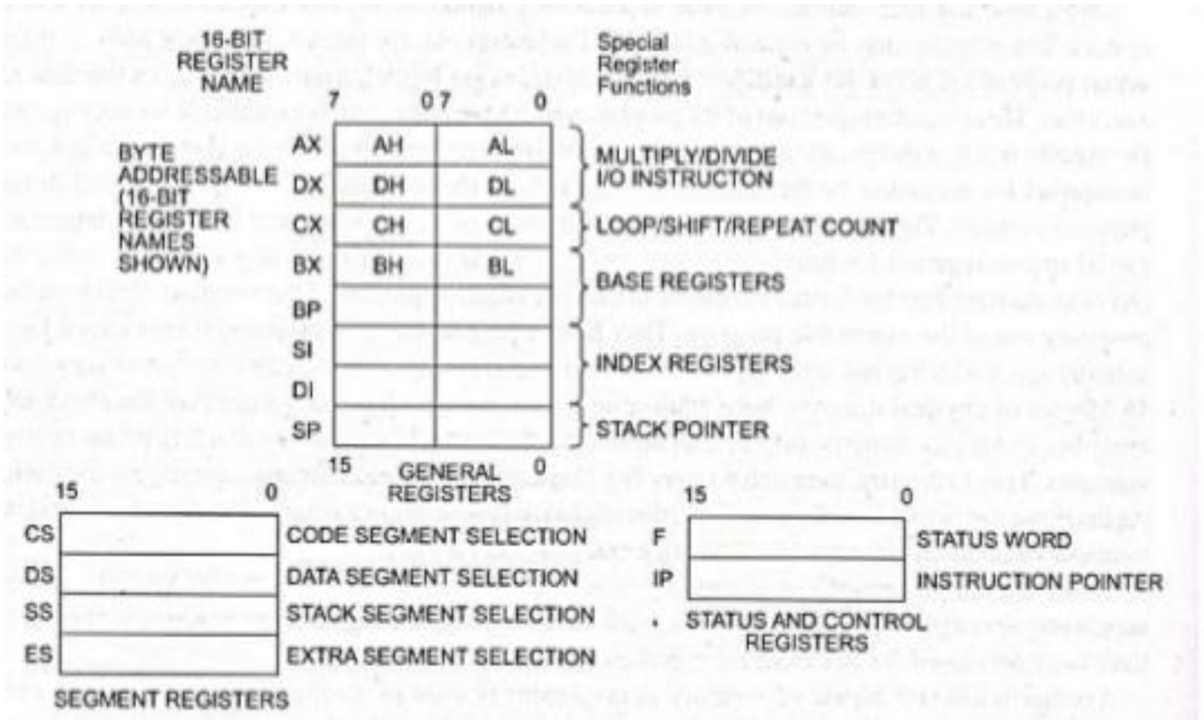
- RS232 serial communication is used in old generation PCs for connecting the peripheral devices like mouse, printers, modem etc.
- Nowadays, RS232 is replaced by advanced USB.
- It is still used by some microcontroller boards, receipt printers, point of sale system (PoS), etc.

UNIT VII – Advanced Microprocessors

80286 Microprocessor

- The **Intel 80286** is a high-performance 16-bit microprocessor introduced in 1982.
- It has been specially designed for multiuser and multitasking systems.
- 80286 is upwardly compatible with 8086 in terms of instruction set. (That is the 8086,8088,80186,80286 CPU family all contain the same instruction set)
- It has **24 address lines** and **16 data lines**.
- There are two operating modes for 80286
  - The **real** address mode
  - The **protected** virtual memory address mode
- In **real** address mode the processor can address upto **1 MB** of physical memory.
- The **virtual** address mode is for multiuser/multitasking system. In this mode the processor can address upto **1 GB** of virtual memory.

80286 Register Set



The 80286 CPU contains almost the same set of registers, as in 8086.

a) Eight 16-bit general purpose registers.
b) Four 16-bit segment registers.
c) Status and control register
d) Instruction Register.

- The flag register bits are modified according to the result of the execution of logical and arithmetical instructions. These are called status flag bits.

80286 Block Diagram

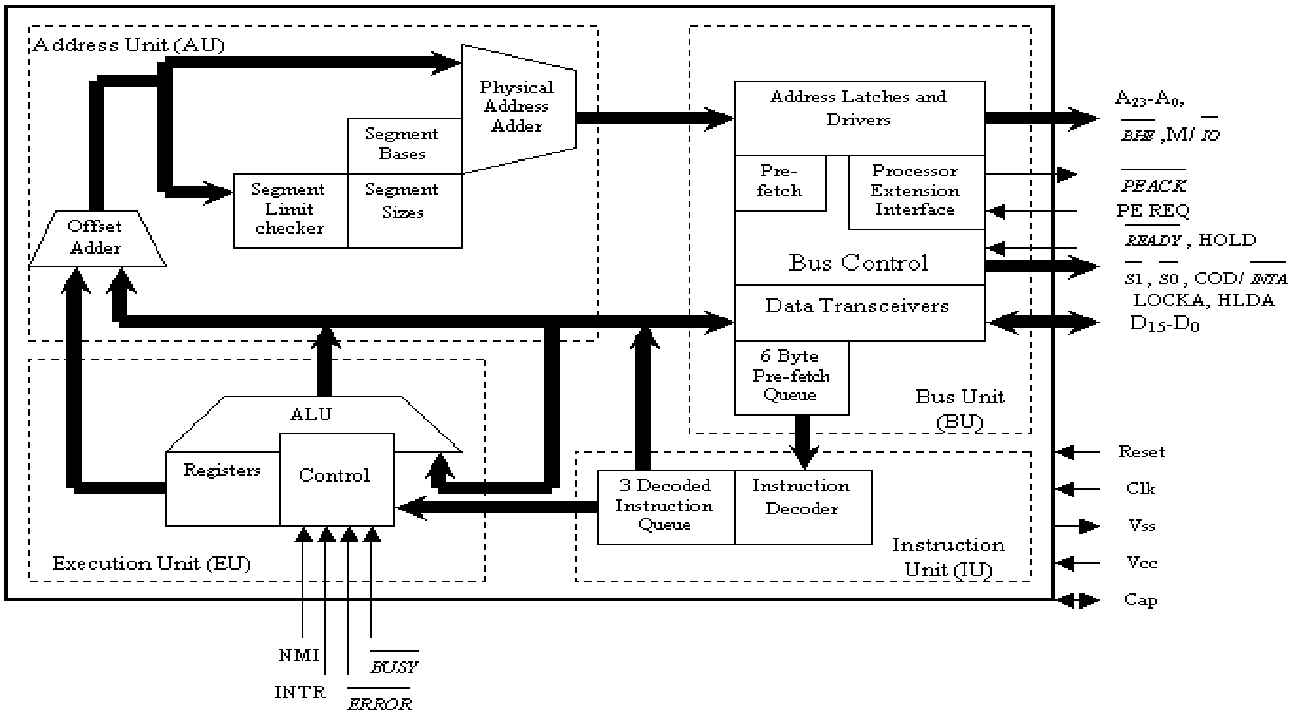


Fig: Block Diagram of 80386

Block diagram of 80286 contain **four** functional parts:

a) Address Unit (AU)
b) Bus Unit (BU)
c) Instruction Set Unit (IU)
d) Execution Unit (EU)

- The **Address Unit (AU)** is responsible for calculating the physical address of instructions and data that CPU wants to access.

- The physical address computed by the address unit is handed over to the **BUS Unit (BU)** of the CPU.
- One of the major function of the bus unit is to fetch instruction bytes from the memory.

- The **Instruction Unit (IU)** accepts instructions from the prefetch queue and an instruction decoder decodes them one by one.

- The **Execution Unit (EU)** is responsible for instructions received from the decoded instruction queue, which sends the data part of the instruction over the data bus. It is responsible for the execution of the decoded instruction.

Operating Modes of 80286

1) Real Address Mode

- 80286 just act as a faster version of 8086.
- And program for 8086 can be executed without modification in 80286.
- In **real** address mode the processor can address upto **1 MB** of physical memory.

2) Protected Virtual Address Mode

- 80286 supports multitasking
- Able to run several program at the same time
- Able to protect memory space for another program
- In this mode the processor can address upto 16 MB of physical memory whereas 8086 can address only 1 MB.
- In this mode the processor can address upto **1 GB** of virtual memory.
- 80286 can treat external storage as it were physical memory and execute programs that are too large to be contained in physical memory.

Flag Register of 80286

Flag register is similar to 8086 which contains 6 conditional/status flags and 3 control flags.

Conditional/Status Flags

- Carry, Auxiliary Carry, Sign, Parity, Zero & Overflow.

Control Flags

- TRAP, Interrupt & Direction Flag.

The **direction flag** is a **flag** that controls the left-to-right or right-to-left **direction** of string processing, stored in the **FLAGS** register.

Interrupt in 80286

Interrupts of 80286 may be divided into three categories,

a) External or Hardware interrupts
b) INT instruction or software interrupts
c) Interrupts generated internally by exceptions (TRAP)

Maskable Interrupt INTR

- This is a maskable interrupt which is to be provided by an external circuit like an interrupt controller.

Non-maskable Interrupt

- NMI has higher priority than the INTR interrupt.
- Once the CPU responds to a NMI request, it does not serve any other interrupt request.

80286 Addressing Modes

(Similar to 8086)

1) Register Addressing Mode
2) Immediate Addressing Mode
3) Direct Addressing Mode
4) Register Indirect Addressing Mode
5) Based Index Addressing Mode
6) Based Addressing Mode
7) Indexed Addressing Mode
8) Input Output Addressing Mode

(Give same example as 8086, because all registers are same & it uses same instruction set).

80286 Memory Management Scheme

Memory is organized into logical segments. Segment size can be anywhere between 1 Byte to 16 KB. All 24 address pins are active and 16 MB of physical memory is available.

Descriptor Cache

- It is 8-byte quantity. Each segment has a descriptor. There are two main types of descriptor -

- Segment Descriptor
    - System control Descriptor
- Descriptors are contained in a descriptor table. There are two categories of descriptor table - global and local.
- A system has only one **global descriptor table or GDT**.
- A **local descriptor table or LDT** is set up in the system for each task or closely related group of tasks. Each task can have its own descriptor table and memory area defined by the descriptors in it.

Accessing Segments

- The 80286 microprocessor keeps the base address and limits for the descriptor tables currently in use in internal registers.
- These registers are load descriptor table register (LDTR) and global descriptor table register (GDTR).
- Descriptor in memory is addressed by adding segment selector to these registers.
- The descriptors contain the base address of segments, which when added with the offset in the virtual address points to the required memory location.

Accessing Segments of Higher Privilege Level

- Tasks operate at the lowest privilege level. Usually, segments at a lower privilege level are not allowed to access segments at a higher privilege level directly.
- However, a lower level segment can access a higher level segment indirectly by a Gate Descriptor. The details of a gate descriptor are given herewith.

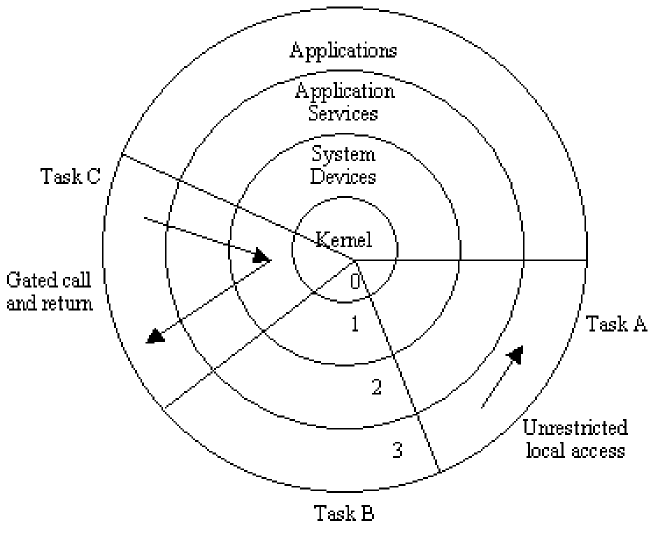


Fig: Privilige Levels

80386 Microprocessor

- 80386 is a **32bit** processor that supports, **8bit/32bit data operands**.
- The **80386 instruction** set is upward compatible with all its predecessors.
- The 80386 can run 8086 applications under protected mode in its virtual 8086 mode of operation.
- With **the 32 bit address bus**, the 80386 can address upto 4Gbytes of physical memory. The physical memory is organised in terms of segments of 4Gbytes at maximum.
- The 80386 CPU supports 16K number of segments and thus the total virtual space of 4Gbytes * 16K = 64 Terrabytes.
- The memory management section of 80386 supports the virtual memory, paging and four levels of protection, maintaining full compatibility with 80286.
- The 80386 offers a set of 8 debug registers DR 0-DR 7 for hardware debugging and control.
- The concept of paging is introduced in 80386 that enables it to organise the available physical memory in terms of pages of size 4Kbytes each, under the segmented memory.

Two versions of 80386 are commonly available:

1) 80386DX

- 32 bit address bus
- 32 bit data bus
- Packaged in 132 pin grid array (PGA)
- Address 4 GB of memory

2) 80386SX

- 24 bit address bus
- 16 bit data bus
- 100 pin flat package
- 16 MB of memory

Block Diagram of 80386 Microprocessor

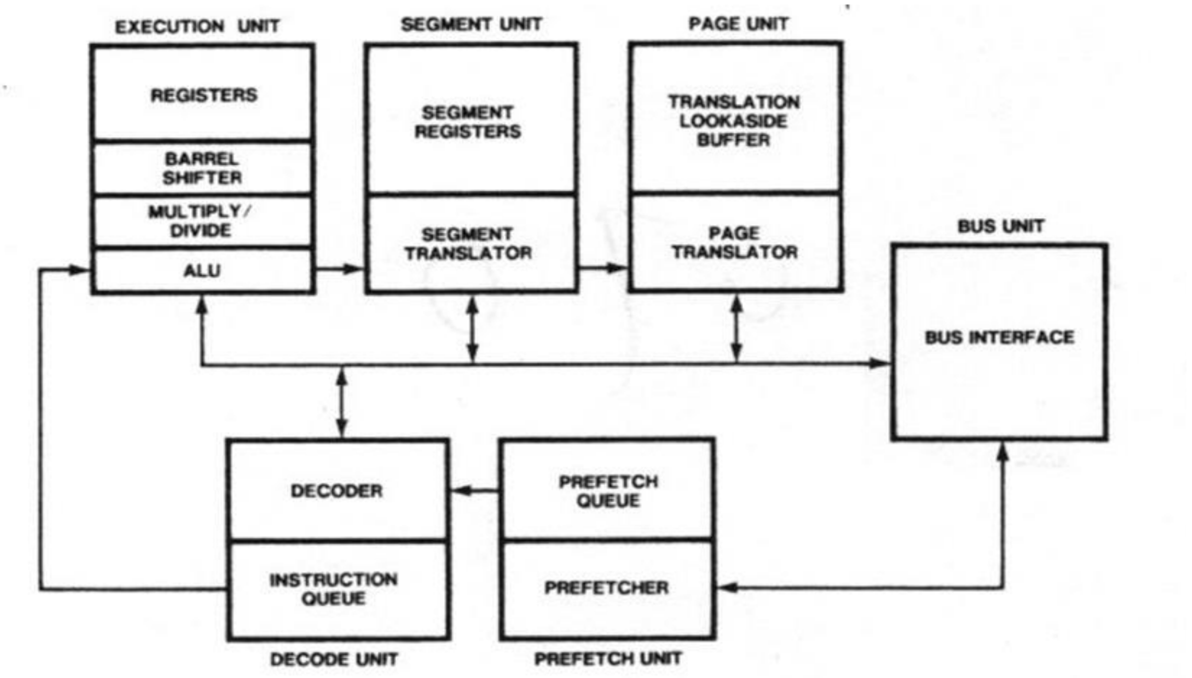


Fig: Block Diagram of 80386 Microprocessor

The Internal Architecture of 80386 is divided into 3 sections.

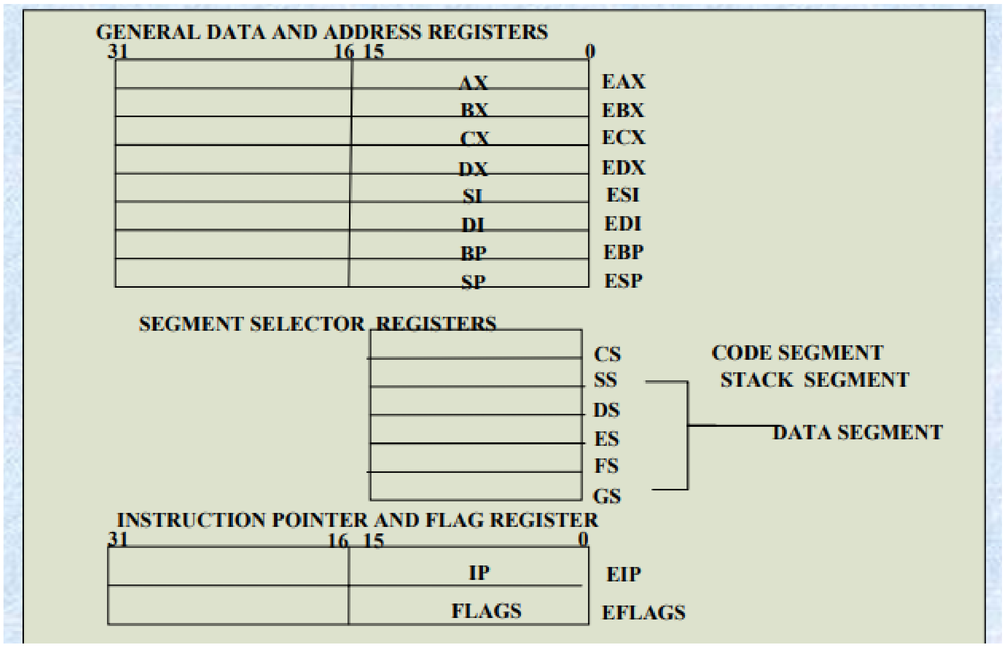
- Central processing unit
- Memory management unit
- Bus interface unit

- Central processing unit is further divided into **Execution unit and Instruction unit**.
- **Execution unit** has 8 General purpose and 8 Special purpose registers which are either used for handling data or calculating offset addresses.
- **The Instruction unit** decodes the opcode bytes received from the 16-byte instruction code queue and arranges them in a 3- instruction decoded instruction queue.

- The **Memory management unit** consists of a **Segmentation unit** and a **Paging unit.**
- **Segmentation unit** allows the use of two address components, viz. segment and offset for relocability and sharing of code and data.
- Segmentation unit allows segments of size 4Gbytes at max.
- The **Paging unit** organizes the physical memory in terms of pages of 4kbytes size each.
- Paging unit works under the control of the segmentation unit, i.e. each segment is further divided into pages. The virtual memory is also organizes in terms of segments and pages by the memory management unit.

- The **Bus control unit** has a prioritizer to resolve the priority of the various bus requests. This controls the access of the bus. The address driver drives the bus enable and address signal A0 – A31.

80386 Register Organization



- The 80386 has eight 32 - bit general purpose registers which may be used as either 8 bit or 16 bit registers.
- A 32 - bit register known as an extended register, is represented by the register name with prefix E.
- Example: A 32-bit register corresponding to AX is EAX, similarly BX is EBX etc.
- The 16 bit registers BP, SP, SI and DI in 8086 are now available with their extended size of 32 bit and are names as EBP, ESP, ESI and EDI.
- AX represents the lower 16 bit of the 32-bit register EAX.
- BP, SP, SI, DI represents the lower 16 bit of their 32 bit counterparts, and can be used as independent 16 bit registers.
- The six **segment registers** available in 80386 are CS, SS, DS, ES, FS and GS.
- The CS and SS are the code and the stack segment registers respectively, while DS, ES, FS, GS are 4 data segment registers.
- A 16-bit **instruction pointer IP** is available along with 32-bit counterpart EIP.
- The **Flag register** of 80386 is a 32-bit register.
- **Control Registers:** The 80386 has three 32-bit control registers CR), CR 2 and CR 3 to hold global machine status independent of the executed task. Load and store instructions are available to access these registers.
- **Debug and Test Registers:** Intel has provided a set of 8 debug registers for hardware debugging.

Protected Mode of 80386

- All the capabilities of 80386 are available for utilization in its protected mode of operation.
- The 80386 in protected mode support all the software written for 80286 and 8086 to be executed under the control of memory management and protection abilities of 80386.
- The protected mode allows the use of additional instruction, addressing modes and capabilities of 80386.

ADDRESSING IN PROTECTED MODE

- In this mode, the contents of segment registers are used as selectors to address descriptors which contain the segment limit, base address and access rights byte of the segment.

Paging in 80386

- Paging is one of the memory management techniques used for virtual memory multitasking operating system.
- The segmentation scheme may divide the physical memory into a variable size segments but the paging divides the memory into a fixed size pages.
- The pages are just fixed size portions of the program module or data.
- The advantage of paging scheme is that the complete segment of a task need not be in the physical memory at any time.
- The paging unit is a memory management unit enabled only in protected mode. The paging mechanism allows handling of large segments of memory in terms of pages of 4Kbyte size.

- Only a few pages of the segments, which are required currently for the execution need to be available in the physical memory. Thus the memory requirement of the task is substantially reduced, relinquishing the available memory for other tasks.
- Whenever the other pages of task are required for execution, they may be fetched from the secondary storage.
- The previous page which are executed, need not be available in the memory, and hence the space occupied by them may be relinquished for other tasks.
- Thus paging mechanism provides an effective technique to manage the physical memory for multitasking systems.

8085 Programs

1. **Program to add two 8-bit numbers.**

   Statement: *Add numbers 05H & 13H and display result in output port 03H.*

   ```
   MVI A,05H     //Move data 05H to accumulator
   MVI B,13H     //Move data 13H to B register
   ADD B         //Add contents of accumulator and B register
   OUT 03H       //Transfer result to output port 03H
   HLT           //Terminate the program.
   ```

   Input: A=05H B=13H **Output**: (port 03H) = 18H

2. **Program to add two 8-bit numbers.**

   Statement: *Add numbers from memory location 2050H & 2051H and store result in memory location 2055H.*

   ```
   LDA 2051H     //Load contents of memory location 2051 to accumulator
   MOV B,A       //Move contents of accumulator to B register
   LDA 2050H     //Load contents of memory location 2050 to accumulator
   ADD B         // Add contents of accumulator and B register
   STA 2055H     //Store contents of accumulator in memory location 2055H
   HLT           //Terminate the program.
   ```

   Input:

Memory Location	**Data**
2050H	45H
2051H	53H

   Output:

Memory Location	**Data**
2055H	98H

3. **Program to subtract two 8-bit numbers.**

   Statement: *Subtract numbers 25H & 12H and display result in output port 01H.*

   ```
   MVI A,25H     //Move data 05H to accumulator
   MVI B,12H     //Move data 13H to B register
   SUB B         //Add contents of accumulator and B register
   OUT 01H       //Transfer result to output port 01H
   HLT           //Terminate the program.
   ```

   Input: A=25H B=12H **Output**: (port 03H) = 13H

4. **Program to subtract two 8-bit numbers.**

   Statement: *Subtract numbers from memory location 2050H & 2051H and store result in memory location 2055H.*

   ```
   LDA 2051H     //Load contents of memory location 2051 to accumulator
   MOV B,A       //Move contents of accumulator to B register
   ```

```
LDA 2050H     //Load contents of memory location 2050 to accumulator
SUB B         // Add contents of accumulator and B register
STA 2055H     //Store contents of accumulator in memory location 2055H
HLT           //Terminate the program.
```

__Input:__

Memory Location	**Data**
2050H	65H
2051H	53H

__Output:__

Memory Location	**Data**
2055H	12H

5. **Program to find 1's complement of a number.**
   Statement: *Input number from memory location 2013H and store result in memory location 2052H.*

```
LDA 2013H     //Load contents from memory location 2013H to accumulator
CMA           //Complement contents of accumulator
STA 2052H     //Store result in memory location 2052H
HLT           //Terminate the program.
```

__Input:__

Memory Location	**Data**
2013H	12H

__Output:__

Memory Location	**Data**
2052H	EDH

6. **Program to find 2's complement of a number.**
   Statement: *Input number from memory location 2013H and store result in memory location 2052H.*

```
LDA 2013H     //Load contents from memory location 2013H to accumulator
CMA           //Complement contents of accumulator
ADI 01H       //Add 01H to the contents of accumulator
STA 2052H     //Store result in memory location 2052H
HLT           //Terminate the program.
```

__Input:__

Memory Location	**Data**
2013H	12H

__Output:__

Memory Location	**Data**
2052H	EEH

7. **Program to right shift 8-bit numbers.**
   Statement*: Shift an eight-bit data four bits right. Assume data is in memory location 2051H. Store result in memory location 2055H.*

```
LDA 2051H     //Load data from memory location 2051H to accumulator
RAR           //Rotate accumulator 1-bit right
RAR
RAR
RAR
STA 2055H     //Store result in memory location 2055H
HLT           //Terminate the program.
```

8. **Program to left shift 8-bit numbers.**
   Statement: *Shift an eight-bit data four bits left. Assume data is in memory location 2051H. Store result in memory location 2055H.*

```
LDA 2051H     //Load data from memory location 2051H to accumulator
RAL           //Rotate accumulator 1-bit left
RAR
RAR
RAR
STA 2055H     //Store result in memory location 2055H
HLT           //Terminate the program.
```

9. **Program to add two 16-bit numbers.**
   Statement: *Add numbers 1124H & 2253H and store result in memory location 2055H & 2056H.*

```
LXI H,1124H   //Load 16-bit data 1124H to HL pair
LXI D,2253H   //Load 16-bit data 2253H to DE pair
MOV A,L       //Move contents of register L to Accumulator
ADD E         //Add contents of Accumulator and E register
MOV L,A       //Move contents of Accumulator to L register
MOV A,H       //Move contents of register H to Accumulator
ADC D         //Add contents of Accumulator and D register with carry
MOV H,A       //Move contents of Accumulator to register H
SHLD 2055H    //Store contents of HL pair in memory address 2055H & 2056H
HLT           //Terminate the program.
```

Input:

Register Pair	**Data**
HL	1124H
DE	2253H

Output:

Memory Location	**Data**
2055H	77H
2056H	33H

10. **Program to add two 16-bit numbers.**
    Statement: *Input first number from memory location 2050H & 2051H and second number from memory location 2052H & 2053H and store result in memory location 2055H & 2056H.*

```
LHLD 2052H    //Load 16-bit number from memory location 2052H & 2053H to HL
pair
XCHG          //Exchange contents of HL pair and DE pair
LHLD 2050H    //Load 16-bit number from memory location 2050H & 2051H to HL
pair
MOV A,L       //Move contents of register L to Accumulator
ADD E         //Add contents of Accumulator and E register
MOV L,A       //Move contents of Accumulator to L register
MOV A,H       //Move contents of register H to Accumulator
ADC D         //Add contents of Accumulator and D register with carry
MOV H,A       //Move contents of Accumulator to register H
SHLD 2055H    //Store contents of HL pair in memory address 2055H & 2056H
HLT           //Terminate the program.
```

Input:

Memory Location	**Data**
2050H	33H
2051H	45H
2052H	24H
2053H	34H

Output:

Memory Location	**Data**
2055H	57H
2056H	79H

11. **Program to subtract two 16-bit numbers.**

Statement: *Subtract number 1234H from 4897H and store result in memory location 2055H & 2056H.*

```
LXI H,4567H   //Load 16-bit data 4897H to HL pair
LXI D,1234H   //Load 16-bit data 1234H to DE pair
MOV A,L       //Move contents of register L to Accumulator
SUB E         //Subtract contents of Accumulator and E register
MOV L,A       //Move contents of Accumulator to L register
MOV A,H       //Move contents of register H to Accumulator
SBB D         //Subtract contents of Accumulator and D register with borrow
MOV H,A       //Move contents of Accumulator to register H
SHLD 2055H    //Store contents of HL pair in memory address 2055H & 2056H
HLT           //Terminate the program.
```

Input:

Register Pair	**Data**
HL	4897H
DE	1234H

Output:

Memory Location	**Data**
2055H	63H
2056H	36H

12. Program to subtract two 16-bit numbers.

Statement: *Input first number from memory location 2050H & 2051H and second number from memory location 2052H & 2053H and store result in memory location 2055H & 2056H.*

```
LHLD 2052H    //Load 16-bit number from memory location 2052H & 2053H to HL pair
XCHG          //Exchange contents of HL pair and DE pair
LHLD 2050H    //Load 16-bit number from memory location 2050H & 2051H to HL pair
MOV A,L       //Move contents of register L to Accumulator
SUB E         //Subtract contents of Accumulator and E register
MOV L,A       //Move contents of Accumulator to L register
MOV A,H       //Move contents of register H to Accumulator
SBB D         //Subtract contents of Accumulator and D register with carry
MOV H,A       //Move contents of Accumulator to register H
SHLD 2055H    //Store contents of HL pair in memory address 2055H & 2056H
HLT           //Terminate the program.
```

Input:

Memory Location	**Data**
2050H	78H
2051H	45H
2052H	24H
2053H	34H

Output:

Memory Location	**Data**
2055H	54H
2056H	11H

13. Program to multiply two 8-bit numbers.

Statement: Multiply 06 and 03 and store result in memory location 2055H.

```
MVI A,00H
MVI B,06H
MIV C,03H
X: ADD B
DCR C
JNZ X
STA 2055H
HLT
```

14. **Program to divide to 8-bit numbers.**

Statement: Divide 08H and 03H and store quotient in memory location 2055H and remainder in memory location 2056H.

```
MVI A,08H
MVI B,03H
MVI C,00H
X: CMP B
   JC Y
   SUB B
   INR C
   JMP X
Y: STA 2056H
MOV A,C
STA 2055H
HLT
```

15. **Program to find greatest among two 8-bit numbers.**

Statement: Input numbers from memory location 2050H & 2051H and store greatest number in memory location 2055H.

```
LDA 2051H
MOV B,A
LDA 2050H
CMP B
JNC X
MOV A,B
X: STA 2055H
HLT
```

16. **Program to find smallest among two 8-bit numbers.**

Statement: Input numbers from memory location 2050H & 2051H and store smallest number in memory location 2055H.

```
LDA 2051H
MOV B,A
LDA 2050H
CMP B
JC X
MOV A,B
X: STA 2055H
HLT
```

17. **Program to find whether a number is odd or even.**

Statement: Input number from memory location 2050H and store result in 2055H.

```
LDA 2050H
ANI 01H
JZ X
MVI A,0DH
```

```
JMP Y
X: MVI A,0EH
Y: STA 2055H
HLT
```

18. Program to count no. of 1's in given number.

Statement: Input number from memory location 2050H and store result in 2055H.

```
LDA 2050H
MVI C,08H
MVI B,00H
X: RAR
   JNC Y
   INR B
Y: DCR C
   JNZ X
MOV A,B
STA 2055H
HLT
```

19. Display number from 1 to 10.

```
LXI H,2050H
MVI B,01H
MVI C,0AH
X: MOV M,B
   INX H
   INR B
   DCR C
   JNZ X
HLT
```

20. Find sum of numbers from 1 to 10.

```
LXI H,2050H
MVI B,01H
MVI C,0AH
MVI A,00H
X: ADD B
   INX H
   INR B
   DCR C
   JNZ X
STA 2055H
HLT
```

21. Display all odd numbers from 1 to 10.

```
LXI H,2050H
MVI B,01H
MVI C,0AH
X: MOV M,B
   INX H
   INR B
```

```
    INR B
    DCR C
    DCR C
    JNZ X
HLT
```

22. Display all even numbers from 1 to 20.

```
LXI H,2050H
MVI B,02H
MVI C,14H
X: MOV M,B
    INX H
    INR B
    INR B
    DCR C
    DCR C
    JNZ X
HLT
```

23. Display all even numbers from 10 to 50.

```
LXI H,2050H
MVI B,0AH
MVI C,32H
X: MOV M,B
    INX H
    INR B
    INR B
    DCR C
    DCR C
    JNZ X
HLT
```

24. Find sum of 10 numbers in array.

```
LXI H,2050H
MVI C,OAH
MVI A,00H
X: MOV B,M
    ADD B
    INX H
    DCR C
    JNZ X
STA 2060H
HLT
```

25. **Find the largest element in a block of data. The length of the block is in the memory location 2200H and block itself starts from memory location 2201H. Store the maximum number in memory location 2300H.**

```
LDA 2200H
MOV C,A
LXI H,2201
MVI A,00H
X: CMP M
   JNC Y
   MOV A,M
Y: INX H
   DCR C
   JNZ X
 STA 2300H
HLT
```

26. **Find smallest number in array.**

```
LDA 2200H
MOV C,A
LXI H,2201H
MVI A,00H
X: CMP M
   JC Y
   MOV A,M
Y: INX H
   DCR C
   JNZ X
 STA 2300H
HLT
```

27. **Generate Fibonacci series upto 10th term.**

```
LXI H,2050H
MVI C,08H
MVI B,00H
MVI D,01H
MOV M,B
INX H
MOV M,D
X: MOV A,B
   ADD D
    MOV B,D
    MOV D,A
    INX H
    MOV M,A
    DCR C
    JNZ X
HLT
```

28. **Sort 10 numbers in ascending order in array.**

```
MVI C,0AH
DCR C
X: MOV D,C
   LXI H,2050H

Y: MOV A,M
   INX H
   CMP M
   JC Z

MOV B,M
MOV M,A
DCX H
MOV M,B
INX H

Z: DCR D
   JNZ Y
   DCR C
   JNZ X
HLT
```

29. **Sort numbers in descending order in array. Length of array is in memory location 2050H.**

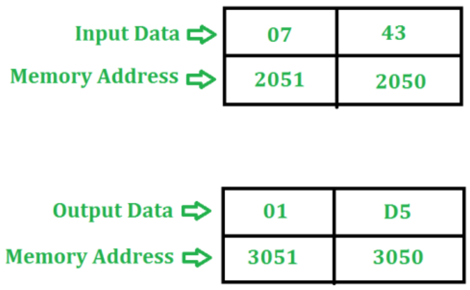
```
LDA 2050H
MVI C,A
DCR C
X: MOV D,C
   LXI H,2051H

Y: MOV A,M
   INX H
   CMP M
   JNC Z

MOV B,M
MOV M,A
DCX H
MOV M,B
INX H

Z: DCR D
   JNZ Y
   DCR C
   JNZ X
HLT
```

30. Multiply two 8 bit numbers 43H & 07H. Result is stored at address 3050 and 3051.

```
LXI H,0000H
MVI D,00H
MVI E,43H
MVI C,07H
X: DAD D
   DCR C
  JNZ X
SHLD 2050H
HLT
```

31. Multiply two 8 bit numbers stored at address 2050 and 2051. Result is stored at address 3050 and 3051.

```
LDA 2050H
MOV E,A
LDA 2051H
MOV C,A
MVI D,00H
LXI H,0000H
X: DAD D
   DCR C
  JNZ X
SHLD 3050H
HLT
```



Input Data ⇨	07	43
Memory Address ⇨	2051	2050

Output Data ⇨	01	D5
Memory Address ⇨	3051	3050

8086 Programs

1. **Program to add two 8-bit numbers.**
   Statement: Add data 05H & 13H and store result in memory location 2050H.
   ```
   MOV AL,05H
   MOV BL,13H
   ADD AL,BL
   MOV 2050H,AL
   HLT
   ```

2. **Program to add two 8-bit numbers.**
   Statement: Input numbers from memory location 2050H and 2051H and store in memory location 2055H.
   ```
   MOV SI,2050H
   MOV AL,[SI]
   INC SI
   MOV BL,[SI]
   ADD AL,BL
   MOV 2055H,AL
   HLT
   ```

3. **Program to add two 16-bit numbers.**
   Statement: Add numbers 1122H & 2233H and store result in memory location 2055H.
   ```
   MOV AX,1122H
   MOV BX,2233H
   ADD AX,BX
   MOV 2055H,AH
   MOV 2056H,AL
   HLT
   ```

4. **Program to subtract two 8-bit numbers.**
   Statement: Subtract data 05H from 13H and store result in memory location 2050H.
   ```
   MOV AL,13H
   MOV BL,05H
   SUB AL,BL
   MOV 2050H,AL
   HLT
   ```

5. **Program to subtracct two 8-bit numbers.**
   Statement: Input numbers from memory location 2050H and 2051H and store in memory location 2055H.
   ```
   MOV SI,2050H
   MOV AL,[SI]
   INC SI
   MOV BL,[SI]
   SUB AL,BL
   ```

```
MOV 2055H,AL
HLT
```

6. **Program to subtract two 16-bit numbers.**
   Statement: Subtract numbers 1122H from 2233H and store result in memory location 2055H.

```
MOV AX,2233H
MOV BX,1122H
SUB AX,BX
MOV 2055H,AH
MOV 2056H,AL
HLT
```

7. **Program to multiply two 8-bit numbers.**
   Statement: Multiply 06H & 03H and store result in memory location 2055H.

```
MOV AL,06H
MOV BL,03H
MUL BL
MOV 2055H,AL
HLT
```

8. **Program to multiply two 8-bit numbers.**
   Statement: Multiply 43H & 13H and store result in memory location 2055H and 2056H. (This program works for 16-bit too.)

```
MOV AX,0043H
MOV BX,0013H
MUL BX
MOV 2055H,AH
MOV 2056H,AL
HLT
```

9. **Program to divide two 8-bit numbers.**
   Statement: Divide 43H & 13H and store result in memory location 2055H.

```
MOV AL,43H
MOV BL,13H
DIV BL
MOV 2055H,AL
HLT
```

10. **Program to divide two 8-bit numbers.**
    Statement: Divide 43H & 13H and store quotient in 2055H and remainder in 2056H.

```
MOV AL,43H
MOV BL,13H
MOV CL,00H
X: CMP AL,BL
   JNC Y
   SUB AL,BL
```

```
    INC CL
Y: MOV  2056H,AL
   MOV AL,BL
   MOV 2055H,AL
HLT
```

11. Program to divide two 16-bit numbers.

Statement: Divide 1243H & 0013H and store result in memory location 2055H & 2056H.

```
MOV AX,1243H
MOV BX,0013H
DIV BX
MOV 2055H,AL
MOV 2056H,BL
HLT
```

12. Program to find sum of numbers from 1 to 10.

```
MOV AL,00H
MOV BL,01H
MOV CL,0AH
X: ADD BL
    INC BL
    LOOP X
MOV 2055H,AL
HLT
```

13. Program to display numbers from 1 to 20.

```
MOV SI,2050H'
MOV BL,01H
MOV CL,14H
X: MOV [SI],BL
    INC BL
    LOOP X
HLT
```

14. Program to find factorial of given number. (Number is in memory location 2050H).

```
MOV CX,2050H
MOV AX,00H
X: MUL CX
    LOOP X
MOV 2055H,AH
MOV 2056H,AL
HLT
```

(Likewise all programs done in 8085 can be done in 8086)

15. Program to display string “I love my country” in screen.

```
.DATA
MESSAGE DB "I love my country$"

.CODE
START:
  MOV AX,DATA
  MOV DS,AX

  MOV AH,09H
  INT 21H

  MOV AH,4CH
  INT 21H

END START
```

16. Program to display string “I love my country” in screen character by character.

```
.DATA
   MESSAGE DB "I love my country$"
.CODE

START:
    MOV AX,DATA
    MOV DS,AX

    LEA SI,MESSAGE
    MOV CL,11H
    L1:MOV DX,[SI]
      MOV AH,02H
     INT 21H
     INC SI
     LOOP L1

    MOV AH,4CH
    INT 21H

END START
```

17. Program to reverse any string.

```
.DATA
   MESSAGE DB "BSC CSIT$"
.CODE

START:
    MOV AX,DATA
    MOV DS,AX

    LEA SI,MESSAGE

    MOV CL,08H

    L1:MOV BX,[SI]
      PUSH BX
      INC SI
      LOOP L1

    MOV CL,05H
    L2:POP DX
      MOV AH,02H
      INT 21H
      LOOP L2

    MOV AH,4CH
    INT 21H

END START
```

(This program can be used for example program for stact PUSH and POP operation)