pa

# UNIT 1

| Q.1 | Explain features of 8086<br>(each 3 mks) min 3 points in feature<br>(each 4 mks) – min 4 points in feature | PPT | 3/4 |
|---|---|---|---|

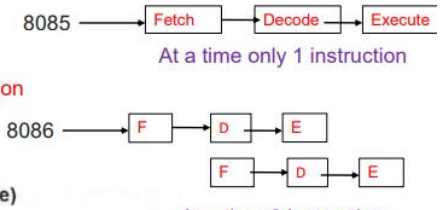
## 1.Basic Features

- Processor Size
- Speed of processor
- Address bus size for memory
- Address bus size for I/O

(1) It is a 16-bit processor. This implies that
   (a) It has a 16-bit ALU that can perform 16-bit operation simultaneously.
   (b) It has 16-bit registers and internal data bus.
   (c) It has 16-bit external data bus.
(2) It has three versions based on the basis of frequency of operation.
   (i) 8086 → 5 MHz.
   (ii) 8086-2 → 8 MHz.
   (iii) 8086-1 → 10 MHz.
(3) 8086 has 20-bit address lines to access memory, hence it can access
   $2^{20}$ = 1 MB memory locations
(4) It has 16-bit address lines to access I/O devices, hence it can access
   $2^{16} = 2^6 \times 2^{10} = 64 \times 1\text{ K}$
   = 64 K I/O locations

## 2. Special Features

- 8086 is a pipelined processor
- 8086 can operate in 2 modes
- 8086 uses memory banks
- 8086 uses memory segmentation




**1) 8086 is a pipelined processor (i.e. it supports pipelined architecture)**

- It uses a two stage pipelining i.e. **Fetch stage** that pre-fetches up to 6 bytes of instructions stores them in the queue and **Execute stage** that executes these instructions.
- Pipelining improves the performance of the processor i.e. the operations are faster.

**2) 8086 can operate in 2 modes**

a. **Minimum mode** → A system with only 1 processor i.e. 8086.
b. **Maximum mode** → A system with 8086 and other processors like 8087-(Math Co-processor), 8089-(IO processor) or multiple 8086 processors.

**3) 8086 uses memory banks**

- The 8086 uses a memory banking system i.e. the entire data is not stored sequentially in a single memory of 1 MB but the memory is divided into two banks of 512KB each.
- The banks are called Lower bank (or even bank, because it stores the data bytes at even locations i.e. 0, 2, 4. ...) and Higher Bank (or odd bank, because it stores the data bytes at odd locations i.e. 1, 3, 5...).
- The benefit of this is that 16-bit data can be accessed in a single access even though the memory chip can store only 8-bit at a location.

**4) 8086 uses memory segmentation**

- A 16-bit address in an instruction or a 16-bit address in a register can access a memory location, although 8086 has 20 address lines. This is made possible using the concept of Segmentation that divides the memory into logical components.
- Here the memory is divided into 16 segments of a capacity of $2^{16}$ (= 65536 B = 64 KB) each and is used as: Code, Stack, Data and Extra Segment.

## 3. Miscellaneous Features

- Interrupts
- Registers
- Instruction set
- Data size for ALU

(1) **It has 256 vectored interrupts :** There are also non-vectored interrupts in 8086, but they are routed to one of these interrupts.
(2) It has 14, 16-bit registers.
(3) It has a powerful instruction set, that supports multiply and divide operations also. (These operations were not possible in the processors earlier to 8086).
(4) 8086 can perform operations on bit, byte (8-bit), word (16-bit) or a string (block of data) types of data.

**Features of 8086 Microprocessor**

**Basic Features *(3 marks – any 3 points)***

1. **16-bit Processor – 16-bit ALU, 16-bit registers, 16-bit external data bus.**
2. **Speed –**
    - **8086 → 5 MHz**
    - **8086-2 → 8 MHz**
    - **8086-1 → 10 MHz**
3. **Memory Addressing – 20-bit address lines → 220=1 MB2^{20} = 1\ \text{MB} memory access.**
4. **I/O Addressing – 16-bit address lines → 216=64 K I/O locations2^{16} = 64\ \text{K I/O locations}.**

**Special Features *(4 marks – any 4 points)***

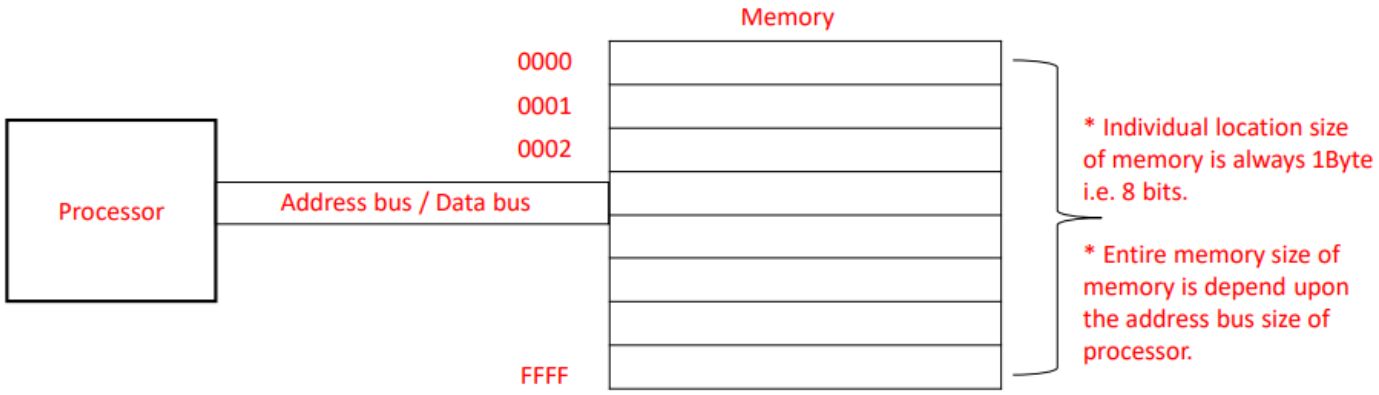
1. **Pipelined Architecture – Two stages (Fetch & Execute), can pre-fetch up to 6 bytes, executes one while fetching next.**
2. **Two Modes –**
    - **Minimum Mode → Single processor.**
    - **Maximum Mode → With 8087/8089 or multiple 8086 processors.**
3. **Memory Banks – 1 MB split into two 512 KB banks (Lower bank → even addresses, Higher bank → odd addresses), allows 16-bit data access in one cycle.**
4. **Memory Segmentation – 16 segments of 64 KB each (Code, Stack, Data, Extra Segment), enables addressing >64 KB using segment+offset.**

| Q.2 | Define term Microprocessor - def 2mks , function – 2mks | PPT | 4 |
|---|---|---|---|

A **microprocessor** is an electronic component that is used by a computer to do its work.
It is a central processing unit on a single integrated circuit chip containing millions of very small components including transistors, resistors, and diodes that work together.

## Basic Functions of processor

1. Programmer write a program using HLL or ALP(instructions)
2. Assembler will convert HLL/ALP into machine code.(opcode of instructions (hex))
3. Loader is responsible to load program into memory (binary form)




1. Processor calculates the memory address and fetch the content of that memory location.
2. Processor decodes the fetched instruction and execute it.

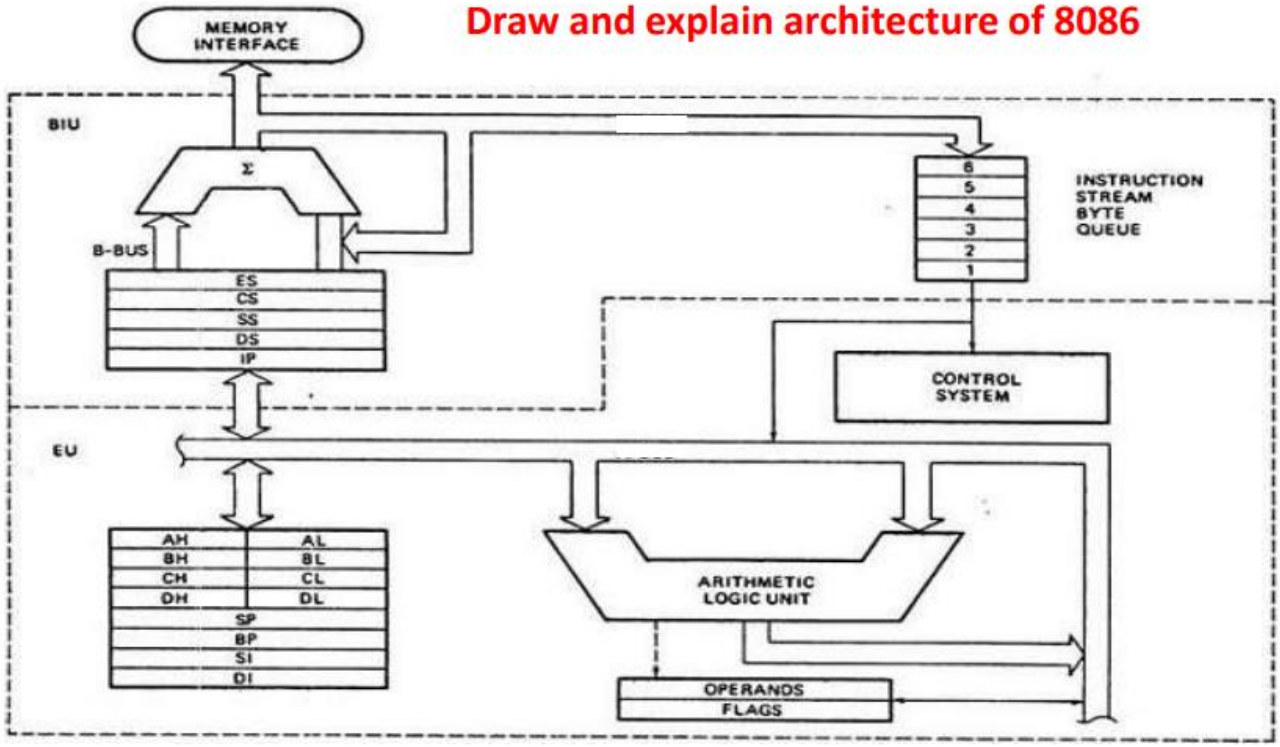
Content of memory location can be Data/instructions

Q.3 | Draw and explain architecture of 8086 | PPT | 6/8

Only draw – 3 mks

Draw explain any one specific block - 2mks or any two (each for 2 mks )

## Draw and explain architecture of 8086




**The Execution Unit (EU)** Main function of EU is **decoding** and **execution** of the instructions. In order to carry out its tasks it has the following units :

- Arithmetic Logic Unit (ALU)
- General Purpose Registers.
- Decoder
- Flag Register.
- Control Unit
- Pointer and Index Register

### Arithmetic Logic Unit (ALU)

- The ALU in the EU is a 16 bit unit i.e. it can perform 16-bit operation simultaneously
- It is capable of performing a variety of arithmetic and logic operations such as add, subtract, AND, OR, NOT, EX-OR, increment, decrement, shift etc.

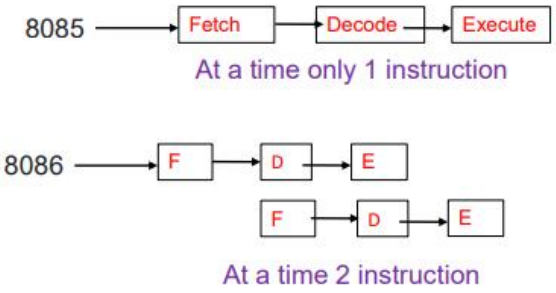
### Control Circuitry

The control circuit is a part of EU. It is used for directing the internal operations.

### A Decoder

- "The process of translation from instructions into action is known as decoding"
- A decoder in the execution unit (EU) is used for translating the instructions fetched from the memory into a series of actions.
- The EU will actually carry out these actions.

## Pipelining :



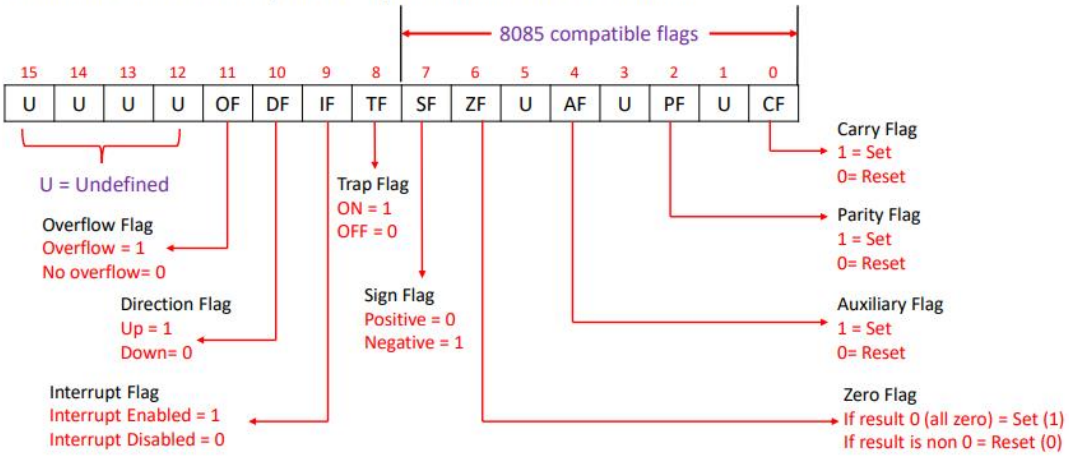

- Pipelining is one of the special feature of 8086 processor.
- Due to pipelining 8086 processor architecture is divided into two parts.
- BIU will fetch the instruction from memory and it will pass that fetched instruction to the Execution unit.
- While execution unit executes the current instruction BIU will fetch the next instruction.
- Pipelining helps in order to increase the speed of processor.

## Advantages of Pipelining :

- The EU always reads the next instruction byte from the queue in BIU. This is much faster than sending out an address to the memory and waiting for the next instruction byte to come.
- In short pipelining eliminates the waiting time of EU and speeds up the processing.

## Flag Register of 8086

- Flag register is a part of Execution Unit.
- It is a 16-bit register with each bit corresponding to a flip-flop.
- Flag register is used to give status of operation performed by processor.
- A flag is flip-flop.
- It indicates some condition produced by the execution of an instruction.




## Carry Flag (CF)

- It can also be called as a final carry.
- This flag is set whenever there has been a carry out of, or borrow into, the MSB of the result (8 bit / 16 bit).
- The flag is used by the instruction that add and subtract multibyte numbers.
- CF = 1 , if there is a carry out from the most significant bit (MSB)
- CF = 0 , if no carry out from MSB

**Example 1 (8 bit) :**

**ADD BL, CL where BL = 02 H and CL= 51 H**

```
  BL = 02 h =  0000 0010
+ CL = 51 h =  0101 0001
-------------------------
       0101 0011
```

MSB LSB

Carry is not generated from MSB

| U | U | U | U | OF | DF | IF | TF | SF | ZF | U | AF | U | PF | U | CF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | 0 |

**Example 2 (8 bit) :**

**ADD BL, CL where BL = 83 H and CL= 81 H**

```
  BL = 83 h =  1000 0011
+ CL = 81 h =  1 1000 0001
                  1  1
-------------------------
       0000 0100
```

MSB LSB

Carry is generated from MSB

| U | U | U | U | OF | DF | IF | TF | SF | ZF | U | AF | U | PF | U | CF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | 1 |

## Parity Flag (PF)

- This flag is normally used to check data transmission errors.
- PF = 1 , when the result has even parity, an even number of 1's
- PF = 0 , when the result has odd parity, an odd number of 1's

**Example 1 :**

**ADD BL, CL where BL = 02 H and CL= 51 H**

```
  BL = 02 h =  0000 0010
+ CL = 51 h =  0101 0001
-------------------------
       0000 0011
```

Result has even number of 1's

| U | U | U | U | OF | DF | IF | TF | SF | ZF | U | AF | U | PF | U | CF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | 1 | | |

**Example 2 :**

**ADD BL, CL where BL = 83 H and CL= 81 H**

```
  BL = 83 h =  1000 0011
  CL = 81 h =  1 1000 0001
                  1  1
-------------------------
       0000 0100
```

Result has odd number of 1's

| U | U | U | U | OF | DF | IF | TF | SF | ZF | U | AF | U | PF | U | CF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | 0 | | |

| Q.6 | Explain special functions of general purpose registers (AX,BX,CX,DX) 1. Each for 2 mks (minimum 2 functions)....4/6<br>2. any one ...3mks (minimum 3 functions).. 3 | PPT | 3/4/6 |
|---|---|---|---|

# Special functions of general purpose registers of 8086

## 1. Register AX : Accumulator

*AX is the "16-bit accumulator"* while *AL is "8-bit accumulator"*

*Accumulator has the following special functions :*

(i) Some of the operations, such as Multiplication and Division, require that one of the operands be in the accumulator and also the result is stored in accumulator. Some other operations, such as Addition and Subtraction, may be applied to any of the registers (that is, any of the eight general- and special-purpose registers) but are more efficient when working with the accumulator.

(ii) It works as a via register for I/O accesses i.e. a data is routed through accumulator for the communication of the processor and I/O devices.
For OUT instruction the data in accumulator (AL for 8-bit data and AX for 16-bit data) can only be given to the output device.
For IN instruction the data taken from the input device can be taken only in accumulator (AL for 8-bit data and AX for 16-bit data)

(iii) It also works as a via register for string instructions. Whenever a data is to be brought from memory or given to memory in case of string operations it is routed through accumulator only.

## 2. Register BX : Base

*BX is the "base" register.*

(i) It is the only general-purpose register which may be used for indirect addressing. (various addressing modes are discussed in chapter 4.)

(ii) For example, the instruction MOV [BX], AX causes the contents of AX to be stored in the memory location whose address is given in BX.

## 3. Register CX : Counter

*CX is the "count" register.* It works as a default counter register for three instructions viz :

(i) The looping instructions (LOOP, LOOPE, and LOOPNE), to indicate the number of iterations

(ii) The shift and rotate instructions (RCL, RCR, ROL, ROR, SHL, SHR, and SAR), to indicate number of shifts or rotations (Here only CL is used and not entire CX)

(iii) The string instructions (with the prefixes REP, REPE, and REPNE) to indicate the size of the string block.

## 4. Register DX : Data

*DX is the "data" register*

(i) It is used together with AX for the word-size MUL and DIV operations, when the operand size is greater than the register AX i.e. operand is 32-bit.

(ii) It also holds the port number for the IN and OUT instructions. For 16-bit address accesses of I/O ports only DX can be used as a pointer.

**Summary of Implicit use of General Purpose Registers**

| Registers | Operations |
|---|---|
| AX | Word multiply, Word divide, Word I/O and Word string |
| AL | Byte multiply, byte divide, byte I/O, byte string and decimal / ASCII arithmetic. |
| BX | Store address information |
| CX | Counter for String operations and loops |
| CL | Counter for Variable shift and rotate |
| DX | Word multiply, word divide, Indirect I/O |

Q.7 | Explain use of any pin or signal of 8086 (each for 2 mks) | PPT | 4

**Power and Clock**

- **Pin 40 - VCC:** Provides the +5V power supply to the chip.
- **Pins 1, 20 - VSS:** Serve as the ground connections for the processor.
- **Pin 19 - CLK:** Receives the system clock signal to synchronize all operations.

**Address, Data, and Status Bus**

- **Pins 2-16, 39 - AD0-AD15:** Function as a multiplexed bus for the lower 16-bit address and 16-bit data.
- **Pins 35-38 - A16/S3-A19/S6:** Function as a multiplexed bus for the upper 4-bit address and status signals.
- **Pin 34 - BHE/S7:** Enables data on the high byte of the data bus (D8-D15).

**Common Control Signals**

- **Pin 32 - RD:** Active-low signal that indicates the CPU is reading from memory or an I/O device.
- **Pin 22 - READY:** An acknowledgment from a slow device or memory that it is ready for data transfer.
- **Pin 21 - RESET:** Resets the processor to its initial state when held high for four clock cycles.
- **Pin 23 - TEST:** An input tested by the WAIT instruction to pause execution until the pin goes low.
- **Pin 33 - MN/MX:** A mode selector pin for minimum (single-processor) or maximum (multi-processor) operation.
- **Pin 28 - M/IO:** Distinguishes between a memory operation (high) and an I/O operation (low).

**Interrupts**

- **Pin 18 - INTR:** A maskable interrupt request line that the processor can choose to ignore.
- **Pin 17 - NMI:** A non-maskable interrupt which forces the CPU to respond immediately.

**Minimum Mode Signals**

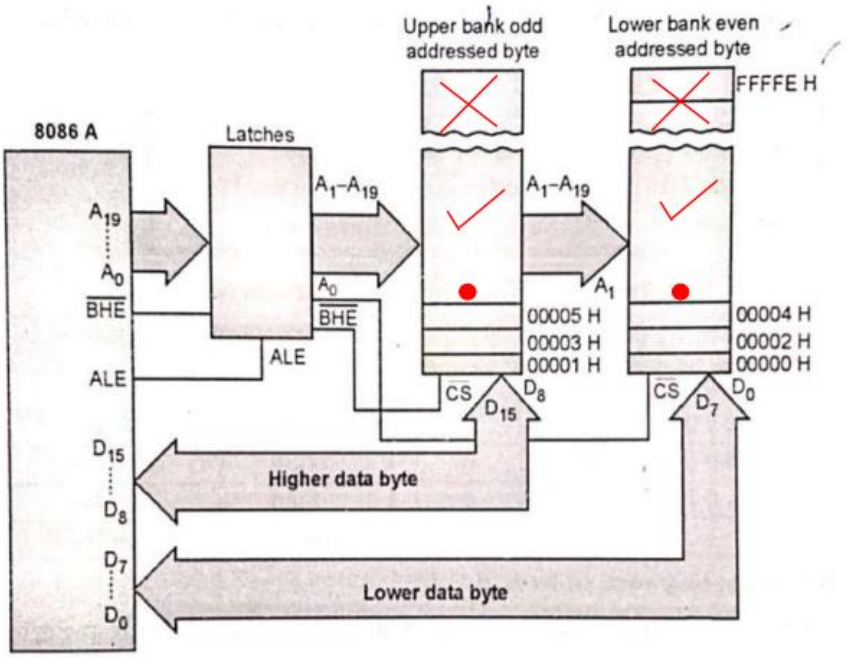
- **Pin 29 - WR:** Active-low signal indicating the CPU is writing to memory or an I/O device.
- **Pin 24 - INTA:** Serves as an interrupt acknowledgment to the interrupting device.
- **Pin 25 - ALE:** Address Latch Enable signal used to demultiplex the address/data bus.
- **Pin 26 - DEN:** Data Enable signal used to activate external data bus buffers.
- **Pin 27 - DT/R:** Controls the direction of data flow (transmit/receive) on the data bus.
- **Pin 31 - HOLD:** Receives requests from other devices to take control of the system bus.
- **Pin 30 - HLDA:** Acknowledges a HOLD request, indicating the bus is available for another master.

**Maximum Mode Signals**

- **Pins 26-28 - S2, S1, S0:** Status signals that are decoded by an external bus controller to generate control signals.
- **Pin 29 - LOCK:** An active-low signal that prevents other processors from taking control of the bus.
- **Pins 30-31 - RQ/GT1, RQ/GT0:** Bidirectional lines used to request and grant bus control in a multi-processor system.
- **Pins 24-25 - QS1, QS0:** Provide status on the internal instruction prefetch queue.

**Block diagram of 8086 memory bank**

| $\overline{BHE}$ | A0 | Bank |
|---|---|---|
| 0 | 0 | Both bank (16 bit) |
| 0 | 1 | Odd (8 bit) |
| 1 | 0 | even (8 bit) |
| 1 | 1 | No Bank |




**Memory Bank of 8086 – Explanation**

The 8086 uses **two 8-bit memory banks** (Lower Bank & Higher Bank) to form a 16-bit data bus.

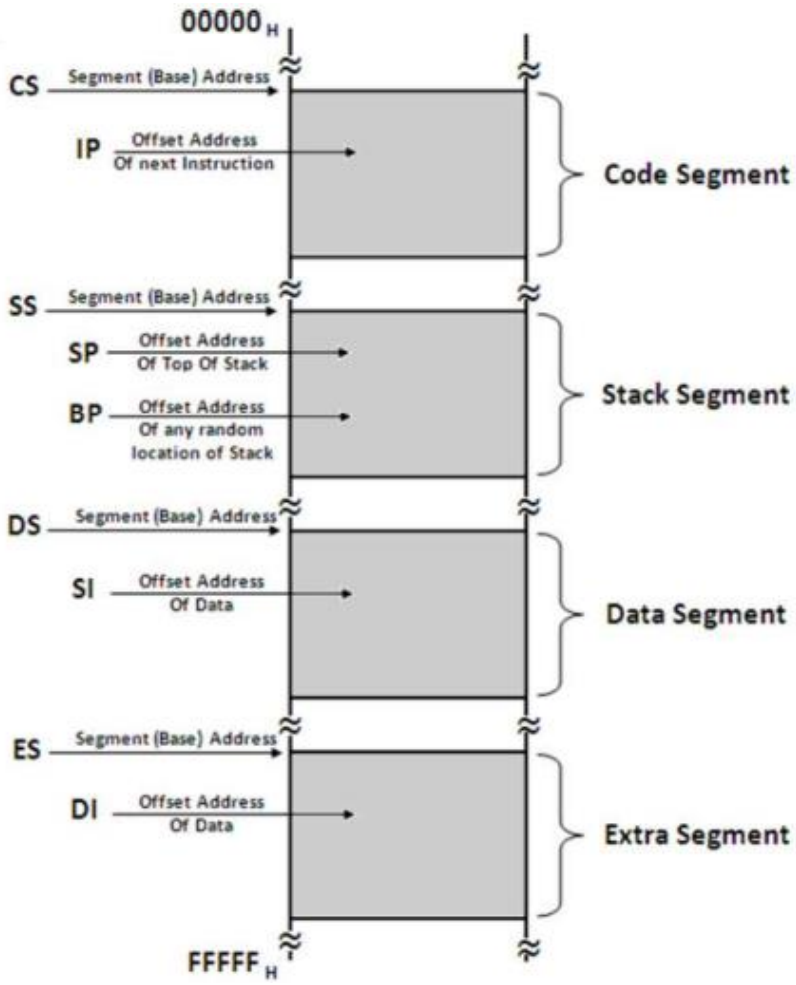
- **Lower Bank (Even Bank)** → stores bytes at even addresses (A0 = 0).
- **Higher Bank (Odd Bank)** → stores bytes at odd addresses (A0 = 1).
- The signal **$\overline{BHE}$** (Bus High Enable, active low) is used to enable the higher bank.
- This design allows **16-bit (word) data** to be accessed in one cycle if aligned at an even address.

**Cases of Access**

| Case | Address Example | Banks Active | Access Cycles | Explanation |
|---|---|---|---|---|
| **1. Word at Even Address** | 2000H | Lower & Higher bank active together | 1 cycle | Lower byte in even bank, upper byte in odd bank — both read in parallel. |
| **2. Word at Odd Address** | 2001H | First cycle: Higher bank → lower byte; Second cycle: Lower bank → upper byte | 2 cycles | Word crosses bank boundary; requires two separate reads. |
| **3. Byte at Even Address** | 2000H | Lower bank only | 1 cycle | Only the even bank is enabled. |
| **4. Byte at Odd Address** | 2001H | Higher bank only | 1 cycle | Only the odd bank is enabled. |

**Key Points:**

- **Even-aligned words** are faster to access (1 cycle).
- **Odd-aligned words** need 2 cycles because they span both banks at different addresses.
- Byte operations only require one bank.




## Code Segment

This segment is used to hold the **program** to be executed.
**Instruction are fetched** from the Code Segment.
**CS** register holds the 16-bit **base** address for this segment.
**IP** register (Instruction Pointer) holds the 16-bit **offset** address.

## Data Segment

This segment is used to hold **general data**.
This segment also holds the **source** operands during **string** operations.
**DS** register holds the 16-bit **base** address for this segment.
**BX** register is used to hold the 16-bit **offset** for this segment.
**SI** register (Source Index) holds the 16-bit **offset** address during String Operations.

## Stack Segment

This segment holds the **Stack** memory, which operates in LIFO manner.
**SS** holds its **Base** address.
**SP** (Stack Pointer) holds the 16-bit **offset** address of the **Top** of the Stack.
**BP** (Base Pointer) holds the 16-bit **offset** address during **Random Access**.

## Extra Segment

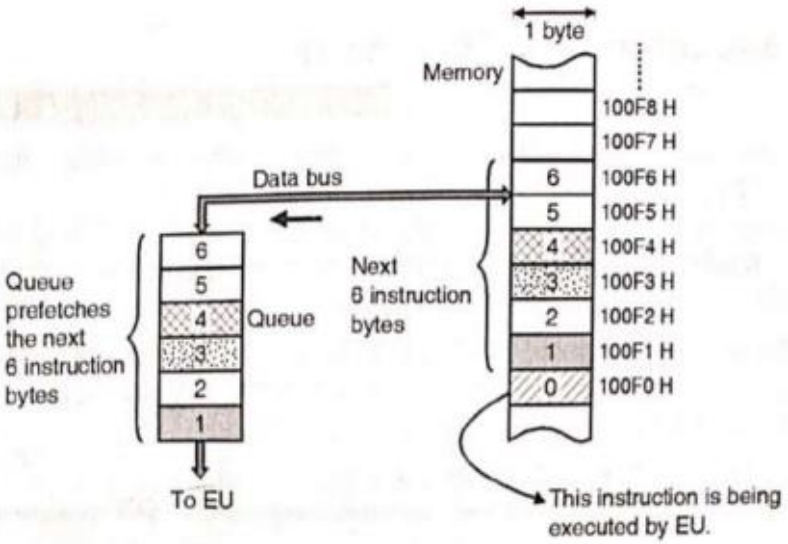
This segment is used to hold **general data**
Additionally, this segment is used as the **destination** during **String Operations**.
**ES** holds the **Base** Address.
**DI** holds the **offset** address during string operations.

## Advantages of Segmentation:

1) It permits the programmer to access 1MB **using only 16-bit address**.
2) Its **divides** the **memory logically** to store Instructions, Data and Stack separately.

## Disadvantage of Segmentation:

1) Although the total memory is 16*64 KB, **at a time only 4*64 KB memory can be accessed**.




## The Instruction Queue

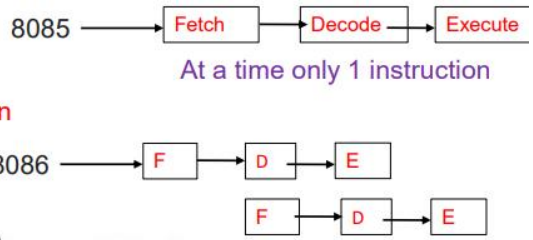
- The execution unit is supposed to decode or execute an instruction. Decoding does not require the use of buses.
- When EU is busy in decoding and executing an instruction, the BIU fetches upto six instruction bytes for the next instructions.
- These bytes are called as the prefetched bytes and they are stored in a first-in-first-out (FIFO) register set, which is called as a "**queue**."

### Significance of Queue

- To understand the significance of the queue, refer Fig. 2.5.1.
- As shown in Fig. 2.5.1, while the EU is busy in decoding the instruction corresponding to memory location 100F0, the BIU fetches the next six instruction bytes from locations 100F1 to 100F6 numbered as 1 to 6.
- These instruction bytes are stored in the 6 byte Queue on the first-in-first-out (FIFO) basis.
- When EU completes the execution of the existing instruction, and becomes ready for the next instruction, it simply reads the instruction bytes in the sequence 1, 2, .... from the Queue.
- Thus the Queue will always hold the instruction bytes of the next instructions to be executed by the EU.

## 2. Special Features

- 8086 is a pipelined processor
- 8086 can operate in 2 modes
- 8086 uses memory banks
- 8086 uses memory segmentation




**1) 8086 is a pipelined processor (i.e. it supports pipelined architecture)**

- It uses a two stage pipelining i.e. **Fetch stage** that pre-fetches up to 6 bytes of instructions stores them in the queue and **Execute stage** that executes these instructions.
- Pipelining improves the performance of the processor i.e. the operations are faster.

**2) 8086 can operate in 2 modes**

a. **Minimum mode** → A system with only 1 processor i.e. 8086.
b. **Maximum mode** → A system with 8086 and other processors like 8087-(Math Co-processor), 8089-(IO processor) or multiple 8086 processors.

**3) 8086 uses memory banks**

- The 8086 uses a memory banking system i.e. the entire data is not stored sequentially in a single memory of 1 MB but the memory is divided into two banks of 512KB each.

---

- The banks are called Lower bank (or even bank, because it stores the data bytes at even locations i.e. 0, 2, 4...) and Higher Bank (or odd bank, because it stores the data bytes at odd locations i.e. 1, 3, 5...).
- The benefit of this is that 16-bit data can be accessed in a single access even though the memory chip can store only 8-bit at a location.

**4) 8086 uses memory segmentation**

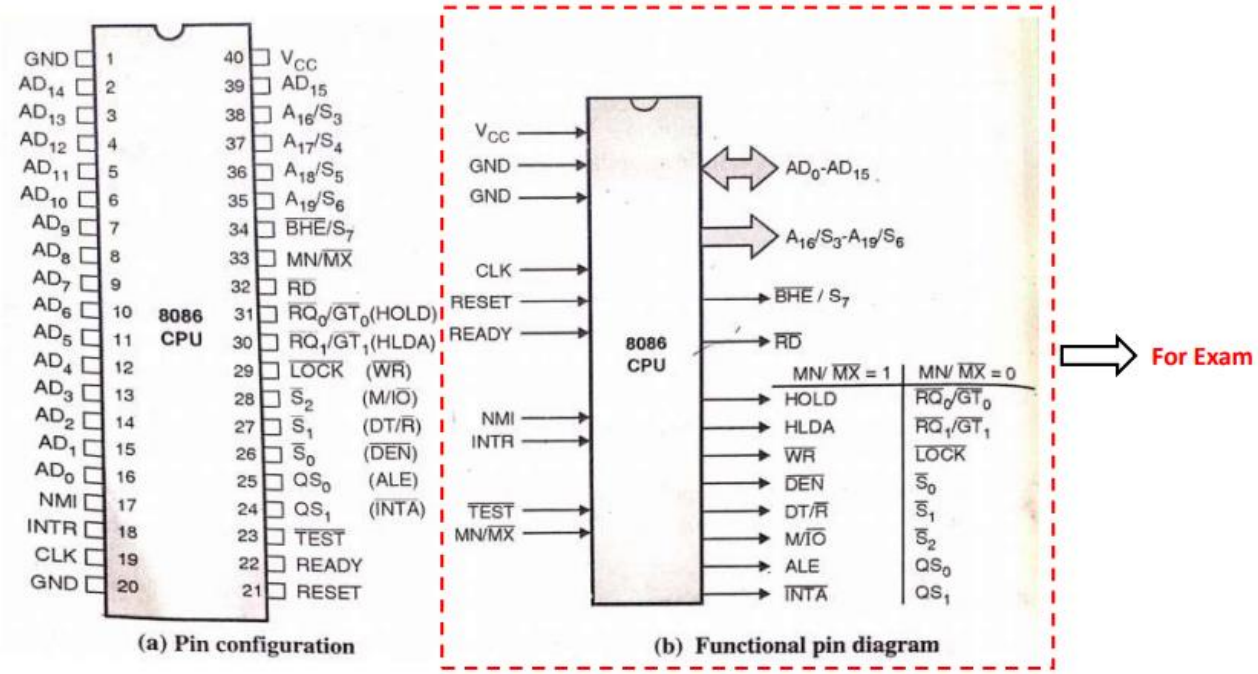
- A 16-bit address in an instruction or a 16-bit address in a register can access a memory location, although 8086 has 20 address lines. This is made possible using the concept of Segmentation that divides the memory into logical components.
- Here the memory is divided into 16 segments of a capacity of $2^{16}$ (= 65536 B = 64 KB) each and is used as: Code, Stack, Data and Extra Segment.

## 3. Miscellaneous Features

- Interrupts
- Registers
- Instruction set
- Data size for ALU

(1) **It has 256 vectored interrupts :** There are also non-vectored interrupts in 8086, but they are routed to one of these interrupts.
(2) It has 14, 16-bit registers.
(3) It has a powerful instruction set, that supports multiply and divide operations also. (These operations were not possible in the processors earlier to 8086).
(4) 8086 can perform operations on bit, byte (8-bit), word (16-bit) or a string (block of data) types of data.

# Pin diagram of 8086




For Exam

## Q.13 | Explain BIU and EU of the architecture of 8086 | | 6

**The Execution Unit (EU)** Main function of EU is **decoding** and **execution** of the instructions. In order to carry out its tasks it has the following units :

- Arithmetic Logic Unit (ALU)
- General Purpose Registers.
- Decoder
- Flag Register.
- Control Unit
- Pointer and Index Register

### Arithmetic Logic Unit (ALU)

- The ALU in the EU is a 16 bit unit i.e. it can perform 16-bit operation simultaneously
- It is capable of performing a variety of arithmetic and logic operations such as add, subtract, AND, OR, NOT, EX-OR, increment, decrement, shift etc.

### Control Circuitry

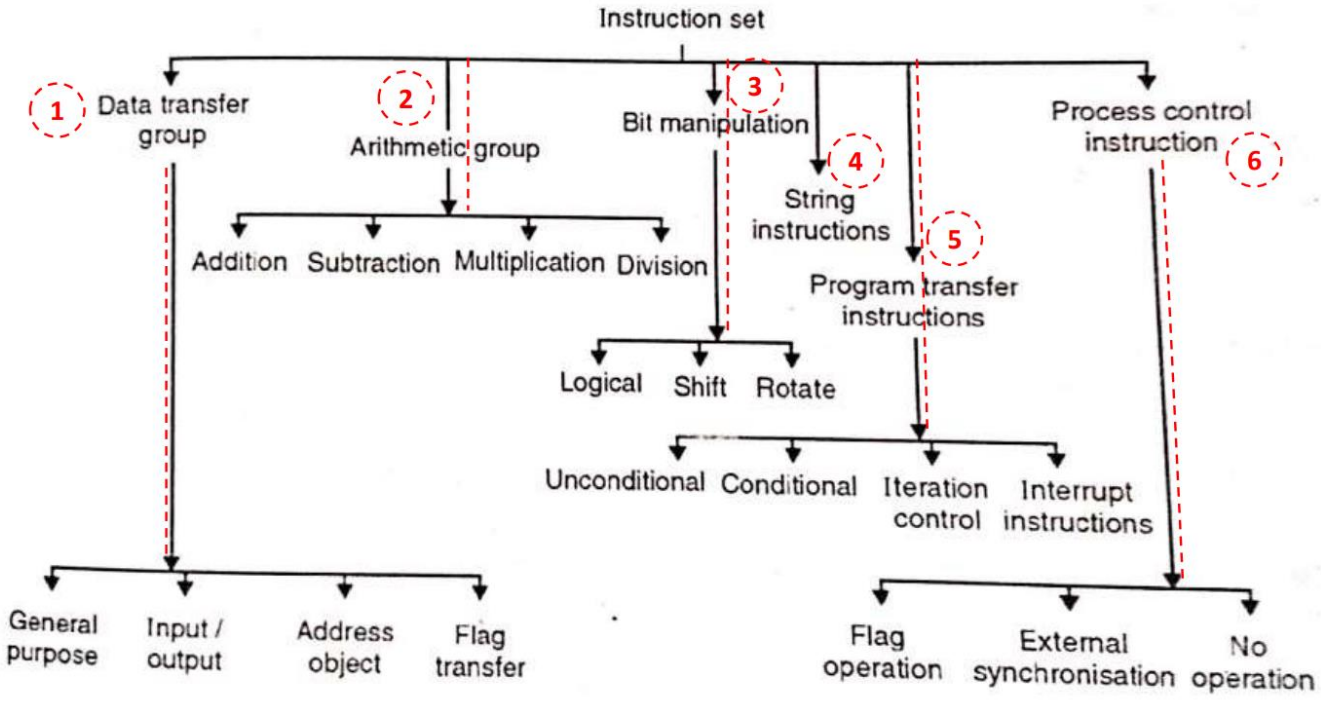
The control circuit is a part of EU. It is used for directing the internal operations.

### A Decoder

- "The process of translation from instructions into action is known as decoding"
- A decoder in the execution unit (EU) is used for translating the instructions fetched from the memory into a series of actions.
- The EU will actually carry out these actions.

# UNIT 2

Q.1 | Explain any specific individual / group of instructions for 8086 e.g explain with example PUSH and POP instruction | PPT | 4/6/8




# leave this ans you have to memorize every single instuctions with examples

| Q.2 | Explain addressing modes of 8086 (each for 3mks )<br>1mk -for explanation<br>2mk – for example | PPT | 4/6/8 |
|---|---|---|---|

1. Immediate addressing mode ( Data in Instruction)

2. Register addressing mode ( Data in register)

3. Direct addressing mode ( Address in Instruction)

4. Indirect addressing mode ( Address in Register)

5. Implied addressing mode ( Nothing is given in instruction)

## 1. Immediate addressing mode ( Data in Instruction)

In immediate addressing mode the data to be used is immediately given in the instruction.

Example :

MOV CL, 02 H ⟶ 02 (8 bit data) is transfer into reg. CL

MOV CX, 2005 H ⟶ 2005 (16 bit data) is transfer into reg. CX in following manner :

## 2. Register addressing mode ( Data in register)

In register addressing mode data to be operand is in general purpose register

Example :

MOV CL, BL ⟶ Content (8 bit data) of reg. BL is transferred into reg. CL

MOV CX, BX ⟶ Content (16 bit data) of reg. BX is transferred into reg. CX

## 3. Direct addressing mode ( Address in Instruction)

In direct addressing mode operand is given by a direct address where the data is present.

Example 1 :

Anything in [ ] refers address

MOV CL, [2000] ⟶ The Content which present at memory location 2000 transfer into reg. CL

CL ⟵ DS : [2000]

## 4. Indirect addressing mode ( Address in register)

In indirect addressing mode the instruction dose not have the address of the data to be operated on. But the instruction points where the address is stored i.e. it is indirectly specifying the address of memory location where the data is stored or is to be stored.

There are four types of indirect addressing mode :

1. Register Indirect (address simply given by register)
2. Register relative ( address in reg + relative)
3. Based indexed (address in base reg + index reg)
4. Based relative indexed (address in base reg + index reg + relative)

## 1.Register Indirect addressing mode ( Address in reg)

In direct addressing mode operand is given by a direct address where the data is present

Example : MOV CL,[BX] ⟶ The Content of memory location 4000 transfer into CL

CL ⟵ DS : [4000].

## 2.Register relative addressing mode ( Address in reg + relative)

In this mode, the operand address is calculated using one of the base registers and an 8 bit or a 16 bit displacement.

Example : MOV CL,[BX + displacement] ⟶ The Content of memory location 4002 transfer into CL

MOV CL,[BX + 02 h ] CL ⟵ DS : [4000 + 02].

MOV CL,[BX + displacement] Increment memory locations by displacement

Note : • Only BX register is used

MOV CL,[BX - displacement] Decrement memory locations by displacement

## 3.Based Index addressing mode ( Address in base reg + Index)

In this mode, operand address is calculated as base register plus an index register and an 8 bit or a 16 bit displacement.

Example : MOV CL,[BX + SI] ⟶ This instruction moves a byte from the address pointed by BX + SI in data segment to CL.
CL ← DS : [4000 + 2000].

## 4.Based relative Index addressing mode ( Address in base reg + Index + relative)

In this mode, operand address is calculated as base register plus an index register and 8 or 16 bit displacement.

Example :

MOV CL,[BX + SI + displacement ] ⟶ This instruction moves a byte from the address pointed by BX + SI + 02 in data segment to CL.
MOV CL,[BX + SI + 02h ] CL ← DS : [2000 + SI + 02].

## 5. Implied addressing mode ( Nothing is given in instruction)

In this mode, the operands are implied and are hence not specified in the instruction.

Example :

Operations are related with specific register

STC ⟶ Set carry flag

DAA ⟶ Decimal adjust after addition

# UNIT 3

**1.)List and explain the use of any one pin of minimum mode (Listing – 1mk, explain use 2 mks)(4)**

## Control & Status Pins (Minimum Mode Specific)

1. **M/IO̅** *(Memory/Input-Output)*
   - **Use:** Tells whether the operation is for memory or I/O.
     - **High (1)** → Memory operation
     - **Low (0)** → I/O operation
2. **RD̅** *(Read)*
   - **Use:** Indicates that the processor is performing a read operation from memory or I/O.
     - Active low signal → Data is read from addressed location into the CPU.
3. **WR̅** *(Write)*
   - **Use:** Indicates that the processor is writing data to memory or I/O.
     - Active low → Data is placed on the bus by CPU and written to the addressed device.
4. **ALE** *(Address Latch Enable)*
   - **Use:** Activates external latches to separate the address from multiplexed address/data lines.
     - High → Address is available on the bus and should be latched.
5. **INTA̅** *(Interrupt Acknowledge)*
   - **Use:** Used by the CPU to acknowledge receipt of an interrupt request from an interrupt controller or device.

### Status Signals

6. **DT/$\bar{R}$** *(Data Transmit / Receive)*
   - **Use:** Tells direction of data flow through transceivers.
     - High (1) → CPU transmitting data
     - Low (0) → CPU receiving data
7. **DE$\bar{N}$** *(Data Enable)*
   - **Use:** Enables the data transceivers to allow data flow on the data bus during read or write operations.
8. **HOLD**
   - **Use:** Indicates that an external device is requesting control of the system bus.
9. **HLDA** *(Hold Acknowledge)*
   - **Use:** CPU signals that it has granted control of the bus to the requesting device.
10. **READY**
    - **Use:** Used to insert wait states in the CPU's operation until slower memory or I/O devices are ready for data transfer.

### Clock & Reset

11. **CLK**
    - **Use:** Provides the timing signal for all CPU operations.
12. **RESET**
    - **Use:** Initializes the CPU, clears registers, sets CS:IP to FFFF0H to start execution.

**Q.2 )List and explain the use of any one pin of maximum mode Listing – ( 1mk, explain use 2 mks) (4)**

## Control Signals (Maximum Mode)

1. **S2, S1, S0** *(Status Lines)*
   - **Use:** These 3 lines indicate the type of operation currently in progress (like opcode fetch, memory read, I/O write, interrupt acknowledge, etc.).
   - Example:
     - 000 → Interrupt Acknowledge
     - 010 → Memory Write
     - 101 → I/O Read
2. **RQ/$\overline{GT}$0** *(Request/Grant 0)*
   - **Use:** Bi-directional line for bus access control.
     - External device sends **Request** → CPU responds with **Grant**.
     - Priority: RQ/$\overline{GT}$0 has higher priority over RQ/$\overline{GT}$1.
3. **RQ/$\overline{GT}$1** *(Request/Grant 1)*
   - **Use:** Same as above, but lower priority.
4. **LOC$\overline{K}$**
   - **Use:** When low, it prevents other processors from accessing the system bus.
     - Used during certain critical instructions like `XCHG` to ensure exclusive bus control.

## Common to Both Modes (but used differently in Max Mode)

5. **CLK**
   - **Use:** Provides timing for CPU operations.
6. **RESET**
   - **Use:** Resets CPU and sets CS:IP to FFFF0H.
7. **READY**
   - **Use:** Inserts wait states for slow devices.

**3.) Explain use of any pin or signal minimum or maximum mode (each for 2 mks)(4/6)**

## Minimum Mode – Pins & Uses (2 marks each)

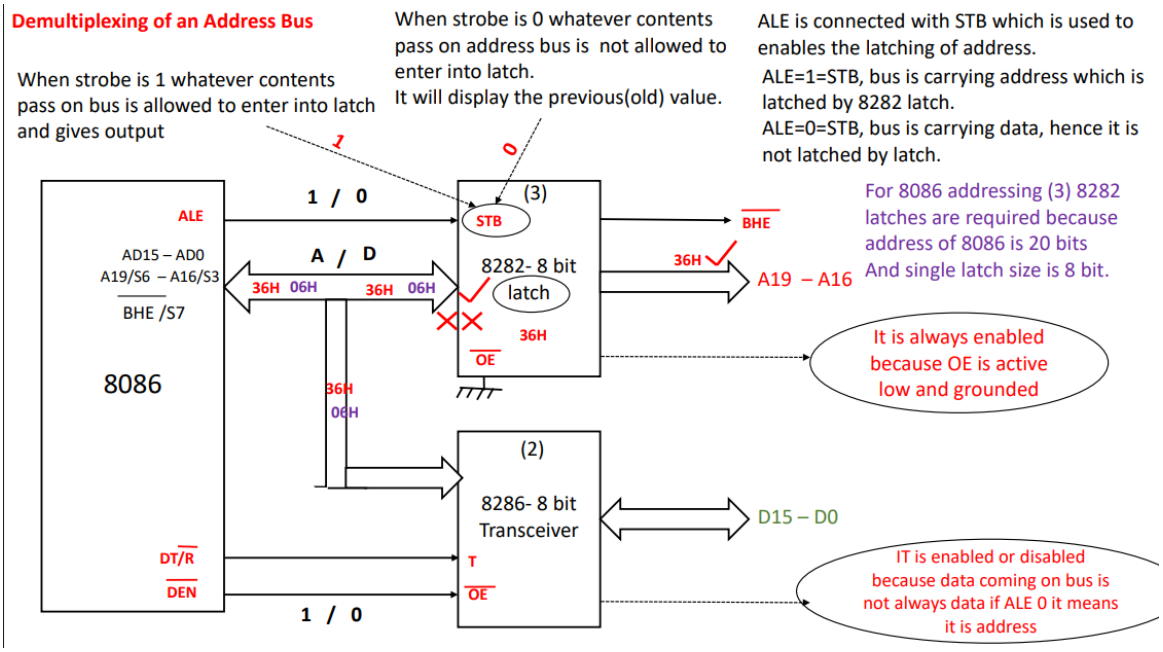
| Pin / Signal | Use – Only 2 Points |
|---|---|
| M/IO̅ | 1. High → Memory, Low → I/O.<br>2. Helps external circuits identify operation type. |
| RD̅ | 1. Active low read signal.<br>2. Makes memory/I/O put data on bus. |
| WR̅ | 1. Active low write signal.<br>2. CPU sends data to memory/I/O. |
| ALE | 1. Latches address from multiplexed bus.<br>2. High when valid address present. |
| INTA̅ | 1. Acknowledges interrupt request.<br>2. Allows device to place interrupt vector. |
| DT/R | 1. Tells data direction.<br>2. High = transmit, Low = receive. |
| DE̅N̅ | 1. Enables data transceivers.<br>2. Controls data bus flow. |
| HOLD | 1. External request for bus.<br>2. CPU releases bus on accept. |
| HLDA | 1. CPU grants bus to device.<br>2. Comes after HOLD is accepted. |
| READY | 1. Inserts wait states.<br>2. Syncs CPU with slow devices. |
| RESET | 1. Initializes CPU.<br>2. Sets CS:IP = FFFF0H. |
| CLK | 1. Provides timing.<br>2. Synchronizes all CPU actions. |

## Maximum Mode – Pins & Uses (2 marks each)

| Pin / Signal | Use – Only 2 Points |
|---|---|
| S2, S1, S0 | 1. Status lines for operation type.<br>2. Example: 000 = INT Acknowledge. |
| RQ/GT0 | 1. Highest priority bus request.<br>2. Same pin used for grant. |
| RQ/GT1 | 1. Lower priority bus request.<br>2. Same pin used for grant. |
| LOCK̅ | 1. Prevents bus access by others.<br>2. Used in critical instructions. |
| READY | 1. Inserts wait states.<br>2. Allows sync with slow devices. |
| RESET | 1. Initializes CPU.<br>2. Loads CS:IP = FFFF0H. |
| CLK | 1. Provides clock signal.<br>2. Controls timing of all ops. |

**Q.4)Explain following methods with the help of diagram** Each for 3 marks (1 mk for diagram , 2 mks for explanation)

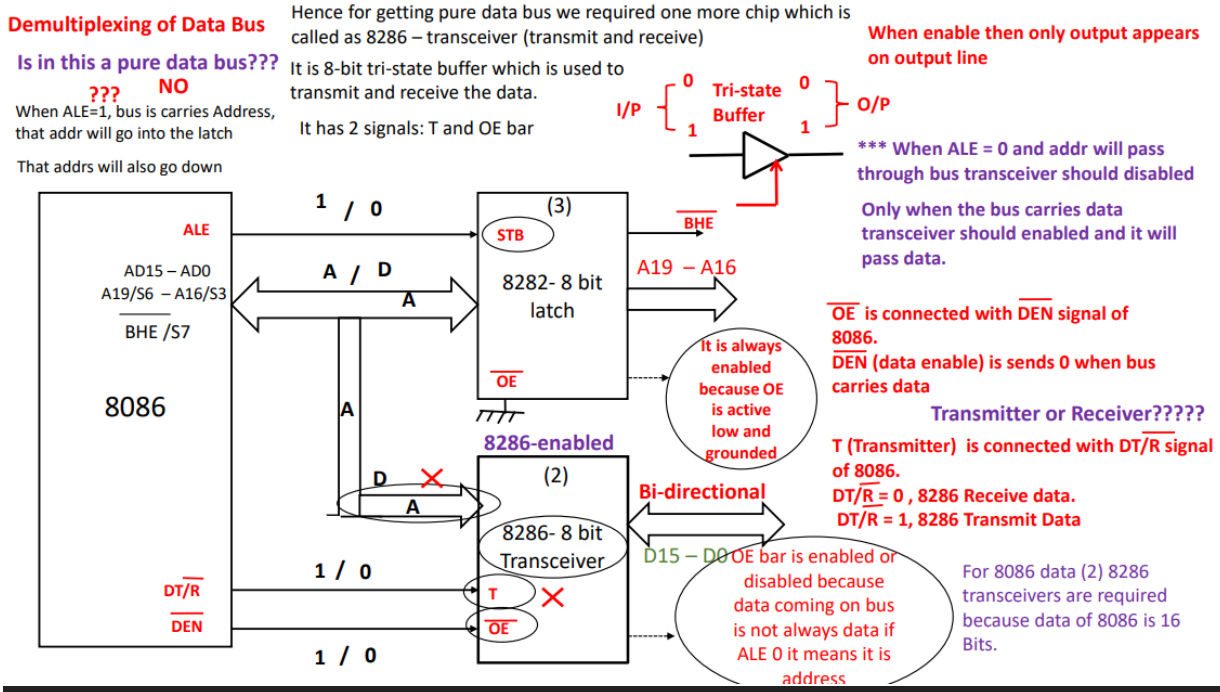
## 1. Demultiplexing of Address Bus




**Demultiplexing of Address Bus in 8086**

1. In 8086, **AD15–AD0** lines are multiplexed to carry address and data at different times.
2. **8282 latches** are used to store the address when **ALE = 1.**
3. When **ALE = 0**, the bus carries data while the latch holds the previous address.
4. Higher address lines **A19–A16** and **BHE** are also latched using 8282.
5. Three 8282 latches are required for the 20-bit address bus of 8086.

## 2. Demultiplexing of Data Bus

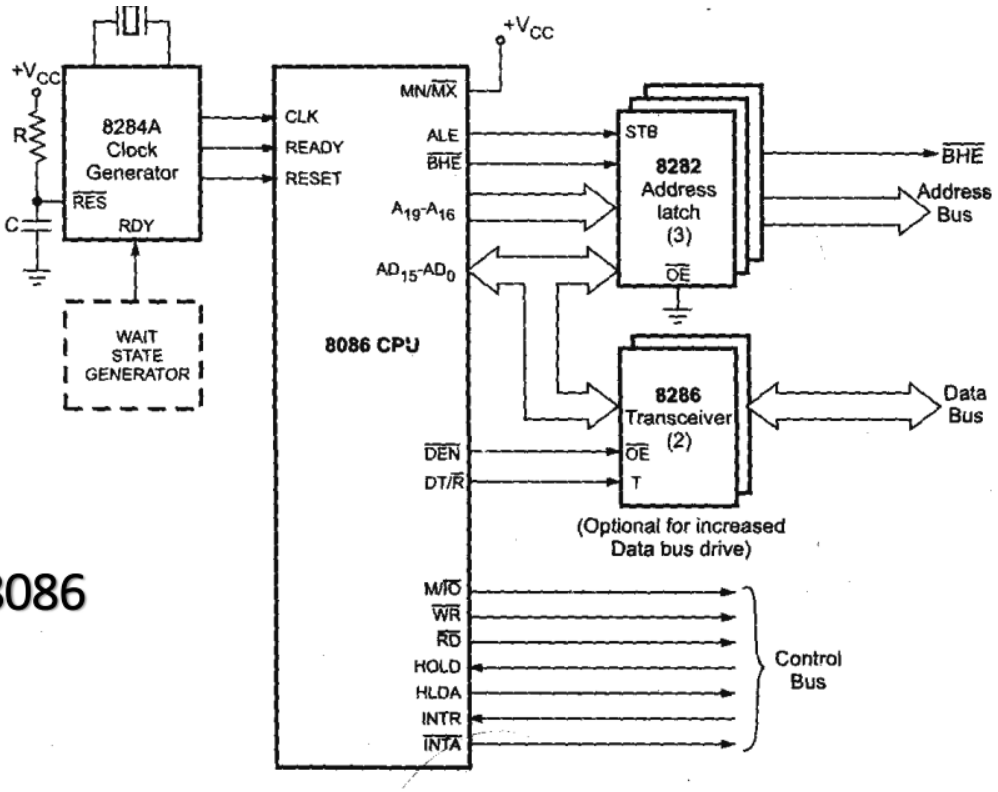



**Demultiplexing of Data Bus in 8086**

1. In 8086, **AD15–AD0** lines are multiplexed to carry address (ALE = 1) and data (ALE = 0).
2. To get a pure data bus, **8286 bidirectional transceivers** are used.
3. **OE̅** of 8286 is connected to **DEN̅** so that the transceiver is enabled only during data transfer.
4. **T** pin of 8286 is connected to **DT/R̅** to control direction: transmit when DT/R̅ = 1, receive when DT/R̅ = 0.
5. Two 8286 transceivers are required for the 16-bit data bus of 8086.

## Q.5 Draw and explain Minimum mode of 8086(6/8)

# Minimum mode of 8086



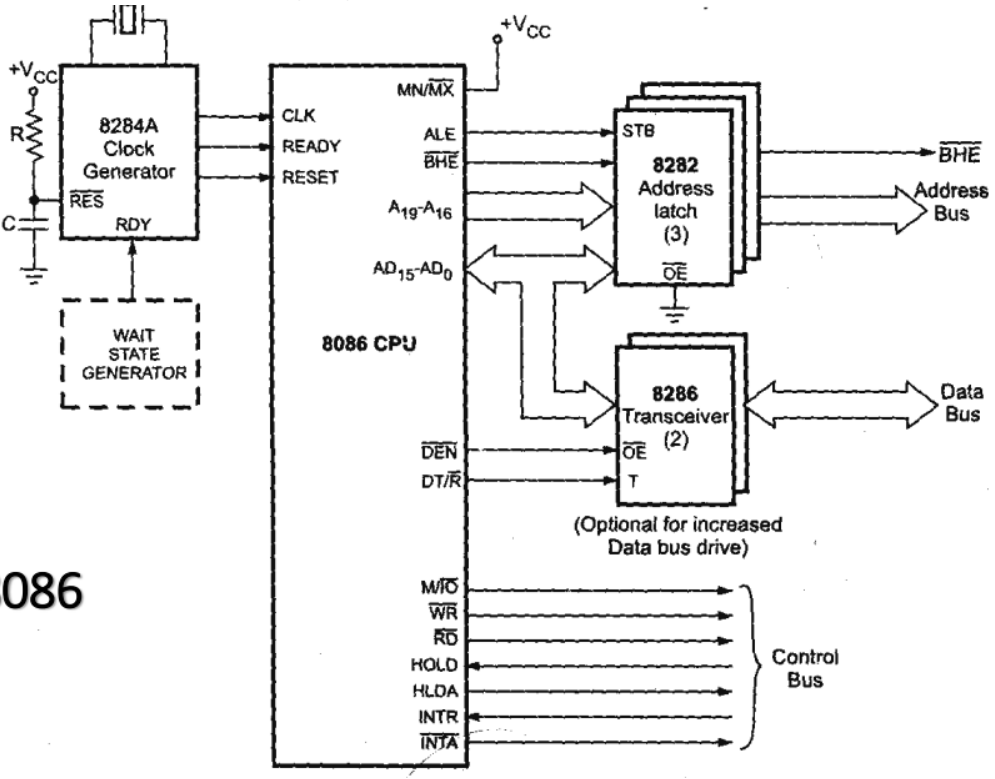

---

1. **Address/Data Lines**
   - **AD15–AD0** carry address when **ALE = 1**, and data when **ALE = 0**.
   - **A19–A16 + BHE** give the remaining address lines for the 20-bit address bus.
2. **Control Signals**
   - **M/IO̅** – High for memory, Low for I/O operation.
   - **RD̅ / WR̅** – Active-low signals for read/write operations.
   - **ALE** – Latches the address from AD lines.
   - **DEN̅ / DT/R̅** – Enable and set direction for data transceivers.
   - **INTA̅** – Acknowledges an interrupt request.
3. **Bus Control**
   - **READY** – Synchronizes CPU with slower devices by inserting wait states.
   - **HOLD / HLDA** – Used for bus request and bus grant between CPU and external devices.
4. **Operation**
   - During address phase, lower lines carry the address, latched using ALE.
   - During data phase, same lines carry data for read/write operations.
   - Control signals ensure correct coordination between CPU, memory, and I/O devices.
5. **Application**
   - Used in **small, single-processor systems** like microcomputers where cost and complexity must be minimal.

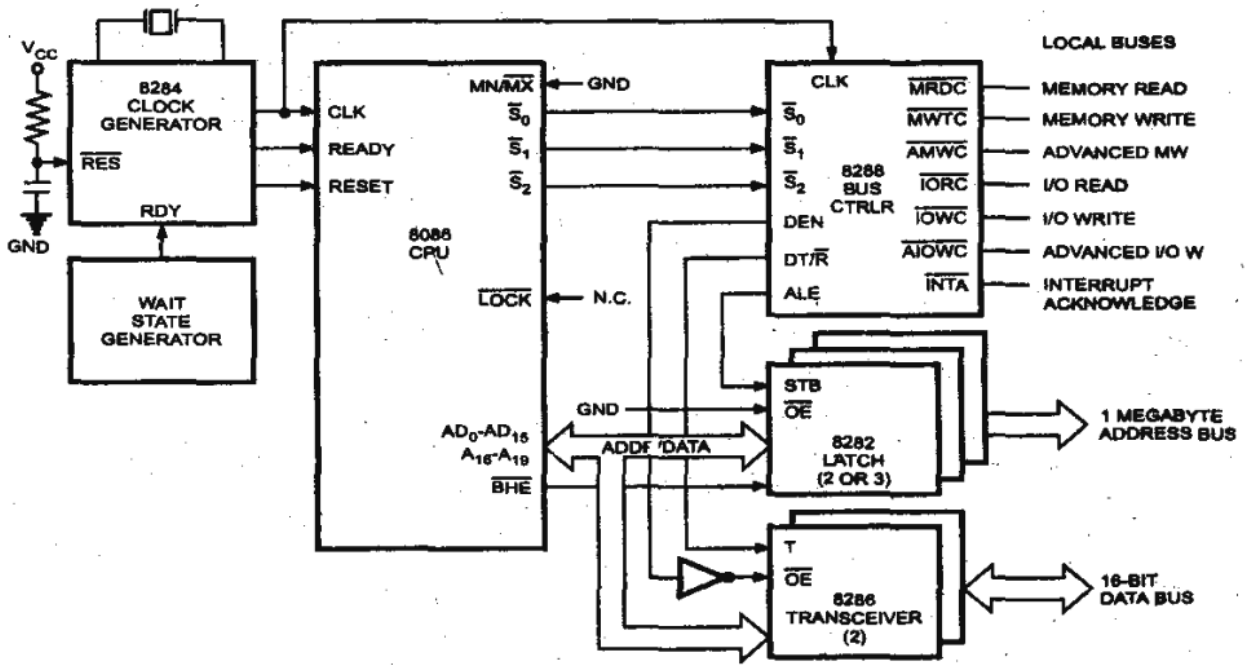
**Summary:**

In minimum mode, 8086 directly generates all control signals, uses ALE for demultiplexing address/data, and communicates with memory/I/O without any external bus controller, making it ideal for single-processor systems.

**Q.6 Draw minimum/Maximum mode of 8086(6/8)**

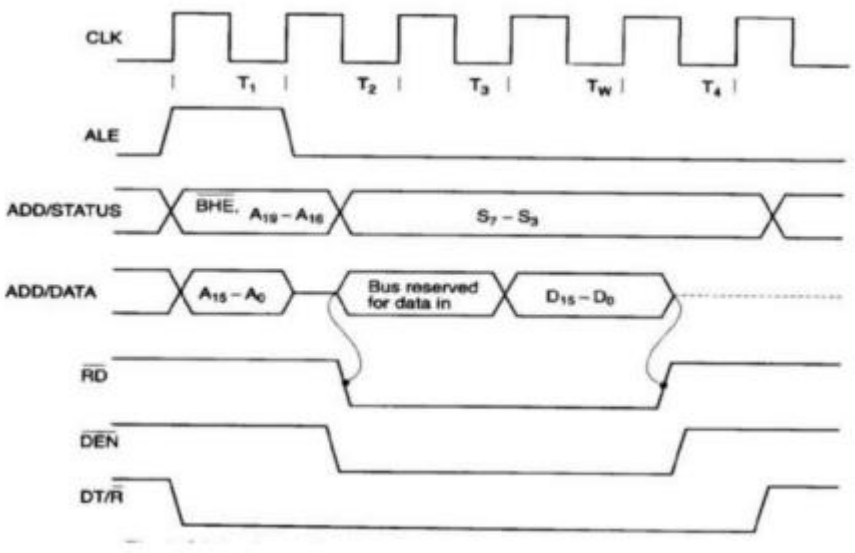



## Minimum mode of 8086




## Maximum mode of 8086

**Q.7 Draw timing diagram for any one machine cycle of minimum mode of 8086(4)**




**Q.8 Explain the use of 8288 bus controller in maximum mode of 8086(4)**

**Uses of 8288 Bus Controller in Maximum Mode of 8086:**

1. **Generates Control Signals** – The 8288 decodes the status signals ($S_2$, $S_1$, $S_0$) from the 8086 and generates necessary control signals like `MEMR`, `MEMW`, `IOR`, `IOW`, `INTA`, etc., for memory and I/O devices.
2. **Coordinates Multiple Processors** – In maximum mode, 8086 can work with coprocessors (like 8087) or multiple processors; the 8288 helps manage control signals for proper bus sharing.
3. **Provides Timing Control** – Ensures that read/write operations are properly timed by generating commands at the correct phase of the bus cycle.
4. **Reduces CPU Pin Count** – 8086 in maximum mode does not directly output all control signals; 8288 handles their generation, simplifying CPU design.
5. **Supports High-Speed Operation** – Improves overall system performance by providing fast, accurate, and synchronized control signals for memory and I/O communication.

**Q.9 State the function following signals ALE,BHE,DT/R,DEN,IOWR,IORD,MR,MW, HOLD,HOLDA,LOCK (4/6/8)**

- **ALE (Address Latch Enable)** – Indicates that the address bus carries a valid address.
- **BHE (Bus High Enable)** – Enables the high byte of the data bus (D8–D15) during a memory or I/O operation.
- **DT/R (Data Transmit/Receive)** – Determines whether the 8086 is sending or receiving data on the bus.
- **DEN (Data Enable)** – Enables the data bus transceivers for read or write operation.
- **IOWR (I/O Write)** – Indicates a write operation to an I/O device.
- **IORD (I/O Read)** – Indicates a read operation from an I/O device.
- **MR (Memory Read)** – Signals that the 8086 is reading data from memory.
- **MW (Memory Write)** – Signals that the 8086 is writing data to memory.
- **HOLD** – Request from another device to take control of the system bus.
- **HOLDA** – Acknowledges the HOLD request and relinquishes control of the bus.
- **LOCK** – Locks the bus to prevent other devices from accessing it during a critical operation.