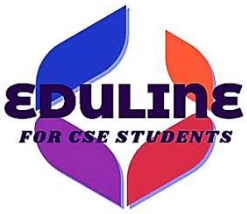
# MODULE 3

## STACK & INTERRUPTS

CO – Students will be able to Demonstrate interrupts, its handling and programming in 8086



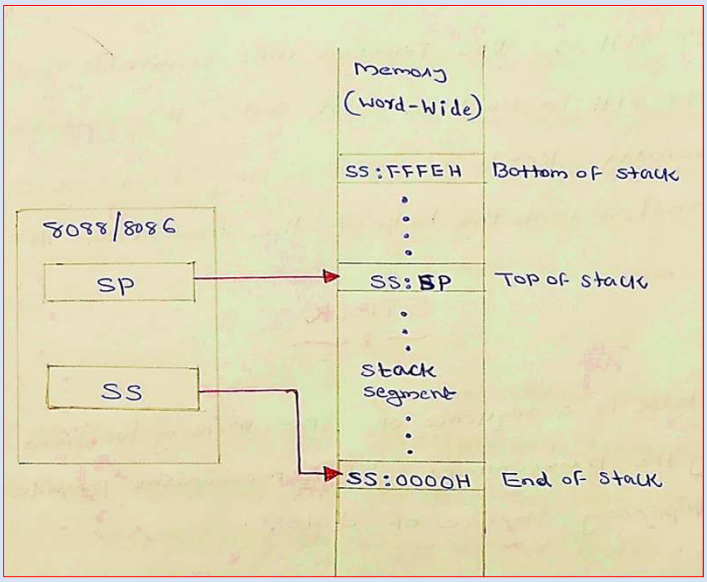

Prepared By Mr. EBIN PM, Chandigarh University, Punjab

## STACK STRUCTURE OF 8086

- Stack is a sequence of RAM memory locations defined by the programmer. Stack mechanism provide a temporary storage of data.
- Stack is a block of memory locations which is accessed using the SP and SS register. Stack is a Last In First Out (LIFO) data segment.
- In the case of subroutine and interrupt, the address of re-entry in to the main program (return address) may be stored in the stack.
- The process of storing the data in the stack is called Pushing into the stack.
- The process of transferring the data back from the stack to the CPU register is called popping off the stack.

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 2

## Stack Segment of Memory

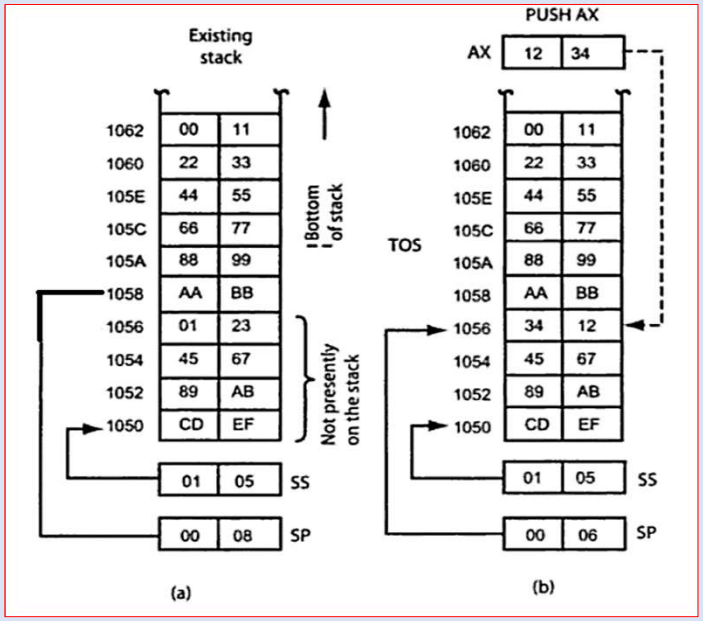





- **Top of stack** ⟶ Physical address of the last storage location in the stack to which data were pushed. This address is obtained from the content of SS and SP ( SS: SP)
- **Bottom of stack** ⟶ At the microcomputer's startup, the value in SP is initialized to FFFEH. Combining this value with the current value in SS gives the highest address word location in the stack (SS: FFFEH), called bottom of the stack.
- 8086 can push data and address information on to the stack from its internal register or a storage location in memory.
- Data transferred to and from the stack are word-wide, not byte-wide

- Each time a word is to be pushed on to the stack, the value in SP is automatically decremented by two
- Therefore, stack grows down in memory from the bottom of the stack (SS: FFFEH) towards the end of the stack (SS: 0000H)
- Each time a word is popped from the top of stack, the value in SP is automatically incremented by two





- Above figure , is the state of stack prior to execution of the PUSH AX instruction.
- The stack segment register contains 105H. Bottom of stack address is derived from SS and offset FFFEH

  Bottom of stack address = 1050H + FFFEH

  = 1104EH

- SP contains 0008H. Top of stack can be derived from SS and SP

  Top of stack = 1050H + 0008H

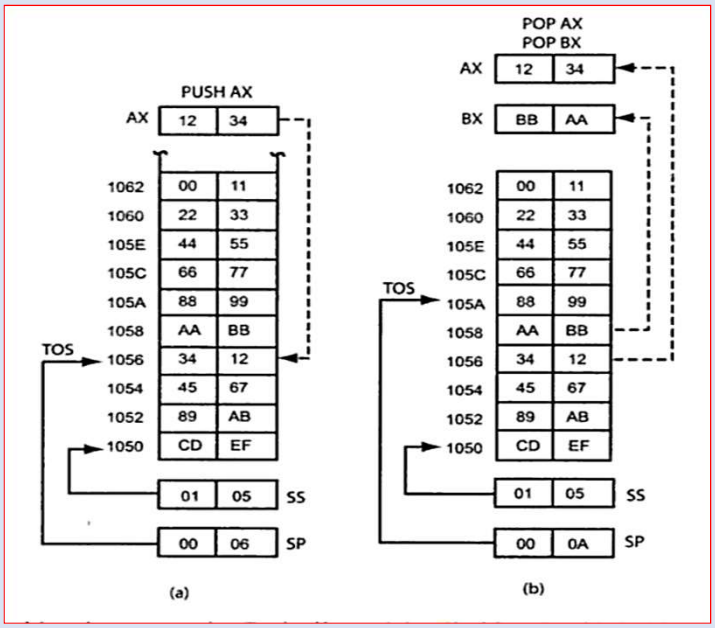
  = 1058H



## ❖PUSH OPERATION

- AX initially contains the number 1234H.
- Execution of push instruction causes the stack pointer to be decremented by two.
- Therefore the next stack access is to the location corresponding to address 1056H. This location is where the value in AX is pushed.
- The MSB of AX (ie,12H) resides in memory address 1057H and LSB of AX (ie, 34H) is held in memory address 1056H

## ❖POP OPERATION






- The execution of the first instruction POP AX, causes the 8086 to read the value from the Top of the stack and put it in to AX register as 1234H
- SP is incremented to give 0008H and the other read operation POP BX, causes the value BBAAH to be loaded into the BX register.
- SP is incremented once more and now equals 000AH. Therefore , the new top of stack is at address 105AH

- Stack is an area of memory for keeping temporary data. Stack is used by **CALL** instruction to keep return address for procedure, **RET** instruction gets this value from the stack and returns to that offset.
- Quite the same thing happens when **INT** instruction calls an interrupt, it stores in stack flag register, code segment and offset. **IRET** instruction is used to return from interrupt call.
- We can also use the stack to keep any other data, there are two instructions that work with the stack:

  **PUSH** - stores 16 bit value in the stack.

  **POP** - gets 16 bit value from the stack.



Syntax for **PUSH** instruction:

```
PUSH REG
PUSH SREG
PUSH memory
PUSH immediate
```

**REG**: AX, BX, CX, DX, DI, SI, BP, SP.

**SREG**: DS, ES, SS, CS.

**memory**: [BX], [BX+SI+7], 16 bit variable, etc...

**immediate**: 5, -24, 3Fh, 10001101b, etc...

Syntax for **POP** instruction:

```
POP REG
POP SREG
POP memory
```

**REG**: AX, BX, CX, DX, DI, SI, BP, SP.

**SREG**: DS, ES, SS, (except CS).

**memory**: [BX], [BX+SI+7], 16 bit variable, etc...

- **PUSH** and **POP** work with 16 bit values only

- It is very important to do equal number of **PUSH**s and **POP**s, otherwise the stack maybe corrupted and it will be impossible to return to operating system.
- As you already know we use **RET** instruction to return to operating system, so when program starts there is a return address in stack (generally it's 0000H).
- **PUSH** and **POP** instruction are especially useful because we don't have too much registers to operate with, so here is a trick:

➢Store original value of the register in stack (using **PUSH**).

➢Use the register for any purpose.

➢Restore the original value of the register from stack (using **POP**).



**Example:**

```
ORG   100h

MOV   AX, 1234h
PUSH  AX        ; store value of AX in stack.

MOV   AX, 5678h  ; modify the AX value.

POP   AX        ; restore the original value of AX.

RET

END
```

- Another use of the stack is for exchanging the values.
- The exchange happens because stack uses **LIFO** (Last In First Out) algorithm, so when we push **1212h** and then **3434h**, on pop we will first get **3434h** and only after it **1212h**.

```
ORG   100h

MOV   AX, 1212h   ; store 1212h in AX.
MOV   BX, 3434h   ; store 3434h in BX


PUSH  AX          ; store value of AX in stack.
PUSH  BX          ; store value of BX in stack.

POP   AX          ; set AX to original value of BX.
POP   BX          ; set BX to original value of AX.

RET

END
```



- The stack memory area is set by **SS** (Stack Segment) register, and **SP** (Stack Pointer) register. Generally operating system sets values of these registers on program start.

➢"**PUSH *source***" instruction does the following:
Subtract **2** from **SP** register.
Write the value of ***source*** to the address **SS:SP**.

➢"**POP *destination***" instruction does the following:
Write the value at the address **SS:SP** to ***destination***.
Add **2** to **SP** register.

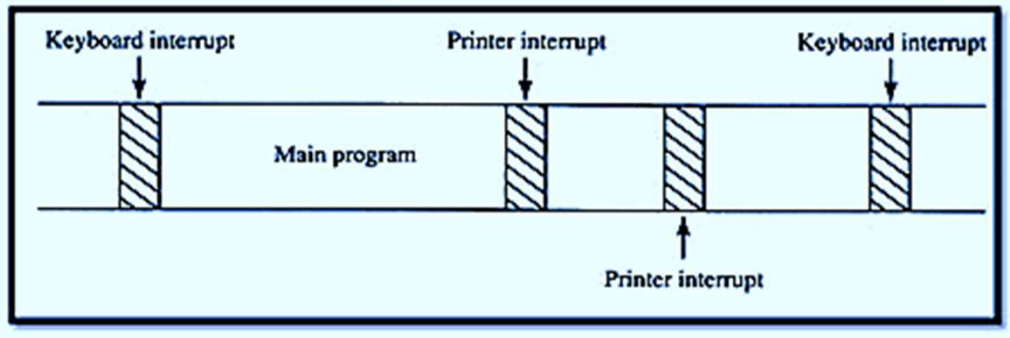
- The current address pointed by **SS:SP** is called **the top of the stack**.

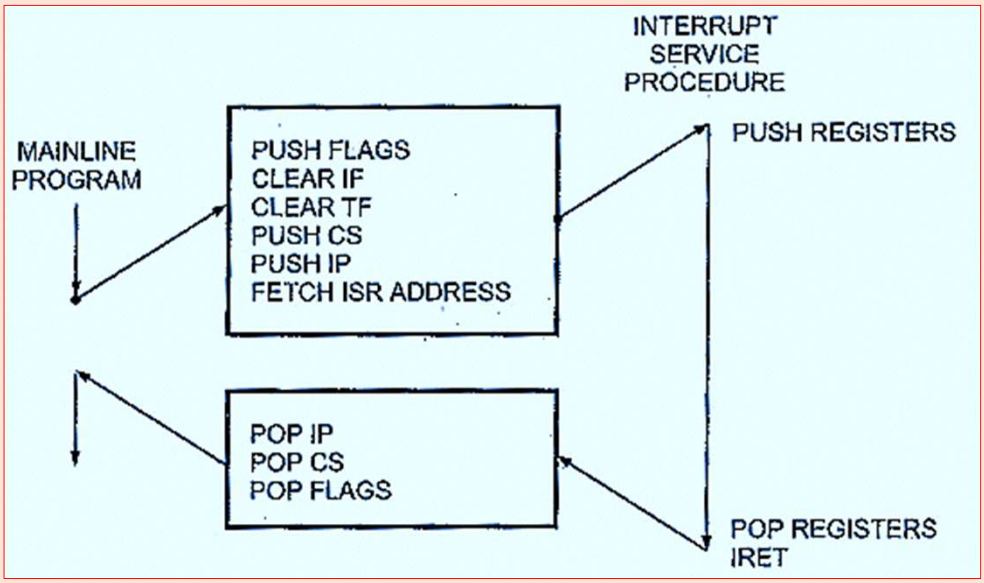
# INTERRUPTS

- **Interrupt** – break the sequence of operation
- While the CPU executing a program, an interrupt breaks the normal sequence of execution of instructions, divert its execution to some other program called **Interrupt Service Routine (ISR)**
- After executing ISR, the control is transferred back again to the main program
- Whenever a number of devices interrupt the CPU at a time, and if the processor is able to handle them properly, it is said to have **multiple interrupt processing capability**.



- The following time line shows typing on a keyboard, a printer removing data from memory, and a program executing.
- The keyboard interrupt service procedure, called by the keyboard interrupt, and the printer interrupt service procedure called by the printer interrupt. Each take little time to execute



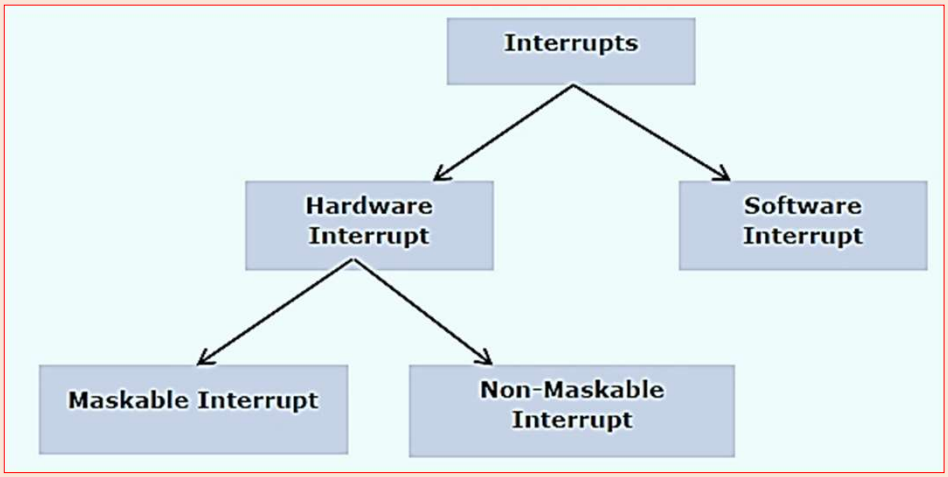





- It decrements stack pointer by 2 and pushes the flag register on the stack.
- It disables the INTR interrupt input by clearing the interrupt flag in the flag
- It resets the trap flag in the flag register.
- It decrements stack pointer by 2 and pushes the current code segment register contents on the stack.
- It decrements stack pointer by 2 and pushes the current instruction pointer contents on the stack.
- It does an indirect far jump at the start of the procedure by loading the CS and IP values for the start of the interrupt service routine (ISR).
- An IRET instruction at the end of the interrupt service procedure returns execution to the main program.

# TYPES OF INTERRUPTS

- In 8086, there are two interrupt pins. **NMI** and **INTR**






- NMI is a **nonmaskable interrupt** input pin which means that any interrupt request at NMI input cannot be masked or disabled by any means.
- The INTR interrupt may be masked using the Interrupt Flag (IF)

## ❖Hardware Interrupts

- Hardware interrupts are those interrupts which are caused by any peripheral device by sending a signal through a specified pin to the microprocessor.
- The 8086 has two hardware interrupt pins, i.e. NMI and INTR. NMI is a non-maskable interrupt and INTR is a maskable interrupt having lower priority. One more interrupt pin associated is INTA called interrupt acknowledge.

## a) NMI (Non Maskable Interrupt)

- It is a single nonmaskable interrupt pin (NMI) having higher priority than the maskable interrupt request pin (INTR)and it is of type 2 interrupt.

➢When this interrupt is activated, these actions take place:

1. Completes the current instruction that is in progress.
2. Pushes the Flag register values on to the stack.
3. Pushes the CS (code segment) value and IP (instruction pointer) value of the return address on to the stack.
4. IP is loaded from the word location 00008H.
5. CS is loaded from the word location 0000AH.
6. Interrupt flag and trap flag are reset to 0.



## b) INTR (Maskable Interrupt)

- The INTR is a maskable interrupt
- The INTR interrupt is activated by an I/O port.

➢The actions are taken by the microprocessor:

1. First completes the current instruction.
2. Activates INTA output and receives the interrupt type, say X.
3. Flag register value, CS value of the return address and IP value of the return address are pushed on to the stack.
4. IP value is loaded from the contents of word location X × 4
5. CS is loaded from the contents of the next word location.
6. Interrupt flag and trap flag is reset to 0

- If the **IF (Interrupt Flag)** is reset, the processor will not serve any interrupt appearing at this pin.
- If the IF flag is set, the processor is ready to respond to any INTR interrupt
- Once the processor respond to an INTR signal, the IF is automatically reset.
- If one wants the processor to further respond to any type of INTR signal, the IF should again be set.



## ❖Software Interrupts

- These are instructions that are inserted within the program to generate interrupts.
- There are 256 software interrupts in 8086 microprocessor.
- The instructions are of the format "**INT type**" where type ranges from 00 to FF.
- The starting address ranges from 00000 H to 003FF H.
- These are 2 byte instructions.
- IP is loaded from **type* 04 H** and CS is loaded from the next address give by **(type * 04) + 02 H**

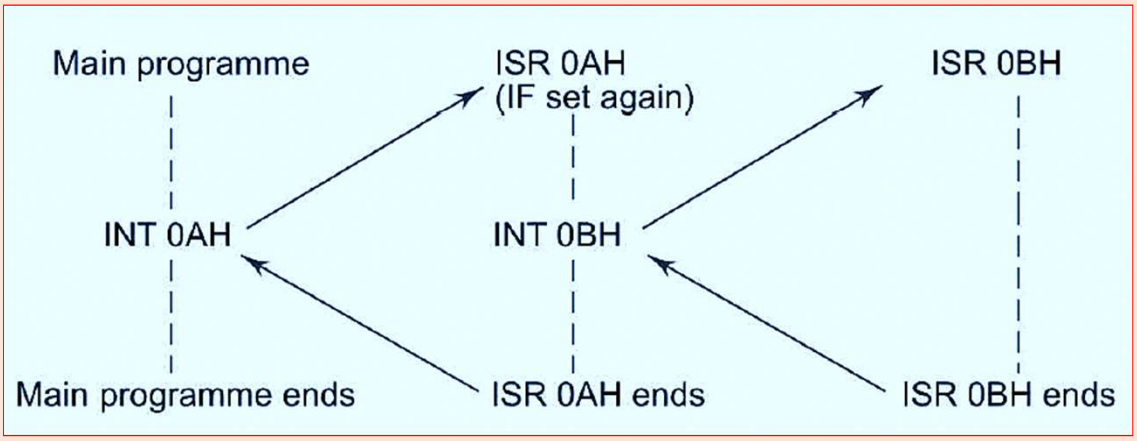
- Some important software interrupts are:

1. ***TYPE 0*** corresponds to division by zero(0).
2. ***TYPE 1*** is used for single step execution for debugging of program.
3. ***TYPE 2*** represents NMI and is used in power failure conditions.
4. ***TYPE 3*** represents a break-point interrupt.
5. ***TYPE 4*** is the overflow interrupt.

- The interrupts from Type 5 to Type 31 are reserved for other advanced microprocessors.



## ❖ Transfer of control of nested interrupt



## Interrupt Service Routine (ISR)

- For every interrupt, there must be an interrupt service routine (ISR), or **interrupt handler**.
- When an interrupt is invoked, the microprocessor runs the interrupt service routine.
- For every interrupt, there is a fixed location in memory that holds the address of its ISR.
- The group of memory locations set aside to hold the addresses of ISRs is called the interrupt vector table.



## Interrupt Vectors

- The starting address of an ISP(Interrupt Service Provider) is called the interrupt vector. There fore the table is called interrupt vector table.
- The interrupt vector table contains 256 four byte entries, containing the CS: IP
- Interrupt vector table is located in the first 1024 bytes of memory at addresses 000000H – 0003FFH .
- It contains the address of the interrupt service provider

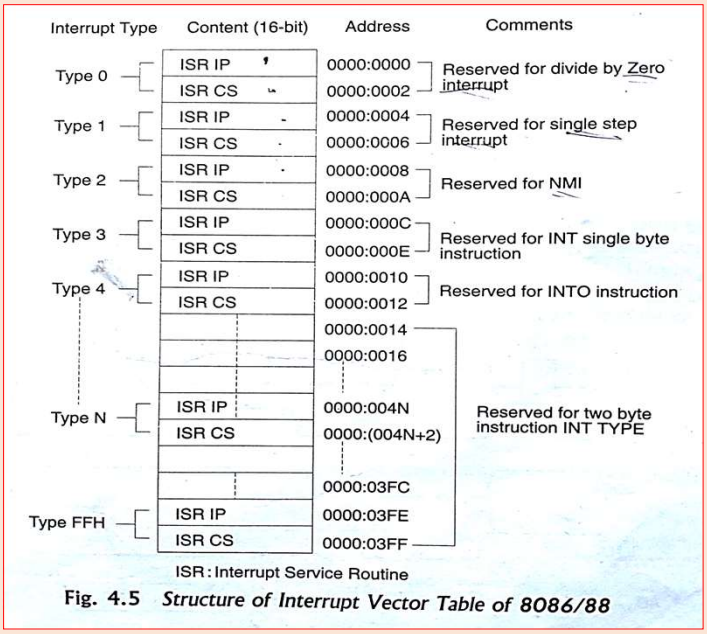



Fig. 4.5 *Structure of Interrupt Vector Table of 8086/88*



# 8259 PROGRAMMABLE INTERRUPT CONTROLLER (PIC)

- Programmable interrupt controllers are used to enhance the number of interrupts of a microprocessor.
- 8259 is a programmable interrupt controller which shows compatibility with 8086 microprocessor.
- It is also known as a **priority interrupt controller** and was designed by **Intel** to increase the interrupt handling ability of the microprocessor.
- An 8259 PIC never services an interrupt; it simply forwards the interrupt to the processor for the execution of interrupt service routine.

## ❖Need of Programmable Interrupt Controller

- Whenever an interrupt occurs then the microprocessor suspends the current program and switches to the Interrupt Service Routine (ISR).
- We know 8085/86 has 5 interrupts, which are: Trap, RST7.5, RST6.5, RST5.5 and INTR.
- Among all these, only INTR is a non-vectored type of interrupt, rest are vectored interrupts
- Vectored interrupts are those interrupts whose ISR address is known to the processor. Or we can say in case of vectored interrupts, the processor holds the address of the memory location where ISR is stored.



- But in case of non-vectored interrupts, the interrupt generating device provides the ISR address to the microprocessor.
- An 8085/86 has 5 major interrupts for which a fixed number of lines are present in the chip. But there are many devices connected to a processor. So, for such a case the processor must have more number of lines to handle several interrupts.
- But it is not practically possible to increase the number of lines each time with the increase in the number of interrupts.
- So, to overcome this problem 8259 PIC chip is used. 8259 allows the combining of multiple interrupts and providing them to the processor based on priority through a common line.

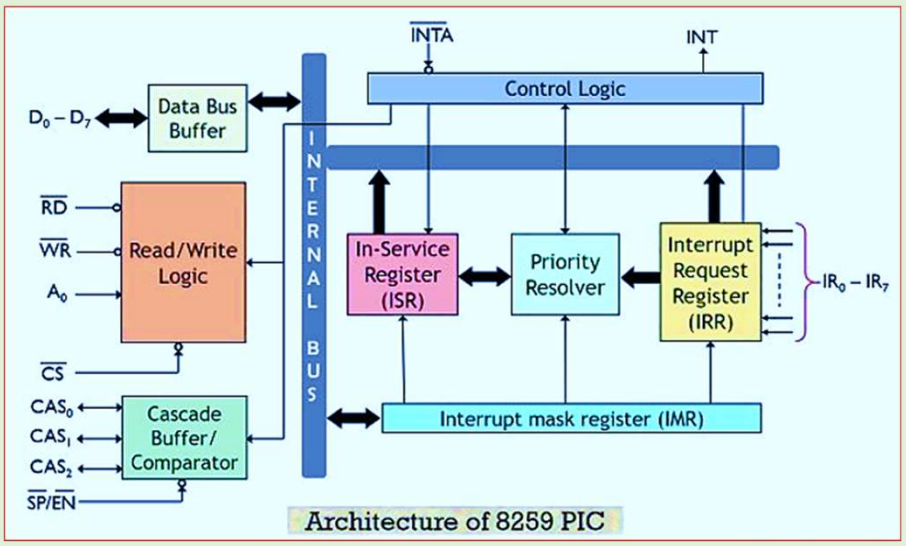
- The processor holds the address of ISR in case of vectored interrupts. So, it is not possible to combine a non-vectored interrupt with a vectored one.
- 8259 is used to combine various interrupts which are non-vectored in nature.
- If the processor gets two INTR signals at the same time but how does the processor get to know that from where the interrupt is generating and where to send the INTA in order to have the ISR address.
- This shows the necessity of 8259. The programmable interrupt controller tells the microprocessor about the interrupt. Basically the external devices initially interrupt the 8259 and further the 8259 interrupts the microprocessor.



## ❖Features of 8259

- The 8259 programmable interrupt controller has 8 interrupt pins and can handle 8 interrupt inputs.
- The priority of interrupts in 8259 can be programmed.
- A single 8259 can handle 8 interrupt inputs but by cascading multiple 8259, it can handle maximal 64 interrupt inputs.
- 8259 allows individual masking of each generated interrupt using interrupt mask register.
- 8259 can handle either edge-triggered or level-triggered interrupt request at a time.
- If multiple interrupts are generated, then 8259 holds the status of interrupts that are masked, in-service and pending.

- It reduces the software and real-time overhead generated due to handling multilevel priority interrupts.



Architecture of 8259 PIC



# INTERFACING MEMORY WITH 8086

- When we are executing any instruction, we need the microprocessor to access the memory for reading instruction codes and the data stored in the memory.
- For this, both the memory and the microprocessor requires some signals to read from and write to registers.
- The interfacing process includes some key factors to match with the memory requirements and microprocessor signals.
- The interfacing circuit therefore should be designed in such a way that it matches the memory signal requirements with the signals of the microprocessor

- In memory interfacing: 8 bit data line, 16 bit address line , control signals are connected to corresponding lines of memory IC.
- Almost all systems contain four common types of memory:
  - Read only memory (ROM)
  - Flash memory (EEPROM)
  - Static Random access memory (SRAM)
  - Dynamic Random (DRAM)
- Before attempting to interface memory to the microprocessor, it is essential to understand the operation of memory components.



## ❖Read-only memory (ROM)

- Read-only memory (**ROM**) permanently stores programs/data resident to the system, and must not change when power disconnected.
- Often called nonvolatile memory, because its contents *do not* change even if power is disconnected.
- The **EPROM** (erasable programmable read-only memory) is programmed in the field on a device called an EPROM programmer.
- Also erasable if exposed to high-intensity ultraviolet light, depending on the type of EPROM.

- The **PROM** (programmable read-only memory) is also programmed. Once it is programmed, it cannot be erased.
- A newer type of read-mostly memory (**RMM**) is called the flash memory.
- Flash memory is also often called an **EEPROM** (electrically erasable programmable ROM) or **EAROM** (electrically alterable ROM) or a **NOVRAM** (nonvolatile RAM)
- Electrically erasable in the system, but they require more time to erase than normal RAM.
- The flash memory device is used to store setup information for systems such as the video card in the computer.



## ❖Static Random Access Memory (SRAM)

- A Static RAM is a volatile memory device which means that the contents of the memory array will be lost if power is removed.
- Unlike a dynamic memory device, the static memory does not require a periodical refresh cycle and generally runs much faster than a dynamic memory device.
- Static RAM is used when the size of the read/write memory is relatively small, today, a small memory is less than 1M byte.
- The main difference between ROM and RAM is that RAM is written under normal operation, whereas ROM is programmed outside the computer and normally is only read.

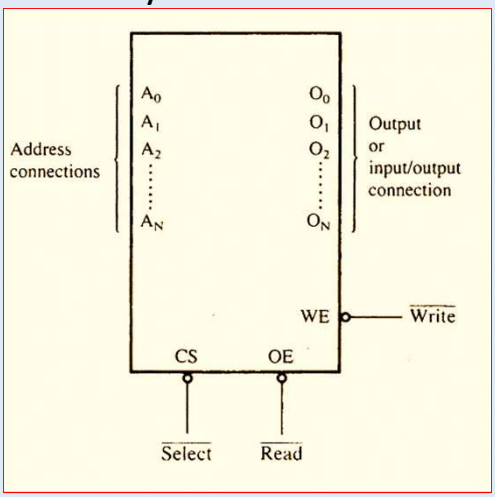
## ❖Dynamic Random Access Memory (DRAM)

- Available up to 256M X 8 (2G bits).
- DRAM is essentially the same as SRAM, except that it retains data for only 2 or 4 ms on an integrated capacitor.
- After 2 or 4 ms, the contents of the DRAM must be completely rewritten *(refreshed)*, because the capacitors, which store a logic 1 or logic 0, lose their charges.



- Memory Pin Connections in the following figure shows a general form diagram of ROM and RAM pins.
- Pin connections common to all memory devices are:

♦ Address connections
♦ Data connections
♦ Selection connections
♦ Control connections



1. **Address connections:** All memory devices have address inputs that select a memory location within the memory device. Address inputs are labeled from A0 to An
2. **Data connections:** All memory devices have a set of data outputs or input/outputs. Today many of them have bi-directional common I/O pins.
3. **Selection connections:** Each memory device has an input that selects or enables the memory device. This kind of input is most often called a chip select ( $\overline{CS}$ ), chip enable ( $\overline{CE}$ ) or simply select ( $\overline{S}$ ) input.



**4. Control connections:**

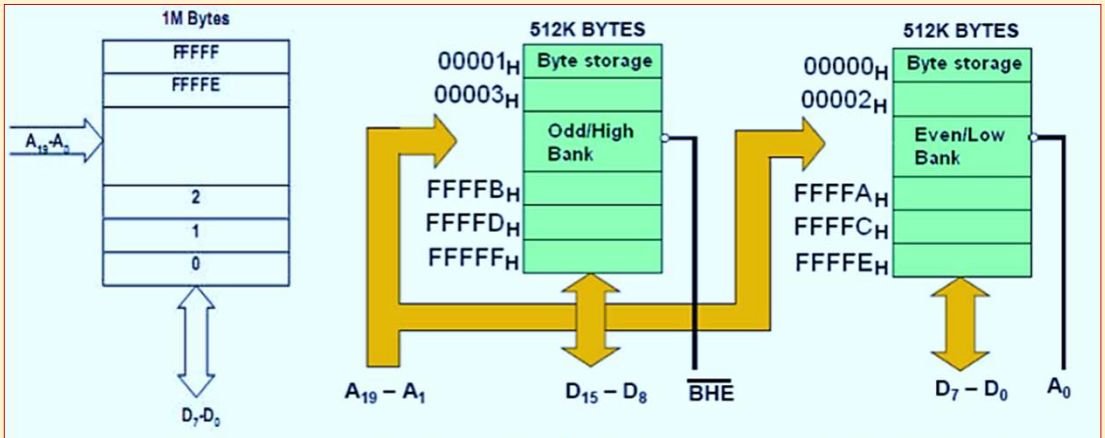
- A ROM usually has only one control input, while a RAM often has one or two control inputs.
- The control input most often found on the ROM is the output enable ($\overline{OE}$ ) or gate ($\overline{G}$ ), this allows data to flow out of the output data pins of the ROM.
- A RAM memory device has either one or two control inputs. If there is one control input it is often called WR .
- This pin selects a read operation or a write operation only if the device is selected by the selection input ( $\overline{CS}$ )

- The memory address space of the 8086-based microcomputers has different logical and physical organizations.
- Logically, memory is implemented as a single 1M × 8 memory bank.
- The byte-wide storage locations are assigned consecutive addresses over the range from 00000H through FFFFFH8086 Memory Organization
- Physically, memory is implemented as two independent 512 Kbyte banks:

  The low (even) bank

  The high (odd) bank





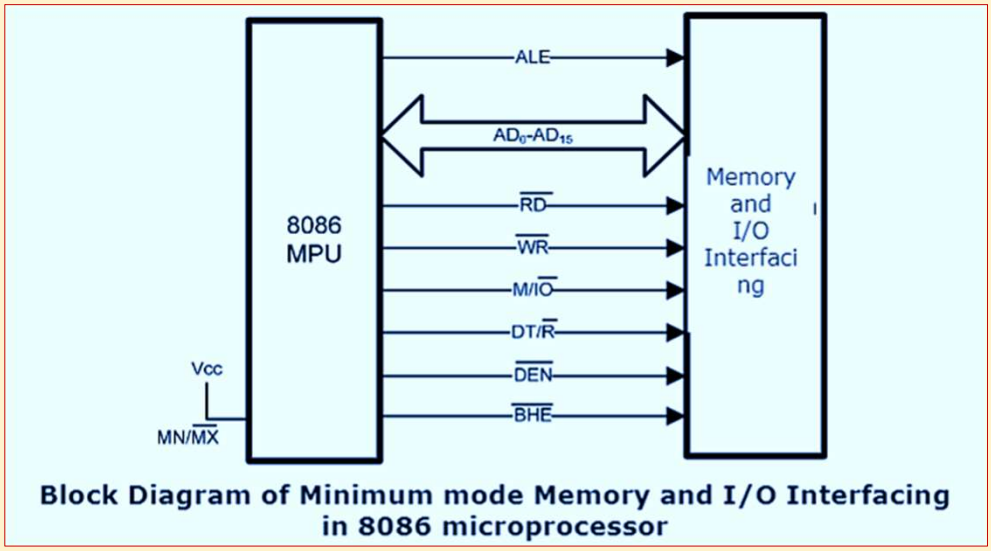

(a) Logical memory organization, and (b) Physical memory organization (high and low memory banks) of the 8086 microprocessor.

- To distinguish between odd and even bytes, the CPU provides a signal called BHE (bus high enable).
- BHE and A0 are used to select the odd and even byte, as shown in the table below

| $\overline{BHE}$ | A0 | Function |
|---|---|---|
| 0 | 0 | Choose both odd and even memory bank |
| 0 | 1 | Choose only odd memory bank |
| 1 | 0 | Choose only even memory bank |
| 1 | 1 | None is chosen |



## ❖Minimum mode Memory and I/O Interfacing



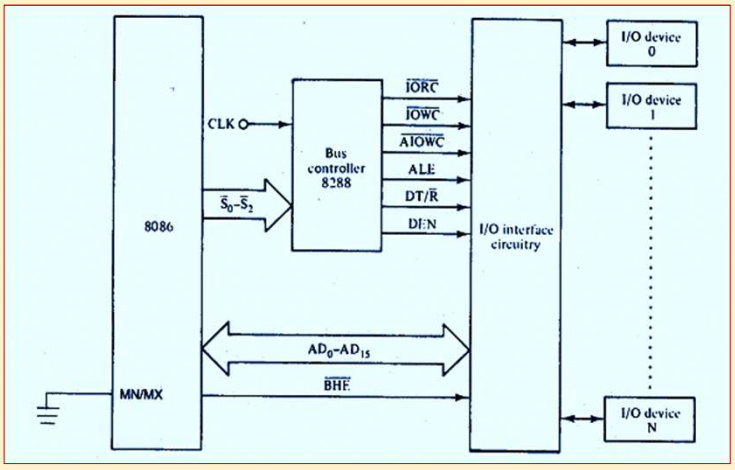

Block Diagram of Minimum mode Memory and I/O Interfacing in 8086 microprocessor

- The control signals provided to support the interface to the memory subsystem are ALE , M/$\overline{IO}$ , DT/$\overline{R}$ , RD, $\overline{WR}$ , DEN and BHE
- When Address latch enable (ALE) is logic 1 it signals that a valid address is on the bus.
- This address can be latched in external circuitry on the 1-to-0 edge of the pulse at ALE.
- M/$\overline{IO}$ (memory/IO) and DT/$\overline{R}$ tells external circuitry whether a memory or I/O transfer is taking place over the bus, and whether the 8086 will transmit or receive data over the bus.
- The bank high enable ($\overline{BHE}$) signal is used as a memory enable signal for the most significant byte half of the data bus, D8 through D15.
- The signals $\overline{WR}$ (write) and $\overline{RD}$ (read) identify that a write or read bus cycle is in progress.



- DEN (data enable), is also supplied. It enables external devices to supply data to the microprocessor

❖**Maximum mode Memory Interface**



- In maximum mode the 8086 not directly provides all control signal to support the memory interface.
- Instead, an external Bus Controller (8288) provides memory commands and control signals