THE STACK STRUCTURE OF 8086:

- The stack is a block of memory that may be used for **temporarily storing the contents of the registers** inside the CPU. It is a t**op-down data structure whose elements are accessed using the stack pointer (SP)** which gets decremented by two as we store a data word in the stack and gets incremented by two as we retrieve a data word from the stack back to the CPU register.
- The stack segment, like any other segment, may have a memory block of a **maximum of 64 Kbytes** locations and thus may overlap with any other segments. The stack Segment register (SS) contains the base address of the stack segment in the memory. In 8086 microprocessor-based system, the stack is created by loading a 16-bit base address in the Stack Segment (SS) register and a 16-bit offset address in Stack Pointer (SP). The 20-bit physical address of the stack is computed by multiplying the contents of the SS register by 10H and then adding the contents of SP to this product. Here the content of SP is the offset address of the stack.

SS	⇒ 5000 H					
SP	⇒ 2050 H					
SS	⇒	0101	0000	0000	0000	
10H * SS	⇒	0101	0000	0000	0000	0000
	+					
SP	⇒		0010	0000	0101	0000
Stack-top		0101	0010	0000	0101	0000
address		5	2	0	5	0

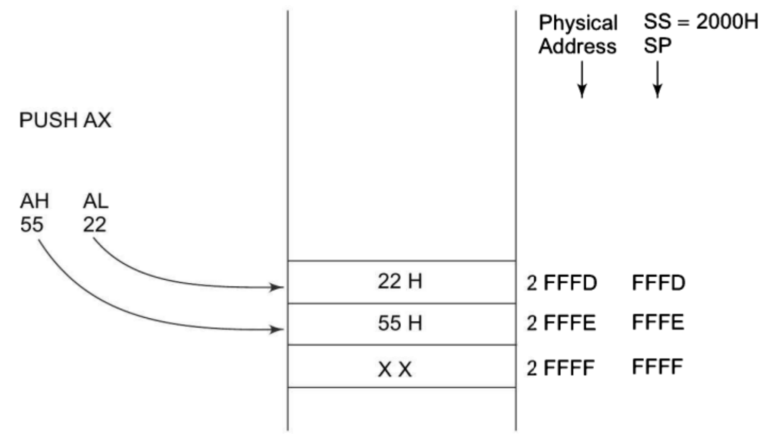
Upon reset, the SS register and SP are cleared to zero.

- **PUSH: Push to stack**

  The PUSH instruction decrements the stack pointer (SP) by 2 and copies a word from a specified source to the location in the stack segment to which the stack pointer points. The source of the word can be a general-purpose register, segment register, or memory. The stack segment register and the stack pointer must be initialized before this instruction can be used. PUSH can be used to save data on the stack so that it will not be destroyed by a procedure. This instruction does not affect any flag.

  Eg: MOV AX , 5522H
  PUSH AX
  Assume now the effective address (EA) = 10H*DS+SP = 10H*2000H + FFFFH = 2FFFF.
  To push the data in AX to stack,

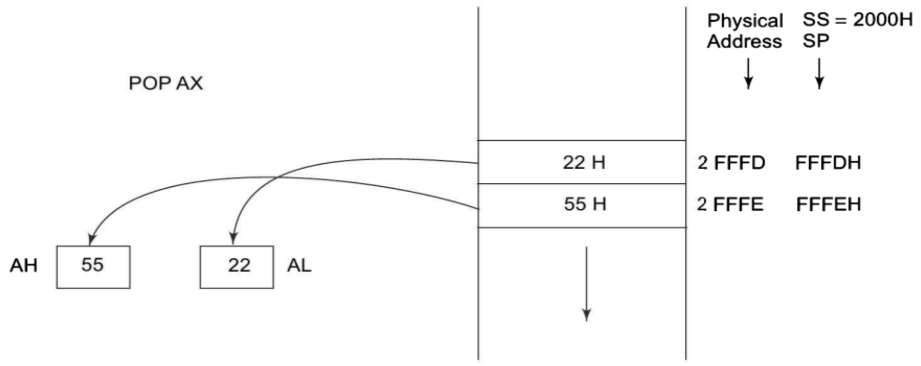
  1. SP is decremented by 1 such that EA = 2FFFE.
  2. Push the content in AH (upper byte) to 2FFFE.
  3. Then again SP is decremented by 1 such that EA = 2FFFD.
  4. Push the content in AL (lower byte) to 2FFFD.



Pushing Data to Stack Memory

- **POP: Pop from stack** The POP instruction copies a word from the stack location pointed to by the stack pointer to a destination specified in the instruction. The destination can be a general-purpose register, a segment register, or a memory location. The data in the stack is not changed. After the word is copied to the specified destination, the stack pointer is automatically incremented by 2 to point to the next word on the stack.

  Pushing register content from the stack:
  Eg: PUSH AX
  Assume now the effective address (EA) = 10H*DS+SP = 10H*2000H + FFFDH = 2FFFD.
  To push the data in AX to stack,
  1. Contents of stack top memory location 2FFFD are stored in AL.
  2. Then SP is incremented by 1 such that EA = 2FFFE.
  3. Now the contents of the memory location pointed by SP are stored in AH.
  4. Then SP is again incremented by 1 such that EA = 2FFFF.



Popping Register Contents from Stack Memory

- **When a number of registers have to be stored and retrieved in the stack, the order of retrieval should be reversed to that of the order of storage**. For example, let BX be pushed to the stack first and DX next. When the stored information has to be retrieved to appropriate registers then the top of the stack should be popped to DX first and then to BX next. The storage and retrieval in the stack are in reverse order because the SP is decremented for every write operation into the stack and SP is incremented for every read operation from the stack. Therefore the stack in an 8086 is called **Last-In-First-Out (LIFO) stack, i.e., the last stored information can be read first.**

Programming using Stack:

1. Write an ALP to change a sequence of nine 2-byte numbers from ascending to descending order. Use the LIFO property of the stack.

```
            MOV AX, 6000H
            MOV SS, AX                  ; Initialize stack segment
            MOV SP, 0500H               ; Initialize offset address of the stack with
                                        ; the starting address of input array
            MOV CL, 09H                 ; Initialize counter with the number of words
                                        ; to be sorted
            MOV BX, 0700H+09H           ; Initialize BX as the last address of the
                                        ; result array
NEXT:       POP AX                      ; Get the first word from the input array
            MOV DX, SP                  ; Save source stack pointer
            MOV SP, BX                  ; Get destination stack pointer
            PUSH AX                     ; Save AX to the result array
            MOV BX, SP                  ; Save destination stack pointer
            MOV SP, DX                  ; Get source stack pointer for the next
                                        ; number
            DEC CL                      ; Decrement count
            JNZ NEXT                    ; If count is not zero go to the next number
            HLT
```

INTERRUPTS IN 8086

The microprocessors allow normal program execution to be interrupted **in order to carry out a specific task/work**. The processor can be interrupted in the following ways (sources of interrupts)

1. By an external signal generated by a peripheral
2. By an internal signal generated by a special instruction in the program
3. By an internal signal generated due to an exceptional condition (eg: divide by zero)

Interrupt: *The process of interrupting the normal program execution to carry out a specific task/work.*

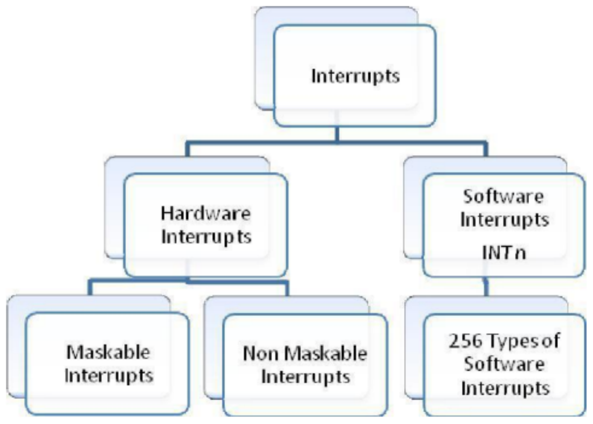
- When a microprocessor receives an interrupt signal it stops executing current normal program, save the status (or content) of various registers (IP, CS and flag registers in case of 8086) in stack and then the processor executes a subroutine/procedure in order to perform the specific task/work requested by the interrupt. The subroutine/procedure that is executed in response to an interrupt is also called **Interrupt Service Subroutine (ISR)**. At the end of ISR, the stored status of registers in stack is restored to respective registers, and the processor resumes the normal program execution from the point (instruction) where it was interrupted.

<u>Classification of Interrupts of 8086:</u>

In general, the interrupts can be classified in the following three ways :

1. Hardware and software interrupts
2. Vectored and Non Vectored interrupt
3. Maskable and Non Maskable interrupts

1. **<u>Hardware and Software Interrupts</u>**:
   - ➢ The interrupts initiated by external hardware by sending an appropriate signal to the interrupt pin of the processor are called hardware interrupts. The 8086 processor has two interrupt pins INTR and NMI.
   - ➢ The software interrupts are program instructions. These instructions are inserted at desired locations in a program. While running a program, if software interrupt instruction is encountered then the processor initiates an interrupt. The 8086 processor has 256 types of software interrupts. The software interrupt instruction is INT n, where n is the type number in the range 0 to 255.



2. **<u>Vectored and Non Vectored Interrupt:</u>**
   - ➢ When an interrupt signal is accepted by the processor, if the program control automatically branches to a specific address (called vector address) then the interrupt is called vectored interrupt. The automatic branching to vector address is predefined by the manufacturer of processors. (In these vector addresses the interrupt service subroutines (ISR) are stored).

- ➢ In non-vectored interrupts the interrupting device should supply the address of the ISR to be executed in response to the interrupt.
- ➢ **All the 8086 interrupts are vectored interrupts**. The vector address for an 8086 interrupt is obtained from a vector table implemented in the first 1kb memory space (00000h to 03FFFh).

3. **Maskable and Non-Maskable Interrupts:**
   - ➢ The interrupts whose request can be either accepted or rejected by the processor are called maskable interrupts.
   - ➢ The interrupts whose request has to be definitely accepted (or cannot be rejected) by the processor are called non-maskable interrupts. Whenever a request is made by a non-maskable interrupt, the processor has to definitely accept that request and service that interrupt by suspending its current program and executing an ISR.
   - ➢ In 8086 processor all the hardware interrupts initiated through INTR pin are maskable by clearing interrupt flag (IF). The interrupt initiated through NMI pin and all software interrupts are non-maskable.

Classification of Interrupts of 8086:

1. Predefined(or Dedicated) Interrupts
2. Software Interrupts of 8086
3. Hardware Interrupts of 8086

1. **Predefined(or Dedicated) Interrupts:**
   a. **Division by zero (Type-0 interrupt):** the result of a division operation is too large to fit in the destination register and this interrupt is initiated
   b. **Single step (Type-1 interrupt):** When the Trap/Trace Flag (TF) is set to one, this interrupt is initiated.
   c. **Nonmaskable interrupt, NMI (Type-2 interrupt):** a low-to-high transition on its NMI input pin will result in this interrupt.
   d. **Break Point interrupt (Type-3 interrupt):** this interrupt is used to implement a breakpoint function, which executes a program partly or up to the desired point and then returns the control to the user.
   e. **Interrupt on overflow (Type-4 interrupt):** the Overflow Flag (OF) will be set if the signed arithmetic operation generates a result whose size is larger than the size of the destination register/memory and this interrupt happens

The predefined interrupts are only defined by INTEL and INTEL has not provided any subroutine/procedure to be executed for these interrupts. To use the predefined interrupts the user/ system designer has to write Interrupt Service Routine (ISR) for each interrupt and store them in memory.

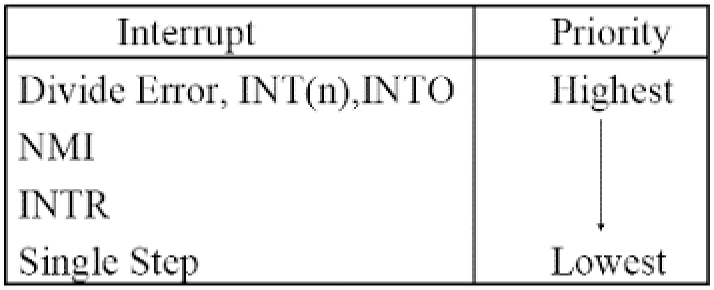
2. **Software Interrupts:**

   - ➢ The software interrupts are program instructions. These instructions are inserted at desired locations in a program. While running a program, if software interrupt instruction is encountered then the processor initiates an interrupt. The 8086 processor has 256 types of software interrupts. The software interrupt instruction is INT n, where n is the type number in the range 0 to 255.
   - ➢ The software interrupts are nonmaskable and have higher priority than hardware interrupts.

3. **Hardware Interrupts:**

   - ➢ The interrupts initiated by external hardware by sending an appropriate signal to the interrupt pin of the processor are called hardware interrupts. The 8086 processor has two interrupt pins INTR and NMI.
   - ➢ The interrupting device can send a type number in the range of 0 to 25510. Therefore, all the 256 types of interrupts can be initiated through INTR pin.
   - ➢ The hardware interrupts initiated through INTR are maskable by clearing the Interrupt Flag (IF).
   - ➢ The hardware interrupt NMI is nonmaskable and has higher priority than interrupts initiated through INTR.

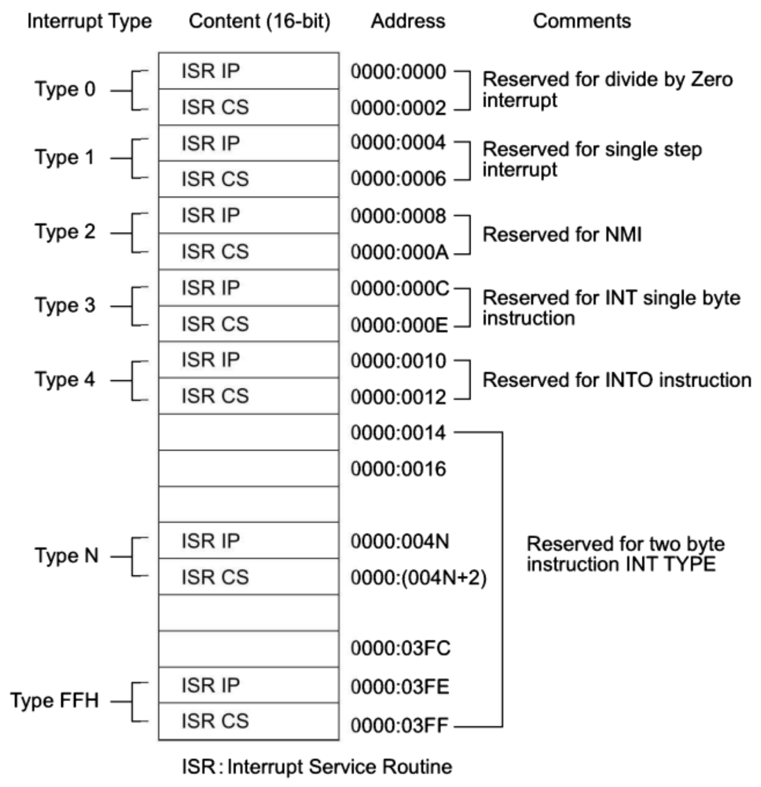
Priorities of Interrupts of 8086:



Interrupt	Priority
Divide Error, INT(n),INTO	Highest
NMI	↓
INTR	↓
Single Step	Lowest

Interrupt Vector Table (IVT):

- ➢ The interrupt Vector table contains the IP (Instruction Pointer) and CS (Code Segment base address) of all the interrupt types stored sequentially from the address **0000:0000 to 0000:03FFH.**
- ➢ **The interrupt type number N is multiplied by 4 and this hexadecimal multiplication gives the offset address** of in the code segment with base address 0000H (IVT) at which the IP and CS of the ISR are stored.



Structure of Interrupt Vector Table of 8086/88

Servicing An Interrupt By 8086

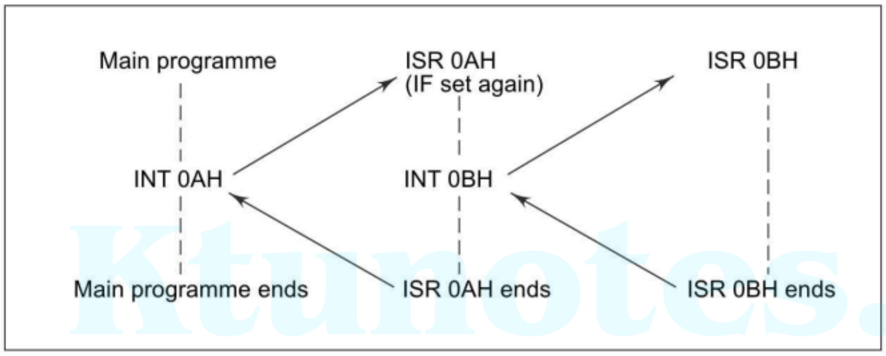
The 8086 processor checks for interrupt requests at the end of each instruction cycle. If an interrupt request is deducted, then the 8086 processor responds to the interrupt by performing the following operations:

1. The SP is decremented by two and the content of flag register is pushed to stack memory.
2. The interrupt system is disabled by clearing Interrupt Flag (IF).
3. The single-step trap flag is disabled by clearing Trap Flag (TF).
4. The stack pointer is decremented by two and the content of CS-register is pushed to stack memory.
5. Again, the stack pointer is decremented by two and the content of IP is pushed to stack memory.
6. In case of hardware interrupt through INTR, the processor runs an interrupt acknowledge cycle to get the interrupt type number.
7. For software interrupts, the type number is specified in the instruction itself.
8. For NMI and exceptions, the type number is defined by INTEL.
9. The processor generates a 20-bit memory address by multiplying the type number N by 4. This memory address is the address of the interrupt vector table, where the vector address of the Interrupt Service Routine (ISR) is stored by the user/system designer.
10. The first word pointed by vector table address is loaded in IP and the next word is loaded in CS-register. Now the content of the IP is the offset address and the content of the CS-register is the segment base address of the ISR to be executed
11. The 20-bit physical memory address of ISR is calculated by CS*10H + IP

12. The processor executes the ISR to service the interrupt.
13. The ISR will be terminated by the IRET instruction. When this instruction is executed, the top of stack is popped to IP, CS and flag register one word by one word. After every pop operation, the SP is incremented by two.
14. Thus, at the end of ISR, the previous status of the processor is restored and so the processor will resume the execution of the normal program from the instruction where it was suspended.

Nested interrupt:

Nested interrupt handling is **where the software is prepared to accept another interrupt, even before it finishes handling the current interrupt**. This enables you to prioritize interrupts and make significant improvements to the latency of high priority events at the cost of additional complexity.

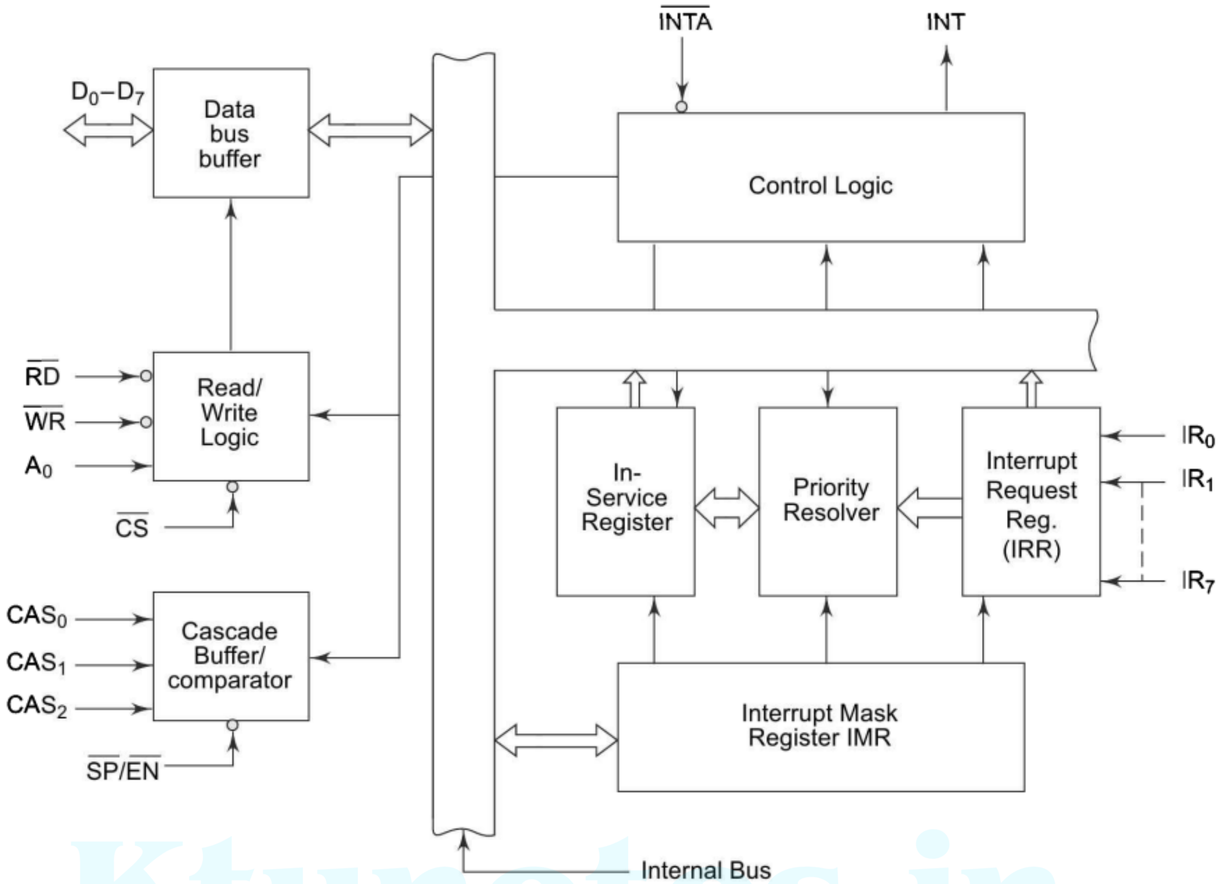


Transfer of Control for Nested Interrupts

PROGRAMMABLE INTERRUPT CONTROLLER - 8259

- Programmable interrupt controller 8259A which is able to handle a number of interrupts at a time. This controller takes care of a number of simultaneously appearing interrupt requests along with their types and priorities. This will **reduce the processor's burden of handling interrupts**. The 8259 A interrupt controller can
    1. Handle **eight interrupt inputs**. This is equivalent to providing eight interrupt pins on the processor in place of one INTR/INT pin.
    2. Resolve **eight levels of interrupt priorities**
    3. **Mask** each interrupt request individually.
    4. Read the status of pending interrupts, in-service interrupts, and masked interrupts.
    5. Be set up to accept either the level-triggered or edge-triggered interrupt request.
    6. Nine 8259 can be **cascaded** in a master-slave configuration to handle 64 interrupt inputs.

ARCHITECTURE OF 8259



8259A Block Diagram

Data Bus Buffer This tristate bidirectional buffer interfaces internal 8259A bus to the microprocessor system data bus. Control words, status and vector information pass through buffer during read or write operations.

Read /Write Control Logic This circuit accepts and decodes commands from the CPU. This also allows the status of the 8259A to be transferred onto the data bus.

Interrupt Request Register (IRR): The interrupts at IRQ input lines are handled by Interrupt Request Register internally. IRR stores all the interrupt requests in it in order to serve them one by one on a priority basis.

Priority Resolver This unit determines the priorities of the interrupt requests appearing simultaneously. The highest priority is selected and stored in the corresponding bit of ISR during the INTA pulse. The IR0 has the highest priority while the IR7 has the lowest one.

In-Service Register (ISR) This stores all the interrupt requests those are being served, i.e ISR keeps a track of the requests being served.

Interrupt Mask Register (IMR) This register stores the bits required to mask the interrupts.

Control Logic This block manages the interrupt and interrupt acknowledge signals to be sent to the CPU for serving one of the eight interrupt requests. This also accepts interrupt acknowledge (INTA) signal from the CPU (8086) that causes the 8259A to release vector address onto the data bus.

Cascade Buffer/Comparator This block stores and compares the ID's of all the 8259As used in the system in a cascade configuration. The three I/O pins CAS0 - CAS2 act as output when the 8259A is used as a master. The same pins act as input when 8259 is in slave mode.

The Interrupt sequence in an 8086-8259A system is described as follows:

1. One or more IR lines are raised high that set corresponding IRR bits.
2. 8259A resolves priority and sends an INT signal to CPU (8086)
3. The CPU acknowledges with an INTA pulse.
4. Upon receiving an INTA signal from the CPU, the highest priority ISR bit is set and the corresponding IRR bit is reset. The 8259A does not drive data during this period.
5. The 8086 will initiate a second INTA pulse. During this period 8259A releases an 8-bit pointer (pre-programmed ISR address) onto a data bus from where it is read by the processor.
6. This completes the interrupt cycle. The ISR bit is reset at the end of the second INTA pulse if the automatic end of interrupt (AEOI) mode is programmed. Otherwise, the ISR bit remains set until an appropriate EOI command is issued at the end of the interrupt subroutine.

PIN DESCRIPTION

CS: This is an active low chip select signal for enabling RD and WR operations of 8259A.

WR: This pin is an active low write enable input to 8259A. This enables it to accept command words from CPU.

RD: This is an active low read enable input to 8259A. A low on this line enables 8259A to release status onto the data bus of CPU.

D7-D0: These pins form a bidirectional data bus that carries 8-bit data either to control word or from status word registers. This also carries interrupt vector information.

CAS0-CAS2 Cascade Lines: A single 8259A provides eight vectored interrupts. If more interrupts are required, the 8259A is used in cascade mode.

PS/EN: This pin is a dual purpose pin. When the chip is used in buffered mode, it can be used as buffer enable to control buffer transceivers. If this is not used in buffered mode then the pin is used as input.

INT This pin goes high whenever a valid interrupt request is asserted. This is used to interrupt the CPU and is connected to the interrupt input of CPU.

IR0-IR7(1nterrupt Requests): These pins act as inputs to accept interrupt requests to the CPU.

INTA (Interrupt Acknowledge): This pin is an input used to strobe-in (acknowledge) 8259A by 8086 to get interrupt vector data onto the data bus.

INTERFACING MEMORY WITH 8086

- Most memory ICs are byte-oriented i.e., each memory location can store only one byte of data.
- The 8086 is a 16-bit microprocessor, it can transfer 16-bit data. So in addition to byte, word (16-bit) has to be stored in the memory.
- To implement this, the entire memory is divided into two memory banks: Bank0 and Bank1.
- Bank0 is selected only when A0 is zero and Bank1 is selected only when BHE' is zero.
- A0 is zero for all even addresses, so Bank0 is usually referred to as even-addressed memory bank.
- BHE' is used to access higher order memory bank, referred to as odd-addressed memory bank.
- Every microprocessor-based system has a memory system. Almost all systems contain two basic types of memory, read-only memory (ROM) and random access memory (RAM) or read/write memory.
- ROM contains system software and permanent system data such as lookup tables, IVT..etc.
- RAM contains temporary data and application software.
- ROMs/PROMs/EPROMs are mapped to cover the CPU's reset address since these are non-volatile.
- **When the 8086 is reset, the next instruction is fetched from the memory location FFFF0H. So in the 8086 system, the location FFFF0H must be in the ROM location.**

Interfacing RAM, ROM, EPROM to 8086:

The general procedure of static memory interfacing with 8086

1. Arrange the available memory chips so as to obtain 16-bit data bus width (even and odd bank). The upper 8-bit bank is called the 'odd address memory bank'. The lower 8-bit bank is called the 'even address memory bank'.
2. Calculate the number of address lines required to address the available memory size. Connect those address lines of the processor with the address lines of memory chips and also connect the RD and WR inputs to the corresponding processor control signals.
3. Connect the 16-bit data bus of the memory bank with that of the microprocessor 8086 (D0-D7 to even memory and D8-D15 to odd memory).
4. The remaining address lines of the microprocessor, BHE' and A0 are used for decoding the required chip select (CS) signals for the odd and even memory banks. The CS of memory is derived from the output of the decoding circuit.

Problem 1:

Interface two 4K x 8 EPROM and two 4K x 8 RAM chips with 8086. Select suitable maps.

Solution:

Memory size: 2 x 4 = 8 K ROM and 2 x 4 = 8 K RAM

For addressing N = 8K memory, the number of address lines required is n

$2^n = N = 8\text{ K} = 8 \times 1024 = 2^3 \times 2^{10} = 2^{13}$

n = 13

Therefore address lines A0 - A12 are connected to the address lines of RAM/ROM.

We know that, after reset, the IP and CS are initialized to form address FFFF0H. Hence, this address must lie in the EPROM. The address of RAM may be selected anywhere in the 1MB address space of 8086, but we will select the RAM address such that the address map of the system is continuous.

$8\text{ K} = (8192)_{10} = (2000)_H$

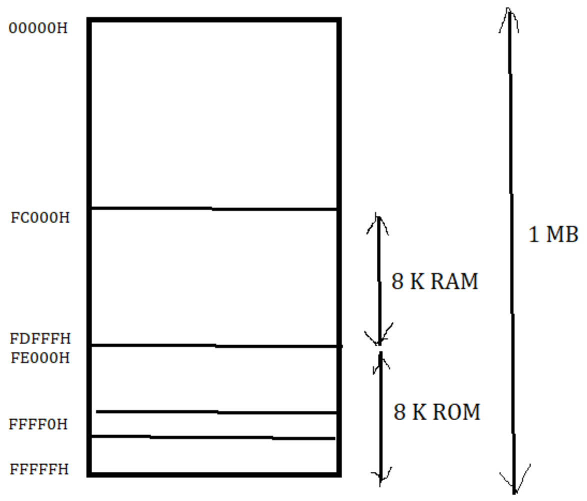
2000H - 0001H = 1FFFH

The ending address of ROM is chosen as FFFFFH.

Therefore the starting address of ROM is FFFFFH - 1FFFH = FE000H

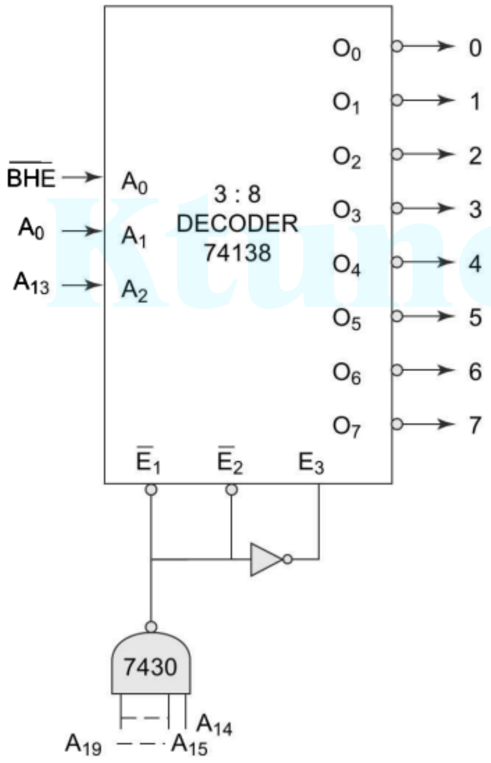
The ending address of RAM is FE000H - 00001H = FDFFFH

The starting address of RAM is FDFFFH - 1FFFH = FC000H



Address	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_{09}	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}
FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
							EPROM						8K × 8							
FE000H	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
FDFFFH	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
							RAM					8K × 8								
FC000H	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- ➢ Address lines A13 - A19 are used for decoding to generate the chip select. In this problem a decoder IC is used.
- ➢ A13 is used to select RAM or ROM; A13 = 0 for selecting RAM; A13 = 1 for selecting ROM.
- ➢ A0 and BHE' are used to select even and odd memory.
- ➢ A14 - A19 are all high and can be used to enable chip select decoder IC



A_2 *A_{13}*	*A_1* *A_0*	*A_0* *$\overline{BHE}$*	*Selection/* *Comment*
0	0	0	Even and odd addresses in RAM
0	0	1	Only even address in RAM
0	1	0	Only odd address in RAM
1	0	0	Even and odd addresses in ROM
1	0	1	Only even address in ROM
1	1	0	Only odd address in ROM

Output of decoder:

0 = 0 if Even and odd address in RAM

1 = 0 if Only even address in RAM

2 = 0 if Only odd address in RAM

3 = 0 if NOP

4 = 0 if Even and odd address in ROM

5 = 0 if Only even address in ROM

6 = 0 if Only odd address in ROM

7 = 0 if NOP

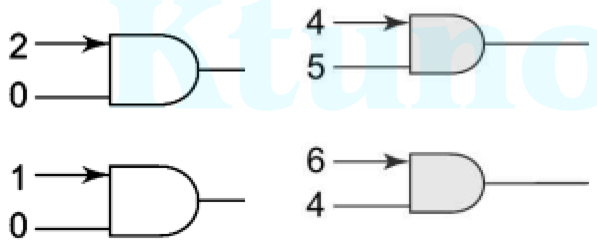
Therefore the chip select logic for RAM even address is LOW only when decoder outputs 0 or 1 is LOW

The chip select logic for RAM odd address is LOW only when decoder outputs 0 or 2 is LOW

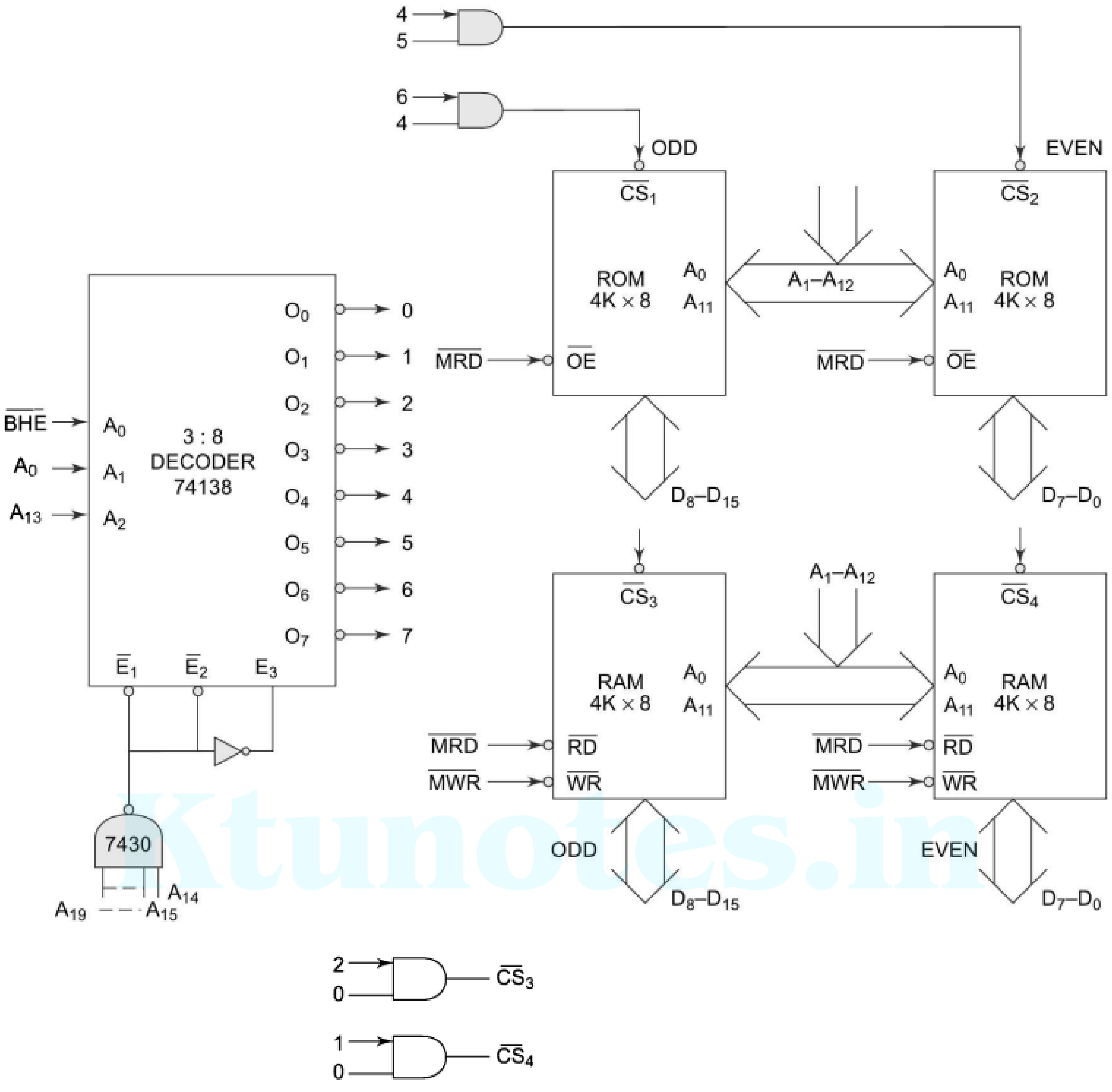
The chip select logic for ROM even address is LOW only when decoder outputs 4 or 5 is LOW

The chip select logic for ROM odd address is LOW only when decoder outputs 4 or 6 is LOW

Implemented as



(PTO)



4
5
6
4
ODD
EVEN
$\overline{CS}_1$
$\overline{CS}_2$
ROM 4K × 8
A_0
A_{11}
A_1–A_{12}
$\overline{MRD}$
$\overline{OE}$
D_8–D_{15}
D_7–D_0
3 : 8 DECODER 74138
$\overline{BHE}$
A_0
A_{13}
A_1
A_2
O_0 0
O_1 1
O_2 2
O_3 3
O_4 4
O_5 5
O_6 6
O_7 7
$\overline{E}_1$
$\overline{E}_2$
E_3
7430
A_{14}
A_{19}
A_{15}
$\overline{CS}_3$
$\overline{CS}_4$
RAM 4K × 8
$\overline{RD}$
$\overline{WR}$
$\overline{MWR}$
2
0
1
0