

MODULE- 3 (STACK AND INTERRUPTS)

Stack structure of 8086, programming using stack- Interrupts - Types of Interrupts and Interrupt Service Routine- Handling Interrupts in 8086- Interrupt programming - Programmable Interrupt Controller - 8259, Architecture (Just mention the control word, no need to memorize the control word)- Interfacing Memory with 8086.

Interrupts

Definition: The meaning of 'interrupts' is to break the sequence of operation. While the CPU is executing a program, on 'interrupt' breaks the normal sequence of execution of instructions, diverts its execution to some other program called Interrupt Service Routine (ISR). After executing ISR, the control is transferred back again to the main program. Interrupt processing is an alternative to polling.



Whenever a number of devices interrupt a CPU at a time, and if the processor is able to handle them properly, it is said to have multiple interrupt processing capability.

- There are two interrupt pins in 8086. NMI and INTR

Need for Interrupt: Interrupts are particularly useful when interfacing I/O devices that provide or require data at relatively low data transfer rate.

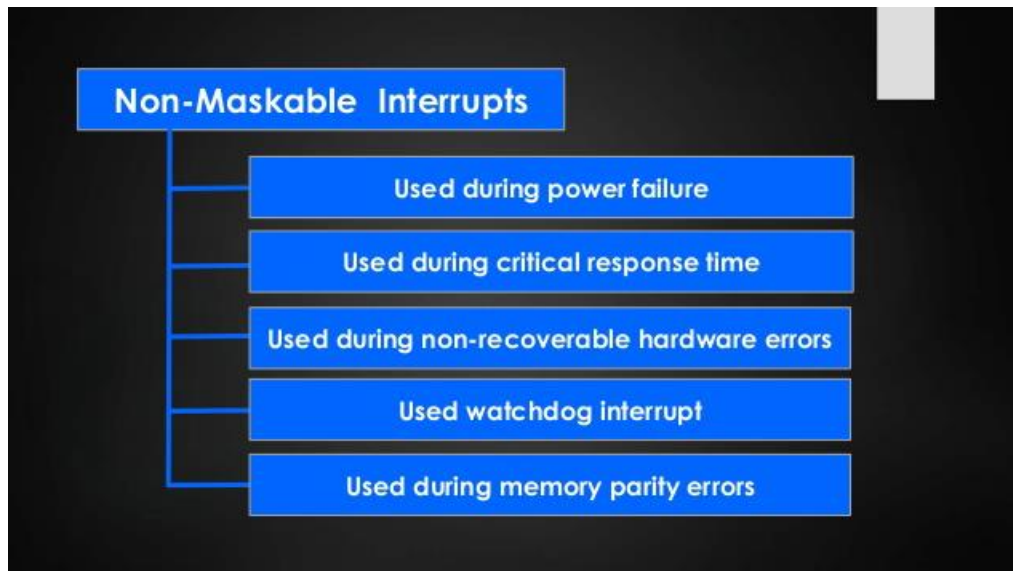
NMI

It is a single non-maskable interrupt pin (NMI) having higher priority. When this interrupt is activated,

these actions take place –

- Completes the current instruction that is in progress.
- Pushes the Flag register values on to the stack.

- Pushes the CS (code segment) value and IP (instruction pointer) value of the return address on to the stack.
- IP is loaded from the contents of the word location 00008H. (Type 2*4=00008 H)
- CS is loaded from the contents of the next word location 0000AH.
- Interrupt flag and trap flag are reset to 0.



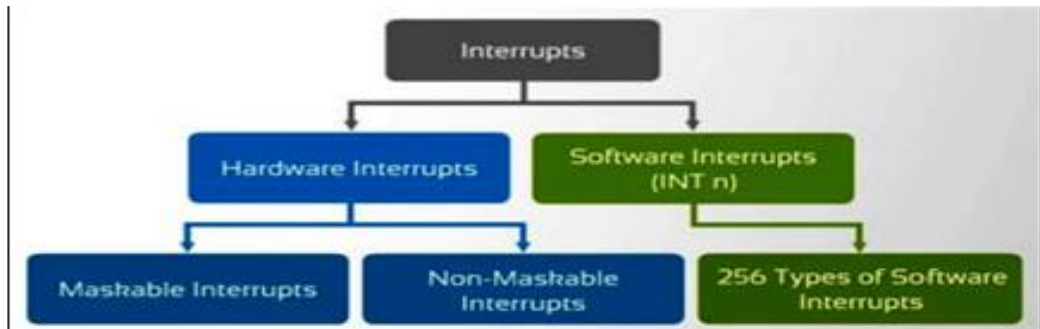
INTR

- The INTR is a maskable interrupt pin. It can be accepted (enable) or rejected (masked).
- The microprocessor enabled the interrupt using set interrupt flag instruction. It should disable using clear interrupt Flag instruction.
- These actions are taken by the microprocessor –
- First completes the current instruction.
- Activates INTA output and receives the interrupt type, say X.
- Flag register value, CS value of the return address and IP value of the return address are pushed on to the stack.
- IP value is loaded from the contents of word location $X \times 4$
- CS is loaded from the contents of the next word location.
- Interrupt flag and trap flag is reset to 0

Types of Interrupts: There are two types of Interrupts in 8086.

Internal (or) Software Interrupts are generated by a software instruction and operate similarly to a jump or branch instruction.

External (or) Hardware Interrupts are caused by an external hardware module.



HARDWARE INTERRUPTS

Hardware interrupts are generated by hardware devices when something unusual happens; this could be a

key-press or a mouse move or any other action.

It can be divided into two

1. Maskable
2. Non maskable

Maskable Interrupts:

There are some interrupts which can be masked (disabled) or enabled by the processor.

Non-Maskable Interrupts:

There are some interrupts which cannot be masked out or ignored by the processor. These are associated

with high priority tasks which cannot be ignored (like memory parity or bus faults).

SOFTWARE INTERRUPTS

Interrupts are generated by a software instruction and operate similarly to a jump or branch instruction.

- 256 interrupts are there

INT n is invoked as software interrupts- n is the type no in the range 0 to 255(00 to FF)

Interrupts are divided into three groups

Type 0 to Type4 (Dedicated Interrupts)

- TYPE 0 interrupt represents division by zero situation.
- TYPE 1 interrupt represents single-step execution during the debugging of a program.

- TYPE 2 interrupt represents non-maskable NMI interrupt.
- TYPE 3 interrupt represents break-point interrupt.
- TYPE 4 interrupt represents overflow interrupt.

Type 5 to 31(Not used by 8086, reserved for higher processor like 80286,80386....

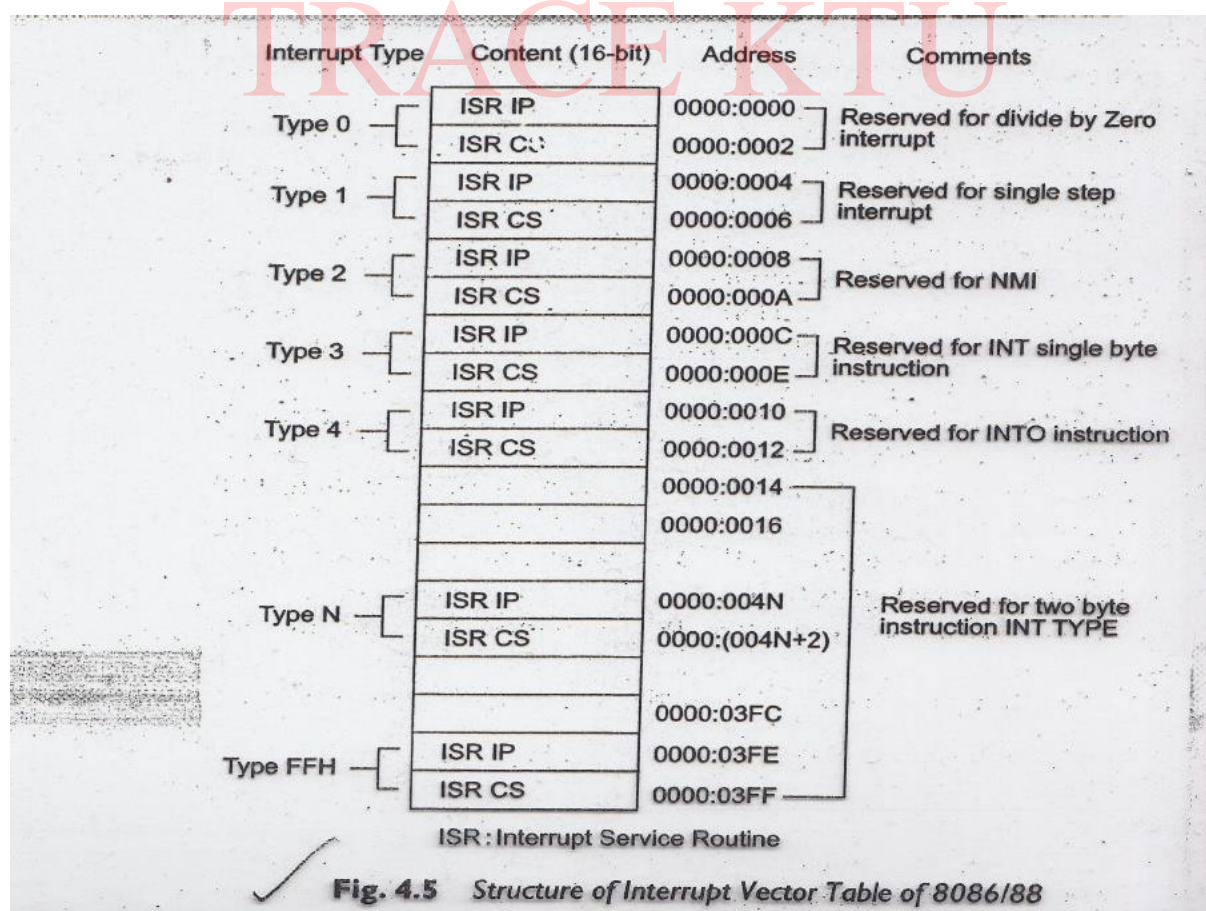
Type 32-255(Available for user)

- User defined interrupts

Interrupt Service Routine

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.

When an interrupt is occurred, the microprocessor stops execution of current instruction. It transfers the content of program counter (CS and IP) into stack. After this, it jumps to the memory location specified by Interrupt Vector Table (IVT). After that the code written on that memory area will execute.



The first 1Kbyte of memory of 8086 (00000 to 003FF) is set aside as a table for storing the starting addresses of Interrupt Service Procedures (ISP). Since 4-bytes are required for storing starting addresses of ISPs, the table can hold 256 Interrupt procedures. The starting address of an ISP is often called the Interrupt Vector or Interrupt Pointer. Therefore, the table is referred as Interrupt Vector Table.

NMI (Non mask-able interrupt)

- o This is a non-mask-able, edge triggered, high priority interrupt.
- o On receiving an interrupt on NMI line, the microprocessor executes INT
- o Microprocessor obtains the ISR address from location $2 \times 4 = 00008H$ from the IVT.
- o It reads 4 locations starting from this address to get the values for IP and CS to execute the ISR.

INTR

This is a mask-able, level triggered, low priority interrupt.

- o On receiving an interrupt on INTR line, the microprocessor executes 2 INTA pulses.
- o 1st INTA pulse – The interrupting device calculates (prepares to send) the vector number.
- 2nd INTA pulse – The interrupting device sends the vector number 'N' to the microprocessor.
- o Now microprocessor multiplies $N \times 4$ and goes to the corresponding location in the IVT to obtain the ISR address. INTR is a mask-able interrupt.
- o It is masked by making $IF = 0$ by software through CLI instruction.
- o It is unmasked by making $IF = 1$ by software through STI instruction.

The Operation of an Interrupt sequence on the 8086 Microprocessor:

1. External interface sends an interrupt signal, to the Interrupt Request (INTR) pin, or an internal interrupt occurs.
2. The CPU finishes the present instruction (for a hardware interrupt) and sends Interrupt Acknowledge (INTA) to hardware interface.
3. The interrupt type N is sent to the Central Processor Unit (CPU) via the Data bus from the hardware interface.
4. The contents of the flag registers are pushed onto the stack.
5. Both the interrupt (IF) and (TF) flags are cleared. This disables the INTR pin and the trap or single-step feature.

6. The contents of the code segment register (CS) are pushed onto the Stack.
7. The contents of the instruction pointer (IP) are pushed onto the Stack.
8. The interrupt vector contents are fetched, from $(4 \times N)$ and then placed into the IP and from $(4 \times N + 2)$ into the CS so that the next instruction executes at the interrupt service procedure addressed by the interrupt vector.
9. While returning from the interrupt-service routine by the Interrupt Return (IRET) instruction, the IP, CS and Flag registers are popped from the Stack and return to their state prior to the interrupt.

Interrupt programming - Programmable Interrupt Controller - 8259, Architecture

Necessity of 8259A

In a system, microprocessor may need to perform the following tasks in an efficient way using interrupt:

- ☐ Read ASCII characters from a keyboard on an interrupt basis.
- ☐ Count interrupts from a timer to produce a real time clock of seconds, minutes and hours.
- ☐ Communicate with an A/D converter.
- ☐ Communicate with a display or printer.
- ☐ Detect several emergency signals like power failure etc on an interrupt basis.

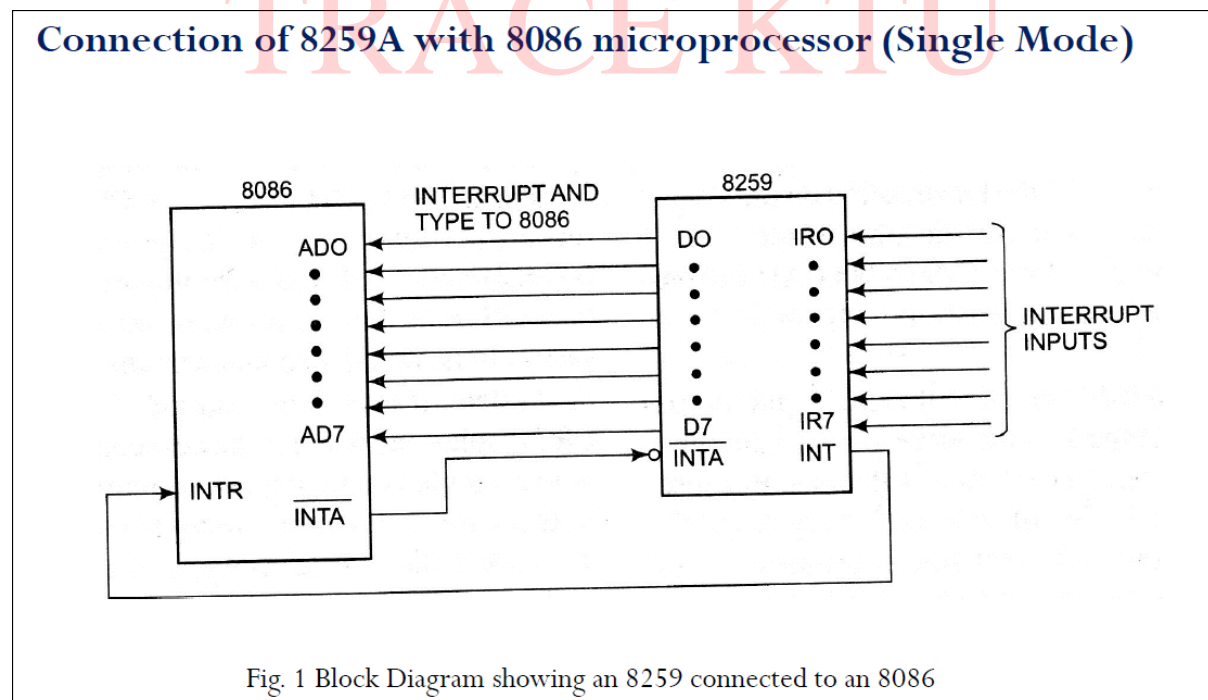
Each of these interrupt applications requires a separate interrupt pin. But, the 8086 has only two interrupt inputs: **NMI** and **INTR**. If we use NMI for a power failure interrupt, this leaves only one interrupt input for all other applications. The solution is to use an external device called a priority interrupt controller (PIC) such as Intel 8259A.

Internal 8259 Block Diagram includes eight blocks: data bus buffer, read/write logic, control logic, three registers (IRR, ISR and IMR), priority resolver, and cascade buffer.

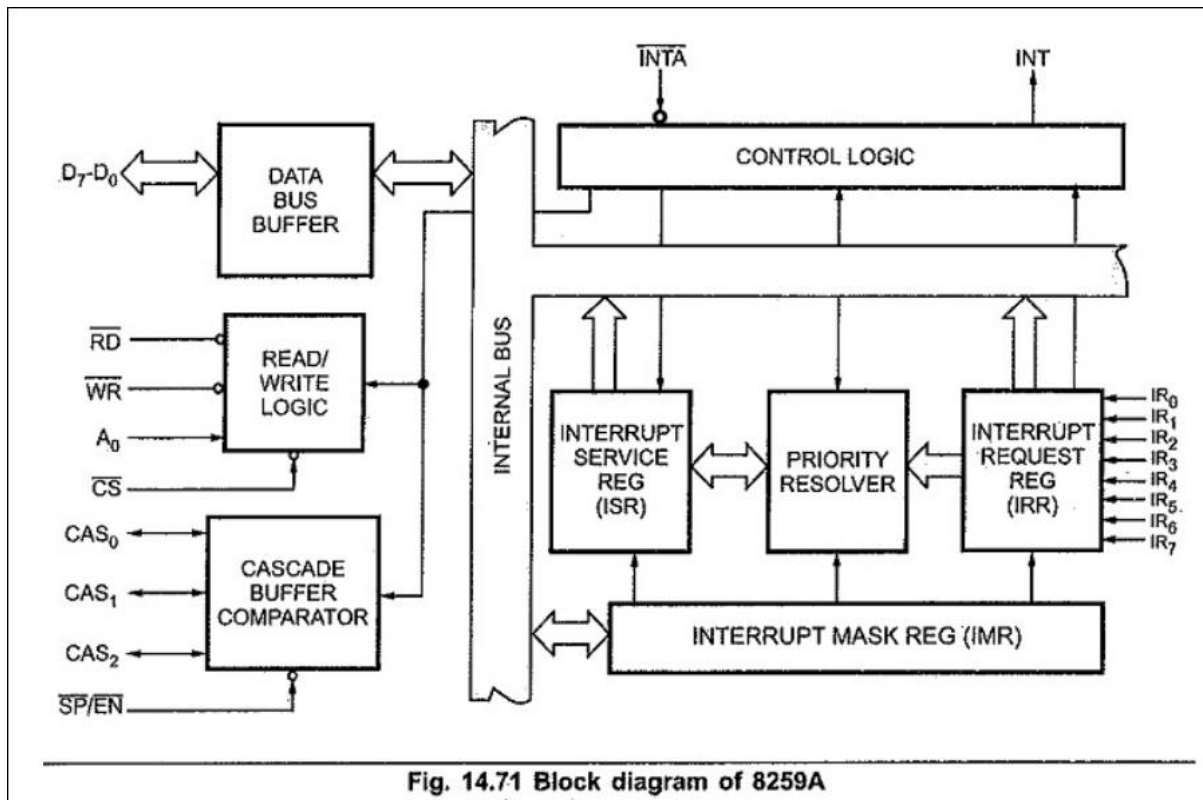
Function of 8259A

The Programmable Interrupt Controller (PIC) functions as an overall manager in an interrupt-driven system environment.

- It accepts requests from the peripheral equipment, determines which of the incoming requests is of the highest importance (priority), ascertains whether the incoming request has a higher priority value than the level currently being serviced and issues an interrupt to the CPU based on this determination.
- Each peripheral device or structure usually has a special program or, routine that is associated with its specific functional or operational requirements; that is referred to as a service routine or service procedure.
- The 8259A PIC, after issuing an interrupt to the CPU, must somehow input information (interrupt vector number) into the CPU that can point the program counter to the service procedure associated with the requesting device.



The 8259A PIC adds eight vectored priority encoded interrupts to the microprocessor. This controller can be expanded without additional hardware, to accept up to 64 interrupt requests. This requires a master 8259A and eight 8259A slaves.



Internal Block Diagram of 8259A

Data Bus Buffer:

The data bus buffer allows the 8085 to send control words to the 8259A and read a status word from the 8259 Block Diagram. The 8-bit data bus buffer also allows the 8259A to send interrupt opcode and address of the interrupt service subroutine to the 8085.

8-bit data bus:

The 8-bit data bus ($D7-D0$) allows the 8086-

- * To send control words to the 8259A and read a status word from the 8259A.
- * To send interrupt types to the 8086.

The eight data lines are always connected to the lower half of the 8086 data bus because the 8086 expects to receive interrupt types on lower 8-bit data lines. ***RD, WR and CS:***

The **RD** and **WR** inputs control data transfer when the device is selected by asserting its chip select (**CS**) input low. Usually **RD** and **WR** pins are connected to the system **RD** and **WR** lines. **CS** may be connected to address decoder's output.

Address pin A0: A0 input of 8259A is used to select one of the two internal addresses in the device. This pin may be connected to any of the system address lines.

Cascade Buffer Comparator:

This section generates control signals necessary for cascade operations. It also generates Buffer-Enable signals. As stated earlier, the 8259 Block Diagram can be cascaded with other 8259s in order to expand the interrupt handling capacity to sixty-four levels. In such a case, the former is called a **master**, and the latter are called **slaves**. The 8259 can be set up as a master or a slave by the SP/EN pin.

CAS₀— CAS₂

For a master 8259, the CAS₀-CAS₂ pins are output pins, and for slave 8259, these are input pins. When the 8259 is a master (that is, when it accepts interrupt requests from other 8259s), the CALL opcode is generated by the Master in response to the first INTA. The vector address must be released by the slave 8259. The master sends an identification code of three-bits to select one out of the eight possible slave 8259s on the CAS₀-CAS₂ lines. The slave 8259s accept these three signals as inputs (on their CAS₀ – CAS₂ pins) and compare the code sent by the master with the codes assigned to them during initialization. The slave thus selected (which had originally placed an interrupt request to the master 8259) then puts the address of the interrupt service routine during the second and third INTA pulses from the CPU.

The cascade lines (CAS₂-CAS₀) are used as outputs from the master to the slaves for cascading multiple 8159As in a system. The master outputs a 3-bit slave identification number on these lines. Each slave in a system is assigned a 3-bit ID as part of its initialization. Sending this 3-bit ID number enables the slave.

Slave program/Enable buffer (SP/EN):

This pin is a dual function pin, when the 8259A is in buffered mode, this is an output that controls the data bus transceivers in a large microprocessor-based system. When the 8259A is not in the buffered mode, this pin programs the device as a master (1) or a slave (0). When we use only one 8159A in our system, the **SP/EN** pin is tied high (1).

The SP/EN signal is tied high for the master. However, it is grounded for the slave.

In large systems where buffers are used to drive the data bus, the data sent by the 8259 in response to INTA cannot be accessed by the CPU (due to the data bus buffer being disabled). If an 8259 is used in the buffered mode (buffered or non-buffered modes of operation can be specified at the time of initializing the 8259), the SP/EN pin is used as an output which can be used to enable the system data bus buffer whenever the data bus outputs of 8259 are enabled (i.e. when it is ready to send data).

Thus, in non-buffered mode, the SP/EN pin of an 8259 is used to specify whether the 8259 is to operate as a master or as a slave, and in the buffered mode, the SP/EN pin is used as an output to enable the data bus buffer of the system.

INT Pin:

The interrupt output (INT) pin of 8259A is connected to the INTR pin on the microprocessor (8086) when there is only one 8259A in the system. In a system with master and slaves, only master's INT pin is connected to 8086. The slave's INT pins are connected to different IR pins of the master.

Interrupt acknowledge (INTA) pin:

This input pin of 8259A is connected to the **INTA** output of the 8086.

Interrupt request inputs(IR7–IR0):

The eight interrupt request inputs (IR7-IR0) are used to request an interrupt by the external devices in case of single 8159A system. In case of multiple 8259A system these input pins (IR7-IR0) of the master is connected to output INT pins of the slaves. Unused IR inputs should be tied to ground so that a noise pulse cannot accidentally cause an interrupt. An interrupt signal must remain high on an IR input until after the falling edge of the first **INTA** pulse.

The interrupt Mask Register (IMR):

This register is used to disable (mask) or enable (unmask) individual interrupt inputs. Each bit in this register corresponds to the interrupt input with the same number. We can unmask an interrupt input by sending a command word with a 0 in the bit position that corresponds to that input.

The Interrupt Request Register (IRR):

The IRR keeps track of which interrupt inputs are asking for service. If an interrupt input has an interrupt signal on it, then the corresponding bit in the IRR will be set.

The In-Service Register (ISR):

The ISR keeps track of which interrupt inputs are currently being serviced. For each input that is currently being serviced, the corresponding bit will be set in the ISR.

The Priority Resolver:

The Priority Resolver acts as a “judge” that determines if and when an interrupt request on one of the IR inputs gets serviced.

Read/Write Logic:

The RD and WR inputs control the data flow on the data bus when the device is selected by asserting its chip select (CS) input low.

Control Logic:

This block has an input and an output line. If the 8259A is properly enabled, the interrupt request will cause the 8259A to assert its INT output pin high. If this pin is connected to the INTR pin of an 8085 and if the 8085 Interrupt Enable (IE) flag is set, then this high signal will cause the 8085 to respond INTR as explained earlier.

8259A Interrupt Operation

To implement interrupt, **the interrupt enable flip-flop in the microprocessor should be enabled by writing the EI instruction** and the **8259A should be initialized by writing control words** in the control register. The 8259A requires two types of control words:

- (a) Initialization Command Words (ICWS)
- (b) Operational Command Words (OCWs)

ICWs are used to set up the proper conditions and specify RST vector address.

The OCWs are used to perform functions such as masking interrupts, setting up status-read operations etc.

Step-1: The IRR of 8259A stores the request.

Step-2: The priority resolver checks 3 registers-

* The IRR for interrupt requests. * IMR for masking bits and * The ISR for interrupt request being served. It resolves the priority and sets the INT high when appropriate.

Step-3: The MPU acknowledges the interrupt by sending signals in *INTA*.

Step-4: After the *INTA* is received, the appropriate priority bit in the ISR is set to indicate which interrupt level is being served and the corresponding bit in the IRR is reset to indicate that the request for the CALL instruction is placed on the data bus.

Step-5: When MPU decodes the CALL instruction, it places two more *INTA* signals on the data bus.

Step-6: When the 8259A receives the second *INTA*, it places the low-order byte of the CALL address on the data bus. At the 3rd *INTA*, it places the high order byte on the data bus. The CALL address is the vector memory location for the interrupt, this address is placed in the control register during the initialization.

Step-7: During the 3rd *INTA* pulse, the ISR bit is reset either automatically (Automatic-End-of-Interrupt-AEOI) or by a command word that must be issued at the end of the service routine (End of Interrupt-EOI). This option is determined by the initialization command word (ICW).

Step-8: The program sequence is transferred to the memory location specified by the CALL instruction.

INTERFACING MEMORY WITH 8086

Semiconductor memories are of two types, viz. RAM (Random Access Memory) and ROM (Read Only Memory). The semiconductor RAMs are of broadly two types-static RAM and dynamic RAM. The semiconductor memories are organized as two-dimensional arrays of memory locations. For example, 4Kx8 or 4K byte memory contains 4096 locations, where each location contains 8-bit data and only one of the 4096 locations can be selected at a time.

The general procedure of static memory interfacing with 8086 is briefly described as follows:

1. Arrange the available memory chips so as to obtain 16-bit data bus width. The upper 8-bit bank is called "odd address memory bank" and the lower 8-bit bank is called "even address memory bank".
2. Connect available memory address lines of memory chips with those of the microprocessor and also connect the memory and inputs to the corresponding processor control signals. Connect 16-bit data bus of the memory bank with that of the microprocessor 8086.
3. The remaining address lines of the microprocessor, and A0 are used for decoding the required chip select signals for the odd and even memory banks. The of memory is derived from the O/P of the decoding circuit.

Relation between number of address pins and memory capacity

Number of address pins	Memory capacity			Range of address in hexa
	in decimal	in kilo	in hexa	
10	$2^{10} = 1024$	1k	400	000 to 3FF
11	$2^{11} = 2 \times 2^{10} = 2048$	2k	800	000 to 7FF
12	$2^{12} = 2^2 \times 2^{10} = 4 \times 2^{10} = 4096$	4k	1000	000 to FFF
13	$2^{13} = 2^3 \times 2^{10} = 8 \times 2^{10} = 8192$	8k	2000	0000 to 1FFF
14	$2^{14} = 2^4 \times 2^{10} = 16 \times 2^{10} = 16384$	16k	4000	0000 to 3FFF
15	$2^{15} = 2^5 \times 2^{10} = 32 \times 2^{10} = 32768$	32k	8000	0000 to 7FFF
16	$2^{16} = 2^6 \times 2^{10} = 64 \times 2^{10} = 65536$	64k	10000	0000 to FFFF
17	$2^{17} = 2^7 \times 2^{10} = 128 \times 2^{10} = 131072$	128k	20000	00000 to 1FFFF
18	$2^{18} = 2^8 \times 2^{10} = 256 \times 2^{10} = 262144$	256k	40000	00000 to 3FFFF
19	$2^{19} = 2^9 \times 2^{10} = 512 \times 2^{10} = 524288$	512k	80000	00000 to 7FFFF
20	$2^{20} = 2^{10} \times 2^{10} = 1024 \times 2^{10} = 1048576$	1024 k=1M	100000	00000 to FFFFF

Problem 1

Interface two 4Kx8 EPROM and two 4Kx8 RAM chips with 8086. Select suitable maps.

Memory Map Table

Address	A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
EPROM								8K X 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 X 8												
FC000H	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Total 8K bytes of EPROM need 13 address lines A0-A12 (since $2^{13} = 8K$).

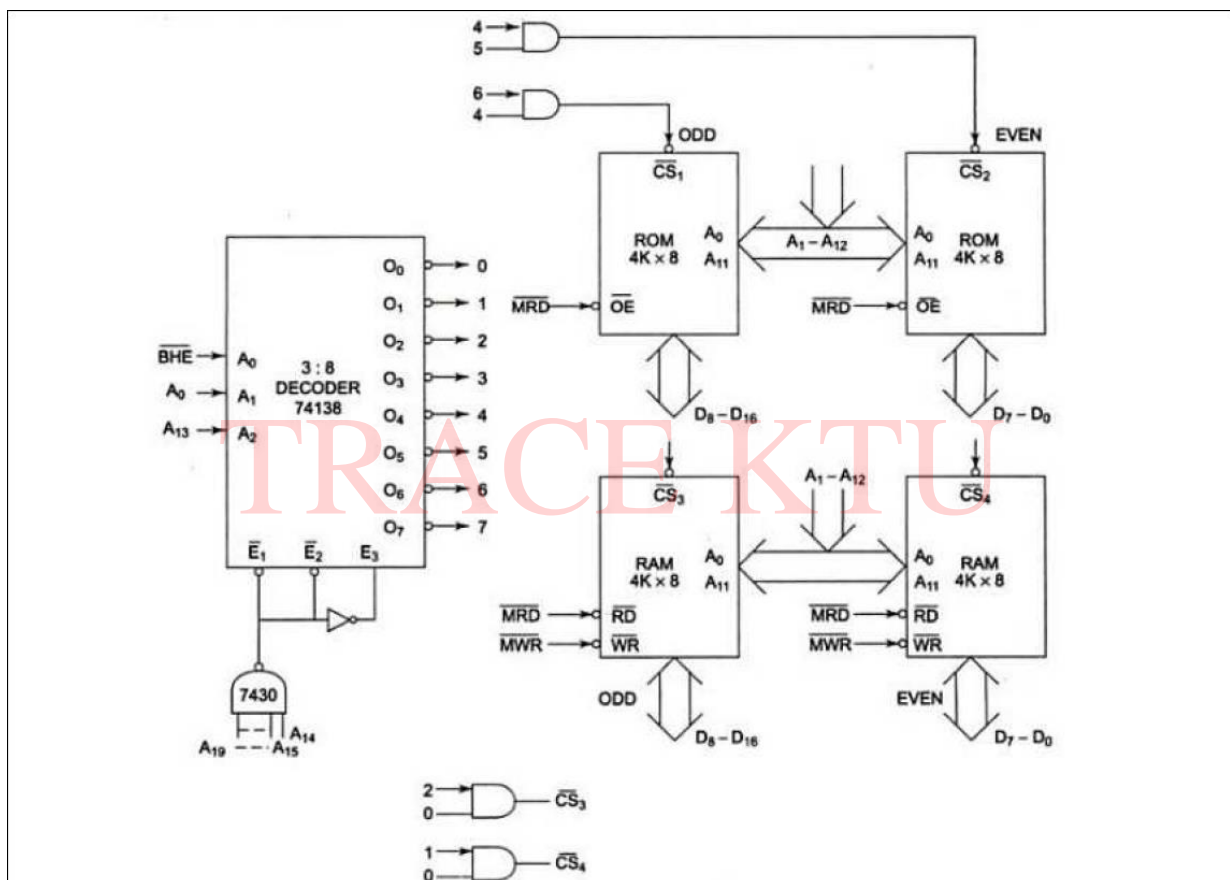
Address lines A13 - A19 are used for decoding to generate the chip select.

The signal goes low when a transfer is at odd address or higher byte of data is to be accessed.

Let us assume that the latched address, and demultiplexed data lines are readily available for interfacing.

The memory system in this problem contains in total four 4K x 8 memory chips.

The two 4K x 8 chips of RAM and ROM are arranged in parallel to obtain 16-bit data bus width. If A₀ is 0, i.e., the address is even and is in RAM, then the lower RAM chip is selected indicating 8-bit transfer at an even address. If A₀ is 1, i.e., the address is odd and is in RAM, the goes low, the upper RAM chip is selected, further indicating that the 8-bit transfer is at an odd address. If the selected addresses are in ROM, the respective ROM chips are selected. If at a time A₀ and both are 0, both the RAM or ROM chips are selected, i.e., the data transfer is of 16 bits. The selection of chips here takes place as shown in table below.



Memory Chip Selection Table:

Decoder I/P --> Address/ \overline{BHE} -->	A ₂ A ₁₃	A ₁ A ₀	A ₀ \overline{BHE}	Selection/ Comment
Word transfer on D ₀ - D ₁₅	0	0	0	Even and odd address in RAM
Byte transfer on D ₇ - D ₀	0	0	1	Only even address in RAM
Byte transfer on D ₈ - D ₁₅	0	1	0	Only odd address in RAM
Word transfer on D ₀ - D ₁₅	1	0	0	Even and odd address in RAM
Byte transfer on D ₇ - D ₀	1	0	1	Only even address in RAM
Byte transfer on D ₈ - D ₁₅	1	1	0	Only odd address in ROM

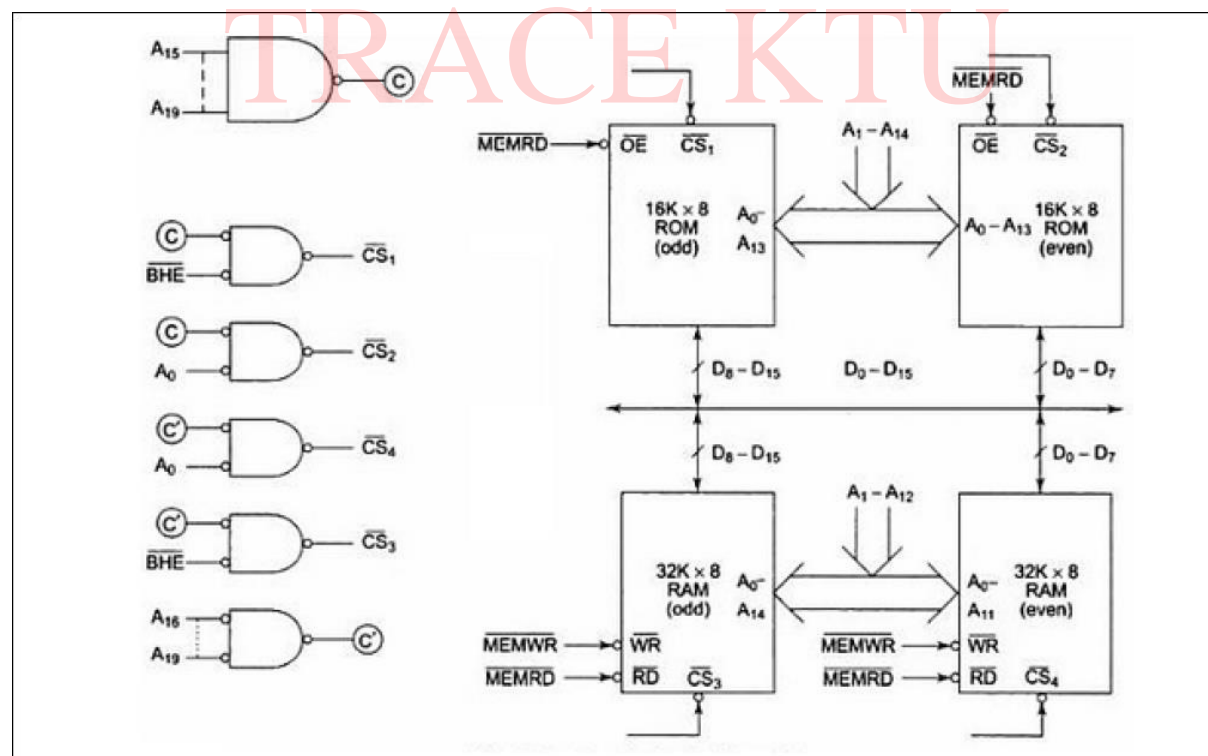
Problem 2:

Design an interface between 8086 CPU and two chips of 16K×8 EPROM and two chips of 32K×8 RAM. Select the starting address of EPROM suitably. The RAM address must start at 00000 H.

Solution: The last address in the map of 8086 is FFFFF H. after resetting, the processor starts from FFFF0 H. hence this address must lie in the address range of EPROM.

Address Map for Problem																				
Addresses	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
32KB EPROM																				
F8000H	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0FFFFH	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
64KB RAM																				
00000H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

It is better not to use a decoder to implement the above map because it is not continuous, i.e. there is some unused address space between the last RAM address (0FFFF H) and the first EPROM address (F8000 H). Hence the logic is implemented using logic gates.



Problem3: It is required to interface two chips of $32K \times 8$ ROM and four chips of $32K \times 8$ RAM with 8086, according to following map.

ROM 1 and ROM 2 F0000H - FFFFFH, **RAM 1 and RAM 2** D0000H - DFFFFH, **RAM 3 and RAM 4** E0000H - EFFFFH. Show the implementation of this memory system.

Solution:

Address Map for Problem																				
Address	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
F0000H	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ROM 1 and 2																				
FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
D0000H	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RAM 1 and 2																				
DFFFFH	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
E0000H	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RAM 3 and 4																				
EFFFFH	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

