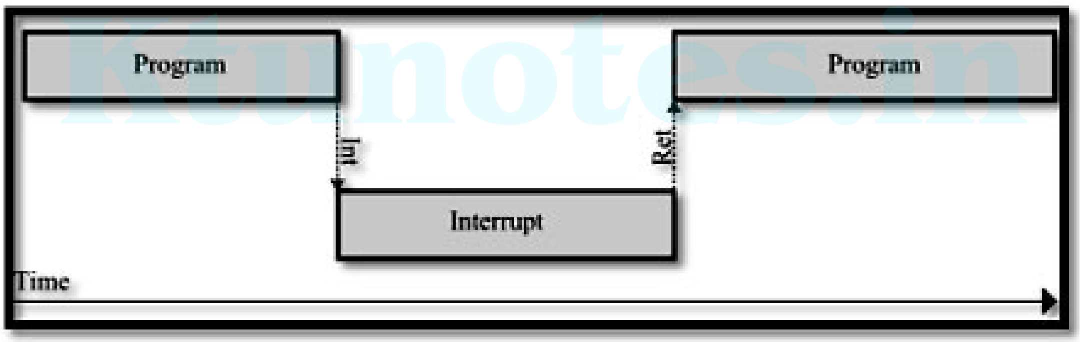
KTU NOTES

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

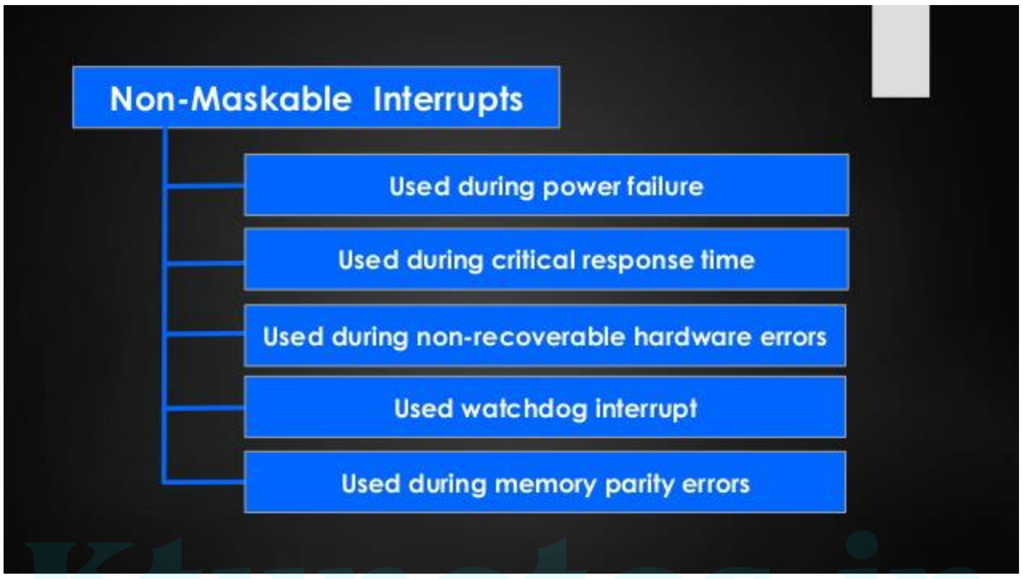
Need for Interrupt: Interrupts are particularly useful when interfacing I/O devices that provide or require data at relatively low data transfer rate.

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.


Non-Maskable Interrupts
Used during power failure
Used during critical response time
Used during non-recoverable hardware errors
Used watchdog interrupt
Used during memory parity errors

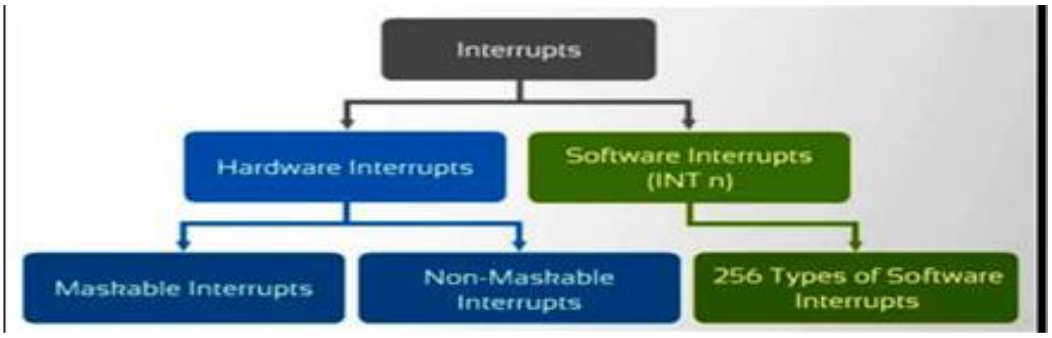
INTR

- 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 × 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.

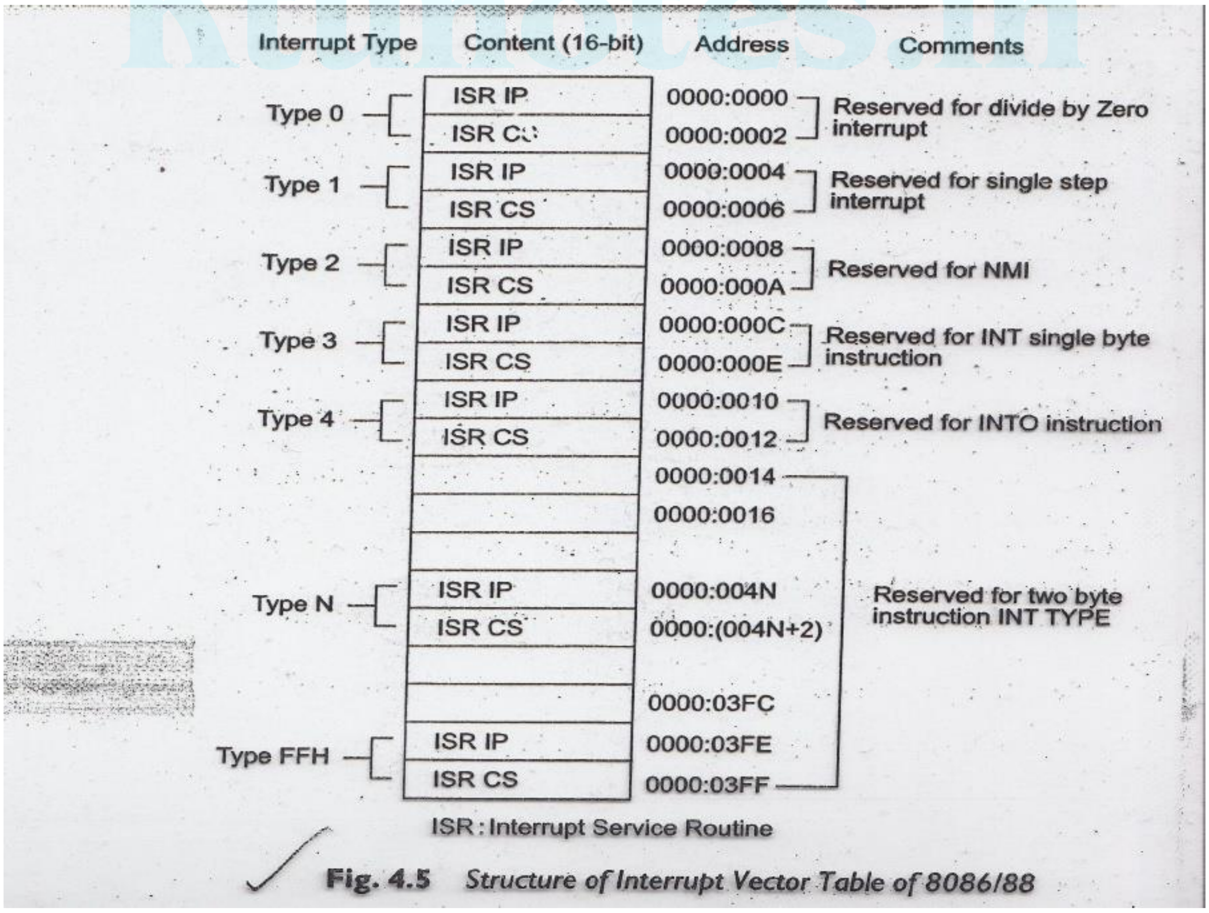


Fig. 4.5 *Structure of Interrupt Vector Table of 8086/88*

The first 1Kbyte of memory of 8086 (00000 to003FF) 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 x 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 x 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 x N) and then placed into the IP and from (4 x 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.

Connection of 8259A with 8086 microprocessor (Single Mode)

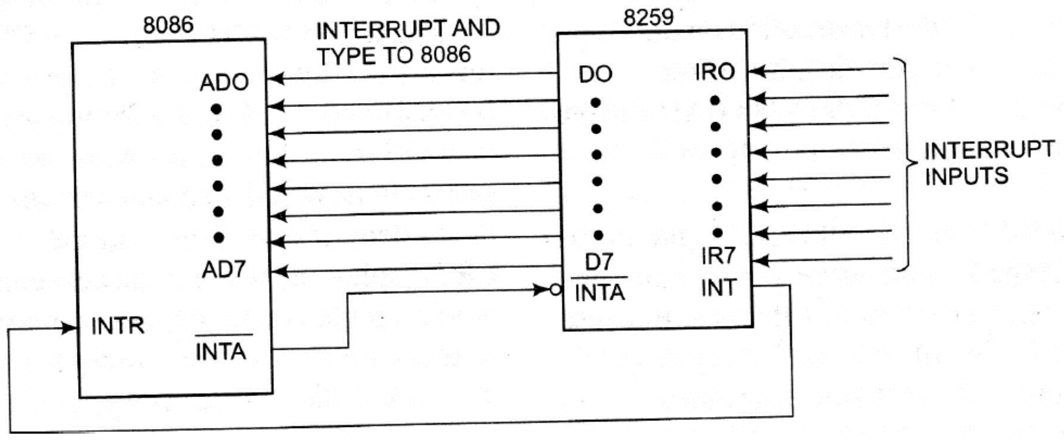


Fig. 1 Block Diagram showing an 8259 connected to an 8086

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 require a master 8259A and eight 8259A slaves.

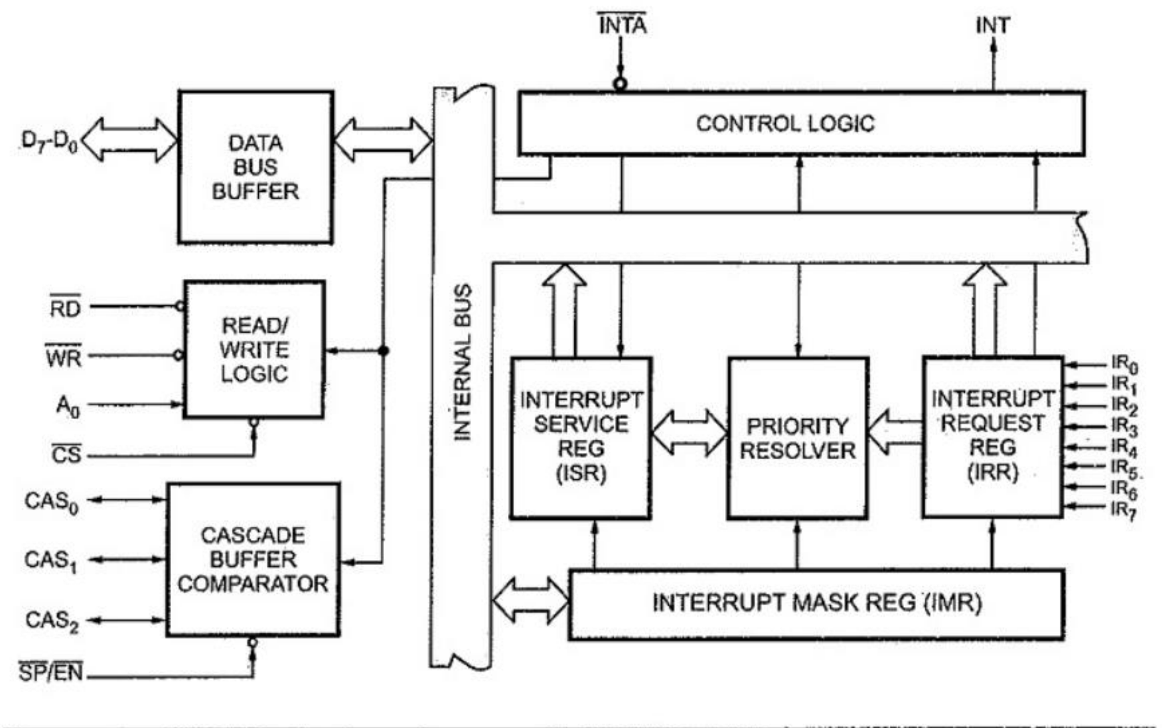


Fig. 14.71 Block diagram of 8259A

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. $\boldsymbol{RD}$,$\boldsymbol{WR}$ ***and*** $\boldsymbol{CS}$**:**

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

Address pin $\boldsymbol{A0}$: $A0$ 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.

CAS_0— CAS_2

For a master 8259, the CAS_0-CAS_2 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 CAS_0-CAS_2 lines. The slave 8259s accept these three signals as inputs (on their CAS_0 – CAS_2 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 ($\boldsymbol{SP}/\boldsymbol{EN}$):

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 $\boldsymbol{SP}/\boldsymbol{EN}$ 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 ($\boldsymbol{INTA}$) pin:

This input pin of 8259A is connected to the $\boldsymbol{INTA}$ output of the 8086.

Interrupt request inputs($\mathbf{IR7-IR0}$):

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 $\boldsymbol{INTA}$ 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

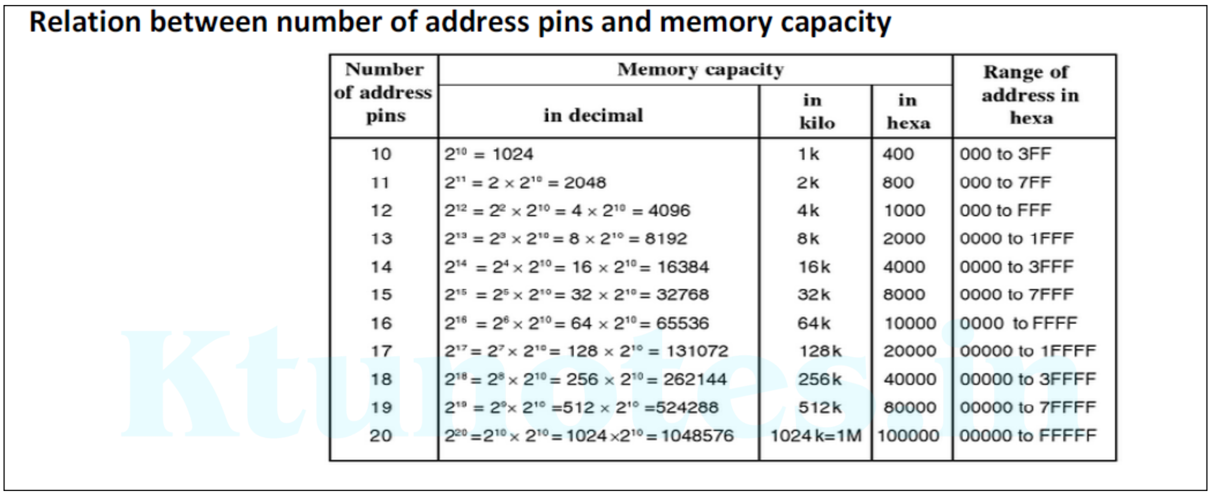
Semiconductor memories are of two types, viz. RAM (Random Access Memory) and ROM (Read Only Memory). The semiconductor RAMs are of broadly two types-static RAM and dynamic RAM. The semiconductor memories are organized as two-dimensional arrays of memory locations. For example, 4Kx8 or 4K byte memory 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.



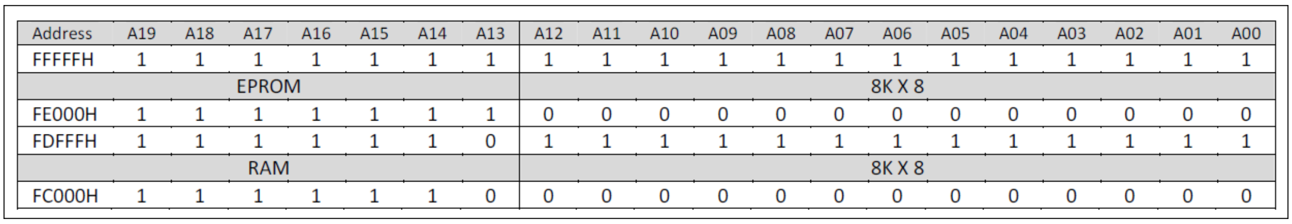
Relation between number of address pins and memory capacity

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	$2^{20} = 2^{10} \times 2^{10} = 1024 \times 2^{10} = 1048576$	1024k=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 z13 = 8K).

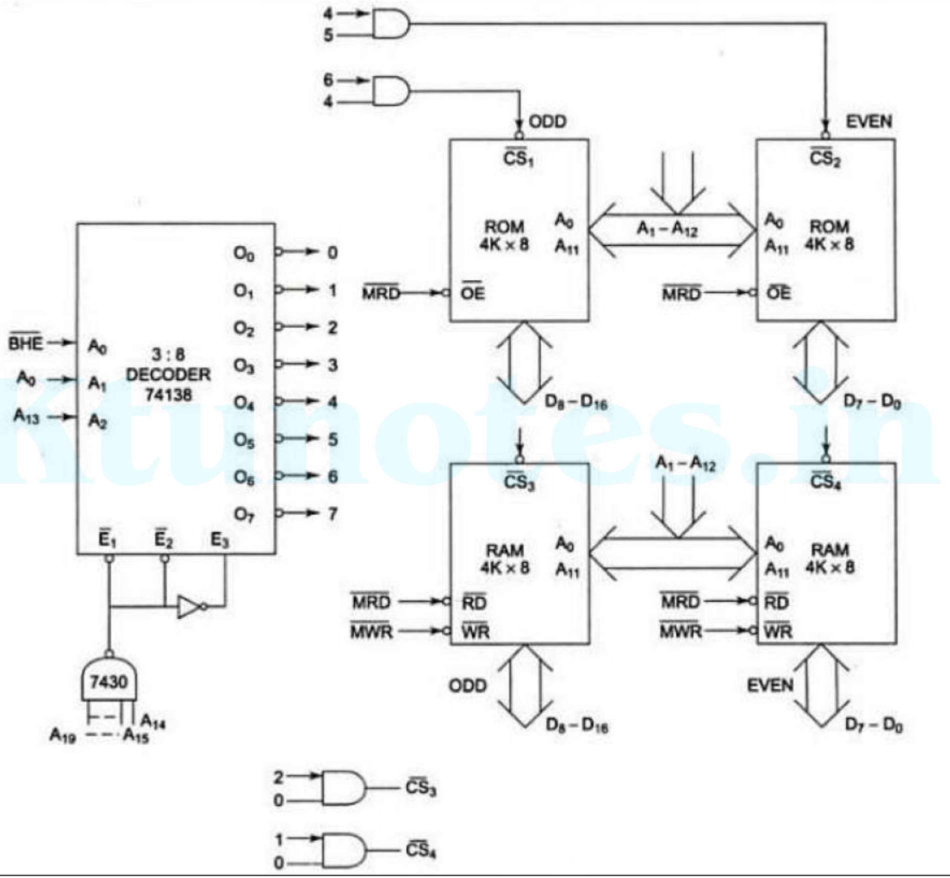
Address lines A13 - A19 are used for decoding to generate the chip select.

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 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.



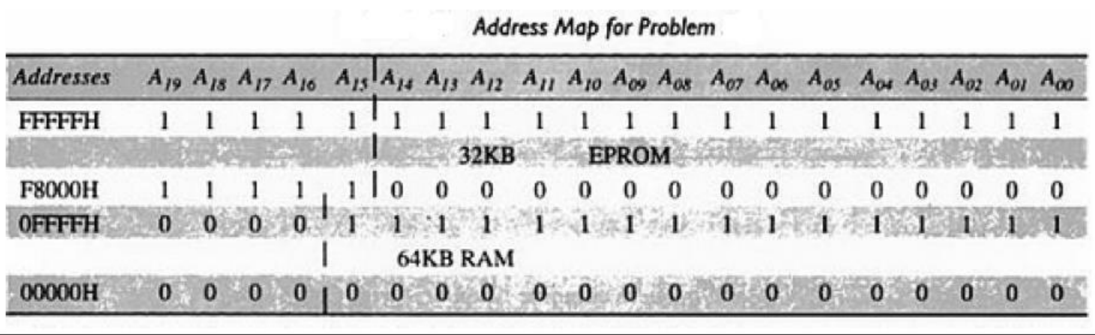
Memory Chip Selection Table:

Decoder I/P --> Address/$\overline{BHE}$ -->	A2 A13	A1 A0	A0 $\overline{BHE}$	Selection/ Comment
Word transfer on D0 - D15	0	0	0	Even and odd address in RAM
Byte transfer on D7 - D0	0	0	1	Only even address in RAM
Byte transfer on D8 - D15	0	1	0	Only odd address in RAM
Word transfer on D0 - D15	1	0	0	Even and odd address in RAM
Byte transfer on D7 - D0	1	0	1	Only even address in RAM
Byte transfer on D8 - D15	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.

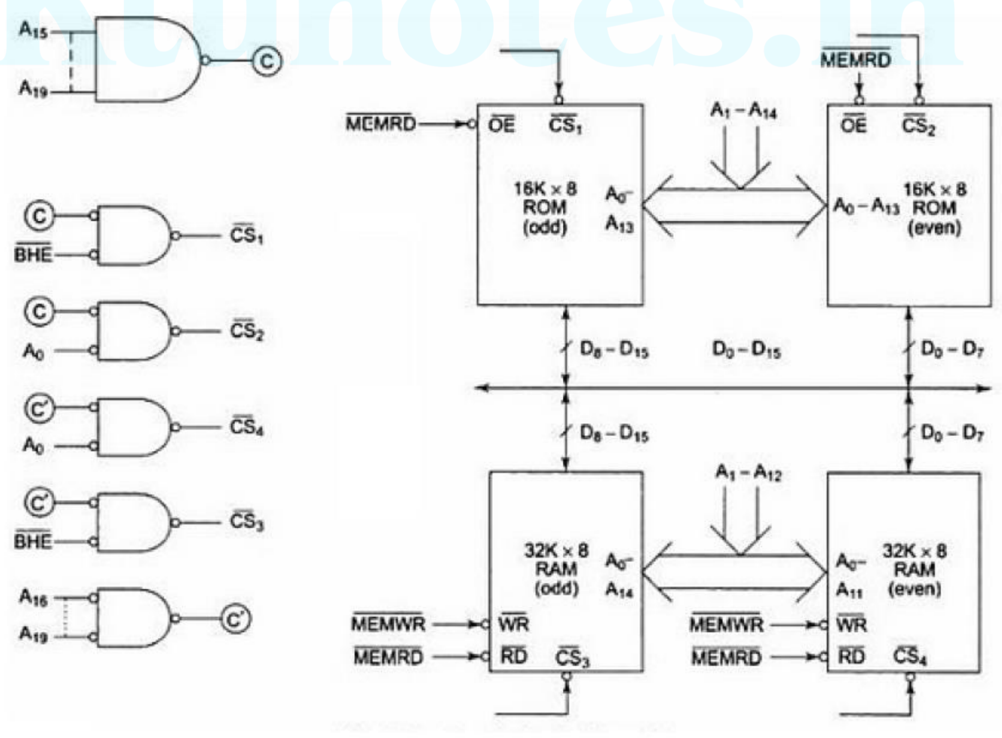
Solution: The last address in the map of 8086 is FFFFF H. after resetting, the processor starts from FFFF0 H. hence this address must lie in the address range of EPROM.



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

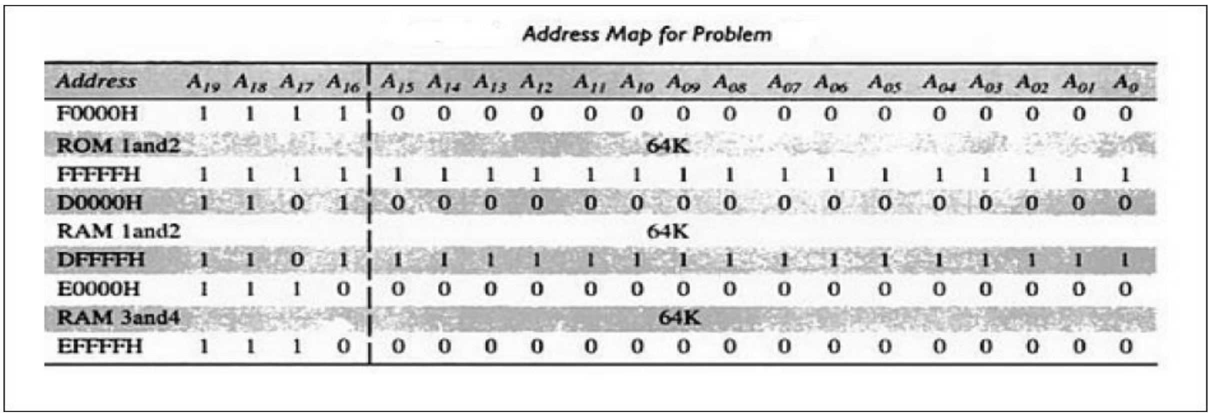
It is better not to use a decoder to implement the above map because it is not continuous, i.e. there is some unused address space between the last RAM address (0FFFF H) and the first EPROM address (F8000 H). Hence the logic is implemented using logic gates.



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

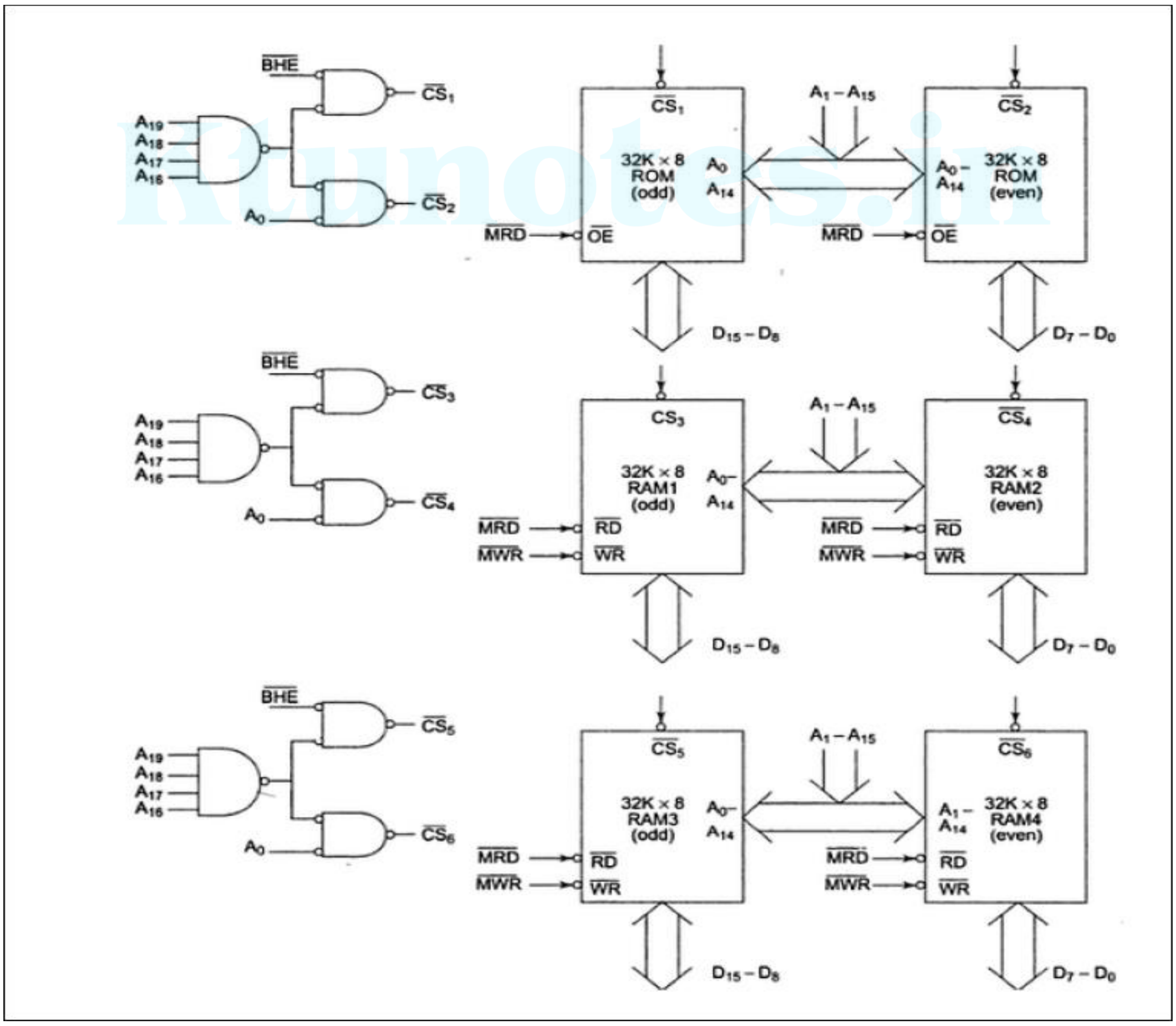
ROM 1 and ROM 2 F0000H - FFFFFH, RAM 1 and RAM 2 D0000H - DFFFFH, RAM 3 and RAM 4 E0000H - EFFFFH. Show the implementation of this memory system.

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_{09}	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{0}
F0000H	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ROM 1and2												64K								
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 1and2												64K								
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 3and4												64K								
EFFFFH	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



MODULE 3 STACKS AND INTERRUPTS OF 8086

STACK STRUCTURE OF 8086

Stack is a block of memory that may be used for temporarily for storing contents of register inside CPU. The stack is a block of memory that is accessed using SP and SS registers. The stack works in LIFO (Last In First Out) manner.

Stack contains a set of sequentially arranged data bytes, with last item appearing on top of it. This item is popped off from the stack when used by CPU.

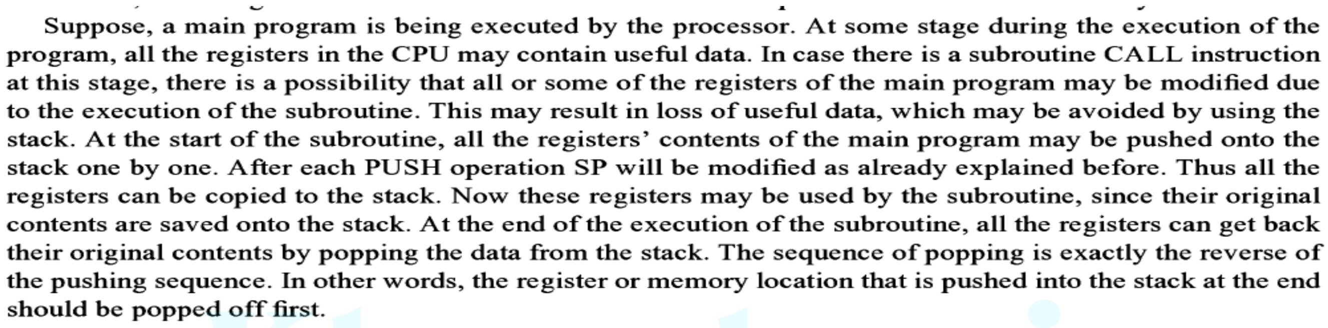
The stack pointer is a16 bit register that contains the offset of the address that lies in the stack segment.

The stack segment has maximum 64 Kbytes locations, and thus may overlap with other segments.

The stack segment register contains the base address of the stack in the memory.

The stack segment register and stack pointer register together address the stack top.

Each push operation decrements the stack pointer, while each pop operation increments the stack pointer.



Suppose, a main program is being executed by the processor. At some stage during the execution of the program, all the registers in the CPU may contain useful data. In case there is a subroutine CALL instruction at this stage, there is a possibility that all or some of the registers of the main program may be modified due to the execution of the subroutine. This may result in loss of useful data, which may be avoided by using the stack. At the start of the subroutine, all the registers' contents of the main program may be pushed onto the stack one by one. After each PUSH operation SP will be modified as already explained before. Thus all the registers can be copied to the stack. Now these registers may be used by the subroutine, since their original contents are saved onto the stack. At the end of the execution of the subroutine, all the registers can get back their original contents by popping the data from the stack. The sequence of popping is exactly the reverse of the pushing sequence. In other words, the register or memory location that is pushed into the stack at the end should be popped off first.

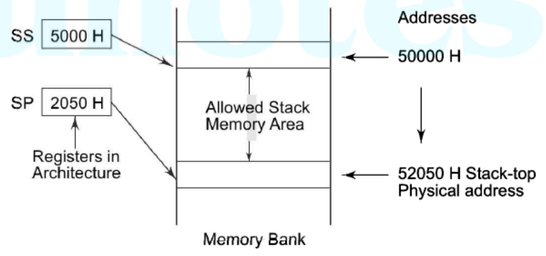

SS
5000 H
SP
2050 H
Registers in Architecture
Allowed Stack Memory Area
Addresses
50000 H
52050 H Stack-top Physical address
Memory Bank

Fig. 4.1 *Stack-top Address Calculation*

INTERRUPTS

An INTERRUPT is a condition that causes the microprocessor to temporarily work on a different task and then return to its previous task. Interrupt is an event or signal that request to attention of CPU. Whenever an interrupt occurs the processor completes the execution of the current instruction and starts the execution of an Interrupt Service Routine (ISR) or Interrupt Handler. ISR is a program that tells the processor what to do when the interrupt occurs. After the execution of ISR, control returns back to the main routine where it was interrupted.

INTERRUPT PINS

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

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 tothe 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. The following 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 pushedon to the stack.
- IP value is loaded from the contents of word location X × 4
- CS is loaded from the contents of the next word location.
- Interrupt flag and trap flag is reset to 0

<u>OPERATION SEQUENCE OF INTERRUPTS</u>

1. External interface sends an interrupt signal, to the Interrupt Request (INTR) pin, or an internal interruptoccurs.
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 hardwareinterface.
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-stepfeature.
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 x N) and then placed into the IP and from (4 x 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

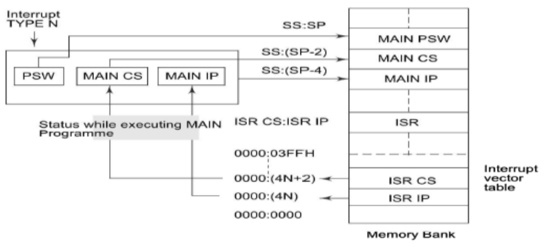


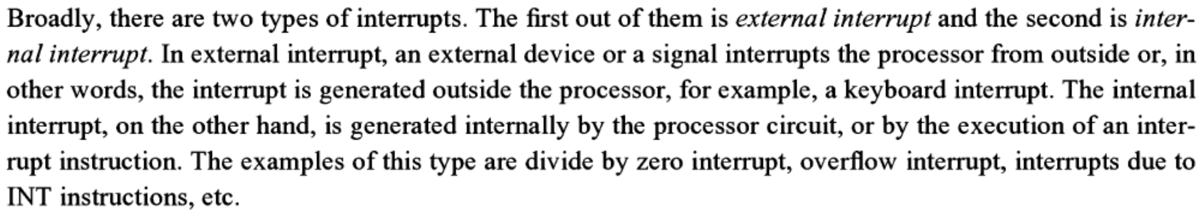
Fig. 4.4 *Interrupt Response Sequence*

<u>DIFFERENCE BTWEEN POOLING AND INTERRUPTS</u>

In INTERRUPT method, whenever any device needs service from microprocessor, the device notifies to processor by sending signal called interrupt. Upon receiving an interrupt signal, the microprocessor holds whatever it is doing and serves the corresponding device. The program associated with the interrupt is called the interrupt service routine (ISR) or interrupt handler.

In POLLING method, the microprocessor continuously monitors the status of a given device; when the status condition is met, it performs the service. After that, it moves on to the next device until each one is serviced. Although polling can monitor the status of several devices and serve each of them if certain conditions are met.

SOURCES OF INTERRUPTS


Broadly, there are two types of interrupts. The first out of them is *external interrupt* and the second is *internal interrupt*. In external interrupt, an external device or a signal interrupts the processor from outside or, in other words, the interrupt is generated outside the processor, for example, a keyboard interrupt. The internal interrupt, on the other hand, is generated internally by the processor circuit, or by the execution of an interrupt instruction. The examples of this type are divide by zero interrupt, overflow interrupt, interrupts due to INT instructions, etc.

TYPES OF INTERRUPTS

In general, there are two types of Interrupts:

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

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

HARDWARE INTERRUPTS: One source is from an external signal applied to NMI or INTR input pin of the processor. The interrupts initiated by applying appropriate signals to these input pins are called 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 two1. Maskable2. 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 associatedwith 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 Type 4 (Dedicated Interrupts)
- **TYPE 0** interrupt represents division by zero situation.
- **TYPE 1** interrupt represents single-step execution during the debugging of 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 afixed 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 into stack. It also stores the current status of the interrupts internally but not on 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

INTERRUPT VECTOR TABLE

The interrupt vector (or interrupt pointer) table is the link between an interrupt type code and the procedure that has been designated to service interrupts associated with that code. 8086 supports total 256 types i.e. 00H to FFH. The first 1Kbyte of memory of 8086 (00000 to003FF) 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.**

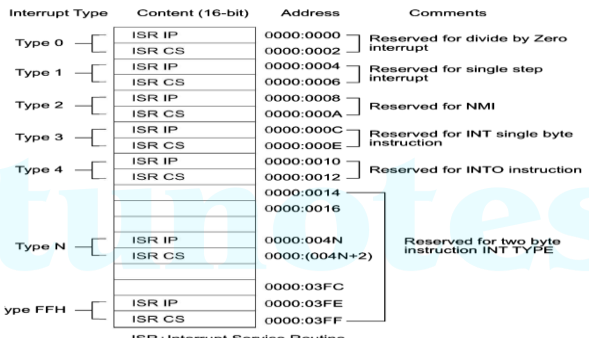


Fig. 4.5 *Structure of Interrupt Vector Table of 8086/88*

TYPES OF INTERRUPTS:

Predefined Interrupts:

Divide-By-Zero Interrupt-Type 0: The 8086 will automatically do a type 0 interrupt if the result of a DIV operation or an IDIV operation is too large to fit in the destination register. For a type 0 interrupt, the 8086 pushes the flag register on the stack, resets IF and TF and pushes the return addresses on the stack.

Single Step Interrupt-Type 1: The use of single step execution feature is found in some of the monitor & debugger programs. When we tell a system to single step, it will execute one instruction and stop. We can then examine the contents of registers and memory locations.

Non-maskable Interrupt-Type 2: The 8086 will automatically do a type 2 interrupt response when it receives a low to high transition on its NMI pin. When it does a type 2 interrupt, the 8086 will push the flags on the stack, reset TF and IF, and push the CS value and the IP value for the next instruction on the stack. It will then get the CS value for the start of the type 2 interrupt service procedure from address 0000AH and the IP value for the start of the procedure from address 00008H.

Breakpoint Interrupt-Type 3: The type 3 interrupt is produced by execution of the INT3 instruction. The main use of the type 3 interrupt is to implement a breakpoint function in a system. The breakpoint feature executes all the instructions up to the inserted breakpoint and then stops execution. The mnemonic for the instruction is INT3. Whenever we insert a breakpoint, the system executes the instructions up to the breakpoint and then goes to the breakpoint procedure.

Overflow Interrupt-Type4: The 8086 overflow flag will be set if the signed result of an arithmetic operation on two signed numbers is too large to be represented in the destination register or memory location

MASKABLE AND NON MASKABLE INTERRUPTS

NMI (Non mask-able interrupt)-

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

INTR (Maskable Interrupt)-

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

- On receiving an interrupt on INTR line, the microprocessor executes 2 INTA pulses.
- 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 themicroprocessor.
- Now microprocessor multiplies N x 4 and goes to the corresponding location in the IVT to obtain the ISR address. INTR is a mask-able interrupt.
- It is masked by making IF = 0 by software through CLI instruction.
- It is unmasked by making IF = 1 by software through STI instructions.

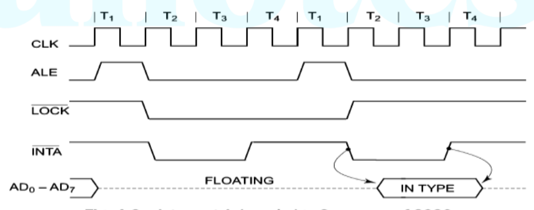


Fig. 4.6 *Interrupt Acknowledge Sequence of 8086*

Suppose an external signal interrupts the processor and the pin $\overline{LOCK}$ goes low at the trailing edge of the first ALE pulse that appears after the interrupt signal preventing the use of bus for any other purpose.
The pin $\overline{LOCK}$ remains low till the start of the next machine cycle.
With the trailing edge of $\overline{LOCK}$, the $\overline{INTA}$ goes low and remains low for two clock states before returning back to the high state.
It remains high till the start of the next machine cycle, i.e. next trailing edge of ALE.
Then $\overline{INTA}$ again goes low, remains low for two states before returning to the high state.The first trailing edge of ALE floats the bus AD_0-AD_7, while the second trailing edge prepares the bus to accept the type of the interrupt. The type of the interrupt remains on the bus for a period of two cycles.

INTERRUPT PROGRAMMING

While programming for any type of interrupt, the programmer must, either externally or through the program, set the interrupt vector table for that type preferrably with the CS and IP addresses of the interrupt service routine. The method of defining the interrupt service routine for software as well as hardware interrupt is the same. Figure 4.7 shows the execution sequence in case of a software interrupt. It is assumed that the interrupt vector table is initialised suitably to point to the interrupt service routine. Figure 4.8 shows the transfer of control for the nested interrupts.

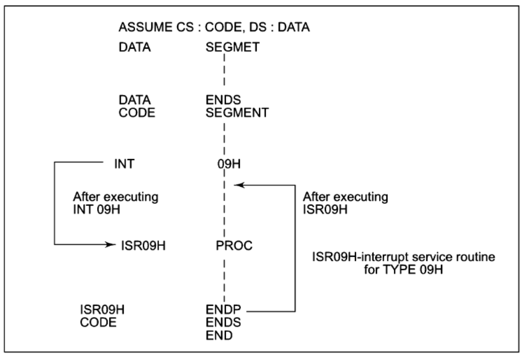


Fig. 4.7 *Transfer of Control during Execution of an Interrupt Service Routine*

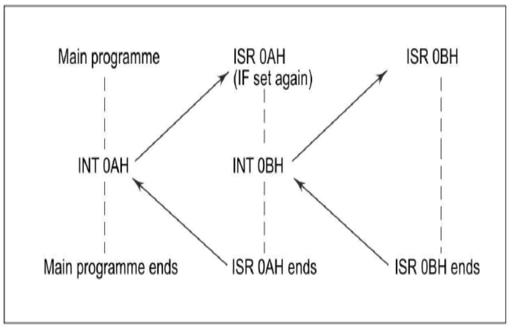


Fig. 4.8 *Transfer of Control for Nested Interrupts*

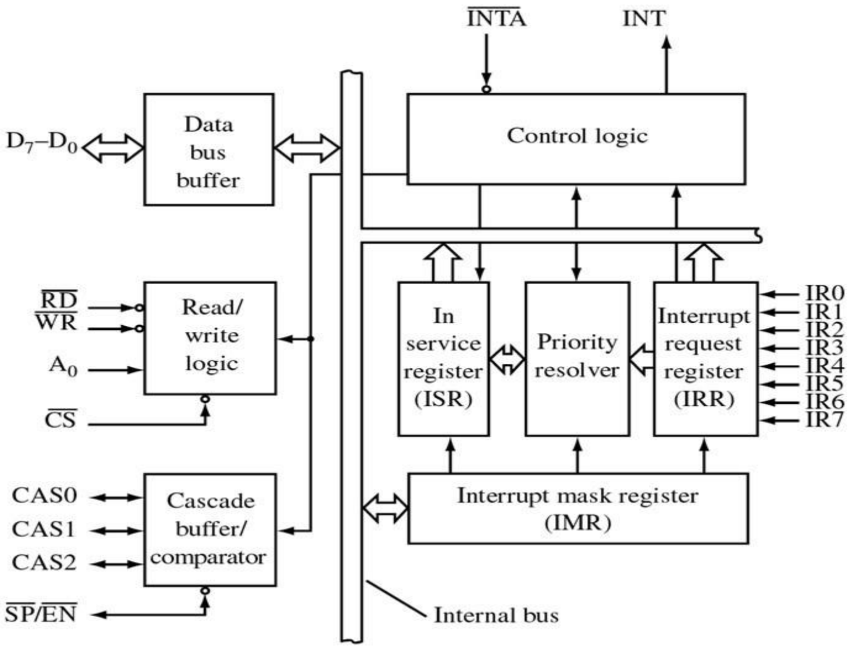
PROGRAMABLE INTERRUPT CONTROLLER 8259A

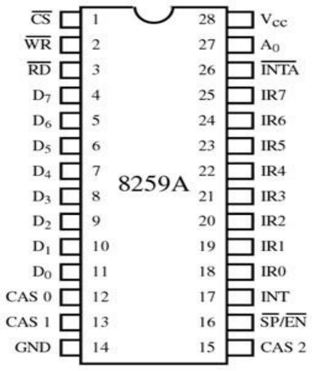
The 8259A is a programmable interrupt controller designed to work with Intel microprocessor 8080 A, 8085,8086, 8088.Programmable Interrupt controller able to handle no of interrupts at a time.

Features:

- 8 levels of interrupts.
- Can be cascaded in master-slave configuration to handle 64 levels of interrupts.
- Internal priority resolver.
- Individually maskable interrupts.
- Modes and masks can be changed dynamically.
- Accepts IRQ, determines priority, checks whether incoming priority > current level being serviced, issues interrupt signal.
- In 8086 mode, provides 8 bit vector number.
- Starting address of ISR or vector number is programmable.
- No clock required.

ARCHITECTURE OF 8259A





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 the priority basis.

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.

Priority Resolver This unit determines the priorities of the interrupt requests appearing simultaneously. The highest priority is selected and stored into the corresponding bit of ISR during $\overline{\text{INTA}}$ pulse. The IR_0 has the highest priority while the IR_7 has the lowest one, normally in fixed priority mode. The priorities however may be altered by programming the 8259A in rotating priority mode.

Interrupt Mask Register (IMR) This register stores the bits required to mask the interrupt inputs. IMR operates on IRR at the direction of the Priority Resolver.

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

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 data buffer during read or write operations.

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

Cascade Buffer/Comparator This block stores and compares the IDs of all the 8259As used in the system. The three I/O pins CAS0-2 are outputs when the 8259A is used as a master. The same pins act as inputs when the 8259A is in the slave mode. The 8259A in the master mode, sends the ID of the interrupting slave device on these lines. The slave thus selected, will send its pre-programmed vector address on the data bus during the next $\overline{\text{INTA}}$ pulse.

PIN DESCRIPTION:

Pins	Description
CS	Active low chip for enabling RD/WR operations.
$\overline{\textbf{WR}}$	Active low write enable input. This enables to accept command words from CPU
$\overline{\textbf{RD}}$	Active low read input. Enables to release status onto data bus
$D_7 – D_0$	These pins forms bidirectional data bus that carries 8 bit data either to control word or from status word register. These also carries interrupt vector information
$CAS_2 – CAS_0$	A single 8259A gives 8 interrupts, if more interrupts are required, the 8259A is used in cascaded mode to provide 64 interrupts lines. Also act as select lines for addressing slaves in 8259A
$\overline{\textbf{PS}}/\overline{\textbf{EN}}$	This pin is a dual-purpose pin. When chip is used in buffered mode, it can be used as buffer enable to control buffer transceivers. If not used in buffered mode then pin is used as input to designate whether chip is used as master or slave.
INT	This pin goes high whenever a valid interrupt request is asserted. This is used to interrupt CPU and is connected to interrupt input of CPU
$IR_0 – IR_7$	These pins act as input to accept requests to CPU.
$\overline{\textbf{INTA}}$	This pin is an input used to strobe 8259A interrupt vector data onto to the data bus.

INTERRUPT SEQUENECE IN 8086

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.
3. The CPU acknowledge with INTA pulse.

4. Upon receiving an INTA signal from the CPU, the highest priority ISR bit is set and the corresponding IRR bitis 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 on to a data busfrom where it is read by the CPU.
6. This completes the interrupt cycle. The ISR bit is reset at the end of the second INTA pulse if automatic end ofinterrupt (AEOI) mode is programmed. Otherwise ISR bit remains set until an appropriate EOI command is issued at the end of interrupt subroutine.

INTERFACING MEMORY WITH 8086

Static RAM Interfacing:

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', as described in memory organisation in Chapter 1.
2. Connect available memory address lines of memory chips with those of the microprocessor and also connect the memory $\overline{RD}$ and $\overline{WR}$ inputs to the corresponding processor control signals. Connect the 16-bit data bus of the memory bank with that of the microprocessor 8086.
3. The remaining address lines of the microprocessor, $\overline{BHE}$ and A_0 are used for decoding the required chip select signals for the odd and even memory banks. The $\overline{CS}$ of memory is derived from the O/P of the decoding circuit.

Program 5.1

Interface two 4K × 8 EPROMS and two 4K × 8 RAM chips with 8086. Select suitable maps.

Table 5.1 *Memory Map for Problem 5.1*

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							

Table 5.1 (*Contd.*)

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}
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

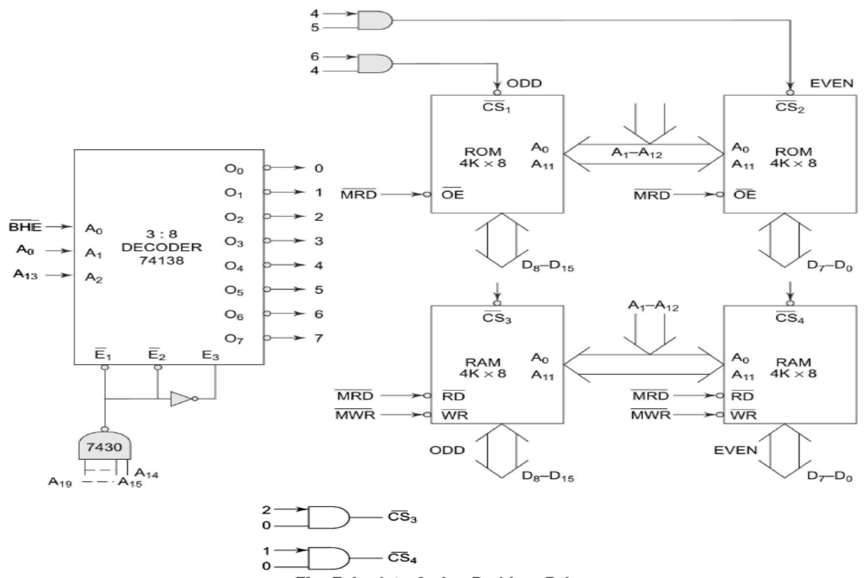


Fig. 5.1 *Interfacing Problem 5.1*

Table 5.2 *Memory Chip Selection for Problem 5.1*

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 addresses 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 addresses in ROM
Byte transfer on $D_0 - D_7$	1	0	1	Only even address in ROM
Byte transfer on $D_8 - D_{15}$	1	1	0	Only odd address in ROM

Program 5.2

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 00000H.

Table 5.3 *Address Map for Problem 5.2*

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

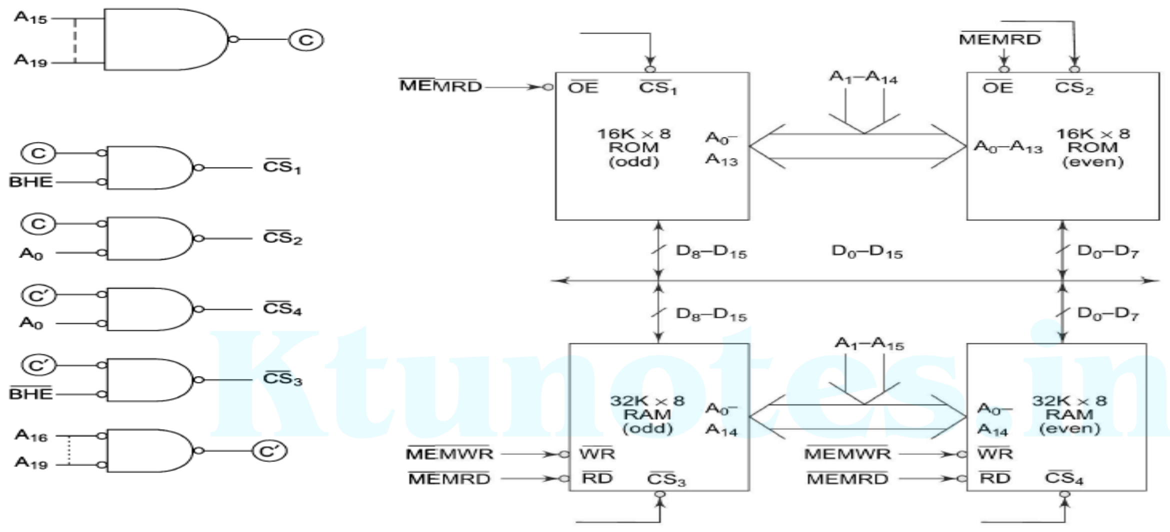


Fig. 5.2 *Interfacing Problem 5.2*

Problem 5.3

It is required to interface two chips of 32K × 8 ROM and four chips of 32K × 8 RAM with 8086, according to the following map.

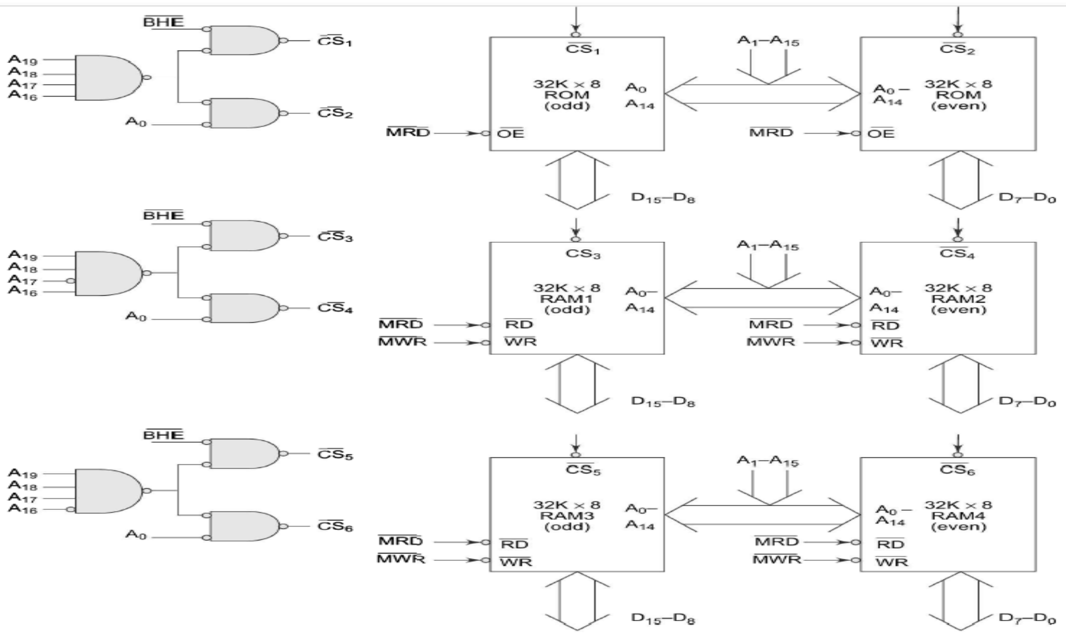
ROM 1 and 2 F0000H - FFFFFH, RAM 1 and 2 D0000H - DFFFFH

RAM 3 and 4 E0000H - EFFFFH

Show the implementation of this memory system.

Table 5.6 *Address Map for Problem 5.3*

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_0
F0000H	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ROM 1and2										64K										
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 1and2										64K										
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 3and4										64K										
EFFFFH	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



BHE
A19
A18
A17
A16
A0
CS1
CS2
CS3
CS4
CS5
CS6
32K × 8 ROM (odd)
32K × 8 ROM (even)
32K × 8 RAM1 (odd)
32K × 8 RAM2 (even)
32K × 8 RAM3 (odd)
32K × 8 RAM4 (even)
A0–A14
A1–A15
MRD
MWR
OE
RD
WR
D15–D8
D7–D0