Definitions

Instruction cycle -

Time required by the µp to execute one instruction is called Instruction cycle.

One instruction consists of either one or more than one machine cycles.

Total no of clock-cycles (T-states) varies from 4 clock cycles to 18 clock cycles.

Machine cycle -

Time required by µp to perform one operation is called machine cycle.

Examples of m/c cycles (or operations) :-

i) Opcode fetch
ii) Memory read
iii) Memory write
iv) I/o Read
v) I/o write
vi) Halt, Hold, Reset

Clock cycle or T-state -

µp uses a square wave signal called as clock signal. The time period of one clock cycle is called one T-state.

Crystal freq = 6.28 MHz

Clock freq. = **3.14 MHz**

$$T\text{-state} = \frac{1}{f} = \frac{1}{3.14\,MHz} = \boxed{0.3\,\mu s}$$

How to recognize Machine cycles / T-states

or

Calculation of Machine cycles / T-states required by instructions.

There is no direct relationship b/w no. of bytes in an instruction & no. of m/c cycles required to execute the instruction.

However, in MOV C, A M_1 = opcode fetch
T-states = 4

MVI A, 32H M_1 = opcode fetch (4 T-states)
M_2 = memory read (3 T-states)

Examples :-

		bytes	M/c cycles	T-states
(1)	STA 2065H	3	4	13

M_1 = opcode fetch for STA — 4
M_2 = for reading lower byte (65H) / memory read — 3
M_3 = for reading higher byte (20H) / memory read — 3
M_4 = memory write / for writing contents of A on to the memory location. — 3

$\Rightarrow$ Total m/c cycles = $M_1 + M_2 + M_3 + M_4 = 4$
Total T-states = $4 + 3 + 3 + 3 = 13$

		Bytes	M/c cycles	T-states
(2)	LXI D, 9506H	3	3	10

M_1 = opcode fetch for LXI D — 4
M_2 = memory read (06H) — 3
M_3 = " " (95H)

Total m/c cycles = 3 ($M_1 + M_2 + M_3$)
" T-States = $4 + 3 + 3 = 10$

Ques 1. Calculate M cycle / T states for following instructions:-

i) MOV A, B (1M/4 T-states) 4
ii) MOV C, M (2M/7 T-states) 3
iii) LDAX B (2M/7 T-states) 3
iv) LHLD 9506 H (5M/16 T-states) 3
v) DCR B (1M/4 T-states) DCR M (3M/10 T-states) LHLD 1050 H
vi) DCX B / INX B (1M/6 T-states) → extra 2 for decrementing reg. pair
vii) ORA C (1M/4 T-states)
viii) PUSH B (3M/ 6+3+3 = 12 T-states)
ix) POP D (3M/10 T-states)
x) JMP 9000 H (3M/10 T-states)
xi) cond. jump (2M/7T if cond. is not true; 3M/10 T if cond. is true)
xii) CALL 5000 H (5M/18 T states)
xiii) RET (3M/10 T-states) = (4/t2) + 3 + 3 + 3 + 3.
decrementing SP; for storing contents of PC in memory location

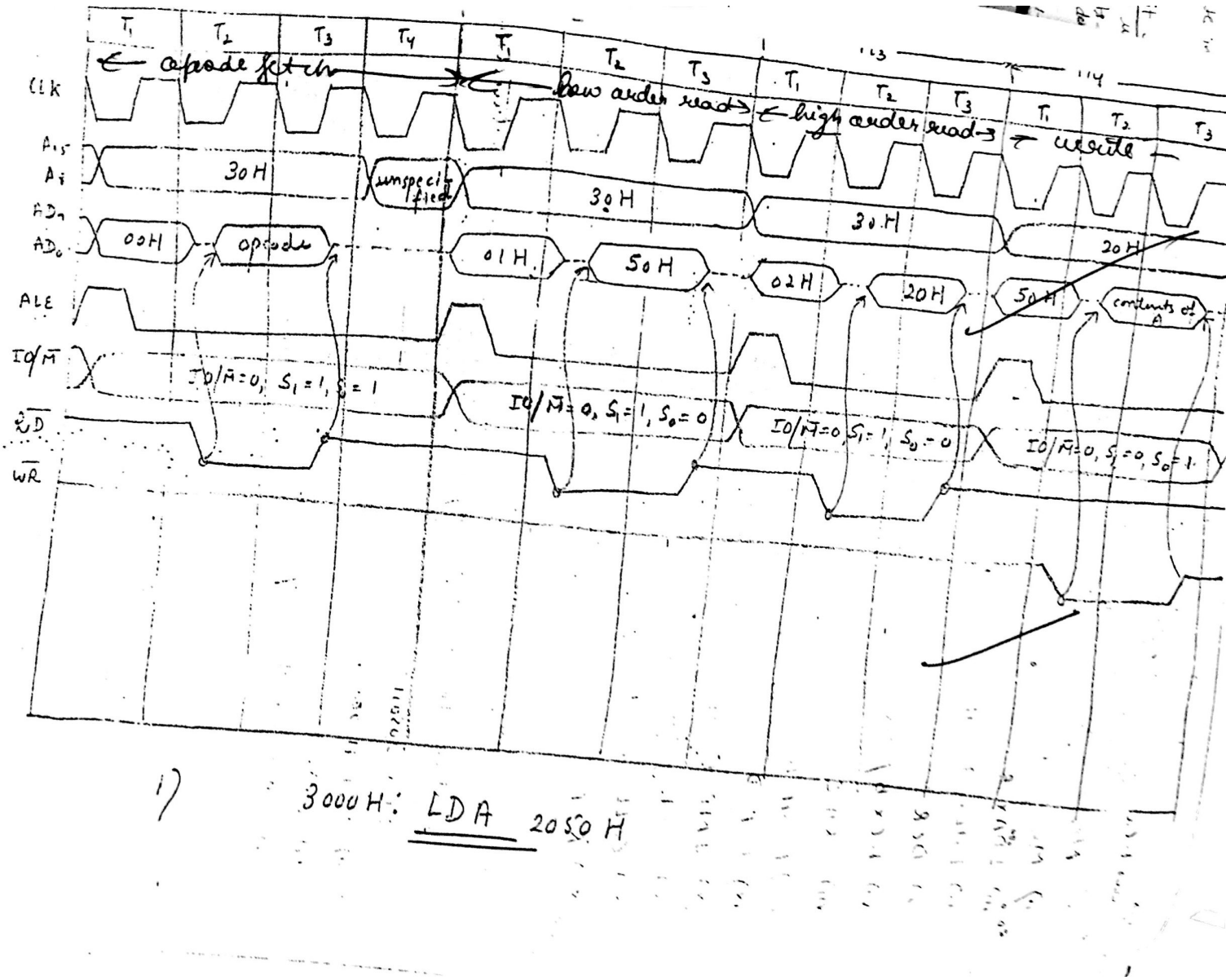
Ques. 2. Draw the timing diagram of the following instructions:-

i) LDA 2050H (4M/13 T-states)
ii) LXI H, 2000H (3M/10 T-states)
iii) LDAX R_p (2M/7 T-states)
iv) CALL 6000H (5M/18 T-states)

13

i) LDA 3000 H : LDA
3001 H : 50H
3002 H : 20H

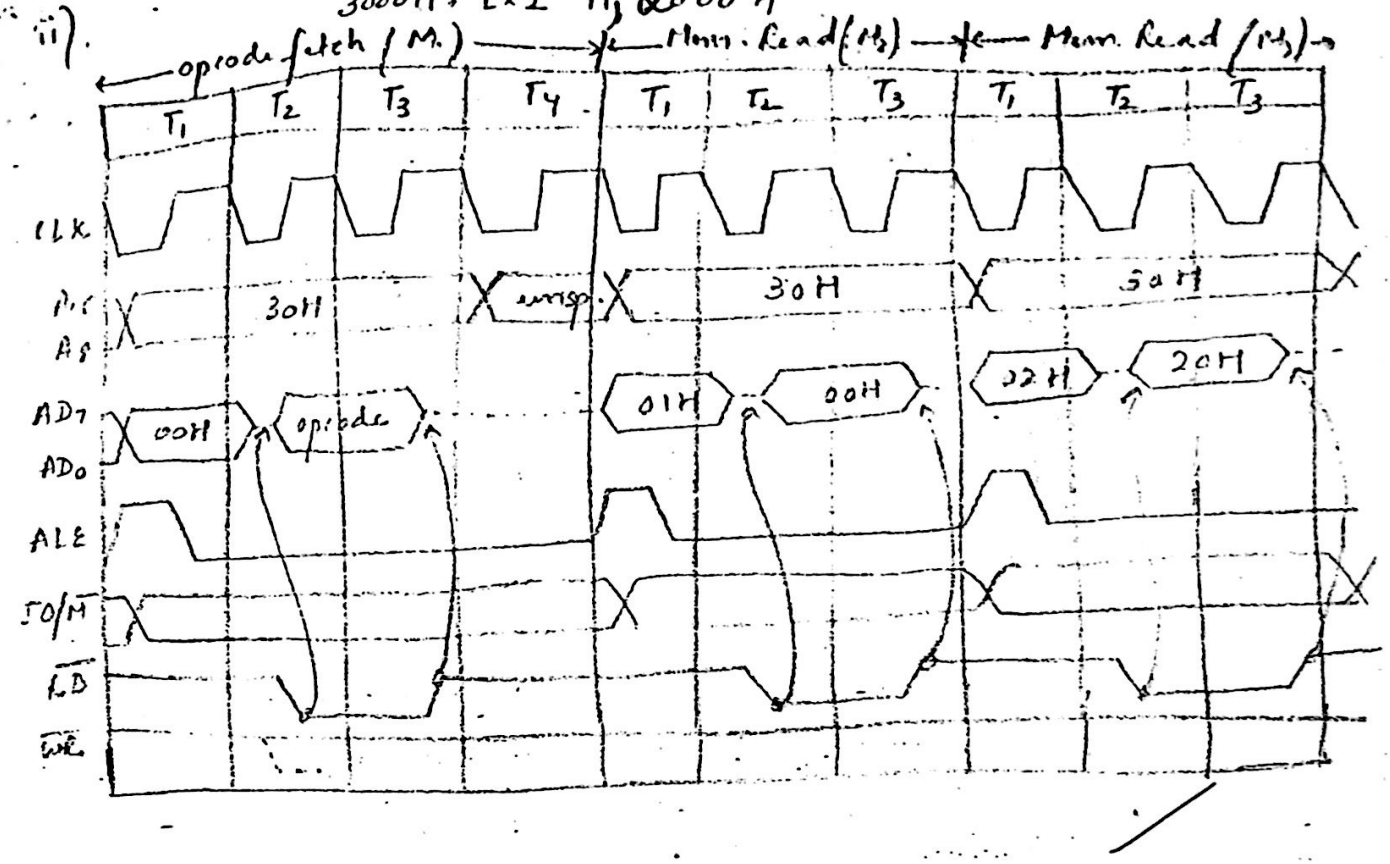
Total M/c cycles = $M_1 + M_2 + M_3 + M_4$
M_1 - opcode fetch, M_2 - Memory read, M_3 Memory read, M_4 Memory write



1) 3000H: LDA 2050H

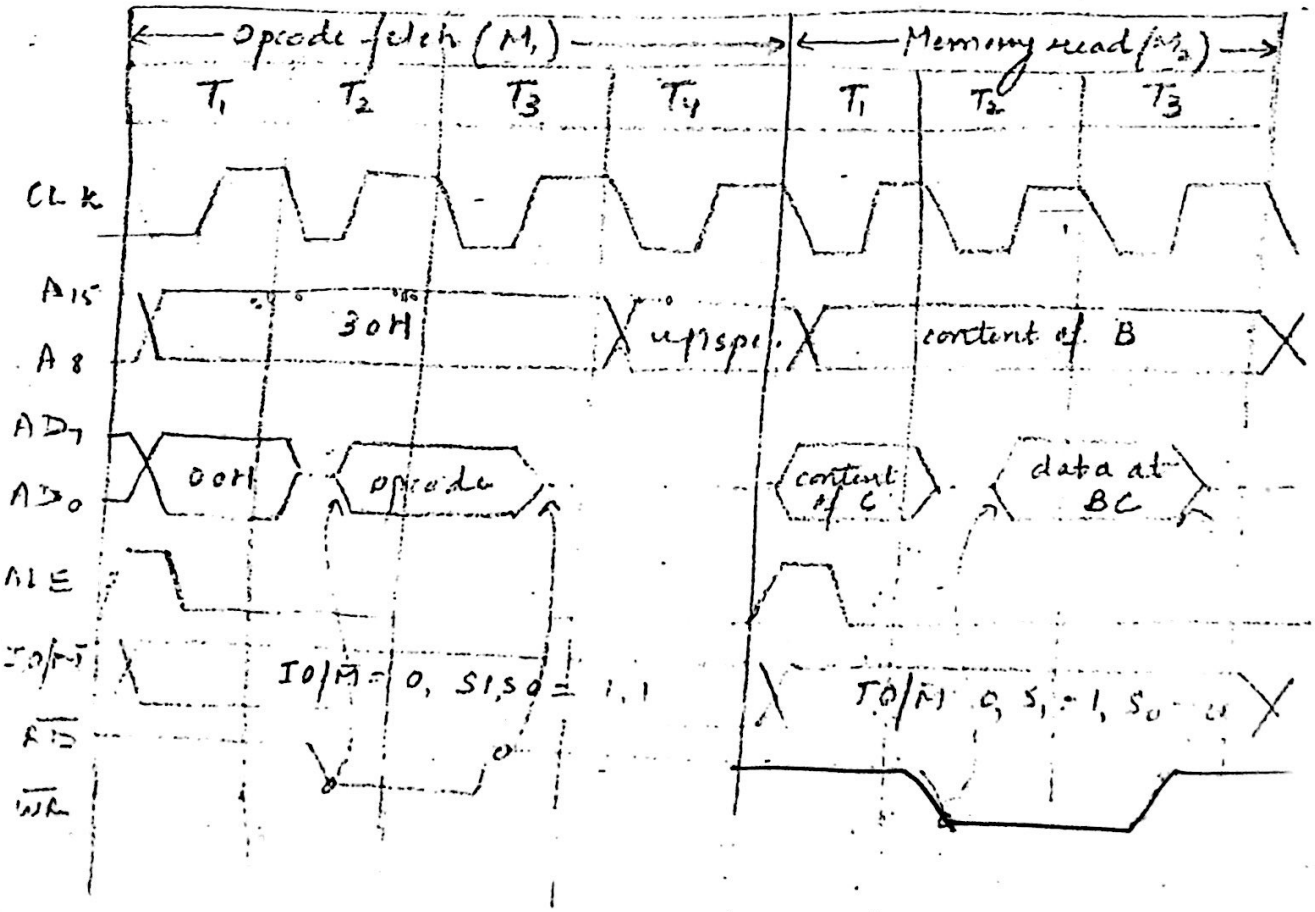
ii)

3000H : LXI H, 2000H

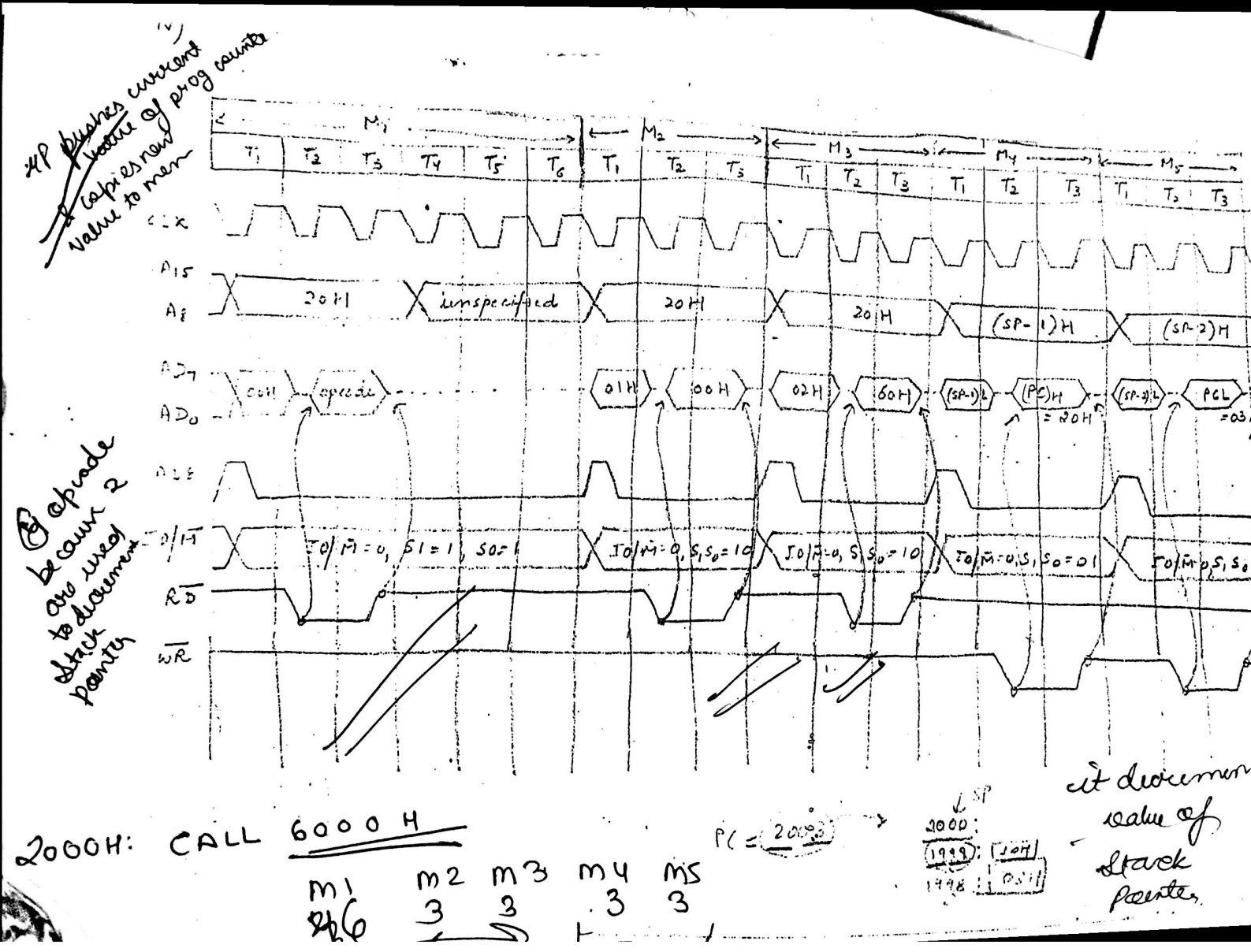


iii)

3000H : LDAX B



(v)



2000H: CALL 6000H

M1	M2	M3	M4	M5
6	3	3	3	3

PC = 2000

SP
2000:
1999: 20H
1998: 00H

Scanned by CamScanner

Interfacing I/O Devices

I/O devices are the means through which MPU communicates with the outside world.

MPU accepts binary data as i/p from devices such as keyboards & A/D converters and, sends data to o/p devices such as LEDs or printers

There are two techniques for getting data into or out:-

i) Parallel I/O mode or interface - Here, data enters or exits in group of 8-bits. H/w adds to the cost but it is fast.

ii) Serial I/O mode or interface - Here, one bit is transferred using one data line. Here, disadvantage is low speed. But, it is cheaper & useful for long distance transmission.

MPU needs to identify I/O devices with a binary number. There are two different methods by which I/O devices can be identified:- (These binary nos. are referred to as I/O device add. or port numbers.)

one uses an 8-bit address
Peripheral-mapped I/O or
I/O-mapped I/O

i) In this type of I/O, MPU uses eight address lines to identify an i/p or an o/p device; this is known as I/O-mapped I/O.

This is also known as I/O space (separate from memory space) & it ranges from 00H to FFH.

00H to FFH

ii) Here, device is enabled by I/O related control signals ($\overline{IOR}$, $\overline{IOW}$)

other uses a 16-bit address
Memory-mapped I/O

i) In this type of I/O, MPU uses 16 add. lines to identify an I/O device; an I/O is connected as if it is a memory register. This is known as memory-mapped I/O.

This results in numbering scheme ranging from 0000H to FFFF H, also known as memory space.

ii) Here, device is enabled by memory-related control signals ($\overline{MEMR}$ & $\overline{MEMW}$)

iii) Here, data bytes are transferred using IN/OUT instructions.

iii) Here, data bytes are transferred using memory related instructions such as LDA, STA.

Peripheral - mapped I/O -

(1.) Peripheral I/O Instructions :-

8085 MPU has 2 instructions for data transfer b/w the processor and the I/O device : IN and OUT

i) D3 ? | OUT 8-bit port address | (opcode D3)

This is a 2-byte instruction ; first byte is D3 (opcode for OUT) & 2nd byte is port address of o/p device.

This instruction transfers (copies) data from acc. to the o/p device.

ex:- 2050 D3 OUT (01H) — say for LED display.
2051 01 ; 8085 can communicate with 256 diff-o/p ports with device add. ranging from 00H to FFH.

DB

ii) | IN 8-bit port address | (opcode DB)

This is a 2-byte instruction ; first byte is DB & second byte is port address of an i/p device.

This instruction transfers (copies) data from an i/p device & places that byte in the accumulator.

ex:- 2065 DB IN (84H) — Similarly, IN can accept data from 256 different i/p ports.
2066 84

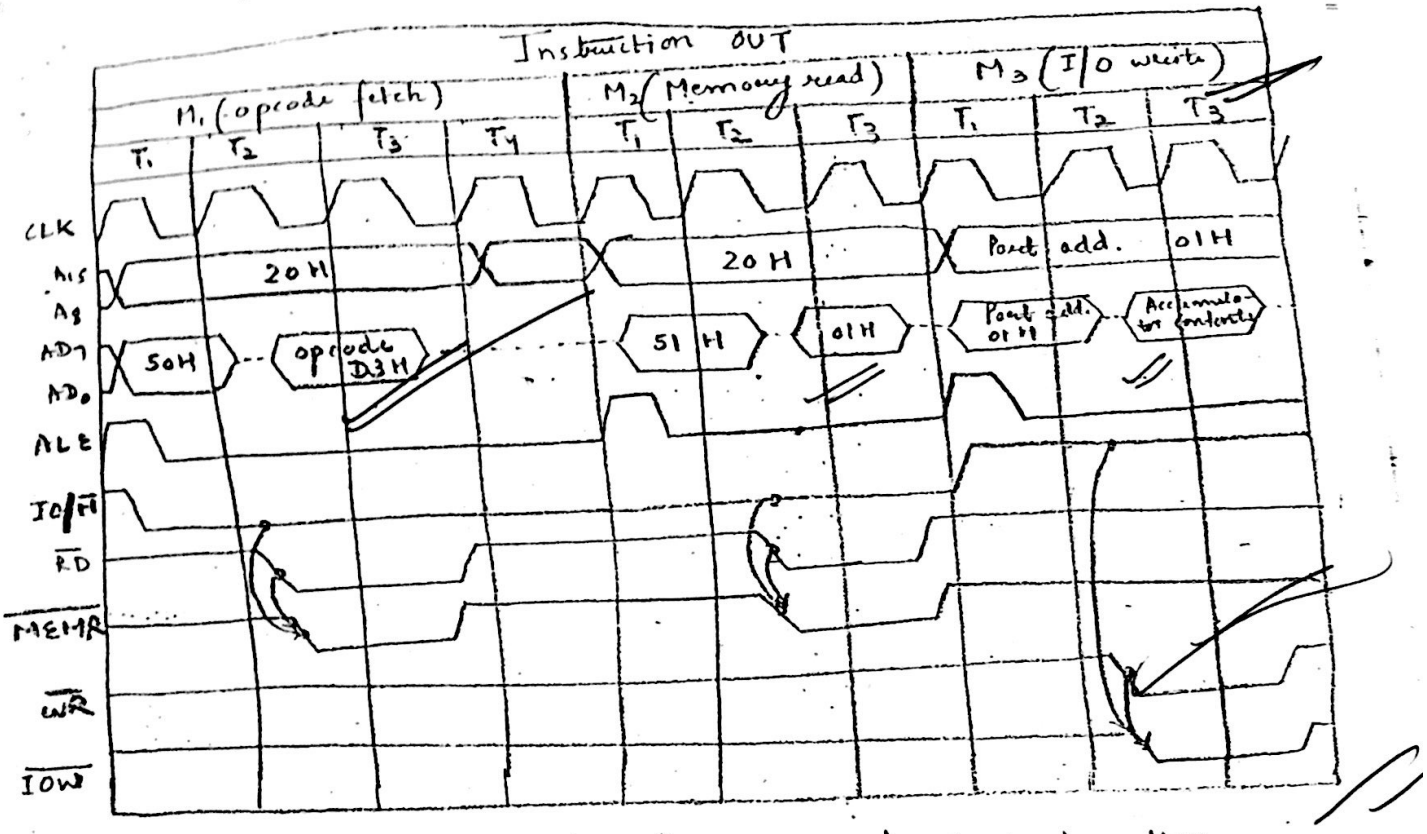
4+3+3

(2) I/O execution :-

i) 2050 OUT 01 H

8085 executes OUT instruction in 3 M/c cycles & ~~ten T-states~~.

2050 OUT 01H



Timing for execution of OUT instruction.

M_1 – 8085 places high order add. 20H on $A_{15}-A_8$ & low order address 50H on AD_7-AD_0. Here ALE goes high & IO/$\bar{M}$ goes low (mem. related operation).

At T_2, $\overline{RD}$ is combined with IO/$\bar{M}$ to generate $\overline{MEMR}$, used to fetch D3 from data bus.

M_2 – 8085 places next add. 2051 on add. bus & gets device add. 01H on data bus

M_3 (I/O write) – 8085 places 01H on low order add. bus as well as high order add. bus. IO/$\bar{M}$ goes high (∵ it is an I/O operation).

At T_2, acc. contents are placed on the data bus.

ii) 2065 : IN 84H

Here, M_1, M_2 cycles are identical.
In M_3, 8085 places add. of i/p port on low order as well as high order address bus & asserts $\overline{IOR}$ signal. $\overline{IOR}$ enables i/p port & data from i/p port is placed on data bus & transferred to the accumulator.

Imp.

(3.) Device Selection & Data transfer -

By interfacing an o/p device we get info. from processor or accumulator & store it or display it.

∴, data bus must be connected to a latch & latch must be enabled when IO/$\overline{M}$ is high & $\overline{WR}$ is active low.

The steps & block dig. are given below:-

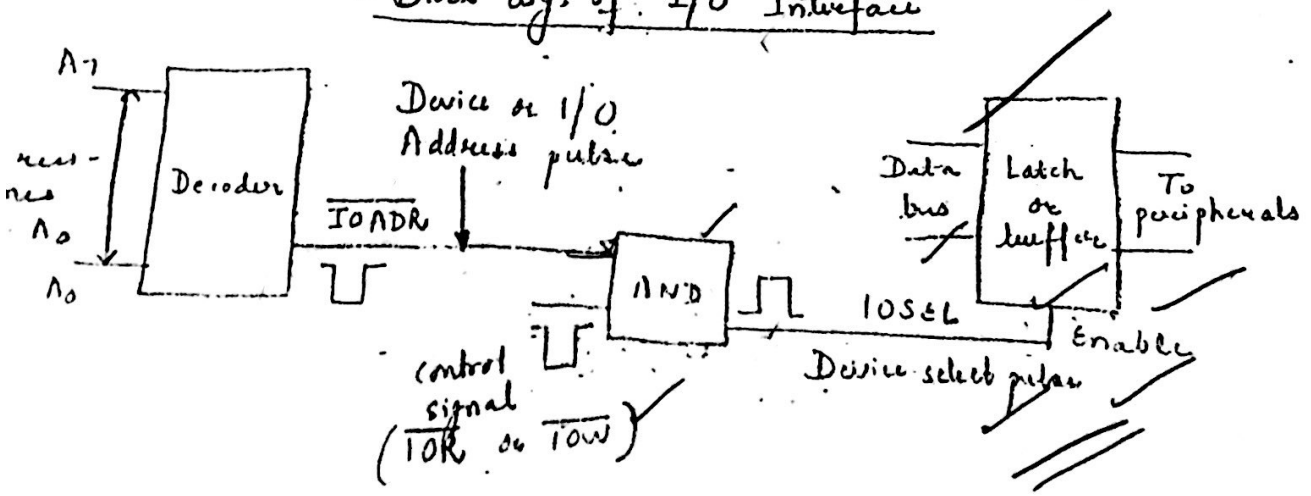
i) Decode the address bus to generate a unique pulse corresponding to device address on the bus - called as - device address pulse or I/O add. pulse.

ii) Combine (AND) device address pulse with control signal to generate a device select (I/O select) pulse that is generated only when both signals are asserted.

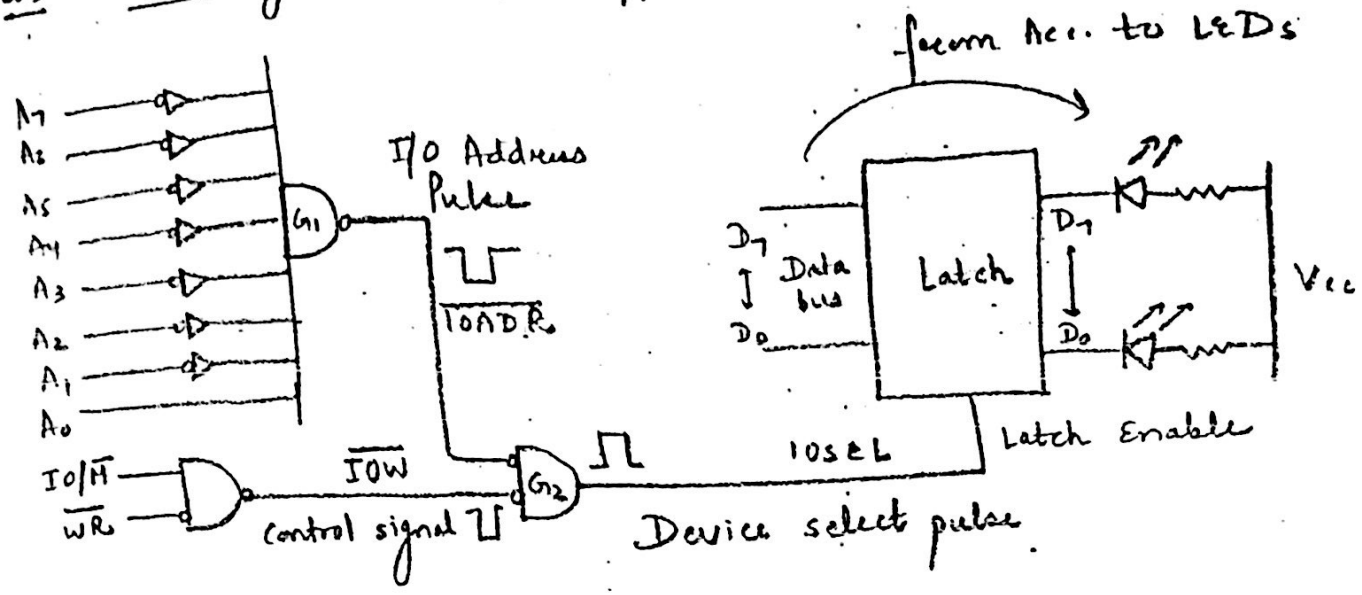
iii) Use I/O select pulse to activate interfacing device (I/O port).

Imp.

Block dig. of I/O Interface



ex:- Decoding circuit for o/p device with address 01H



OUT 01 H

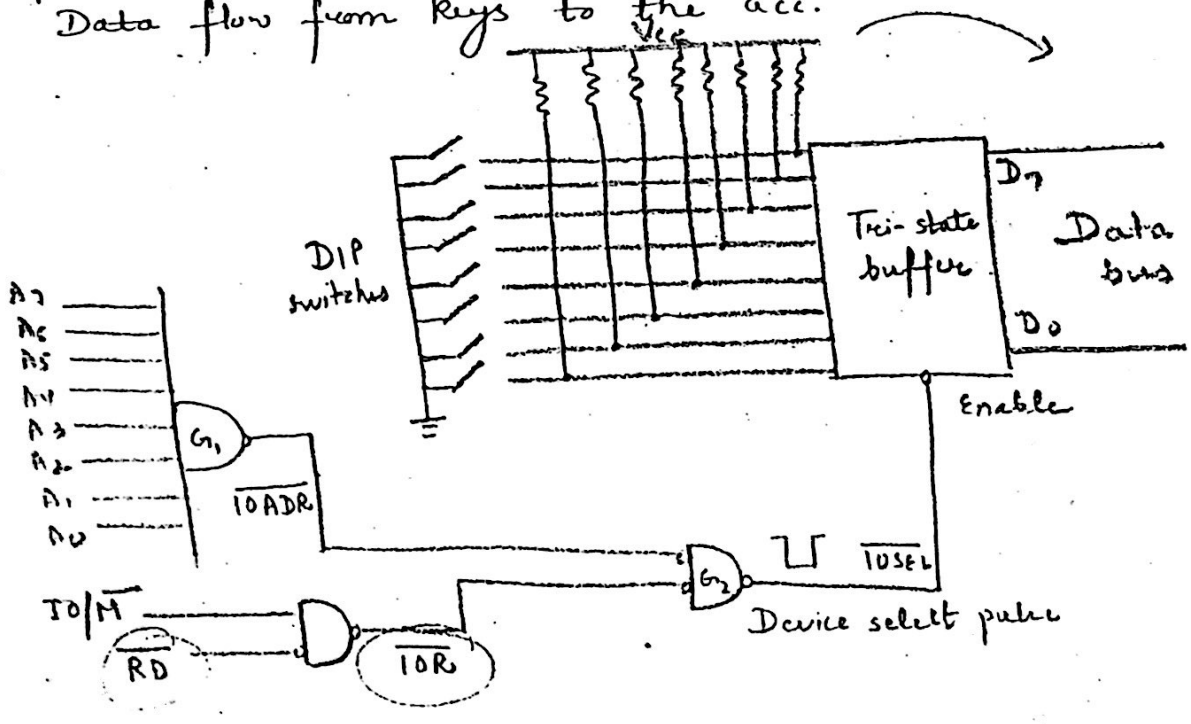
Decode logic for LED o/p port

ex:- Decoding circuit for i/p device with address - ffH.

(1) use $\overline{IOR}$ in place of $\overline{IOW}$

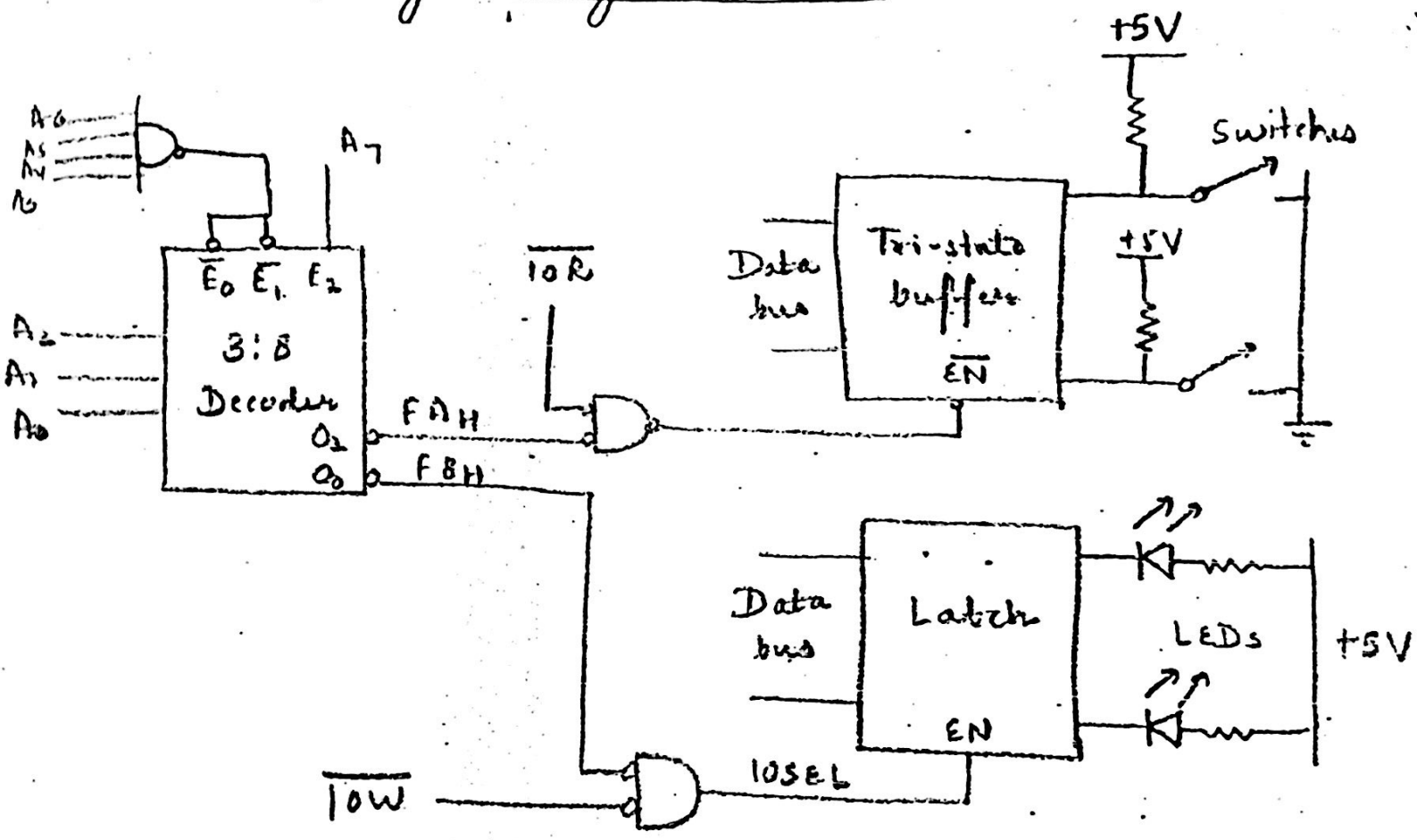
(2) Tri-state buffer is used as an interfacing port in place of the latch

(3) Data flow from keys to the acc.



Decode logic for Dip-switch I/P port

Interfacing using decoders



Address decoding using 3:8 Decoder

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
1	1	1	1	1	0	0	0	= F8H
1	1	1	1	1	0	1	0	= FAH

Memory - mapped I/O -

i) Here, i/p & o/p devices are assigned to identified by 16-bit addresses.

ii) Here, memory related instructions (such as LDA & STA) memory control signals ($\overline{MEMR}$ & $\overline{MEMW}$) are used.

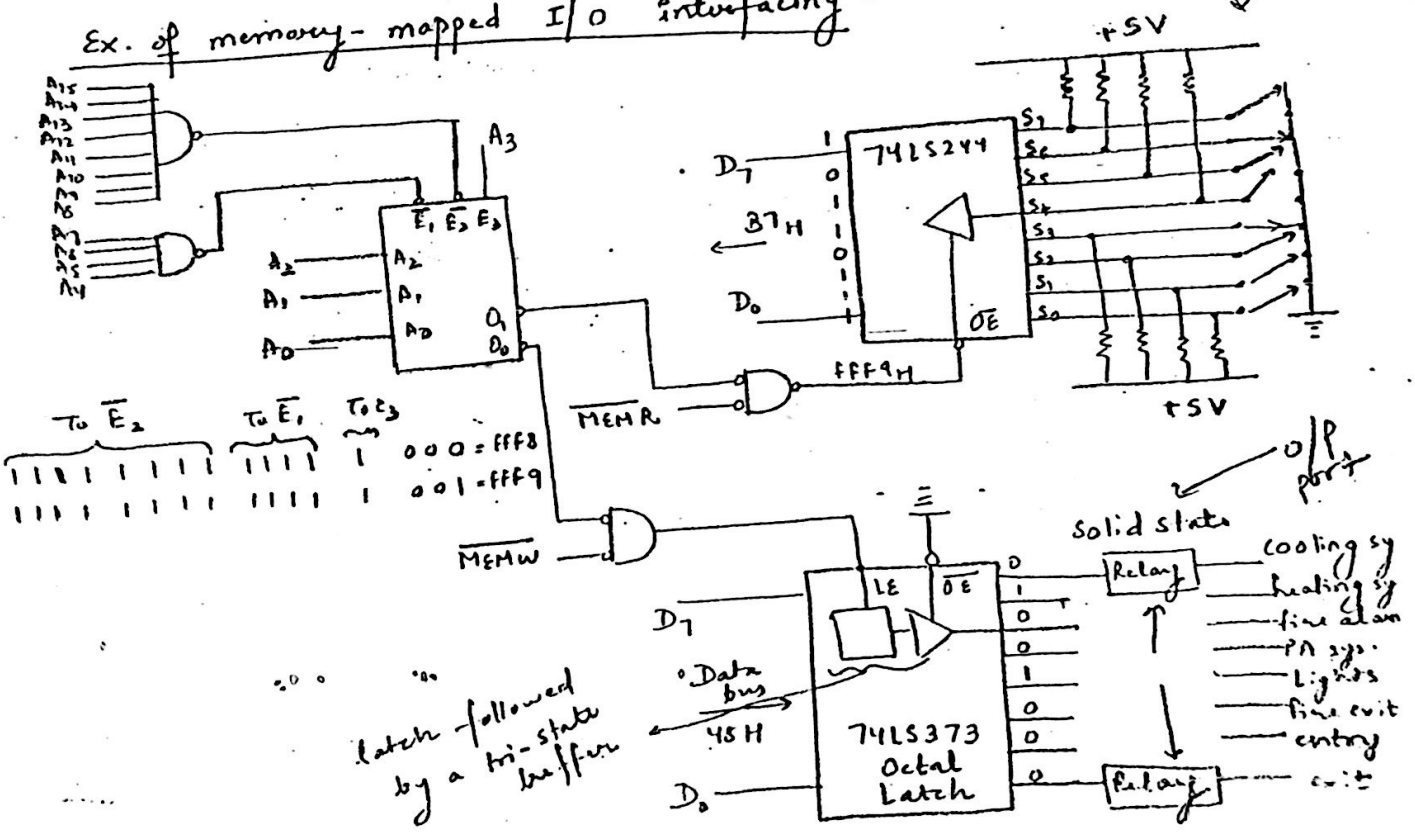
iii) MPU communicates with I/O device as if it were one of the memory locations.

ex:- 2050 : 32
2051 : 00
2052 : 80
} STA 8000 H (transfers data from acc. to memory location)

LDA 2000 H (transfers data from memory location to acc.)

Execution of memory related data transfer instructions (already discussed - timing dig.)

Ex. of memory - mapped I/O interfacing -



Memory - mapped I/O interfacing

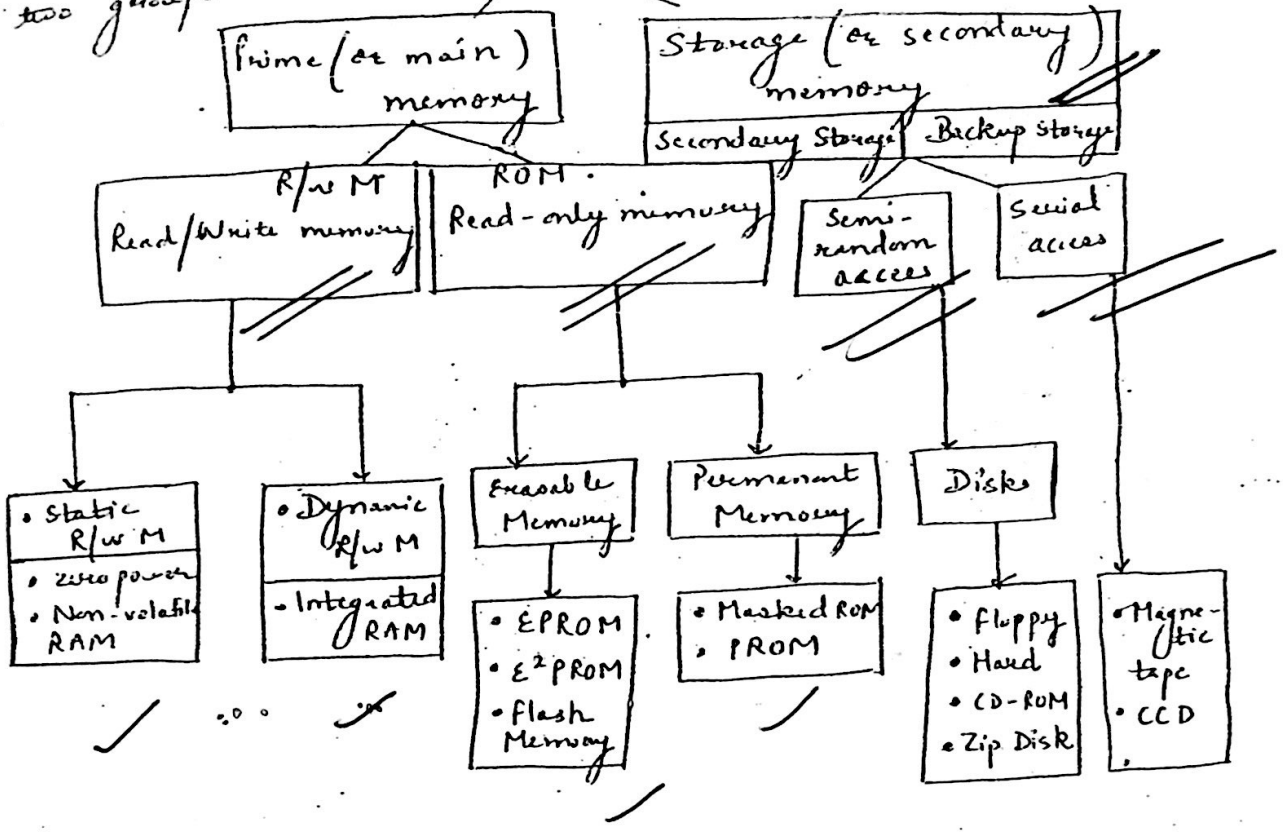
Comparison of Memory-mapped I/O & Peripheral I/O

Characteristics	Memory-mapped I/O	Peripheral I/O
1) Device address	16-bit	8-bit
2) control signals for I/O	MEMR / MEMW	IOR / IOW
3) Instructions available	STA, LDA STAX, LDAX MOV MR ADD M SUB M ANA M	IN and OUT
4) Data transfer	B/w any reg & I/O	only b/w I/O & acc.
5) Max. no. of I/Os possible	Memory map (64K) is shared b/w I/Os and system memory.	I/O map is independent of memory map; 256 i/p devices & 256 o/p devices can be connected
6) Execution speed	13 T-states (STA, LDA) 7 T-state (MOV M,R)	10 T-states
7) H/w requirements	More h/w is needed to decode 16-bit addr.	less h/w is needed to decode 8-bit addrs.
8) Other features	Arithmetic or logical op. can be directly performed with I/O data	Not available.

Memory

Memory is an essential component of a microcomputer system; it stores binary instructions and data for the microprocessor.

There are various types of memory, which can be classified in two groups :-



1. Prime Memory -

i) This memory is used by microprocessor in executing and storing programs.

ii) This memory should be able to respond fast enough to keep up with the execution speed of the microprocessor.

iii) It should be random access memory i.e. µp should be able to access information from any register with the same speed.

iv) Size of memory is specified in term of bits. ex:- 1K memory chip means 1K (1024) bits.

But in PC, it is specified in bytes. Ex: 4M means PC hai 4 Megabytes of memory.

It is further divided into two groups:-

(1.) R/W Memory - (or RAM)

µp can write into or read from this memory. It is used for information that is likely to be altered such as programs or receiving data. This is volatile (when power is turned of all contents are destroyed).

There are two types of R/W memories:-

Static memory (SRAM)	Dynamic memory (DRAM)
i) This memory is made up of FFs, and it stores bit as a voltage.	i) This memory is made up of MOS transistor gates, & it stores bit as a charge
ii) The memory has low density & consumes more power than dynamic RAM.	ii) It has high density & low power consumption.
iii) This memory is more expensive	iii) It is cheaper than static memory.
iv) This memory has high speed.	iv) Here, charge information (bit info.) leaks ∴ stored information needs to be read & written again every few mseconds. This is called refreshing the memory. It requires extra circuitry, adding to the cost of the system & also decreases the speed.
v) For small systems, static memory is appropriate	v) Dynamic memory is economical to use when system memory size is atleast 8K.
vi) In high speed processors like Intel i5 & Pentium, SRAM known as Cache memory	vi) DRAM is too slow ∴ various techniques are being used to increase speed of DRAM which resulted in high speed chips such as -

is included on the processor chip.

EDO (Extended Data Out),
SDRAM (Synchronous DRAM),
RDRAM (Rambus DRAM).

Cache Memory -

i) This memory is placed b/w the CPU & main memory.
ii) It is a semiconductor memory & consists of static RAMs.
iii) Its access time is about 10ns which is much less than access time of main memory i.e. about 50ns.
iv) Its capacity is 2 to 3% of main memory.
v) It stores instruction codes & data which are to be immediately used by the CPU. This reduces the average access time for instructions & data normally stored in main memory.
vi) Cache memory also needs a cache controller.
vii) 32-bit & 64-bit µp operate at very high speed of 400MHz to 3.8GHz. So, memory used with these high-speed µp must be very fast ∴, fast memory is expensive & has to operate with several wait states with main memory, high speed cache is used to supply currently needed instructions & data to the CPU.
viii) There are two types of cache schemes :- Write-through and write-back.
ix) Write-through :- Here, main memory is updated each time CPU writes into cache ∴, main memory always contains the same data as cache contains. This is desirable in a system which used DMA (Direct Memory Access) transfer. Therefore, I/O devices communicating with DMA receive most recent data

x) Write-back scheme - In this scheme, only the cache memory is updated during a write-operation. The updated locations are marked by a flag so that later on when the word is removed from the cache, it is copied into the main memory. (The words are removed from time to time to make space for new block of words). This is done during free processor cycles. This requires additional h/w support, but improves performance, ∵ exchanges s/w cache & main memory are fewer & better timed. This scheme is faster & hence it is preferred.

xi) Mobocache is static RAM used in computers.

SDRAM (Synchronous DRAM) -

i) It uses the same clock rate as the CPU. As a result, the memory chip remain ready to transfer data when the CPU expects them to be ready.

ii) They run at the processor memory bus without imposing wait states.

SGRAM (Synchronous Graphics RAM) -

i) It is synchronous RAM suitable for graphics applications.

DDR SDRAM (Double Data Rate Synchronous DRAM) -

i) It has evolutionary advantage / advancement over SDRAM as it doubles the data transfer rate.

ii) It is faster version of SDRAM.

iii) Standard SDRAM does all actions on the rising edge of the clock signal whereas DDRAM transfer data on both edges of the clock, resulting in double transfer rate for burst data transfer.

iv) To achieve double-data-rate, memory cell array is arranged in two banks. Each bank can be accessed separately. Consecutive words of a given block of data are stored in diff. banks. Such interleaving of words permits simultaneous access to two words which are transfered on successive edge of the clock.

v) SDRAM & DDRAM are used for block transfer of data. Ex: in computers from main memory to cache & in video display.

DDR2 & DDR3 are later versions of DDR. DDR2 is widely used in portable desktop & server computers. Data transfer rates of DDR, DDR2, DDR3 are 0.4, 0.8 and 1.6 Gbps respectively.

RDRAM (RAM bus DRAM) -

i) It was not succesful much.

ii) Later on, improved RAM (called XDR DRAM) was developed which uses octal data rate transfers to transfer 8-bit of data. It gives 64Gbps BW for 16-bit bus & operates at a frequency of 400 MHz.

EDO (Extended Data Output) -

i) It is modified DRAM.

ii) Here, any memory access (including refresh) stores 256 bits into latches. ∴, latches hold 256 bits of info., so in most programs which are sequentially executed, data are available without wait states.

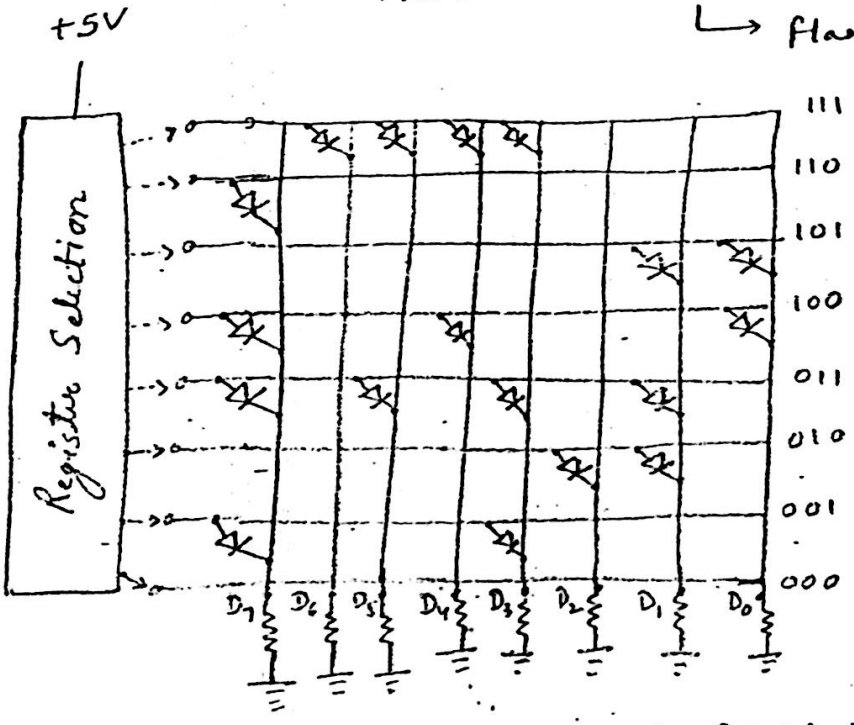
Zero power or NV RAM - They protect data, even in the absence of power, & have fast RAM write speeds with unlimited endurance ex:- Flash memory.

Integrated RAM or IRAM - Here, RAM is already integrated on the motherboard with its own heatsink & heatpipe. This IRAM can be clocked to 1500MHz & we get 50% random access improvement over typical separate RAM module installation.

(2.) Read-only Memory (ROM)-

ROM is nonvolatile memory (retains info. even if power is turned off). This memory is used for programs & data that need not be altered. The bit information is stored which is permanent or semi-permanent.

- Masked ROM
- PROM
- EPROM
- E^2PROM
- Flash



Functional Representation of ROM Memory cell

Concept of ROM can be explained with the diodes arranged in a matrix format. Horizontal lines are connected to vertical lines only through diodes.

Each hor. line can be viewed as register with binary address ranging from 000 to 111. Presence of a diode stores 1 & its absence stores 0. When a reg. is selected, voltage of that line goes high & -lines where diode is connected go high.

ex:- If reg. 111 is selected, data byte 01111000 = 78H can be read at data lines D_7-D_0.

Permanent Memory -

Masked ROM -

i) In Masked ROM, a bit pattern is permanently recorded by the masking and metalization process.

ii) It is an expensive and specialized process, but economical for large production quantities.

PROM (Programmable Read-only Memory) -

i) This memory has nichrome or polysilicon wires arranged in matrix, which can be viewed as diodes or fuses.

ii) This memory can be programmed by the user with a special PROM programmer that selectively burns the fuses acc. to the bit pattern to be stored. This is known as "burning the PROM". & information stored is permanent.

Erasable Memory -

EPROM (Erasable Programable ROM) -

i) This memory stores a bit by charging the gate of FET.

ii) All the information can be erased by exposing the chip to ultraviolet light through its quartz window, and the chip can be reprogrammed.

iii) This chip can be reused many times, so it is suitable for product development.

iv) EPROM must be taken out of the circuit to erase it & erasing process takes 15 to 20 minutes.

EEPROM (Electrically Erasable PROM) -

i) It is functionally similar to EPROM but information can be altered by using electrical signals at the register level rather than erasing all the information.

ii) This has an advantage in field & remote control appn.

This chip can be updated from a central comp. by using a remote link.

iii) This memory includes a Chip Erase mode, ∴, entire chip can be erased in 10 mins. vs 15 to 20 min to erase EPROM.

iv) This memory is expensive compared to EPROM or flash.

Flash Memory - (fast reprogramming capability)

i) It is electrically erasable and reprogrammable.

ii) Unlike EEPROM, it uses one-transistor memory cell resulting in high packing density, lower cost & higher reliability.

iii) It uses floating gate tx. - which contains two gates - a control gate & a floating gate.

iv) Here, it is possible to read the content of a single cell but it is not possible to write to a single cell. It allows to write an entire block of cells.

v) Before writing, previous contents are erased. Flash memory is not byte by byte erasable like e²PROMs but it must be erased in its entirety or at the block level.

vi) Flash memory has a no. of blocks. ex:- 128 MB chip has 8192 blocks of 16kB each.

vii) Power consumption of flash memory is very small.

viii) Flash memory is available in 2 forms —

- flash cards
  - 128 MB - 64GB
  - used in digital camera & MP3 player.
- flash drives
  - 512 MB - 64GB
  - called pendrive, flash stick or thumb drive.
  - mobile phones, digital cameras.

128 MB-64GB

1 3 28 X4 512

2. Storage memory -

i) The memory is used to store programs and results after the execution / completion of program.

ii) Information stored is non-volatile.

iii) CPU cannot directly process programs stored in these devices ; program needs to be copied into R/W memory first.

iv) Size of storage memory is unlimited, when disk is full, next one can be used.

v) Storage memory is divided into 2 groups -

Secondary storage (similar to what we put on a shelf in our study) & backup storage. (similar to what you store in your attic)

vi) Includes devices such as disks, magnetic tapes, charged-coupled devices.

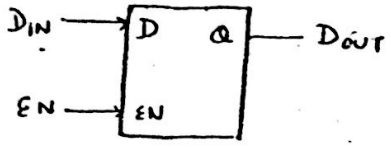
vii) Primary features are high capacity, low cost & slow access.

viii) Access to information stored in disk is semirandom whereas devices such as tapes are serial i.e. it (info. from b/w) can be accessed after running the half tape.

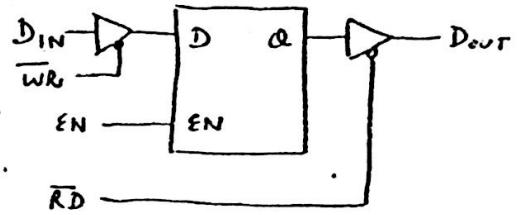
Memory

The R/W Memory is made of registers, and each register has a group of FFs; these FFs are called memory cells. The no. of bits stored in a register is called a memory word.

FF or latch as a storage element -



Basic Latch



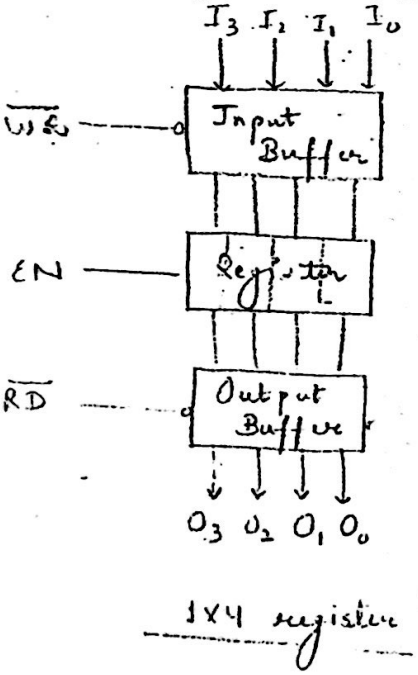
Latch with two tri-state buffers

⇓

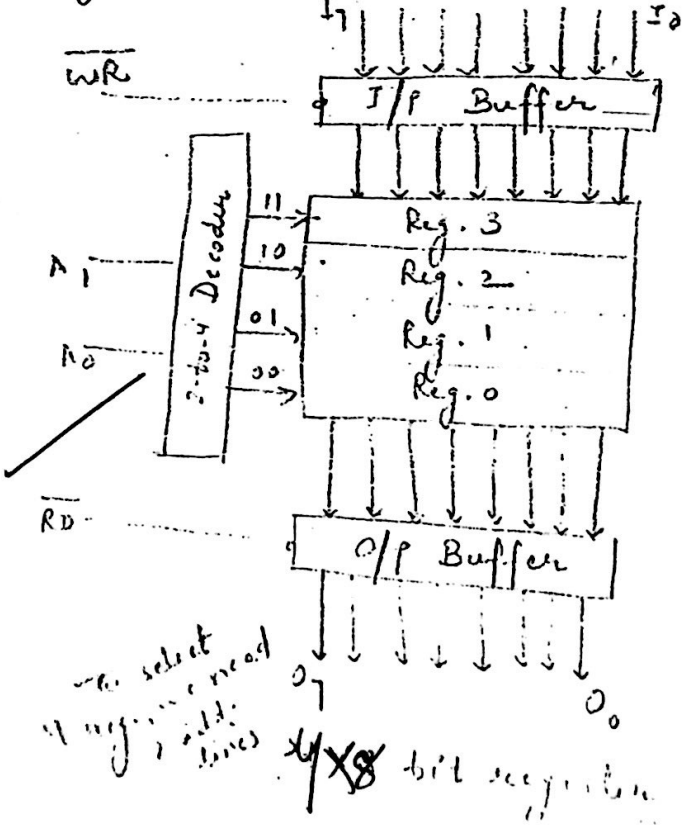
To avoid unintentional change in i/p to control availability of o/p, we use tri state buffers on the latch.

Latch which can store one binary bit is called a "memory cell".

ex:- A register which has 4 cells is specified as 4-bit or 1X4 bit ⇒ one register with 4 cells or 4 I/O lines.

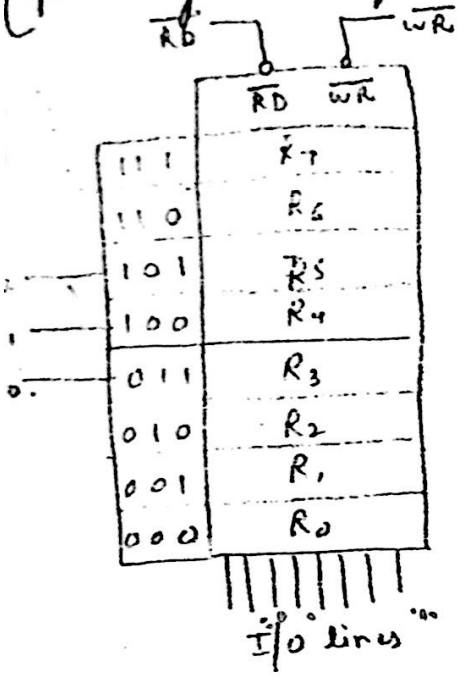


1X4 register



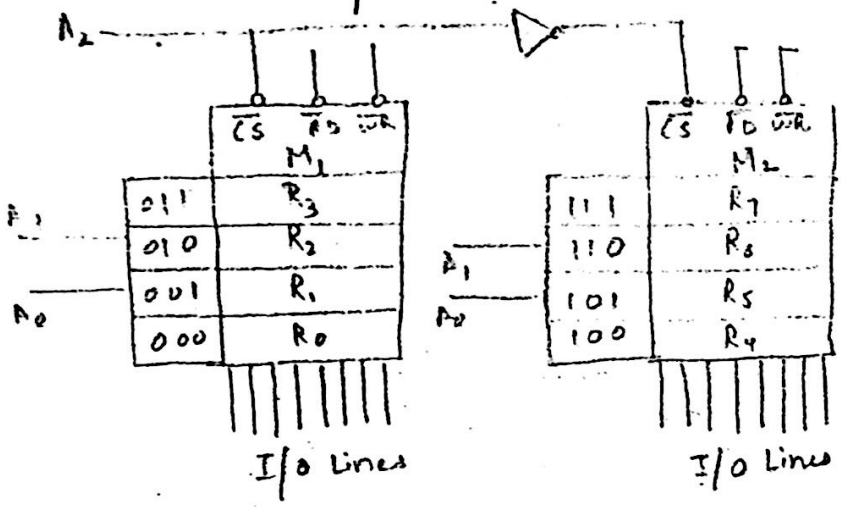
4X8 bit register

If we have a total of 8 reg. we need three address lines. (for 16 reg., we require 4)



8×8 chip

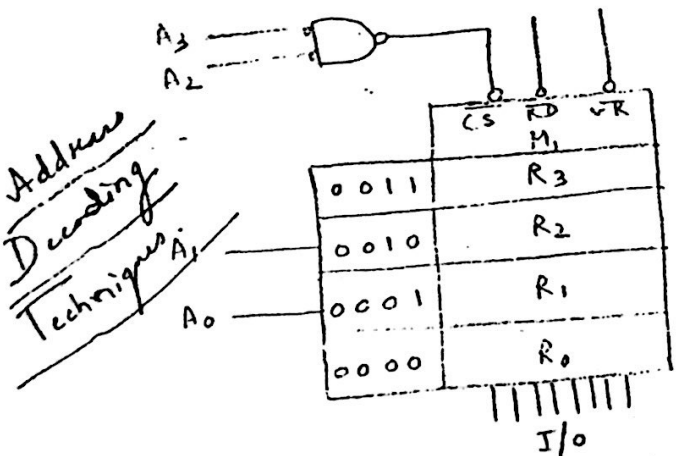
⇒ But if two chips of 4 reg. each are available, we need 3 add. lines, one to select b/w 2 chips.



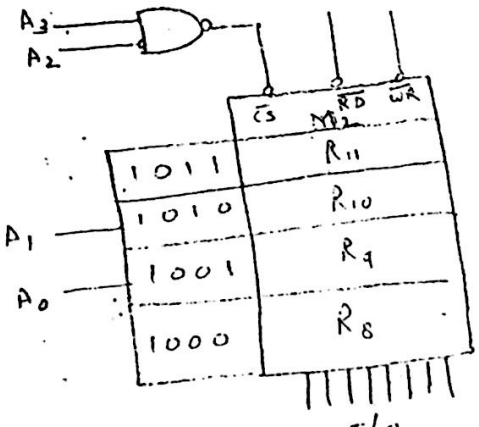
8×8 chip using two 4×8 chips (2 add. lines + 1 line for CS)

Suppose, we have available four address lines & 2 memory chips with 4 registers as above. But 4 add. lines can identify 16 registers; however we need only 3 to identify 8 registers.

⇒ In this case, we can use the fourth line for CS also, as shown below.



Address Decoding Techniques



This is called absolute decoding or complete decoding.

⇒ Another option is to leave the fourth line as don't care. This is called partial address decoding.

We conclude the following:-

A memory chip requires address lines to identify a memory register.

"No. of add. lines required is determined by no. of registers in a chip."

No. of registers in a chip = 2^n

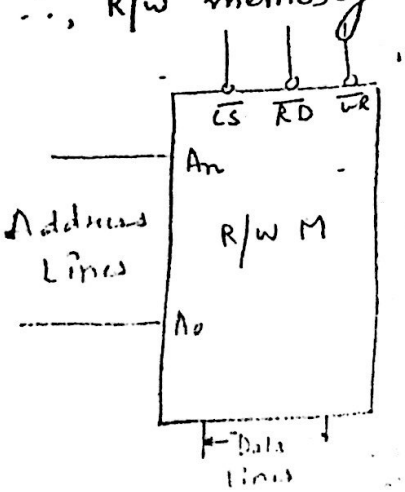
where, n = no. of add. lines.

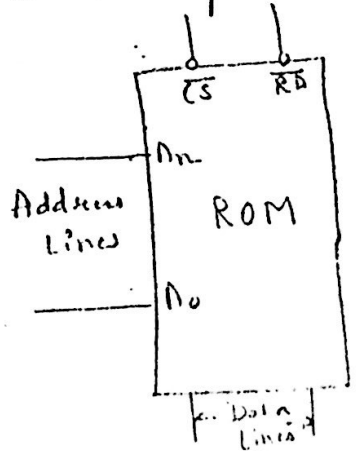
2.) A memory chip requires a Chip select ($\overline{CS}$) signal to enable the chip.
Add. lines from step 1 can be connected to $\overline{CS}$ through an interfacing logic.

3.) Address lines connected to $\overline{CS}$ selects the chip & add. lines connected to add. lines of memory chip select the register.
∴, memory address of a reg. is determined by all the address lines.

4.) Control signal $\overline{RD}$ enables output buffer & data from selected reg. are available on o/p lines.
Control signal $\overline{WR}$ enables Input buffer & data on i/p lines are written into memory cells.
µp can use $\overline{MEMR}$ & $\overline{MEMW}$ signals to enable buffers.

⇒ ∴, R/W memory & ROM can be represented as shown?:-





Memory Map and Addresses :-

In an 8-bit µp. (8085),

Address lines = 16

∴, these lines are capable of identifying 2^{16} memory registers

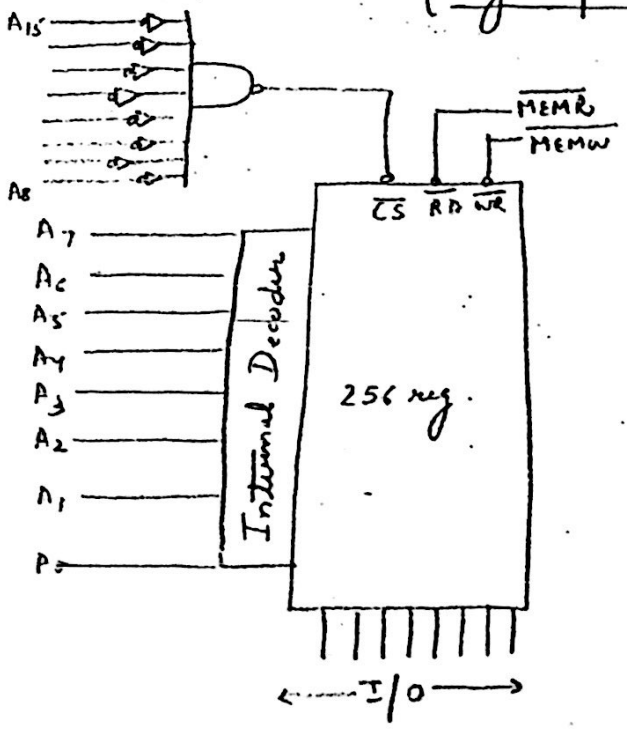
= 65536 " "

Each register have a 16-bit address, that can range from 0000H to FFFFH in hexadecimal.

Memory map :- It is a pictorial representation in which memory devices are located in the entire range of addresses

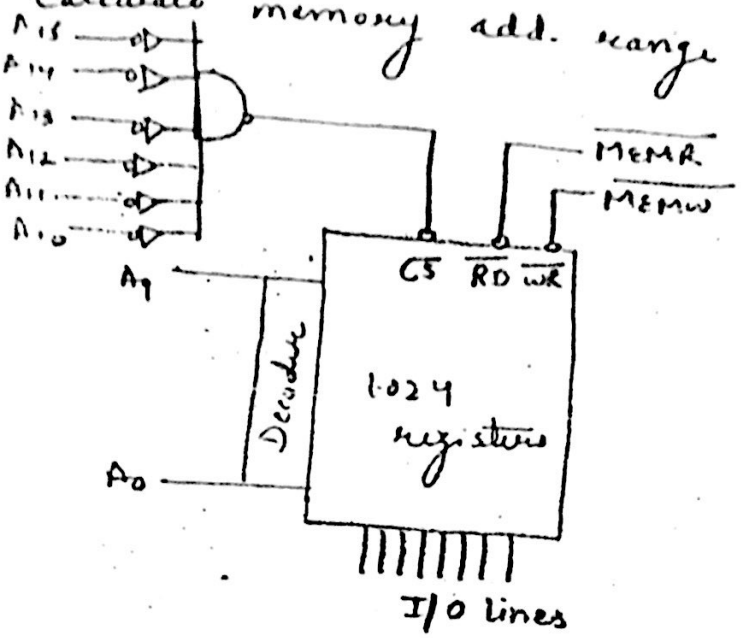
Memory addresses :- It provide the locations of various memory devices in the sys.

Example :- We have a memory chip with 256 registers. ∴, it requires 8 add. lines as $2^8 = 256$. Remaining 8 add. lines can be assigned fixed logic to generate a constant no. (Logic for $\overline{CS}$)



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	
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	= 0000H
$\overline{CS}$ logic	1 1 1 1 1 1 1 1	= 00FFH

Ques. Calculate memory add. range of 1K (1024 × 8) memory.



$2^x = 1024$

$\log 2^x = \log 1024$

$x = \frac{\log 1024}{\log 2} = 10$

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	
0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	= 0000H
	1 1 1 1 1 1 1 1 1 1	= 03FFH

Memory word size –

ex:- 1024 × 4 has 1024 reg. with 4 I/O lines.

Ques. How many memory chips are needed to design 8k-byte memory if the available memory chip is of size 1024 × 1.

No. of chips = $\frac{8 \times 1024 \times 8}{1024 \times 1} = 64$

0011

3

KB.

10

Memory Interfacing

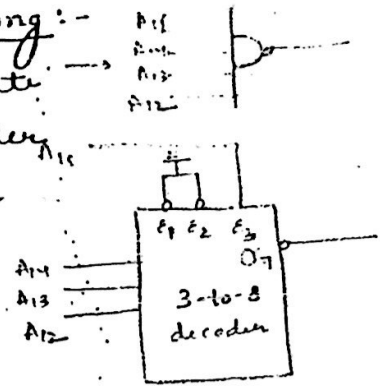
Address Decoding -

Identify address lines such that few address lines are connected to the memory chip & remaining are decoded to obtain $\overline{CS}$ logic. Not all the remaining lines may be decoded.

There are two methods of decoding :-

i) one by using a NAND gate

ii) other by using a decoder



Address decoding techniques -

1. Fully address decoding / Absolute / complete
2. Partial address decoding / Linear add. decoding

Fully address decoding

i) Here, all unused lines are decoded to generate chip select. i.e. every unused line have a particular value either 1 or 0.

ii) Each location has fixed address.

iii) Size of memory is not reduced. H/w is complicated & is expensive.

iv) future expansion of memory is easy.

Partial address decoding

i) All unused lines are not decoded to generate $\overline{CS}$. The value of undecoded address bit is don't care.

ii) Partial decoding generates fold-back memory i.e. each register will have more than 1 address.

iii) Job gets done away with minimum h/w.

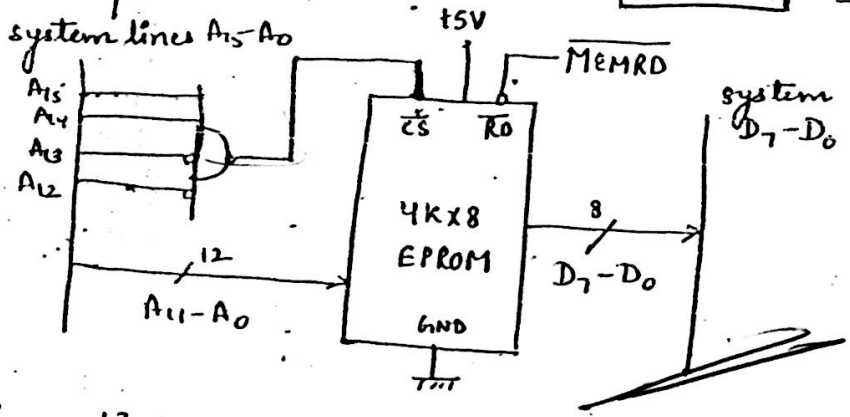
iv) future expansion of memory is difficult & generally requires redesign of address decoder.

Ques. 1. i) Connect 4kB EPROM with 8085 starting from address C000H.

ii) Connect 8kB SRAM with system lines of 8085 performing partial address decoding i.e. use A_{15} for generating $\overline{CS}$. Starting add. = 8000H

i) $4kB = 2^2 \times 2^{10} = 2^{12}$

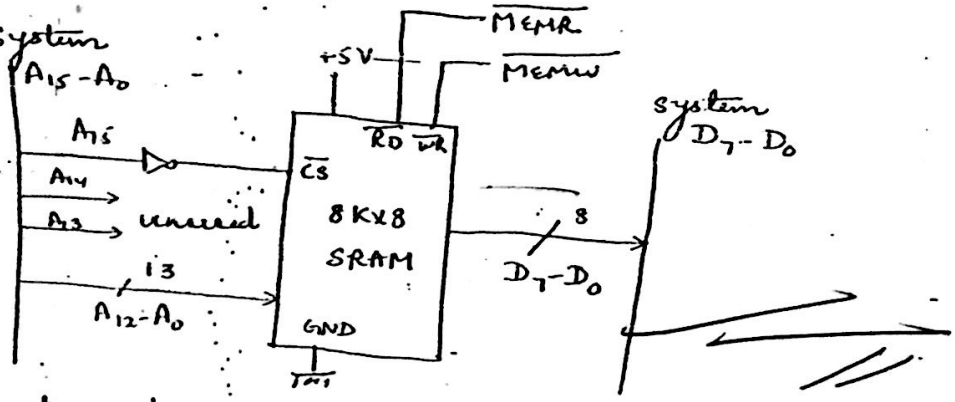
A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	= C000H ✓
1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	= DFFFH ✓



ii) $8kB = 2^3 \times 2^{10} = 2^{13}$

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	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	= 8000H
1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	= 9FFFH

Not used for decoding



∵, A_{14} & A_{13} are not used, it is partial decoding. ∵, add. lines in partial decoding, $n = 2$, no. of addresses of each memory location will be $2^n = 4$

∵, 8kB will cover space of 4 x 8kB = 32kB memory.

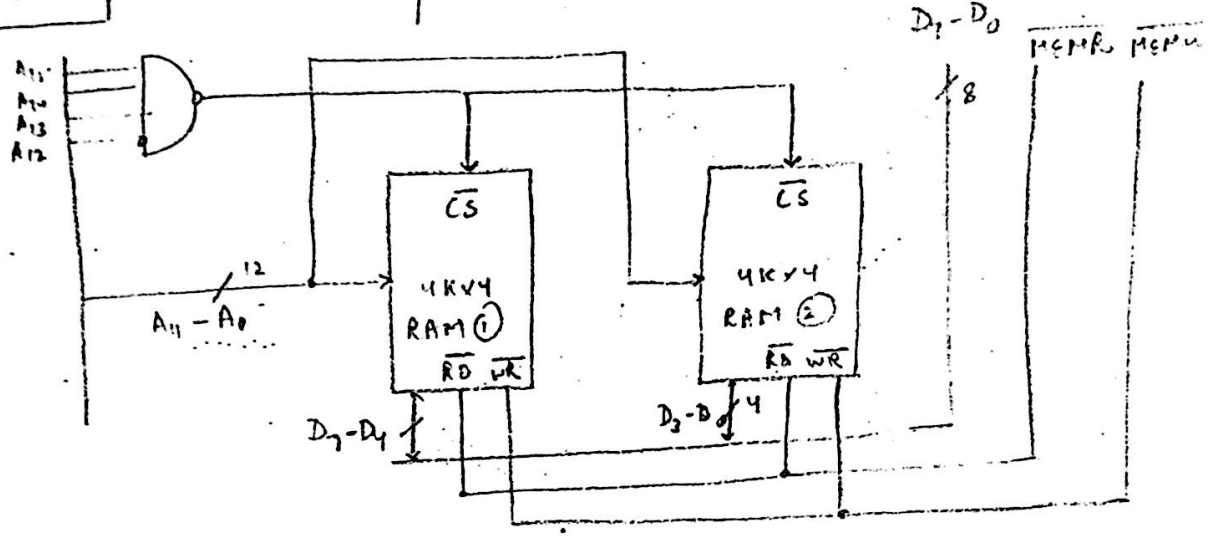
Ques 2. Connect 4k nibble RAM with system lines of 8085. Store the address from E000H.

$4k \times 4 = 4 \times 2^{10} \times 4$ → 4 I/O Lines

2^{12}

∵ there are 4 I/O lines, we have to connect two such ICs in parallel.

	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	
2 ICs 4K×4 in parallel = 4K×8 RAM	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	= E000H
	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	= EFFFH



Type 2 :- Memory mapping of more than one IC of same capacity:- While interfacing memory of different types, address is always assigned in the following sequence -

- ROM
- EPROM
- Expansion Socket
- RAM

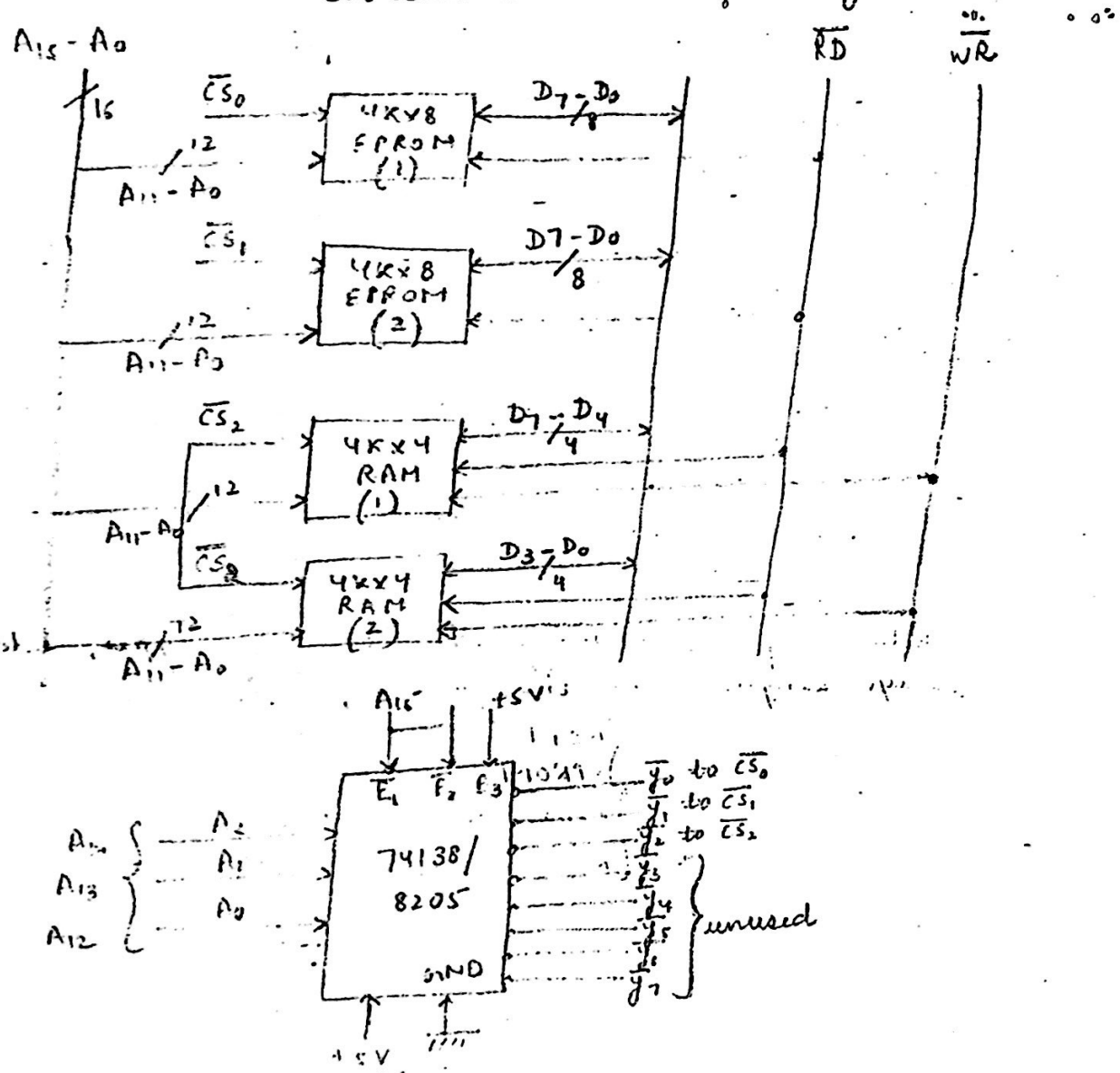
Ques 3. Connect 8k byte EPROM & 4k byte RAM with 8085. Memory ICs available are 4K×8 EPROM & 4K×4 RAM.

Total EPROM = 8k×8 ; Available = 4K×8 } connect two in series

Total RAM = 4K×8 ; Available = 4K×4 } two in parallel.

		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
4K x 8 EPROM (1)	0000H	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
	0FFFH	0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
4K x 8 EPROM (2)	1000H	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
	1FFFH	0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
4K x 4 2 RAM in parallel = 4K x 8	2000H	0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
	2FFFH	0 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1

for $\overline{CS}$ — I/P to 3:8 decoder — $A_{11} - A_0$ to each memory IC



Que.4. Connect 4K byte EPROM & 2K byte RAM with 8085. Memory IC available are 2K byte EPROM & 2KX4 RAM.
2 in series 2 in parallel

Que.5. Draw address decoding table to generate $\overline{CS}$ of the following memory:-

a.) 2 IC's of (2kx4) ROM - 2 in II
b) 4 ICs of (2kx4) RAM - 2 in series & 2 in II with them
c.) 2kx8 expansion socket - 1.

Type 3:- Memory mapping of different Capacity IC's without foldback.

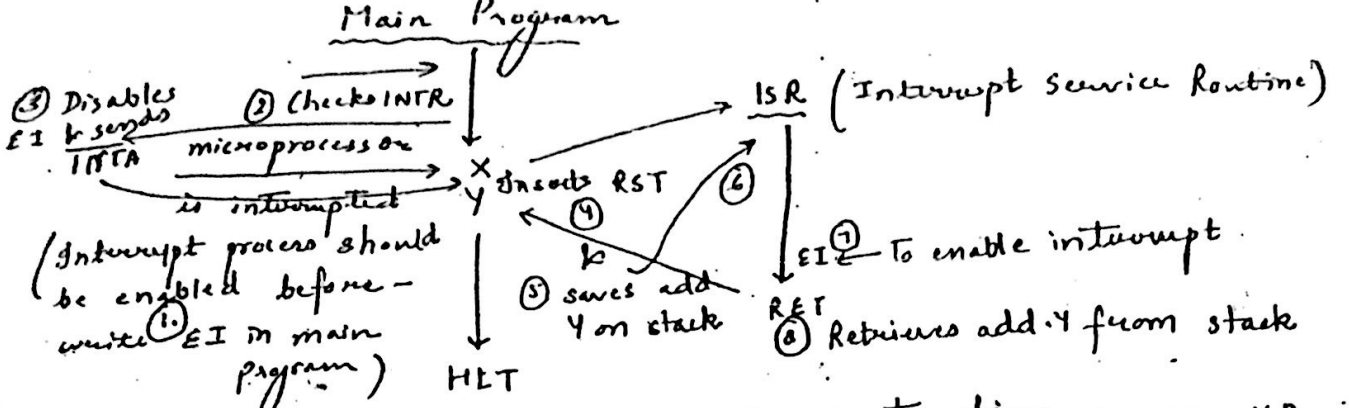
To perform memory mapping without fold back, decoder is designed acc. to smallest capacity memory IC to be connected.
If decoder is designed for IC with capacity x then for 2x capacity, two decoder o/p's will be required.
Here, memory space is not lost & each memory location has only one address but more no. of decoder o/ps are required.

Que.6. Connect following with 8085 -
i) 1kx8 ROM ii) 2KX8 EPROM iii) 1KX8 RAM

		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
1KX8 ROM	0000H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	03FFH	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
2KX8 EPROM	0400H	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
	0BFFH	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1
1KX8 RAM	0C00H	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
	0FFFH	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
		To $\overline{EN}$ of decoder				To decoder		A_9 - A_0 of 1k IC									

Interrupts of 8085

When microprocessor executes any program, it is called main program. In b/w the execution of main program, if µp is interrupted (at X), then it will branch from main program instruction to subprogram called as Interrupt Service Routine.



µp executes this ISR & when RET instruction comes, µp returns from ISR to main program i.e. Y (from where it has left

There are two types of interrupts :-

Software Interrupts

If µp is interrupted by giving instruction in the main program, then it is called as S/w Interrupt.

ex:- CALL 16-bit address

RSTn (0-7)

Hardware Interrupts (5)

If µp is interrupted by giving signal on h/w interrupt pins, it is called h/w interrupt.

ex:- TRAP

RST 5.5, 6.5, 7.5

INTR

The 8085 Interrupt process can be described in the following 8 steps :- (compare with telephone with a blinking light)

(1.) Interrupt process should be enabled by writing EI instruction in the main program.

(Keeping receiver on hook)

* Already discussed EI & DI.

EI (Enable Interrupt) -

* 1 byte x^n
* sets Interrupt enable ff to enables interrupt process.
* system reset or interrupt disables interrupt process.

DI (Disable interrupt) -

* 1 byte x^n
* resets IE ff to disables interrupt process.
* It should be included if interrupt from an outside source cannot be tolerated. (except TRAP)

(2.) When µp is executing a program, it checks the INTR line during execution of each instruction. (glancing the light)

(3.) If INTR is high & EI is enabled, µp completes current instruction, disable IE ff & sends $\overline{INTA}$. (This means no more calls can be received if user is busy)

(4.) $\overline{INTA}$ inserts (roommate asks to check if windows open) RSTn or Call through external h/w.

(5.) When µp receives RST, it saves memory address of next x^n on to the stack. (Inserts a bookmark to check window)

(6.) Program is transferred to call location i.e. task is written as a subroutine at the specified location. This subroutine is called Interrupt Service Routine (ISR).

(7.) ISR should include EI to enable the interrupt again. (putting the receiver back on the hook).

(8.) At the end of subroutine, RET retrieves memory address from stack where program was interrupted & continues the execution. (you starts reading book)

Software Interrupts -

(1.) CALL 16-bit address (already discussed)

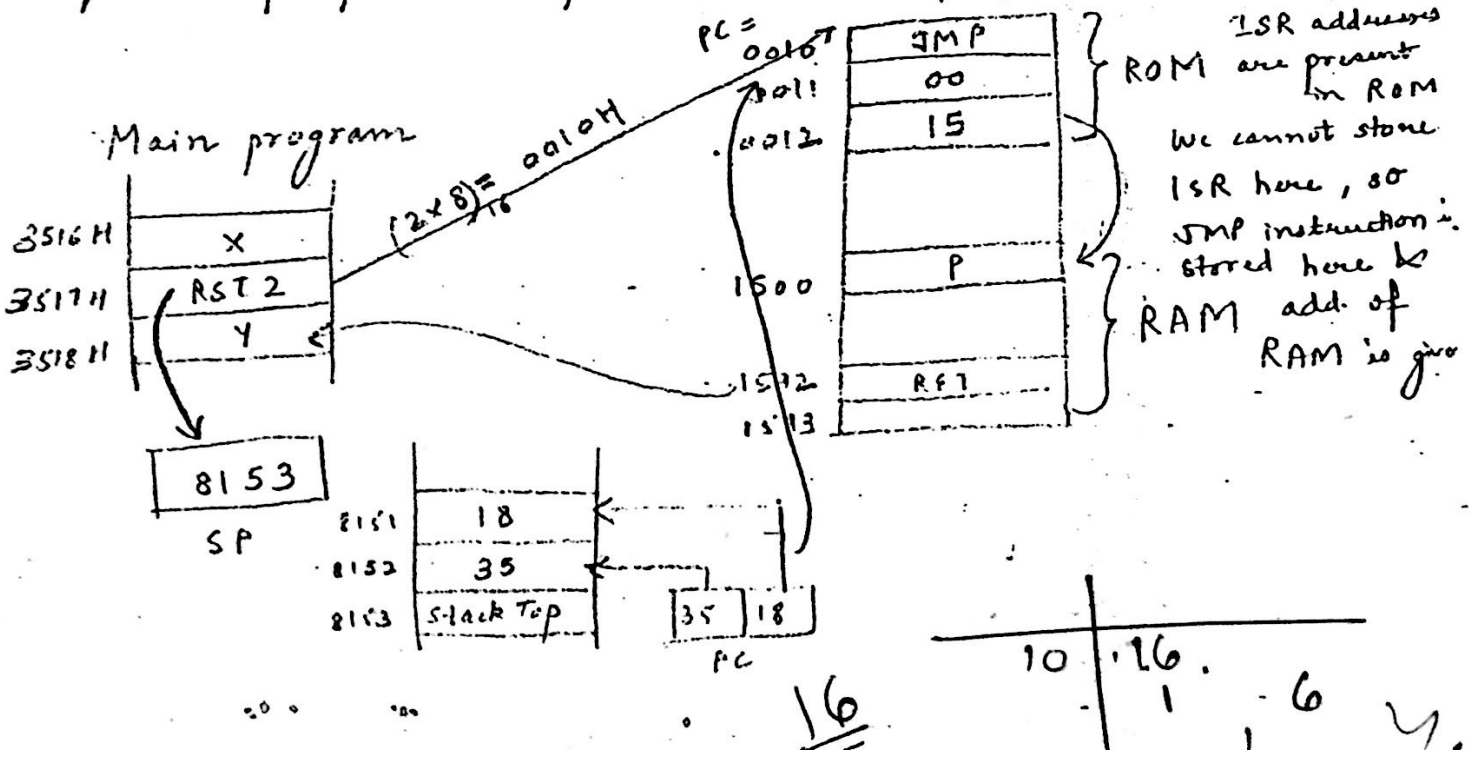
(2.) RSTn (Restart Instructions) -

* 8085 instruction set includes eight RST instructions i.e. n = 0 to 7.
* These are 1-byte Call instructions that transfer the program execution to a specific location on page 00H.

Address = (8 × n) H

Mnemonics	Binary code	Hex code	Call location
RST 0	1 1 0 0 0 1 1 1	C7	0×8 = 0000 H
RST 1	1 1 0 0 1 1 1 1	CF	1×8 = 8_{10} = 0008 H
RST 2	1 1 0 1 0 1 1 1	D7	2×8 = 16_{10} = 0010 H
RST 3	1 1 0 1 1 1 1 1	DF	3×8 = 24_{10} = 0018 H
RST 4	1 1 1 0 0 1 1 1	E7	4×8 = 32_{10} = 0020
RST 5	1 1 1 0 1 1 1 1	EF	5×8 = 40_{10} = 0028
RST 6	1 1 1 1 0 1 1 1	F7	6×8 = 48_{10} = 0030
RST 7	1 1 1 1 1 1 1 1	FF	7×8 = 56_{10} = 0038

Sequence of operation of RST 2 - Interrupt Service Routine



10 | 16
1 | 6

16

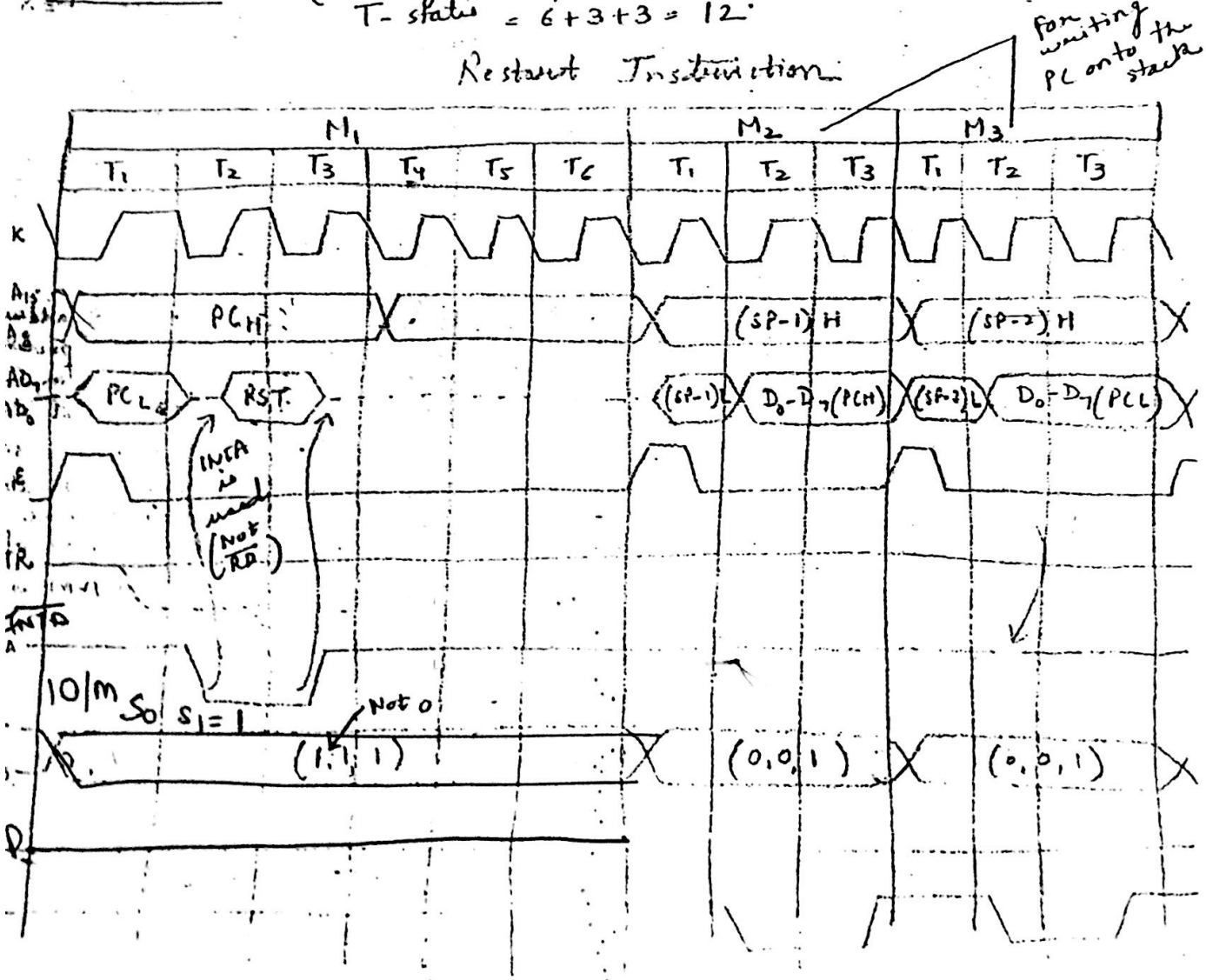
Difference b/w CALL, JMP & RSTn instructions :-

JMP 16-bit add.	CALL 16-bit add.	RSTn
i) 3-byte instruction	i) 3-byte instruction	i) 1-byte instruction
ii) Add. of next instruction is not saved from PC to stack mem.	ii) 16-bit address of next ins. is saved from PC to stack.	ii) 16-bit add. of next inst. is saved from P.C to stack.
iii) 16-bit branching add. is given alongwith the inst. & this add. can be from 0000H to FFFFH	iii) 16-bit branching add. is given alongwith 'x' & this add. can be from 0000H to FFFFH.	iii) Branching ISR add. is fixed = (8×n)H. They are hence called as vectored interrupt.

Timing diagram of Interrupt ack. M/c cycle & execution of RST - (3 M/c cycles = op. fetch with 6 T-states + mem. w + mem. w)

T-states = 6+3+3 = 12

Restart Instruction



Hardware Interrupt -

i) 8085 has 5 h/w interrupts :- INTR, TRAP, RST 7.5, RST 6.5, RST 5.5

ii) RSTs & TRAP are automatically transferred (vectored) to specific locations on memory page 00H without any external hardware. They do not require INTA signal on an i/p port, necessary h/w is already implemented inside the 8085. (TRAP: non-maskable)

iii)

		Call locations
TRAP		0024H
RST 7.5	⟶	003CH
RST 6.5	⟶	0034H
RST 5.5	⟶	002CH

iv) Priority order:

HOLD (used for DMA) > TRAP > RST 7.5 > RST 6.5 > RST 5.5 > INTR

.....

(1.) TRAP :- It need not be enabled & cannot be disabled.

i) It is a non-maskable interrupt (NMI).

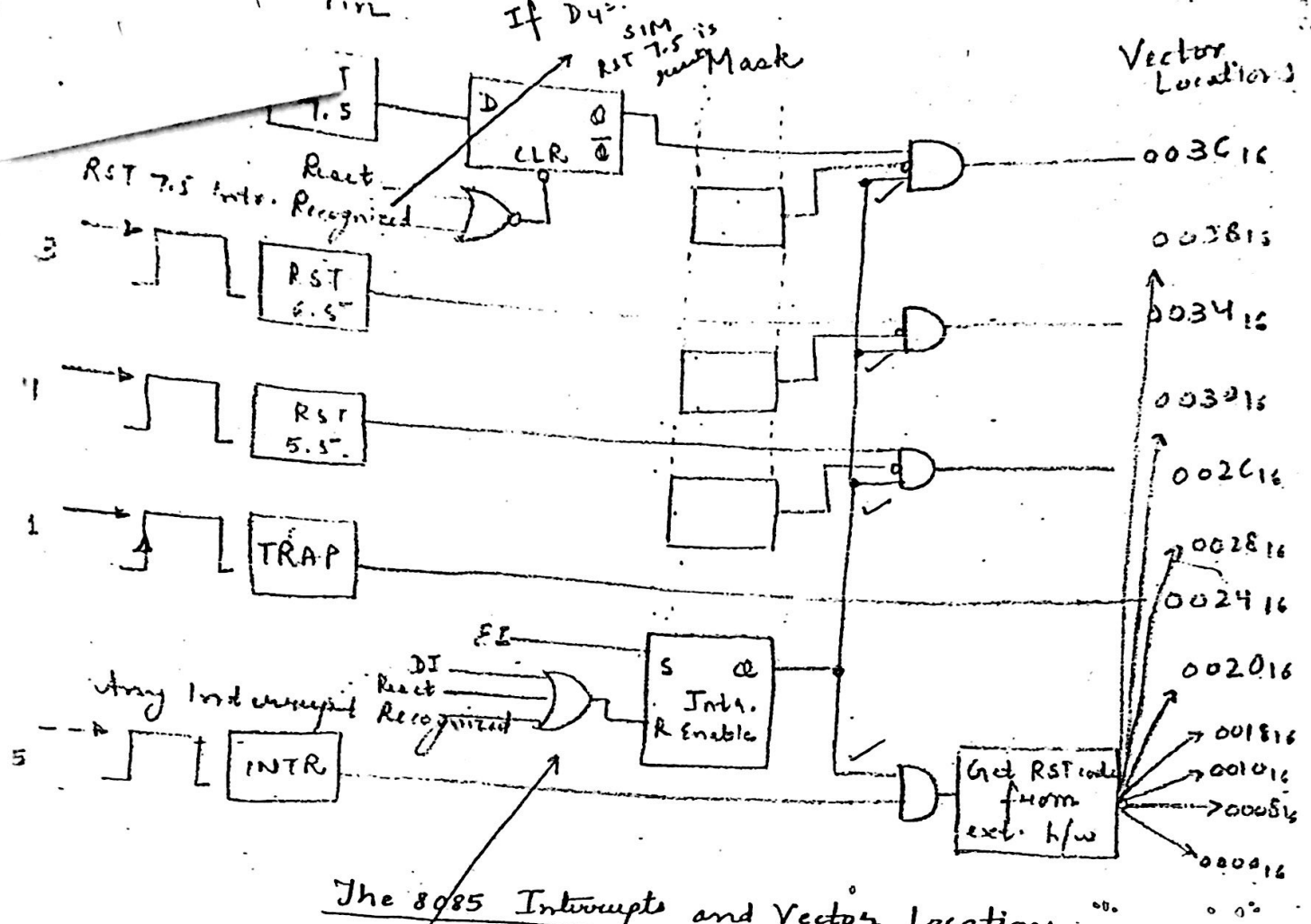
ii) It has highest priority among all the interrupt signals

iii) It is level & edge-sensitive (it should go high & remain high till it is acknowledged). It cannot be acknowledged again until it makes transition from high to low to high.

iv) When this interrupt is triggered, program control is transferred to 0024H without any external h/w & Interrupt enable instruction.

v) TRAP is generally used for such critical events as - power failure & emergency shut-off.

The h/w interrupts of 8085 are shown next :-



The 8085 Interrupts and Vector Locations

(2) RST 7.5, 6.5 & 5.5 -

) These are maskable interrupts & can be enabled by two instructions - EI & SIM.

) The entire interrupt process can be disabled by resetting Interrupt enable ff in one of the 3 ways:-

- → Instruction DI
- → System Reset
- → Recognition of an Interrupt Request

RST 7.5 - It is rise edge triggered & can be triggered with a short pulse. This is stored internally by DFF.

RST 6.5 & 5.5 - They are level triggered, i.e. level should be on until µp completes execution of current instruction. If µp is

by external h/w.

Pending Interrupts -

There are several interrupt lines & when one request is being served, other interrupt requests may occur & remain pending.

To sense pending interrupts RIM instruction is used.

Ques. Assume µp is completing RST 7.5 request, check to see if RST 6.5 is pending. If it is pending, enable RST 6.5 w/t affecting any other interrupts, otherwise return to the main program.

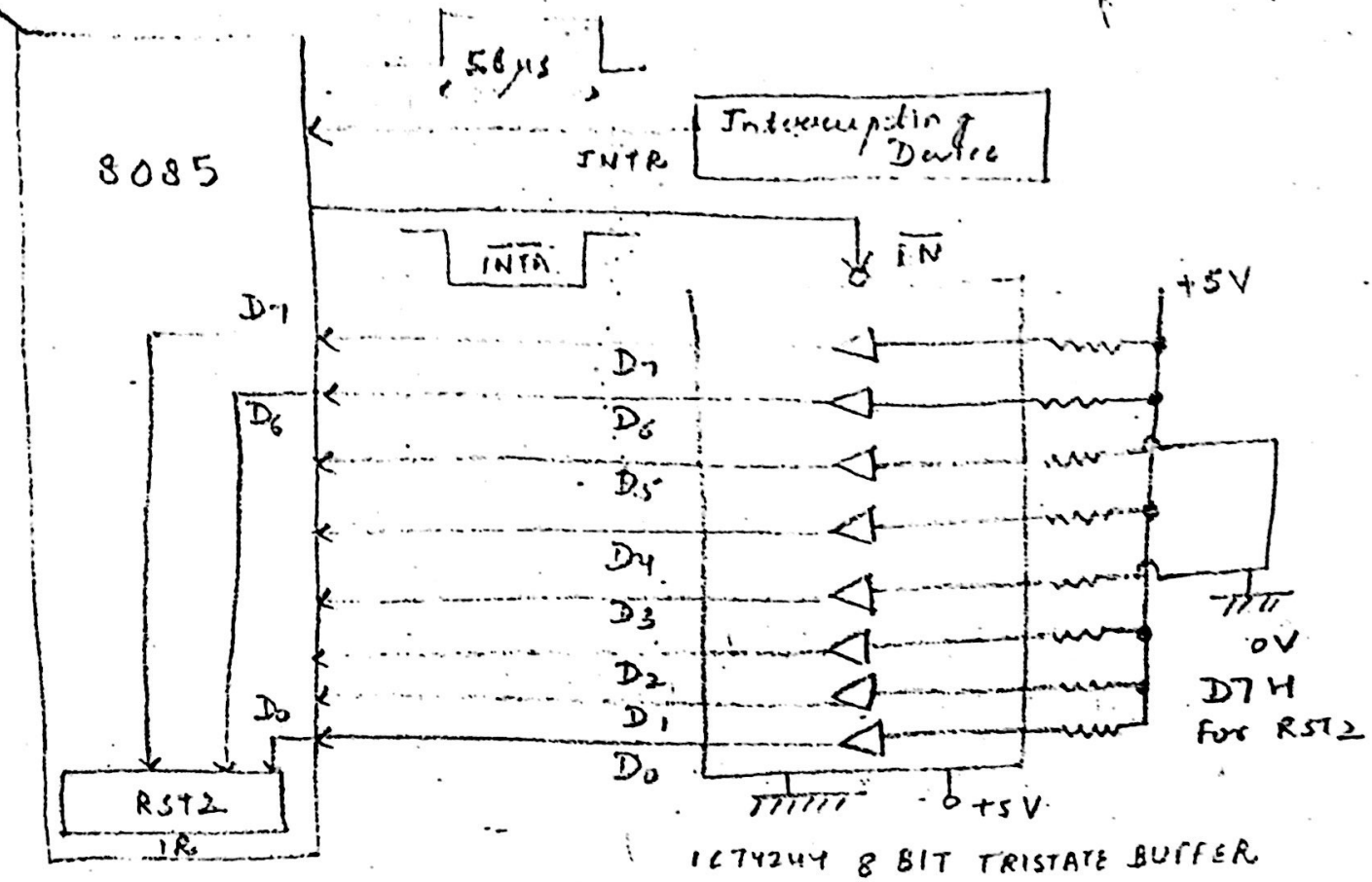
```
.....
        RIM         ; read interrupt mask
        MOV B,A     ; save mask info.
        ANI 20H     ; check whether RST 6.5 is pending
        JNZ NEXT    ; if cond. is true, intr. is pending
        EI
        RET
NEXT:   MOV A,B     ; get RIM Mask
        ANI 0DH     ; Enable RST 6.5
        ORI 08H     ; Enable SIM by setting D3 = 1
        SIM
        JMP SERV
```

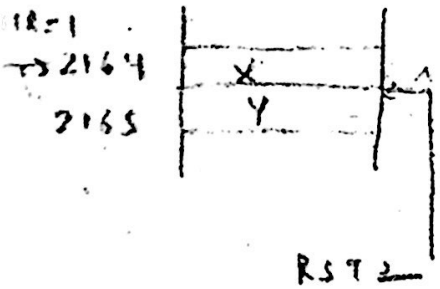
Sequence of execution of h/w interrupts -

i) Sequence of execution of TRAP, RST 7.5, 6.5, 5.5 -

same as RSTn (as explained earlier)

ii) Sequence of execution of h/w interrupt INTR -





→ µp is currently executing X.

→ Interrupting device sends '1' on INTR. Pulse width should be 5.8µs (as CALL needs 18 clock cycles)

→ When µp receives INTR=1, it gives $\overline{INTA}=0$.

→ When $\overline{INTA}=0$, $\overline{EN}=0$, tristate buffer is enabled, opcode of RST2 = D7H is transferred to D7-D0 & finally µp transfers D7H in IR.

→ µp will execute RST2 before 'Y' from location 8X2 = 10D = 0010D & will save PC = 2165 on stack.

Instructions controlling h/w interrupts of Microprocessor -

1. **EI : Enable Interrupts** - EI (none)

i) It is a 1-byte Instruction.

ii) This instruction sets Interrupt Enable (IE) ff & all interrupts are enabled.

iii) After a system reset or ack. of an interrupt, IE ff is reset, thus disabling the interrupts. This instruction is necessary to reenable the interrupts.

2. **DI : Disable Interrupts** - DI (none)

i) It is a 1-byte instruction.

ii) This instruction resets IE ff & disables all interrupts of 8085 except TRAP.

iii) This instruction is commonly used when execution of a code sequence cannot be interrupted. for ex:- in critical time delays.

3. **SIM : Set Interrupt Mask** -

This is a 1-byte instruction & can be used for 3 diff. function

7	6	5	4	3	2	1	0
SOD	SDE	XXX	R7.5	MSE	M7.5	M6.5	M5.5

- RST 7.5 MASK, " 6.5 ", " 5.5 " { 0 = available, 1 = masked
- Mask Set Enable { If 0, bits 0-2 are ignored; If 1, mask is set
- Reset 7.5 : If 1, RST 7.5 FF is reset off
- Ignored
- If 1, bit 7 is o/p to Serial o/p data latch
- Serial o/p data : ignored if bit 6 = 0

irst function is to set mask for RST 7.5, 6.5 & 5.5 interrupts. This instruction reads contents of Acc. & enables & disables interrupts acc. to content of the acc. If bit D3 = 0, D0, D1 & D2 are ignored. If D3 = 1, bits D0, D1 & D2 will be effective. Logic 0 on them will enable interrupts & logic 1 will disable them.

Second function is to reset RST 7.5. If D4 = 1, RST 7.5 is reset. This is used to override RST 7.5 w/t servicing it.

ii) Third function is to implement serial I/O. Bit D6 = 1 enables the serial I/O & bit D7 is used to transmit bits.

4. RIM : Read Interrupt Mask -

This is a 1-byte instruction that can be used for following functions:-

7	6	5	4	3	2	1	0
SID	I7.5	I6.5	I5.5	IE	M7.5	M6.5	M5.5

Interrupt Masks: 1 = Masked

Interrupt Enable Flag: 1 = enabled

Pending Interrupts: 1 = pending

serial i/p data bit

i) To read interrupt masks. This inst. loads the acc with 8-bits indicating current status of interrupt masks.

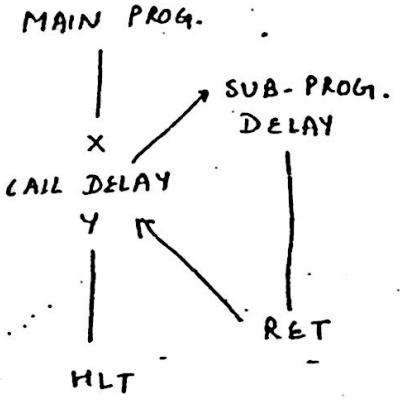
ii) To identify pending interrupts (Bits D4, D5 & D6)

iii) To receive serial data. (D7).

If X & Y are two successive instructions in a program & if a time gap or time delay is required b/w the execution of X & Y, then this time delay can be produced either by - hardware or software.

Here, µp is made to execute a delay subroutine. If delay required is T_d, delay subprogram should be such that time t_d is required by µp to execute that subroutine.

µp executes instruction X, then executes delay subroutine for time t_d & after t_d executes instruction Y.



Delay subroutine using 8-bit reg. -

	T-states required	
MVI A, n	7	4+3
L1: DCR A	4	4
JNZ L1	7/10	7/10

MVI A,n
L1: DCR A
JNZ L1

$T_d = T_1 + T_2 + T_3$

$= 7 + 4n + \{10(n-1) + 7\}$

$= 7 + 4n + (10n - 10 + 7)$

$T_d = 14n + 4$ T-states

If $n = FFH = (255)_{10}$, then $T_d = 14 \times 255 + 4 = 3574$ T-states

For $f_c = 6$ MHz, $f_c/2 = 3$ MHz $\Rightarrow T = \frac{1}{3\text{MHz}} = \boxed{0.33\ \mu\text{sec}}$

∴, Max. delay using 1-8bit reg. = $3574 \times 0.33\ \mu\text{sec}$

$T_{d\,max} = 1179.42\ \mu\text{sec}$

To increase T_d, NOP instructions may be added.

Delay subroutine using register pair -

	LXI B, n	10
executed n-times	LI: DCX B	6
	MOV A, C	4
	ORA B	4
	JNZ LI	7/10

$T_d = T_1 + T_2 + T_3$

$= 10 + 14(n) + \{10(n-1) + 7\}$

$= 24n + 7$ T-states

If $n = (FFFF)_H = (65535)_D$; $T_d = 24(65535) + 7$

$= 1572847$ T-states

For $T = 0.5$ μsec

$T_d = 786423.5$ μsec.

For $T = 0.33$ μsec

$T_d = 1572847 \times 0.33$ μsec.

Ques. Calculate the execution time for following pgm.

XTAL f = 4 MHz (f_c = 2 MHz, T = 0.5 μsec)

	PUSH PSW	12
	MOV A, B	4
	PUSH B	12
	MVI B, 49H → $(73)_D$	7
LI:	NOP	4
	DCR B	4
	JNZ LI	7/10
	POP B	10
	NOP	4
	POP PSW	10
	RET	10

$T_d = (12+4+12+7) + (4+4) \times 73 + (10 \times 72 + 7) + (10+4+10+10) \times 1380$

$T_d = 1380 \times 0.5$ μsec

$T_d = 690$ μsec