Post Midterm

ASHOKA
Date: 10/10/18
Page No.

Status signal :-

	IO/$\overline{M}$	S1	S0	control signals
OF	0	1	1	$\overline{RD} = 0$
MR	0	1	0	—"—
MW	0	0	1	$\overline{WR} = 0$
IOR	1	1	0	$\overline{RD} = 0$
IOW	1	0	1	$\overline{WR} = 0$
INTA	1	1	1	$\overline{INTA} = 0$

select lines

They work acc to combination of status signals.

Reset	Z	X	X
HALT	Z	0	0
HOLD	Z	X	X

$\overline{RD} = \overline{WR} = Z$
$INTA = 1$

Z = Tristate

Timing Diagram :-

1) It is important because it enables us to see how µp executes these instructions
2) which signals are activated at what i/p
3) Graphical Plots.

3 Machine cycles → OF MR MW

- Instruction cycle → Time Required to execute an Instruction is Ins. time / cycle / period.
- Machine cycles → Ins. cycle Must have multiple Machine cycles. Time Required to access Memory or I/O.
  → OF → IOR → IOW
  → MR → MW

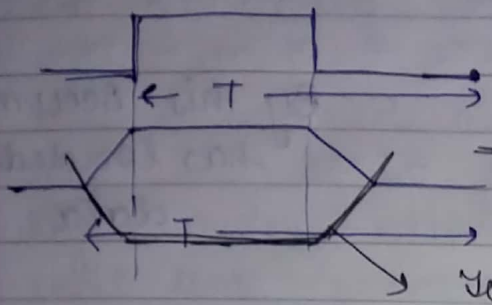
call have max no. of Machine cycles.

ASHOKA
Date:
Page No.

- In Ins. cycle there will be one opcode and Multiple MR, MW, IOR, IOW

  IC = OF + XMR + XMW + XIOR + XIOW

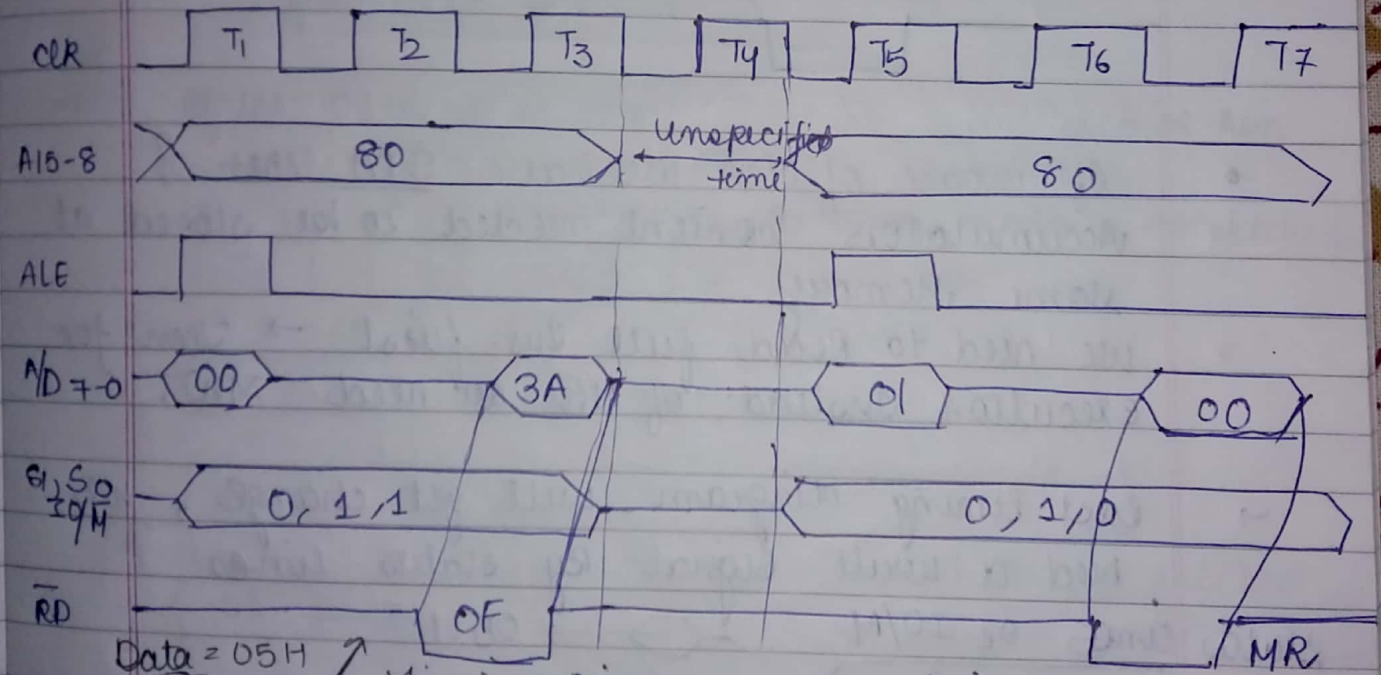
  X can be from 0 to 4/5.

- 

→ In digital way clock look like this

→ But Practically it look like this

→ To double the performance of µP we use both the phases of clock.

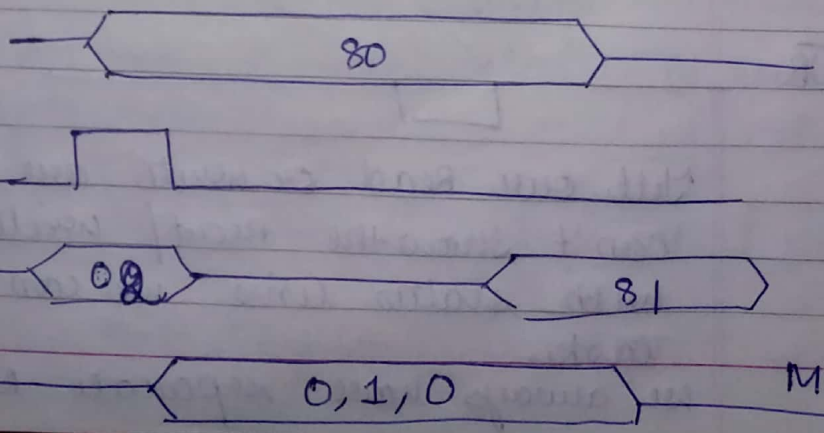


Data = 05H

LDA 8100

8000	3A
8001	00
8002	81

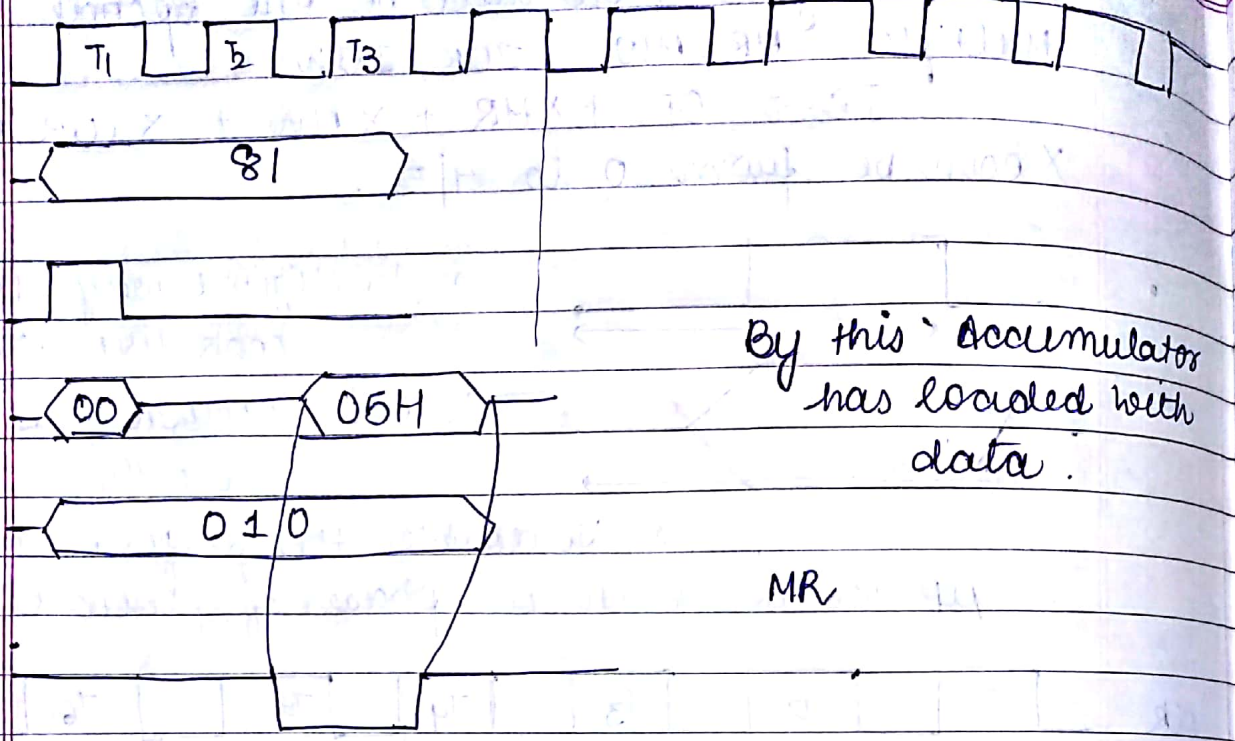
Timing diagram for this



By this Reading of Inst done

ASHOKA
Date:
Page No.

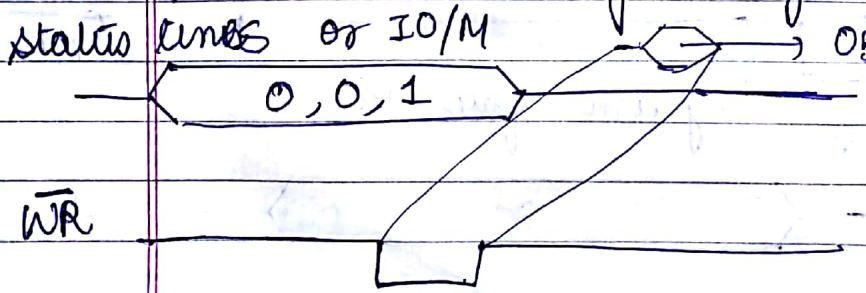
Execution of Information :-



By this Accumulator has loaded with data.

- If incase of LDA we have STA then ? Accumulators content needed to be stored at some Memory.
  We need to Read full Ins. first → Then for execution instead of MR we need MW.

→ Last timing diagram will get change, we need a write signal By status lines.



(data gets written)

Till our Read or write are not activated we can't show the Read/ write operation only with status lines we can't achieve this task.
We always have separate Read/ write Pin.

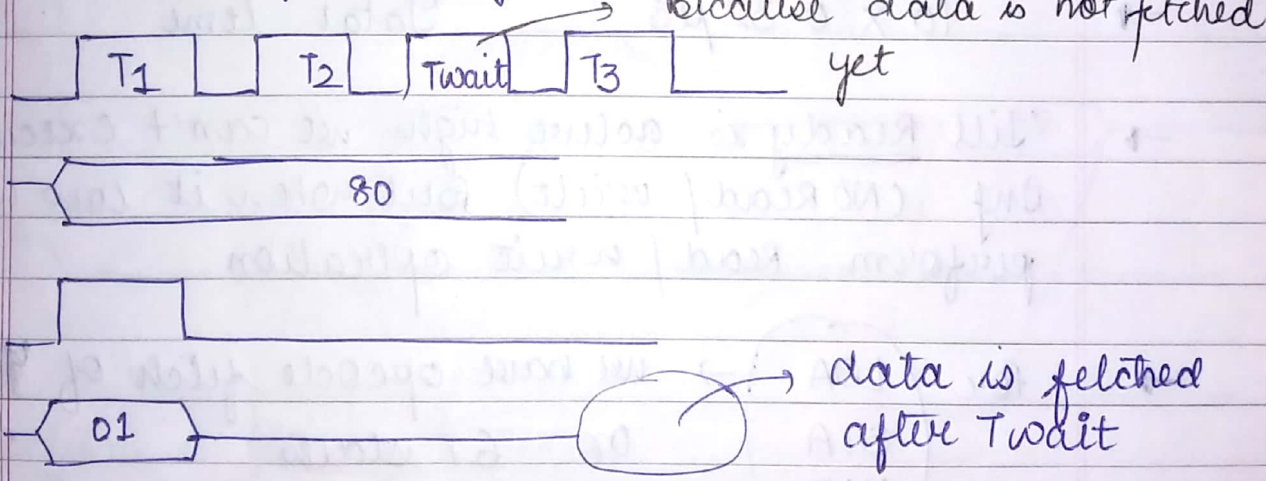
ASHOKA
Date:
Page No.

Can we execute it without select line?

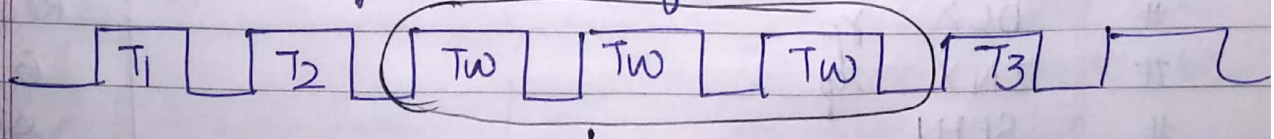
→ select lines set the mode of operation means before T3 we know that we need to perform Read operation we can activate it before some time so lagging will be Removed by this.

Why unspecified time? Because this is the only time of we get delay in Reading higher order address we can comprehend that time in unspecified time. upto T3 if we get some delay it gets compensated in the T4 clock state.

→ If the Memory is very slow in our device then Instead of T3 we get Twait state.



We donot count Twait in our counting clock states let's suppose we get 3 Twait states it becomes a Machine cycle in self.



BIM ↓ cycle

Bus idle Machine cycle

ASHOKA
Date:
Page No.

→ wait state means we are getting delay. It is counted in Total execution time.

$$f = \frac{1}{3\ MHz} = 0.32\ \mu S$$

Total execution time = 11 × 0.32 μS

11 machine cycles

11 → $T_1 - T_4 + T_1 - T_2 + 3Tw + T_3 - T_4$

→ When processor use BIMC?

In DAD rp Instruction — T3 opcode

1 Byte size of Instruction

DAD rp uses 2 BIM cycles. For this

→ OF + BIMC + BIMC

Because of complexity of this Instruction. Processor takes a lot of time to execute it.

→ 10 × 0.32 μS = Total time.

→ Till Ready is active high we can't execute any Inp (No Read/write) But when it low we can perform Read/write operation

→ for (LDA, STA, HLT) → we have opcode fetch of 4 T states or 6 T states

→ conditional or unconditional

Call → All calls Required 6 T states
RET → conditional
DCX rp
INX rp
SPHL
PCHL

They Require 6 T states

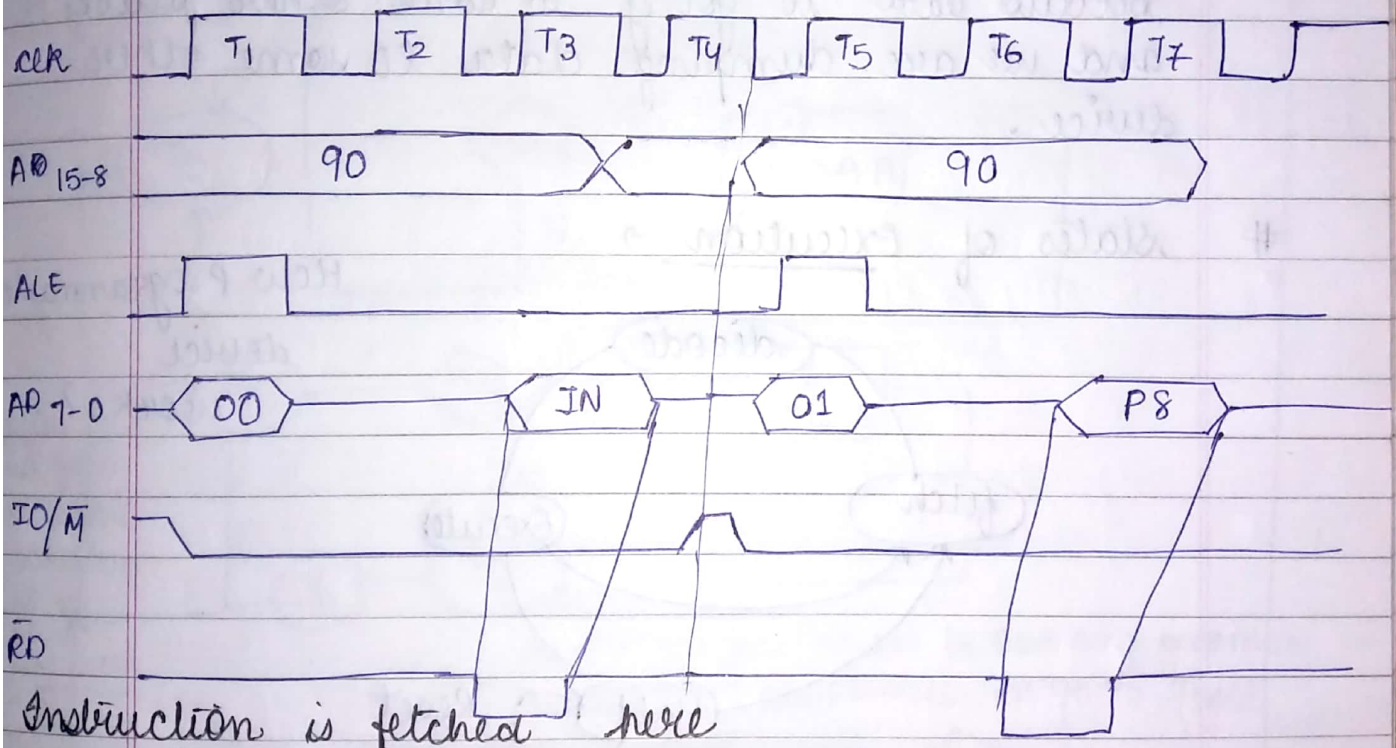
RET → 4 T states
RNZ → 6 T states
CNC → 6 T states (call if not carry)

ASHOKA
Date:
Page No.

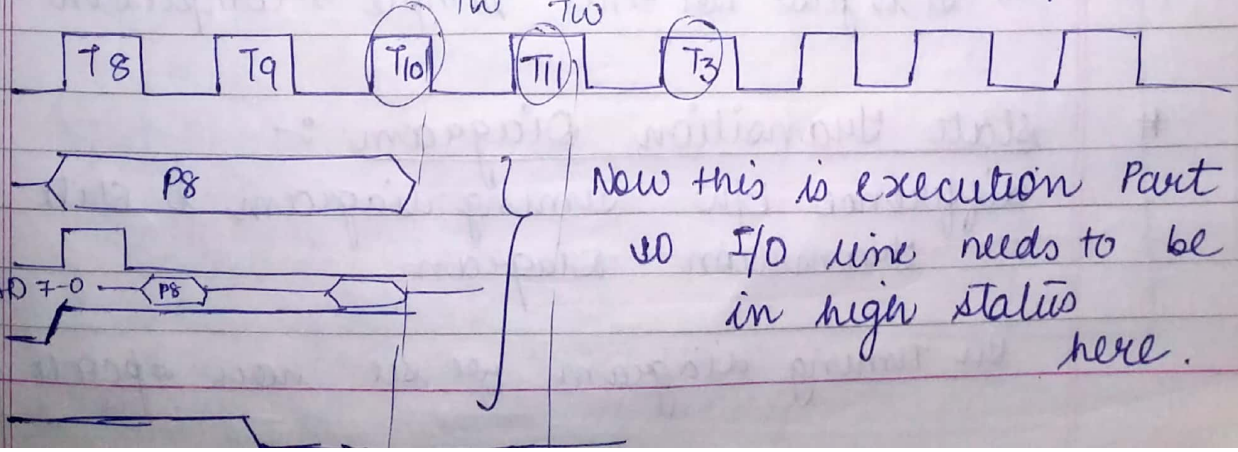
On conditional Return it need to check the condition so it needs more time.

12/10/18 Some instructions in which wait state may happen. 39 P8 / 40 P8
IN P8 / out P8

9000	39
9001	



Instruction is fetched here
P8 → Address where we need to do read operation



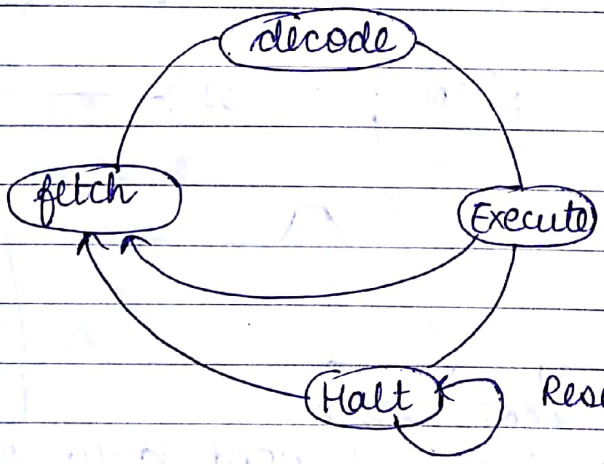
Now this is execution Part so I/O line needs to be in high status here.

If the device is slow and as soon as Ready gets low T3 becomes Tw state. When the ~~device~~ device gets Ready and data is Ready for execution Then T3 state will occur after 2Tw states.
This is ~~8~~ not BIMC because BIMC only when we have 3 Tw states.

What will happen in the case of out ?
For I/O out the 3rd cycle will become I/O write Because data is going to some other place and we are dumping data to some other device.

States of Execution →

How Programmable devici works?



→ It is just not that simple → complex in Nature

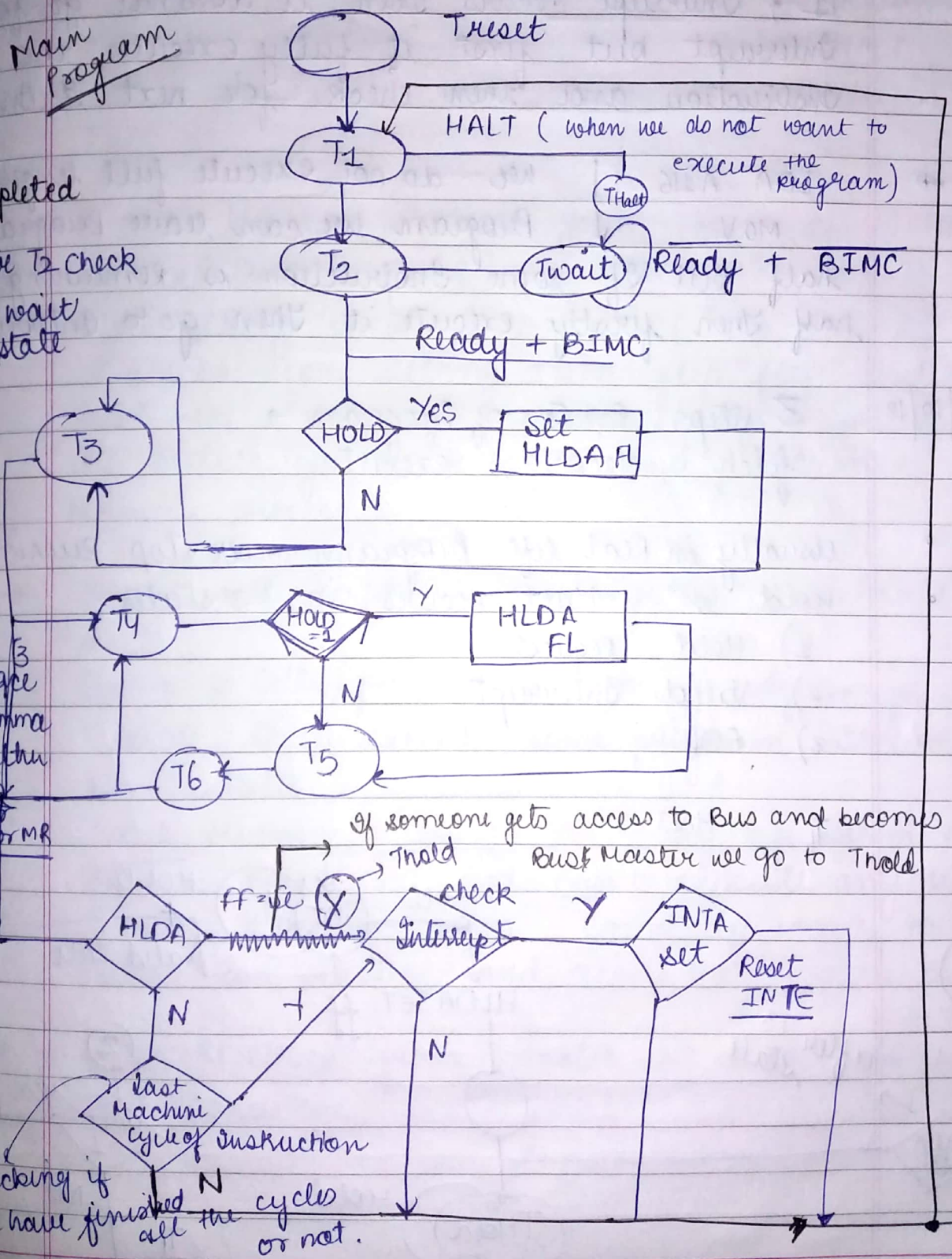
State Transition Diagram :-
difference b/w Timing diagram & State Transition Diagram.

In Timing diagram we see how opcode fetch

In Timing diagram we have T_1, T_2, T_3, T_4 and we will discuss them in STD.
In 4T states One machine cycle.

ASHOKA
Date:
Page No.

is taking place, how an Inf is getting executed but in state transition diagram we look for different-different states.



BIMC is just because of Ready

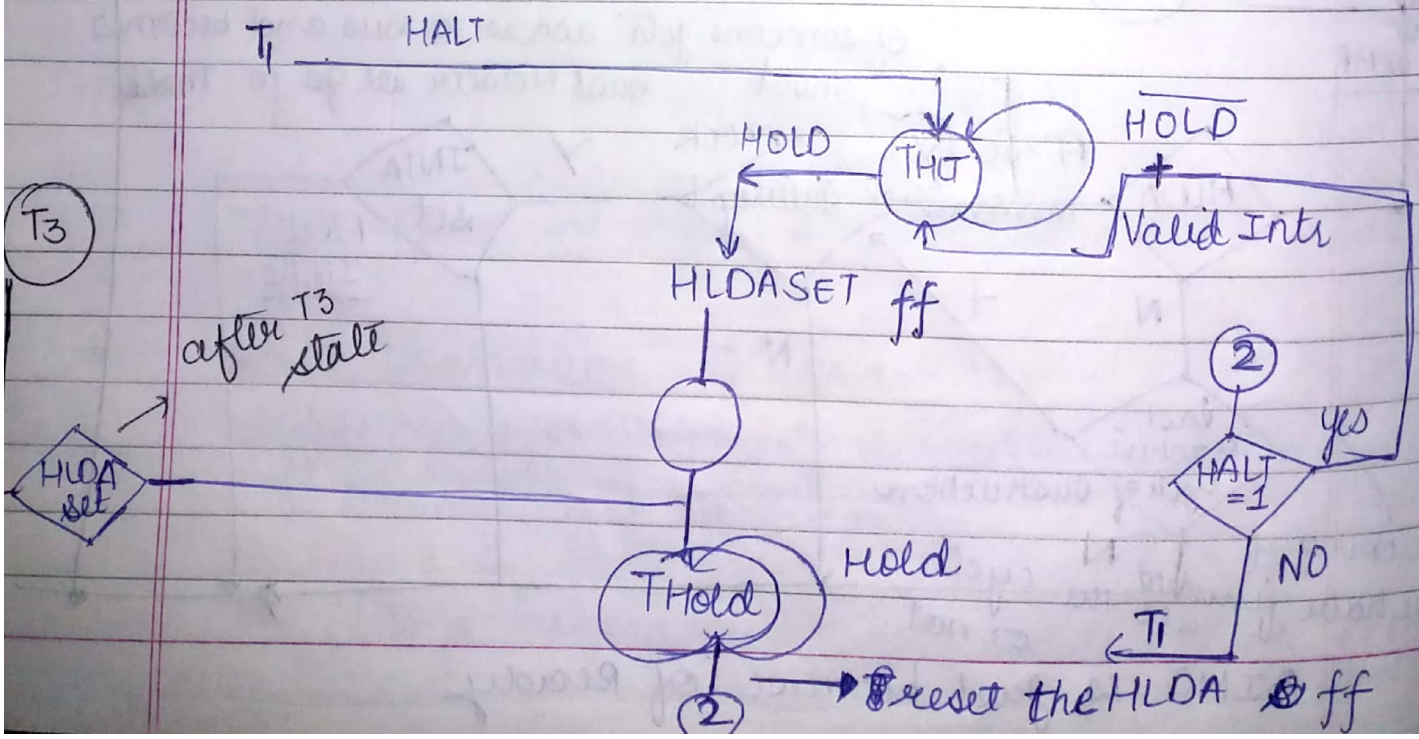
Page No.

- Half of Instruction never gets executed, it is its prior duty to fully execute the Instruction and then step forward That's why if after T1, Interrupt occurs Then it does not go for Interrupt but first it fully executes the Instruction and then check for next Instr.

→ LDA A16 ↓ we do not execute full length MOV Program we can leave Program half But if some Instruction is Remaining half Then firstly execute it Then go to Interrupt.

15/10/18 3 steps Parts of Program → fetch, decode, execute

- Usually in Real life Programs never stop Running.
- ~~Hold~~ ~~or~~ Halt occurs in 3 states.
  1) Hold Request
  2) Valid Interrupt
  3) EOP.

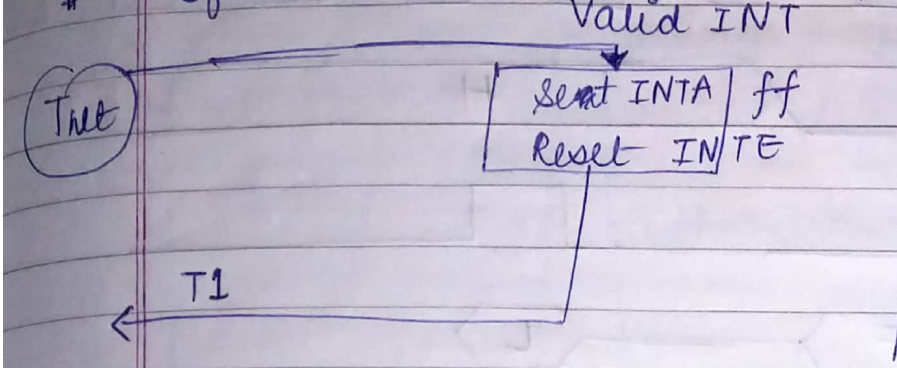


Hold means stopping the current Program can occur due to

→ HLDA request
→ Valid Interrupt

ASHOKA
Date:
Page No.

If it is EOP we handle it differently.



- If after Halt = 1 ans yes go to Thalt If NO Means Regular Process go back to T1 again.
- After getting valid Interrupt (6.5, 5.5, 7.5) it checks for setting INTA flip flop and Reset the Interrupt enable. After setting this enable go back to T1 and perform Normal process.

→ What will happen when system is in Treset state?
System is Idle, performing Nothing. System pointing to 00. To execute some program, it comes to T1 states.

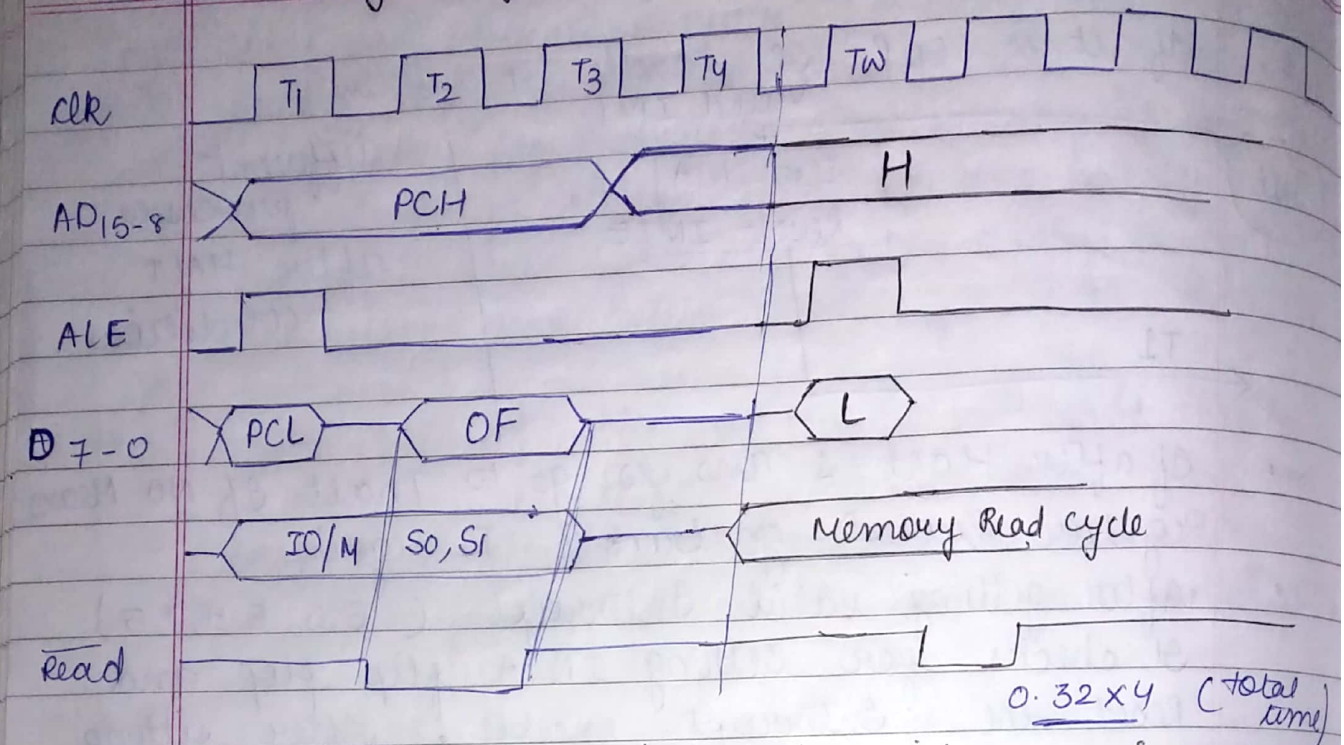
→ Twait Means system is Not Ready to perform the further operations and can't execute anything.

→ Thold It occurs when somebody wants to become Bus master and access the system Bus.

→ After Resetting from Thold we check for Halt condition (Yes / NO).

ASHOKA
Date:
Page No.

Timing Diagram of HALT :-



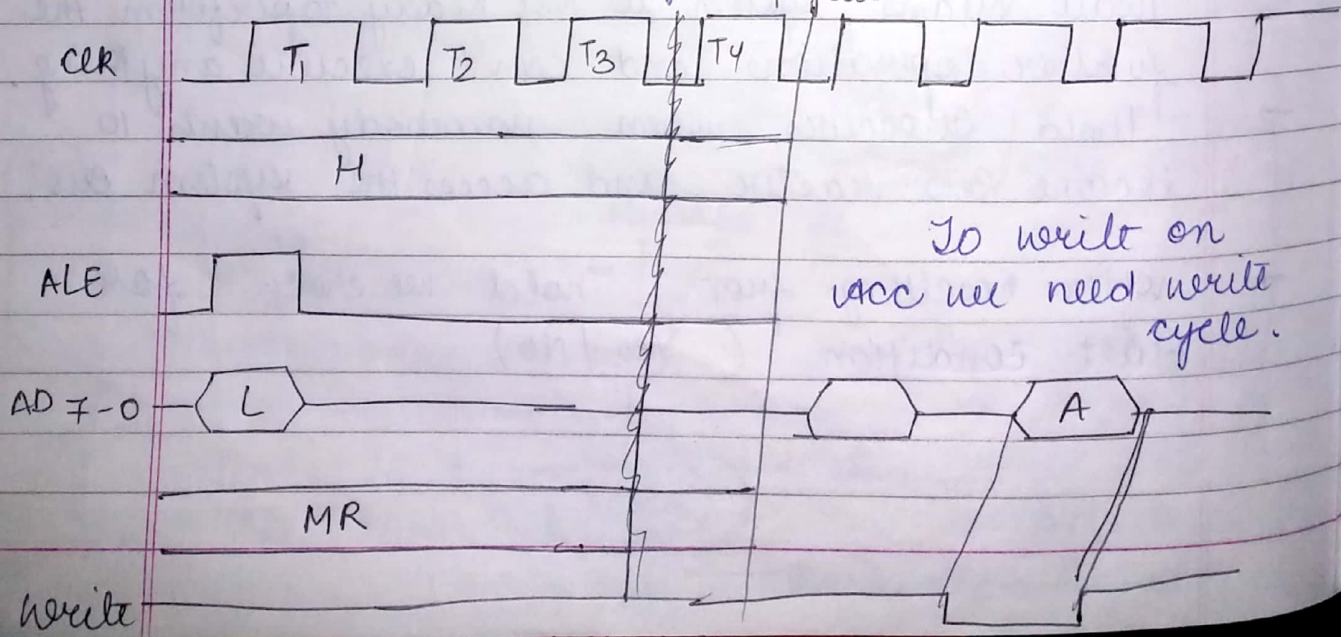
0.32 × 4 (total time)

After opcode fetch S/s goes to wait state in execution of Halt command.

Halt is not shut down (same with EOP). It Means system Remains in Idle state. Nothing to execute.

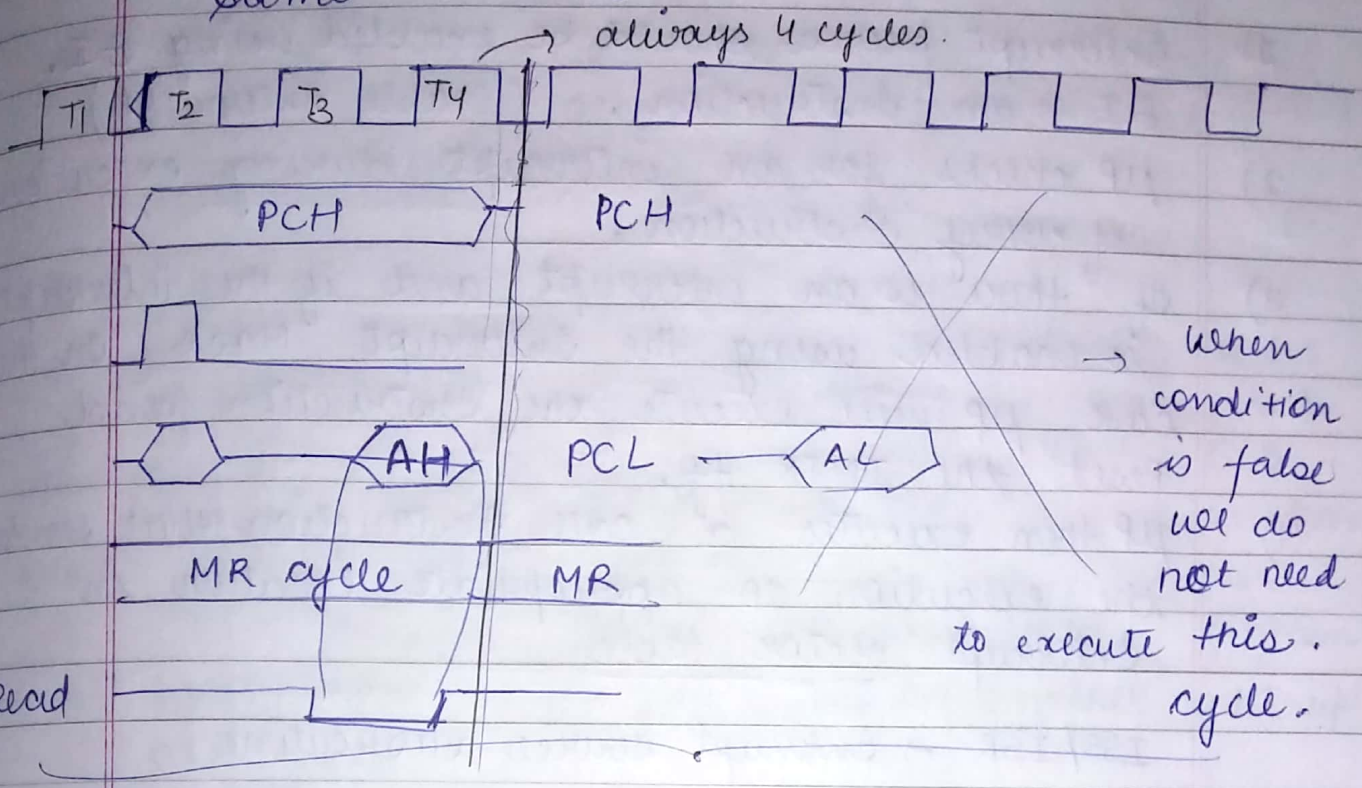
→ MOV M A → 1 Byte
↓ HL pair Register

} Black pen Diagram above



ASHOKA
Date:
Page No.

→ JNC 2001 → 1st opcode fetch cycle same as Halt



→ When condition is false we do not need to execute this cycle.

When condition is true.

Jmp A16 , STAX B , LDAX D

- In opcode fetch CPU only Realise what to do, it does not execute anything.
After OF it starts execution of program & this is now timing diagram is formed.

17/10/18

MOV A, M
MOV M, A
↳ Pointing to HL Register Pair

In MOV M,A cycle (In 2nd Machine cycle we need write cycle to write the content of Accumulator in M Register.

ASHOKA
Date:
Page No.

Unit

Interrupt :-

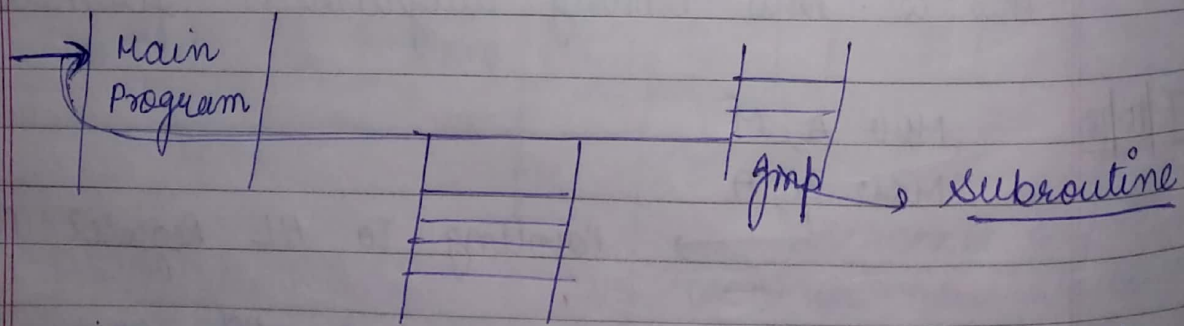
1) Interrupt process should be enabled using EI. EI is an Instruction. (Enable Interrupt)
2) µP checks for an interrupt during execution of every Instruction.
3) If there is an interrupt and if the interrupt is enabled using the Interrupt Mask. In this case µP will execute the Instruction and Reset the INTR ff.
4) µP then executes a call Instruction that sends the execution to appropriate location in Interrupt vector table.

ISS/ISR → Interrupt service subroutine

\# The call address is already being set before (Things we need to see in vector table).

⑤ DI → 1st line of code. First line of subroutine is always DI. and 2nd last line of subroutine EI. Last Ins. of subroutine is Return (conditional | unconditional).

→



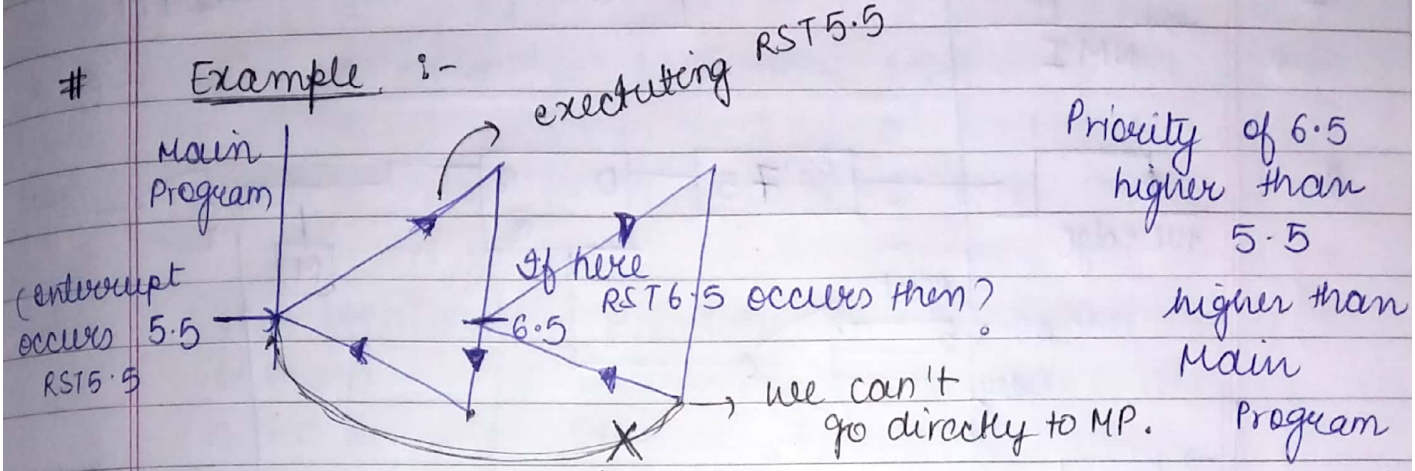
With call we can even handle 13 Interrupts because we ~~not~~ are working with the help of stack But with Return we can't do this job Perfectly.

ASHOKA
Date:
Page No.

conditional call is never a part of Interrupt handling. 'Call do help us.'

→ We need to know about the equivalent address of the Interrupt occurring. That's what vector Table tell us.

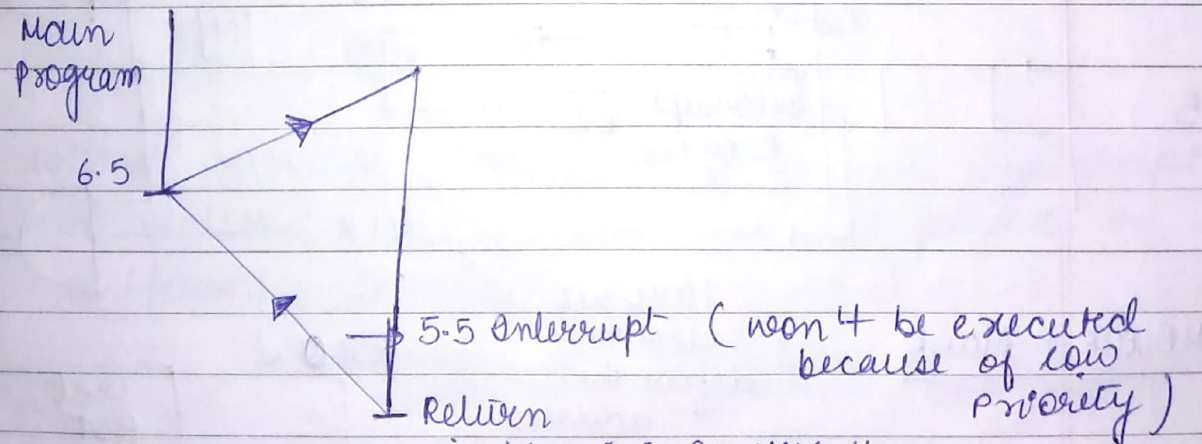
Example :-

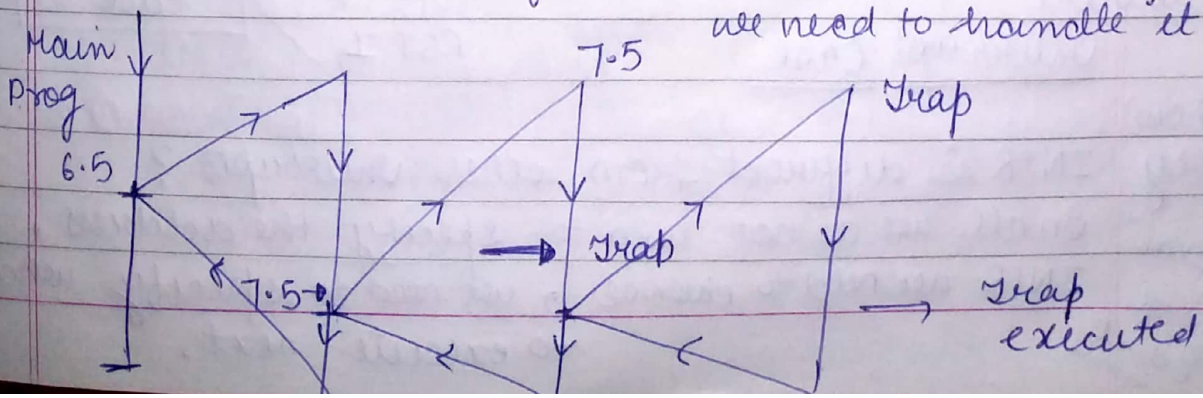


Until unless EI flip flop is on no another Interrupt can occur.

Because here we are dealing with stack LIFO

→ How to decide we need to enable DI or not ?

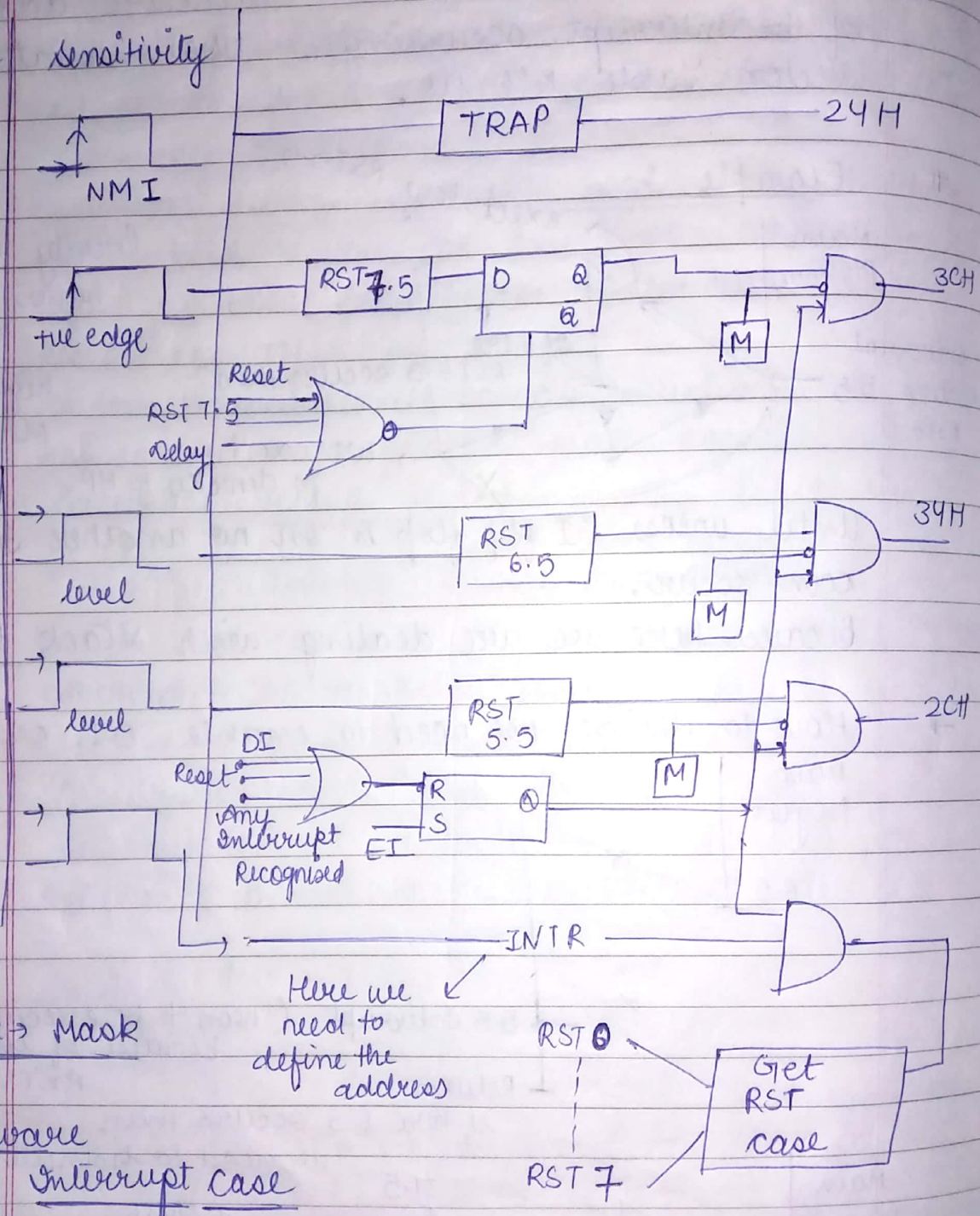




→ If you enter in a subroutine then you have to execute it fully upto Return.

ASHOKA
Date:
Page No.

→ The only way to come out from a subprogram is to stop, Reset and Return.



MBLOCK → Mask

Hardware Interrupt case

(how) why INTR is different from other Interrupts?

In all we do not need to specify the address. In INTR we need to choose, we need to specify what to execute next.

Hop → Peripheral knows all things to execute

Vectored Interrupt Table :-

ASHOKA
Date:
Page No.

Software Interrupt :-

	D7	D6	D5	D4	D3	D2	D1	D0	HEX
RST 0	1	1	1	0	0	0	1	1	C 7
" 1	1	1	1	0	0	1	1	1	C F
" 2	1	1	1	0	1	0	1	1	D 7
" 3	1	1	1	0	1	1	1	1	D F
" 4	1	1	1	1	0	0	1	1	E 7
" 5	1	1	1	1	0	1	1	1	E F
" 6	1	1	1	1	1	0	1	1	F 7
" 7	1	1	1	1	1	1	1	1	F F

continued →

	Call location	Relation b/w Hardware & software Interrupt.
0	0000	
1	0008	24 H
2	0010	3C H
3	0018	RST 4.5 (Trap also known as)
Trap 4 →	0020	Trap (24 H)
5.5 5 →	0028	2CH
6.5 6 →	0030	34 H
7.5 7 →	0038	3CH

You check b/w their Relationship.

RST 0 → vectored Interrupt because we know its Interrupt

INTR → Non vectored Interrupt because it points to different locations

Steps for execution of Interrupt

1) EI
2) INTR → check
3) INTA → accessed when (INTR reset)
4) ~~RST~~ RST → external H/W
5) RST → PC → stack → ISR

If INTR occurs → systems will check for the suitable RST.

Next page

ASHOKA
Date:
Page No.

⑥ finish ISR → EI
⑦ RET

→ First thing what S/S do is acknowledgement any then it starts for recognising.

→	Non Maskable Interrupt	Maskable Interrupt :
1)	Can't be Masked	Masked or Made Pending
2)	High Priority	Low Priority
3)	Interrupt disable all Maskable Interrupt	doesn't disable
4)	Power failure / smoke detection (applications)	4) Used for Peripheral Interface

	Software Interrupt	Hardware Interrupt (coming Randomly)
1)	Synchronous event	Asynchronous event
2)	This Interrupt is Requested by executing Instructions	This Interrupt is Requested by external device
3)	PC → PC++ (PC gets Incremented)	PC → PC (PC never get Incremented, Remains same)
4)	Low Priority	High Priority
5)	Ignored (can be)	Can't be Ignored
6)	Applications → Break point / BIOS, debug	Applications → with GPIO's

ASHOKA
Date:
Page No.

→ 5 Hardware Interrupts and 8 software Interrupts Then how many devices we need?

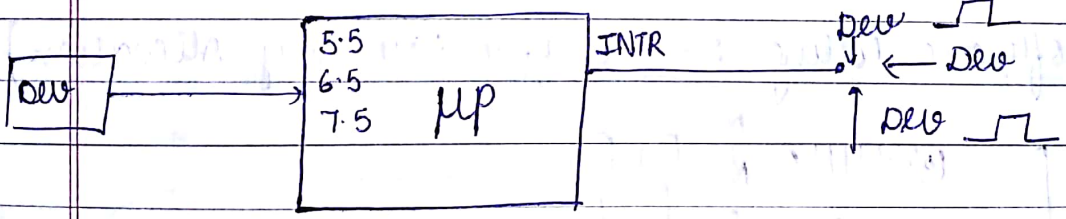
22/10/18 Why do we need to acknowledge INTR with INTA Request But not RST 6.5, 7.5, 5.5?

Because INTR is non vectored Interrupt, and we do not know exact address for the execution?

→ Highest priority Interrupt 4.5 and last one is 7.5.

By Trap, 6.5, 5.5, 7.5 we will always jump to specific address location.

→ If Multiple events are occurring at the same time and they are happening at INTR side Then How we will handle such complexities?
If Multiple events wants CPU time at the same time?

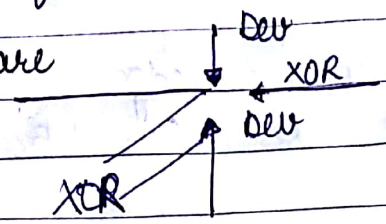


Priorities of all the devices are set Initially. Priority is not any problem here That if Dev1 wants CPU time Than Initiate RST0 and so on.

We need an Intermediator (device) b/w device and μP who handles such type of complexities. computer can't perform this. Hire someone as a consultant and then tell computer what to do.

→ Let's put logic gates b/w device and μP.
If XOR is that device.

(Hardware Polling)

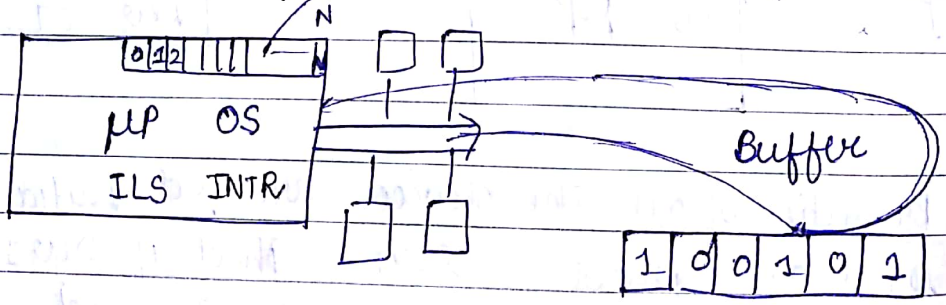


all devices are XORed If all occurs at the same time Than they are neglected. If 2 occur at the same time also Neglected but If one device is ON It gets a chance One device is coming at one time But what about other devices who also wants CPU time at the same time and Making several attempts for that so logic gates are efficient for small and cheap devices.

Polling :- When ~~they~~ there are multiple Requests, so to handle those Requests in a certain manner at the same is Polling concept.

- Software Polling
- Hardware Polling

1) Software Polling :- → Buffer (Memory Allocation)



Devices Put their Request in Buffer and then accordingly They are given time.

0 → No time
1 → Need CPU time

→ Most Imp thing in Interrupt is Priority.

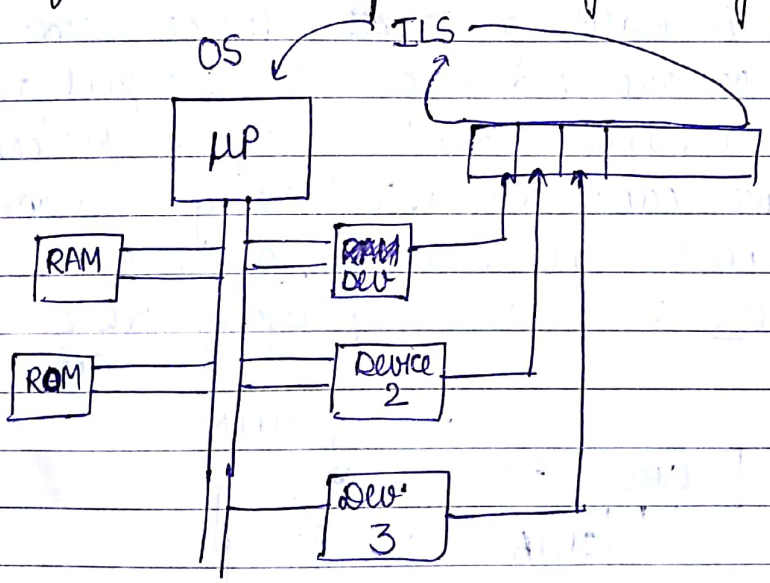
ASHOKA
Date:
Page No.

When OS gets to know how many devices need CPU time Then our work gets easy. Initially S/S didn't even know how many devices want CPU time Because of one Pin. Initially we were getting Interrupt and then handling it. Now with the OS we know total no. of devices.

→ Suppose device 1, 4, 6 wants CPU Time Than we give highest Priority to 1 and put 4 and 6 on hold. When we are using ILS (Interrupt level Subroutine) It is is software Polling. ILS and IRS are totally different. OS decides on the Basis of Interrupt priority

Software Interrupt Polling diagram :-

(A)

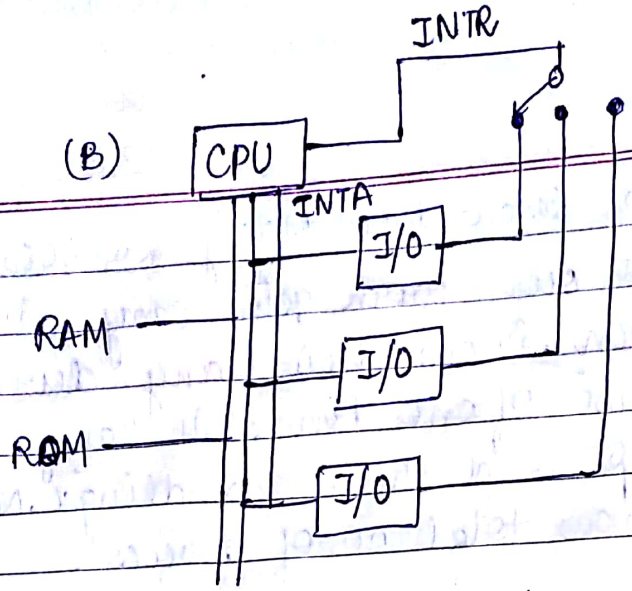


Better option

Hardwired/Hardware Interrupt Polling diagram :-

INTA connected to all I/O port so whichever is needed, (Requested) Acknowledgement is sent to that.

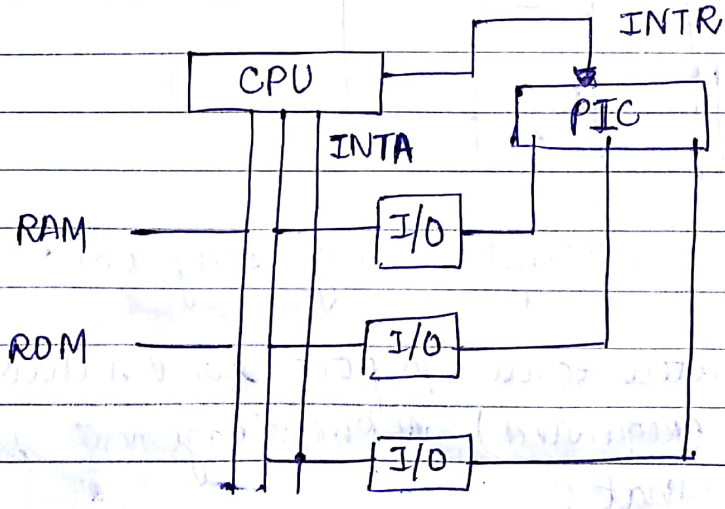
ASHOKA
Date:
Page No.



→ On level of compatibility both are same. But A (software -) is much better than B because it is cheap (cost effective), design better than B. In B we need I/O devices (means one more device connection). But in A we have device directly connected.

ifixit (website). But in A, one part of our prog is busy in watching which device needs CPU time. By this we are engaging CPU, we put an ∞ loop for the continious checking of the devices.

So we can't see after every 5 seconds which device needs CPU time. So we go with another solution :- PIC (programmable Interrupt controller).



Date: 24/10/18 ASHOKA
Page No.

Interrupt is a process of data transfer where an external or peripheral can inform the processor that it is Ready for communication and it Requires attention.

OR It is a subroutine call Initiated by external device through hardware or µp itself.

H/w Interrupt — S/w Interrupt

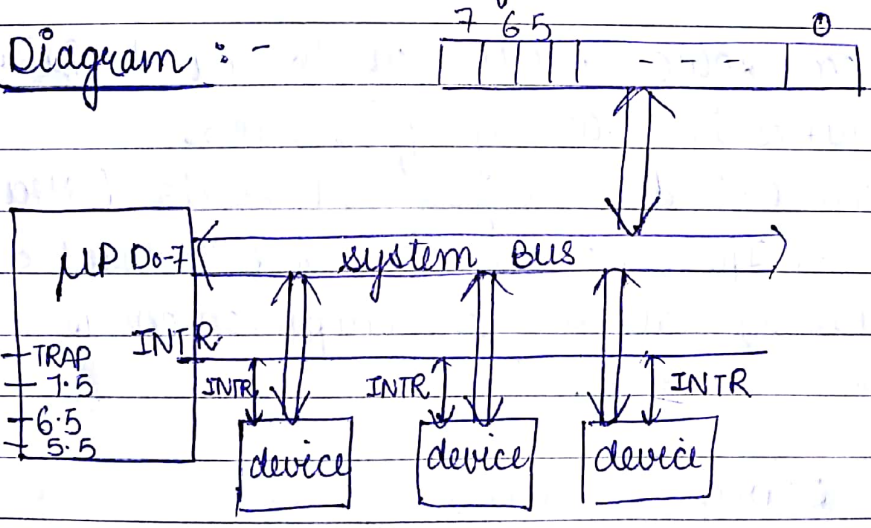
Software Int. are not initated by external devices they are Initiated by µP itself.

Polling :- Requires no special H/w and all the data Transfer are controlled by CPU Program.

→ It is a synchronous Mechanism by which devices are serviced in an order (sequential)

Limitation :- Wasting Processor time (needlessly checks the status of devices)

Diagram :-



Through hardware CPU gets to know that someone wants CPU time But it is not getting exactly known who needs CPU time. Once we know this thing, our work gets sorted.

INTR → Hardwired Interrupt.

ASHOKA
Date:
Page No.

→ By data Bus ILS gets Initialed.
From device 0 to device 7 → They get initialed
But we need to give them priority which needs first our attention.

→ For simply Handling I/O we are not sufficient to handle the Interrupt alone, we need a candidate so we do hiring.
PIC (programmable Interrupt controller).
↳ It can Manage 8 Interrupts
↳ can Manage 8 Interrupt Requests
↳ can vector an Interrupt Request.

If we know the vectored address of Interrupt, we know where to go when that Interrupt occurs. Vectored address helps us to know what to do (what is the subroutine).

↳ It can solve 8 level of Interrupt Priorities.
Priorities in variety of Modes.
↳ It can extended upto 64 levels (Means If we connect 64 I/O devices it will give vectored add of all, Manage all the Interrupt which is a very complex thing).
8 devices

8259 Device Name of PIC

→ Priority Resolver → All the IRQ's are given token no. to know about the priority. Generally it is higher the no. higher the priority and lower no ⇒ lower priority
Windriver Rocklos

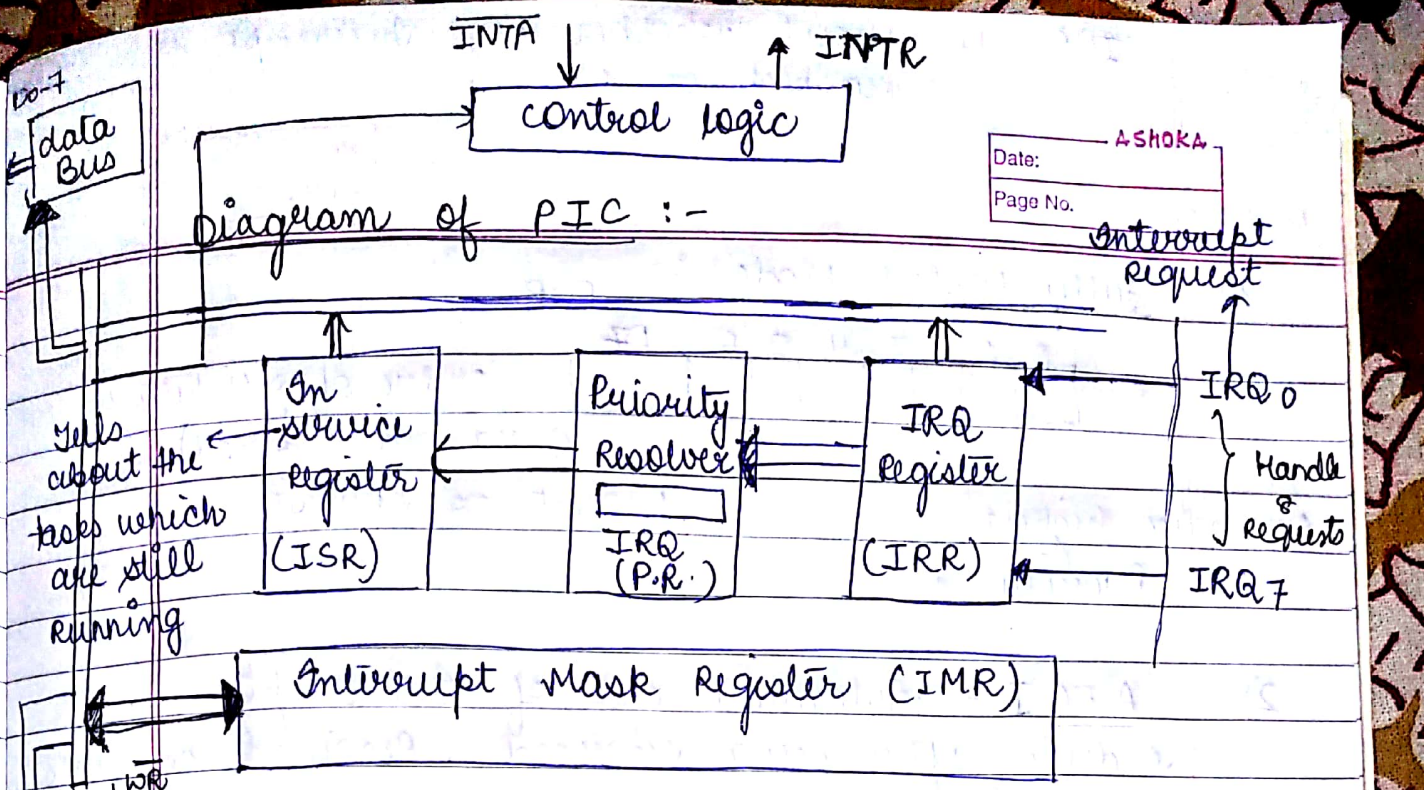


Diagram of PIC :-

Interrupt is a data transfer / sharing of Inf Initialed by external devices.

chip select → It is very Important, tells us (when it is enabled) that some other device wants to connect

SP/EN → which devices needs to be Master and which devices needs to be slave

- In this current status of execution / Running is always told by ISR.
- A0 → Address line (Zroth line) If chip is selected or enabled. A0, CS → 11 To enable the whole chip we need Both of them high.
- → SP → slave Program Enable.

Here Process is call 8011

3 INTA

INTA 1 → ACKnowledgment
2 → LOB
3 → HOB

How it is different from those 7 steps

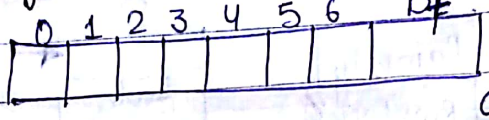
Execution → (after the ISR chip)

IMR tells about whether any Interrupt is masked or not.

ASHOKA
Date:
Page No.

Modes of ——

1) Fully Nested Mode :- EOP



order of priority is arranged (fixed) Highest to lowest

D7 have highest Priority.

2) AEOI → Automatic end of Interrupt
A device after being serviced, Received the lowest Priority.
eg In above D7 have highest Priority when D7 gets executed it gets the lowest Priority and D6 comes on top.

3) Specific Rotation Mode → If someone is served, it gets the lowest Priority and the Interrupt next to it gets the highest Priority. It the most common user choice Because here we are deciding the priority of Interrupt.
(Planned and perfectly designed)

By ISR, Interrupt Request sent to μP with the help of INTA (externally Initiated signal).

26/10/18 (4) Initialisation control / comment word (ICW)

It does not work like RIM/SIM. Here we need to work Bit vise Bit.
A0 → enable and Mode select Both

with this ICW we can tell device working as Master or slave.

1 → level triggered
0 → edge triggered

ASHOKA
Date:
Page No.

A0	7	6	5	4	3	2	1	D0
0	A7	A6	A5	1	LTIM ET	ADI	SNG	IC4

1 Yes
0 NO

Address Interval
Interval of 4 or 8
(1) (0) → to set

(1/0) Address Interval

IRQ1 Call 8000
IRQ2 Call 8004/8008

} creating Interval helps alot in controlling the flow of data and to Reduce corruption of data.

SNG (Single) are you single NO → Means Multiple pins are connected (single PIC or multiple PIC)

IC4 → Related with Information, If more Information is Required or not

ICW2 (passed with ICW1 to get full address)

A0

1	15	14	13	12	11	10	9	A8

(giving half address)

N ← single → Y

ICW3

S7	S6	S5	S4	S3	S2	S1	S0

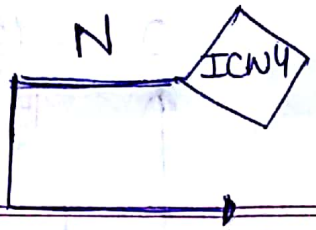
0 → Not Available
1 → Yes/Available

N ← ICW4 → Y (do you need ICW4 or not)

1	0	0	0	SFNM	AEOI	MPU	M/S

SFNM: 0/1 special fully Nested Mode
AEOI: 1/0
MPU: 1 – 8085 used, 0 – 8086 used
M/S: Master/slave

(continued next Page)



ASHOKA
Date:
Page No.

Ready to accept Interrupt

Call 2030 :-

			MW	MW
$T_0 - T_4$	MR	MR	PCL	PCH
OF			SR_1	
			SP_L	

RET :- Unconditional

$T_0 - T_4$	MR	MR
OF		SP

RET conditional

$T_0 - T_6$

OF