Prepared and Written By
RAZEEN AHMED

CSE-341

[MICROPROCESSOR]

HANDWRITTEN NOTE

Computer Science and Engineering
BRAC University

Github: github.com/razeen
LinkedIn: linkedin.com/in/razeenahmed

Subject : (Microprocessor Basics)

Date :
Time :

"Wherever programming is required we need a microprocessor"

Inside Computer memory there stores two type of thing :-

(i) Program (ii) Data

When we say memory when we are learning 8086 microprocessor then we are talking about only RAM & ROM. Because, only RAM & ROM existed in the system of 8086 microprocessor (No secondary memory existed)

RAM → Random access memory
&
ROM → Read only memory
→ PRIMARY MEMORY

RAM (Two types)

(i) STATIC RAM (S RAM) : uses flipflop

(ii) DYNAMIC RAM (D RAM) : uses capacitor

★ All the data's are stored in only zero's and one's format
[ALSO EVERYTHING STORES IN THE MEMORY AS 0's]

3 bit per pixel
↳ 8 bit color shades

4 bit per pixel
↳ 16 bit color shades

INSTRUCTION CYCLE :

(i) Fetch → gets the instruction binary from the memory.

↓

(ii) Decode → Tries to understand the instruction

↓

(iii) Execute → Executes the program

Subject :

Date :
Time :

(i) a = b+c High Level Language

(ii) ADD B,C Assembly language

(iii) 01101011 ; Machine Languages

Compiler (Converting to MC)

Assembler (Converting to ML)

OPCODE

[Using the opcode the machine execute its specific operation]

OPCODE:- It is a specific instruction in the machine language of a computer architecture. It represents a fundamental operation that the CPU can execute

"OPCODE" Short for "Operation Code"

2 bit opcode → 4 combinations of instruction
2^2 [2:4 decoder]

3 bit opcode → 8 combinations of instr.
2^3 [2:8 decoder]

4 bit " → 16 combinations of instr.
2^4 [2:16 decoder]

5 " " → 32 combinations of instr.
2^5 [2:32 decoder]

6 " " → 64 combination of instru.
2^6 [2:64 decoder]

7 " " → 128 combinations of instruc.
2^7 [2:128 decoder]

8 " " → 256 combinations of intr.
2^8 [2:256 decoder]

00 → ADD
01 → SUB
10 → MUL
11 → DIV

Subject : Time :

"H" → represents hexadecimal number

HEXADECIMAL	BINARY
0 H	0000
2 H	0010
3 H	0011
4 H	0100 (4 bit)
10 H	0110
15 H	1111
43 H	0100 0011
210 H	0010 0110 (8 bit)

Here,

8 bit NO's

00 H

(It is a 8 bit number)

(But, in the computer we don't write 8 bit no. we use it hexadecimal form. Also inside it will be stored as 8 bit means "0000 0000")

FF H

16 bit NO's

0000 H

FFFF H

"Whatever the data is (be it song, image, movie etc), it is actually transferring only 0's and 1's through BUS".

→ 1 bit bus line can transfer 1 bit at a time

"size of a bus is the number of lines"

ADDRESS Bus: The job of Add Bus is to carry the address.

DATA Bus: The job of the data Bus is to carry the data

CONTROL Bus: The job of the control Bus is to ctrl the operation.

Subject :

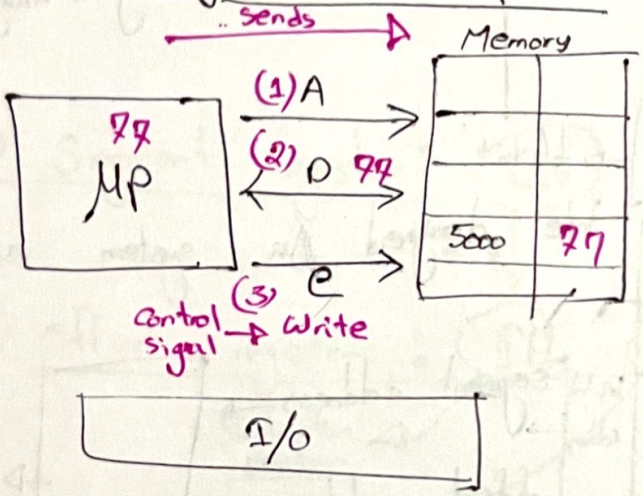
Date :

Time :

READ means processor is getting DATA from the memory

WRITE " " " sending DATA to " "

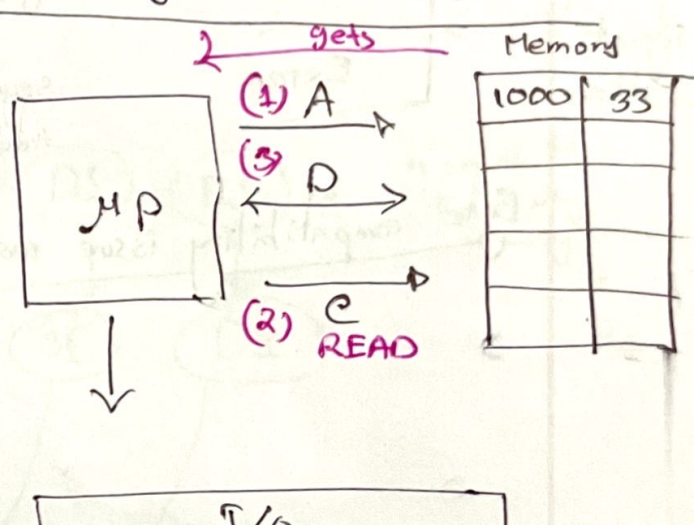
WRITE Cycle in the processor



77 → MP → A. BUS

C. BUS (Write) ← D. BUS

READ Cycle in the processor



MP → A. BUS → C. BUS (READ)

D. BUS

Subject : every offset increases by 1

Date : 28/01/24
Time :

offset address

offset

MEMORY SEGMENTATION :

Segments

- CODE → Program, applications
- STACK
- DATA → image, video
- EXTRA

Purpose
- → Byte compatibility issue resolve
- → Better memory managment

Starting Address

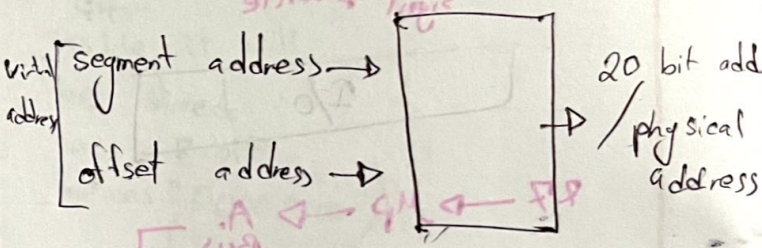
let's say 1 MB Memory

segment address

segment address + offset address → 16 bit

We desiged a system using :-

Byte compatibility issue resolve :-



The segment has been done by thinking every segment as a multiple of 10.

Formula for Physical address :-

Physical address = Segment address × 10 + Offset address.

(imp)

offset range → 0000 H

2^{16} → FFFF H

$= 2^6 \times 2^{10}$

$= 64KB$ → locations

friendship

Subject : Time :

8086 REGISTERS

(i) General Purpose Registers:- AX, BX, CX, DX ← (16-bit registers)

AH AL BH BL CH CL DH DL ← (8 bit registers)

(ii) Segment register: CS, SS, DS, ES ← 4 segment thats why 4 register

(iii) Offset register: IP, SP, SI, DI

IP → Instruction pointer

SP → Stack pointer

BP → base pointer ← for stack random access

SI → source index

DI → destination index

PROBLEM :-

Given, SS = 3000 H from what location stack segement will start?

Ans:-

3000 H x10 = 30000 H

PROBLEM :-

Stack segment starts from SS

Physical address:

54320 H → 5432

97880 H → 9788

97513 H → invalid ← because start segment should be a multiple of 10

Subject :

Time :

PROBLEM :-

A segment ends at 54320 H ← 1 byte

21 ← 1 byte

23 ← "

24 ← "

29 ← "

2A ← "

2F ← "

54330 H ← "

minimum memory → 16 byte

STACK PUSH & POP OPERATION :-

SP-2	BL
SP-1	BH
SP	XX

PUSH BX → BH, BL

QUIZ

- PIPELINING
- Memory banking
- segmentation.

POP BX

SP	34
SP+1	12
SP+2	

→ BH | BL

~~SS : [BH]~~

~~SS : [BL]~~

SS : [SP-1] = BH

SS : [SP-2] = BL

SP = SP-2

BH = SS : [SP+1]

BL = SS : [SP]

SP = SP+2

Subject :

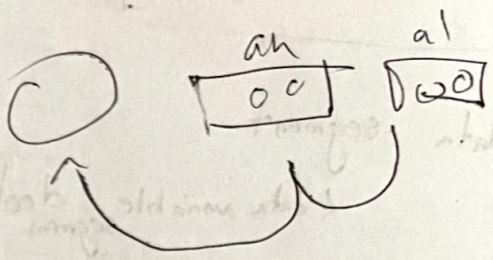
Time :

Divide:- The number we want to divide :→ AX (must be) (Dividend)

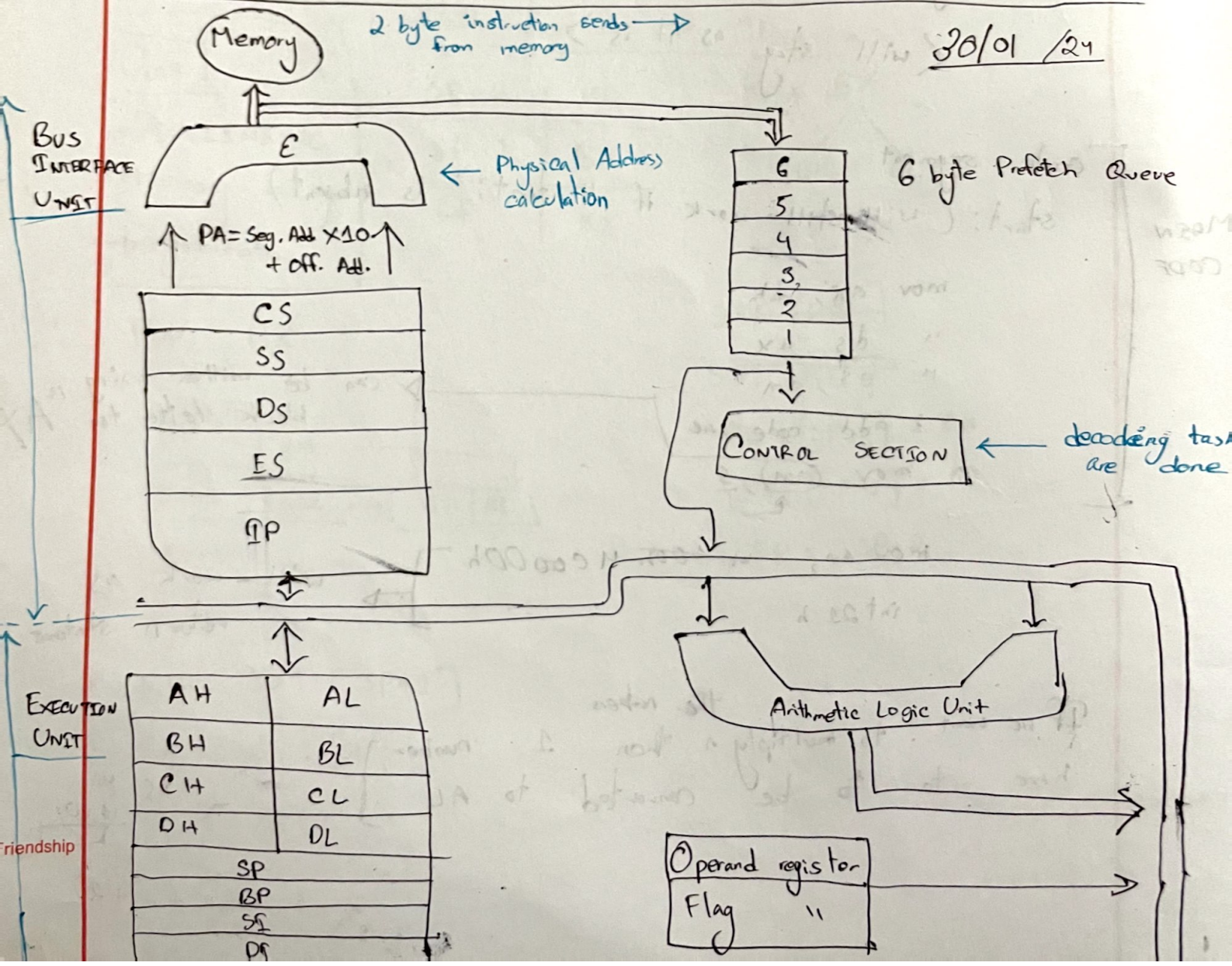
" " which we will " :→ Any register (Divisor)

Task:

$6 + (9/5) + (8*2)$

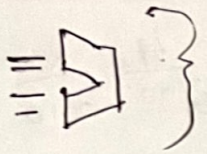


30/01/24



Friendship

Subject : Time :



READ → sends Address
write → Add. & data sends

☆ Instruction pointer always points to the next instruction which needs to be executed.

Why prefetch Queue size is 6 byte?

Operand register get used used in case of swapping. in assembly code.

Flag register :-

If "OF" = 1 (bit overflow)
" OF = 0 (not ")

Status Flag :- OF, SF, & ZF, AF, PF, CF.

Control Flag :- OF, TF, & IF. controls bit operation

8 bit signed range :-

-128 → 127
-80H → 7FH

16 bit signed range :-

-8000H → 7FFFH

(positive no. + negative no.)
↳ can't be overflow (There is no possibility)

(postive no. + positive no.)

(negative " + negative ")
↳ can be overflow. (There is possibility)

Subject :

Date :
Time :

PROBLEM :-

(i) 42H & 43H

4	2	8 4 2
0100	0010	0 1 00

4	3
0100	0011

42 ⟶ 01000010

43 ⟶ 01000011

no carry (no auxillary)

```
  0 1 0 0   0 0 1 0
  0 1 0 0   0 0 1 1
-----------------------
1 0 0 0   0 1 0 1
↳ SF
```

CF ⟶ 0 (no extra bit)

OF ⟶ 1 (7th to 8th bit)

SF ⟶ 1 (starting bit)

AC ⟶ 0 (no carry from travelling from upper lower 4 to upper 4)

PF ⟶ 0 (coz we have odd nom. of 1)

Subject :

RRH

Physical to logical address with

Date :

Time :

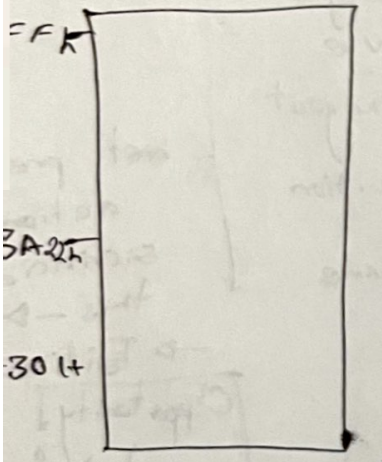
0000 h
FFFF h

Physical Address = 23A22 H

☆ Find the highest/maximum and the minimum/lowest logical Addresss.

highest logical address = Seg : offset
(4 hex) (4 hex)

= 23A2 : 0002 h



Subject :

If its divisible by 16 then its starting address.

Lec-8 important

Date :
Time :

☆ PROBLEM

Physical Address = 1F2F3h

(i) Calculate the 3rd highest logical address.

(ii) Calculate " " lowest " "

Solve :-

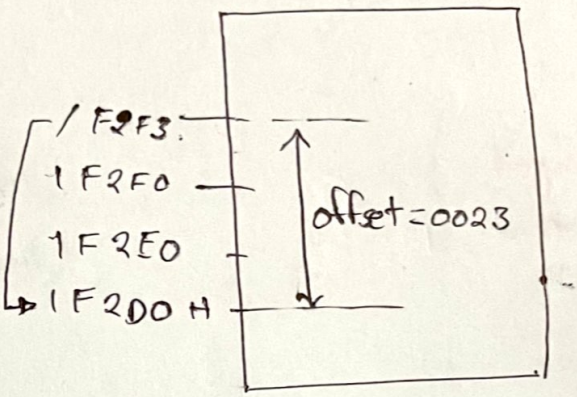
1st find out the highest logical address then gradually decrease the number

(i)

Ans :-

3rd highest logical address :-

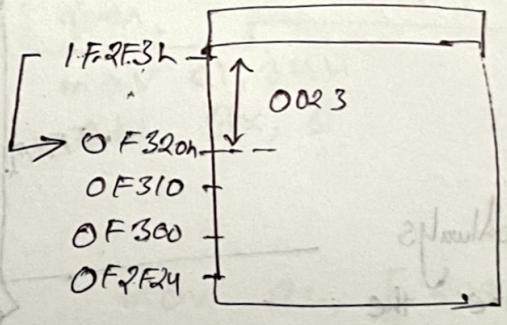
= 1F2D0 : 0023 H



100
0FF
:
0F5
0F4
0F3

(ii) 3rd lowest logical Address :-

= 0F320 : FFD3



Subject :

Date :

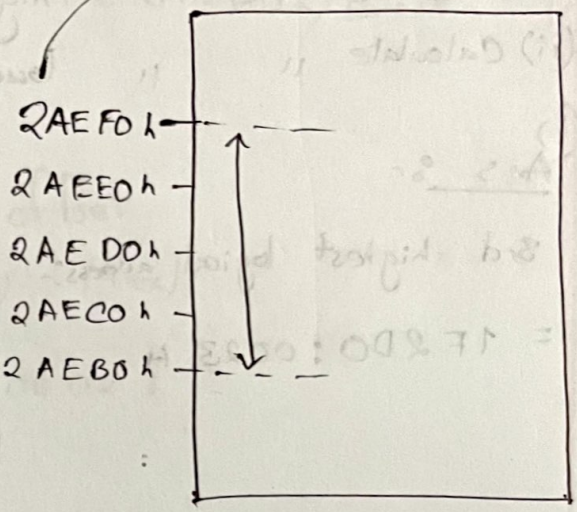
Time :

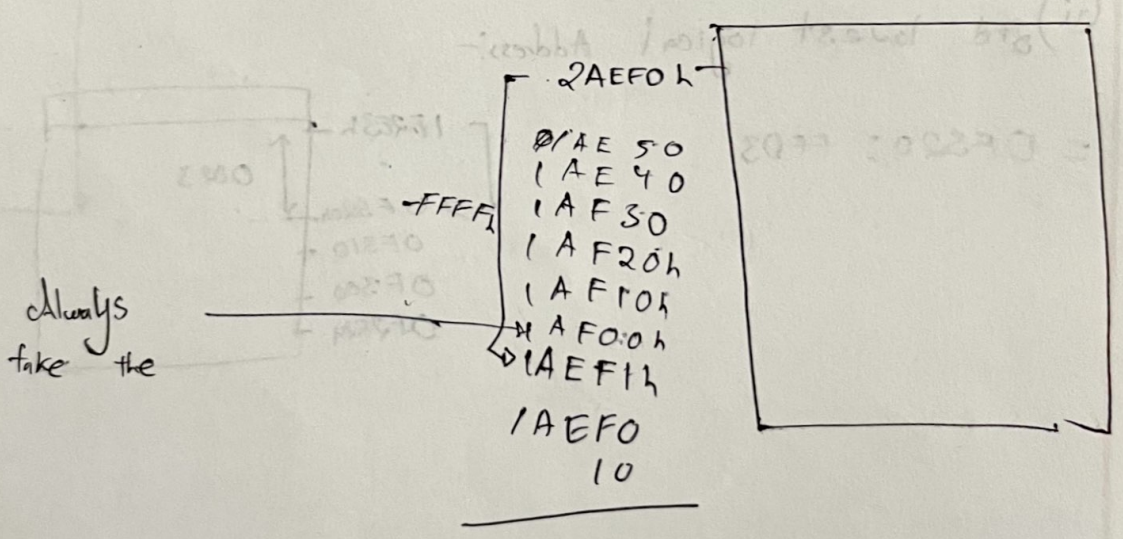
PROBLEM Physical address = 2AEF0 h.

(i) Calculate the 5th highest and 6th lowest logical address

highest logical Address cuz divisible by 10

5th highest logical address = 2AEB0





Subject : RRH

FLAG REGISTER

Time :

OF = Overflow flag
SF = Sign flag
ZF = Zero flag
AF = Auxillary flag
PF = Parity flag
CF = Carry flag

carry out ← carry in

MOV AL, FFh → 1111 1111
MOV BL, 01h → 0000 0001

ADD AL, BL | 1 0000 0000

extra class
SUN → 9-30 10E-3L
11:00 9C-15C
2 pm JA-01C

SF = 0 [msb = 1]
ZF = 1
AF = 1
PF = 1 [If there is even no. of 1] ('0' 1 even)
CF = 1
OF = 0 [There is both carry in and carry out]

OF = 1 (If there is either carry in or carry out)

MOV AX, FFACh → high byte 1111 1111 | low byte 1010 1100
MOV BX, 2388h → 0010 0011 | 1000 1000
carry in
carry out

ADD AX, BX

1 0010 0011 1001 0110

SF = 0 [msb = 0]
ZF = 0
AF = 1
PF = 1 [In low 8 byte there is even no. of 1]

CF = 1 [there is carry bit]
OF = 0 [there is both carry in and carry out]

Subject : LECTURE-4 Time :

BX → BH, BL

[ ] → Memory location; offset

MOV AL, 25h

MOV AL, BL

MOV AL, [BX] → Source

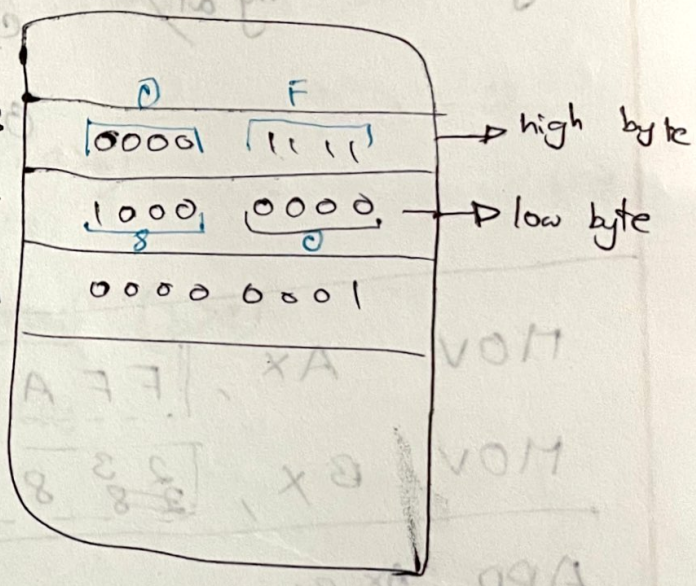
AL → destination

[BX] → P.A. = Seg x 10 + BX

MOV AX, [BX] (Read operation)

Seg x 10 + BX = P.A.

23145h



AX = 0F 80h

AH AL

MOV [BX] AX

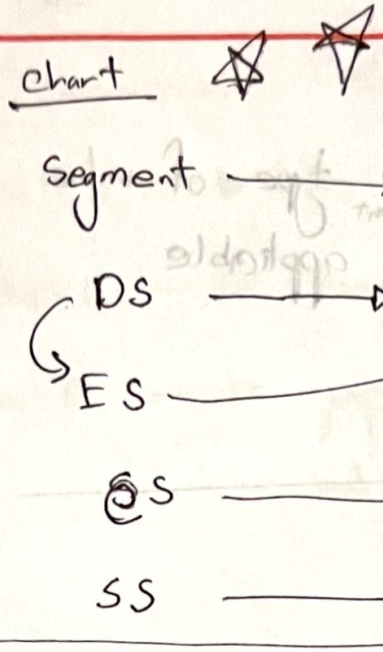
destination source

The data is getting transferred to a certain memory location.

Subject :

Time :

Chart



PROBLEM:

DS = 2F31 h

SS = 1233h

ES = 2F00 h

CS = 1346 h

BX = 1000 h

SI = 0020h

DI = 01F0h

sp = 2346 h

BP = 1000 h

IP = 2246 h

Q. Mov AX, [BX+SI]

(i) Calculate the physical address.

→

P.A. = Seg × 10 + ~~(BX + SI)~~ (offset)

= DS × 10 + (BX + SI)

see the chart

= 2F31 × 10 + (1000 + 0020)

= 30330 h

(ii) Calculate the 2nd highest and 3rd lowest logical address.

Q. ADD [BX + 2200h], AX

(i) Calculate the ~~logical~~ physical address.

P.A = DS × 10 + (BX + 2200)

= 2F31 × 10 + (1000 + 2200)

= 2F310 + 3200

= 32510

Q. MOV AX, [2200h]

→ for this type of value only DS segment is applicable

(i) Calculate P.A.

→

P.A. = DS x 10 + 2200

Q. MOV 25h, AX → Invalid case

Q. ADD [AX], [BX]

→ Because two of them is memory location they need a temp register to add somewhere.

→ Invalid case

Subject : RRH Time :

Q. Identify the addressing mode:-

1. MOV AX, 25h
2. MOV AX, [BX]
3. ~~MOV A~~
4. Add [AX], [BX]
5. MOV AX, [BX + SI + 2000h]
5. ~~MOV~~ ADD [BX+SI+20h], 20h
6. MOV 25h, [BX]
7. ADD, ~~BX~~ AX, [BX + 1000h]
8. MOV AX, [2348h]
9. MOV [CX], [2200h]
10. MOV AX, BX

1 → Read addressing mode

2 → Register indirect

3 → Invalid

4 → Base - relative - plus - index | DS x 10 + (BX + SI + 2000)

5 → Write operation

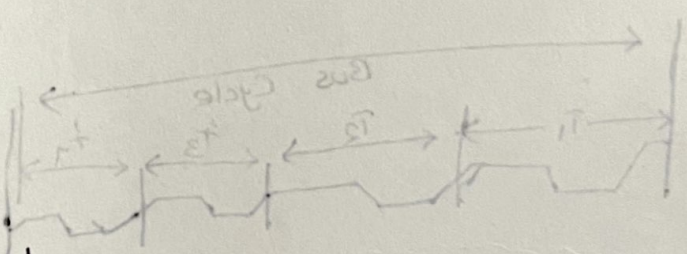
6 → Invalid

7 → register relative | Read

8 → direct addressing mode | Rede

9 → Invalid

10 → Register direct addressing mode



KKH

Date : 18/02/

Subject : Addressing Program Codes in Memory

Time :

When we want to access program code we use

JMP / CALL

CS ← IP

PROBLEM:-

DS = 2200h
CS = 1000h
IP = ?
AX = 2000h

JMP (AX) ← if

(i) Calculate the physical address of the instruction

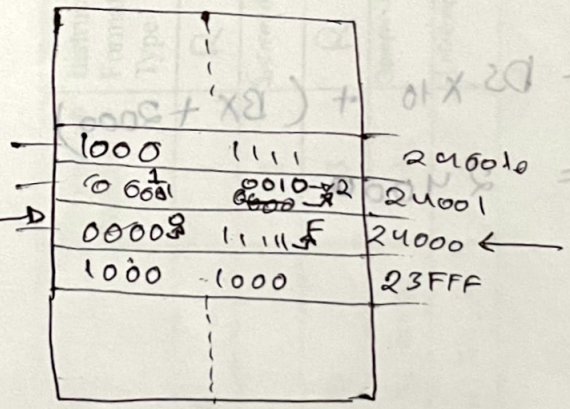
Ans

IP = AX = 2000h

P.A. = CS × 10 + IP

PROBLEM:

JMP [AX]



(i) Calculate the address instruction

P.A. = DS × 10 + AX

= 2200 × 10 + 2000

= 24000h

visited 24000h to get the IP

IP = 120F h

P.A. = CS × 10 + IP

= 1000 × 10 + 120F

JMP [SP]

PA = SS x 10 + SP

= 2200 x 10 + 2000

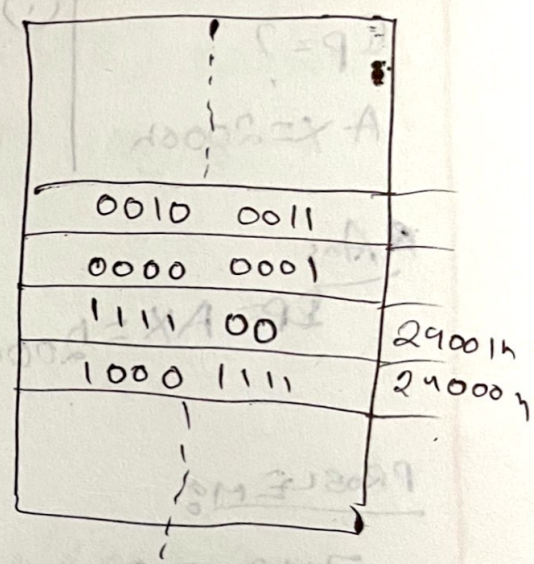
= 24000h

PROBLEM

JMP [BX + 2000]

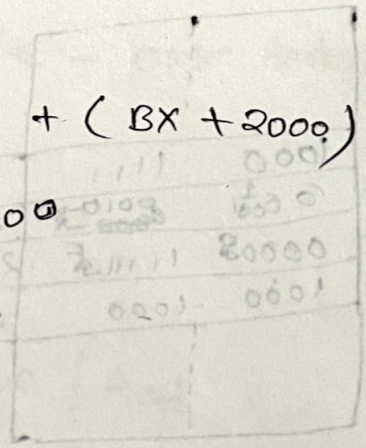
DS = 2000h
CS = 1000h
IP = ?
BX = 3000h

Calculate the physical address of instruction



PA = DS x 10 + (BX + 2000)

= 24000



Subject : INTERNAL ARCHITECTURE

Time :

Why the ~~pf~~ prefetch queue is 6 byte

→ The maximum instruction which 8086 microprocessor can execute is 6 byte.

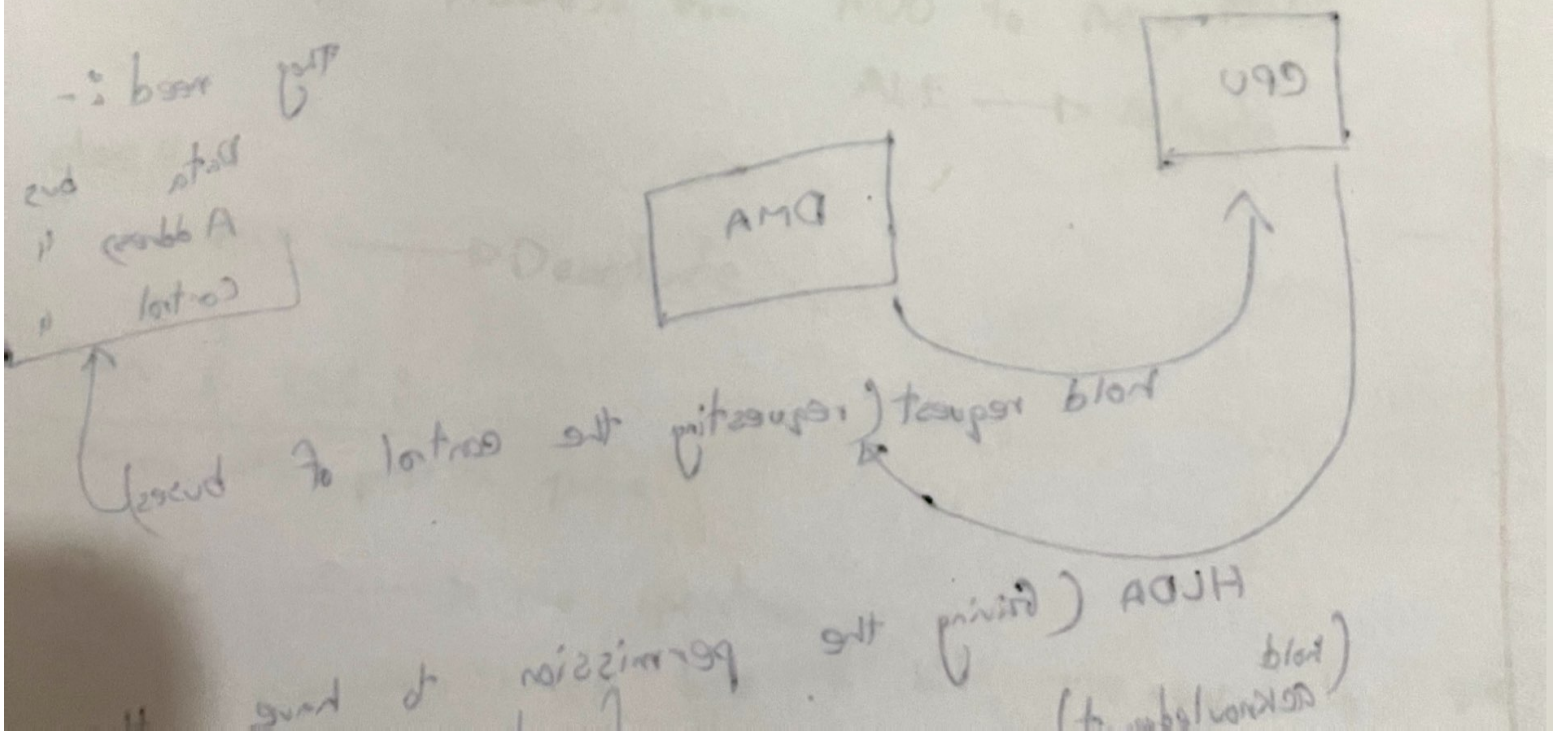
Control system function (In INT. Architecture)

(i) Instruction decoding.

(ii) Sends memory read signal

(iii) " " write "

(iv) Controls the movement of data between CPU register, ALU and I/O devices.

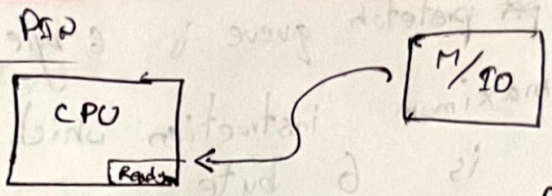


Subject : Time :

INTR

(i) Serve
(ii) Wait
(iii) Deny

READY PIN



(i) When reading a data we need ready pin.

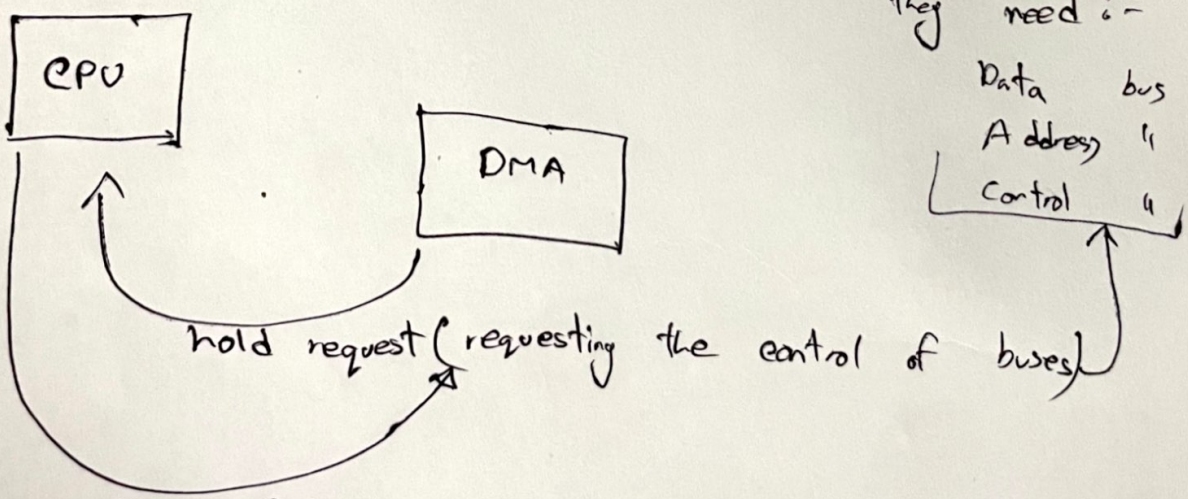
Ready pin is controlled by M/IO device. When the M/IO stays busy it makes the "ready" pin inactive

RESET PIN :-

This pin resets everything every false signals etc.

Minimum & Minimum mode.

HOLD & HLDA INPUT :-



M / $\overline{IO}$ output :-

For 1 ⟶ Memory activates

For 0 ⟶ IO activates

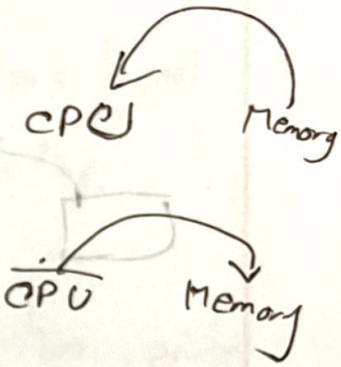
$\overline{WR}$ output :-

activates when writes.

DT / $\overline{R}$ output :-

Data reading ⟶ 0 [Data receives]

Data writes ⟶ 1 [Data transmits]



DEN

If there is any data flow in data bus. [A_{15} - A_{00} এবং Address latch activate.]

ALE output :-

If there is Address from AD0 to ~~AD9~~ AD15

ALE ⟶ Activate

else :-

ALE ⟶ Deactivate.

QS0 & QS1 :-

controls the prefetch queue

0 0 ⟶ No operation

0 1 ⟶ First ~~byte~~ 8 bit instruction goes from memory to queue

1 0 ⟶ ~~Deco~~ Queue to decoding.

1 1 ⟶ Brings the consecutive instruction of the 2nd instruction.

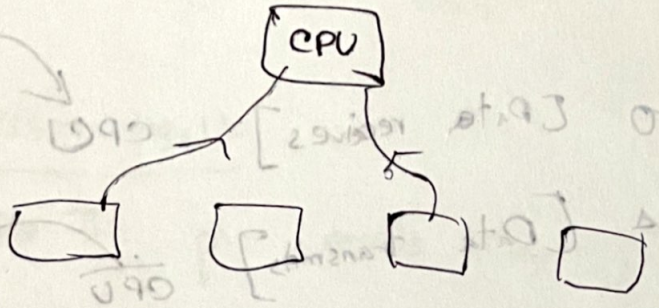
Subject :

Date :
Time :

Lock output :-

all interrupts are masked.

$\overline{RQ_0}$ / AT_0	$\overline{RQ}/\overline{AT_1}$



30 marks

↳ Q5 marks

→ Understanding the internal architecture

Implied addressing :-

CPU — Memory location

MOV DX, [DX + DI]

Memory Data to CPU

⇒ Read

Subject :

Date :

Time :

INTERNAL ARCHITECTURE

Why the microprocessor (8086) is divided into 2 BIU and EU unit?

→ Pipelining.

* BIU → fetch the instruction from the memory

Why there are two calculation unit (ALU & PAΣ) not 1?

→ Because, we want to execute two things at a time.

BIU characteristics:-

(i) fetch the next instruction

(ii) Calculte the physical address

* IP → instruction pointer contains the offset address

CS, DS, SS, ES gives the segment address & IP gives the offset address. Both of them combines to calculate the physical address.

All future instructions come into the prefetch queue

↳ BIU waits for 2 bytes to become empty in the prefetch queue * in order to send new instruction.

Why? → Cuz, 8086 is a 16 bit µp. It can send 16 bit in 1 S and 2 bytes means 16 bit.

Subject : Time :

ADD, BL, CL [How this instruction gets executed]

~~First~~ step-1 :- CS & IP together combinedly forms the PA.

step-2 : Fetches the instruction stored in the memory.

step-3 : Transfers the feteched instruction in the prefetch queue thragh data bus.

step-4 : then instruction is transferred in the control system where the it is decoded into OPCODE.

step-5 :- Then the ~~other time~~ sig control system signals the registers to send stored data to the ALU for calculation.

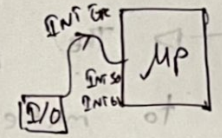
step-6 : The together with the fetched numbers and oper instruction OPCODE, the calculation gets done in the ALU.

Subject : INTERRUPT (RRH Online) Date : Time :

"When microprocessor works on a certain task and disrupts the task of the μp."

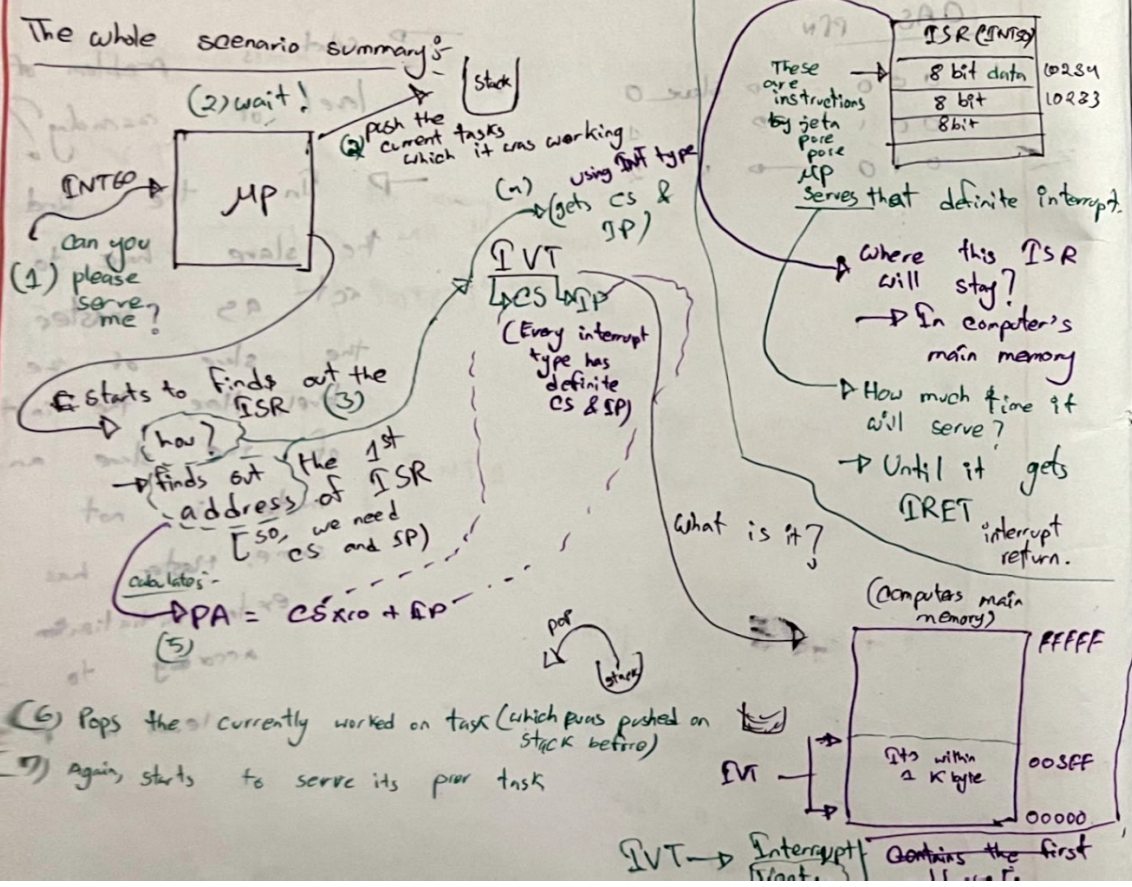
"Every interrupt type has its own ISR"

→ The type interrupt tries to find out its ISR. how?



ISR: Interrupt Service Routine (It is a kind of manual)

The whole scenario summary:-



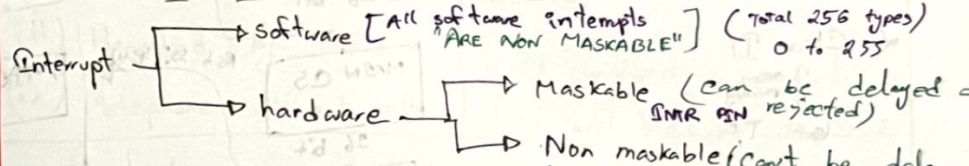
Subject : "Interrupt vector & Interrupt pointer is the same" Date : Time :

How the processor might be interrupted?

(i) By external signal [keyboard, mouse, Printer] [hardware]

(ii) By internal signal generated by special instruction. [software]

(iii) By internal signal generated by die to exceptional condition, which occurs while executing an instruction. [10/0] [Software]



Calculation to go to IVT:-

INT 60 →

Step-1

60×4 = (240)d → convert → (00F0)h

(INT Type)×4 = ( )d → convert → ( )hex

IP low byte address

Step-2

CS = 1062 h

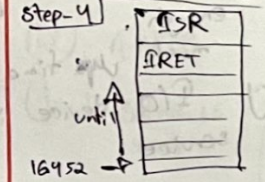
IP = 5832 h

Step-3

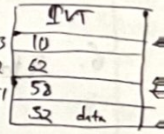
PA = CS×16 + IP

= 16452 h.

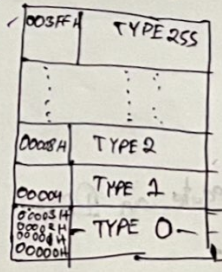
Step-4



Given



[ZOOM IVT VIEW]



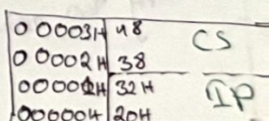
jekhane shobgula interrupter 1st address er bottom er CS and IP dekha ache. (FUNCTION)

CS → Base Add

IP → offset

CS = 4838 H

IP = 3220 H



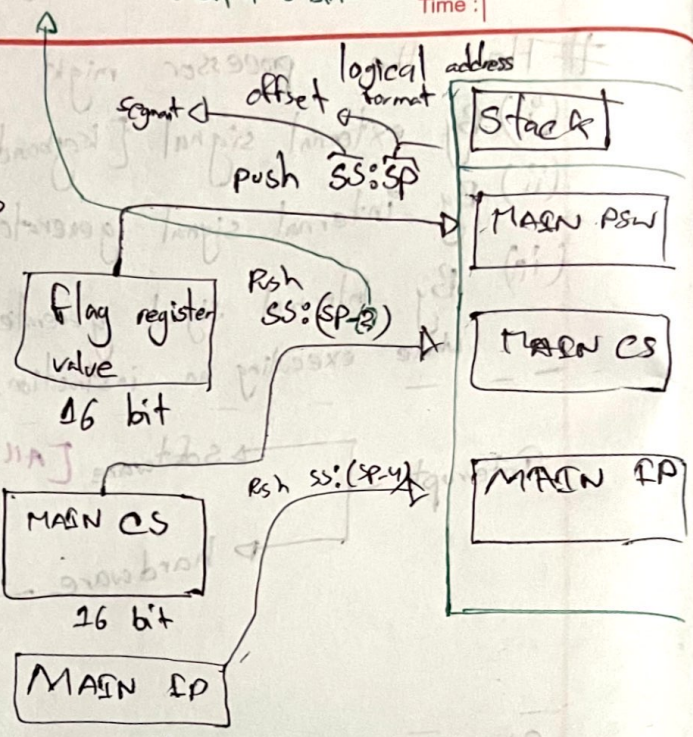
Subject :

Date :

Time :

How Interrupt Is Served ?

(2) how the microprocessor push the current tasks in stack?



Summary :-

1. Push flags
2. Clear IF
   ↕ occurs parallely
3. Clear TF
4. PUSH CS
   ↕ INTO STACK
5. PUSH IP
6. Fetch ISR address
7. POP IP
8. POP CS
9. POP FLAGS

Why Interrupt Is Necessary ?

Before interrupt there was polled system

- not fast enough
- waste too much up time by asking every I/O device if they needed service

Q. why two memory hops required to execute an ISR.

Subject: I/O Time:

POLLING	INTERRUPT
(i) Simple, but slow	(i) fast, but more complicated
(ii) processor regularly checks I/O device to see if it needs attention	(ii) processor is notified by I/O device when device needs attention
(iii) Handled by CPU	(iii) Handled by interrupt-handler
(iv) telephone without bells	(iv) Telephone with bells.

What if two or more interrupt occurs at the same time? → Higher priority interrupt served first.

INT TYPE	Priority
(i) Divide Error, INTn INTO	Highest ↑
(ii) NMI	increasing priority
(iii) INTR	
(iv) SINGLE STEP	lowest

But in the slide why DIV was pushed into when NMI interrupt came at 2? DIV NMI right?

Because NMI is non maskable

Subject : PROGRAMMABLE INTERRUPT CONTROLLER (PIC)

Date :
Time :

Connected in INTR pin of MP with the PIC's INT pin.

Why do we use it?
→ Can handle multiple interrupt requests simultaneously from various devices. Before, we only could take 1 request.

device of PIC :

IRR (Interrupt Request Register) : Receives request from the 8 INT pins. [INT0 - INT7] from various devices.

Priority Resolver :- Resolves the priority between the interrupt. Which one should be served first.

(INTERRUPT SERVICE) [For PIC only]
REG ISR : Tracks which interrupt the CPU is currently serving.

INTERRUPT MASK REG (IMR) : If we want to avoid or mask any interrupt.

Q. → How the pins value changes in IRR, ISR, IMR

Q. → Calculations of 1st ISR address

Q. → How interrupt is served

Q → Slide [22-27]

Q. - when more than 1 interrupt occurs how will it be handle the situation.

Subject :

Date :

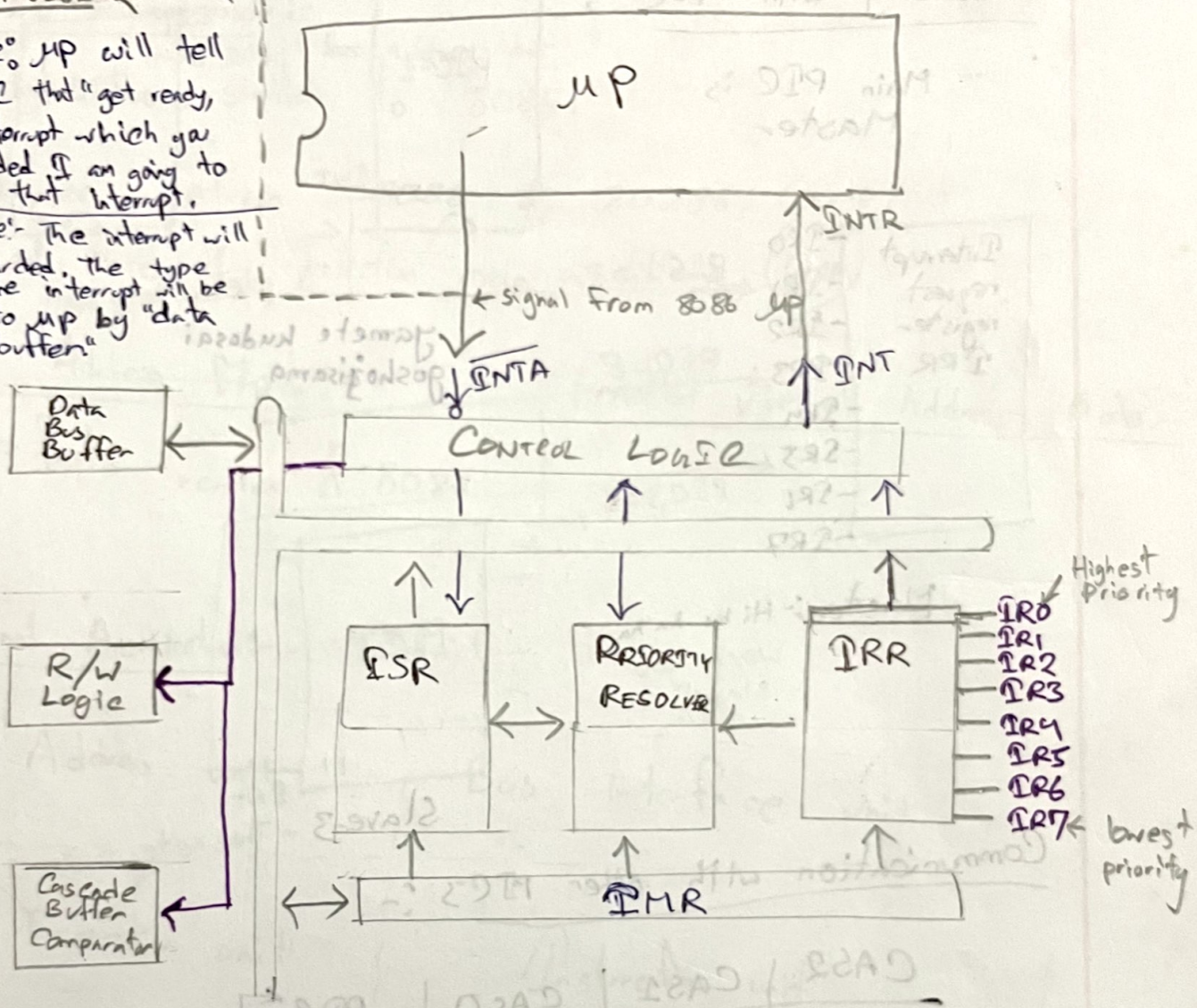
Time :

MOV [ ], AX MOV AX, [ ]

2 PULSE

1st pulse:- µP will tell to PIC that "get ready, the interrupt which you forwarded I am going to serve that interrupt."

2nd Pulse:- The interrupt will be forwarded, the type of the interrupt will be sent to µP by "data bus buffer."



IRR : 0 0 0 1 0 0 0 1 (IR7 IR6 IR5 IR4 IR3 IR2 IR1 IR0)

ISR : 0 0 0 1 0 0 0 0

IMR : 0 0 0 0 0 0 0 0

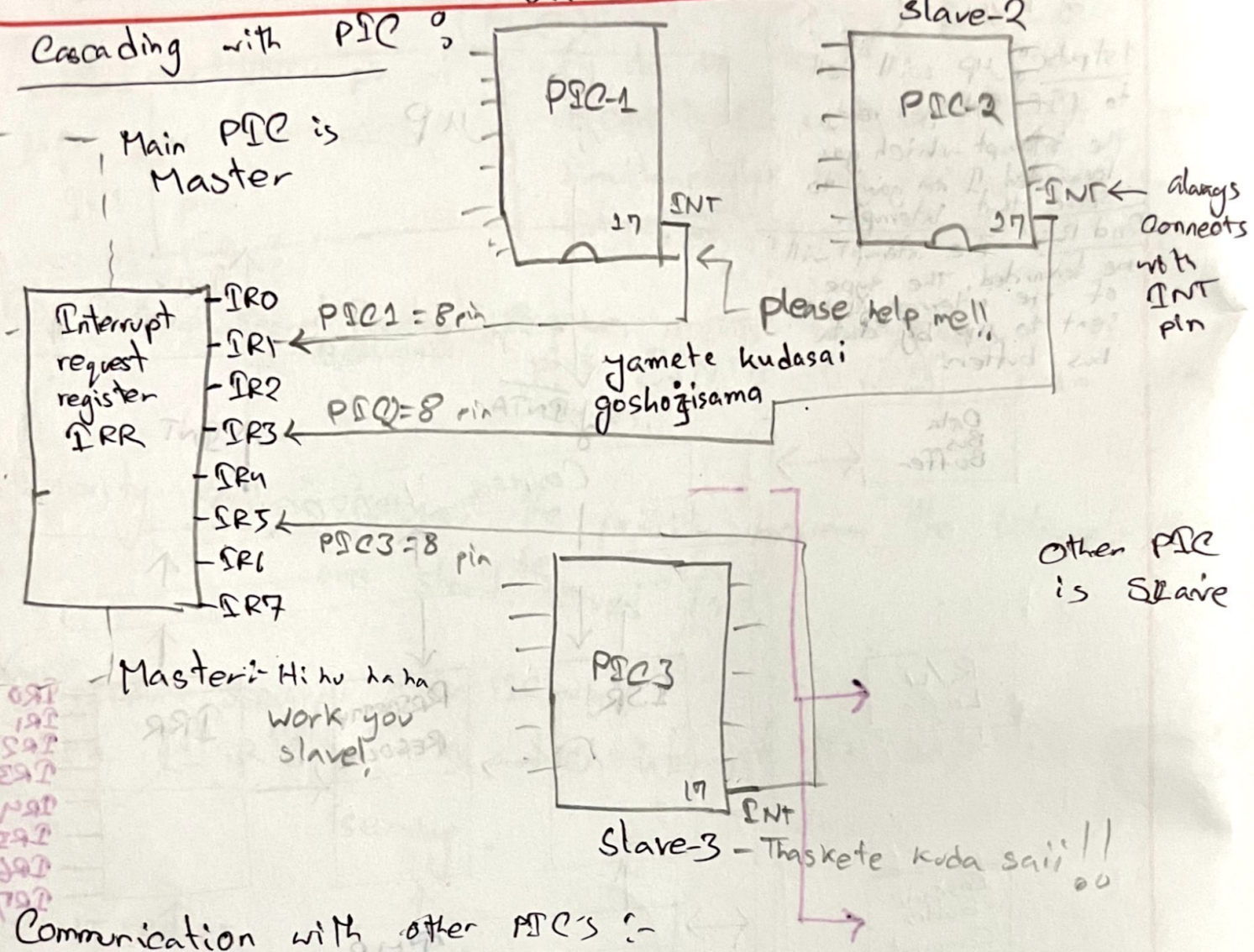
ISR ;

IMR ; IRR ; IR4 IR7 IR5 IR4 IR3 IR2 IR1 IR0

Subject :

Time :

Cascading with PIC :-



Communication with other PIC's :-

CAS2	CAS1	CAS0	PIC ← the name of the "slave" which is working or active!
0	0	0	PIC-1
0	0	1	PIC-2
0	1	0	PIC-3
0	1	1	PIC-4
1	0	0	PIC-5
1	0	1	PIC-6
1	1	0	PIC-7
1	1	1	PIC-8

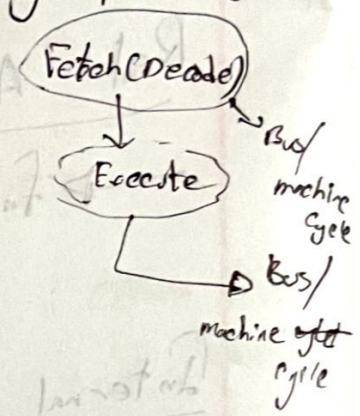
Subject :

Time :

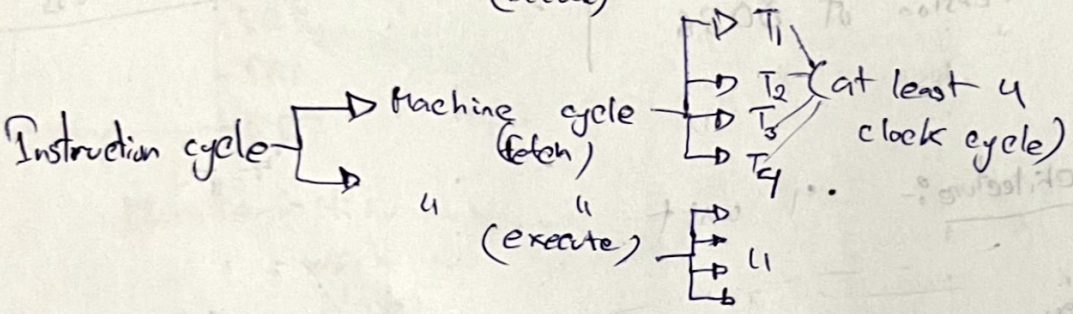
Timing Diagram

☆ The time a µp requires to complete fetch-decode-execute operation of a single instruction is known as instruction cycle

☆ Instruction cycle contains one or more machine cycle / Bus cycle.



Instruction cycle = Fetch cycle (Decode) + Execute cycle



☆ We are using 8284A circuit to generate clock cycle which is connected with CLK(g) pin of the µp.

☆ Time for 1 clock state = $\frac{1}{\text{Frequency}}$

☆ $T_{on} + T_{off}$ = 1 clock state duration.

"8088 µp by default works with 33% duty cycle

☆ Duty cycle = $\frac{T_{on}}{T_{on} + T_{off}} \times 100$

$A_{D_0} - A_{15}$ pins ⟶ carry Address + Data

Subject : Time :

=> Specific defined actions occur in each "T" state.

T_1 → address

T_2 → (Mem/IO, read/write)

T_3 → Data is sepplied

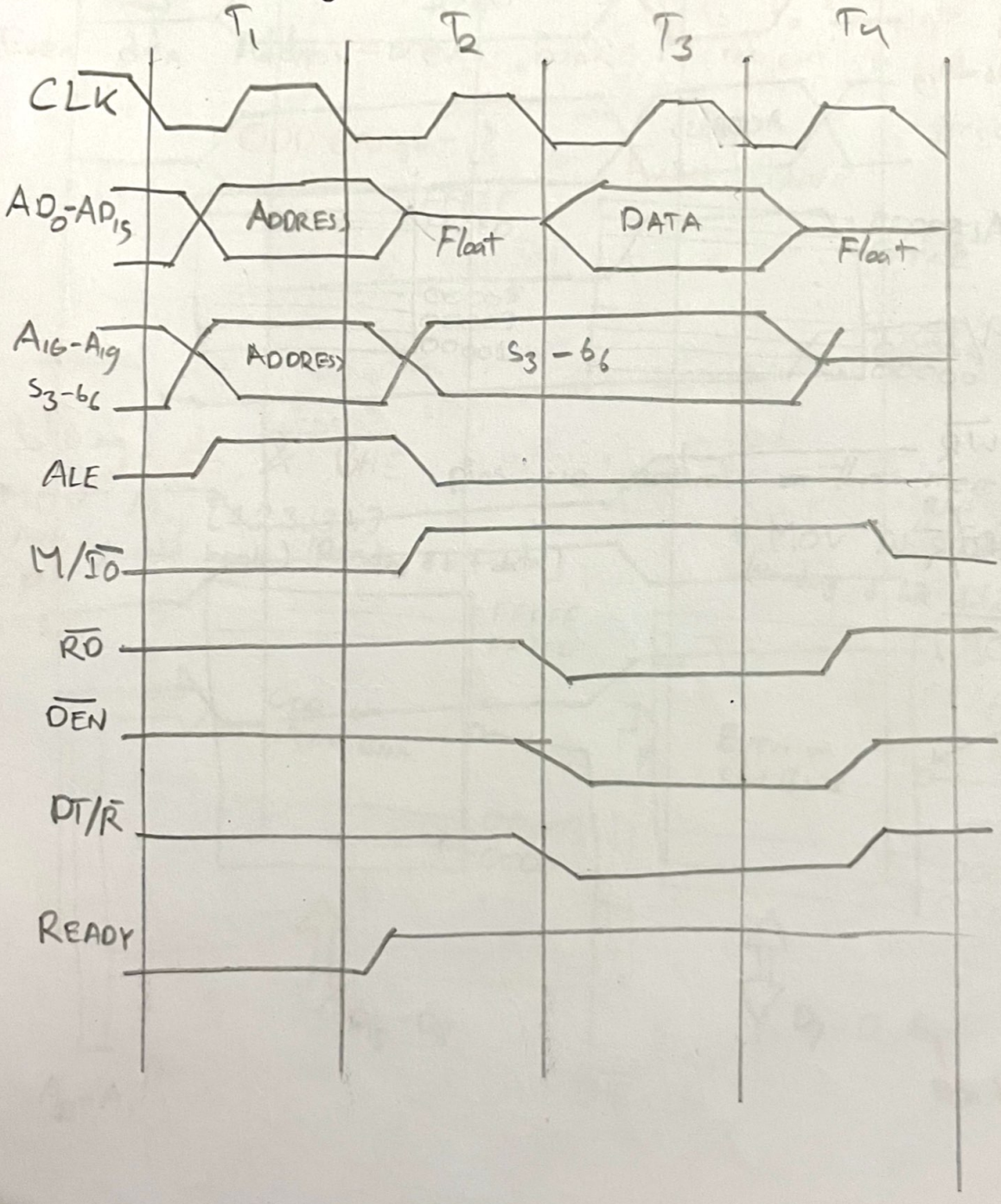
T_4 → Control signal removed

READ CYCLE FROM MEMORY

$DT/\bar{R}$

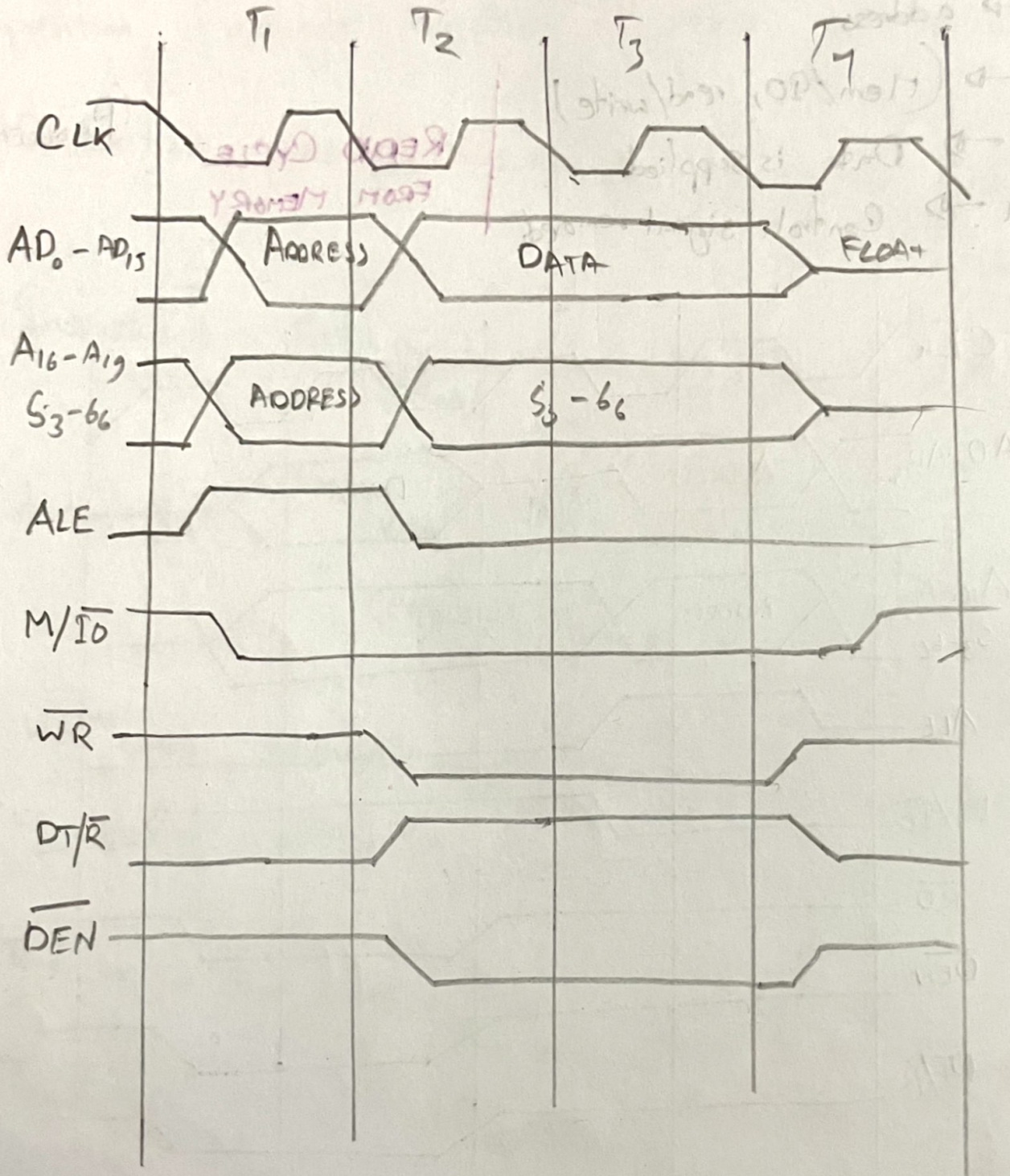
$\bar{RD}/\bar{WR}$

(PROBLEM) :→ $\overline{DEN}$



Subject :

WRITE CYCLE FROM I/O



Subject :

Date :
Time :

MEMORY BANKS

☆ Address bus => 20 bit
☆ Data bus => 16 bit

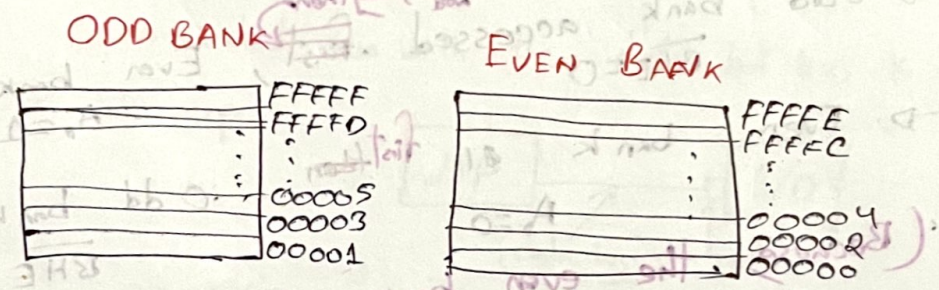
"To read/write : 1 byte / 8 bit data
1 bus cycle required"

1 bus cycle ≡ 4 clock state

2 bus cycle / 16 bit data ≡ 8 clock state

Odd data Address → odd bank
Even data Address → even bank

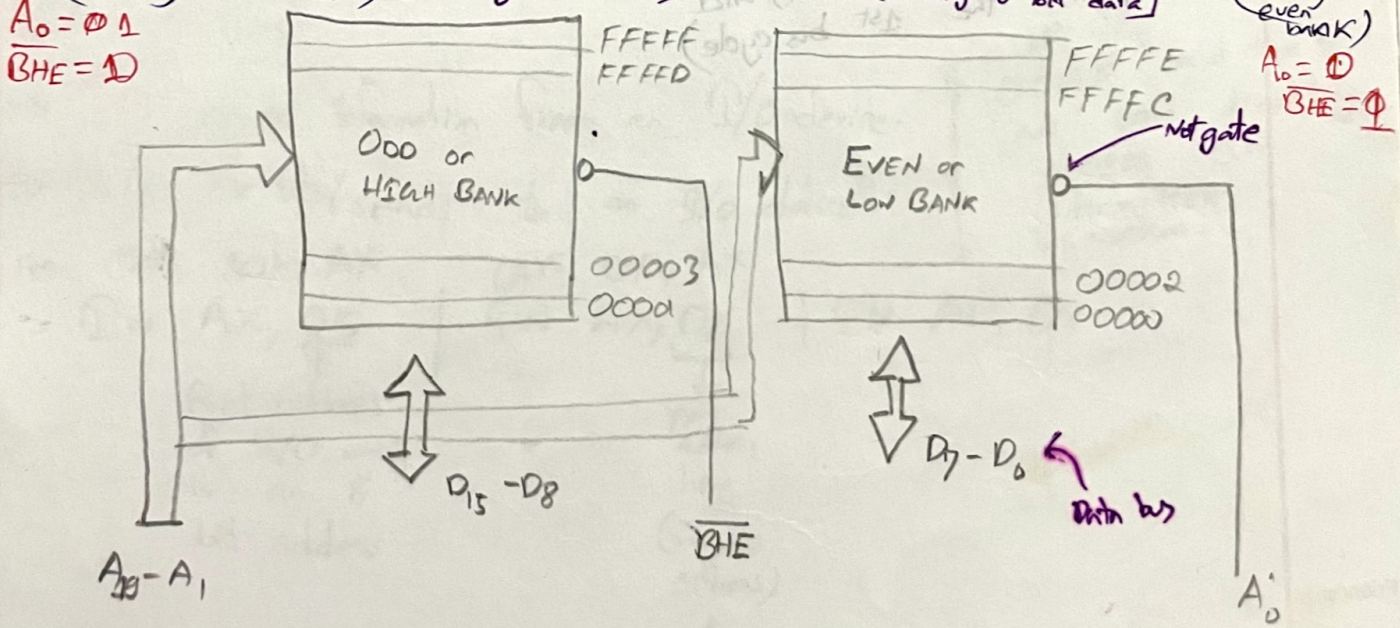
"Objective of memory banking is to complete 16 bit data retrieval in 1 bus cycle."



☆ Using A_0 & $\overline{BHE}$ pins we control the memory banks.

MOV AL, [12333h]
(Accessing odd bank) [Reading 8 bit data]

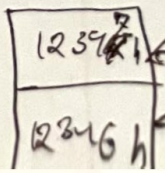
MOV AL, [23456h]
(Reading 8 bit data) (Accessing even bank)



MOV AX, [12346h]
16 bit

$A_0 = 0$
$\overline{BHE} = 0$

Reading 16 bit data starting from even address [aligned word]



Q. Pin values

Friendship

Subject :

Date :
Time :

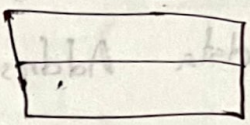
Benefits of memory banking :-

(i) We can access odd bank / even bank / both

When minimization of bus cycle in memory banking fails?

MOV AX, [13457h]

Reading 16 bit data starting from odd address] 2 bus cycle [Unaligned] then



→ odd bank accessed first, Even bank first
$\overline{BHE} = 0$ $A_0 = 1$ = 1 bus cycle

→ Even bank 01 first then, : odd bank
$A_0 = 0$ $\overline{BHE} = 1$ = 1 bus cycle

:(Because the even bank gets executed first because $A_0 = 1$)

= 2 bus cycle

↳ (As a result, takes data from only odd bank in the first and completes the 1st bus cycle)

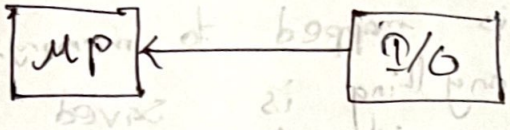
Subject :

BASIC I/O

Date :
Time :

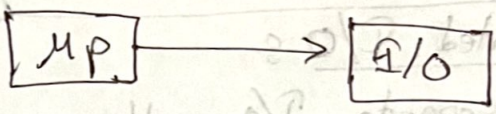
Input device :-

→ Recieve data from peripheral



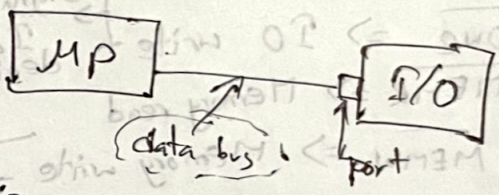
Output device :-

→ Receive data from processor



To communicate with the I/O device we need $AD_0 - AD_{15}$ pins

Data is communicated with the data bus b/w I/O device & μp.



I/O device serves two main purposes :-

(i) To communicate with the outside world.
(ii) To store & transfer data.

I/O instructions :-

→ IN reads information from an I/O device.

→ OUT writes/sends to an I/O device.

Here, OUT 30h, AX

IN AX, 25
↓
Port address of I/O, which is an 8 bit address

port address can be → 8 bit, 16 u

OUT DX, AX

IN AX, DX
↓
port address, here (variable address)
↑
can store both 8 to 16 bit

IN AL, DX

UP to 00FF address we can direct access an instruction.

Subject :

Date :
Time :

Accessing I/O device :-

(i) Memory mapped I/O
(ii) Isolated I/O

Memory mapped I/O :-

→ The port address ~~is~~ of the I/O device is stored in the memory.
So, I/O is mapped to memory.
→ If anything is saved in that location it gets transferred to I/O device. Problem :- some amount of memory gets wasted.

Isolated I/O :-

→ seperate I/O address space
→ Common data &, address bus for I/O, CPU & memory
→ $\overline{IORC}$ => IO read, $\overline{IOWC}$ => IO write → signals for I/O device
→ $\overline{MEMR}$ => Memory read
→ $\overline{MEMW}$ => Memory write → signals for memory

Benefits : Memory isn't getting wasted

Problem :- seperate signals needed to interact with I/O

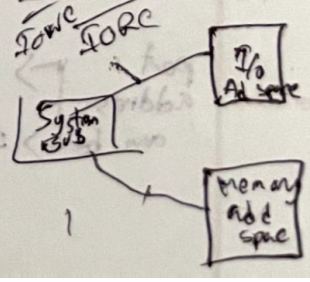
* IN/OUT need to used for data transfer

I/O Data Transfer Techniques :-

(i) Programmed I/O :-

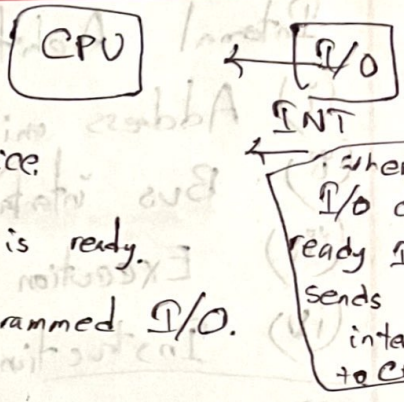
(a) In this case if the CPU needs something from the I/O and the I/O device is busy for some other work. Then CPU waits for the I/O to respond by leaving all of its tasks.
(a) CPU waits for I/O device to complete the operation if its not ready
(b) Programmed I/O wastes CPU time
(c) CPU performance degrades.

Isolate I/O



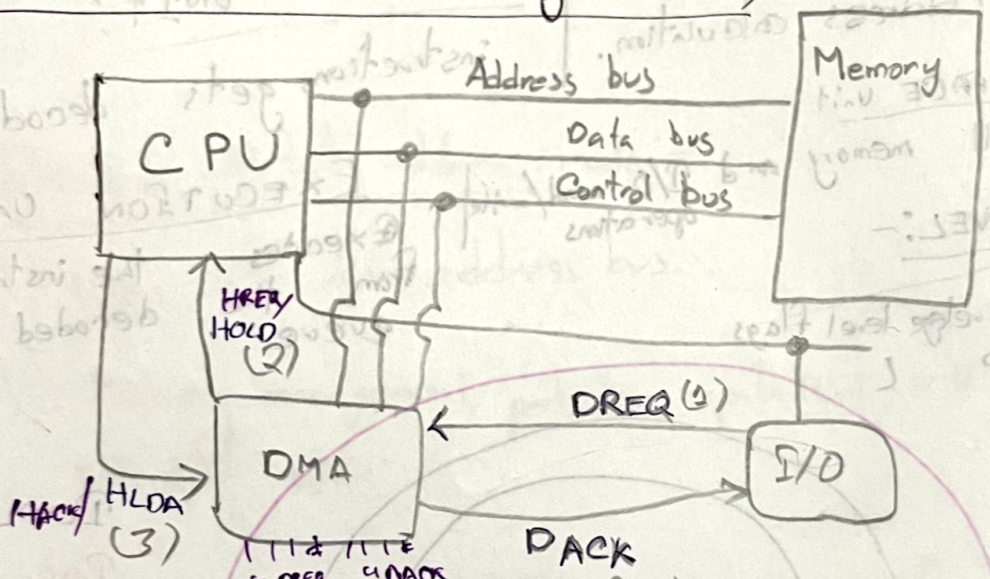
Subject : Time :

(i) Interrupt Driven I/O:-



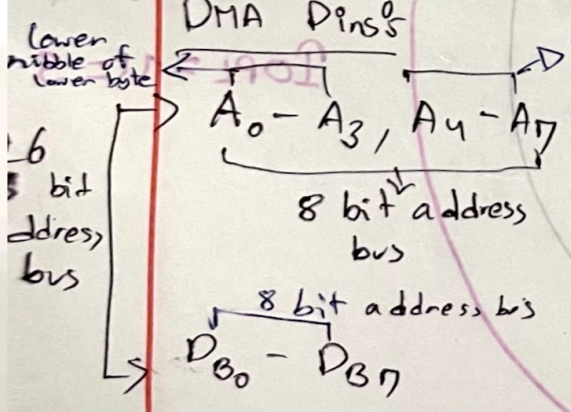
(a) CPU does not continuously wait for I/O device.
(b) I/O device itself alert the CPU when it is ready.
(c) CPU performance is better than programmed I/O.

(ii) DMA (Direct Memory Access):



(i) DMA controller can perform the task of transferring blocks of data between memory and peripherals, relieving the CPU from the burden of transferring data.
(ii) CPU just issues command for data transfer while DMA transfers the data.

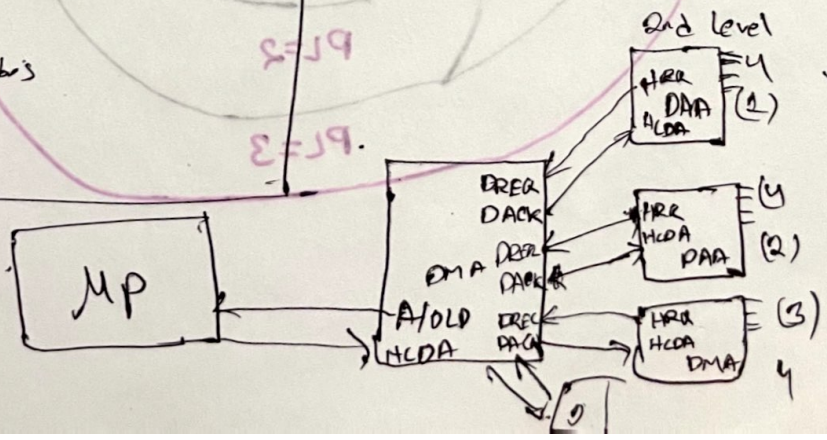
DMA Pins:



1 DMA connects friendship 4 I/O device at a time

DMA Cascade Mode:

Every DMA has 4 channel, 4 DREQ pin connect 16 I/O device at max



Subject : 80286 Microprocessor

Date :
Time :

Internal Architecture is divided into 4 unit:-

(i) Address unit
(ii) Bus interface unit
(iii) Execution unit
(iv) Instruction unit.

ADDRESS UNIT:-

Physical Address calculation.

INSTRUCTION UNIT:-

instructions gets decoded

BUS INTERFACE UNIT:-

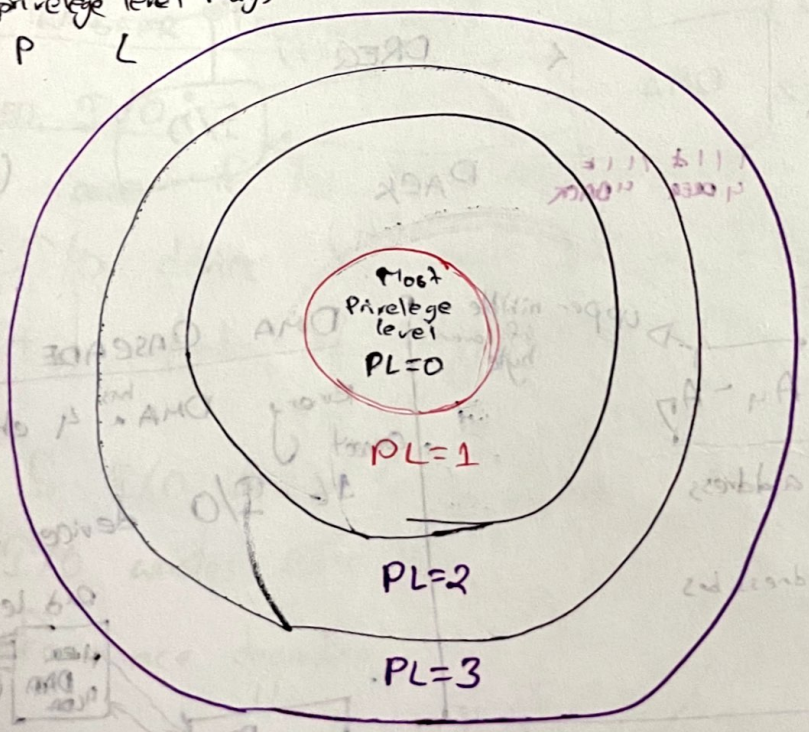
Performs all memory and I/O read/write operations

EXECUTION UNIT:-

Executes the instructions received from the decoded instruction queue

PRIVELEGE LEVEL:-

Input output privelege level flags
I O P L



IOPL = 00 = 0
IOPL = 01 = 1
IOPL = 10 = 2
IOPL = 11 = 3

Subject :

Date :
Time :

REAL ADDRESS MODE :-

(i) Uses 20 bit address bus.

(ii) It keeps off the protection / memory managent mechanism.

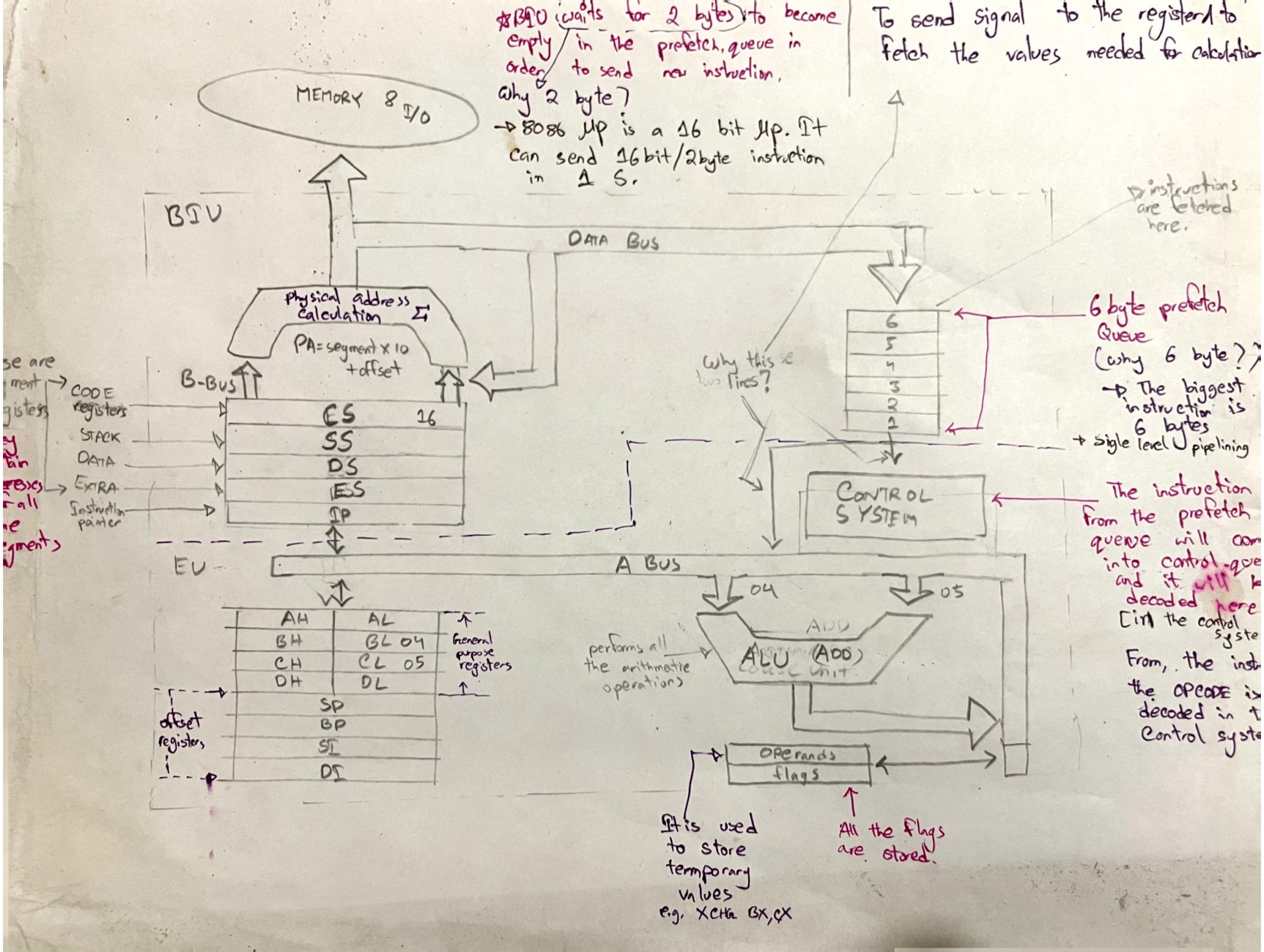
(iii) 6 time faster than 8086

Protected Virtual address mode :-

(i) Uses 24 bit address bus.

(ii) Memory management, protectim capabilities with the advanced instruction set.

(iii) IOPL and NT ~~get~~ tasks get assigned.


*BIU waits for 2 bytes to become empty in the prefetch queue in order to send new instruction.
Why 2 byte?
→ 8086 µp is a 16 bit µp. It can send 16 bit/2byte instruction in 1 S.
To send signal to the register to fetch the values needed for calculation
MEMORY & I/O
BIU
DATA BUS
instructions are fetched here.
physical address calculation
PA = segment x 10 + offset
B-BUS
CODE registers
STACK
DATA
EXTRA
Instruction pointer
ES
SS
DS
IES
IP
16
6 byte prefetch Queue
(why 6 byte?)
→ The biggest instruction is 6 bytes
→ Sigle level pipelining
Why this 2 lines?
CONTROL SYSTEM
The instruction from the prefetch queue will com into control queue and it will be decoded here (in the control syste
From, the inst the OPCODE is decoded in t control syste
EU
A BUS
AH AL
BH BL 04
CH CL 05
DH DL
general purpose registers
SP
BP
SI
DI
offset registers
performs all the arithmetic operations
ALU (ADD)
ADD
operands
flags
It is used to store temporary values e.g. XCHG BX, CX
All the flags are stored.