μp - L - 01

⊛ μp — in PC, mobile, remote → cause there is programming

— not in fan, life → no programming

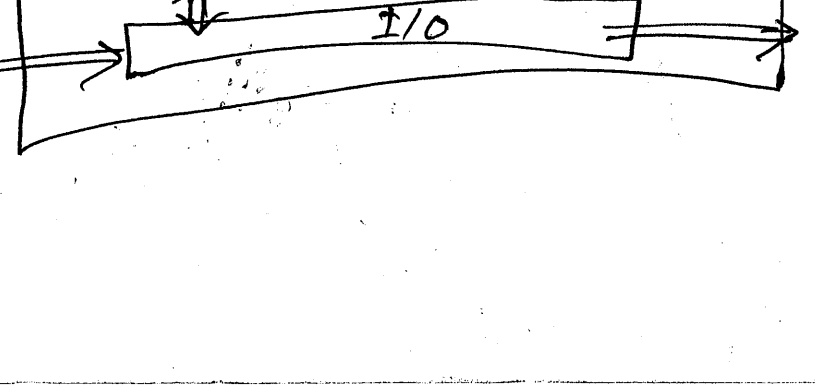
executes the program

⊛ Why not study i-7? Does a traffic light have a core i-7?

⊛ progress: 8085 → 8086 → 80186
286
386
⋮

⊛ μp in PC → CPU → inside the fan

Computer System



⊛ In 8086, memory is <u>RAM & ROM</u>, not the $\underbrace{\text{secondary memories}}$.

floppy disc
CD

⊛ Memory stores data < image, song, movie

↓

how ? series of 0s and 1s.

⊛ <u>Instruction cycle :</u>

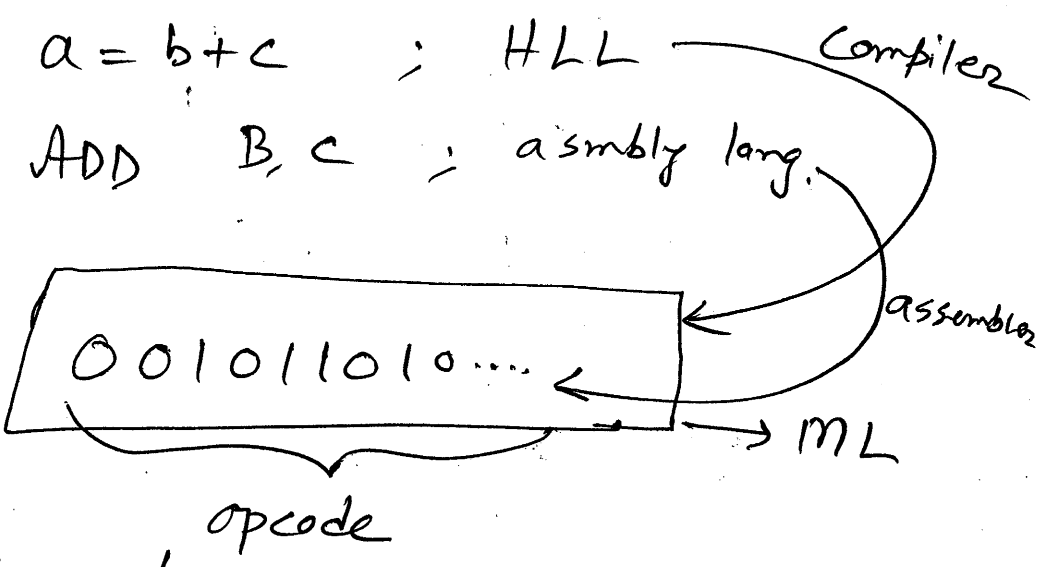
1) fetch instr. from memory

2) <u>Decode</u> :

fetched instr. is 0s and 1s.
understanding the 0s and 1s.

a = b+c ; HLL

ADD B, C ; asmbly lang.



3) Executing

Few basics

Dec	Hex	Bin
0	0 H	0
⋮	⋮	⋮
9	9 H	1001
	A	
	⋮	
	F	

μp uses Hex. system cause more information can be stored.

ex: For a 4-bit system,

decimal max 9999

hex max FFFF

so, more info. in hex system.

⊛ Hex to Bin

$$\begin{tikzcd} 3 \arrow[rrr, bend right] & 5 \arrow[rrr, bend right] & H \arrow[r] & 0011 & 0101 \end{tikzcd}$$

⊛

8 bit nums	16 bit nums
00 H	0000 H
⋮	⋮
FF H	FFFF H

Power of 2

$2^{10} = 1k$

$2^{11} = 2 \times 1k = 2k$

$2^{12} = 2^2 \times 1k = 4k$

$2^{13} = 2^3 \times 1k = 8k$

$\vdots$

$2^{20} = 1k \times 1k = 1M$

$2^{24} = 2^4 \times 1M = 16M$

$2^{30} = 2 \quad 1k \times 1k \times 1k = 1G$

$2^{40} = 1T$

$2^{32} = 4G$

Bus

$$\left.\begin{array}{l}\text{Addr. bus}\\ \text{Data bus}\\ \text{Control bus.}\end{array}\right\}\ \begin{array}{l}\text{transfers}\\ \text{0s and 1s}\end{array}$$

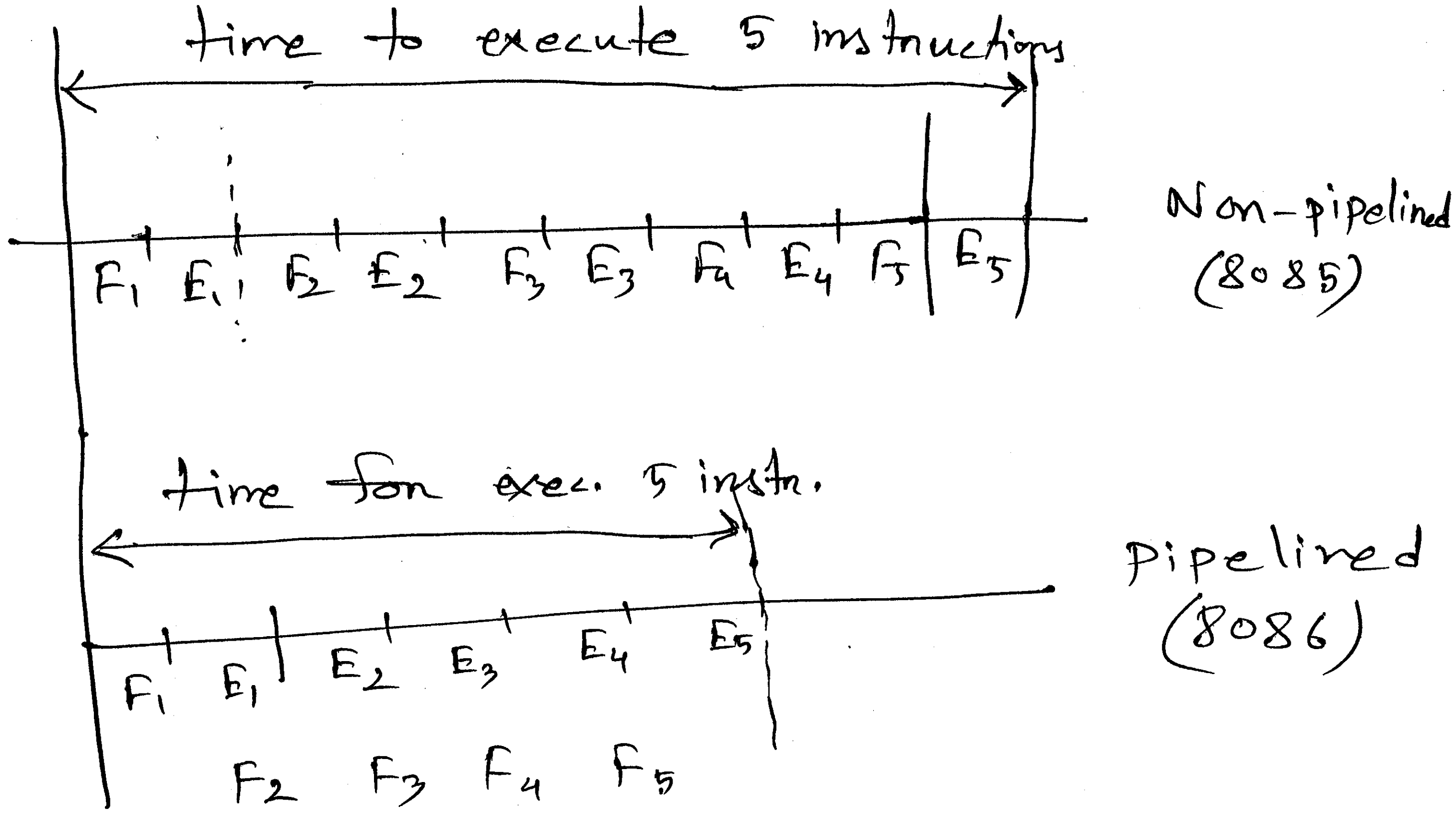
4 bit lines — 4 lines in bus

8 " " — 8 " "

L-02

Pipelining

- What makes up faster?
  why 2.6 GHz, 3.6 GHz these days?
- Ans is Pipelining
- 8086 → 1st processor to introduce pipelining



- That's why - 2 units in 8086 architecture : one for Fetching, one for executing

– this is 2 staged pipelining

Disadvantages :

1) If a instructions needs its previous one's output for execution, it can't start until the previous one is finished.

2) Branching : If instr. 1's execution tells us to go to instruction 8, and we fetch instr. 2 meanwhile, it has to be discarded. (ex: if-else)

Branch issue solution :

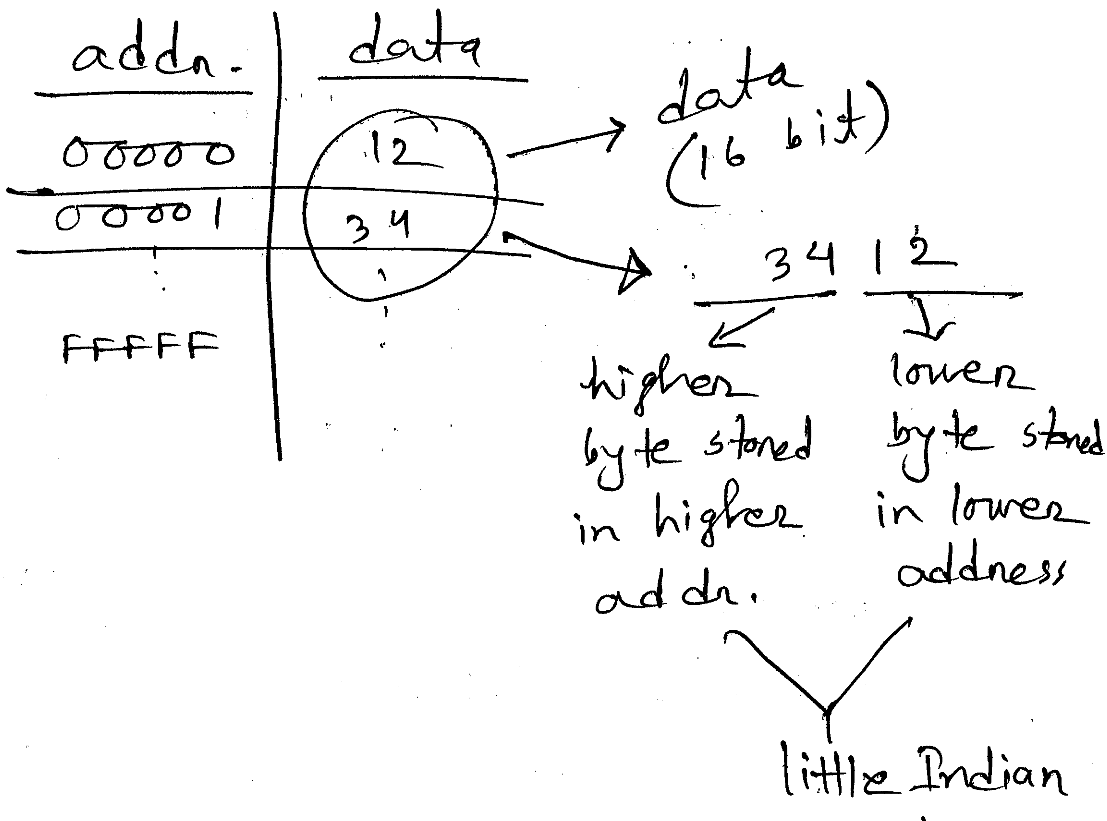
- Branch prediction algorithm
- decides whether it will branch or not based on previous many decisions.

(8086 doesn't implement it)

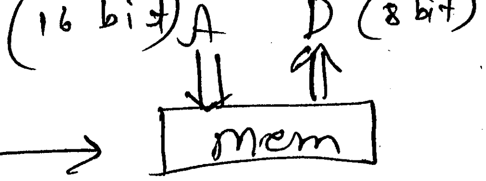
8086 memory banking

- 20 bit address bus
- so, memory size = 2^{20} = 1M
- 1 memory location contains 1 byte of data

ex:



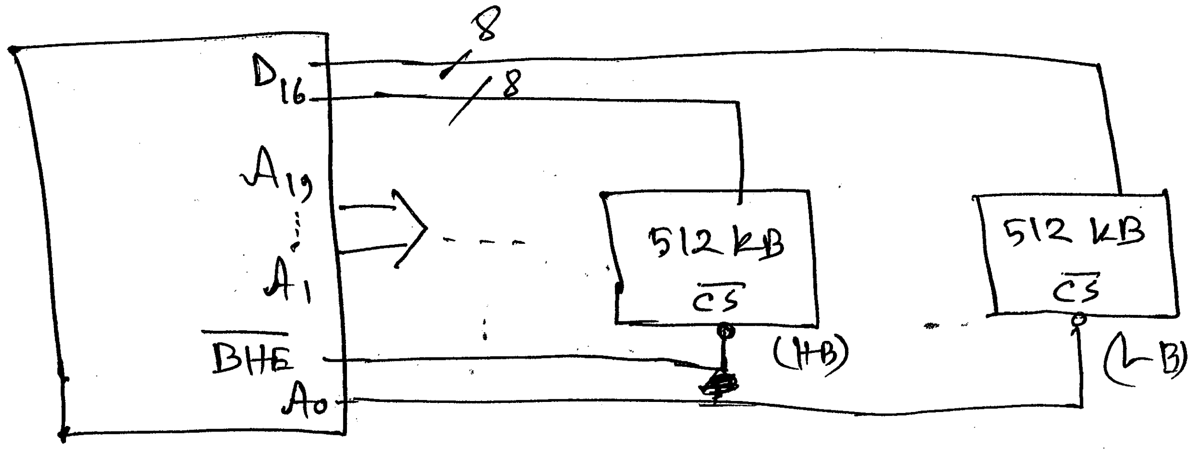
addr.
data
00000
12
00001
34
FFFFF
data (16 bit)
34 12
higher byte stored in higher addr.
lower byte stored in lower address
little Indian Rule



- problem ? ⟶ we want to transfer this data 3412 in one cycle. But up can't generate two addresses in one cycle.

- ~~Stupid soln~~ :

- soln : We want to access 16 bit i.e. 2 bytes of data in a cycle. So, have 2 memory chips.



1 MB

512 kB	512 B
odd bank (higher bank)	even bank (lower bank)
00001 H	00000 H
00003 H	00002 H
5	4
7	6
⋮	⋮
FFFFF	FFFFE

- We can choose addr 00000 H and 00001 H together. So do we can 00002 H and 00003 H.

- A_0 selects which bank to choose
- $A_1 - A_{19}$ selects the memory location. Same mem. location is selected in both banks. That's why 00001H and 00002H aren't selected together.
- $A_0 = 0$; LB selected

  $A_0 = 1$; LB not selected

  $\overline{BHE} = 0$; HB selected

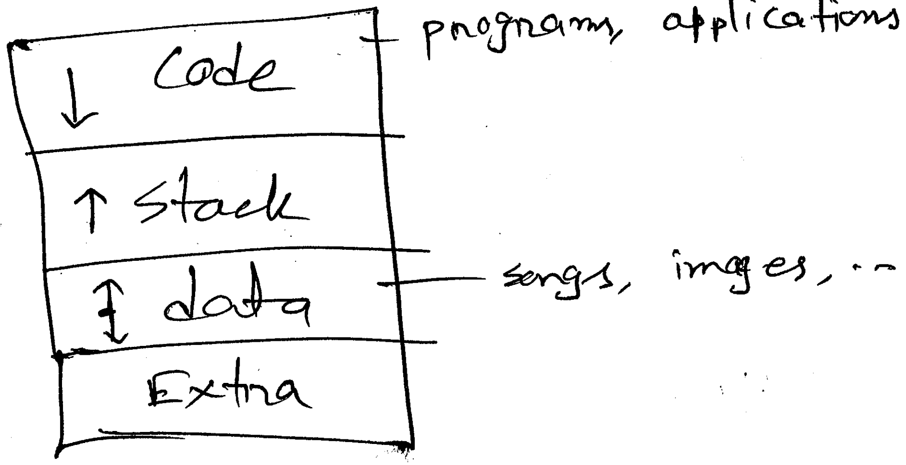
$\overline{BHE}$	A_0	operation
0	0	R/W 16 bit from both banks
0	1	R/W 8 bit from HB
1	0	R/W 8 bit from LB
1	1	None (idle)

that's the benefit of banking. we can access either LB/ HB/ both.

L-03

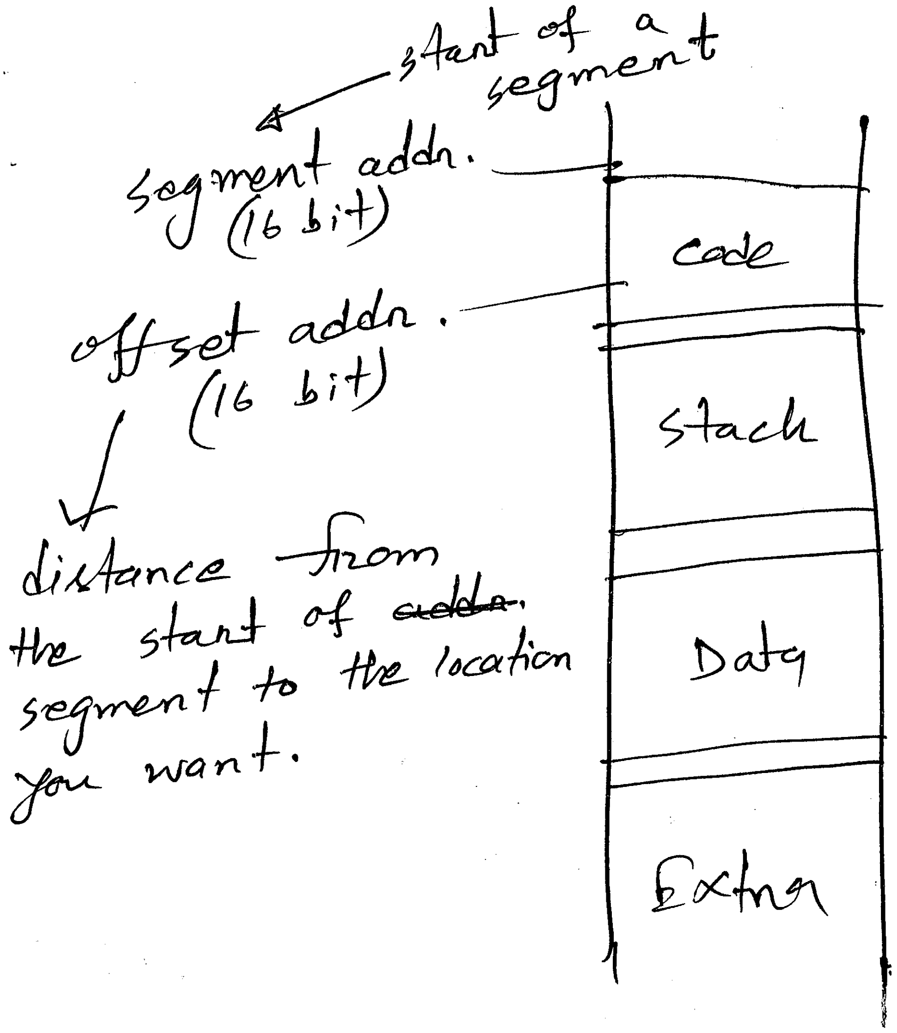
Memory Segmentation

- 4 segments



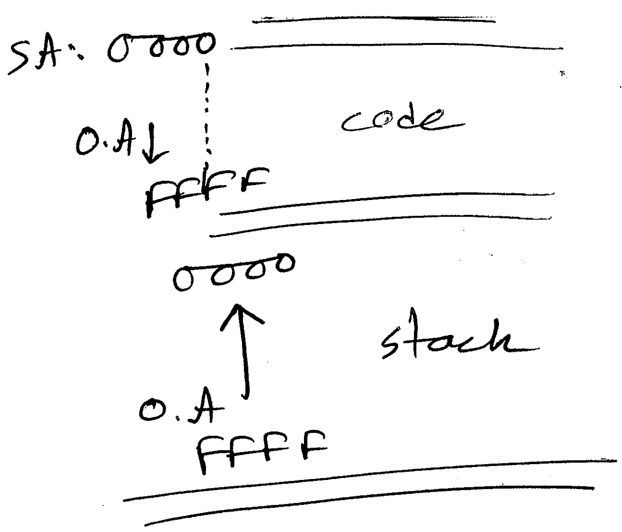
- this is the birth of concept of files. In 1970, no computer had files & folders. But files evolved and file is basically a modern version of segments.

- 1 MB memory ≡ 20 bit address
this 20 bit addr. is physical addr. or actual addr. But, while programming, we need a byte compatible address / 16-bit addr.



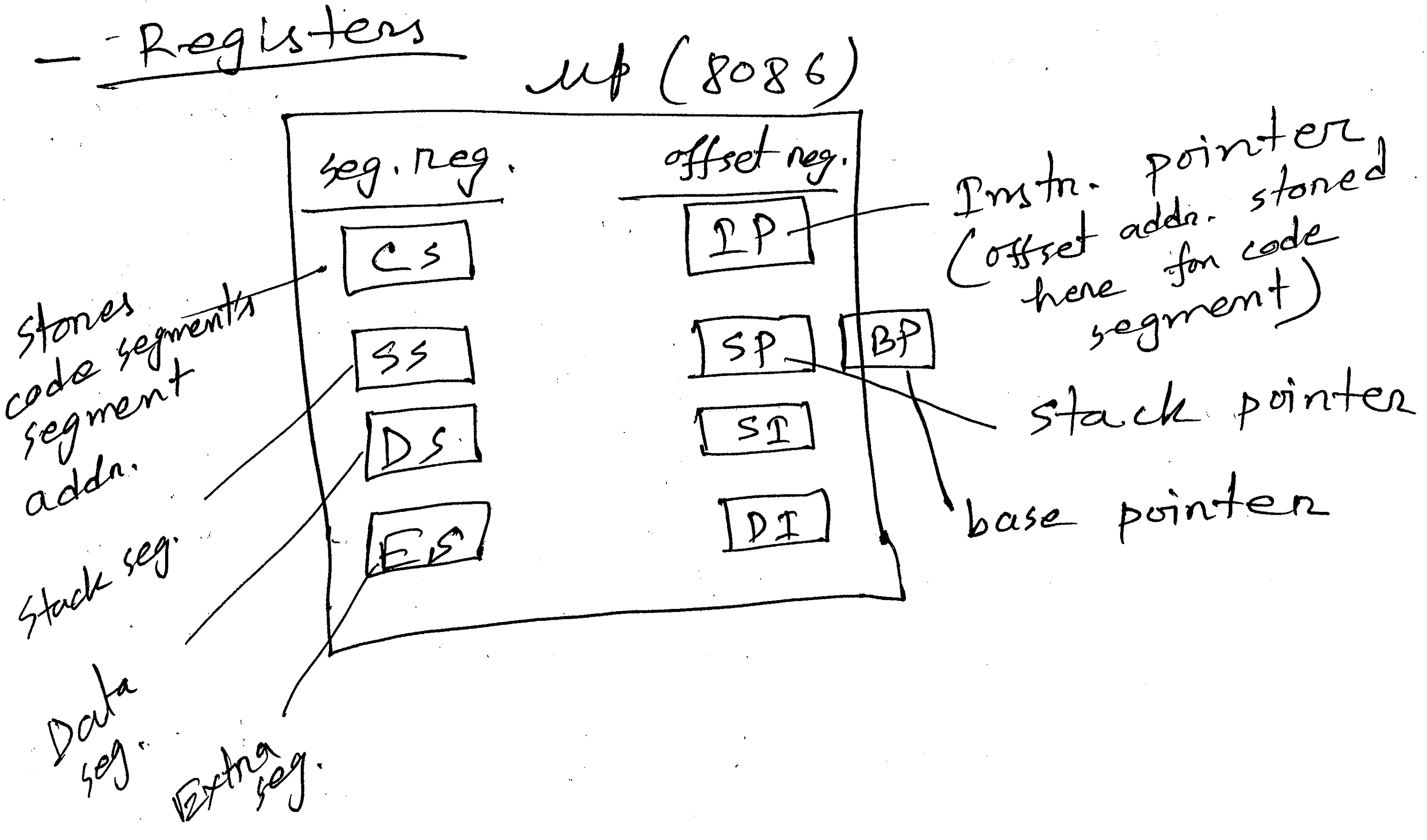
- every time, u don't give both SA and offset. you give SA at first, then give only offset address.
- 8085 didn't have segmentation. Why? (cause 16-bit address)
- Why segments don't overwrite? cause, if you keep increasing the offset addr., it will reach FFFF at one point and u can't go further. If you could, there'd

be overwrite.



- max. range of a segment
  0000H $\longrightarrow$ FFFF H
  max size of segment = 2^{16} = 64k
  (Not all segments are 64k)

- Registers



In short: CS — stores code segment's segment addr.

IP — code segment offset address.

- You give up the segment addr. and offset addr. It's the job of up to convert them to physical address cause up will send PA by 20-bit addr. bus to mem.

- PA = seg addr. × 10 + offset

$\quad$ = CS × 10 + IP

up does this calculation

- If SS = 3000 H, from what physical address, stack segment starts?

= 30000 H

- want to start stack segment

$$\begin{array}{lll} \text{at} \rightarrow & 52350 \rightarrow & \overset{\boxed{SS}}{5235} \\ & 31430 \longrightarrow & 3143 \\ & 52945 \longrightarrow & \boxed{??} \end{array}$$

(possible)

NO! ↓ need to be 10's multiple

- so, if a segment ends at 52340, the next segment can start at 52350. How?

$\boxed{5234}$ DS

$$\left.\begin{array}{l} \boxed{\begin{array}{l} 52340 \\ 52341 \end{array}} \\ \quad\vdots \\ \quad 49 \\ \quad 4A \\ \quad 4B \\ \quad\vdots \\ \quad 4F \end{array}\right\} \text{min size of a segment (16 byte)/(10 H)}$$

52350 ← next segment

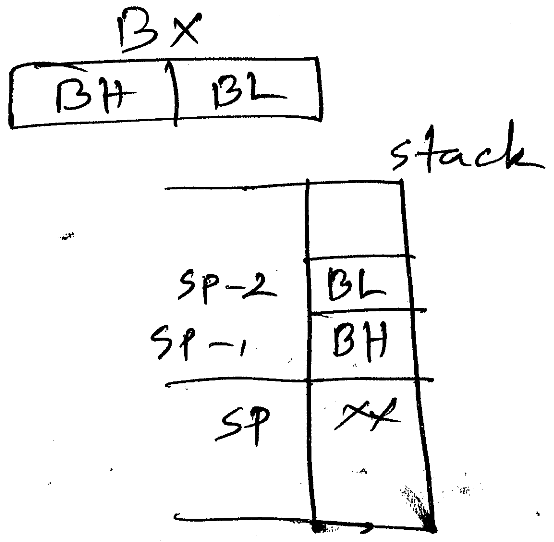
cause the next segment can't start at 52341 or 52342, ... need to be 10's multiple.

- SP and BP

- after push and pop, SP ↑ or ↓.
  SP points to top of stack.
  but BP is used ~~for~~ random access of stack.
- BP can point anywhere on stack.
- ex : incoming sms in phone.

- push , POP

push BX

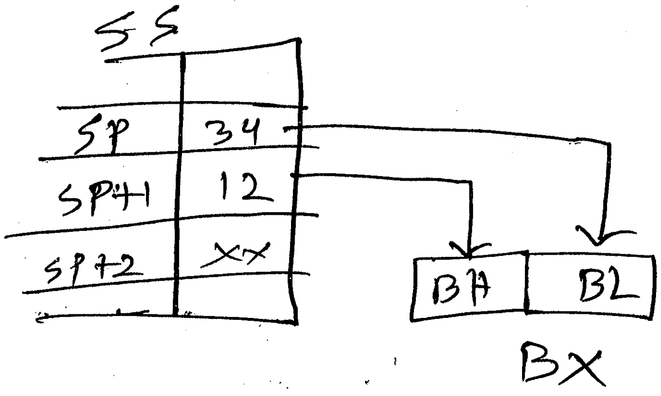


$SS:[SP-1] \leftarrow BH$

$SS:[SP-2] \leftarrow BL$

$SP \leftarrow SP-2$

POP BX



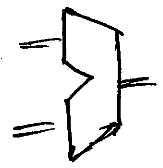
$BL \leftarrow SS:[SP]$

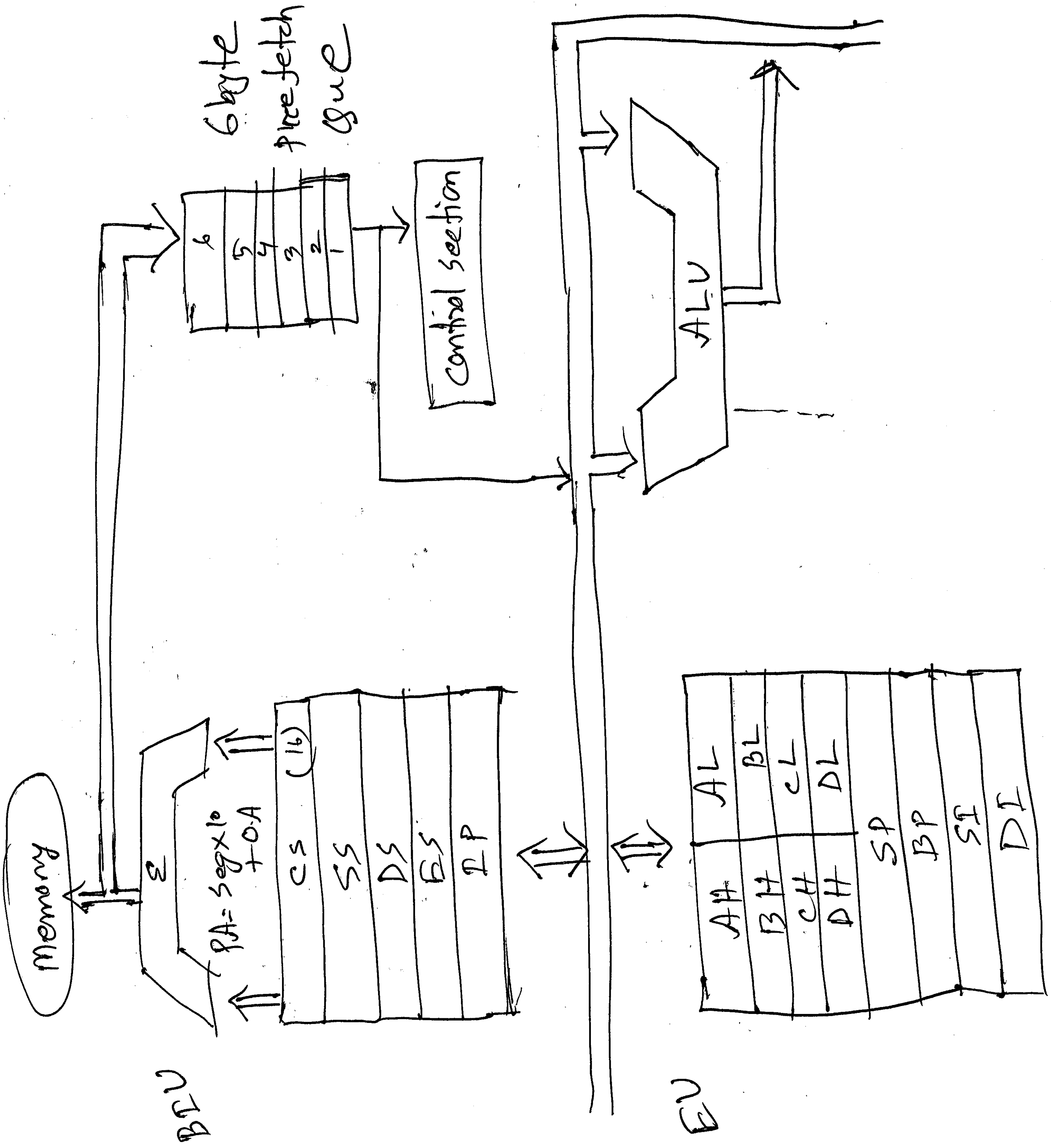
$BH \leftarrow SS:[SP+1]$

$SP \leftarrow SP+2$

L-04

Internal Architecture

⊛  : Arithmetic circuit



- Instr. Q is of 6 bytes not 6 instn. instructions can be of different sizes.
- When 2 bytes location ~~of~~ in Q is free, next 2 bytes of instructions are fetched. If
- 1 byte is free, nothing will happen.
- If a half of instn. comes within 2 bytes, any problem?
  - No, cause when it reaches bottom of Q, it will be a full instructions.

- control section → decodes the instruction

for $\boxed{\text{mov BL, 04 H}}$

control section triggers BL to load value 04 H

for $\boxed{\text{Add BL, CL}}$

control section sends add signal to ALU

(*) operand registers (16 bit)

flag registers (16 bit)

→ XCHG

$$\begin{tikzcd}
\text{BX} \arrow[r, bend left, "2"] & \text{CX} \arrow[dl, "1"] \\
\text{opd reg.} \arrow[u, "3"] & \text{temporary registers} \arrow[l]
\end{tikzcd}$$

Flags Registers

(*) Use slide

(*) sign bit – MSB → sign flag

- 8 bit unsigned numbers
  0 – 255

- 8 bit signed numbers

<u>Dec:</u> −128 - - - - 127

<u>Hex:</u> neg: – 80H- - - - - - - 01H (~~1~~000 0000 / 1111 1111 $\underbrace{}_{FF}$)

pos: 00H- - - - - - - - - 7FH (0000 0000 / 0111 1111)

Ⓐ sign flag gives wrong sign in case of overflow.

If OF = 1, sign flag is wrong

So, never directly check the sign flag.

<u>ex:</u>

```
   42 H        0100  0010
 + 23 H        0010  0011
 ------       -----------
   65 H        0110  0101
```

OF	SF	ZF	AC
0	0	0	0

PF	CF
1	0

```
   37 H        0011   0111
 + 29 H        0010   1001
 ------       ------------
   60 H        0110   0000
```

OF	SF	ZF	AC	PF	CF
0	0	0	1	1	0

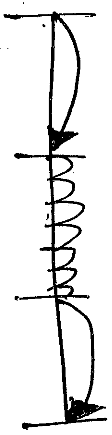
```
   42 H        0100  0010
 + 43 H        0100  0011
 ------       -----------
   85 H        1000  0101
```

OF	SF	ZF	AC	PF	CF
1	1	0	0	0	0

Ⓐ Control flags :

– controlled by us

TF : trap flag



TF = 1

← single stepping

TF = 0

IF = Interrupt flag :

1 : intrpt. enabled

0 : intrpt. disabled

DF : Direction flag

1 : auto dec

0 : auto inc

ex: string copy paste

L-05

Addressing modes

: manner in which operand is given

1) Immediate $\xrightarrow[\text{instn.}]{\text{data in}}$ ex: mov CL, 34H

2) Register $\xrightarrow{\text{data in reg.}}$ ex: mov CL, BL
INC BX

3) Direct $\xrightarrow{\text{addr. in instn.}}$ ex: mov CL, [2000H];
CL ← DS: [2000H]
mov CX, [2000H];
CL ← DS: [2000H]
CH ← DS: [2001H]

mov [2001H], CL
DS: [2001H] ← CL

4) Indirect $\xrightarrow{\text{address in reg}}$

up takes addr. from registers, then use that address to fetch data
So indirect.

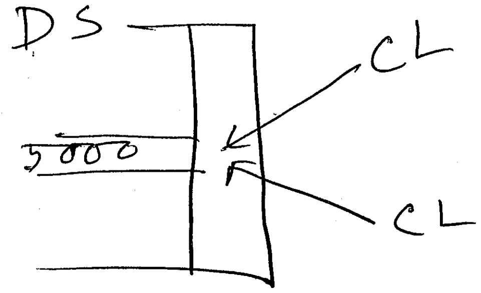
a) Register Indirect:

```
mov   CL, [BX] ;  CL ← DS:[BX]

mov   CX, [BX] ;  CL ← DS:[BX]
                  CH ← DS:[BX+1]
```

Classwork

①



direct

```
mov CL, [5000H]
```

indirect

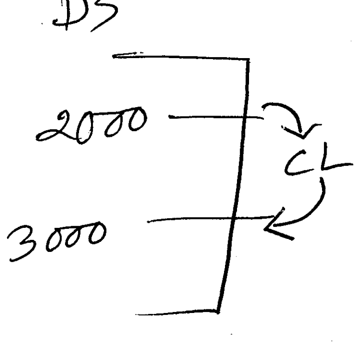
```
mov BX, 5000H
mov CL, [BX]
```

advantage:

$$\boxed{\begin{array}{l}\text{mov ~~B~~CL, [BX]} \\ \text{INC BX}\end{array}}$$

Accessing series of location { in a loop, we can read data from ~~5000H~~ then from ~~5001H~~, ...

②

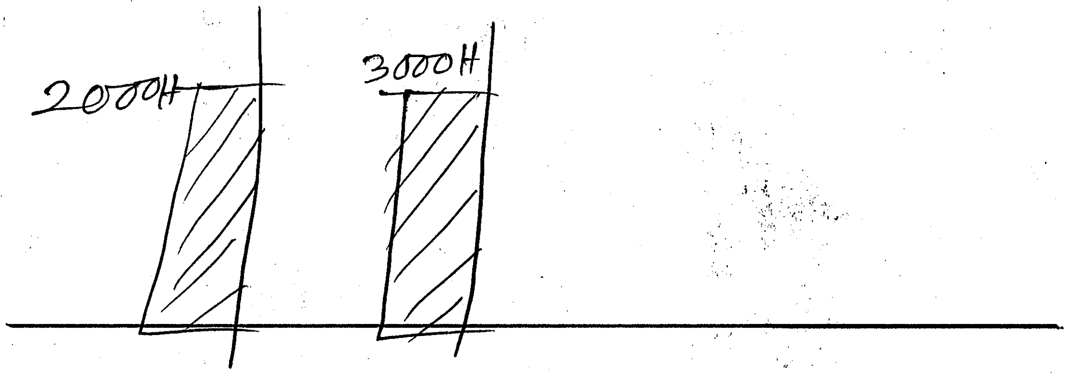


take data from location 2000H and put in 3000H.

Ans:

```
mov CL, [2000H]
mov [3000H], CL
```

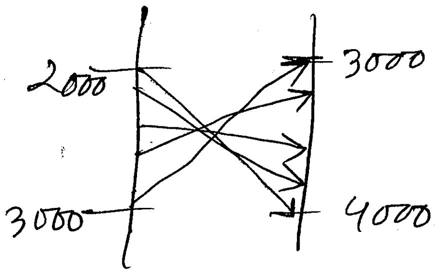
③ block transfer program :



```
mov  SI, 2000H
mov  DI  3000H
 (mov CL, [SI]
  mov [DI], CL   ) loop
  INC  SI
  INC  DI
```

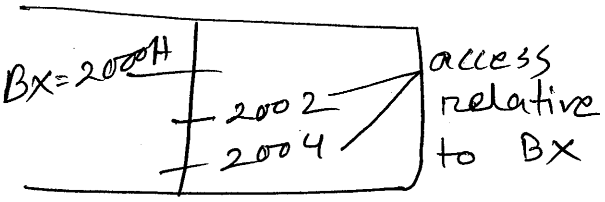
④ block inversion :

```
mov  SI, 2000H
mov  DI, 4000H
 (mov  CL, [SI]
  mov  [DI], CL
  INC  SI
  DEC  DI
```



b) Register Relative (addr ← reg + displacement)

mov CL, [BX + 03H]



CL ← DS[BX + 03H]

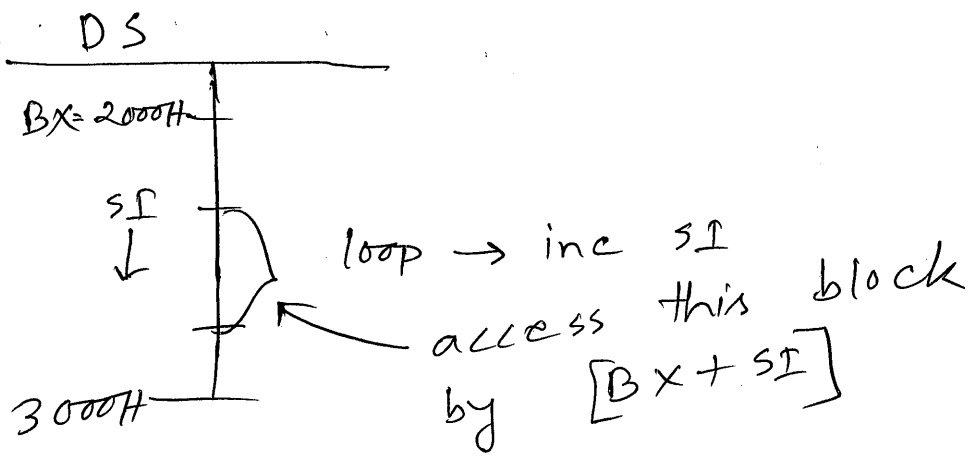
c) Base indexed (addr ← base reg. + index reg)

mov CL, [BX + SI]

CL ← DS:[BX + SI]

mov CL, [BP + SI]

CL ← SS:[BX + SI]



d) <u>Base relative plus indexed</u> :

addr ← base reg + idx reg + displacement

mov CL, [BX + SI + 03H]

CL ← DS:[BX + SI + 03H]

mov CL, [BP + SI + 03H]

CL ← SS:[BX + SI + 03H]

5) Implied $\xrightarrow[\text{is implied}]{\text{~~a~~ operand}}$

we give nothing, some instructions are meant for some operands

ex: STC ; set the carry flag
cf ← 1

CLC ; CF ← 0

~~DAA~~ ;

L-06

8086 Minimum Mode

- MN/$\overline{MX}$ pin

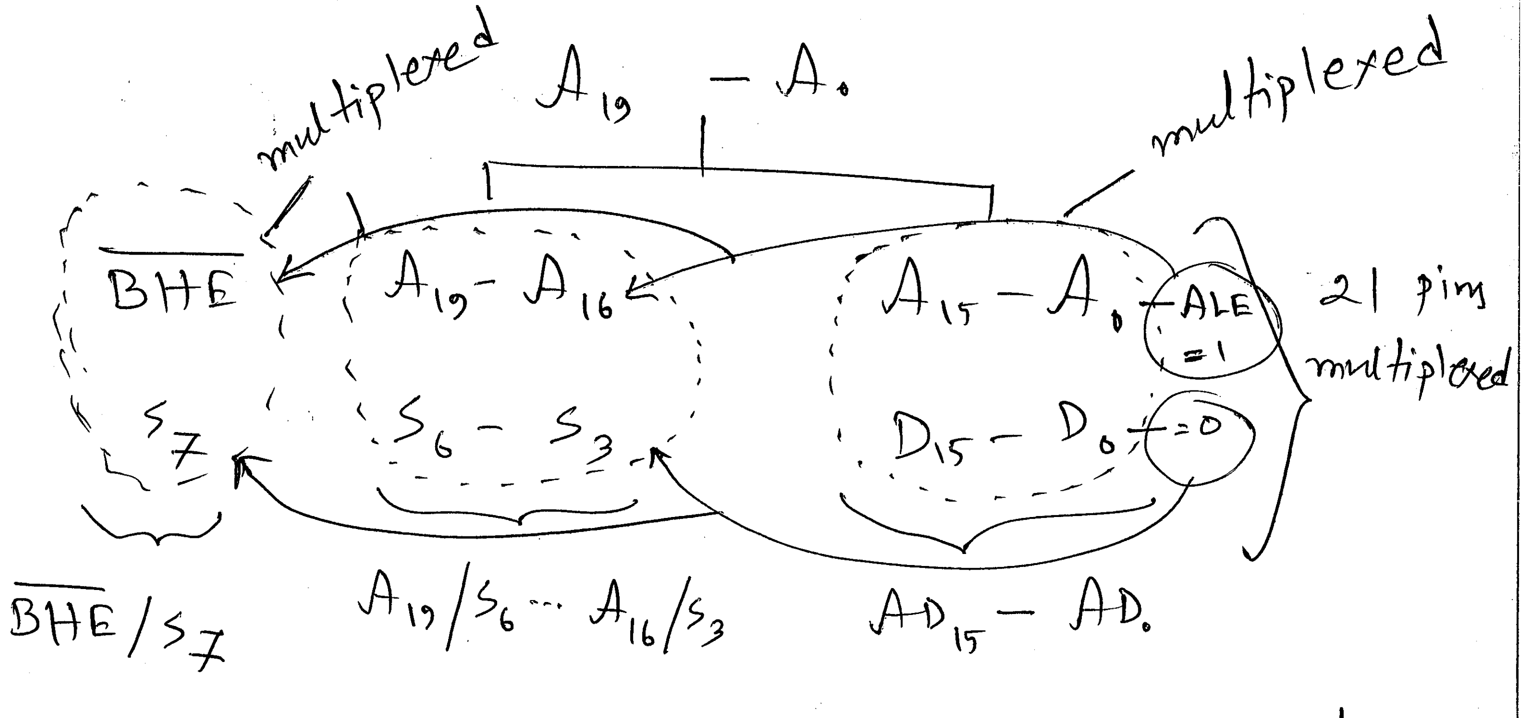
  1 → min mode

  0 → max mode

- min mode — one processor

  max mode — multiple "

- Multiplexing -



if ALE = 0, $AD_{15} - AD_0$ carries data

1, ———————— address

S_4	S_3	
0	0	ES
0	1	SS
1	0	CS
1	1	DS

$S_5 \Rightarrow 0 \Rightarrow IF \Rightarrow 0$

$1 \Rightarrow IF = 1$

$S_6 \Rightarrow 0 \Rightarrow$ 8086 Bus Master

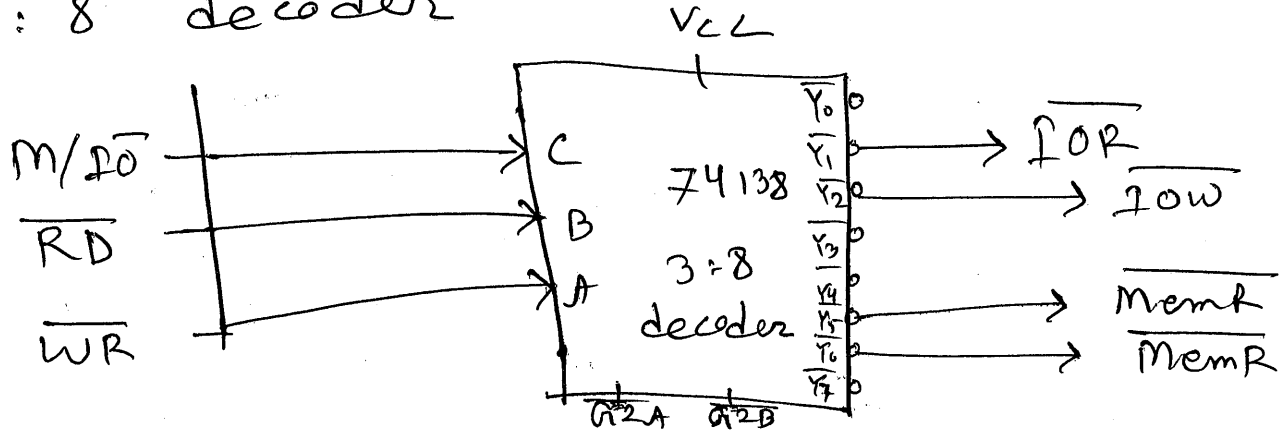
$1 \Rightarrow$ Other BM

So, in min mode, S_6 will be always 0, max mode 1.

(*)

$M/\overline{IO}$	$\overline{RD}$	$\overline{WR}$	OP
0	0	1	$\overline{IO}R$
0	1	0	IOW
1	0	~~0~~ 1	Mem R
1	1	0	Mem W

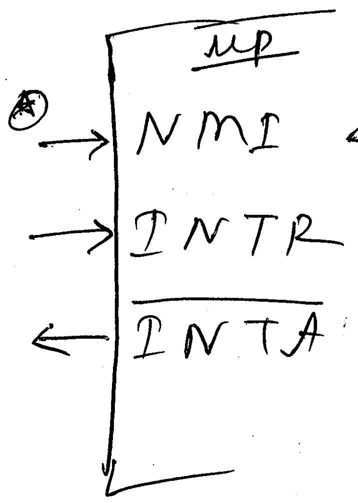
(*) generating these signals using 3:8 decoder



(*)
Clock :

clock is a tick to the up to change its state.

8086 works at 6MHz clk cycle. we've to give 6MHz clock to its clk pin, with 33% duty cycle.

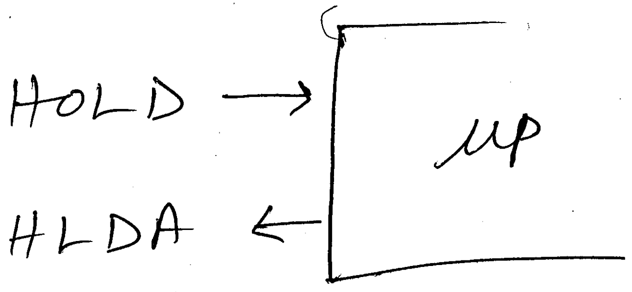


NMI < Non-maskable type 2 intrpt initiated

INTR — maskable, request a h/w intrpt.

INTA — up wants a intrpt vector number by sending intrpt ack.

(*)



by default, up is the BM. DMA Controller sets HOLD = 1 and up sends HLDA to DMAC, it loses control over bus. DMAC becomes new BM and controls the data transfer betn mem. and IO. If DMAC sets HOLD=0 again, μp gets control

Ⓐ DT/$\overline{R}$: 1 → data transmit
0 → data receive

Ⓐ $\overline{DEN}$: enable a transreceiver connected to μp.

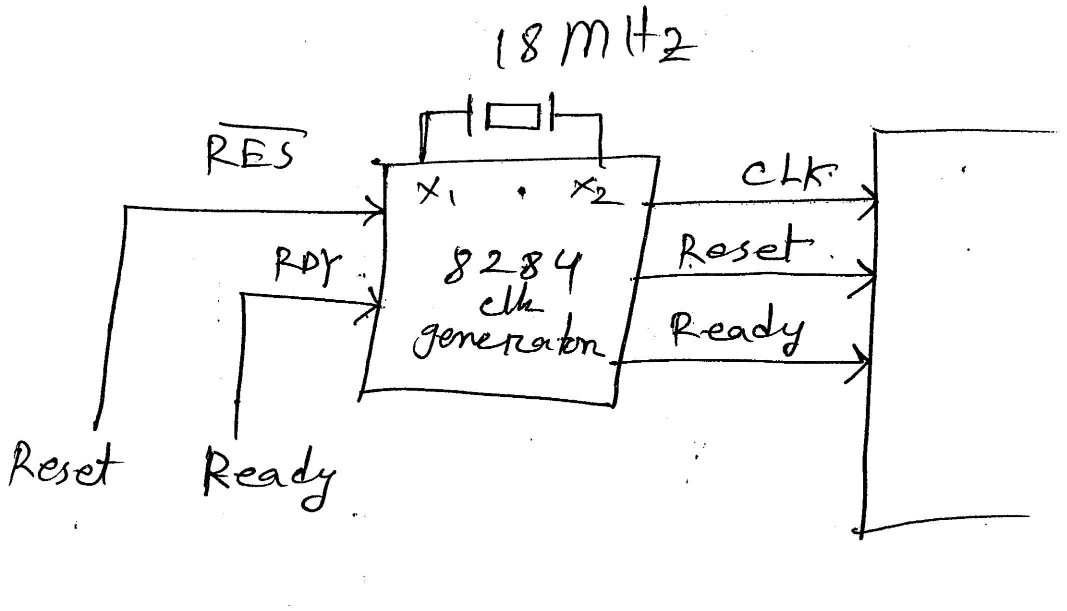
generate clock signal :

purpose is to generate 6 MHz at 33% duty cycle.

↓

transition happens at mid point - 50% duty cycle

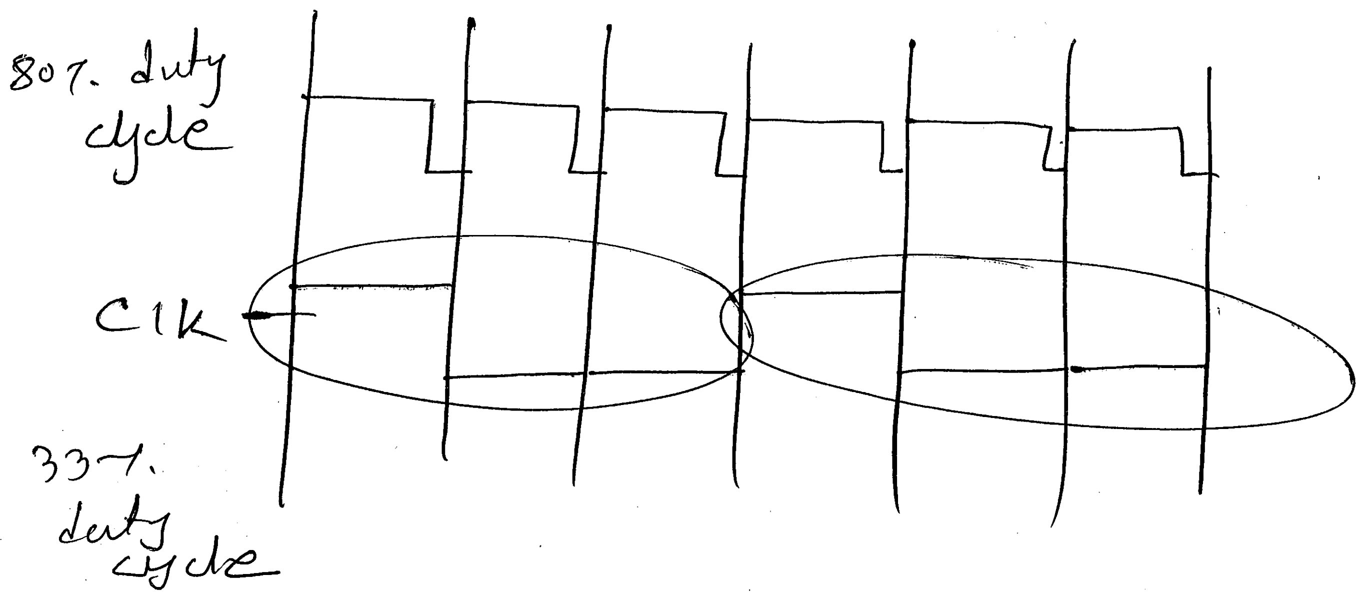
——————— at 1/3rd point - 33% d.c.



⊛ Reset signal provided to 8284, a synch reset signal sent to up, resets the up.

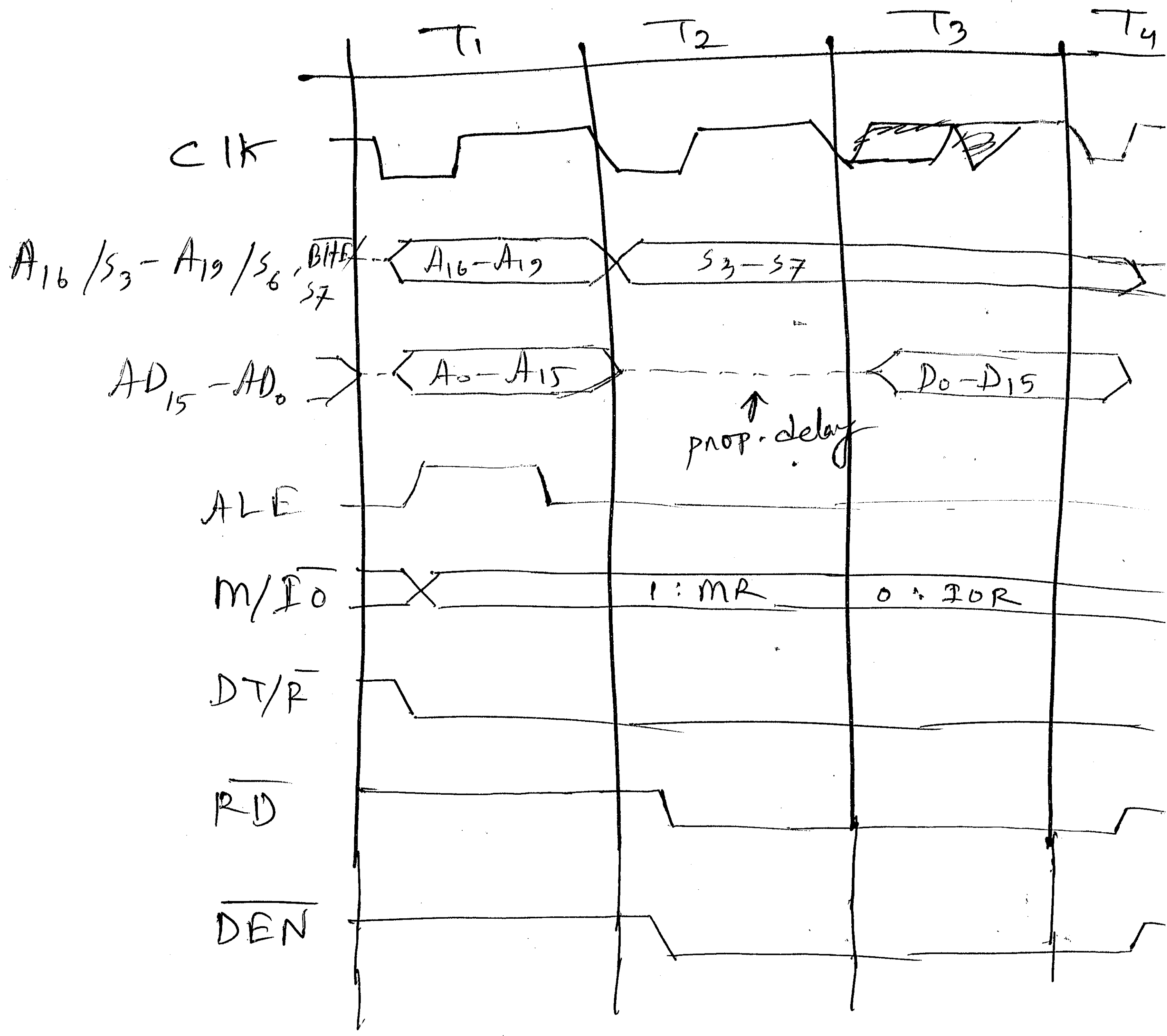
⊛ Ready pin used to synch. the up with slow devices. if Ready = 1, the device is ready, if 0, device not ready, up waits for the device to be ready.
the device gives Ready signal to 8284, that sends a synchronised Ready signal to up.

⊛ 8284, generates a 33% duty cycle from a random duty cycle. doesn't just divide by 3.



we produce ~~3~~ 1 pulse from 3 pulses.

Min mode read cycle

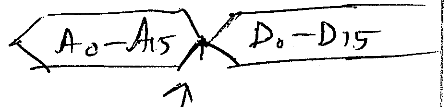


for write timing diagram,

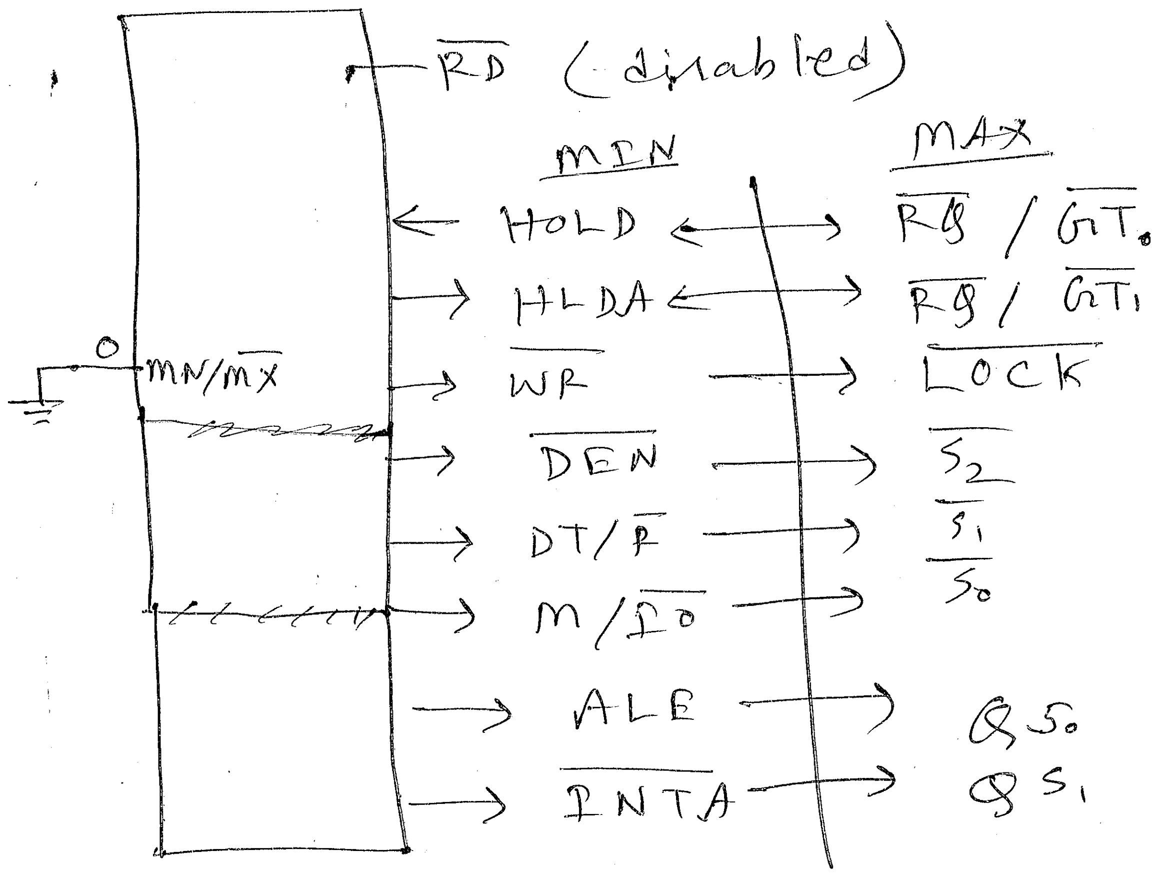
changes → $\overline{RD} \rightarrow \overline{WR}$

$DT/\overline{R} \rightarrow$ high

$AD_{15} - AD_0 \rightarrow$



Max. Mode



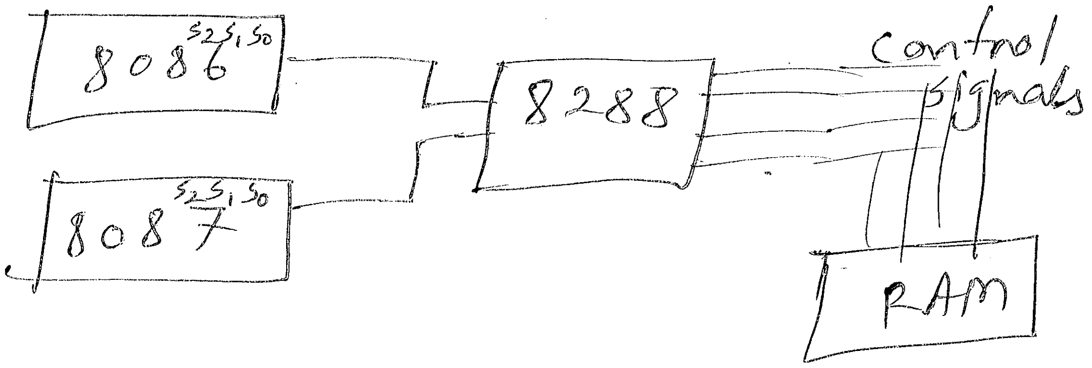
Ⓐ $\overline{RQ}/\overline{GT_0}$: 8087 wants to become BM, sends request by $\overline{RQ}=0$, then 8086 sends grants by setting $\overline{GT_0}=0$.

Ⓐ $\overline{RQ}/\overline{GT_1}$: for another μp.

so, 8086 can have 2 μps with it to transfer bus controls with.

④ $\overline{\text{Lock}}=0$: buses will be locked ~~and~~ until current instruction finished
– No intrpt allowed
– No HOLD granted

⑤ In Max mode, no up can generate control signals cause all ups' control signals will have to connected to RAM and ckt. will be very complicated. to make things simpler, a bus controller 8288 is used.


8086
$\overline{S_2}\,\overline{S_1}\,\overline{S_0}$
8087
$\overline{S_2}\,\overline{S_1}\,\overline{S_0}$
8288
control signals
RAM

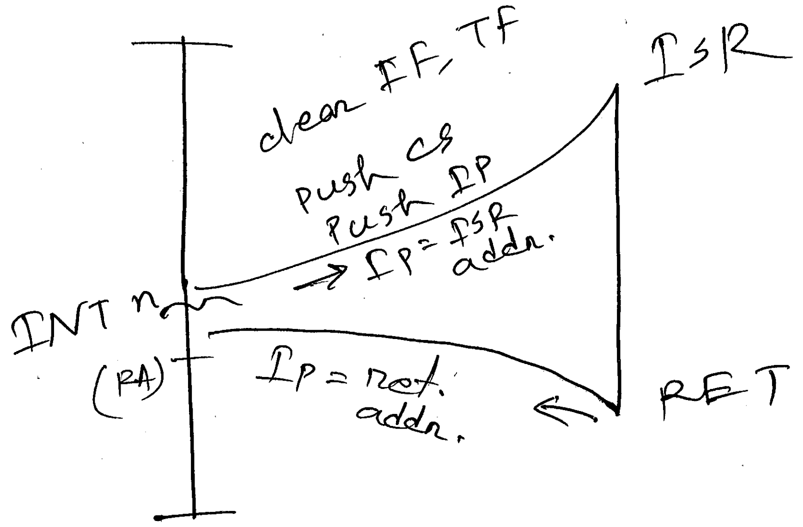
how does 8288 generate control signals?

ans :

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	
0	0	0	INTA
0	0	1	IO Read
0	1	0	IO write
0	1	1	Halt
1	0	0	Instn. fetch
1	0	1	Mem. read
1	1	0	Mem. write
1	1	1	Idle

QS_1	QS_0	prefetch q. status
0	0	No op
0	1	1st byte taken from queue
1	0	queue empty
1	1	subsequent byte taken

Interrupt



$n = 0 \ldots 255$ } 256 interrupts in 8086

when INT n is called, IP gets the value of corresponding ISR addr, then ISR is executed, while returning IP must get