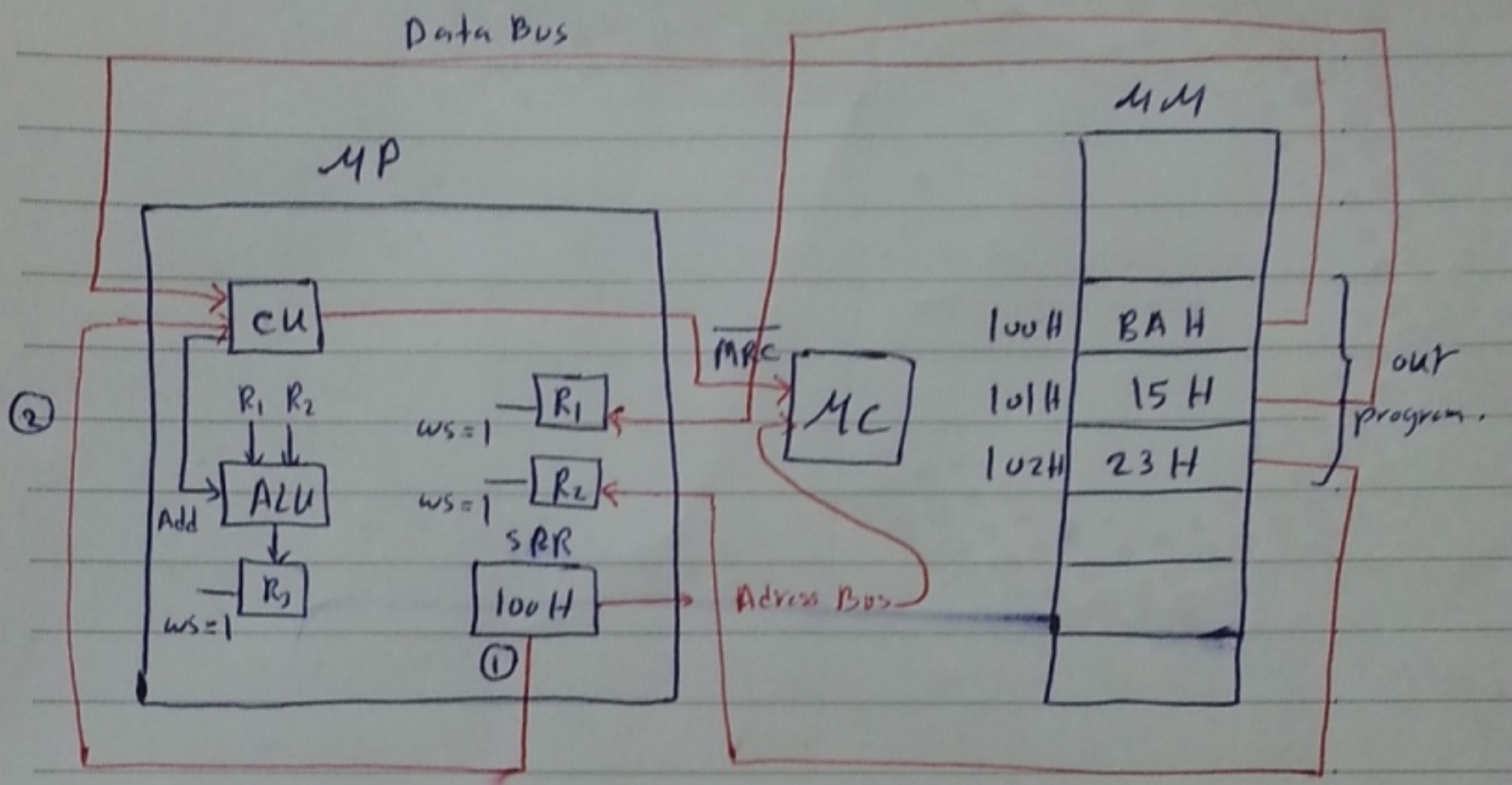
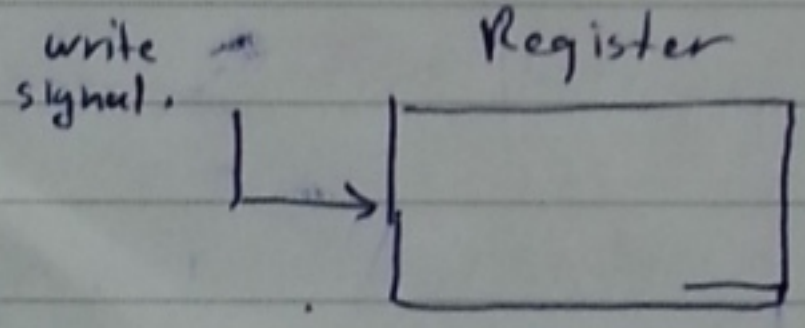
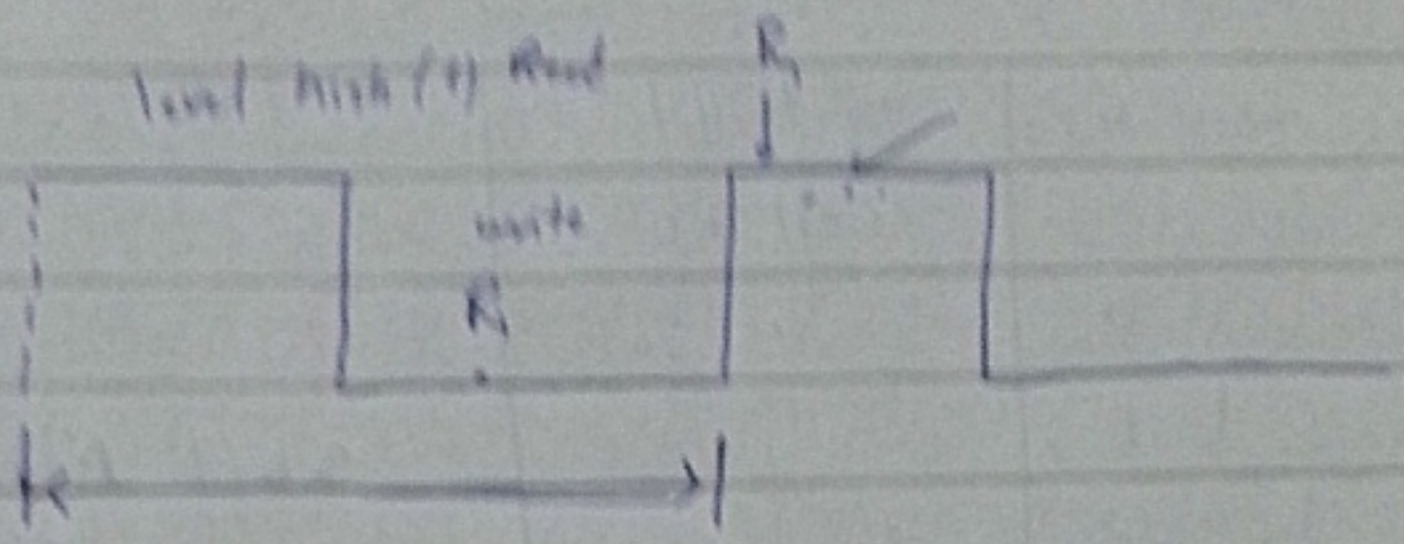
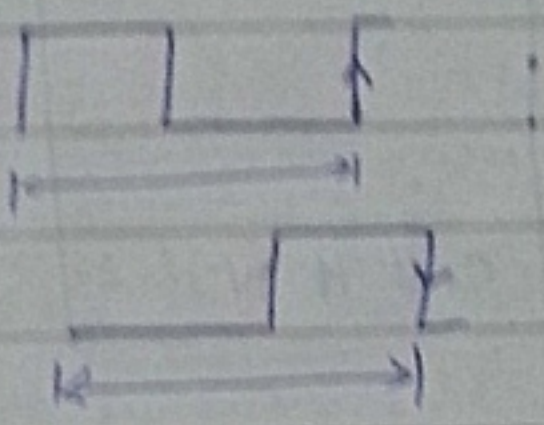
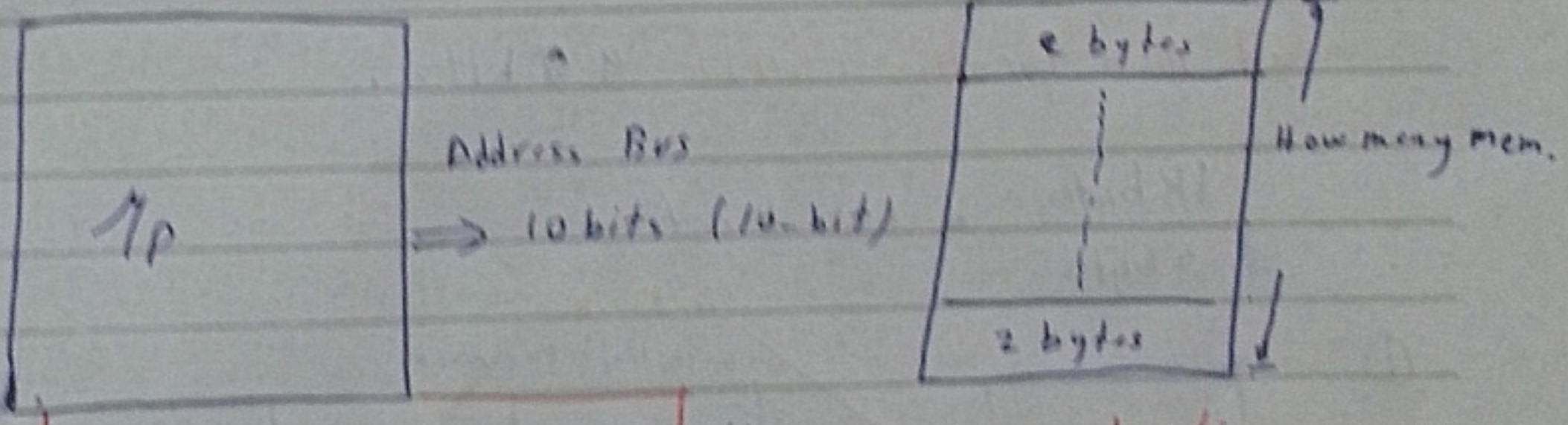

Data Bus
MP
MM
CU
ALU
Add
R1 R2
R3
ws=1
R1
R2
SRR
100 H
①
②
MRC
MC
100H
101H
102H
BA H
15 H
23 H
out
program.
Adress Bus


write
signal.
Register



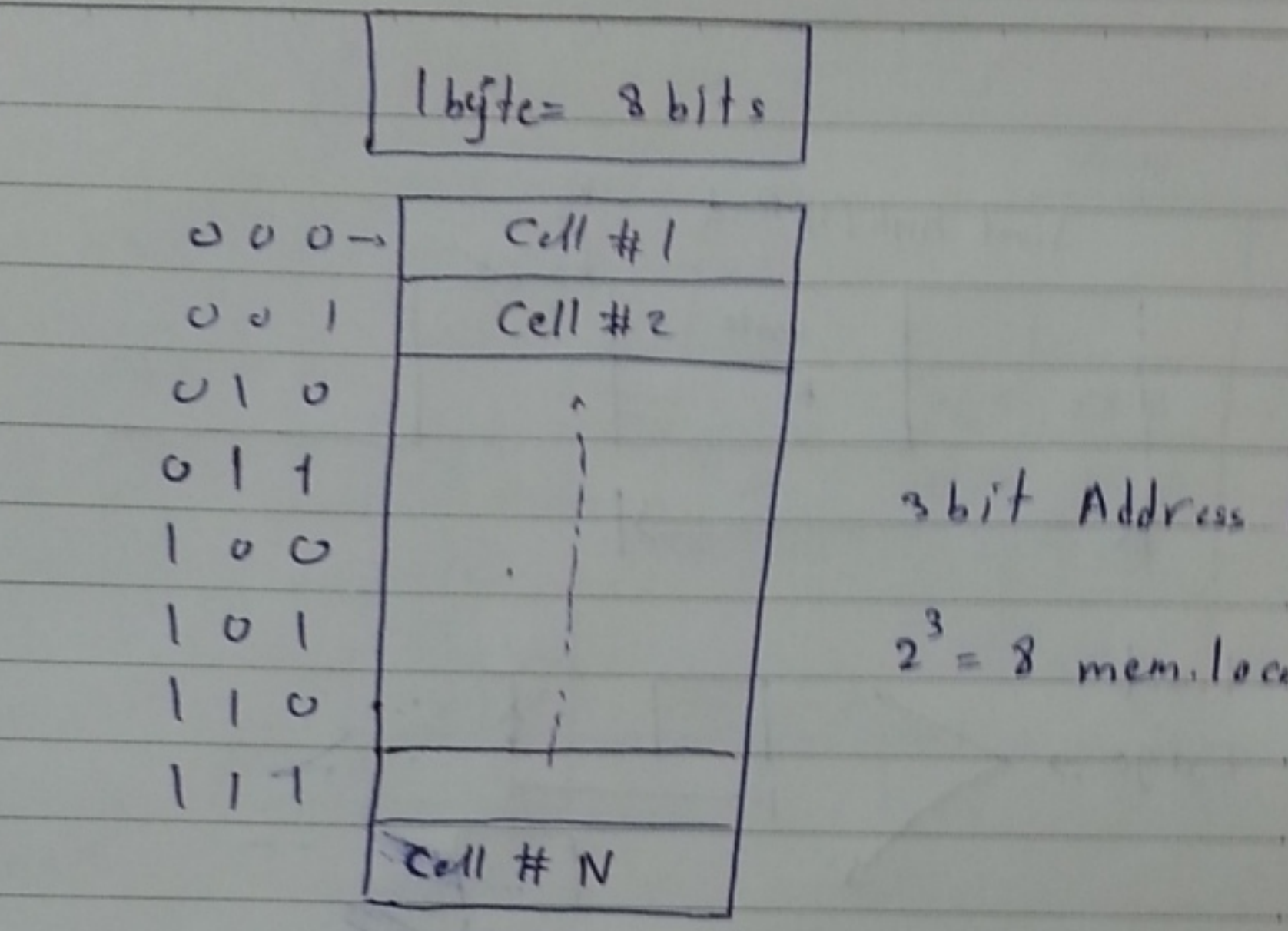
- Level
Edge





$\Rightarrow$ $2^{10} = 1K = 1024$ Mem. location. How many mem. locations are available?

* 32-bit Address Bus
possible number of location? $2^{32} = 2^{10} \cdot 2^{10} \cdot 2^{10} \cdot 2^2$
$= 1K \times 1K \times 1K \times 4$
$= 1M \times 1K \times 4 = 4G$ locations



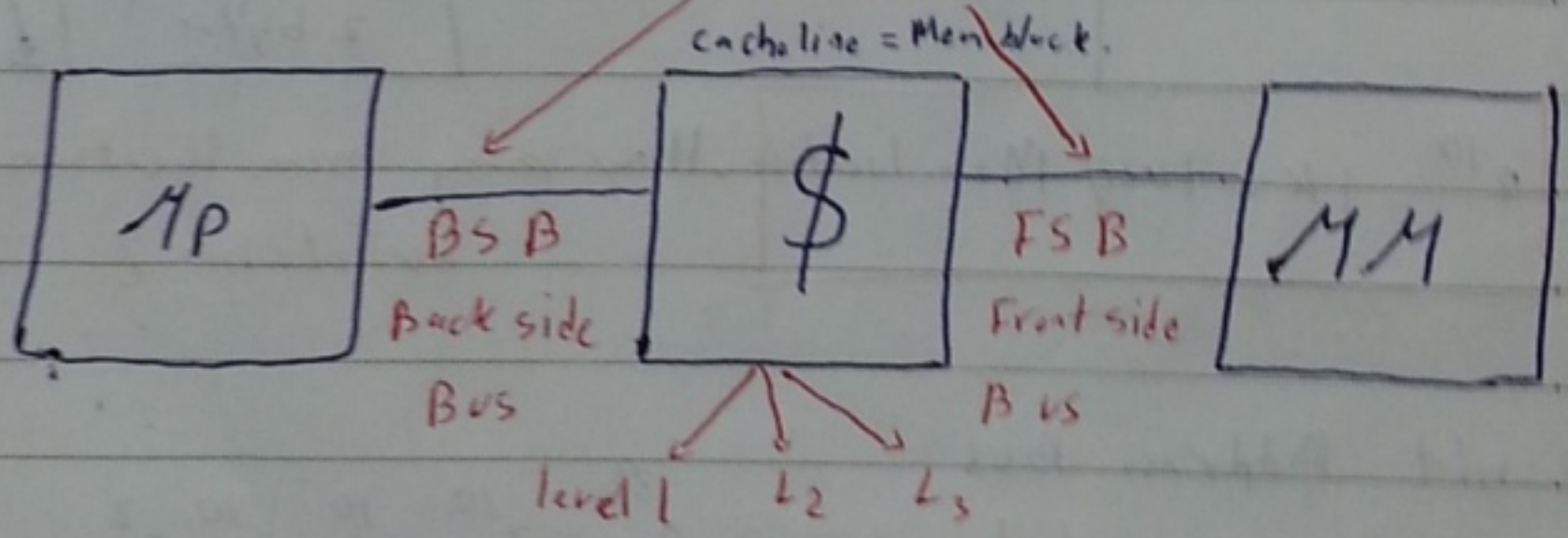
* How many address bits required to access all mem. location

all mem location (512)? 9 bits.

$$\frac{1K\text{ bytes}}{2\text{ bytes}} = 512 \text{ location.}$$

Data Bus

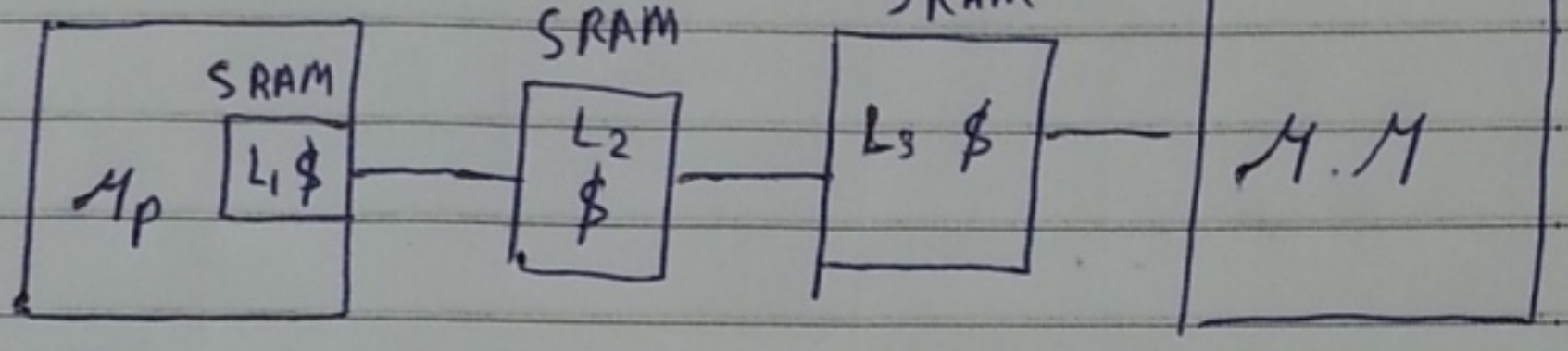
cache line = Mem block.



A
B+
A

(2)

(2)



* ESB $: 32 bits.

Mem. ~~block~~ ? call size = 1 byte.

Mem. block ? $\Rightarrow \frac{32 \text{ bits}}{8 \text{ bits}} = 4$ calls.

~~Block~~. Back side bus size = 1 byte.

① FGL : ooll ooll ooo llll oo.... etc

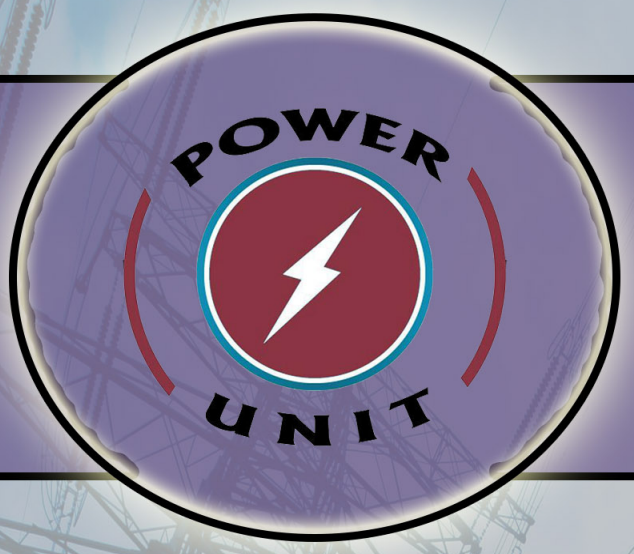
② SGL : ADD $\underline{R_3}$, $\underline{R_{13}}$ $\underline{R_2}$
Dest. Frist. src 1 src 2

} machine Pepundent

③ TGL : K = i + j

④ FGL ; ADD i to j

} machine independant.



Assembly Notes
Dr.khaled darabkeh

By . Mohamad Amera

بأفكارنا_نُبدع

second week

RISC

CISC

ADD R_3 R_2 R_1 ⟵ ADD R_3 R_1 R_2

Dest src src
2 2

load R_2, [100H]

⟵ ADD R_3, R_1, [100H]

ADD R_3, R_2, R_1

src src
1 2

ADD R_3, R_1, R_2 ⟵ ADD [100H], R_1, R_2

store [100H], R_3

(38) X 38 → 46 X

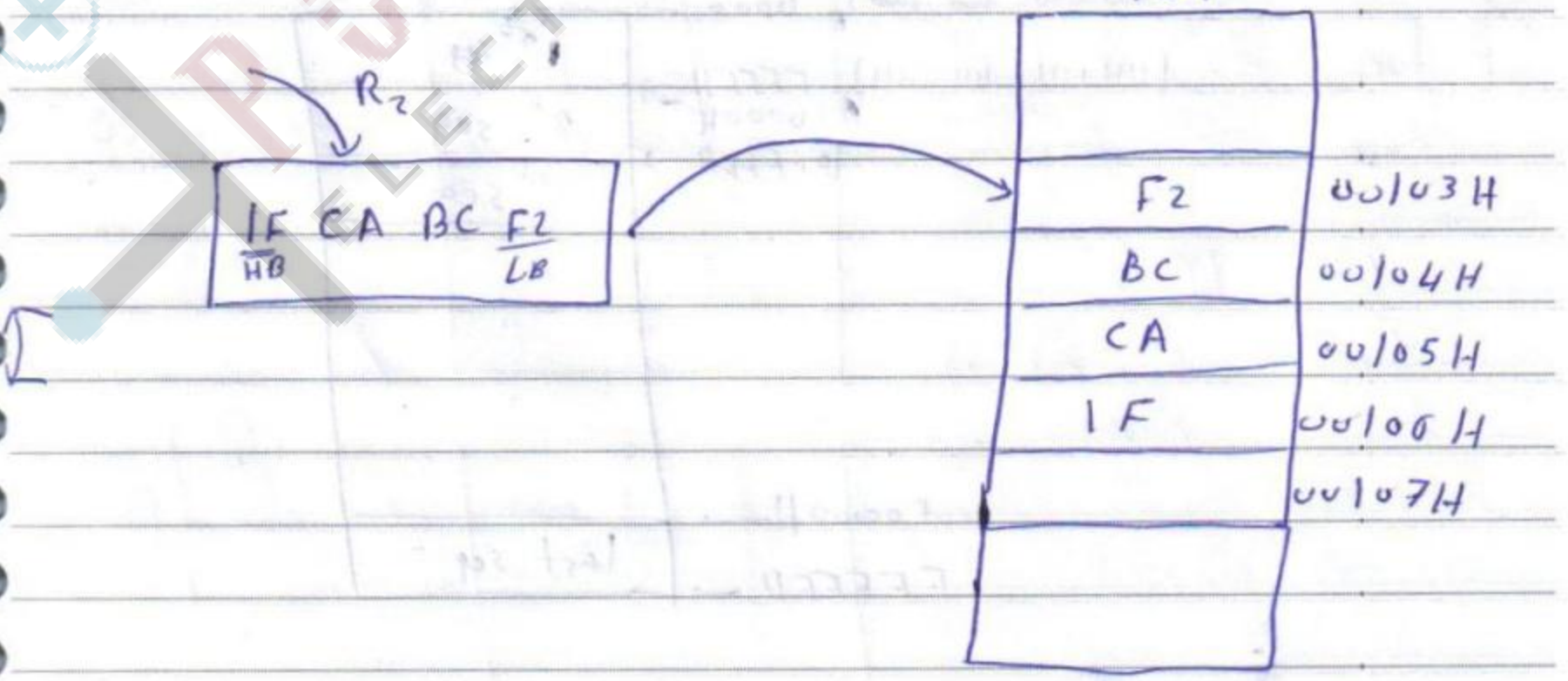
(1) memory - mapped I/O.

(2) separated I/O.

* little Endian.

each location (cell) has a length of 8 bytes.

*



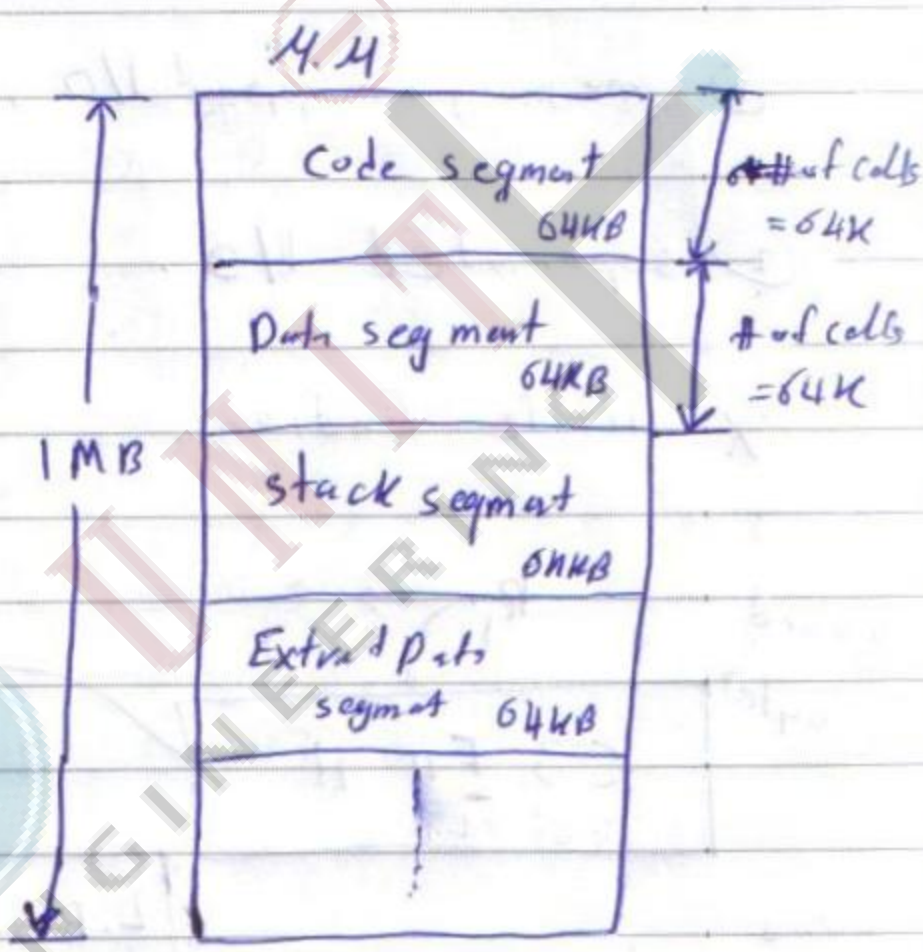
* $\frac{64KB}{1byte} = 64K$ cells.

How many bits required to all of them? 16 bits.

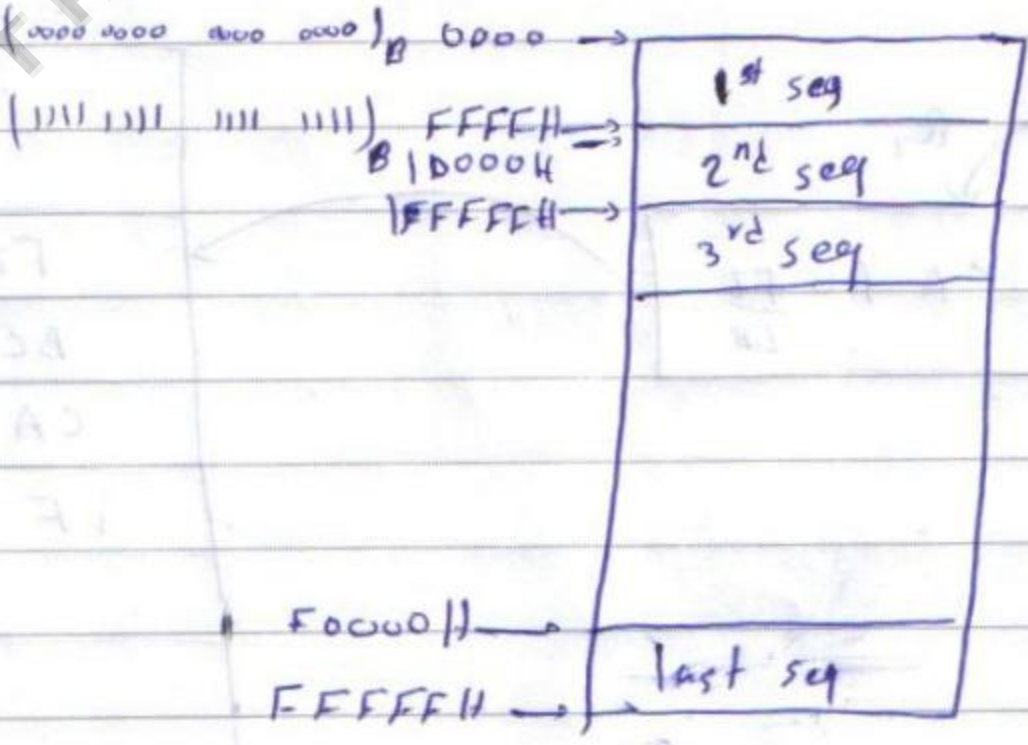
$1MB = 2^{20} B$

$\frac{1MB}{1B} = 1M = 2^{20}$ cells

$64K = 2^6 \, 2^{10} = 2^{16}$



$\Rightarrow$ # of ~~avialb~~ available segs $= 2^{20}_{cell} / 2^{16}_{cell/seg} = 2^4 = 16$ segs



CS : IP

2000 : 3

Frist address of code segment excluding leading zero

offset ( ) address

PA = 2000 * 10 H + 3

= 20003 H

DS ⇒ SI, DI, BX

ES ⇒ DI

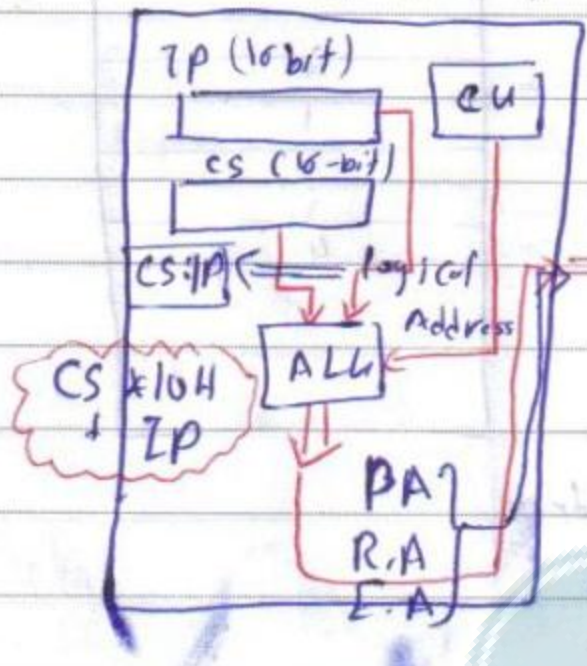
SS ⇒ SP, BP

CS ⇒ IP

مثلاً.

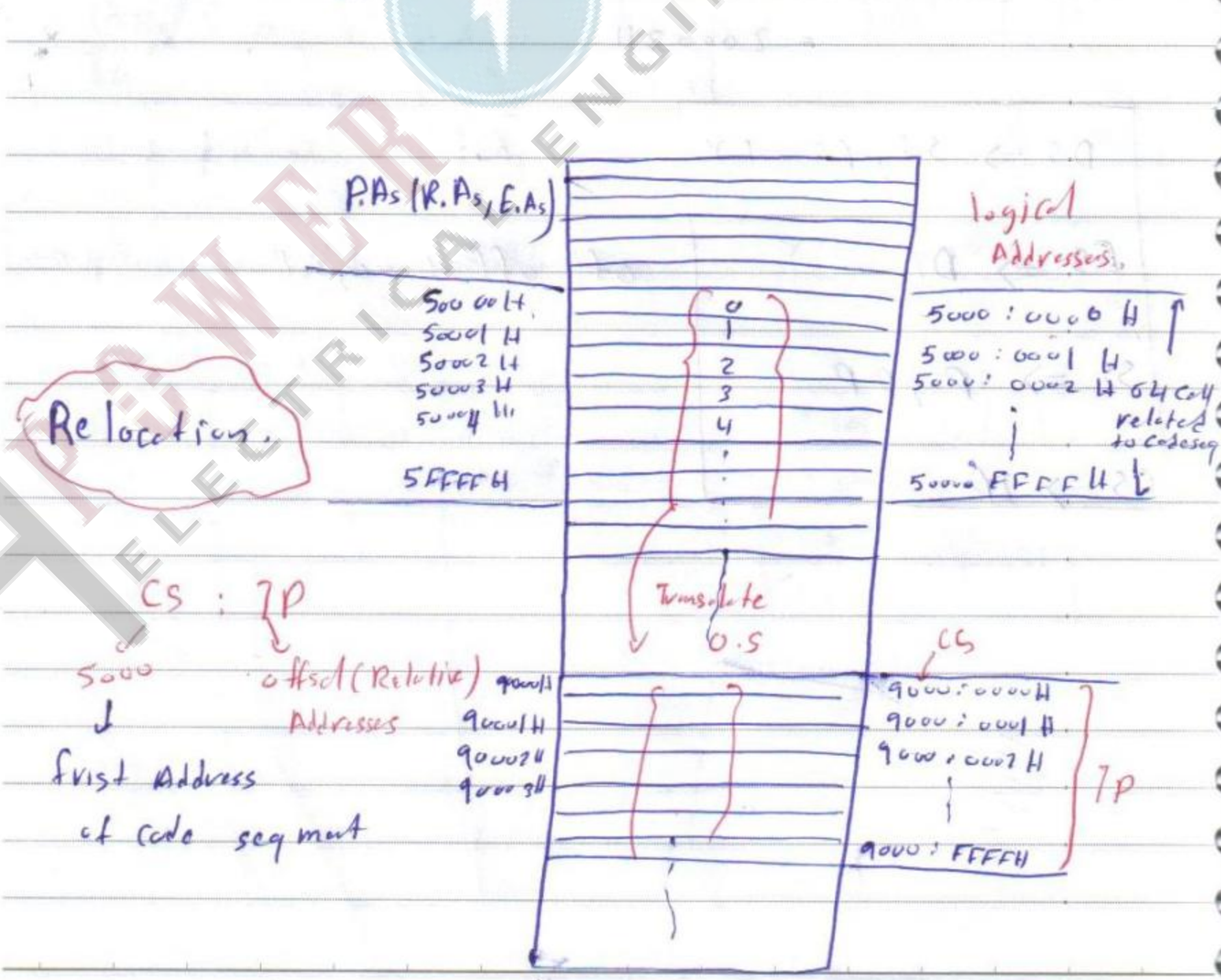
offset register

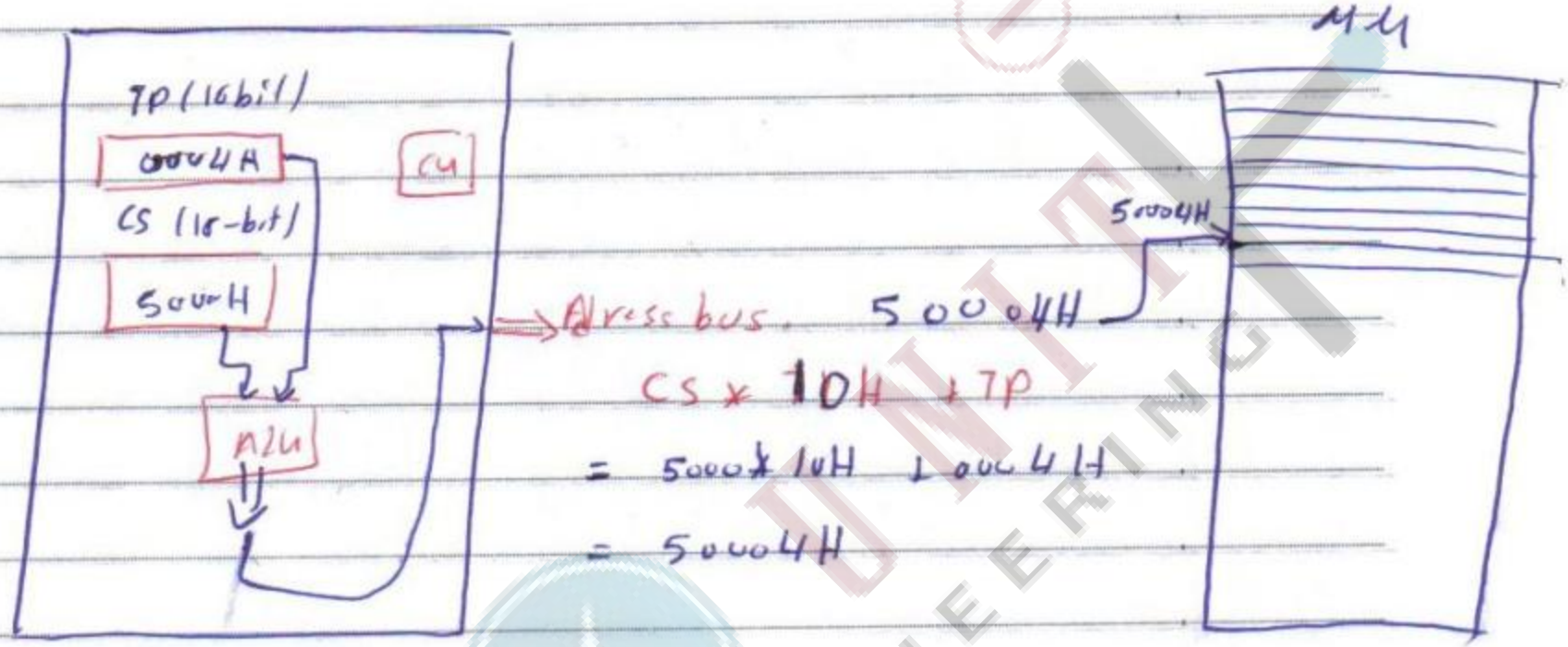
MP (16 bit)

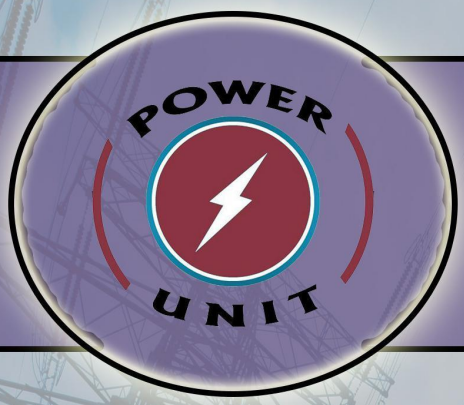


Addres Bus
size = 20 bit

~~all to all~~
to access all
available calls (1 M)




1MB
IP (16bit)
0004H
CS (16-bit)
5000H
CU
ALU
Address bus
50004H
CS * 10H + IP
= 50000 * 10H + 0004H
= 50004H
50004H



Assembly

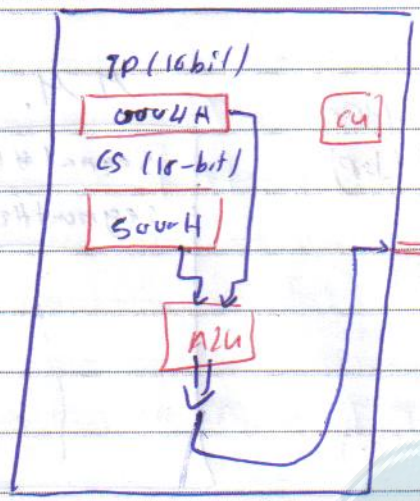
Notes

Dr. Khaled Drabkeh

By: Mohammed 3amera

#بأفكارنا_نُبدع

3rd week



Address bus. 50004H

CS * 10H + IP

= 5000 * 10H + 0004H

= 50004H

SPRs

Level A	Level B	Level C
- IP	- CS	- SP
- FLAGS	Can Read	- DS
Impossible to either read or write. (Don't use Them)	but can't write	- ES
		- SS
		(Read/write) Be careful

* auxiliary Carry

1 → AC

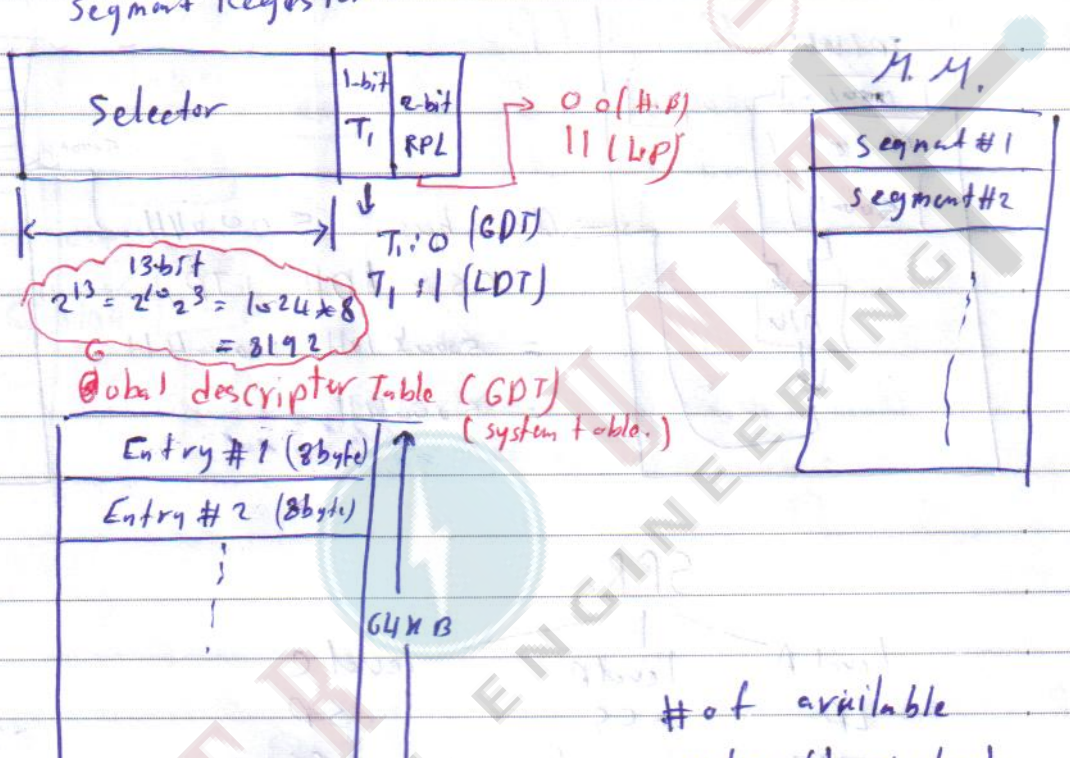
1000 | 001

0110 1100

0101

improve execution time

Segment Register.



106
116
40
156

# of available entry (descriptor)

$= \frac{64KB}{8B} = 8192$

RPL : DPL

00 01 10

10 00

0008 : 0002 H

((logical Address))

(*) Protected mode.

? 100000 Base + offset

0002

100002 H

Real mode.

P.A

0080 + 0002

= 00082 H

① limit : 00FFH

8-bit.

② Base Address : 100000 H

③ Access right byte : 92H

ED = 0 (DS)

1 00 1 00 1 0 → W=1 (Can (write) on it)

P=1 (Pos. is defined)

DPL: 00

(H.P)

E=0

A=0

S=1 (Application Descriptor)

(**) Last Address (Base limit)

100000

00FF

100FFH

(***) segment size = 2^8 = 256

G = 0 limit 0000 H – FFFF H
(max 1M)

G = 1 limit Rang (0 – 1M)

limit is appended with FFF

00000FFFH – FFFFFFFFH

(Max 4G)

Range (4K – 4G)

- Let limit 00FFH ⇒ size segment

(1) segment size = 2^8 = 256

(2) 00 FFF FF ⇒ seg. size = 2^{20} = 1M

(2)

* Consider 80386 & protected Address.

Seg. Address = 000A ; 0003 _ offset Addr

Logical Address

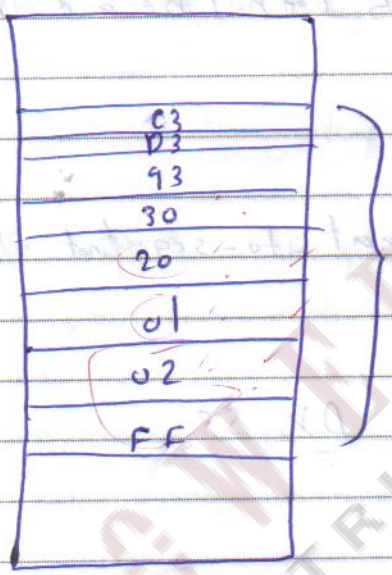
Selector

0000 0000 0000 1010

RPL RPL

T=0 (GDT)

GDT



Entry#1 → Descriptor → limit, Base, Access right

* limit: 302FF

* Base: C3302001

* Access right: 93 H

1001 0011

Descriptor is defined. S=1 (System descriptor) PPL 00 ? 10 Access can't be granted

A=1 Seg. is Accessed

E=0 Data seg

ED=0 Data seg

W=1 can be written

* PA ; Base + offset =

= C3302001 + 0003 =

* last Addr = limit + Base = C3302001 + 302FF

= C3332300

Seg. size =

* Register Addressing Mode :

① Mov AX, CX , ~~write~~ valid.

② Mov CL, BX ; invalid (mixed sizes)
8bit, 16bit

③ Mov CS, DX ; invalid (CS cannot be a destination)

④ Mov CS, DX ; ~~In~~valid

⑤ Mov DS, SS ; invalid (segment_to_segment not allowed)

X Mov DS, SS

Sol.
mov AX, SS
mov DS, AX.

mov DX, SS

CS
↓
SPR (Level B)
↓
(we can read but cannot write)

* Immediate Addressing :

(1) MOV 3AH, CL, invalid (3AH is not a storage element)

(2) MOV CL, AH → Register addressing mode.

MOV CL, 0AH → I. A. M.

(3) MOV CL, F3H, invalid (0 is missing.)

(4) MOV CL, 0F32H ; Valid. (CL = 21H.)

POWER UNIT

Assembly

Notes

Dr. Khaled Drabkeh
By: Moahmad 3amera

#بأفكارنا_نُبدع

4th week

* Immediate Addressing:

① MOV 3AH, CL, invalid (3AH is not a storage element)

② MOV CL, AH → Register addressing mode.

MOV CL, 0AH → I.A.M.

③ MOV CL, F3H, invalid (0 is missing)

④ MOV CL, 0F32H ; Valid. (CL = 21H)

* Given AX = 22F3H,
DS = 1000H, ES = 2000H
SS = F000H, CS = C000H,
offset (Relative) Address

- MOV [1234H], AX

① we want to think about what are inside [ ]
we have a 16-bit number
* we should use DS.

② src. length = 16 bit => 2 byte => mem. 2 calls.

③ PAs: 10000 + 1234H
11234H => direct Addressing
11235H

AX = 22F3H
H.B L.B
HA
L.A
M.M.
Little Endian

11232H	2CH
11233H	F2H
11234H	F3H 23H
11235H	22H 4C
11236H	CA
11237H	AC
11238H	B9
11239H	C4

Direct Data Addressing

Direct Addressing

Two condition should be met

- Using AL, AX, or EAX
- 32 bytes long

(As a machine instruction)

Displacement Addressing

- other registers
- 4 bytes long

x Model small.

x Data

0100 H	number	DB	12 H	0101H → 14H
↑ 0101H	X	DW	3/2F H	32 14H → 0102H → 32 H.
offset Address 0103H	Y	DB	0F3 H	AL = 12H

set by O.S.

Code

startup.

```
Mov AX, Number ; ↔ Mov, AX, [0100H]
Mov BX, 3214 H   x BX = 3214 H
Mov X, BX, ↔ Mov [0101H], BX
Mov CH, Y ↔ Mov CH, [0103H]
Exit
END.
```

x CH = F3H

① MOV AL , [0000H]

② MOV AH , [0001H]

③ MOV [0002H] , AX

④ MOV BX , [0004H]

* Model small

Data

set by O.S

0000 H X DW 0002 H ↔ mov BX, [0002H]

Y DB 1F H

Direct Data Addressing mode (Displacement Addressing)

Given DS = 1000 H

ES = 2000 H

SS = 3000 H

Code

startup

BX = 0002H o/p Mov BX, X ; 1 ↔ Mov BX, [0002H] M.M

CL = 1FH o/p Mov CL, [BX] ; 2 ↔ mov CL, [0002H]

Exit

END.

10000 H	02 H
10001 H	00 H
10003 H	1F H
10004 H	

* Required:

(1) what is the output after ex. this program?

(2) what are the PA(s) generated by instruction 1 & 2.

Instruction 1: P.A(s) ⇒ (10000H, 10001H)

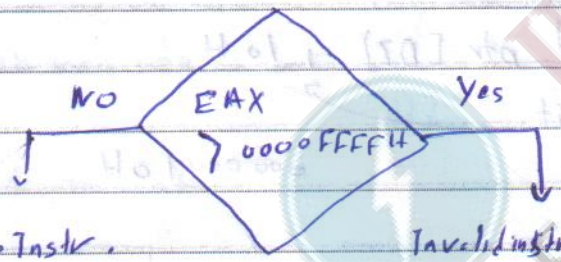
Instruction 2: P.A → 10002 H

* Note 1 :

1. Mov CH , [EAX] ; valid

2. Mov BX , [AX] ; invalid.

More Commone



Keep in mind, we are working on Dos mode (16 bit mode.) (1 mbyte)

mode

offset = 16-bit length.

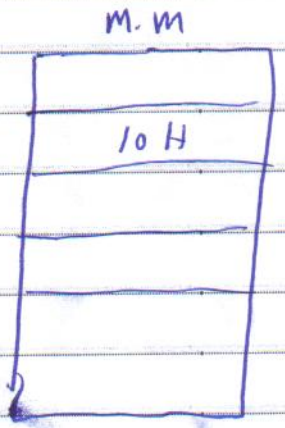
* Note 2 : Leading zeros ignored.

Mov [DI], 10H

Given DS = 1000, DI = 0003H

1000 * 10 + 0003 = 10003H

This is not clear instruction, Hence some direction directions should be added follow



* Direction to be added:

byte ptr, word ptr, dword ptr, Qword ptr.

In other words, this instruction should be written as

Mov dword ptr [DI], 10H

32-bit

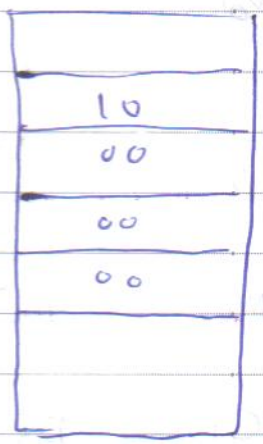
00000010H

Mov AL, [SI+DI]

;invalid

Mov AH, [BX+SI], invalid

Mov AH; [BX] [SI], invalid.



* Base plus Index Addressing.

* Required:

① Base Register → BX, BP

② Index register → SI, DI

Ex: Mov CL, [BX + SI]

or

Mov CL, [BX][SI]

* Note: Mov AX, [BX + BP] ; invalid

Mov AX, [SI + DI] ; invalid.

Move CH, [BP + SI] ; valid.

⇒ which segment will be used?
Data seg. or stack seg?

Answer ⇒ stack segment.

program : model small
Data

0000 x DW 1
Setting ← 0002 y DW 3
0,S 0004 z DB 0F2H

Code
Startup.

MOV SI, X → SI = 1
MOV BX, Y → BX = 3
MOV CH, [SI+BX] → CH = F2H
Exit
END

Ex: MOV CH, [BX+1000H]
Reg. Relative
offset

Ex: Mov CH, [BX + 1000H]
Reg. offset (Relative)

DS = 1000H, ES = 2000H
SS = 3000H, BX = 2

what is the content of CH after ex. This instruction?

11000H	22H
11001H	3FH
11002H	C3H
	2FH
	CBH
	F1H

① P.A = DS x 10 + BX + 1000
= 11002H

CH = C3H

Note:

① Mov BL, [CX] ; invalid.

② Mov CL, [BP] ; valid. (SS)

③ Mov BL, DS:[CX] ; valid.

④ Mov CL, DS[BP] ; valid. (DS)

* Base Relative plus Index Addressing:

① Base Register → BX, BP

② Index Register → SI, DI

③ Relative Date ; → 1000H

Ex: Mov AX, [BX+SI+1000H]

program: model small
Data
X Dw 3
Y Dw 3
Z Dw 0F121H
K Dw 0C121H
code
Startup
Mov BX, X ; BX = 3
Mov SI, Y ; SI = 3
** Mov CX, X[BX+SI]
[1000H], CX = C121H

Two instruction follow displacement addressing

Exit
END.

* Scaled Index Addressing

Required :

① Appear in 32 bit Extension only.

② Index register → ESI, EDI, ECX.

③ Scaling factor → Index register (1, 2, 4, 8)

* Note : (optional)

① Mov CL, [EBX + ECX] ; valid.

② Mov CL, [2*ECX] ; valid.

③ Mov AX, [EBX + 2*[AX]] ; invalid.

④ Mov AX [EBX + 16*ESI] ; invalid.

* stack memory addressing modes :

* two instruction → push
                  → pop.

** SS ⟶ SP

*** push instruction ⟶ Decremant SP
    pop ⟶ Increment SP

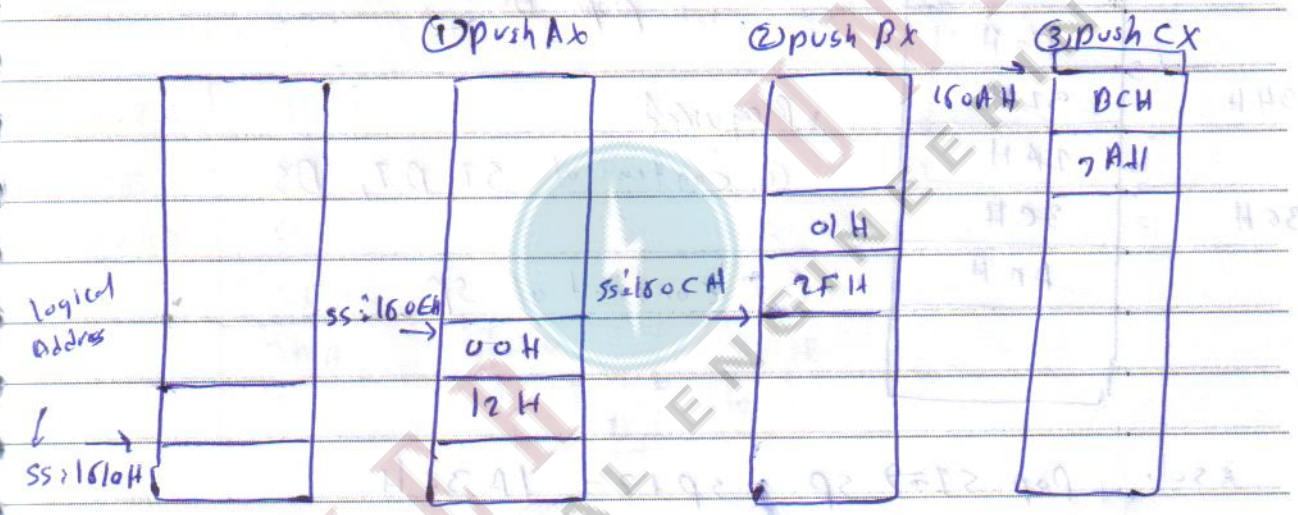
**** storage policy ⟶ LIFO (Lastin - frist out)

Ex: SS = 1000H , SP = 1610H , AX = 12000H
, BX = 2F01H
, CX 2ABC1H

push AX
push BX
push CX

Ex: SS=1000H, SP=1610H, AX=1200H
BX=2F01H
CX=2ABCH

push AX
push BX
push CX



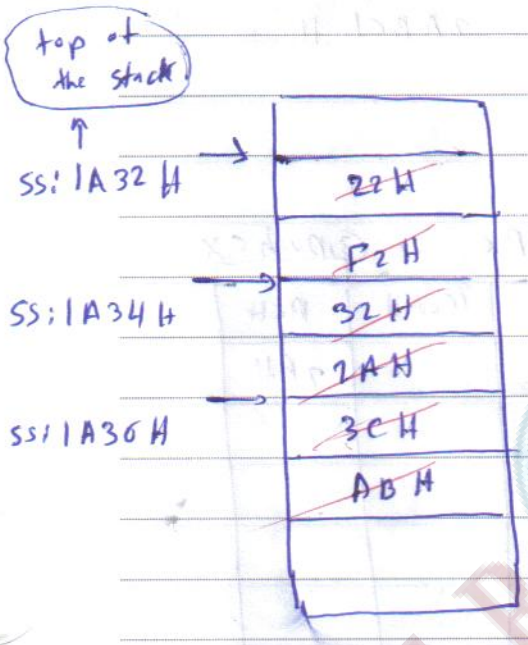
Notes: ① push AL ; invalid (should be atleast 16-bit)

② pop : CS invalid (CS can not be a destination)

③ push [BX] ; invalid.

To make it valid push Byte pointer, word pointer [BX]
dword ptr, & word ptr.

Ex: SS = 1000 H , SP = 1A32 H



2 byte.

pop SI

pop DI

pop DS

+ Required

① content of SI, DI, DS

② content of SP

* sol. pop SI ⇒ sp = sp + 2 = 1A32 H

SI = ~~22~~ F22 H

pop DI → Sp = sp + 2 = 1A36 H

DI = 2A32

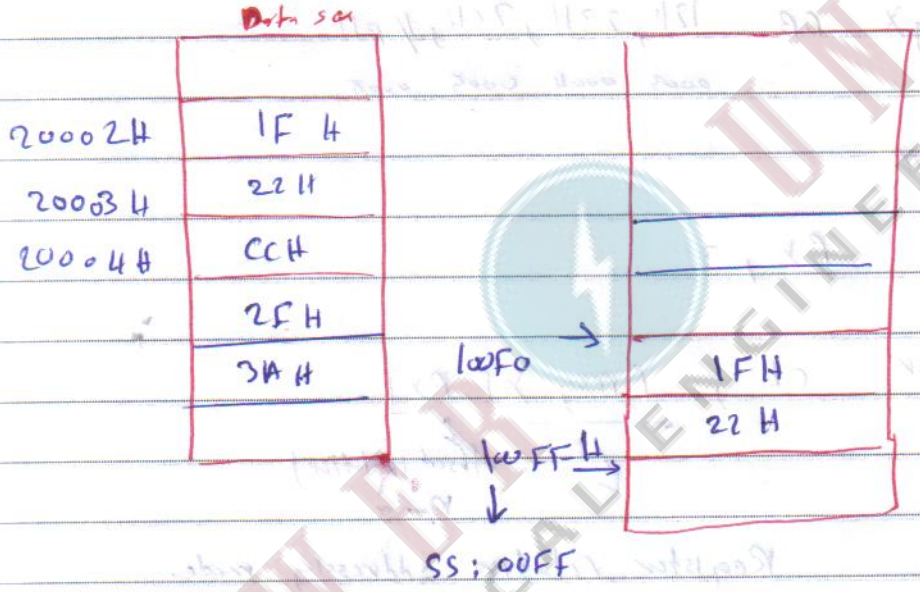
Pop DS → sp = sp + 2 = 1A38 H

DS = AB3CH

Ex: DS = 2000H , Bx = 2

SS = 100H , Sp = 00FFH

push word ptr [Bx]



* How to define arrays:

Array1 DB 12H, 22H, 2CH, 3AH

X DB 33H

Array2 DB 0,0,0,0,0,0,0,0

=

Array3 DB 10 Dup(0)

↓

Directive

Program :

```
model small
Data                0010H   0011H   0012H   0013H
                    ↓       ↓       ↓       ↓
0010H  ArrayX  DB  12H, 23H, 2CH, 4BH
or 0003H            0003    0004    0005    0006
code
Start up

mov  Bx, 3

(*) mov  CL, ArrayX[Bx]
                  ↓      ↓
                  Data   offset (relative)

Exit
ENP
```

Register relative addressing mode.

CL = 4BH

Ch 4: Instruction format:

Mov byte ptr [BX+1000H], 1234H

offset (relative data) (Displacement) Imm. Data.

POWER UNIT

Assembly

Notes

By: Mohammad 3amera
Dr. Khaled Darabkeh

#بأفكارنا_نُبدع

5th Week

ch 4: Instruction format:

Mov byte ptr [BX+1000H] , 1234 H
offset (relative data) (displacement) Imm. Data.

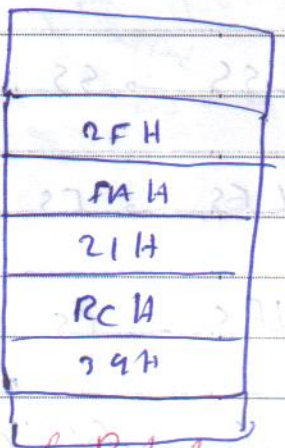
4.3: LEA Instruction:

LEA BX , X (load offset Address)
not real value

Mov BX, offset X
offset Address of X

program: model small
Data

010FH	Data1	DB 2FH	DS: 010FH
	Data2	DW 21FAH	DS: 0110H
	Data3	DW 39BCH	DS: 0112H



Code
status

16-bit
offset Address of Data1
16 bit

Mov SI , offset Data1 ; SI = 010F H
mov AX, [SI+2] , AX = BC21 H

Exit
End.

LEA SI, Data1

Note: Kindy consider the difference between

16-bit DB

mov SI, Data1 ; Invalid (mixed size)

mov SI, offset Data1 ; valid

valid

mov BL, Data1

* load Instruction:

① LDS → DS reg

② LSS → SS reg

③ LES → ES reg

④ LFS → FS reg

⑤ LGS → GS reg.

(load twice)

program model small.

data :

0FA0FH Data1 DB 20H

0F10 Data2 DW 2FABH

DBt. 3 DW 31FCH

code

Start up

LEA BX, Data ; BX = 0F0FH

LDS CX, [BX+1] ; CX = 2FABH

Exit

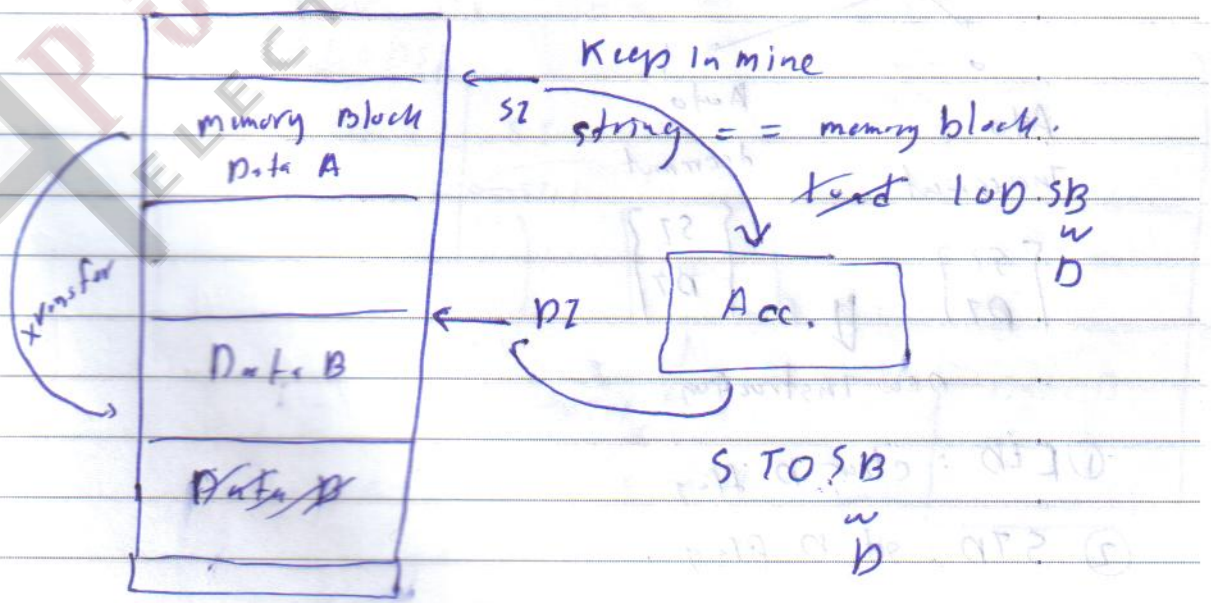
END

LEA BX, Data1 == MOV BX, offset Data1

first load.
second load.

DS = 31FCH

*string Instruction :



String Instructions:

① LODSB ; AL ← DS:[SI]
SI = SI ± 1

② LODSW ; AX ← DS:[SI]
SI = SI ± 2

③ LODSD ; EAX ← DS:[SI]
SI = SI ± 4

④ STOSB ; ES:[DI] ← AL
DI = DI ± 1

⑤ STOSW ; ES:[DI] ← AX
DI = DI ± 4

⑥ STOSD ; ES:[DI] ← EAX
DI = DI ± 4

* Direction flag

0 → Auto Increament → [SI, DI]

1 → Auto decrement → [SI, DI]

↓ new Instruction

① CLD : clear D flag

② STD : set D flag.

* program : model small

Data

Data A DB 12H, 13H, 1AH, 2BH, 3AH,
42H, 26H, 27H, 0AB0H, 10H

Data B DB 10 Dup(??)

Code

Startup

LEA SI, Data A

LEA DI, Data B

CLD ; Auto-increamant.

Mov CX, 10

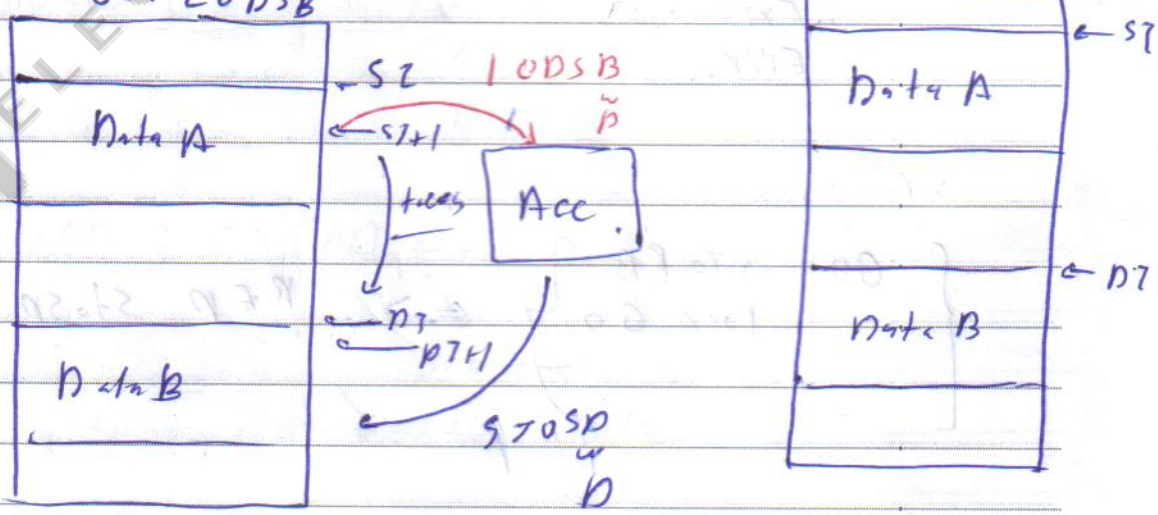
GO: LODSB

STOSB

loop GO

Exit

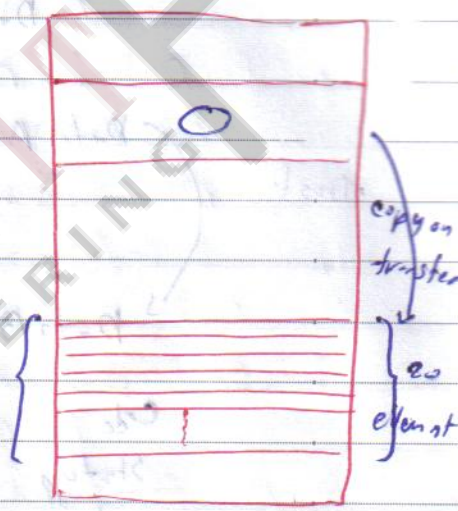
End.



(An Element to block)

Ex:

```
program model small
Data
X DB 3FH
Data DB 20 Dup(4)
Code
startup.
        MOV SI, offset X
        LEA DI, Data
        CLD
        MOV CX, 20
xxx     LODSB      ⟶  AL ⟵ DS:[SI], SI = SI+1
Back:   STOSB      ⟶  ES:[DI] ⟵ AL, DI = DI+1
        loop Back
        Exit
        END.
```



REP STOSB
↓ Some steps done by loop instruction

(*)

* Note:

```
GO: STOSB
    loop GO
```

⟷ REP stosb

* last Lecture (Block-to-Block)

* Difference :

B.M. LODSB ⟹ Block-to-Block
STOSB
loop Back

REP STOSB → An Element to Block.

* New String Instruction :

(1) MOVSB ; ES:[DI] ← DS:[SI]
SI = SI ± 1
DI = DI ± 1

(2) MOVSW ; ES:[DI] ← DS:[SI]
SI = SI ± 2
DI = DI ± 2

(3) MOVSD ; ES:[DI] ← DS:[SI]
SI = SI + 4
DI = DI ± 4

Summary : MOVS = LODS + STOS

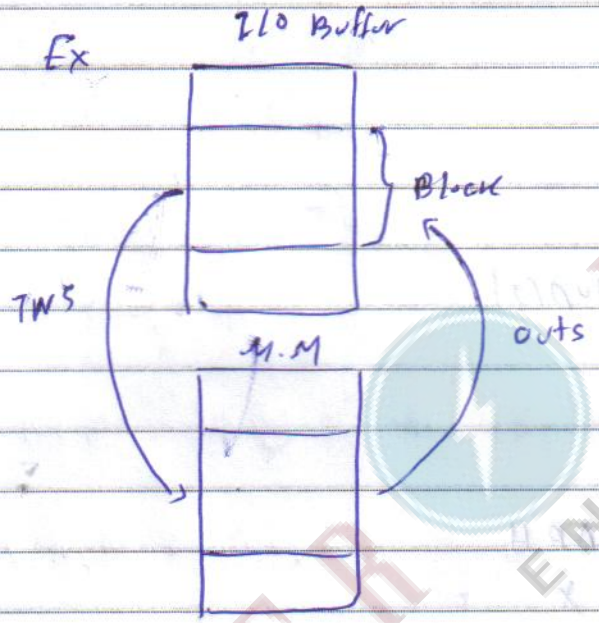
*Note:

Go LODSB
STOSB
loop Go

① ⇒ Go: MOVSB
loop Go

② ⇒ REP MOVSB.

* INS , OUTS Instructions:



* program:

① INSB ; ES:[DI] ← [DX] , DI = DI ± 1
② INSW ; — ; DI = DI ± 2
③ INSD — DI = DI ± 4

④ OutsB ; [DX] ← DS:[SI] , SI = SI ± 1
⑤ OutsW ; = = , SI = SI ± 2
⑥ OutsD ; = = SI = SI ± 4

Note: string Instruction

src → DS:SI

Dest → ES:DI

```
model small
Data
Data X DB 50 DUP(?)

Code
startup
Mov DX , 0F1ABH
LED mov DI , Data X
CLD ; Auto- increment.

Back: INSB
ADD DX ,1
loop Back
Exit
END
```

* XLAT Instruction :

- Bx
- AL
- No Operands.

Result / AL

make small program, Data

Index	SQR_Table
0	0
1	1
2	4
3	9
4	16
5	~~25~~ 2
6	36

```
SQR_Table DB 0,1,4,9,16,25,36
code
startup
mov Bx, offset SQR_table
mov AL, 6
XLAT
Exit
END
```

(1) mov AH, 0
(2) mov DI, AX
(3) mov AL, [Bx+DI]

* IN and Out Instruction :

Notes : ① use AL, AX, or EAX

② port number → 8-bit (constant)
→ 16-bit (DX)

Ex IN, AL, 39H
IN AX, 39H
IN, EAX, 39H

IN AX, DX

out 2FH, AL
out 2FH, AX
out 2FH, EAX
out DX, EAX,

POWER UNIT

Assembly

Notes

Dr. Khaled Darabkeh
By: Rania Daraghmeh

بأفكارنا_نُبدع

6th week

Assembly (continued)

⇒ Scaled Index Addressing :-

Required :

1) Appears in 32-bit extension only.
2) Index register → ESI
   ↘ EDI
   ↘ ECX
3) Scaling factor → Index register (1, 2, 4, 8).

Notes: ① MOV CL, [EBX + ECX] ; valid (optional)
② MOV CL, [ECX] ; valid
③ MOV AX, [EBX + 2*EAX] ; invalid (No index Register).
④ MOV AX, [EBX + 16 * ESI] ; invalid
= Scaling factor not in range.

→ Stack memory addressing modes :-

* Two instructions
  ↙ Push ↘ POP

* Segment Register → Stack Segment (SS) works with Stack pointer (SP).

** Push instruction → Decrement SP.
   POP " → Increment SP.
   Storage policy → LIFO (last in first out)

→ Example:

SS = 1000H, SP = 1610H, AX = 1200H

BX = 2F01H

CX = 2ABCH

PUSH AX

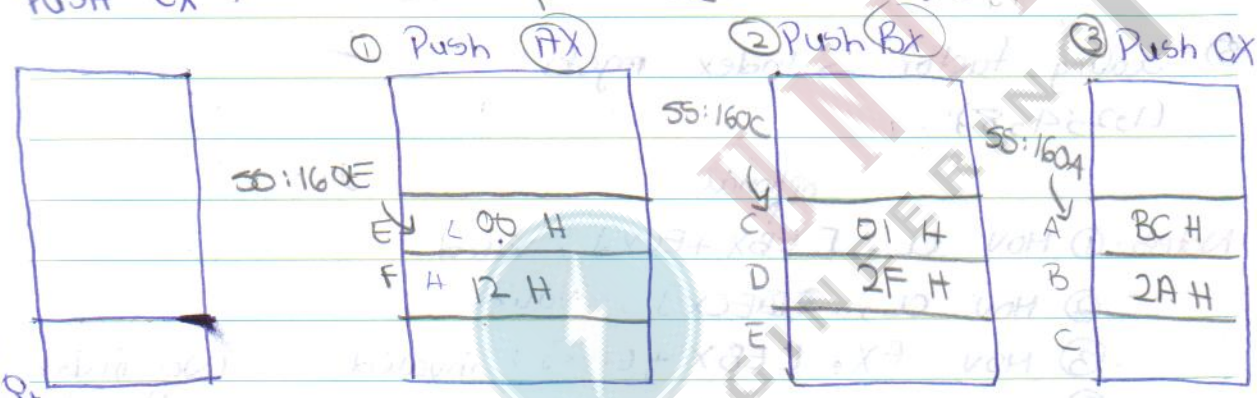
PUSH BX

PUSH CX

2 bytes

decrement by 2

2 bytes

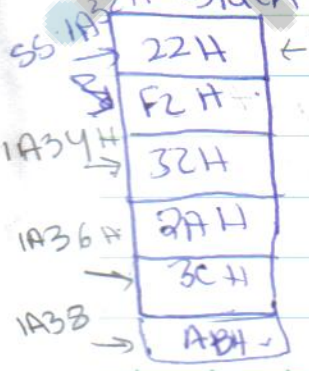


* Notes: push AL ; invalid (should be at least 16-bits)

Pop CS; invalid (CS can't be a destination)

⇒ Example: SS = 1000H, SP = 1A32H

1A32H stack memory.



← Top of stack ***

POP SI.

POP DI

POP DS.

Required : ① Contents of SI, DI, & DS → 2 bytes all of them.
② Content of SP

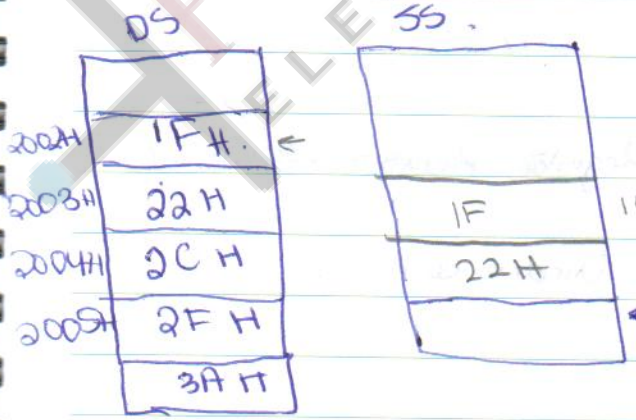
Solution: POP SI → SP = SP + 2 = 1A34 H.
SI → F222 H.
POP DI → SP = SP + 2 = 1A36 H.
DI = 2AB2 H
POP DS → SP = SP + 2 = 1A38 H
DS = AB3C H

→ Note 3. push [BX] ; invalid
(because it's pushing from a Data Segment, DS).

To make it valid : Push ~~ptr~~ byte Ptr [BX].

Push word ptr [BX]
dword ptr [BX]
qword ptr [BX].

Example: DS = 2000 H , BX = 2.
SS = 1000 H , SP = 00FF H. Push word ptr [BX]



PA = 2000 * 10 + 2
= 2002.

Example 3-15 → Check!

Last in first out

PUSH CX

POP AX

AX = CX

after MOV CX, 3C

MOV SP, 10FA H

Push و Pop بيلغوا بعض

cancel each other if & only if they have the same size.

How to define arrays:

Array1 DB 12H, 22H, 2CH, 3AH Memory Block 4 elements

X DB 33H (1 element)

Array2 DB 0000000000

Array3 DB 10 DUP(0) ← instead

Program:

```
.Model small
.Data
Array X DB 12H, 23H, 2CH, 4BH.
.Code
.Startup
MOV BX, 3          → Register Relative add. mode
MOV CL, Array X[BX]
.EXIT
.End.
```

offset بيرجع (Directive data)

CL = 4BH

LEA instruction :-

offset add of X

LEA BX, X ⇒ Same as ⇒ MOV BX, offset X

⇒ Example : ? load offset add. not real value

- Model small
- Data .

010FH Data1 DB 2FH
Data2 DW 21FAH
Data3 DW 39BCH

- Code
- Startup .

MOV SI, offset Data1 ; SI = 010FH
MOV AX, [SI+2] ; AX = BC21H

2 bytes (high-high low-low)

- EXIT
- END .

DS: 010FH	2F
DS: 0110	FA
DS: 0111	21
DS: 0112	BC

*Note : Kindly consider the difference between:

MOV SI, Data1 ⇒ Invalid (mixed sizes)

& MOV SI, offset Data1

MOV BL, Data1 ⇒ valid

MOV SI, Data1 ⇒ Invalid (Mixed sizes) but when I say
16-bit DB

MOV SI, offset Data1 ⇒ valid.
16-bit 16-bit

the same :

LEA SI, Data1

Load instructions:-

1. LDS → DS reg.
2. LSS → SS reg.
3. LES → ES reg. (Load twice).
4. LFS → FS reg.
5. LGS → GS reg.

Example:

```
OFOFH  Data1   DB    20 H   ;
       Data2   DW    2FABH
       Data3   DW    31FCH
       Code
       Startup
       LEA   BX, Data1  ; BX = OFOF H
       LDS   CX, [BX+1] ; CX = 2FAB H
       EXIT                  1st load → DS = 31FCH
       END                   2nd load
```

→ String Instruction :-

* Keep in mind String == memory block.

→ String Instructions:

① LODSB ; AL ← DS:[SI]
SI = SI + 1 .

② LODSW ; AX ← DS:[SI]
SI = SI ± 2

③ LODSD ; EAX ← DS:[SI]
SI = SI ± 4 .

---

④ STOSB ; ES:[DI] ← AL
DI = DI ± 1

⑤ STOSW ; ES:[DI] ← AX DI = DI ± 2.

⑥ STOSD ; ES:[DI] ← EAX DI = DI ± 4 .

Direction Flag → 0 → auto increment (SI, DI) .
→ 1 → auto decrement (SI, DI) .

⇓ New instructi

① CLD : clear Dflag ⇒ 0 ⇒ auto increment

② STD : set Dflag ⇒ 1 ⇒ auto dec .

→ Example: (Block - to - Block)

- Model Smaee
- Data

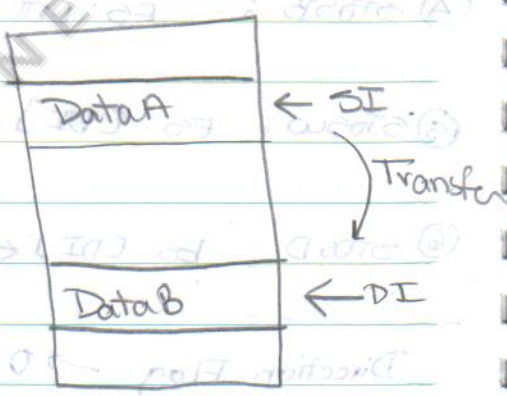
DataA DB 12H, 13H, 1AH, 2BH, 3AH, 42H, 26H,
27H, 0ABH, 10H

DataB DB 10 DUP(?)

→ Can
Not - initialized

- Code
- Startup

LEA SI, DataA
LEA DI, DataB
CLD ; auto - increment
MOV CX, 10
LODSB
STOSB
Loop G0
EXIT
END



Loop instruction:-

Syntax:

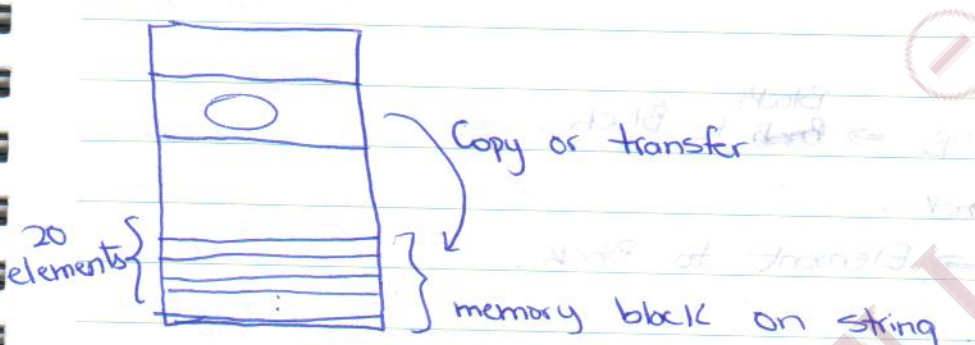
Loop G0
CX

① Decrement CX ② CX != 0

if CX == 0 ⇒ Exit

if CX != 0 ⇒ Loop back.

⇒ Example: (~~One~~ An element to block) :



Program:

```
.Model small .
. Data
X DB 3FH .
DataB DB 20 DUP(?)
.Code
.Startup
MOV SI , offset X .
LEA DI , DataB .
CLD
MOV CX, 20 .
Back: LODSB → AL  DS:[SI] , SI = SI + 1 .  (*)
      STOSB → ES:[DI] ← AL , DI = DI + 1 .
      loop Back .
EXIT
End .
```

REP STOSB
⇓ same task done by loop instruction

*Note: Go: STOSB
Loop Go ⇔ REP STOSB .

Difference between element to block & block-to-block:-

Back LODSB
STOSB. ⇒ Block to Block.
loop Back.

REP STOSB ⇒ Element to Block.

⇒ New string instruction:-

① MOVSB; ES:[DI] ← DS:[SI]
SI = SI ± 1
DI = DI ± 1.

② MOVSW; ES:[DI] ← DS:[SI]
SI = SI ± 2
DI = DI ± 2.

③ MOVSD; ES:[DI] ← DS:[SI].
SI = SI ± 4.
DI = DI ± 4.

Summary:- MOVS = LODS + STOS

*Note: GO: LODSB, STOSB
loop GO
OR
② REP MOVSB.

pw ① GO: MOVSB
loop GO.

⇒ 1st exam till here.

Program

```
.Model Small .
.Data
DataA   DB      'A B C DEFG'
DataB   DB      7 DUP(?)
.Code
.Startup
   MOV  CX, 7
MOV SI, offset DataA
   ADD  SI, 6
LEA   DI, DataB
ADD   DI, 6
   STD
REP  MOVSB
.EXIT
.END
```

INS & OUTS instruction:-

INSB ; ES:[DI] ← [DX], DI = DI ± 1

INSW ; = , DI = DI ± 2

INSD ; = , DI = DI ± 4

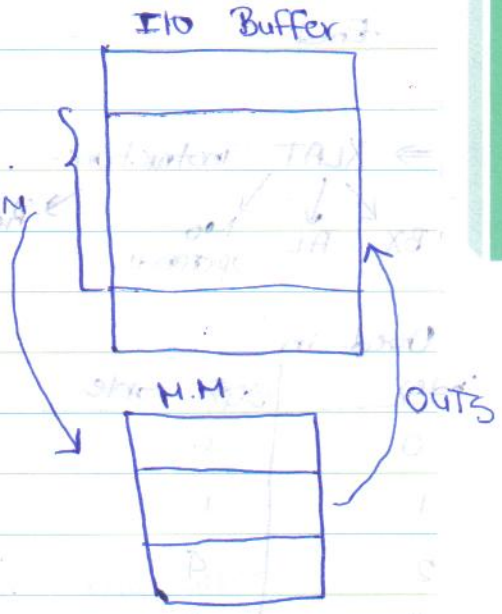
Block Copy block → M.M

INS

OUTSB : [DX] ← DS[SI]

OUTSW = ← SI = SI ± 2

OUTSD = SI = SI ± 4



Note String Instructions:

Source → DS:SI

Destination → ES:DI

Program: .Model Small

.Data

DataX DB 50 dup(?)

.Code

.Startup

MOV DX, 0F1ABH

LEA DI, DataX

CLD; ~~at~~ auto increment

Back: INSB

ADD DX, 1 → DX doesn't increase only DI

loop back or SI ,50 I loop DX

.Exit

.End

⇒ XLAT instruction:-

BX AL No operands → Results → (AL)

Used in

index	SQR-table
0	0
1	1
2	4
3	9
4	16
5	25

.Model Small

.Data

SQR-table DB 0,1,4,9,16,25,39,49

.Code

.Startup

MOV BX, offset SQR-table

MOV AL, 6

XLAT

.Exit

.End

XLAT → MOV AH, 00H
MOV DI, AX
MOV AL, [BX +DI]

⇒ IN & OUT instruction:-

Notes: ① Use AL, AX, or EAX. ONLY!

② Port number
8-bit (Constant) ← → 16-bit (DX)

Example:

IN AL, 39H.
IN AX, 39H.
IN EAX, 39H.
IN AX, DX.

OUT 2FH, AL
OUT 2FH, AX
OUT 2FH, EAX
OUT DX, EAX.

Directives (Slide: 41).
.386
.486
.586 } Pentium (CMOVZ).
.686
.786

* ORG used only in full-segment approach.
* Slide 58 Imp. & ORG → !

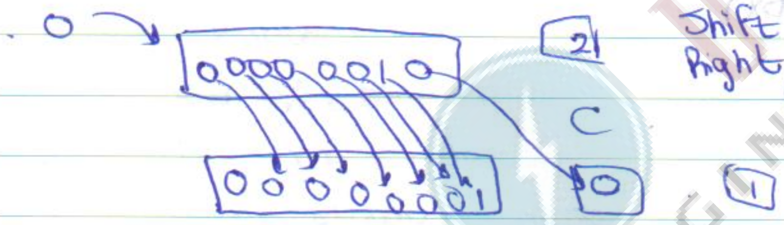
⇒ Shift AH 8-bit

00000010 ②

① SHL AH, 2 → 2 times shift

* لكل مرة بنعمل Shift
كأني بضرب (By 2

② SHR AH, 1



لكل مرة بعمل Shift Right
(بقسم على 2

*Note: Shift amount
Imm. Data ↙ ↘ CL

(If he gave me shift AH, BH → Invalid)

③ SAR AH, 1
if the # was -ve.

Slide 92 مهم جدا Ch.5 (Examples).
بجيب زيه

Rotations:-

OF = 0 always

CF = 1 → Add if the sum is larger than the size of destination

AF = 1 → if the sum of the two first numbers of the src. & dest. (15) (Fh) Decimal أكبر من

Sub ⇒ CF = 1 if the destination is smaller than the source. (ans. = -ve).

AF = 1 إذا كان أول رقم في الطرف الأيمن أصغر من أول رقم في الطرف الأول.

⇒ Chapter 5:-

① SCASB ⇔ AL - ES:[DI], DI = DI ± 1.

② SCASW ⇔ AX - ES:[DI], DI = DI ± 2.

SCASD ⇔ EAX - ES:[DI], DI = DI ± 4.

② Used with conditional loops : Note: loop

Conditional, Unconditional.

LOOPE = loopZ - Loop

loop NE = loopNZ) - REP.

REPE = REPZ

REPNE = REPNZ

LoopE = LoopZ

Loop if Eflag = 1 & CX = 0.

```
Back: SCASB ; AL - ES:[DI]
      JE GO.
   X: loop Back
      JMP over
   GO: Move byteptr [DI]; 'j'
      JMP X
   over:
```

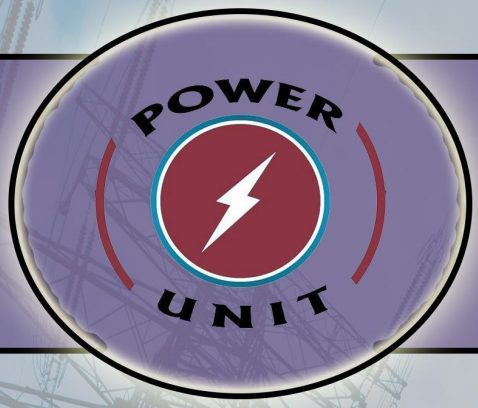
Question in exam:

MOV AH, 01

INT 21H

& I entered 1 2 3 4 5 6 7

⇒ Ans. ⇒ I take 1 , ~~or~~ ASCII $31



Assembly

Notes

Dr. Khaled Darabkeh
By: Mohamad 3amera

#بأفكارنا_نُبدع

7 H
10 | 72
2c
2L

0702
3232

~~Back SCAN B~~

opcode / short jump

* Unconditional JUMP.

short
opcode near jump.
Near
For

1 byte long
Eb + Relative Distance.

3 byte long
Eq + Relative Distance (2 byte)

5 byte long
outside current code segment

Relative distance longs
-128 - 127

Relative Distance range
±32K
(it is relative to IP)

Target logical address (it is relative to IP)

Target label

Target instruction

EA + IP + CS
opcode - 2 byte 2 byte

1200: 0010 GO:
1200: 0012
1200: 0013
1200: 0015
1200: 0016
1200: 0018
1200: 0019

Relative distance

JUMP GO

Target label.

Relative distance.

Target effective Address (s)!

Address #1 = 1200 X 10 + 0040 = 12010H

Address #2 = 12011H

* Required : (IOS)

① what is the target effective address?

② what is the type of Jump instruction?

→ Jump 04 ⇒ 2 byte long ⇒ short.

→ Target offset Address = IP ± Relative distance.
= 0002 + 04
= 0006

Note.

① Relative distance (04)
+ve

* Target effective Address

= CS * 10 + Target offset Address
= 10000 + 0006 = 10006 H.

23/12/25

⇒ what is the target instruction of jump next instruction?

Target offset Address = IP ± Relative distance

= 0009 + 17 = 0020H

Relative distance = 17 (+ve)

IP = 0009

we are done??

* Target instruction: mov bx, ax

⇒ what is the target instruction of jump start instruction?

Target offset Address = Ip + Relative distance.

IP = 24H (00100100) = (36)dec

Relative ~~Address~~ distance = DE (11011110) → 2's complement

00100001

+1

00100010 (-34)dec

Target offset Address = (36) - (34) = (2)dec = (0002)Hex

* Target instruction = mov ax, 1

* Relocatable distance (R)

* Actual relative distance (?)

* Next jump.

Target offset Address = IP ± Relative distance.

0200H = 000A + X

X = 0200H - 000A H

0200 - 1
01FF

= 01F6

* Actual machine instruction;

EQ F6 01

Jun

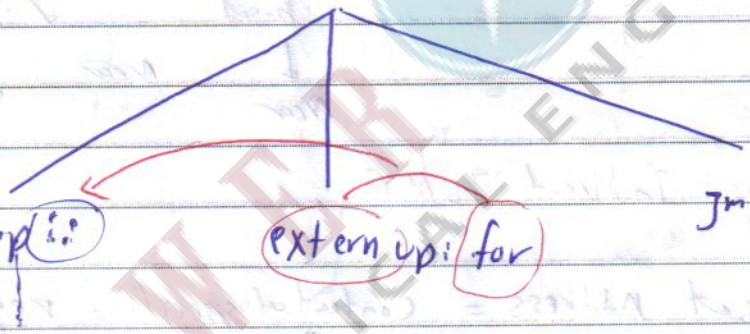
* Target effective Address :

IP = 0127

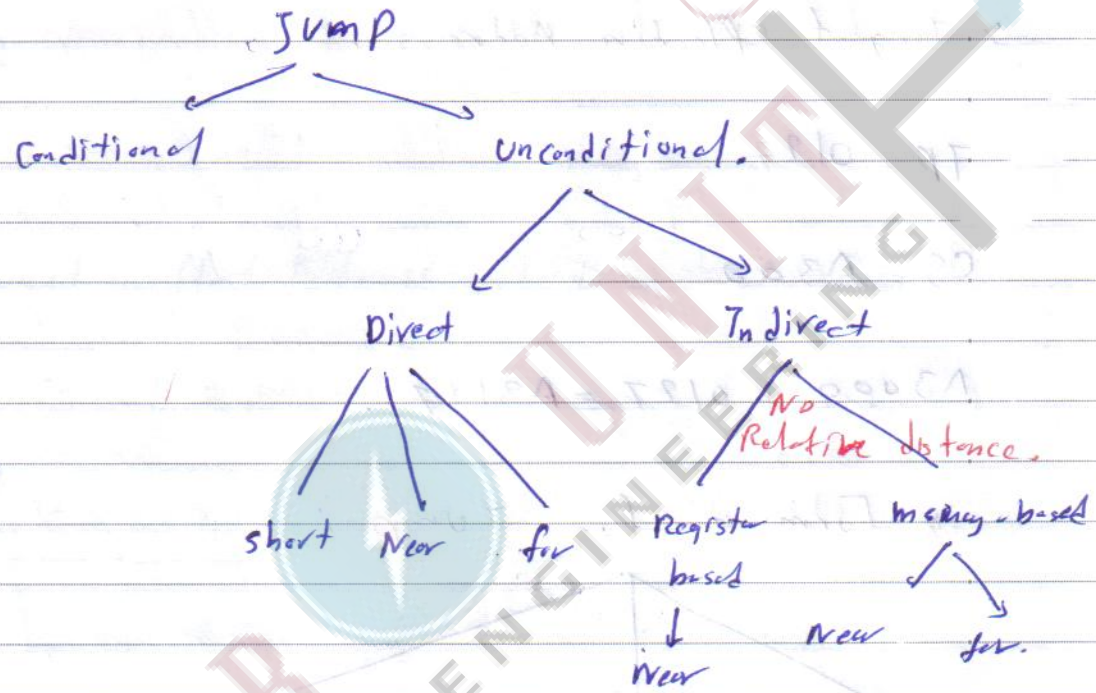
CS = A300

A3000 + 0127 = A3127.

* Form of far Jump :



Jmp up



* Register based Indirect JUMP!

Target offset Address = Content of specified register

Ex.

JMP AX

⇒ Target offset Adress = Content of AX register

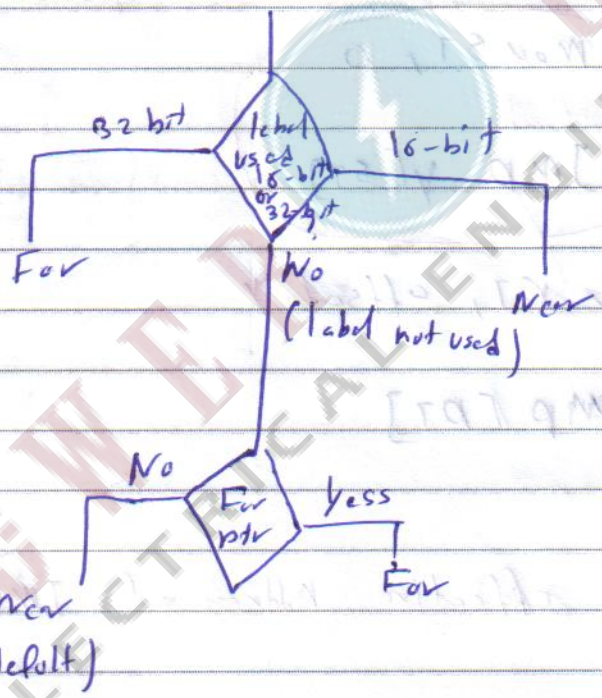
Keep in mind that

IP = Target offset Address

JMP EAX → 32-bit
→ Near

(Real mode) ⇒ EAX should never exceed 0000 FFFF H

* memory based Indirect jump :



Ex

```
model small
Data
X   DW   0100H
Y   DD   30000300H

Code
Startup
```

```
Mov ST, offset y          Mov ST, 0
JMP far ptr [ST]   Equal  JMP y[ST]
```

```
Mov DI, offset X
JMP [DI]
```

Required

① what is the Target effective Address of JMP y[ST] instruction ?

② what is the target offset address of JMP [DI] instruction ?

Jump Y[SI]

0300 H

DS:0000	00
DS:0001	03
DS:0002	00
DS:0003	03
DS:0004	00
DS:0005	30
DS:0006	

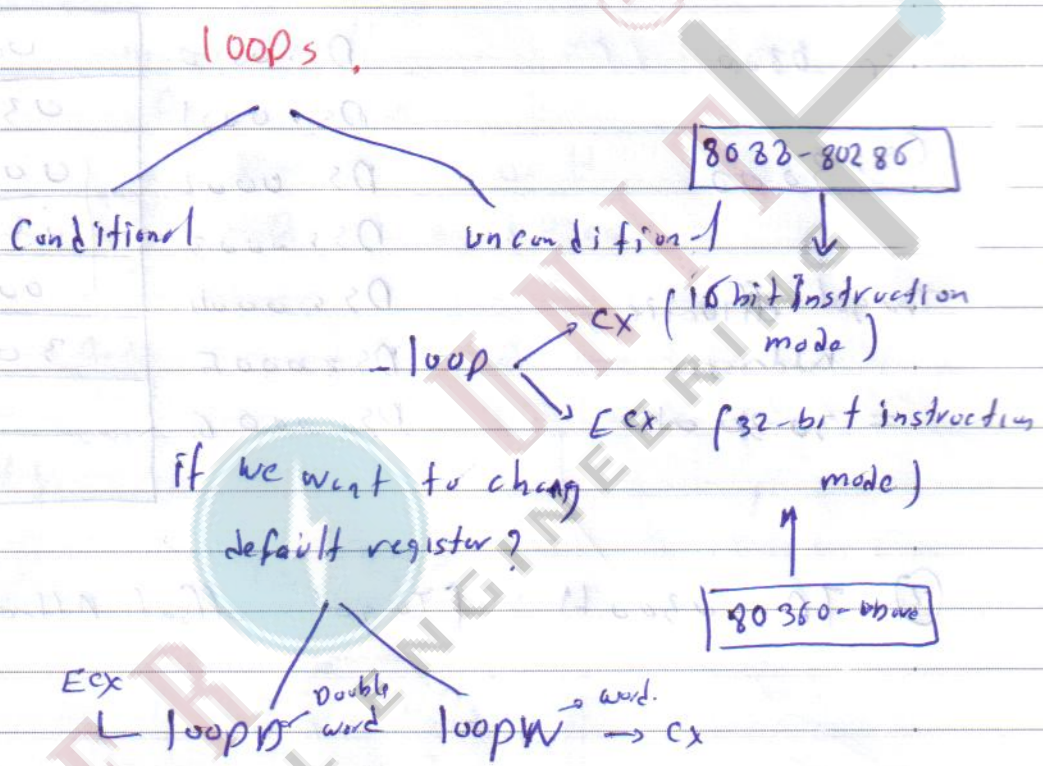
(1) IP = 0300

CS: 3000

Target effective Address

= 3030 H

(2) IP = 0300 H (Target offset Address)



* Conditional:

- loopE / loopZ
- loopNE / loopNZ

CX / ECX ? instruction mode.

- loopEW / loopZW
- loopNEW / loopNZW

* Near call = unconditional Near jump + push (IP)
SP (-)

Far call = unconditional Far jump + push (CS) (IP)
SP (-)

(CS, IP)
(1) (2)

Ex: slide (54)

Sol. Call 3byte long Near
Near (local)

(1) Target effictive address

Target offset address = IP ± Relative distance
(TOA) 0003H + 0FFFF
= 1002H

Target effictive address (TEA) = CS × 10 + TOA
= 11002H

(2) push IP
SS = A000H
SP = FFFFH

SP_{New} = FFFDH

A000 : FFFD	03H
A000 : FFFE	00H
A000 : FFFF	Top of the stack

SP

Ex: slide (57)

call → 5byte → far
→ Global.

$CS_{old} = 1000H$

$IP_{old} = 0005$

$CS_{New} = 1100H$

$IP_{New} = 0002H$

TEA?

$TEA = CS_{New} * 10 + IP_{New}$

$= 11000$
$\quad 0002$
$\overline{11002H}$

$SS = A000H$

$SP = FFFEH$

$SP_{new} = FFFBH$

(-4) → CS(1), IP(2)

A000 : FFFB	05H
A000 : FFFC	00H
A000 : FFFD	00H
A000 : FFFE	10H
A000 : FFFF	Top of the stack

Program 1 model Tiny

Code

start up

mov SP, 0010H

mov SS, 0C000H

3000:0020 ← Call X

3000:0023 move BL, 30H (***)

Ext.

X proc uses AX

mov AX, 80H (**)

X RET

ENDP

END

(1) Find the TEA of the call instruction given that the values of IP & CS at this instruction are 0020H & 3000H, respectively.

(Note: The value of X = 30H)

(2) find the value of SP at label (**) and (***)?

(1) TEA? Call → Near (local)

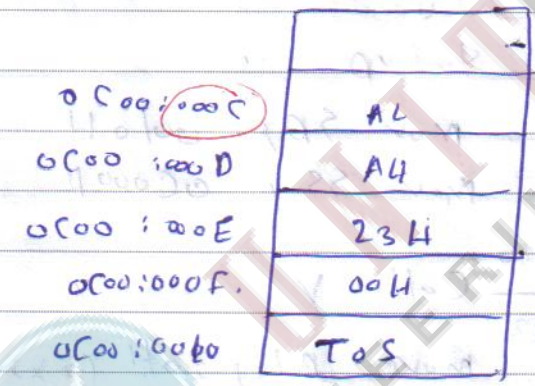
X → relative dist. (0030H)

TOA = IP + ROD = 0023 + 0030 = 0053H

TEA = CS×10 + TOA = 30053H

② SP| = 000CH

label (***)



SP| = 0010H

label (***)