CSE 331 / L-04 / 22.02.2024 /

8086 ⇒ 16 bit
80386 ⇒ 32 bit
Core-2 ⇒ 64 bit ↓ | Latest processor always backward compatible

Recap

⊛ RCX
- used to run loop
- count value will decrease. i.e.: 5, 4, 3, 2, 1, 0

⊛ RDX
- Dividend must be double bit of divisor

- $12\overline{)120}($ Qu·· → 8bit
  - 12 → Divisor → 8bit
  - 120 → Dividend → 16 bit

- if Divisor 8 bit, Dividend 16 bit
  Divisor 16 bit, Dividend 32 bit

$$\begin{tikzcd}
& \text{Dividend 32 bit} \arrow[dl] \arrow[dr] & \\
\text{DX (H)} & & \text{AX (L)}
\end{tikzcd}$$

⇒ DIV BX

$$\begin{tikzcd}
& \text{BX} \arrow[dl] \arrow[dr] & \\
\text{DX} & & \text{AX}
\end{tikzcd}$$

⇒ MUL BX

⇒ AX → 16 bit
  BX → 16 bit
  ―――――――
  DX-AX → 32 bit
  (H) (L)

⊛ RBP

⇒ Physical Address = Seg. × 10 H + offset (BP)
↳ Then segment will be $\underbrace{\text{DS/SS}}_{\text{General Purpose}}$

⊛ RDI & RSI

$$\begin{tikzcd} & & & \text{String} & \\ \Rightarrow & \text{MOVS} \arrow[urr, bend left] & \text{DI} \arrow[dr, bend right] & \text{SI} \arrow[r] & \text{Source} \\ & & & \text{Destination} & \end{tikzcd}$$

⊛ Flag Register Sequence, need to accurate.

DIT ⇒ Control Flag

Others are status Flag

⊛ C ⇒ Carry ⇒ index 0

- Hold carry or borrow after addition or substraction $\boxed{\text{end.}}$

if carry exist here, A=1

$$\begin{array}{r} 1101\ \boxed{0}111 \\ 1101\ \boxed{1}101 \\ \hline (+)\quad \boxed{1011\ \ 0100} \end{array} \qquad \begin{array}{|l} C = 1 \\ P = 1 \\ A = 1 \end{array}$$

→ 4× 1, even (0× 1 is also even)

⊛ P = Parity ⇒ index 2

- count the number of 1 in the result.
- 0, or even number of 1, then even parity, P = 1
- odd number of 1, then odd parity, P = 0

⊛ A = Auxiliary carry ⇒ index 4

- holds the carry or borrow after addition or substruction end in a bit position,

for addition ⇒ 3 to 4

for substraction ⇒ 4 to 3

⊛ Z ⇒ Zero ⇒ index 6

- if result is 0, Z = 1 (True)
- if result is not 0, Z = 0 (False)

⊛ S ⇒ Sign ⇒ index 7

- if result is unsigned, S = 0
- if result is positive, S = 0
- if result is negative, S = 1

⊛ O ⇒ Overflow ⇒ index 11

- if result overflow the machine bit size, O = 1
  else, O = 0

⊛ T ⇒ Trap ⇒ index 8

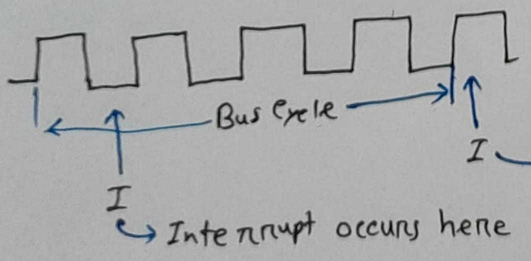
- used for debugging, step by step.
- if trap, t = 1
  - program will execute one by one instruction.

  else, program will execute all instruction at once. consecutively.

8086
⇒ Bus Cycle 4
⇒ 800 ns

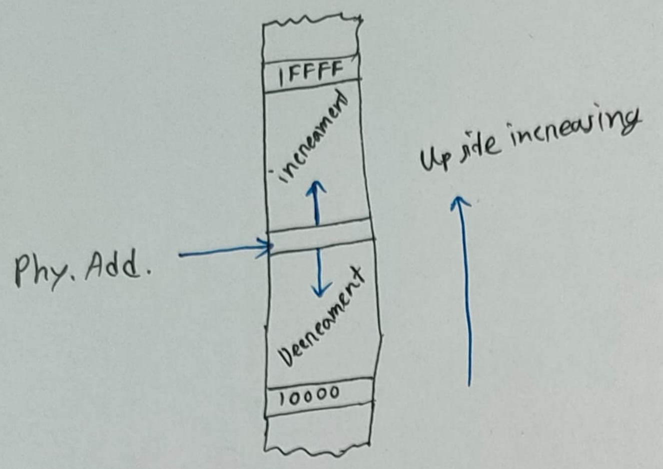
⊛ I ⇒ Interrupt ⇒ index 9

-



But, it will execute after the bus cycle

⊛ D ⇒ Direction ⇒ inden 10



D=0, increament, Go up

D=1, decreament, Go down
↓
Phy. Add. decrease by 1

⊛ Segment Register

- Generate memory address
- 4 types of segment register

⊛ (i) CS
- Code Segment
- holds code of program used by microprocessor

(ii) DS
- Data Segment
- contains data used by a program

(iii) ES
- Extra Segment
- additional data segment used by some instructions to hold destination data

(iv) SS
- Stack Segment
- memory used for the stack

- Real mode memory address must consist of a segment address plus an offset address.

- Segment Address
  - defines the beginning address of any 64k-byte memory segment

- Offset Address
  - selects any location within the 64k byte memory segment.
  - also called as displacement

- Once the beginning address is known, the ending address is found by adding FFFF H.

- Real mode addressing

  CS : IP

$$\begin{tikzcd} \Rightarrow & 1000 : 2000 \arrow[dl] \arrow[dr] & \\ \text{Segment} & & \text{offset} \end{tikzcd}$$

- A memory segment can touch or overlap, if 64k bytes of memory are not required for a segment.

-
  - CS ⇒ IP ⇒ instruction address
  - SS ⇒ SP or BP ⇒ Stack address
  - DS ⇒ BX, DI, SI , 8 bit, 16 bit ⇒ Data address
  - ES ⇒ DI for string instruction ⇒ String destination address

- MOVS $\rightarrow$ String, then DI and ES

  otherwise, DI will go with DS

Chapter-3

Addressing Mode

⊛ 7 types of addressing mode ⇒ 8086

8 types of addressing mode ⇒ 80386 & onward

⊛ each instruction consist of 4 Parts or field

Label:	OP-Code	Operand	; Comments

Label: Start with symble or special character: @, $, -, ?

length: 35

```
NSU: | mov AX, BX
     |    :
     |    :
       Loop NSU → Label
```

NSU: MOV AX, BX

AX → Destination

BX → Source

⊛ We must write comments in exam as well as a programmer also

AX = 1000 H

BX = ABCD H

⇒ MOV AX, BX

AX = ABCD H

BX = ABCD H

→ value will be copied to the destination

L-5/24.02.2024/

Figure 3-2
Lecture 2.3
Slide Page-40

***

⊛ 3rd Bracket means offset,

$$\underbrace{[BX] \text{ or } [1234H]}_{\text{Offset Value}}$$

⊛ MOV [1234H], AX

[1234H] → Destination (8bit) → Value of offset

AX → Source (16 bit)

∴ Phy. Add = DS × 10H + 1234 H
= 11234 H

No increament or decreament identifier
⇒ then automatically it will increase
11235H ⇒ AH
11234H ⇒ AL
} 16 bit

⊛ Base + Index

MOV $\underbrace{[BX + SI]}_{\text{Value of Offset}}$, IP

BX → First register will defined the Segment, ~~Do~~ Here, Data Segment

⇒ also written as [BX][SI]

⊛ Table 4·2 (Page-41)

⊛ Some Restriction:

i. Size mismatch is not allowed ⇒ MOV AX, CL ✗

ii. Segment to segment transfer not allowed ⇒ MOV DS, CS ✗

iii. Memory to memory transfer not allowed.
- except PUSH & POP & String related

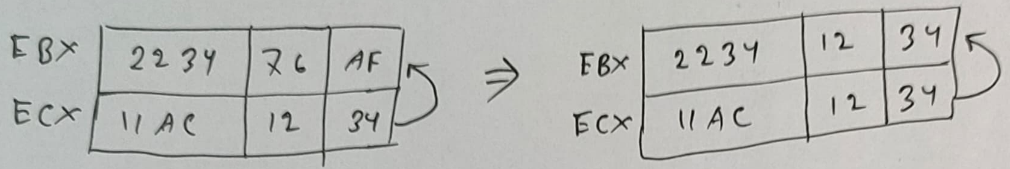
iv. Constant can't be a destination

v. Segment cannot be loaded with data

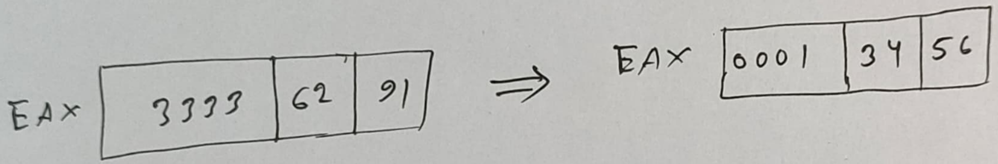
table 3-1
Slide Page -45

MOV DS, 1234H ✗

✱ MOV BX, CX in higher processor 80386



✱ MOV EAX, 13456H



✱ The content of destination register or destination memory location change for all instruction except the cmp and TEST instruction.

✱ Hexadecimal ⇒ must contains H

Decimal ⇒ numbers only

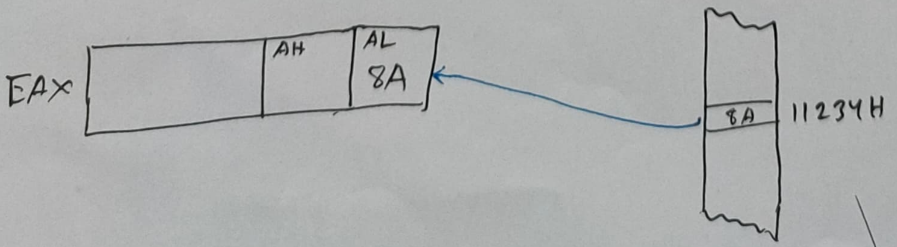
ASCII ⇒ 'character'

Binary ⇒ must contains B/Y

table 3-2
Slide Page -50

✱ MOV AL, [1234H] → Phy. Add = DS×10H + 1234H = 11234H

Offset



Slide - Page - 53

⊛ Registen Indinect Addnessing

- Allows data to be addnessed at any memony location thnough an offset addness held in any of the following negistens
  - BP, BX, DI and SI

⊛ MOV AX, [BX] ⟶ Phy Add. = DS × 10H + BX

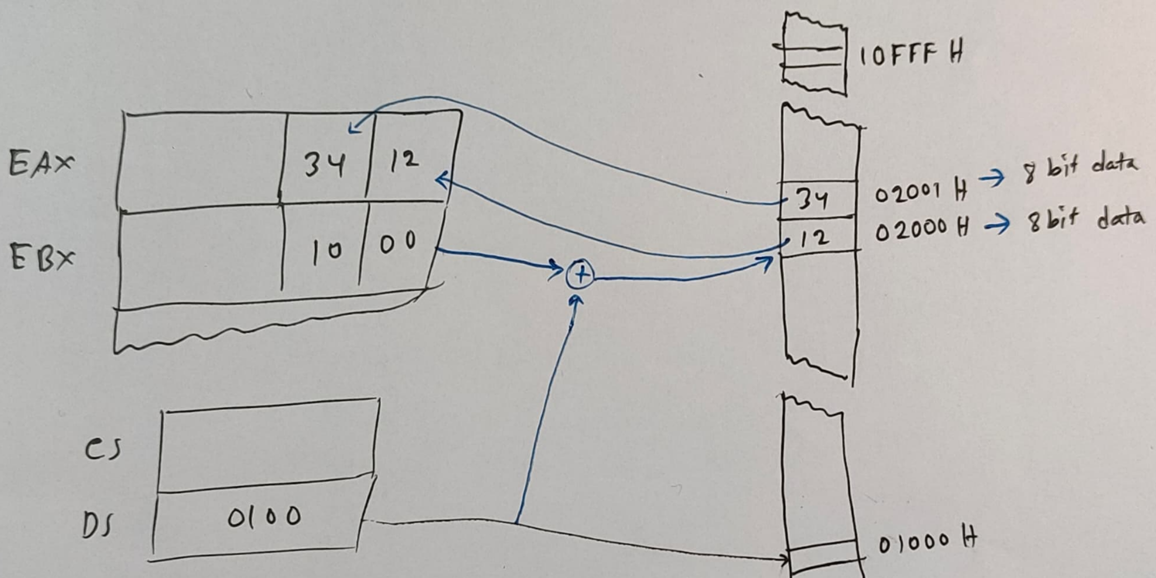
Offset

16bit then Sounce must be 16 bit

= 0100 H × 10H + 1000 H

= 01000 H + 1000 H

= 02000 H



⊛ Special assemblen dinective

- BYTE, WORD, DWORD on QWORD PTR

$\boxed{\text{Table- 3-5 Slide Page 58}}$

⊛ Base-Plus-Inden Addnessing

- Similan to indinect addnessing ~~mo~~ because it indinectly addnesses memony data

$\boxed{\text{Figune 3-8 Slide Page- 61}}$

$\boxed{\text{Slide Page- 63, 64, 66, 67}}$

⊛ Base Relative Plus Index Addressing

- often addresses a two dimensional array of memory data

Slide Page - 72, 74, 75

Assignment - 1

Lecture 2,3

Slide Page - 77

Exercise : $\underbrace{31, 32, 33}$

a & b

+

mention the address mode type

Due - 29.02.2024