Lecture-01

Introduction to Microprocessors

Sat / Sun / Mon / Tue / Wed / Thu / Fri

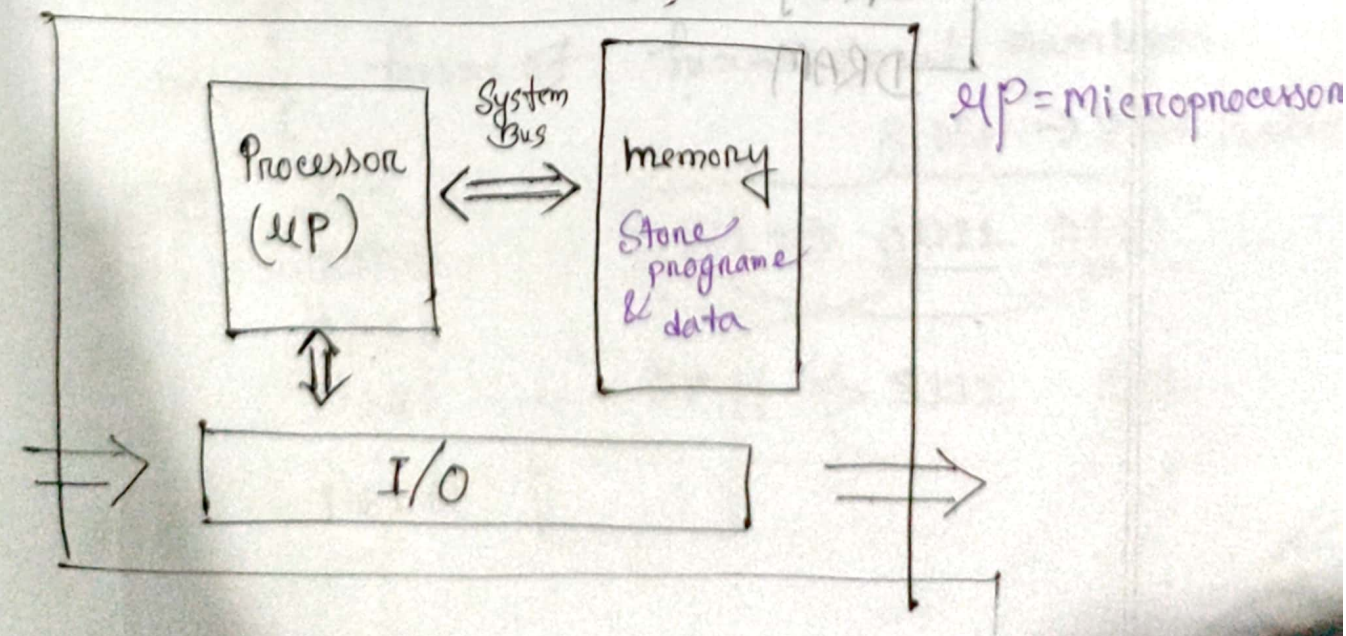
Date :.02./10../..2023

-> If you want to do programming, you need a processor.

-> From pentium series — 80586 (pentium 1) -1993
We can engage with multimedia using computer.
↓
(watch movies)

8085
8086 → We need primary memory (RAM/ROM)
80186 ↓ ↓
286 volatile
386
486
80586 → P_1, P_2, P_3, P_4 ... C2D ... ci7

Computer System:

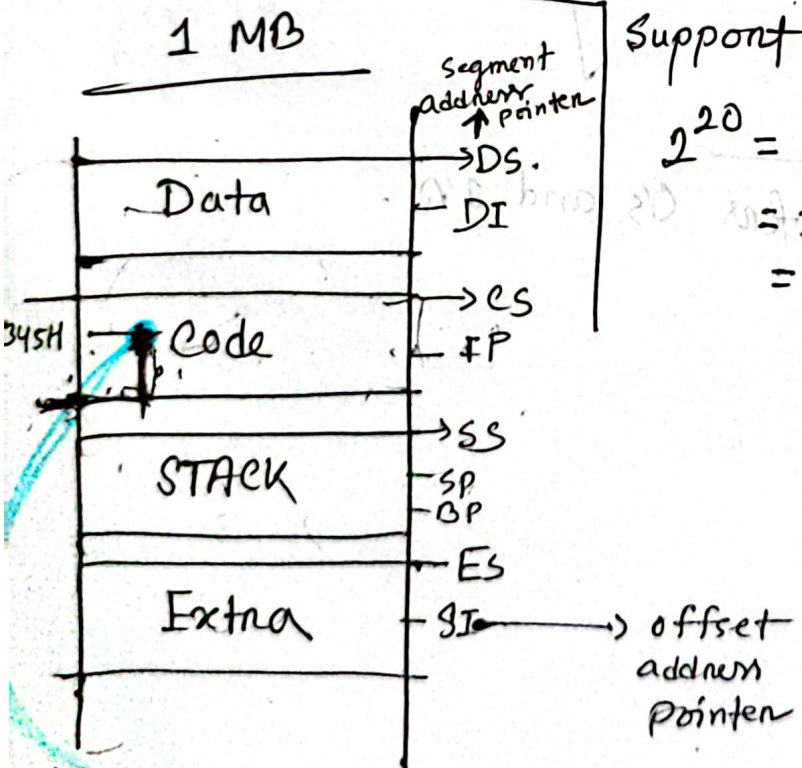


μP = Microprocessor

Architecture 8086

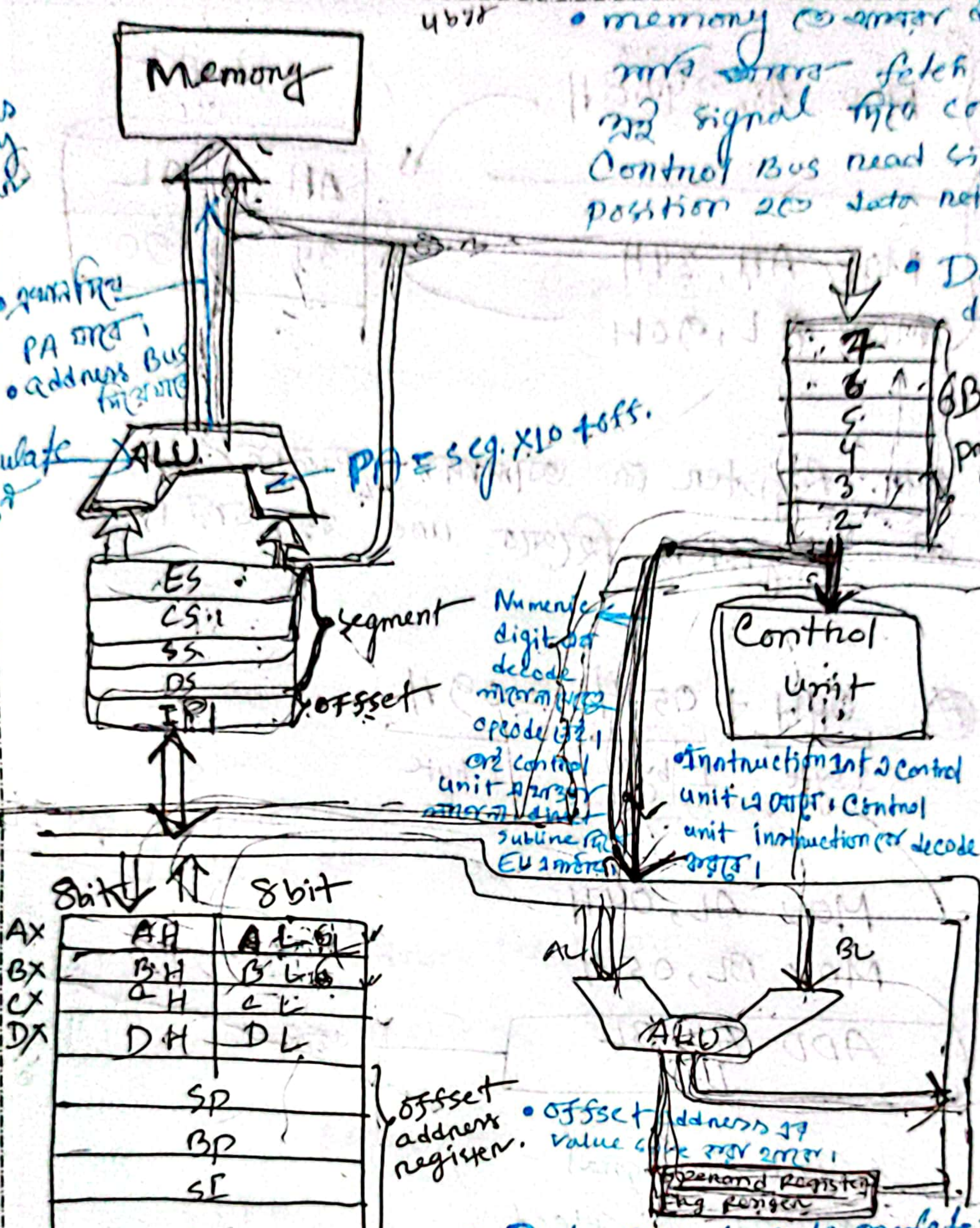
- 16 bit microprocessor
- It's mean
- ata time 16 bit data এর উপর process করতে পারে।
- pipeline microprocessor
  → যখন EU তে execution চলবে তখন BIU memory থেকে instruction fetch করা শুরু করে দিবে (Parallel হয়ে চলে execution - Fetch)
- 20 bit address bus support করে।

$2^{20} = 2^{10} \times 2^{10}$
$= 1KB \times 1KB$
$= 1MB$



PA = (Segment address X10H) + offset address
= (10000H X 10H) + 2345H
= 12345H

MOV AX,BX
Add AL,BL



- memory তে থাকার data গুলোকে আগে আবার fetch করতে হবে। এই signal দিবে control Bus. Control Bus read signal দিলে ঐ position হতে data retrive হয় μP তে আসবে।
- Data transfer data আসা হবে।
- address Bus memory particular ... select করে।
- এখানে দিয়ে PA আসবে।
- address Bus দিয়ে যাবে
- PA calculate এর জন্য
- Numeric digit er decode আলাদা রকম opcode হয়। এর control unit এ সরাসরি ... subline দিয়ে EU তে আসবে।
- Instruction 1st a Control unit এ আসে। Control unit instruction কে decode করবে।
- 8086 microprocessor maximum 6 byte এর instruction Suffient হয়।
- Offset address এর value এখানে ...
- Data গুলো 6 byte prefetch queue (FIFO) তে জমা থাকবে।

1 - EU
2
3
4
5 Prefetch
6
7
8
9
10

1000100 opcode

MOV AL,BL

opcode part — MOV AL, 25H — 1 byte

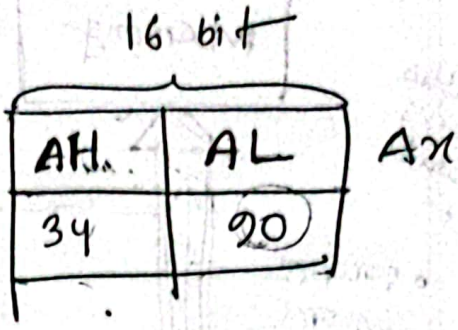
- opcode হলো 0&1 এর byte pattern প্রত্যেকটা রেজিস্টারের জন্য আলাদা আলাদা opcode থাকে। Numeric value এর opcode লাগানো

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date :...../...../.....

MOV AX, 3490H

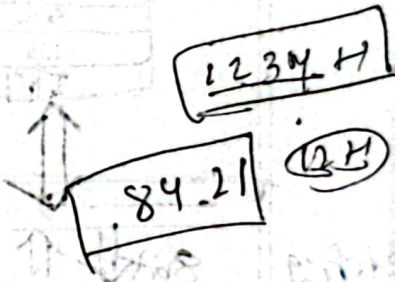
MOV AH, 34H
MOV AL, 90H

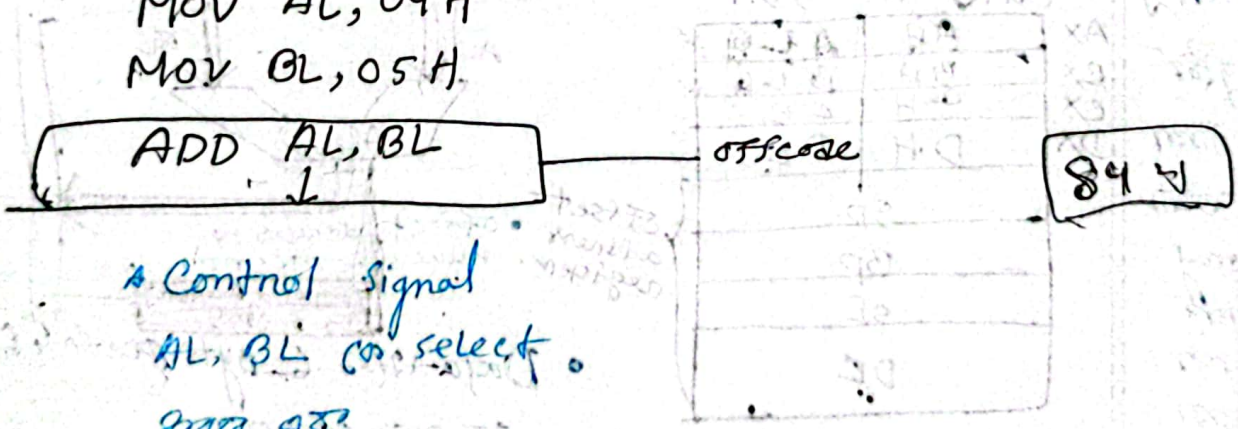


• 16 bit Register কে আলাদা আলাদা 8 bit Register হিসেবে use করা যায়।

$\underbrace{04H}_{1\,byte} + \underbrace{05H}_{1\,byte} = \underbrace{09H}_{1\,byte}$

MOV AL, 04H
MOV BL, 05H
ADD AL, BL





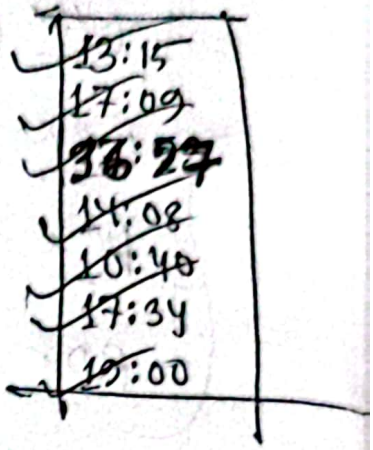
* Control signal
AL, BL কে select.
তারপর
ALU কে add এর নির্দেশ দিবে।

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date :...../...../.....

Registen

Segment Registen
Special purpose Registen
Flag Registen

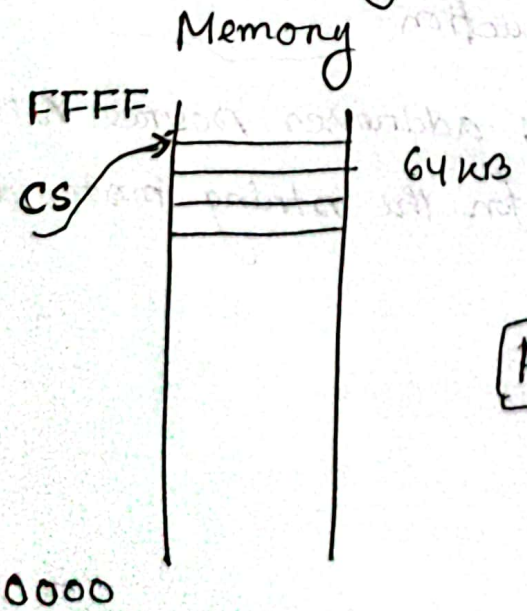


Segment Registen

CS – code segment : hold the code
SS – Stack segment: define area of memory
DS – Data segment: contains most data used by a prg.
ES – Extra segment: additional "

BIU — Fetch, Decode
EU — Executing

* used to address the memory
* generates memory addresses



* Segment Registen are used to address particular segments in this memory.

PA = seg X 10H + Offset

- Book (Memory)
  - Chapters (segment
    - Topics [page no]

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date :...../...../.....

Multi purpose Registers :

- RAX (accumulator) : is used for instruction such Mul, Div and...
- RBX (base index) : holds offset address of a location in memory
- RCX (count) : holds count for various Instruction
- RDX (Data) : holds a part of result from a multiplication & dividend before devision

RBP (Base pointer) : points to a memory location for memory data transfer

RDI (destination index) : destination string data

↑ string instruction ↓

RSI (source index) : addresses source string data. for the string instruction.

Special Purpose Registers

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date :...../...../.....

RIP (Instruction pointer):

Ip points to the next instruction in a program, is used by µp to find the next sequential instruction in a program.

RSP (stack pointer):

Addresses an array of memory called stack. Stack memory stores data using RSP.

RFlags:

condition of µp and control its operation.

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date :...../...../.....

Flag Register : of 8086:

Shows the status of Result.

→ Contains the status of current result.

STATUS Flag:

↓ changed by the ALU

Carry Flag:

```
  1111
  0001
 -----
 10000
```

↳ If Carry ~~flag~~ coming out of the MSB. That mean carry Flag = 1.

Parity Flag:

1000 0011

here 3 one's

so, odd parity.

So; Parity Flag = 0 ✓

Even Parity = 1

odd Parity = 0

Auxiliary Carry:

higher nibble | Lower Nibble (4 bit)

```
  1111 | 1111
  0000 | 0001
 ------|-----
 10000 | 0000
```

- lower Nibble (LSB) → higher nibble → Carry out → Auxiliary carry = 1.

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date :...../...../.....

④

Zero Flag:

Zero Flag = 1 → Result is 0

Zero Flag = 0 → Result not 0.

Sign Flag:

1000 0001

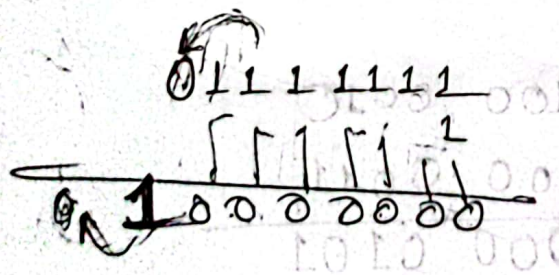
If MSB is 1, number is negative = 1

If MSB is 0, " positive = 0

MSB Result = Sign Flag

Overflow Flag:

If number goes out of range then overflow = 1



OF = 1

Sign Flag check এর পরে overflow flag check করা লাগবে।

OF = 1 → Sign Flag give wrong sign

OF = 0 → Sign Flag give right sign

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date :...../...../.....

(#)

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

OF	SF	ZF	AF	PF	CF
0	0	0	0	1	0

-80 — -01, 00. 7FH

যদি overflow হবে।

-128 . . . 0 . . +127

7F
+1
80

(#)

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

Same হলে Same

হলে = 0

ভিন্ন হলে = 1

OF	SF	ZF	AF	PF	CF
0	0	0	1	1	0

8 4 2 1

1 0 0 0 -

0 1 0 1 -

(#)

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

OF	SF	ZF	AF	PF	CF
1	1	0	0	0	0

added two positive numbers (42, 43) never be negative. Because overflow 1.

Sat / Sun / Mon / Tue / Wed / Thu / Fri

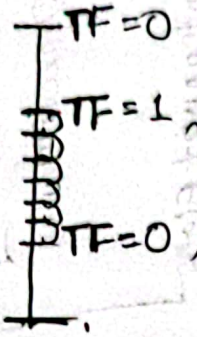
Date :...../...../.....

Control Flags: → Changed by us.

TRAP Flag:

Single stepping

↳ Debug every line by line | 1 = perform single stepping
0 = do not ,,



Interrupt Flag:

process gets interrupt from external sources.

↳ Mobile screen on ~~thakle~~ অবস্থায়
touch করলে interrupt হয়।
Screen off ~~thakle~~ অবস্থায় interrupt
হয় না, because we can disable interrupt.

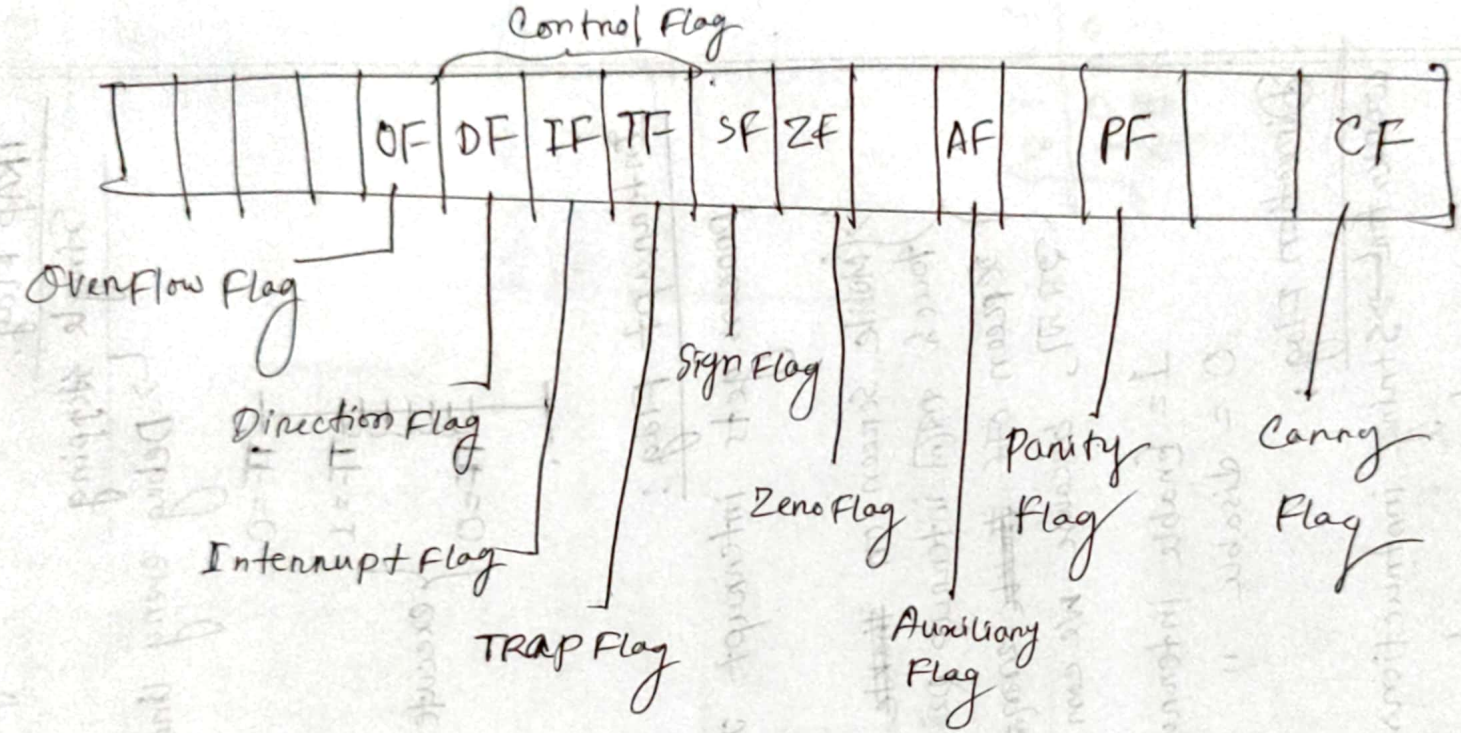
1 = Enable interrupt
0 = disable ,,

Direction Flag:

works for → String instructions

Choose auto incrementing addresses or decrementing addresses.

1 = auto ~~increment~~ decrement
0 = auto increment


Control Flag
OF
DF
IF
TF
SF
ZF
AF
PF
CF
Overflow Flag
Direction Flag
Interrupt Flag
TRAP Flag
Sign Flag
Zero Flag
Auxiliary Flag
Parity Flag
Carry Flag

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date : ...../...../.....

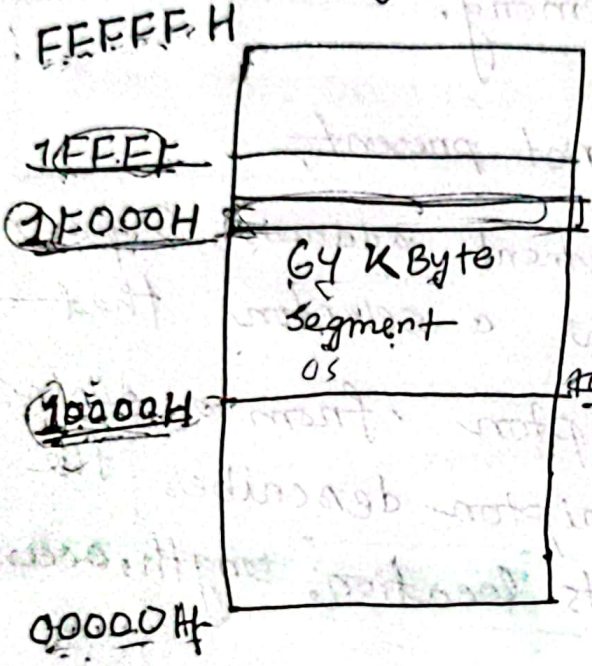
Real mode memory addressing : access only first 1 MB

operation → to address only 1MB of memory.

up addresses memory using logical address.

Memory (1MB)

conventional/ Dos memory system.



offset = F000 → Windows Real mode use 가능

10000H → Real mode allows application software - 8086/8088

⊛ 64 bit cannot run in real mode

Segment Register
2000H
Starting address
20000H
Ending address
2FFFFH
offset

▸ Segment address ↳ memory

Sat / Sun / Mon / Tue / Wed / Thu / Fri

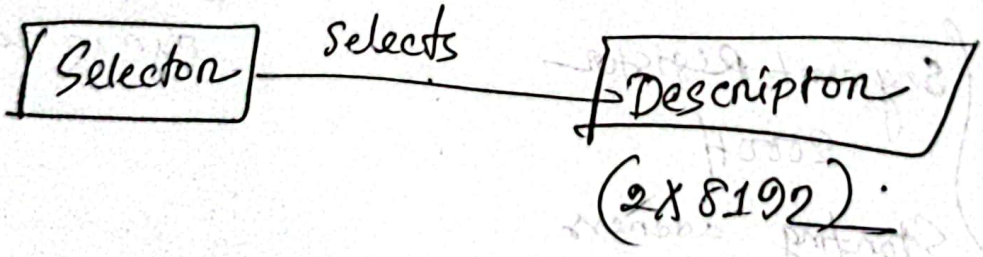
Date :...../...../.....

Protected mode memory addressing: above 80286

→ Allows up to access all the data & programs located above 1 Mbyte of memory. as well as within 1st 1 Mbyte of memory.

→ Segment address not present.

↳ In place of segment address, Segment register contains a selector that selects a descripton from a descripton table. The descripton describes the memory segments location, length, access rights.



→ Offset address can be 32 bit no. instead of a 16 bit no in the protected mode.

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date :...../...../.....

2^{10} = 1 KB

2^{20} = 1 MB

2^{30} = 1 GB

2^{40} = 1 TB

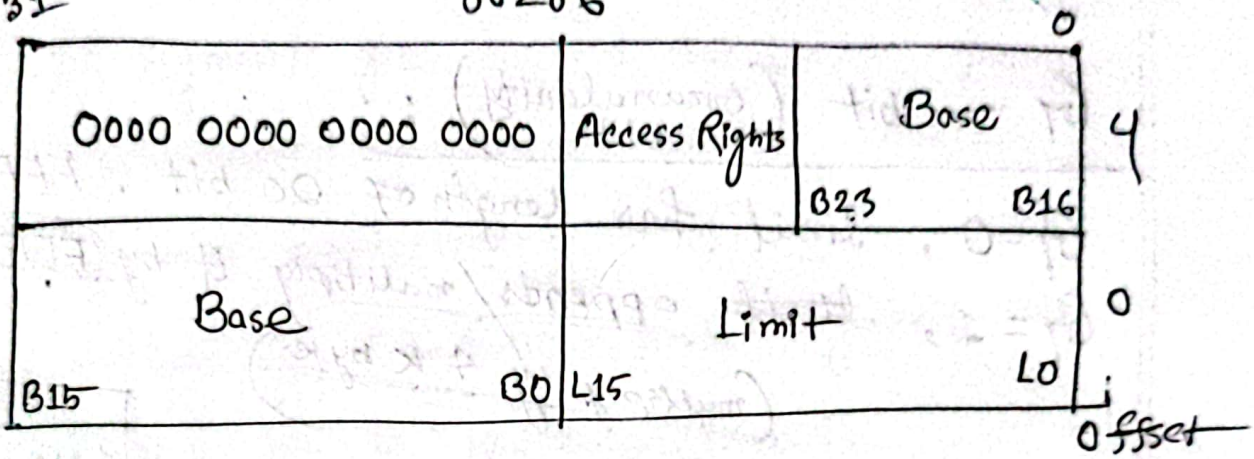
Selectors and descriptors:

→ Selects one of (2X8192) descripton

Segment Register

Selector

80286



Descripton: → Global descripton → System → applies all Progra
→ Local descripton → Applications

(i) Location
(ii) Length
(iii) Access Rights

Base address → starting location of memory segment.

80286 → 24 bits (Segment at any location 16 MB memory)

80386 → 32 bit (4 GByte of memory)

$24 = 2^{24} = 2^4 2^{20}$

$= 16$ MB

$32 = 2^2 \cdot 2^{30} = 4$ GB

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date :...../...../.....

Length → size of segment.

Limit → contains the last offset address in a segment

80286 → 16 bit [Base + Limit] (1-64 KB)

80386 → 20 bit (1 MB or 4 KB or 4GB in length)

G — bit (Granularity bit):

G=0, limit has length of 20 bit. FFFFF H

G=1, ~~limit~~ appends / multiply it by FFFH

(multiply by 4 K bytes)

FFFFFFFFF H

1MB X 4KB

4 GB

G=1

D-bit:

D=0, 16 bit (operand)

D=1, 32 bit

AV bit:

AV=0, Segment not available

AV=1, " available

1MB * 4KB

2^{20} 4×2^{10}

2^{30}

4GB

2^{12} 2^{20}

2^{32}

Sat / Sun / Mon / Tue / Wed / Thu / Fri

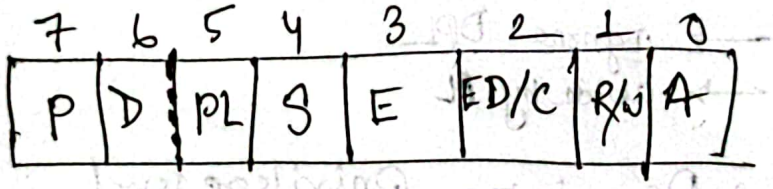
Date :...../...../.....

Access Rights Byte:

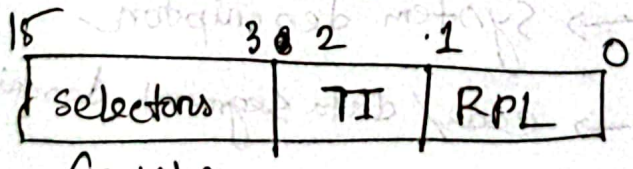
→ Contains access to protected mode segment

→ If segment goes beyond this limit, program is interrupted by microprocessor indicating a general protection fault.

→ Can specify wheather a data segment can be written on is write protected.



Access Rights Byte



Segment Register (80286 – core2)

- TI=0 (Global descripton)
- TI=1 (local descripton)

RPL – Request Privilege level of a memory segment

PL(00) – highest priv

PL(11) – lowest priv

00 – high

10

01

11 – low

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date :...../...../.....

Access Rights — Segment register

- PL < RPL

access is granted.

- A = 0 → Segment not accessed
- A = 1 → Segment can be accessed

- R = 0 → not read
- R = 2 → read

- W = 0 → not write
- W = 1 → write

- ED = 0 → Segment expand upward (data segment)
- ED = 1 → Segment expand downward (stack segment)

- C = 0 → Ignore DPL
- C = 1 → abide by PL

DPL → Description Privilege level.

- S = 0 → System description
- S = 1 → code / data segment description

- P = 0 → description undefined
- P = 1 → Segment valid base + limit

- E = 0 → description describe data segment
- E = 1 → " " code "

Chapter-03
Addressing modes

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date :...../...../.....

MOV AX, BX

Addressing modes : (Bharat)

mov destination, source

→ In which an operand is given in an instruction

↓

25

(1) Immediate:

- data in the instruction

Eg: mov CL, 34H; CL ← 34H

mov CX, 2000H; CX ← 2000H

(2) Register:

- data inside the Register

Eg: mov CL, BL ; CL ← BL

mov CX, BX ; CX ← BX

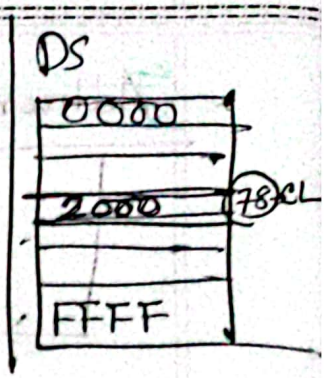
INC BX ; BX ← BX+1

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date :..../..../.....

③ Direct:

- address in the instruction
  - ↳ Telling the processor go to the address from there you will get the data



mov CL, [2000H] ; CL ← DS : [2000H]

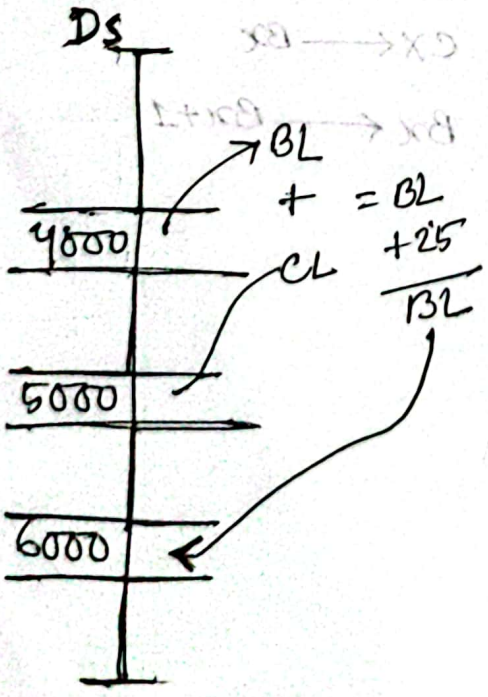
offset address — Data Segment — offset address

[] square indicating address

mov CX, [2000H] ; CL ← DS: [2000H]
CH ← DS: [2001H]

mov [2001H], CL ; DS: [2001H] ← CL

One location always carry 1 Byte of Data



MOV BL, [4000H] → Direct
MOV CL, [5000H] → Direct
Add BL, CL ⟶ Register
Add BL, 25H ⟶ Immediate
MOV [6000H], BL → Direct

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date :

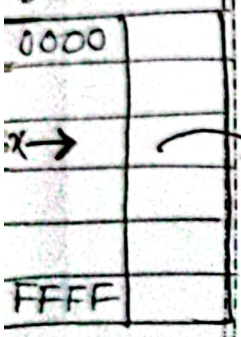
④ Indirect : ——> 32 type

[more than 95% of operation using indirect addressing]

Well developed addressing mode

- Address in Register

↳ In the instruction there will be a register, processor will go to that register. That register will given address. processor will go to the address. From there processor get the data.



(I) Register Indirect: (address ← Register)

Eg: mov CL, [Bx] ; CL ← DS: [Bx]

mov cx, [Bx] ; CL ← DS: [Bx]

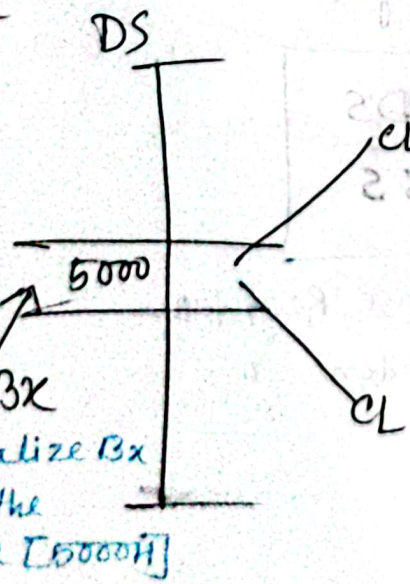
CH ← DS: [Bx+1]

arr [9] - direct

i = 9

arr [i]

↳ indirect



MOV CL, [5000H] ; CL ← DS: [5000H]

Direct addressing

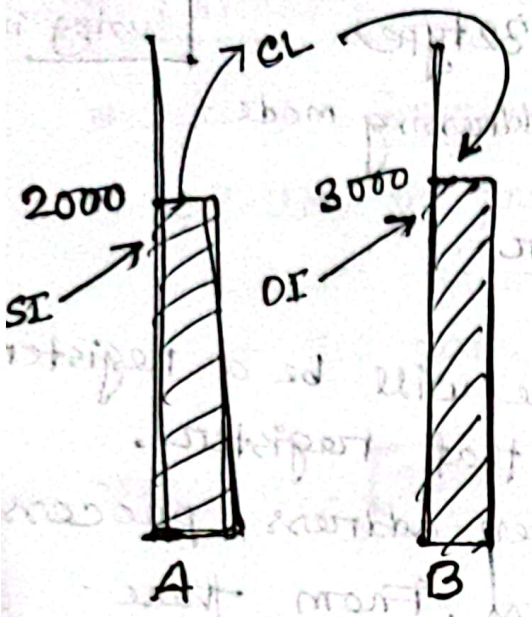
MOV Bx, 5000H

MOV. CL, [Bx] CL ← DS: [5000H]

Indirect addressing → By increment Bx we can get multiple address value

use for series of location

• Initialize Bx with the value [5000H]




```
MOV SI, 2000H
MOV DI, 3000H

MOV CL,[SI]
MOV [DI], CL
INC SI
INC DI
```

Block trasfer A to B

AX
BX → only 1 can use for memory address. (register)
CX
DX

BP, SI, DI → Offset Register can use

Bx, SI, DI ⟶ DS
BP ⟶ SS

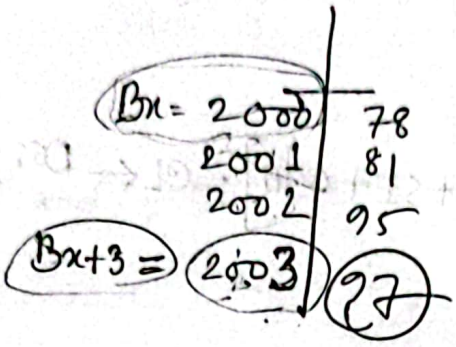
Bx, BP ⟶ Base Register
SI, DI ⟶ Index "

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date :...../...../.....

(II) Register Relative :

(address ← Register + displacement)



MOV CL, [Bx+03H] ; CL ← DS:[Bx+03H]

MOV CL, [BP+05H] ; CL ← SS:[BP+05H]

(III) Base Indexed :

(address ← base register + index Register)

Eg: MOV CL, [Bx+SI] ; CL ← DS:[Bx+SI]

MOV CL, [BP+SI] ; CL ← SS:[BP+SI]

Bx + index

By incrementing index we can get series of data.

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date :...../...../.....

(IV) Base Relative Plus indexed :

(address ← base Register + index Register + displacement)

Ex:

MOV.CL, [Bx+SI+03H]; CL ← DS: [Bx+SI+03H]

Implied:

STC; CF ← 1

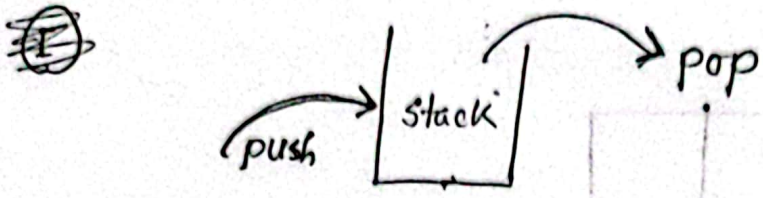
CLC; CF ← 0

DAA; Decimal Adjust after addition

Sat / Sun / Mon / Tue / Wed / Thu / Fri

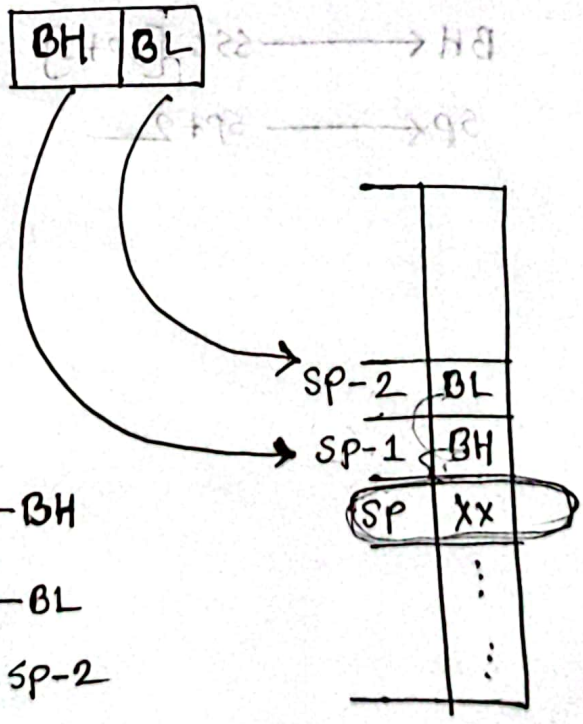
Date :...../...../.....

push & pop Instructions:



- push and pop don't support Immediate addressing mode.
- must be 16 bits (word) Register

(I) push Bx:



SS: [sp-1] ← BH

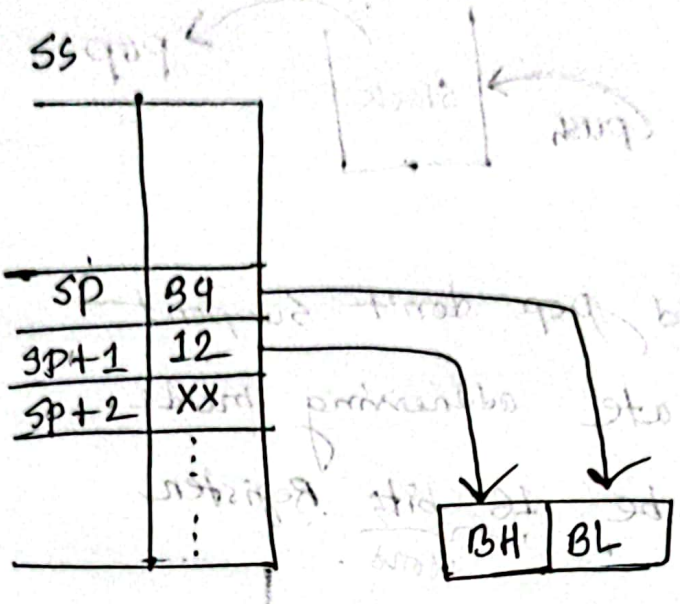
SS: [sp-2] ← BL

SP ← SP-2

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date :...../...../.....

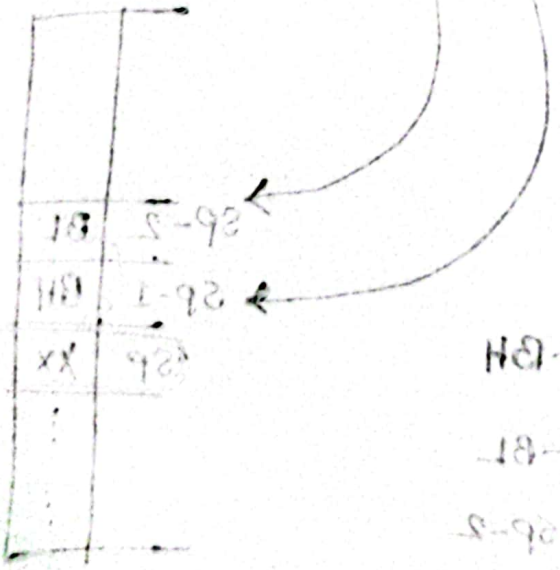
(II) Pop Bx :



BL ← SS:[SP]

BH ← SS:[SP+1]

SP ← SP+2

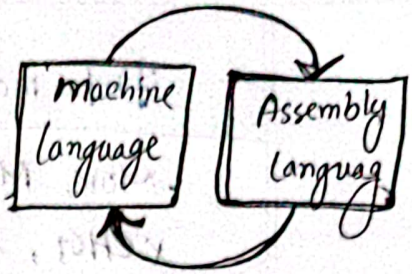


Chapter-04

Data Movement Instructions

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date :...../...../.....



4-1 MOV : Revisited

Machine Language

- The opcode
- MOD Field
- Register Assignments
- R/M Memory addressing
- Special addressing Mode
- 32 bit — Addressing modes
- An immediate instruction
- Segment ← mov instructions

*** Convension ***

- Tables

4-2 push/pop

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date : ...../...../.....

Data movement instructions:

MOV, MOVSX, MOVZX, PUSH, POP, BSWAP
XCHG, XLAT, IN, OUT, LEA, LDS, LES, LFS
LAHF, SAHF

String Instructions:

MOVS, LODS, STOS, INS, OUTS

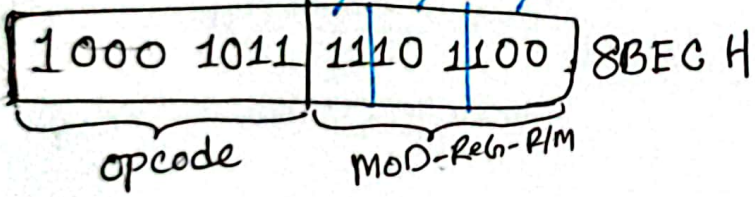
Instruction Format:

MOV Ax, Bx + [1000H]
↳ 010 - - - - Displacement

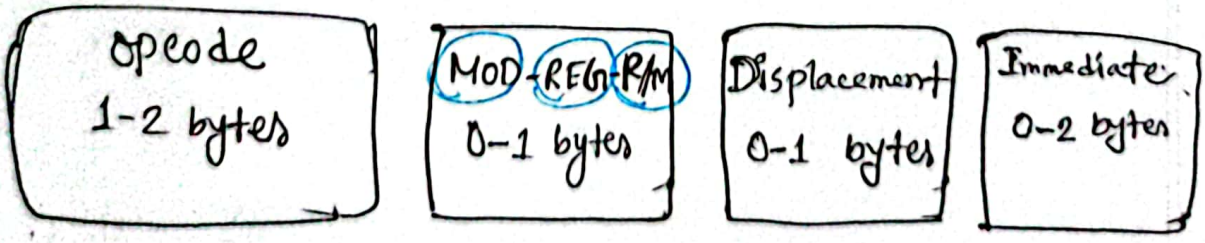
8BEC H
↳ Binary

Binary form এর instruction analysis করতে হবে।

1) 16 bit:



8 – 1000
B – 1011
E – 111 0
C – 1100



MOD
00
01
10
11

(66) 8BECH

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date : ..../......./.........

2) 32 bit:

Address size	Register size	Opcode	MOD-REG-R/M	Scaled index	Displacement	Immediate
0-1 bytes	0-1 bytes	1-2 bytes	0-1 bytes	0-1 bytes	0-4 bytes	0-4 bytes

- 32 bit start with 66 / 67

Opcode:

I) Assembly → Binary

II) Binary → Assembly

MOV AX, BX → 8BECH

opcode

16 bit

1000 1011	1110 1100
Byte 1	Byte 2

→ Assembly:

Byte-01: 1000 10 | D: 1 | W: 1

Byte-02: MOD: 11 | REG: 101 | R/M: 100

opcode : 100010 → intel manual এ দেখে এটার জন্য MOV হবে।

Operation code

Direction

D → 0 : REG → R/M

D → 1 : R/M → REG

W → 0 (move one byte of data) MOV AL, BL

W → 1
- → word (16) AX, BX, CX
- → double word (32) EAX, EBX

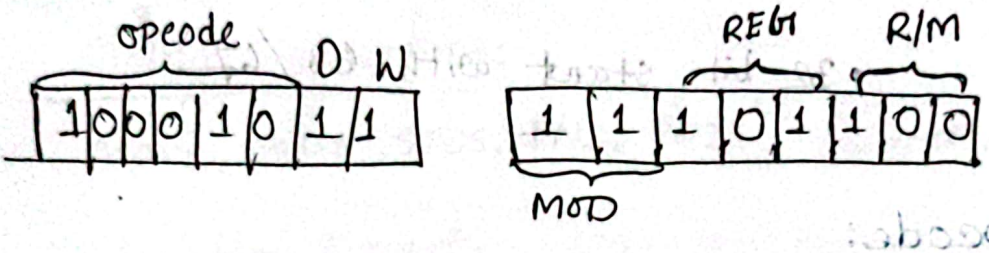
Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date :...../...../.....

MOD, REG, R/M

- Convert binary to ⟷ Assembly

8BECH



D = 1

REG → Destination

R/M → Source

move BX, SI

MOV REG, R/M

MOD	Function
0 0	NO displacement MOV AL, [DI]
0 1	8-bit sign extended displacement MOV AL, [DI+02]
1 0 0	16 bit signed displacement MOV AL, [DI+1000H]
1 1	R/M is a register

MOD → 11

So, R/M is Register

MOV BP, SP

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date : ..../..../.....

REG (when R/M = 11) R/m is a Register :

Code	W=0 (Byte)	W=1 (word)	W=1 (Double word)
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

When R/M ≠ 11

R/M code	Addressing mode	
	16 bit	32 bit
000	DS: [BX+SI]	DS: [EAX]
001	DS: [BX+DI]	DS: [ECX]
010	SS: [BP+SI]	DS: [EDX]
011	SS: [BP+DI]	DS: [EBX]
100	DS: [SI]	Scaled I. bit
101	DS: [DI]	SS: [EBP]*
110	SS: [BP]*	DS: [ESI]
111	DS: [BX]	DS: [EDI]

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date :...../...../.....

Load effective address: LEA

→ Compute the effective address of 2nd operand (Source) & stores it in 1st operand (Destination)

→ Unlike mov instruction LEA

stores Computed address → target register

→ Stores contents (data) of target register

→ LEA BX, [DI] loads the offset address specified by [DI] → BX

→ MOV BX, [DI] loads data stored at memory location [DI] → BX

→ MOV BX, OFFSET LIST perform same function as LEA BX, LIST

* → OFFset works with simple operands (LIST) & not with [DI], LIST [SI].....

→ OFFset is more efficient (1 clk cycle) than LEA (2 clk cycle)

→ MOV BX, OFFSET LIST is faster because assembler calculate offset address of LIST. Microprocessor Calculates address of LEA instruction.

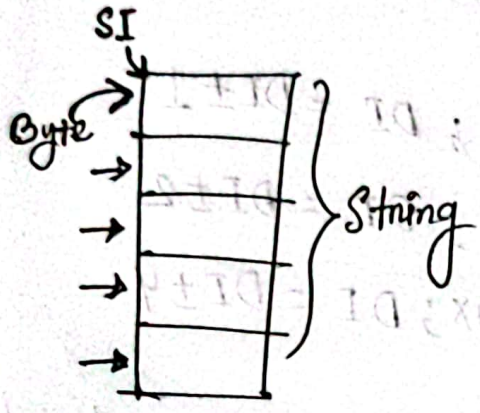
Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date :...../...../.....

String Data Transfer Instruction :

Move AX, [DI] Byte, Word, Doubleword, Quad word

↳ deals with single data



DI & SI : Destination Index, Source Index } String

LODS, STOS, MOVS, INS, OUTS. } string data

LODS :

loads accumulator (AL, AX, EAX, RAX) with the data stored at the data segment, offset address indexed by SI.

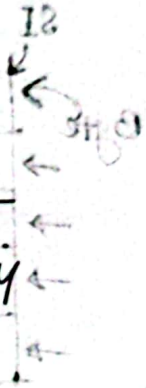
LODSB	AL = DS: [SI]; SI = SI ± 1
LODSW	AX = DS: [SI]; SI = SI ± 2
LODSD	EAX = DS: [SI]; SI = SI ± 4

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date :...../...../.....

STOS: (Store)

→ Stores accumulator (AL, AX, EAX) at the extra segment memory location (addressed DI register)



STOSB ES: [DI] = AL ; DI = DI ± 1

STOSW ES: [DI] = AX ; DI = DI ± 2

STOSD ES: [DI] = EAX ; DI = DI ± 4

MOVS: (transfers string data from one memory location to other memory location.)

~~MOVSB~~ → data segment location addressed by SI
⟹ extra " " " DI

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

MOVSW

MOVSD

Sat / Sun / Mon / Tue / Wed / Thu / Fri

Date :...../...../.....

INS : (Input string)

transfer byte, word, double word from I/O Device to Extra segment.

I/O to memory
harddisk to memory

INSB ES:[DI] = [DX]
I/O address

outs : (output string)

From DS to I/O Device

memory to hard disk

OUTS [DX] = DS:[SI] ; SI = SI ± 1