CPU

ALU	CU	FPU
Registers		BIU.
Caches		

Intel 8086 / 8088
16 bits 8 bits.

↳ 1MB

Address bits = 20 bits.

- Registers:
  - Data Register
  - Segment Register
  - Index / ~~Positive~~ Pointer Register.

For n-bit Register:

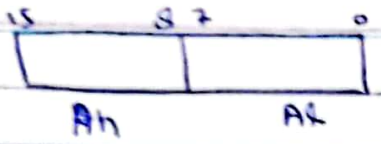
Signed: $-2^{n-1} \rightarrow 2^{n-1}-1$

Unsigned: $0 \rightarrow 2^{n}-1$

Data Registers: General purpose Register. → To store Data.

i) AX (Accumulator):
ii) BX (base):
iii) CX (count):
iv) DX (Data):

} 16 bit.



Similarly,

Bh , BL
Ch , CL
Dh , DL

- Segment Registers:

i) DS (Data)
ii) ES (Extra)
iii) SS (Stack)
iv) Code CS.

} 16 - bit

- To store 16-bit segment address.

- Index and Pointer Registers:

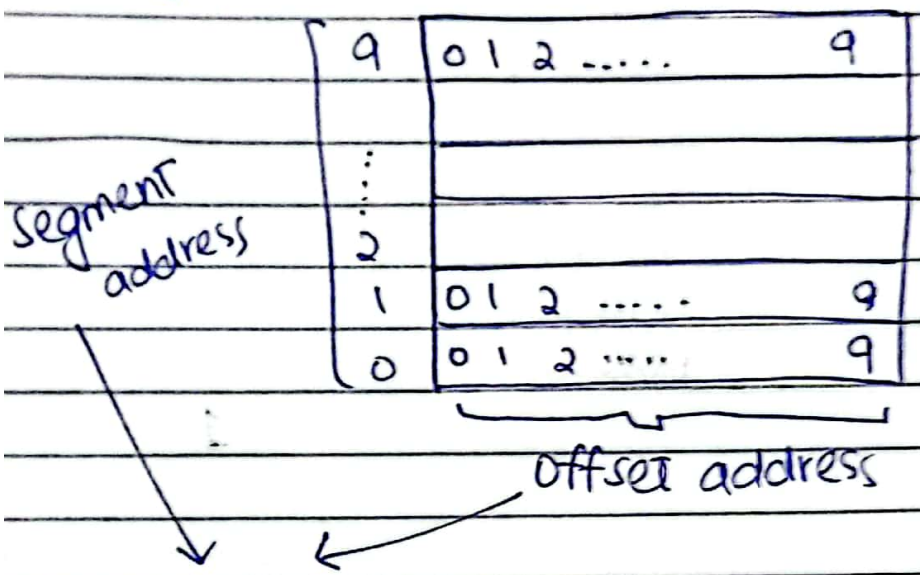
i) SI (Source)
ii) DI (Destination)
iii) IP (Instruction)
iv) BP (Base)
v) SP (Stack).

] works with DS, ES
→ works with CS.
] works with SS.

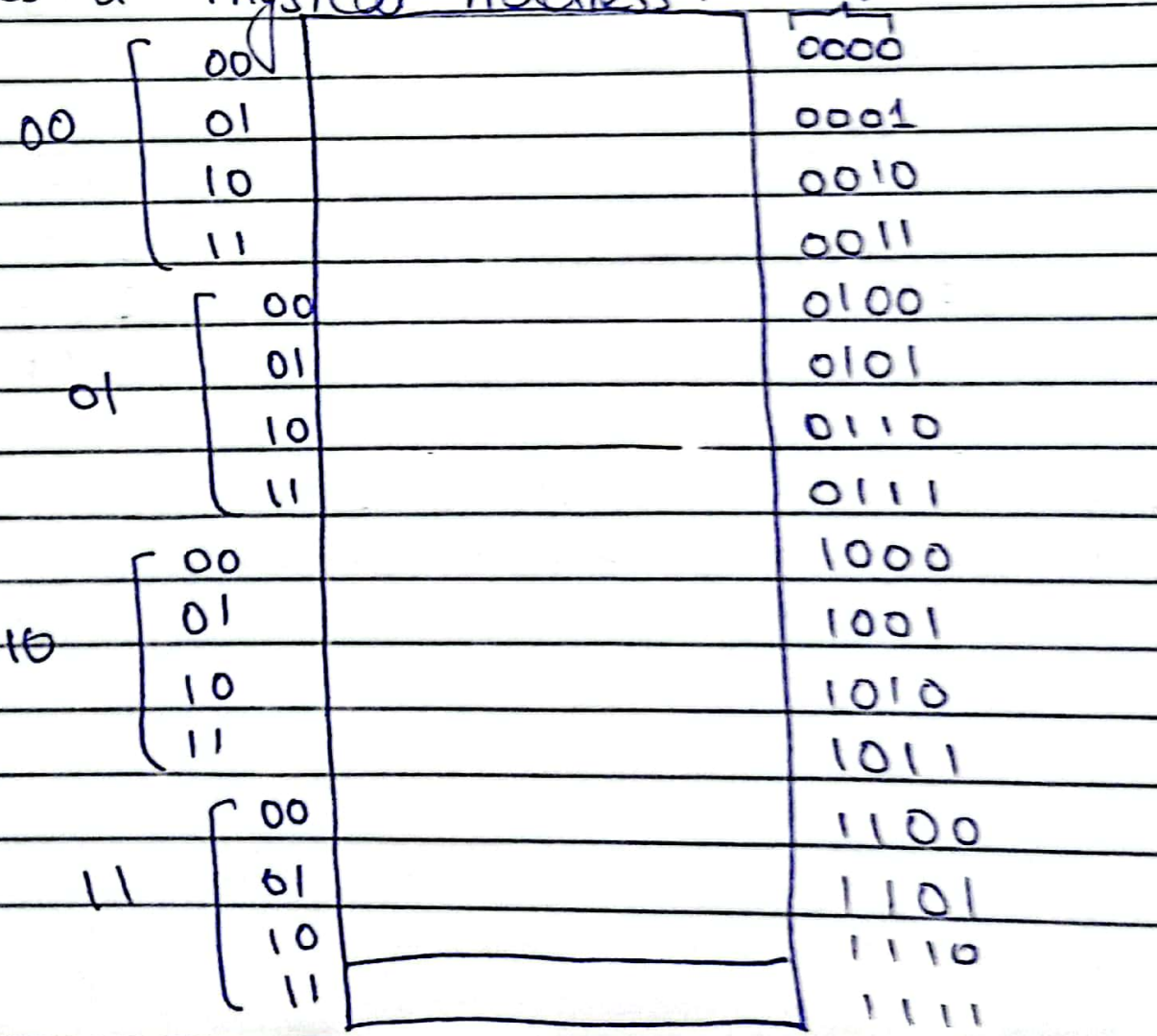
- To store 16 bit off-set address.
- works with segment Registers.

Address of Actual memory is called Physical address.

memory segmentation:



makes a Physical Address.



i) MOV

syntax:

MOV, dest, source.

e.g:

// AX 123

AL 20

Reg → Reg

Reg → Mem

mem → mem.

mem → Reg.

MOV DX [0100]

MOV [0100] AX

ii) INC CX (add 1)

// AX

iii) or DEC CX (subtract 1.)

AX

iv) ADD AX BX (Parent)

v) SUB AX, BX (AX-BX.)

byte = 8 bit.

1 word = 2 byte

1 double word = 4 byte

1 quad word = 8 byte

Flag: 1 flag = 1 bit.
1 flag register = 8 bits

i) zero flag:

0	Reset	NZ
1	set	ZR

if after arithmetic operation, the answer is zero then zero flag is set otherwise reset.

ii) Sign flag: As a signed, the answer is +ive or -ive.

- If answer is positive.
  0 → Reset . PL
- If answer is negative

Answer is negative when msb is 1

  1 → set NG

iii) carry flag:-

0 →	Reset	NC
1 →	set	CY

→ If the answer after computation when we are dealing with unsigned numbers get out of the range.
- If overflow → set
  If in range → Reset.

iv) Overflow flag :

0 → Reset NV
1 → set OV

→ Same as carry flag but we deal with signed numbers.

```
MOV   AL , 80
MOV   BL , 80
ADD   AL , BL
```

$$\begin{array}{r} 80 \\ 80 \\ \hline 100 \end{array}$$

$$\begin{array}{r} 1000\ 0000 \\ 1000\ 0000 \\ \hline 0000\ 0000 \end{array}$$

zero → set
signed → reset.

carry → set
overflow → set.

$$\overline{0\ 1\ 1\ 0\ 0\ 1\ 0\ 0}$$

Lecture : 05

i) Jump :-

→ Unconditional (JMP offset)

conditional

JZ	JNZ	JC	JNC
Jump if zero flag is set	Jump if zero flag is not set	Jump if carry flag is set	Jump if carry flag is not set.

```
XX      MOV  Cx
        .....
        .....
        .....
        DEC  Cx
        JNZ  XX
```

```
1   MOV  Ax , 4000
2   MOV  DS , Ax
3   MOV  Cx , 5
4   MOV  SI , 0100
5   MOV  DI , 0200
6   MOV  Al , [SI]
7   MOV  [DI] , Al
8   INC  SI
9   INC  DI
10  DEC  Cx
11  JNZ  6
```

So if a JA is taken after comparing -2 in the destination with 2 in the source
the jump will be taken. If however JG is used after the same comparison the
jump will not be taken as it will consider the sign and with the sign -2 is smaller than 2. The key idea is that -2 and 65534 were both stored in memory in the same form. It was the interpretation that treated it as a

Lecture: 06

XCHG:

XCHG AL, bL (swap).
AX, BX.
[AL], [SI] ✗
[SI], AL
AL, [SI]

Logic Instructions:-

- AND
- OR
- XOR

} 2 parameter just like MOV.

- Negation.
- Not.

} 1 parameter just like INC

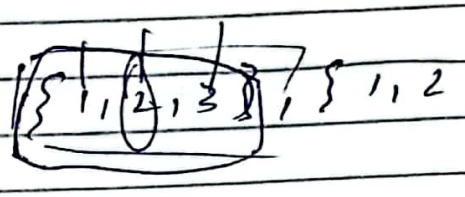
2's complement 1's complement.

→ To Reset a specific bit we AND it with 0.
→ To set a specific bit we OR it with 1.
→ To toggle a specific bit we XOR it with 1.

CMP:- Exactly like MOV.

→ It sets some flags and also reset some of them.

- JE (Jump equal)
- JNE (Jump not equal).




```
        MOV  AL , 9
        MOV  BL , 2
        JE   aa
        MOV  BL , 9
        MOV  AL , 5
        JMP  bb
aa      MOV  AL , 10
        MOV  BL , 5
bb ...
```

$sp = x[0]$

$p[-1] = 2$

(Jump greater)	JG	JA	Jump Above
greater & equal	JGE	JAE	Above & equal
Less	JL	JB	Below
Less & equal	JLE	JBE	Below & equal.
Signed comparison		Unsigned comparism.	

Input & Output on console.

- **Displaying a character:-**

main proc

```
MOV AH , 02h
INT 21h.
```

→ DL

DL = ASCII code of character that is to be printed.

```
MOV AH , 01h
INT 21h
```

} single character read

input is stored in AL (ASCII code).

Displaying a string:-

```
MOV AH , 09h
INT 21h.
```

} → DX میں جس کا offset ہو

- data

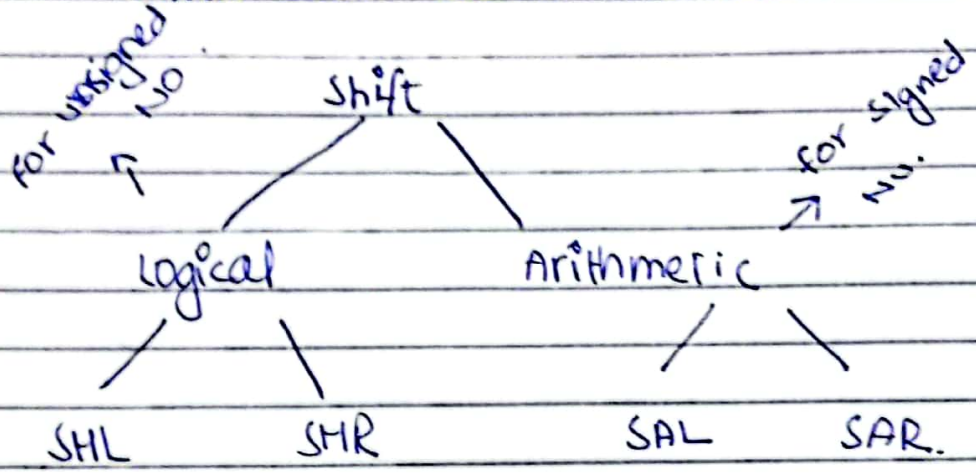
```
string1 db "Hello world$"
```

```
main proc
        mov  ax , @data
        mov  ds , ax

        mov  dx , offset string1
        mov  ah , 09h
        int  21h.

mov ah , 4ch
int 21h

main endp
end main.
```

To output a line, we write

e.g: string1 db "Hello World $"
string2 db 10, 13, "BS CS $"
or 0ah, 0bh, ...

• Shift and rotate instruction:



SAL works as SHL.

add .386 on start to run this

SHL AL, 2 ✗
for more than 1.
MOV BL, 2
SHL AL, BL ✓

SHL AL, 1

C.F | 01000010

01000100

• ROTATE INSTRUCTIONS:-

without carry

ROL

ROL AR, 1

i) AR = 10101001

01010011

Carry flag.

ROR

ROR AR, 1

i) AR = 10101001

11010100

Carry flag.

with carry.

RCL

RCL AR, 1

i) AL = 01000101

10001010

0

C.F

RCR

same

```
    MOV CL , 5
2:  MOV SI , offset arr1 .
    MOV DL , [SI].

    MOV CH , 8
again: SHL DL , 1 .
    JC aa
    MOV DL , '0'
    MOV AH , 2
    INT 21H
    JMP bb
aa:  MOV DL , '1'
    MOV AH , 21
    INT 21H.
bb:  DEC CH
    JNZ again.
    DEC CH
    JNZ cc.
```

```
INC SI
MOV DL , 10
MOV AH , 2
INT 21H

MOV DL , 13
MOV AH , 2
INT 21H.
```

4 0

0 100 0000

~~0100 0000~~

0

```
b'  MOV CL , 5
bb: MOV SI , offset arr1
    MOV bx , [SI].

    MOV CH , 4
again ROL BX , 4

    MOV DL , BL
    ADD DL , 0Fh.
    CMP DL , 9.
    JBE aa:

    ADD DI , 7.

aa: ADD DL , 30h
    MOV AH , 2
    INT 21h
    ~~DEC CH~~ → JNZ again
    ADD , SI , 2
    DEC CL.
    JNZ bb.
```

6A · A.

48 9A.

0100 1000 1001 1010.

1000 1001 1010 0100

bl

0000 1111

0 F

0 d00

9 + 30h / 48

39h

9

ABCDEF

10+55 65 7

11+55

12+55 7

7

4d