3191383 CSE-C2-313 (AB II)

Date : 15/03/2023 Page No. :

COA

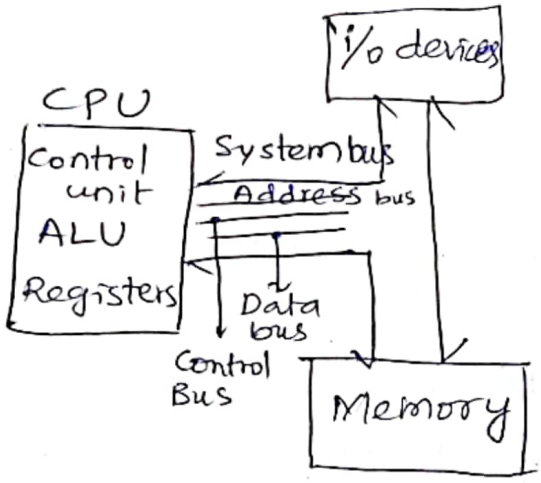
computer organization & Architecture

→ SISD - Single instruction Single data

→ SIMD - single instruction multiple data

→ MISD - Multiple instruction single data

→ MIMD - Multiple instruction Multiple data



Address bus - MAR
Memory address register

Data bus - MBR (or) MDR
Memory Buffer register

Architecture Vs Organisation

Block diagram — circuit

planning

Date :..... Page No. :.....

- Instruction: ADD R_1 R_2

single line execution $R_1 = R_1 + R_2$

Program is set of instructions.

- Data Transfer ~~register~~ Instruction:

MOV R_1 R_2

operation (opcode) — Destination — Source — operands

$R_1 \leftarrow R_2$

Memory upperand

- Register

1 nibble = 4 bits

1 byte = 8 bits hyroriche

(ns) Cache (fastest, KB to MB)

(ms) primary memory (MB to GB)

secondary memory (GB to TB)

external memory

1 KB = 1024 Bytes

1 MB = 1024 KB

1 GB = 1024 MB

1 TB = 1024 GB

1 PB = 1024 TB

Instruction:

HLT: stop the program instruction is HLT

- Zero operand, only operation.

Ex: Ret (Return)

one operand

| opcode | operand |

operation

Ex: | Inc | a | a=a+1

Inc → increment

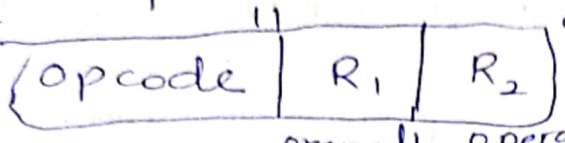
| Dec | a | a=a-1

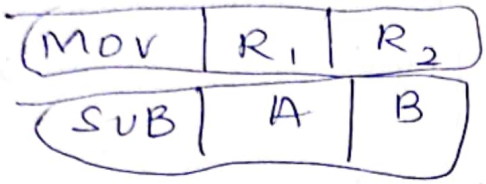
Dec → Decrement.

8085, 8086 Processor Architecture

- Two operand Instruction

is





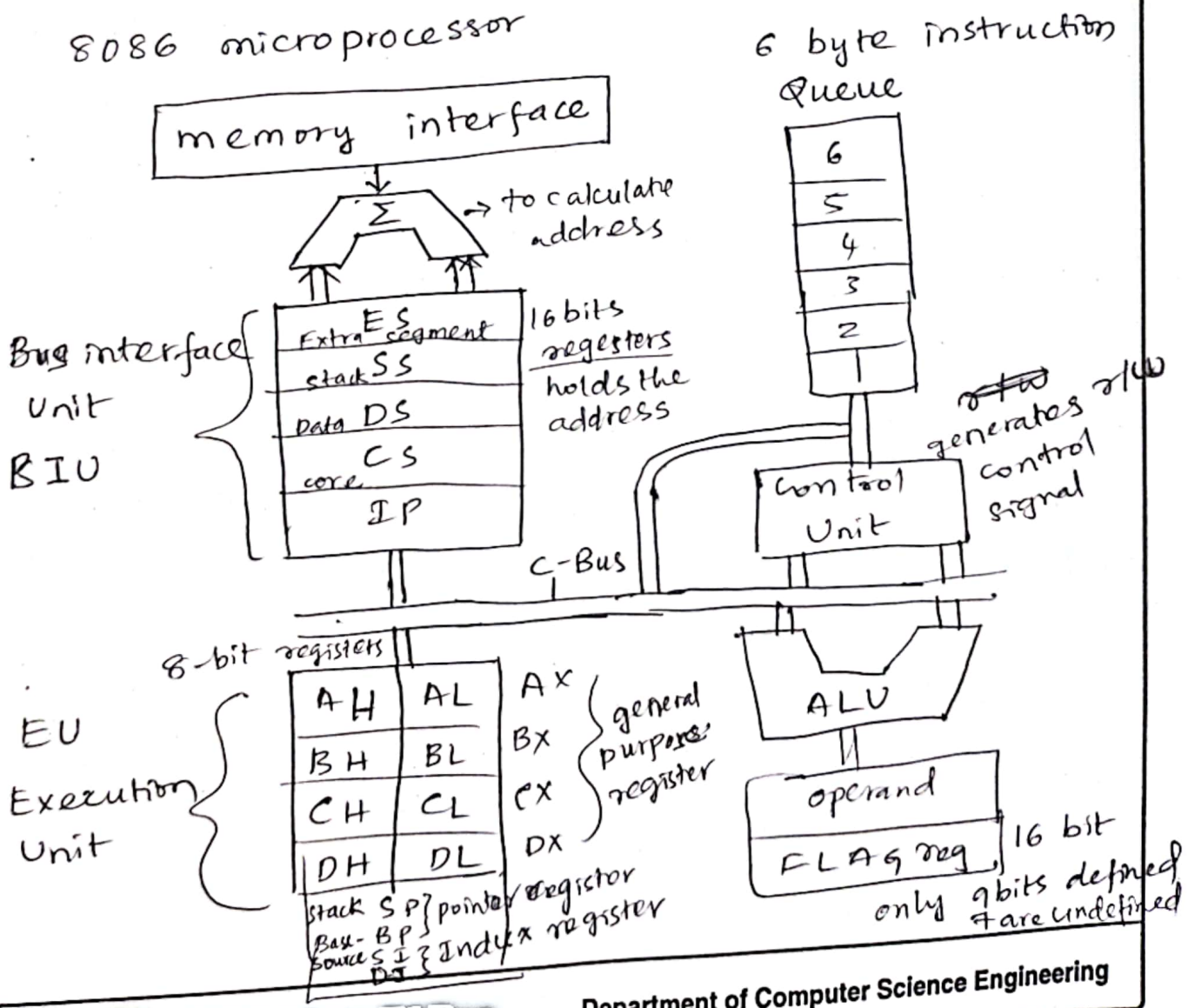
$A = A - B$

$A = A + (\bar{B} + 1)$

2's complement

→ Difference b/w compiler & Interpreter

8086 microprocessor



Date :..... Page No. :.....

PA = Segment register * 10H + offset

Physical address

IP value (Instruction pointer)

FLAG Register 16 bits

Address Bus = 20 bits

Data Bus size = 16 bits

Conditional flag	control flag
Sign SF	Trap flag
carry CF	Directional flag
auxillary AF Binary→BCD	Interupt flag
overflow OF Exor	
parity PF	
zero ZF	

Indicates

Date :..... Page No. :.....

(4 marks)

MEMORY INTERFACE: 8086 Microprocessor - 1MB

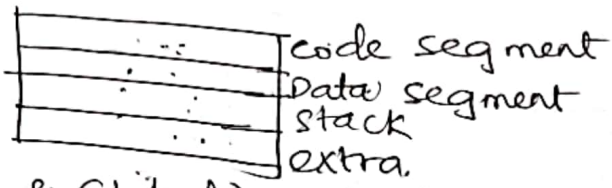
Date :..... Page No. :.....

BIU : BUS Interface Unit

Σ - Crossing unit is used to calculate address

→ registers will hold the memory location

→ program can be stored in 4 memory blocks



→ D.S (local & Global) variables will be stored

→ Stack:

Functions address, operation part, data and parameters.

→ Extra:

Result variables are stored in extra

→ Code segment:

Remaining part is stored in this segment.

Ex: library functions.

→ All registers are 16 Bit (2 Byte) capacity.

→ Registers ~~are~~ will hold their respective starting address.

→ Next instruction address is stored in IP

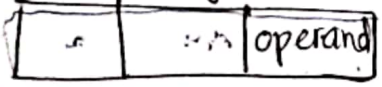
→ Instruction Queue is 6 Byte

→ This is also called as Buffer Queue.

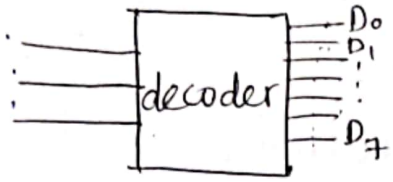
→ The loading of data from the Buffer Queue to IR. Register is called as Fetch operation.

→ After this instruction is decoder.

IR register



$2^3 = 8$ possible operations.



→ only one line is active that will indicate one operation.

Control Unit: It generates the control signal to load the data from general purpose registers to ALU.

→ After the operations the result will be stored in AH (Acumulator register)

H → Higher byte
L → lower byte

→ For Addition and substraction the result is stored in AX

→ For multiplication: result is stored in AX and BX where the product is more than 16 Bit

→ For division: quotient is stored in AX and remainder will be stored in DX (16 bit number)

→ For division: quetient is Stored in AL and remainder will store in AH. (8 bit numbers)

→ In Ubuntu we use

EAX
EBX
ECX
ECD
} Extended base registers.

capacity is 32 Bit

Date :..... Page No. :.....

→ BX is Base register, we can store base address.

Ex: Count register

→ Used for loop instructions

```
        mov   CX, 7
label1: mov   AX  7
        mov   BX  1
        add   AX , BX
        loop  label1
```

→ DX : Data register.

To store the temporary data

→ Flag: it gives the status of ALU

→ SP: It points to stack memory location

→ Stack pointer.

→ During push and pop sp is incremented/decremented.

→ BP: It is also pointing stack memory for accessing parameters.

→ SI : Source Index, it will point to data segment memory location.

→ DI: Data index storing result they point to extra register.

→ FLAG: It is a register with maximum capacity 16 bits, But only 9-bits are defined.

the defined 9-bits are

S – 12
C – 11
Z – 10
O – 9
A – 8
D – 7
} Conditional flag bits.

T – 6
D – 5
I – 4
} Control flags

SF: Sign Flag, It indicates the negative value if result is negative it is 1.

SF = 0 (positive)

$\Rightarrow 7 = \begin{array}{cccc} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ + & 1 & 1 & 1 \\ \hline 1 & 0 & 0 & 0 \end{array}$ $\Rightarrow 7 = \begin{array}{cccc} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ & & & 1 \\ \hline 1 & 0 & 0 & 1 \end{array}$ (2's complement of 7)

$\begin{array}{cccc} 0 & 0 & 1 & 0 \\ \hline 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 1 & 0 & 1 \end{array}$ (+2) (+5)

→ Carry flag:
If carry is generated it is set to 1.

→ zero flag: It indicates if result is '0' then it sets to one.

Ex: $\begin{array}{cccc} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ \hline 0 & 0 & 0 & 0 \end{array}$ — +5, — -5

→ O - overflow : Input carry (XOR) output carry

0 → no overflow

1 ⊕ 1 = 0

→ A (Auxiliary Flag): From lower label to higher label if carry is generated then auxiliary flag will set to 1.

Ex:

higher label lower label

```
  0  0  0  1    1  1  0  0
  1  0  0  0    1  0  0  1
        1  1
-------------------------
  1  0  0  0    0  1  0  1
-------------------------
```

→ (parity ~~generator~~ Flag) P : Even number bit.

→ T (Trap flag) : How many processing unit is activated

If only 1 is there single step mode

In multi step mode it is zero.

→ D (Direction Flag): How process is accessing data!

→ If data is accessed from low byte to higher byte then it is set to 1.

→ Little Indian:

2nd	1st
H.B	L.B

* If 1st Byte stored in Lower byte then it is little Indian, otherwise big Indian

Date : 23/03/2023 Page No. :

→ 8086 is 16 bit micro processor. (10 marks)
→ Address Bus capacity is 20 bit.

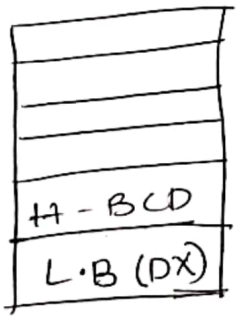
* Data Transfer Instructions:

. mov D.S. (D - destination, S - source)
→ It can be register to register
→ It can be memory to register
→ It can be register to memory

mov AX, [2014H] → memory location address.

mov AX, 5;
→ value 5 is loaded in AX
→ D.S are operand fields.
→ mov AX, BX

2. push DX
DX value is stored into stack memory.
DX is 2byte.



3. Pop DX;
Data can be removed from stack.
→ 2 bytes - 1 word

4. IN (Input)
IN AX, (I/P address)
from i/p device the data is stored in AX

Date :..... Page No. :.....

4. OUT

Out (AX), BL

OUT (I/P), Reg.

from register, the data is transformed to input

5. LEA (Load Effective Address)

One of the memory address is loaded into processor.

LED AX, [BX] [P] → address of memory

↳ Register

6. LDS (load data segment):

Data will be load in the register.

→ only one word is loaded from memory.

(LDS Register, memory).

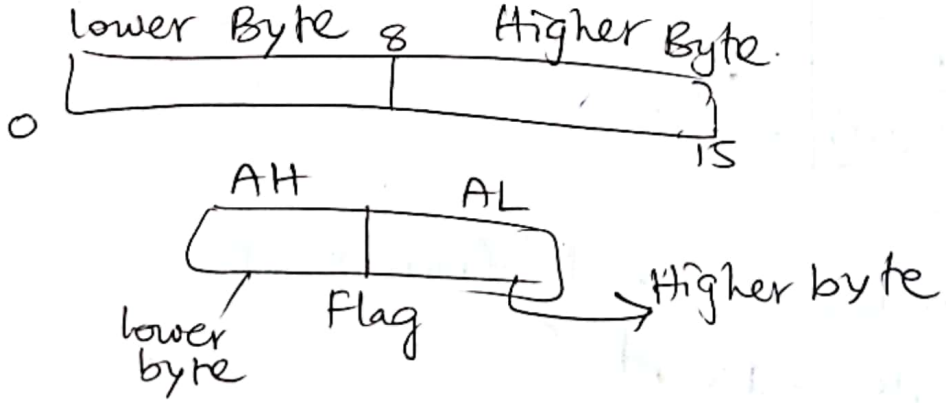
7. LES (load extra segment).

Data will be load in register from extra segment

→ 1 word is loaded from memory

8. LAHF

accumulator register



Date :..... Page No. :.....

→ In AH higher byte of acamulator register load flag, register lower byte

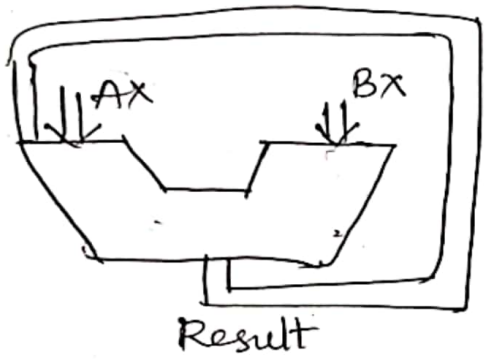
9. SAHF (stores the AH value in flag register lower byte)

2. Arithmetic Instruction:

1. ADD AX, BX.

→ A, B are registers.

AX = AX + BX



2. ADC (add with carry)

ADC AX, BX

AX = AX + BX + Carry

3. DAC
DAA
AAD
AAA
AAS
AAM
AAD

4. INC (Increment)

INC AX

AX = AX + 1

program:

```
mov AX, 10
mov BX, 20
ADD AX, BX
```

→ Subtraction Instructions

SUB AX, BX.

```
mov AX, 100
mov BX, 25
SUB AX, BX.
```

→ SBB (Subtraction with Barrow)

SBB AX, BX

AX = AX − BX − Barrow.

→ DEC (decrement)

DEC AX

AX = AX − 1

→ cmp (Comparision)

cmp AX, BX

AX = 10

BX = 10

AX = AX − BX.

zero flag = 1

→ means both are equal.

→ multiplication operation

```
mov AX, 15
mov BX, 0
```

MUL AX = AX = AX * AX

MUL BX = AX = AX * BX

Unsigned number multiplication → MUL

signed number multiplication → IMUL

→ Division

```
mov AX, 7
mov BX, 5
Div BX   // AX = AX / BX → Quotient
         DX = Remainder
```

If we use only 8-bit register

```
mov AL, 5
mov BL, 2
Div BL   // AL = AL / BL → Quotient
         AH = Remainder
```

* iDiv → for signed number.

→ shift instructions.

```
mov AL, 21H
```

i) SHL (shift left) ✓

```
SHL AL, 1 → no. of times performed.
```

(carry flag)

CF	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	i/p
0	0	0	1	0	0	0	0	1 ←	0
	0	1	0	0	0	0	1	0 →	after left shift

D_7 is shifter to carry flag.

ii) SHR (shift Right)

D_0 is shifted to carry flag.

i/p	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	CF
0	0	0	1	0	0	0	0	1 →	1
	0	0	0	1	0	0	0	0 —	1 → (After shift)

Date :..... Page No. :.....

iii) Arithmetic right shift operation (Instruction)

```
mov AL, 21H
SAR AL, 1
```

D_7

0 0 1 0 0 0 0 1 : CF

D_7 0 0 0 1 0 0 0 0 1

→ D_7 is same

→ From D_6 we have to do shifting

→ SAL AL, 1

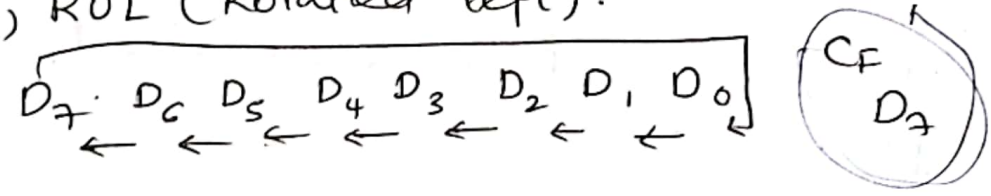
It is same as SHL

C_F

0 0 1 0 0 0 0 1 0

⇒ Rotated Instructions

i) ROL (Rotated left):




```
ROL AL, 1
```

0 0 1 0 0 0 0 1 C_F

0 1 0 0 0 0 1 0 0

ii) ROR (Rotate Right):

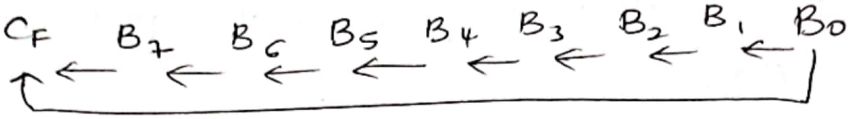
```
ROL AL, 1
```

0 0 1 0 0 0 0 1 C_F

1 0 0 1 0 0 0 0 1

Date : 24/03/2023 Page No. :

→RCL (Rotate left through Carry)

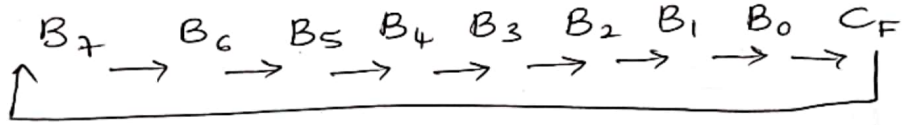


Code for RCL:

```
mov AL, 32H
RCL AL, 2
```

CF		
	0011 0010	
0	0110 0100	1st shift
0	1100 1000	2nd shift

RCR – Rotate Right through Carry




```
mov AL, 32H
RCR AL, 1
```

0011	0010	CF
0001	1001	0

Logical Instructions:

```
AND AL, BL
OR  AX, BX
XOR AL, BL
NOT → 1's compliment
```

X-OR

0	1	1
1	0	1
0	0	0
1	1	0

Date :..... Page No. :.....

Branch instructions (or) Jump instructions:

jump → je instruction → same as go to

```
mov AX, 5
mov BX, 5
cmp AX, BX
je label:
  printf "not equal".
label:
  printf "equal"
```

→ if condition is not true.

```
jne label:
   ≡
label:
   ≡
```

HLT → jne also continuously executed label if we didn't write instruction HLT.

program for ever or odd:

```
start: mov AX, 7
       mov BX, 2
       Div BX:      AX = AX / BX = 3,
                    DX = 1
Comp DX, 00
   je label:
   printf "odd number".
```


Continue next Page:

label : printf "even number".

HLT:
ret;

ja → jump above
jae → " " or equal
jb → " below
jbe → " " or equal
jc → " if carry
jnc → " if no carry.

mov ax, 5
mov bx, 3
cmp (ax, bx)
ja label

label:
printf("...");

HLT;

jae jbe
<= 1
temp = ax, bx
je

Control instructions:

HLT
NOP
WAIT

→ no operand field, it will be zero.
→ STC → set carry Flag
→ CLC → clear Carry Flag
→ CLI → clear interrupt Flag
→ CLD → clear Directional Flag.

al, cl

AX

Date : 27/3/2023 Page No. :

AAA (ASCII Adjust After Addition)

If AL > 9 (or) AF = 1
(lower nibble)

AL = AL + 6
AH = AH + 1
AF = 1
CF = 1.

else

AF = 0
CF = 0

```
mov AL, 38H
mov BL, 35H
ADD AL, BL
AAA
HLT (Halt-To stop the execution)
```

38H = 0 0 1 1 1 0 0 0
35H = 0 0 1 1 0 1 0 1

0 1 1 0 1 1 0 1 → stored in AL
0 1 1 0 → > 9 AL = AL + 6
clear AH
AH = AH + 1

0 1 1 1 0 0 1 1
AL Higher nibble clear 0 0 0 0 0 0 1 1

AH = 01 AL = 03

AX = 0103
AH AL

```
mov  AL , 35H
mov  BL , 52H
ADD  AL , BL
AAA
```

35H = 0 0 1 1 0 1 0 1
52H = 0 1 0 1 0 0 1 0 , AF = 0, CF = 0

1 0 0 0 0 1 1 1

0 0 0 0 0 1 1 1

AX = 0007

```
mov  AL , 19H
mov  BL , 29H
ADD  AL , BL
```

19H = 0 0 0 1 1 0 0 1
29H = 0 0 1 0 1 0 0 1 (∵ AF = 1, CF = 1) AH = AH + 1

0 1 0 0 0 0 1 0

0 0 0 0 1 0 0 0

AH = AH + 1 = 01

AX = 0108.

AAS (ASCII Adjust After Subtraction)

If AL > 9 or AF = 1
AL = AL - 6
AH = AH - 1
AF = 1
CF = 1

else AF = 0
CF = 0

```
mov  AL , 8CH
SUB  AL , 51H
AAS
```

Date :..... Page No. :.....

- AAD (ASCII Adjust before divison)

AL = (AH * 10) + AL | AL = (01 * 10) + 05 = 15

AH = 0 | AH = 00

mov AX, 0105 | AX = 0015 = 0F

AH AL

- AAM (ASCII Adjust after Multiplication)

AH = AL / 10 = Quotient

AL = Remainder = AL % 10

Ex: If AH = 00 AL = 15

then AAM,

AH = 15 / 10 = 01

AL = 15 % 10 = 05

AX = 01 05.

MUL AX, 0015

AAS (ASCII Adjust After subtraction)

	ASCII	16 / 3 remider 0
0	48	30
1	49	31
2	50	32
3	51	33
4	52	34
5	53	35
6	54	36
7	55	37
8	56	38
9	57	39

Hexa decimal values.

~~mov~~ if lower nibble of AL > 9 (or) AF = 1

AL = AL - 6

AH = AH - 1

AF = 1

CF = 1

else

AF = 0

CF = 0

```
mov   AL, 33H
mov   BL, 37H
SUB   AL, BL
AAS
HLT
```

```
  0 0 1 1     0 1 1 1
- 0 0 1 1     0 1 1 1
---------------------
  1 1 1 1     1 0 0 0
            -       6
---------------------
  1 1 1 1     0 0 1 0
---------------------
  0 0 0 0     0 0 1 0   < 9
                2   4
```

AH = AH - 1

= 00H - 1 = 0000 0000 - 1

= 1111 1111

= FF

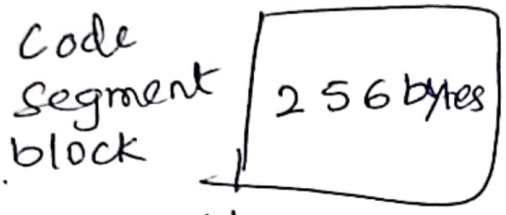
AX = FF 04 = -4

represent -ve

Date :..... Page No. :.....

8085 microprocessor (8-bit)

Data bus capacity



Address Bus (16-bits)

Data bus (8-bit)

→ It will not suppor pipeline architecture (parallel operations)

→ It support SISD, instructions executed in serial manner.

8086 - 16 bit microprocessor

→ Data bus capacity 16-bits

→ All register 16 bits register

→ Address bus capacity 20-bits

→ Support pipeline architecture

AH AL AX
BH BL BX

Date : 3/04/2023 Page No. :.....

Addressing modes (1 mark (or) 2 marks)

Based on addressing mode the data is

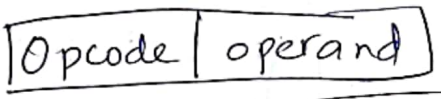
Implicit Addressing mode : Within the instruction field opcode part is the operation part.

Example: INC A

PUSH
POP

$AX = AX + 1$

Immediate Addressing mode : Inside the instruction



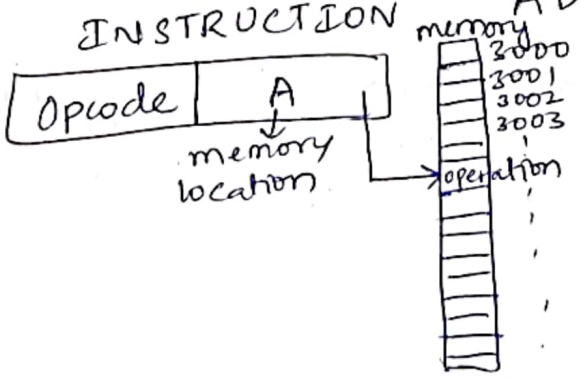
Ex: mov AX, 10 — data value — instruction

(mov: operation; AX: Address to store data value)

mov AL, 5

ADD AX, 10 AX = AX+10

Direct addressing mode:



ADD [A]

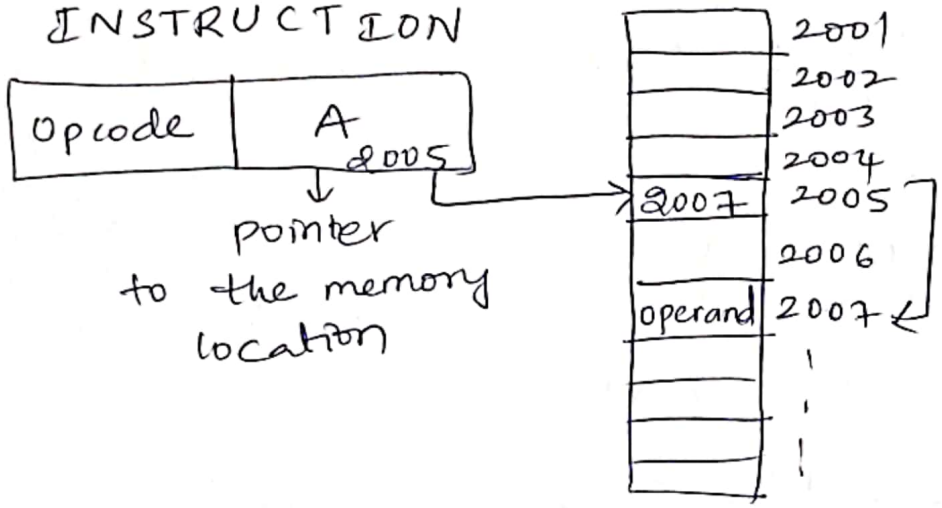
↓

memory loc address

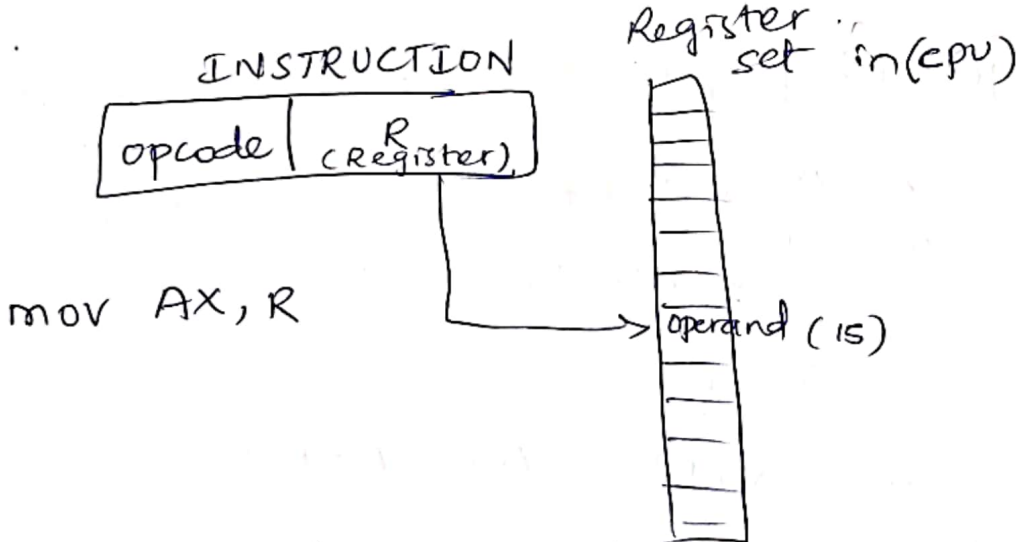
Date :..... Page No. :.....

- Access the value through the memory location address.

Indirecti addressing mode



Register direct Addressing mode:

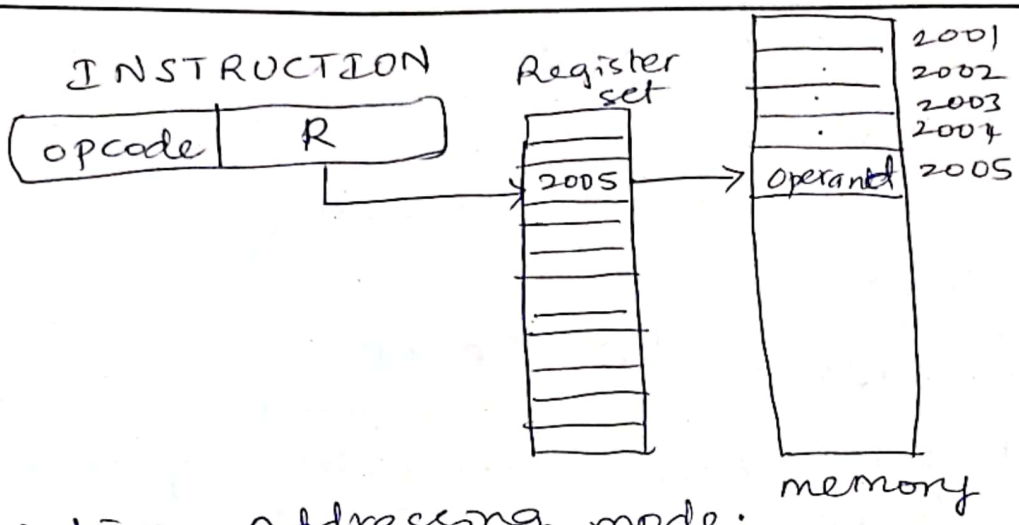


- Access the data from the register.

Register indirect Addressing mode:

- Access the data from memory through register set

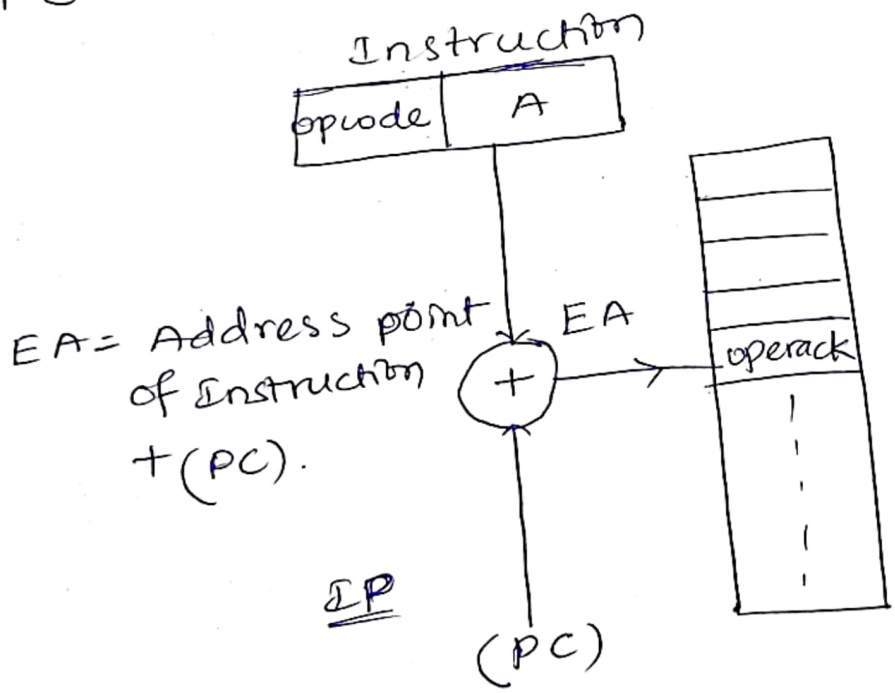
Date :..... Page No. :.....



Relative Addressing mode:

EA : Effective address.

PC : hold the next instruction address.



Date :..... Page No. :.....

Base Index Addressing

EA = Base address + Index value

1000H + 20H = 1020H (actual memory loc address)

. mov R_1, R_2

↓ Base Index

$R_1 \leftarrow (R_1 + R_2)$

Base index

Auto Increment / Decrement addressing mode:

post pre

ADD R_1, $(R_2)+$

ADD $(R_1)-$, R_2