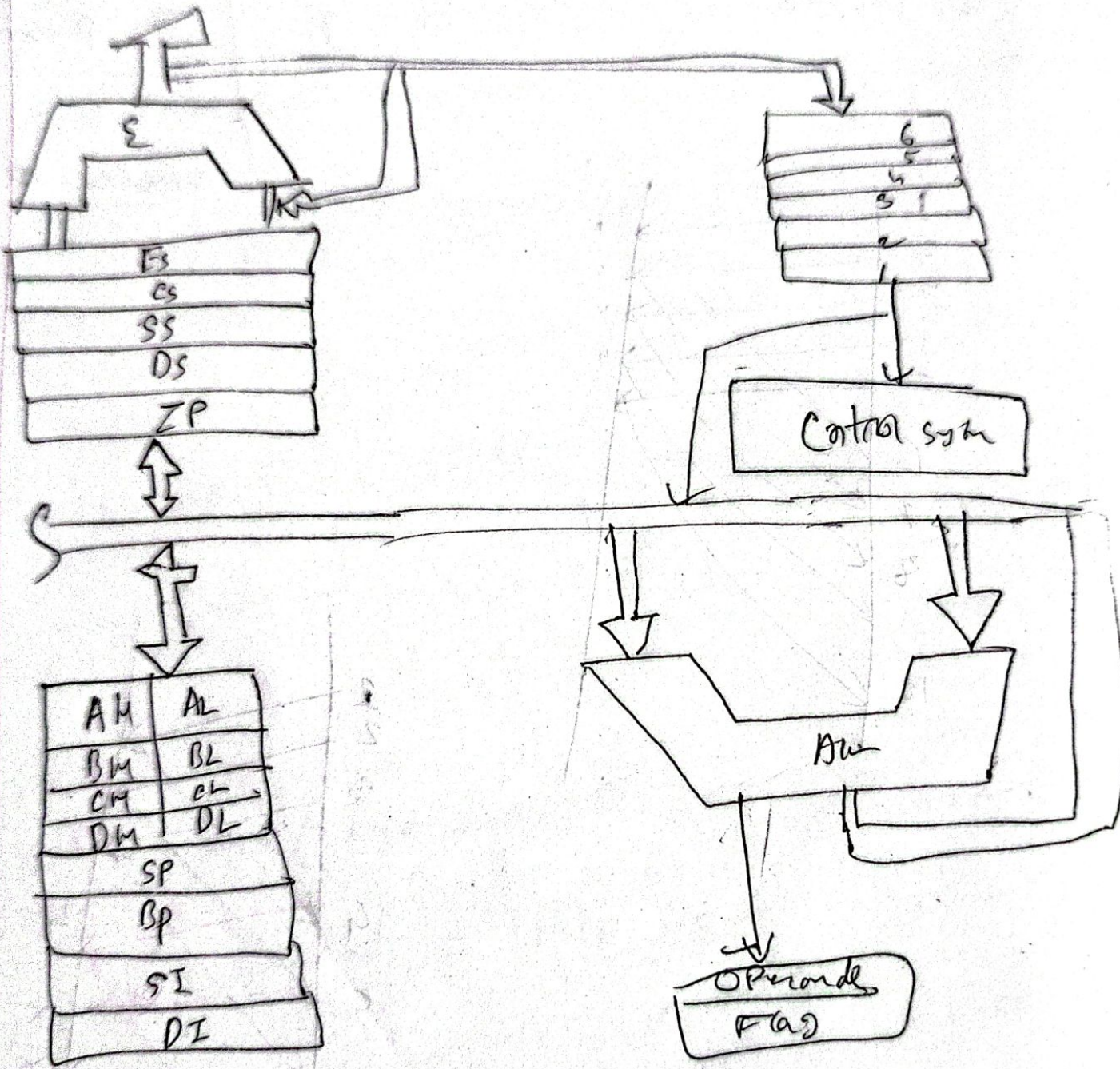
Memory

a)



BIU

segment register
Comp seg address

EU

BZU: Bus interface Unit handles the transaction of Data between and address of EU

Operations:

(I) Instruction fetching
(II) Calculating physical address
(III) Read & Write memory.
(IV) transfer instruction to 6 byte instruction qu

Elements of BZU:

(I) instruction queue.
(II) Address Adder.
(III) segment register
  (1) Extra segment (ES)
  (2) Code segment (CS)
  (3) Stack segment (SS)
  (4) Data segment (DS)
(IV) IP (instruction Poi

EU: known as Execution Unit. it executes the fetched instruction from BIU's instruction queue.

Elements:

(I) ~~Control Unit~~ instruction decoder : decodes the instruction fr qu

(II) Control circuit : executes operation in EU by issuing control signal,

(III) ALU

(IV) Pointer and index/offset register, General Pu

(1) Accumulator register
(2) Base register
(3) Counter "
(4) Data "
(5) Source index
(6) Stack pointer
(7) Base pointer.
(8) Destination Inde

(V) Flag register.

Operation:

(I) Decode instruction from BIU

(II) Executes instruction.

(III) Generate control signal.

24

Pipelining: A technique where the Processor Fetches the next instruction while the current one being executed.

Benifits:

(I) As both are hapenning simultaniously it reduces idle time.

(II) ~~Exter~~ BIU fetches and store them into queue

(III) EU executes instruction.

DrawBacks

① As pipeline technique queue the Data when Branching / Jump comes it fails. It create Complexity. - queue must be dumped and reload Starting.

(3)

The mp acts as Brain of computer because It Controls and Perform Computation, Process Data and Manage tasks. The 8086 is Part of x86 Family, a widely used Processor architecture

(4)

The area of square.

```
.Mode small
- Stack 100H
.Data
   SIDE DB 5
   Area DB ?

.Code
   . main PROC.
   mov AX, @Data
   mov DS, AX

   .mov AX, side.
   MUL .SIDE          | square = a²
   .
   MOV Area, AX

ent: mov ah, 4ch
     int 21h   main endP
                        end main
```

(5)

AX is ~~known as~~ used for Arithmatical operation like multiply or dividing, add, sub → whole number, (called words, 16 bit)

If multiply 2 number one number will be stored in Ax register.

AL → Used for operation involve small numer less than 8 bit

AH → Used for large numeber

Dx: Dx is used when a result is bigger than a single register can hold, (more than 16 bit)

If a multiplication / div result is too big it'll go to Ax but if Ax overflows it'll go to Dx

Dx also holds a part of address.

Dh → higher data than 8 bit

DL → 8 bit or lower data

Div: Divided in Dx, Ax we use Dx for Remainder.

Flag affected:

Cf, Of, ZF, SF

Process of dividing memory into logical block is called segment

(9)

(a) **Segmentation:** is a Process of dividing total m into smaller logical segments to simplify memory management & improve efficiency. Each segm has a base address and an offset

Benifits:

(i) increases modularity & ease of Programming
(ii) facilates multitasking by isolating Proces
(iii) enable efficient use of memory by overlapp segments (faster access & execution)

Example

Total memory = 1mb = 1024 KB

Segment size = 64 KB

$$\text{Number of segments} = \frac{\text{total memory}}{\text{segment size}}$$

$$= \frac{1024}{64} = 16$$

Total 16 seg possible each size 64 KB

(B)

20 bit address bus, 8 bit memory bus,

8086 has a 16-bit data bus ($D_0 - D_{15}$) but accesses memory organized as 8-bit words. ($D_0 - D_7$) & To handle this and address bus 20 bit

8086 use 2 memory banks. Each Bank size 512KB

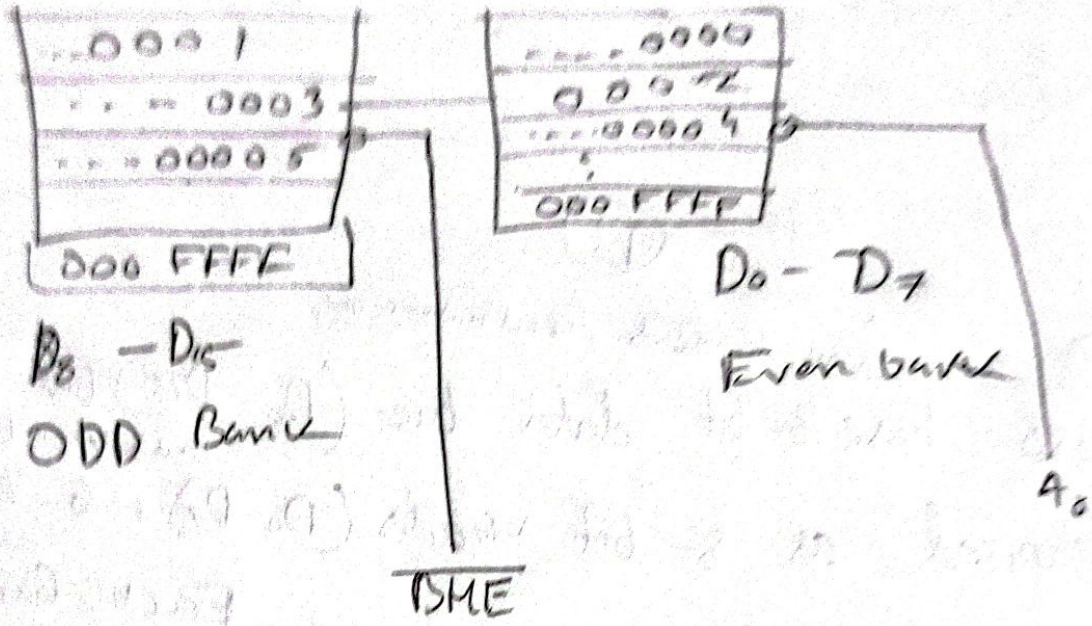
(I) Even Addresses: The lower 8 bit ($D_0 - D_7$) are accessed through the even memory bank

(II) ODD Addresses: the upper 8 bit ($D_8 - D_{15}$) accessed through ODD bank

∞ BHE (Bus high enable) activates odd memory bank

∞ A_0 (Address Line 0) identifies the specific byte (low or high)

when both are 0 it can be accessed,



The data must be aligned to accept, misaligned address can't be accessed at once, because

BHE	A0	
0	0	→ Both active
0	1	→ ODD active
1	0	→ even active
1	1	→ no operation

Advantages of memory bank concepts

① Enables simultaneous access to 2 bytes, improving speed.

② Fully utilizes the 16-bit data bus

(6)

Given,

$AL = 95H$

$= 1001\ 0101$

$BL = 0FH$

$= 0000\ 1111$

① Inr AL

$1001\ 0101$

$+1$

$1001\ 0110$

$AF = OF = 0$

$CF = 0$

$SF = 1$

$ZF = 0$

$PF = 1$

(iv) Neg AL

AL = 1001 0110

Not AL = 0110 1001 → 1's comp

+1

2's Com = 0110 1010

CF = 0, ZF = 0, PF = 1

AF = OF = 0, SF = 0

(iii) XCHNG AL, BL → No flag affected

(v) ADD AL, BL

AL = ~~0110~~ 0000 1111

BL = 0110 1010

AL = 0110 1111

swaped in

(vi) AND AL, 01H

AL = 0110 1111

0000 0001

0000 0001

All flag = 0

(10)

(i) MOV AL, BX | AL is BX is DS segment

Given,

DS = 95H

P

As Both AL and BX is register so the Addressing mod is register addressing mod. PA is not Possible.

(ii) ADD AL, [BX]

It a ~~reg~~ Register indirect Addressing mode.

PA = (Seg + 10h) + offset

= (95H x 10h) + 0098H

= 9E8

Jump is from Code segment

(iii) JMP L

Relative Addressing mode is used, because Jump Directly to L.

given,
L = 9H
CS = 2344H

PA = (2344H × 10h) + 9
= 23449h.

(iv) NOP

is From Implied Addressing mode,
no physical address,

(v) MOV AL, A[BX][SI]

It's a 2D array or Based-Indexed Addressing mode

PA = 05H × 10h + (0098H + 0066H +)
=

(iv) MOV AX, A[BX][SI]

SAR SHR	SAR
(i) Unsigned num	Signed num
(ii) Msb Preserved	(ii) Msb = 0
(iii) Automatic logical	logical Shift Arithmetic
75	-53

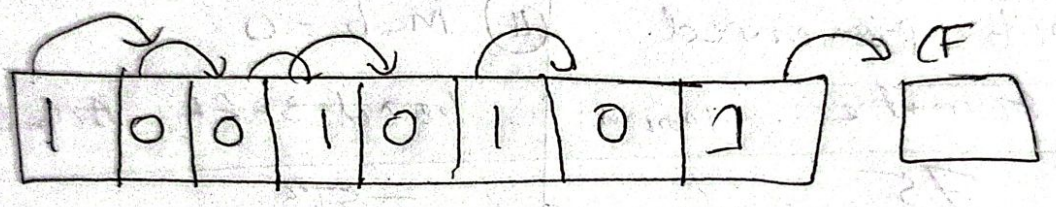
(11)

In 8086 Processor SHL and SAL Instruction Perform Same operation

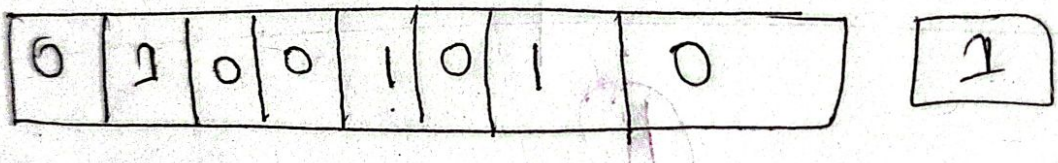
	SHL	SAL
(i)	Shift Left	Shift Automatic Left
(ii)	Used for Unsigned number	Signed number
(iii)		

(i) SAR
~~ROL~~ DL, 1

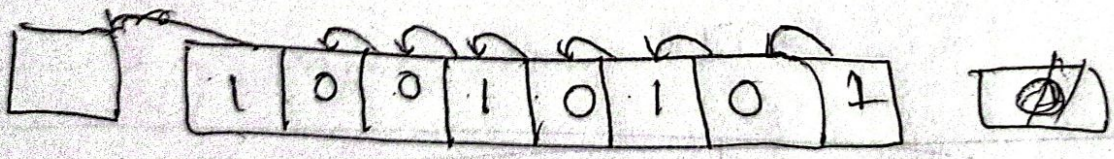
DL = 95H
= 1001 0101



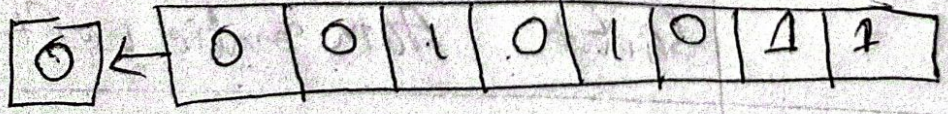
1st step



(ii) ~~SAR DL, 1~~ ROL DL, 1



1st step

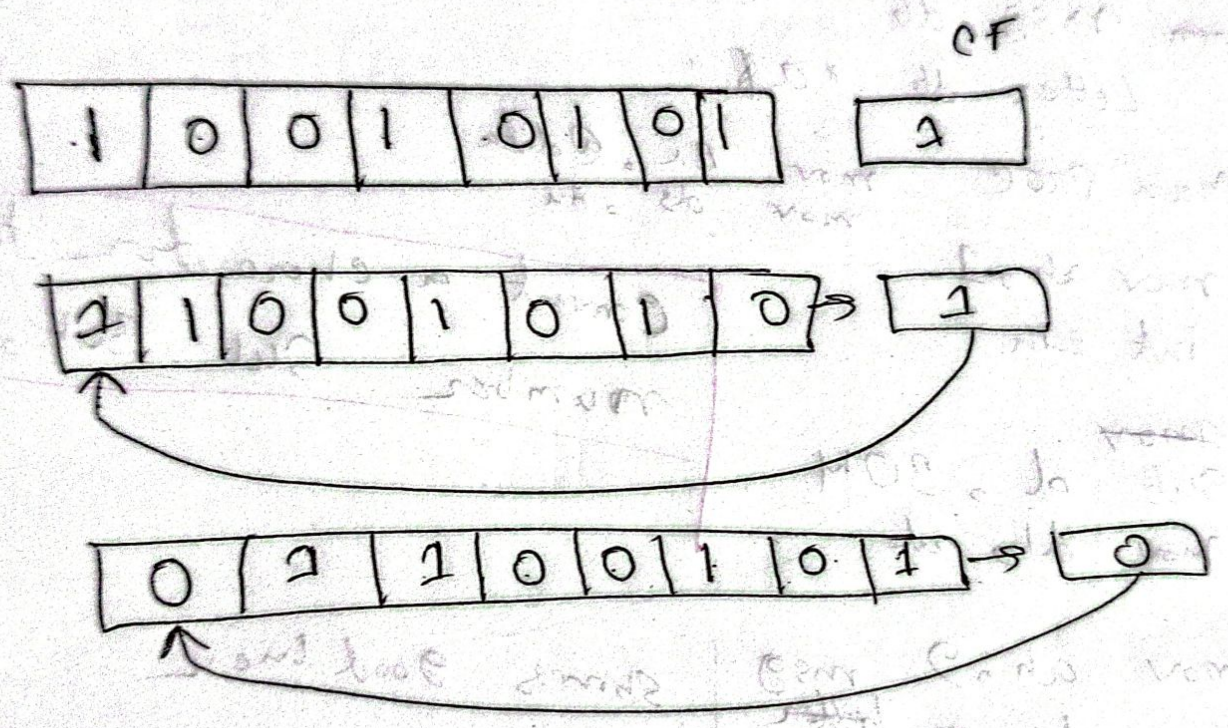


(iii) MOV CL, 2

RCR DL, CL

RCR DL, 2

DL = 1001 0101



```
.model small
.stack 64h
.code
msg db "Good Luck$"
Letter db "a$"
Main PROC  mov ax, @data
           mov ds, ax
mov ah, 1
int 21h
sub al, 30H
mov dl, al
mov ah, 9
lea dx, msg
int 21h
```

Convert a character to number Sub 30h, "0"

shows good luck

```
mov ah, 9
lea dx, letter
```

First letter.

```
Neg dl
```

2's comp = Neg.

1st way without Sub

```
mov  al, 5
mov  bl, 3
not  bl    ; 1st compliment
Add  bl, 1 ; 2s compliment
Add  Al, bl  (A + (-b))
```

2nd Sub with neg

```
mov  al, 5
mov  bl, 3
Neg  bl      (2's comp = Neg)
Add  Al, bl
```

```
.model small
.stack 100h
.code
msg db "have a good day$"
let db "A$"

main PROC
    mov ax, @data
    mov ds, ax

    mov ah, 1          (1)
    int 21h
    mov dl, al

    mov ah, 9          (2)
    lea dx, msg
    int 21h

    mov ah, 9          (3)
    lea dx, let
    int 21h
```

```
mov AL, 5
Neg AL

mov ah, 1
int 21h
Sub AL, '0'
mov dl, AL

Quit:  mov ah, 4ch
       int 21
main endp
end main.
```

(13)

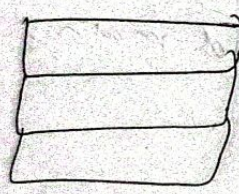
Main Procedure is the entry Point of execution, it doesn't need explicit RET Statement. The OS/emulator handles Programme termination when main Procedure Complete.

CALL saves the return Address and Ret restores it

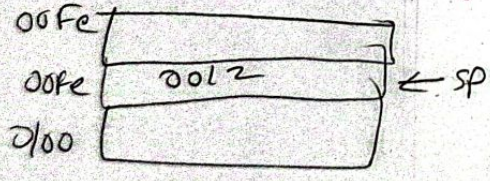
```
main Proc
    CALL Name;
IP → 0012

Name Proc:
    :
    Ret
```

Before call

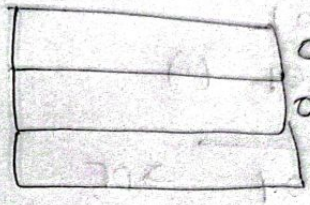


After call



Pushed IP in stack

After Ret



The ip (0012) has been Poped from stack

(14)

(b) Suppose AL = 1001 0101 (bits 7,6,5 and 3,2,1)

Clear bit 1, 5, 7 mask = 0000

	7 6 5	3 2 1 0
AL =	1001	0101
mask	0001	1101 →
	0001	0101

AND AL, 15H

(10) Set bit 0 4,6

AL = 1001 0101

mask = 0101 0001 (+)

1101 0101 → D5

OR

~~MOV~~ AL, 0D5

(11) Complement bit 4 3,5

AL = 1001 0101

1010 1010

mask 0011 1111 = 3F

~~AND~~ XOR AL, 3F

(12) Complement bl = 1001 0101 not

01101010

NOT BL

NOR

1 1 = 0

0 0 = 0

(13) Test LSB of BL

1001 0101

0000 0001

1

Test AL

(16)

- Small model
- Stack 100h
- Code,

msg db "Invalid $"

main Proc.

mov ah, 1

int 21h

cmp AL, 'A'

JB NOT_1

Case Proc:

ADD AL, 20h

mov BL, AL

(I) CMP AL, [BX]

Here Source is memory. So, its Register indirect Addressing mode.

PA = seg × 10h + offset

= (0124h × 10h) + 0567

= 17A7

(II) Sub AX, DX

Both Destination and Source is Register so it Register Addressing mode

No Physical Address.

(III) Mov AX, A [SI]

Its Indexed Addressing mode

PA = 0124 × 10h + (456h + 9h)

= 169F

(iv) mov AX, [DI]

Index Addressing mode

PA = 0124h * 10h (0378h)

= 1588

(v) CALL Sum

Relative Addressing mode

PA = Seg * 10h + off

= 0678h * 10 + 89h

= 68089h

(vi) Implied Addressing mode

No PA

(vii) IN AX, DX

I/O Addressing mode

(viii) MOV AL, A [BX] [SI]

Based Index Add

off = 0124 * 10h + (567 + 456) + 9

= 1BF0

(a) AL = 95H

= 1001 0101

(1) Clear LSB of AL

AL = 1001 0101

1001 0101 →

1001

AND AL, 95h

(b) Set MSB of AL

AL =

OR AL