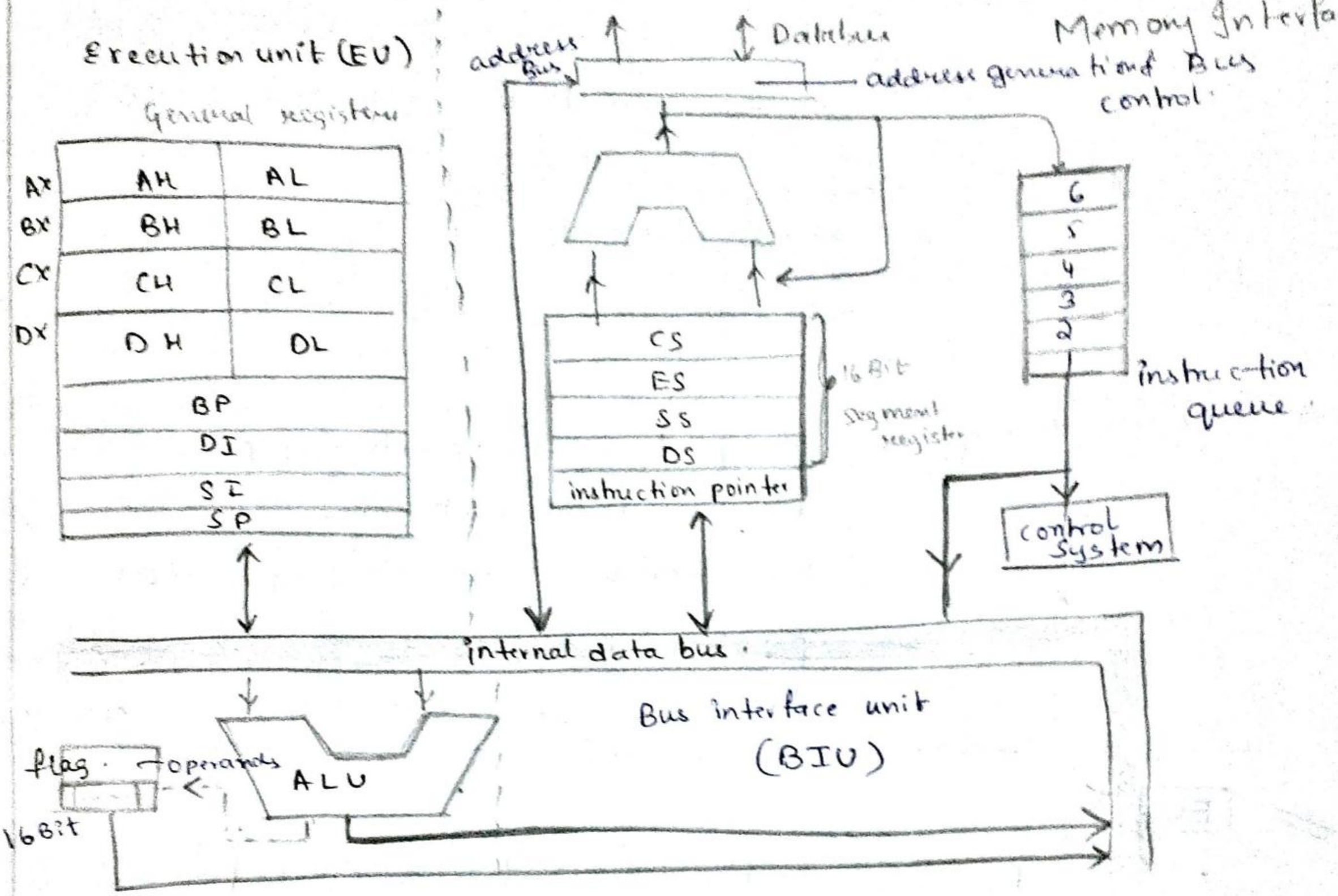
Syllabus discussion. 03-01-2022

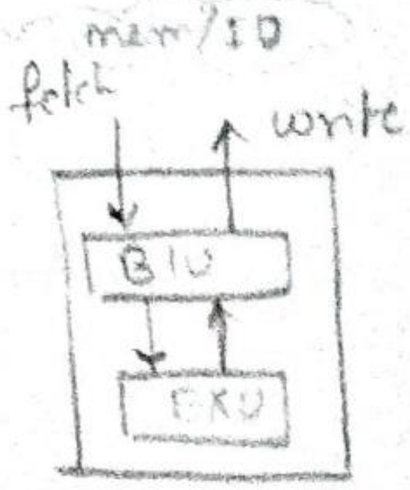
⇒ INTERNAL BLOCK DIAGRAM (8086) 05-01-2022



internal Architecture → BIU
→ EU } a independent functional units.

BIU

- BIU fetch the instructions / data from memory / IO.
- writes the data to memory.
- writes the data to i/p / o/p ports.
- reads data from ports.



3 functional parts. → Instruction Pointer (IP)
→ Segment register
→ instruction queue.

(2) control is not under programmer

→ (1) (IP): 16 Bit registers that keeps address of mem location of coming instruction to be executed
(next)

→ (2) (Segment registers): memory space (1MB) of 8086 is segmented/ divided into 4 Blocks. Each block is specified by register with maximum size 64KB.

CS	— code segment
ES	— Extra segment
SS	— stack segment
DS	— data segment

6 Bytes.
→ (3) (Instruction queue): BIU performs it's operation in parallel with EU. BIU fetches instruction byte while Execution unit is executing operations. prefetched instruction is saved in group of high speed registers. known as instruction queue

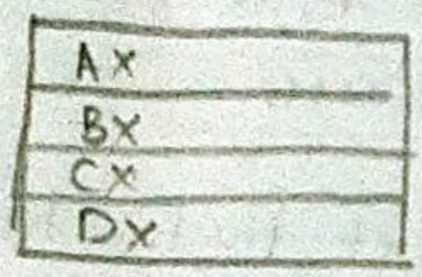
---

// [EU].

- To tell BIU where to fetch the instruction / data frame.
- To decode the instruction
- To execute the instruction.
- EU contains control circuitary to perform various internal operations.

5 functional parts: → General purpose registers
→ pointer & indexed registers
→ ALU
→ flag registers
→ timing & control unit

→ ① GPU : 8086 mp consists of 4 GP (16-Bit each).



these 16 Bits registers can be divided into

AH, AL, BH, BL, CH, CL, DH, DL

H: High order
L: Low order

AX (accumulator) used for as temp. registers.

BX (Base register) also serves as address register

CX (counter) shift/rotate operations

DX (data reg) used to point in I/O operations) (multi/divi)

→ ② Pointer & index reg: 8086 has 2 points & 2 index registers

- Stack Pointer SP
- Base Pointer BP
- Source Index SI
- destination index DI

→ ③ ALU: arithmetic logic unit (16 Bit)

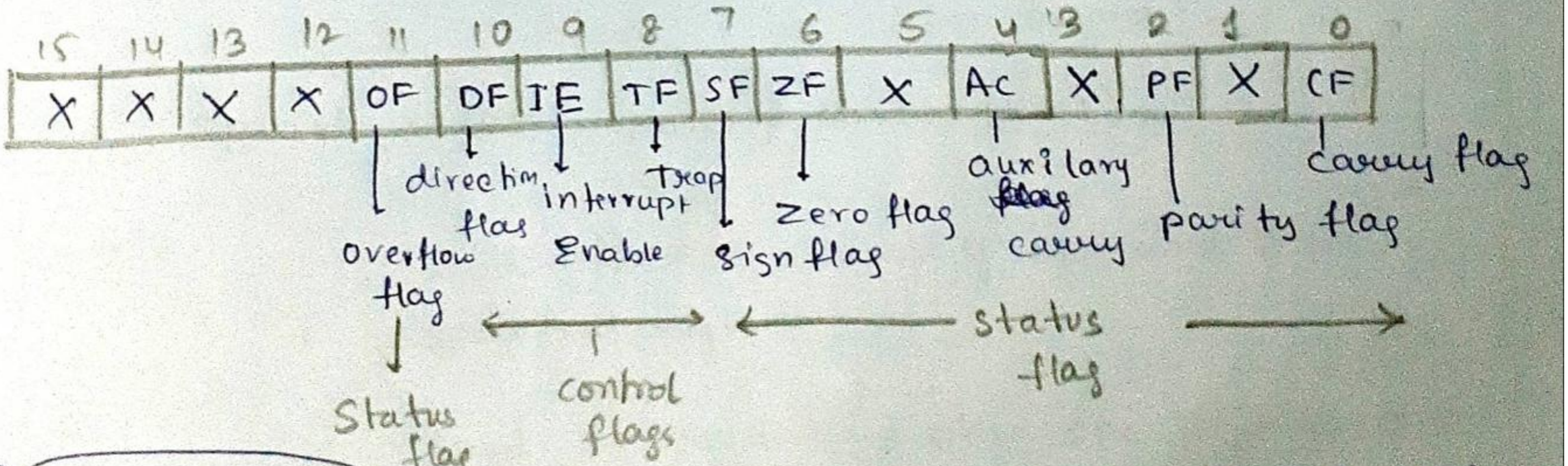
Performs arithmatic and logical operations. of 8-Bit & 16 Bit operations.

→ ④ flag: 16 Bit flag register (7 flags are unused)

we use 9 flags. → status flag (6 status flag)
→ control flag (3 control flags)

⑥ status flags: carry, auxillary carry, zero, sign, parity, overflow

③ control flags: direction, interrupt enable, Trap



→ ⑤ Timming & control: the C.U of E.U directs all internal operations & also responsible for generation of control signals.

① BIU o/p the content of IP onto the address Bus, this will cause selected byte or word to be read into BIU

② IP is incremented by 1 to prepare for the next instruction fetch

③ once fetched. inside BIU, it is passed on to the queue

④ EU draws this instruction from queue & begins execution.

⑤ while EU is executing the current instruction, the BIU proceeds to fetch next instruction.

7 3 conditions that will cause EU to enter "wait mode".

① when an instruction requires access to memory location not in queue

② jump instructions are executed.

③ Slow to execute

eg: AAM → requires 83 clock cycles to complete (advanced acoustic model)

10-01-2022.

⟹ Addressing Modes

- Way in which the processor gets data from the user is called addressing modes
- processor can get data from various sources → register
  → by Instruction
- we can also refer addressing mode as the way in which operand of an instruction is specified.

OPCODE	OPERAND

8 diff addressing modes

① immediate
② register
③ direct
④ register indirect
⑤ Based
⑥ Index
⑦ Based - Index
⑧ Based - Index with displacement

→ Implied mode

operand is specified in instruction itself.

8 bit / 16 Bit data is a part of instruction.

Zero address instruction are framed with implied addressing mod

No operand field

instruction : [data] + instr.

eg: PUSH, CLC (reset carry flag to zero)

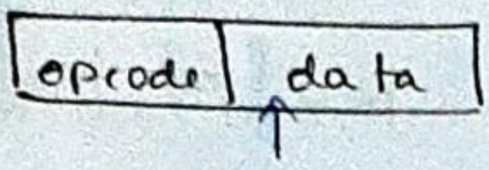
---

① IMMEDIATE

data is placed in address field of instruction.

designed as 1 add.r instruction format

eg: Instruction



8 Bit register

eg. MOV AL, 35H AL ← AL + 35H

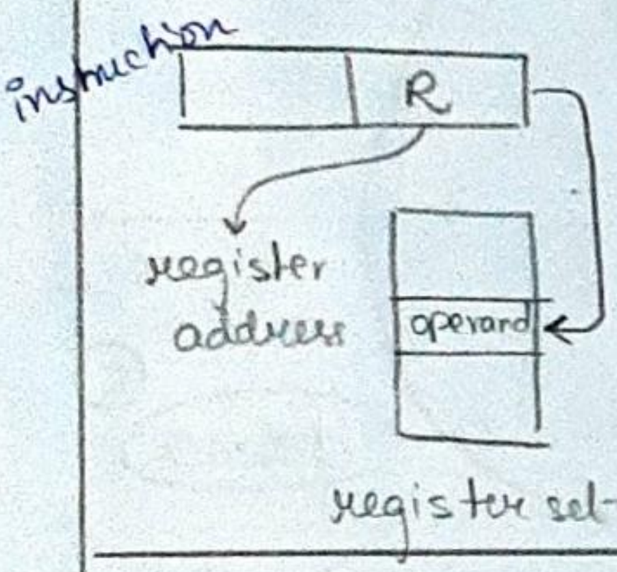
moves data 35 to AL register

limitation: range of data is limited by size of address field.

② REGISTER

In this mode operand is placed in 8 bit / 16 bit register

register is specified in instr.



16Bit

eg. MOV AX, CX

moves the contents of CX register to AX register.

③ ~~DIRECT~~ REGISTER INDIRECT

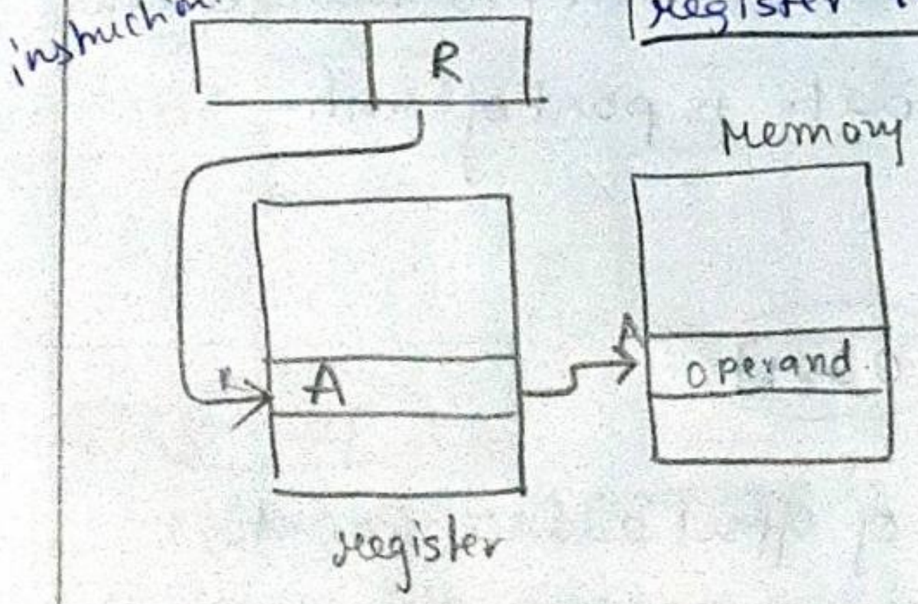
INDIRECT: requires 2 references
① to get effective address
② to access the data

data (operand) will be placed in the memory location.

address of memory will be placed in register.

register will be part of instruction.

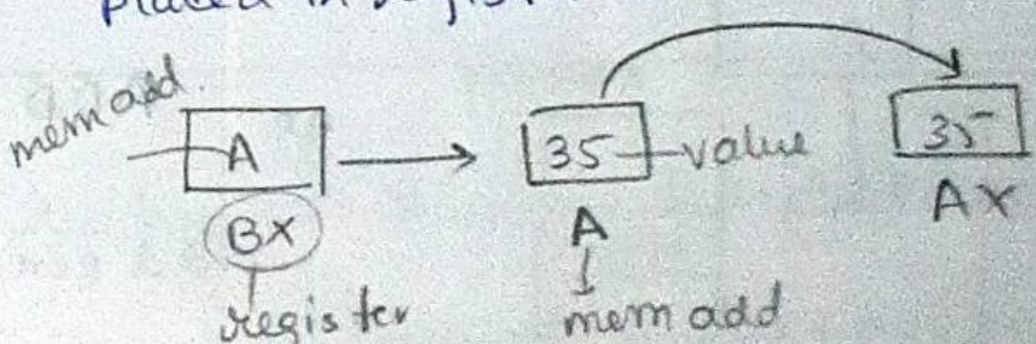
register indirect



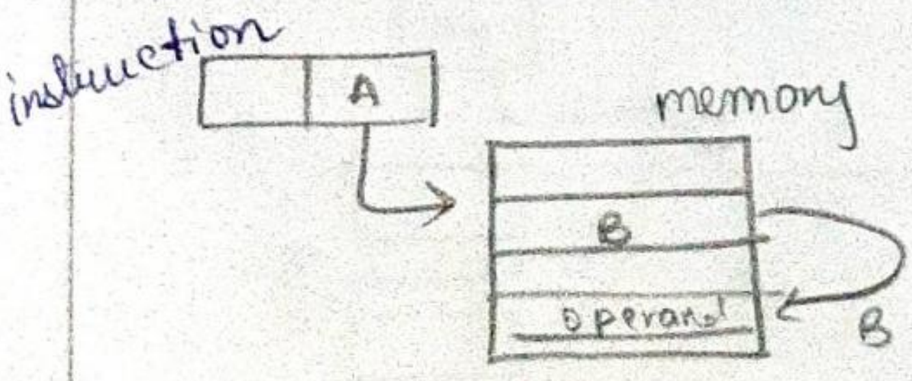
eg: MOV AX, [BX] add.

register

move contents of mem. location add. placed in register BX to register AX.



memory indirect



(effective address is placed in mem & add of mem is part of instruction)

④ INCREMENT MODE

address of operand is placed in register specified in instruction.
after accessing the operand, address in register is incremented to point the next memory location.

auto increment / decrement mode are same, implementation depends on requirements

POP → decrement mode

PUSH → increment mode

eg: ADD R1, (R2)+

(or)

R2 = R2 + d.

R1 = R1 + M[R2].

useful for stepping through arrays in a loop

R2 is start add d → size of mem. element

{ contents of register are decremented to point the previous mem. location }

ADD. R1, -(R2)

(or)

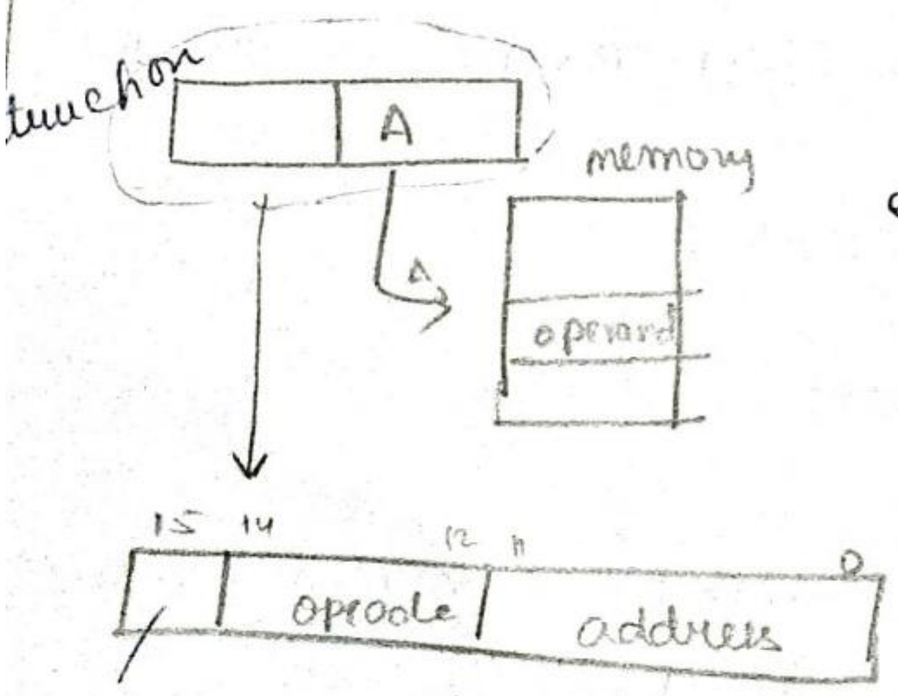
R2 = R2 - d

R1 = R1 + M[R2]

⑤ DIRECT (effective address of memory loc is written directly in instruction)

Operands address is given in instruction as 8 bit / 16 bit element

In this mode the 16 bit address of the data is part of instr.



0: direct

1: indirect

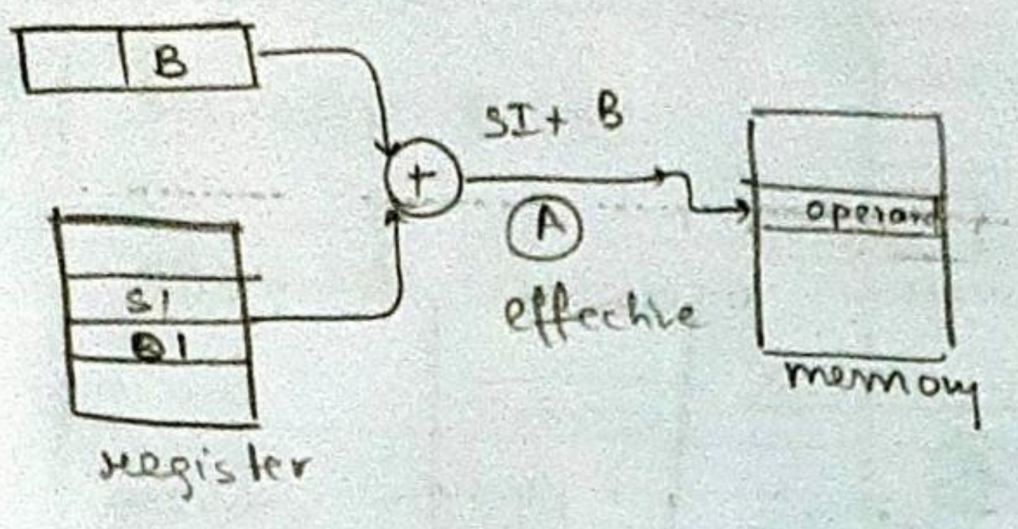
eg. ADD AL, [030L] address

add contents of offset address 30L to contents to AL

(6) INDEX

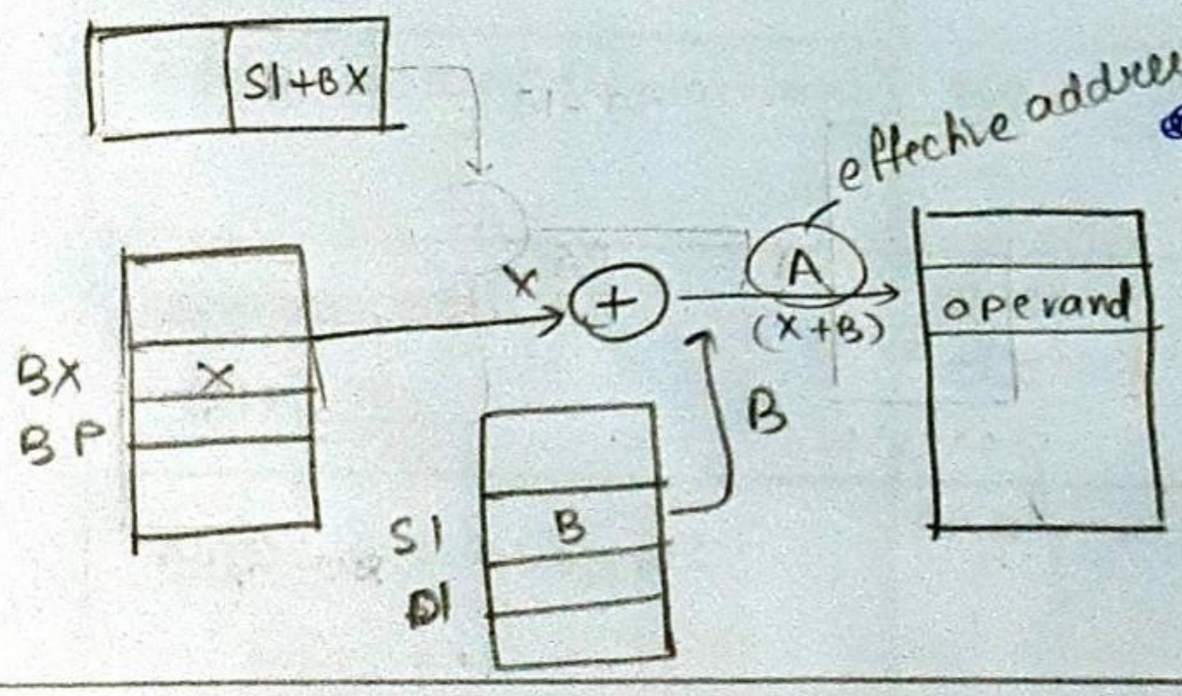
Operands add ing sum of content in index registe. SI or DI & an 8 Bit / 16 Bit displacement

eg: MOV AX, [SI+05]



(7) BASED - INDEX.

Effective addres is obtained by adding base register value to index regis ter (SI or DI)+(BX or BP)



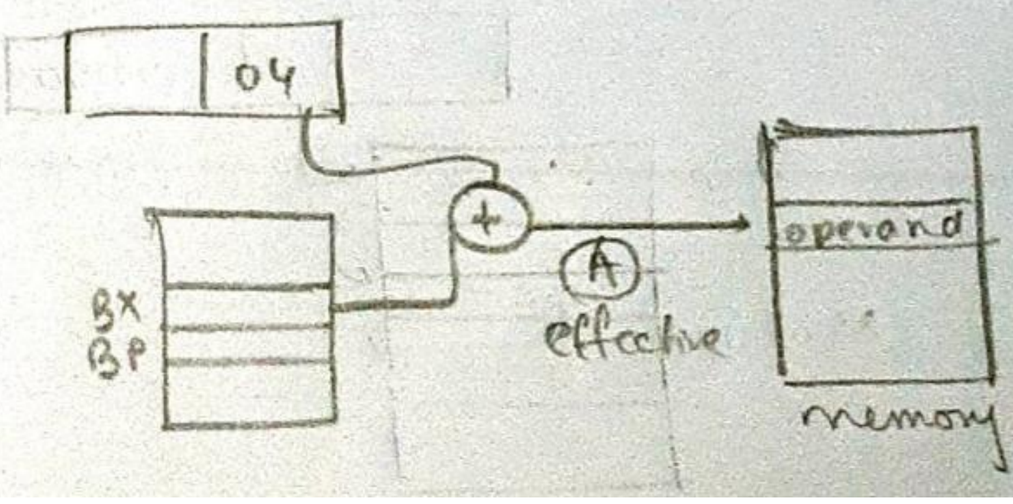
eg: ~~MOV AX,[SI+05]~~.

eg: ADD AX, [SI+BX]

(8) BASED

offset add of operand is computed by summing up base address to 8/16 bit duplacement.

eg: MOV DX, [BX+04]

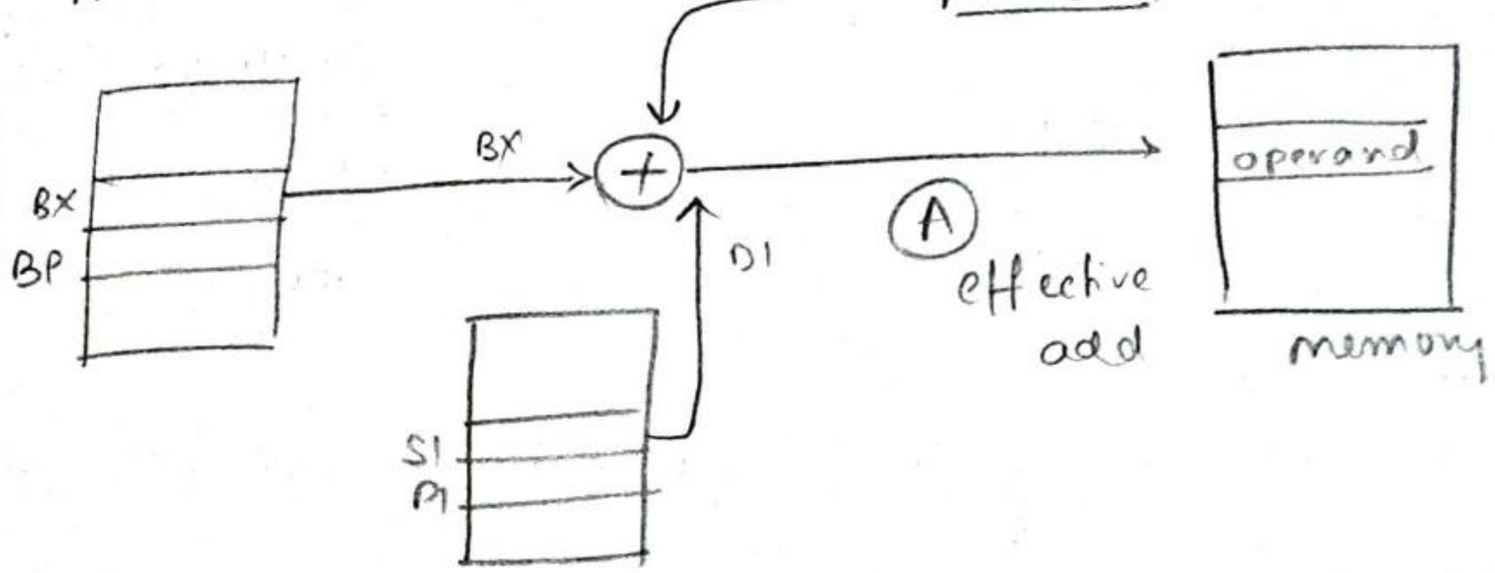


⑨ BASED - INDEXED - DISPLACEMENT

operands offset is computed by adding the base register content + content of index reg + 8/16 Bit displacement.

eg: MOV AX, [BX+DI+08]

ADD CX, [BX+SI+16]



3-Bus system Architecture

12-01-2022

- Microprocessor is a very simple machine that endlessly follows the sequence
  1. Fetch the next instruction in sequence from memory
  2. Execute the instruction
  3. Go to step 1

- Fetch - Memory - read operation (1) program counter
  (Byte or word pointed by PC is transferred from memory to instruction register in CPU.

- Execution - requires additional memory read includes (2) a memory write an I/O read (3) and I/O write (4), an internal CPU activity. (5) we donot need bunch of wires bus

1, 2, 3, 4 ⟶ we need bus to communicate

BUS — 3 set of wire. (transfer data b/w CPU, mem & I/O units)

- address bus
- control bus
- data bus

Bus cycle Timing.

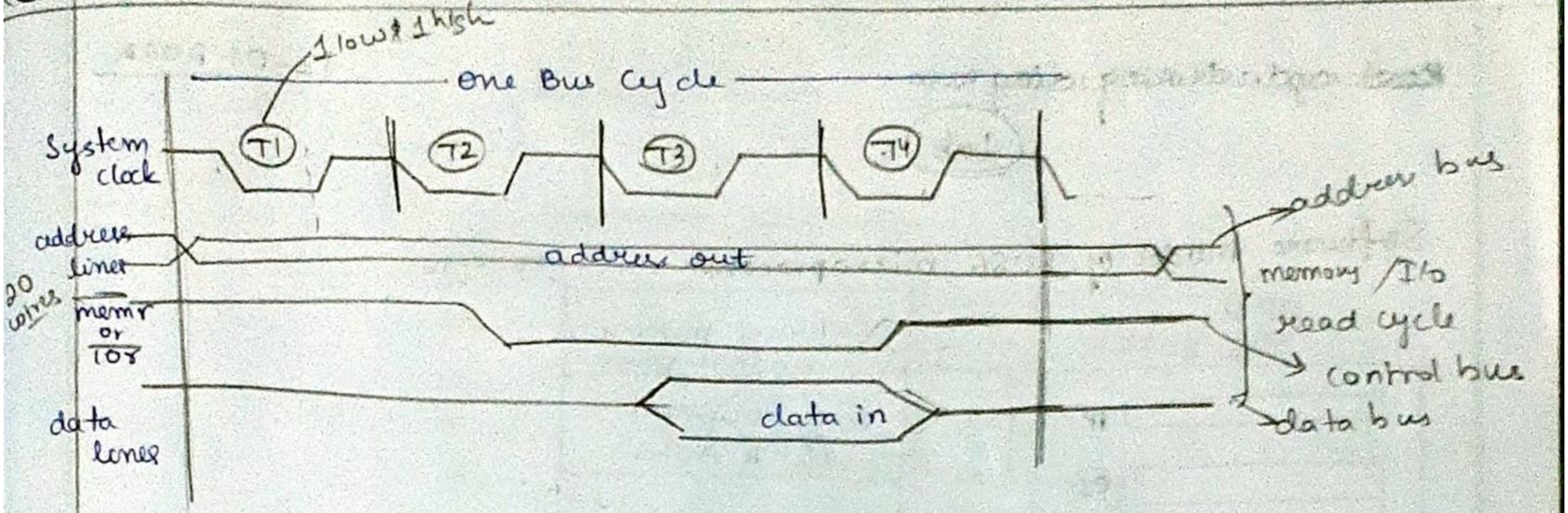
- Each cycle begins with the output of memory / I/O port address during the T1 clock ~~day~~ cycle.
- parallel lines indicate that some of the lines are assumed to be high and others low. (out of 20 lines)
- Examining address lines only it is not possible, to determine if this is a memory or an I/O address.
- unable to tell the direction of data flow. mem-nunchi processor (or) processor to mem.

∴ Control Bus is required ✓

↓

read cycle mem to processor

write cycle processor to mem



→ (T4) : MP expects the data to be on data bus lines. (MEMR) — control signal latches the contents of these lines, releases

control Bus consists of 4 active low signals

- $\overline{MEMR}$ (memory read)
- $\overline{MEMW}$ (memory write)
- $\overline{IOR}$ (I/O read)
- $\overline{IOW}$ (I/O write)

→ (T1) Processor outputs 20 bit address.

⇒ (T2) the $\overline{MEMR}$ control line is driven low. memory unit recognizes this bus cycle as a memory read. prepares to place addressed byte or word onto the data lines.

(data won't be place on data line at T2, it will only search the loc of data's present for 20 bit address given)

→ (T3) MP configures its data bus lines for input.

LOOKUP data byte or word

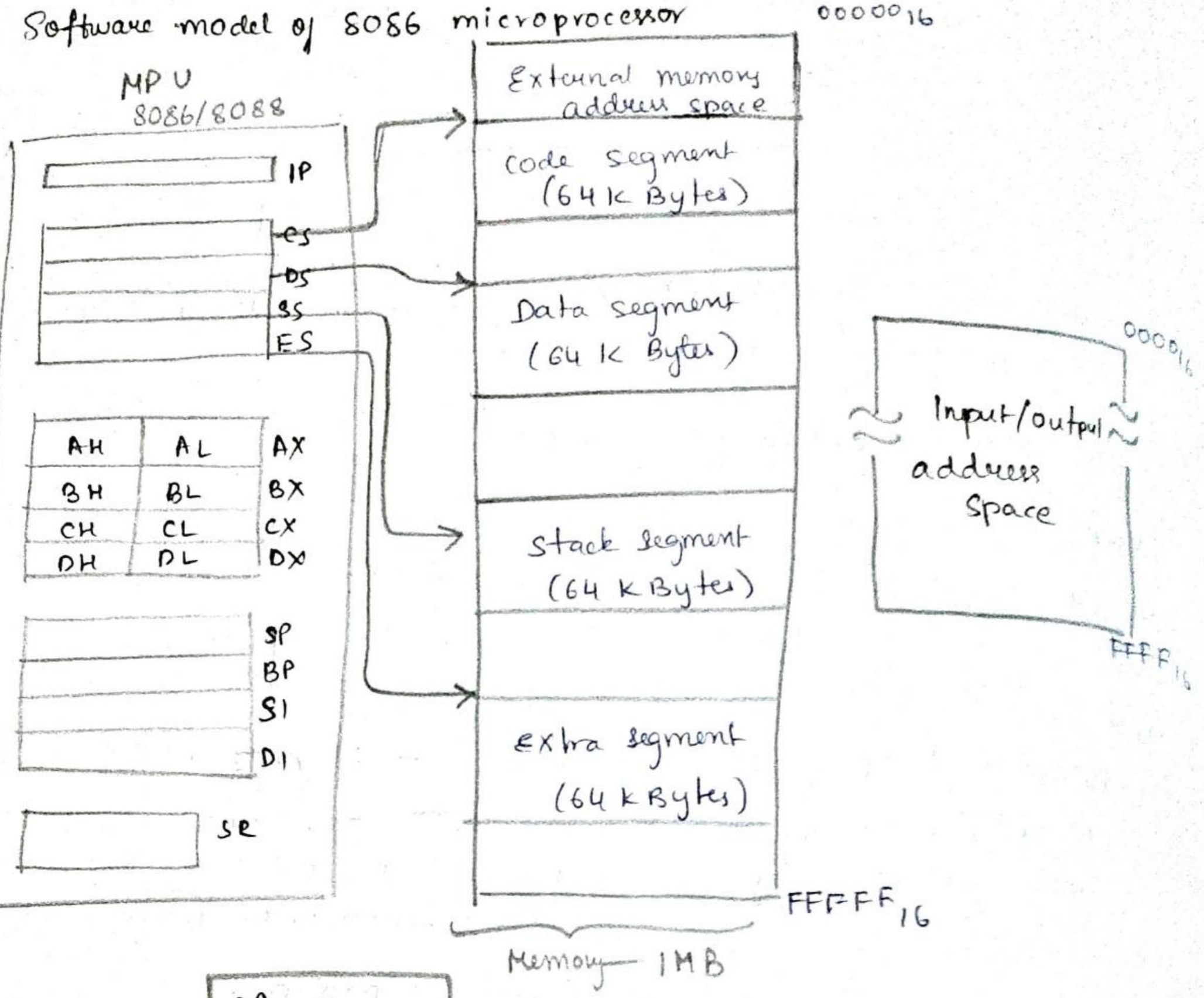
Read Cycle.

13-01-2022

Read cycle timing diagram.

(lab).

Software model of 8086 microprocessor



Pin diagram

- 8086 is 1ST processor available in 40 pins

address → (20) bit
data → (16) bit

- AD0 - AD15 → multiplex (address + data bus) } 16 Bit Microprocessor

when ALE = 1 acts as address bits

when ALE = 0 acts as data bits

- A16, A17, A18, A19 → demultiplexer (address).
- S3, S4, S5, S6, S7 → status signals

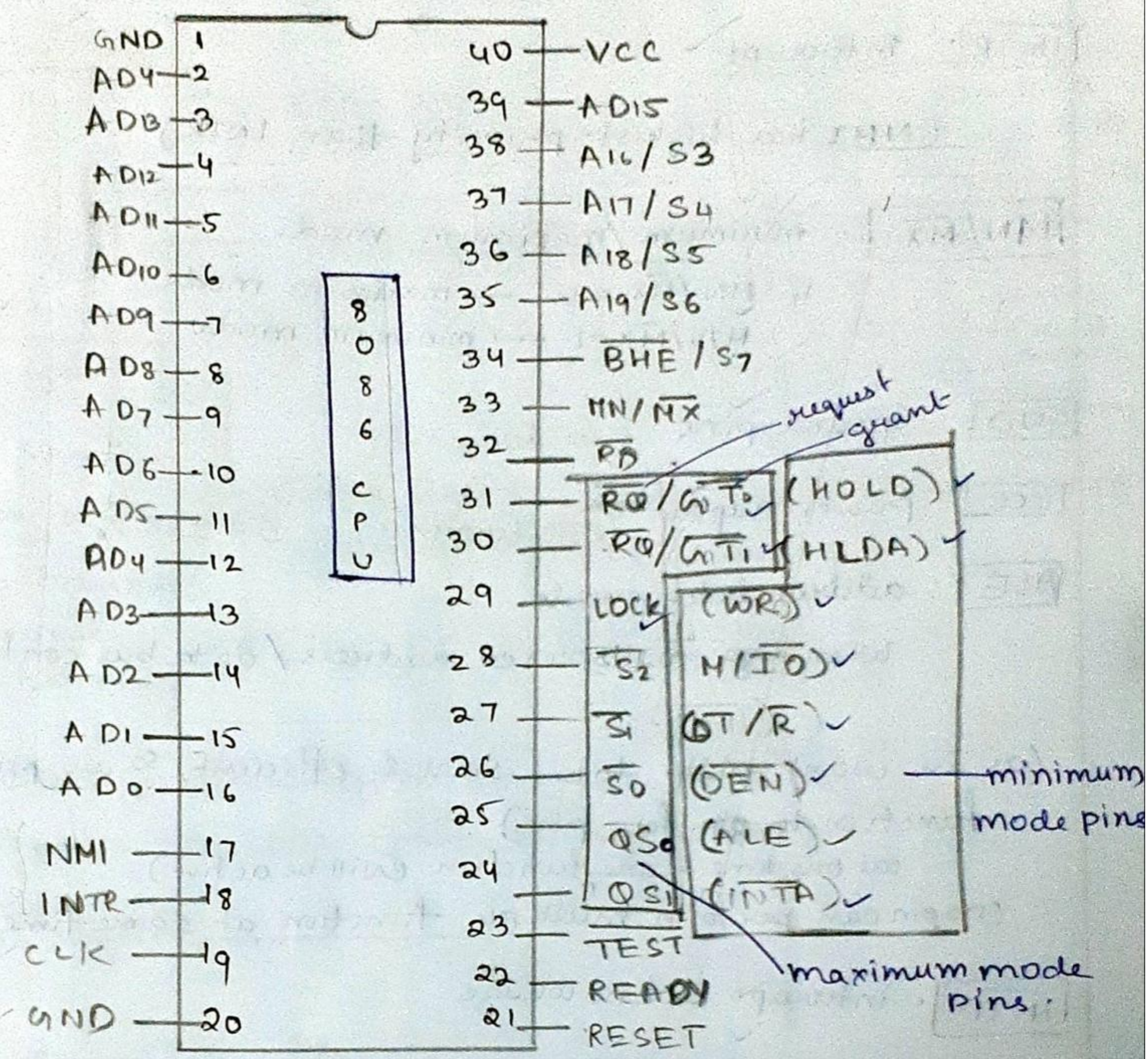
address & status bus.

during 1ST clock cycle it carries 4Bit address.

after that for remaining time it acts as Status Signal so.

8086 – 40PIN DIP (dual in package)

PIN DIAGRAM



BHE: Bus high enable (used to distinguish b/w low byte & high Byte for 16 Bit external data bus)

$\overline{BHE}$	$\overline{A_0}$	
0	0	→ 16 Bit (A0–A15)
0	1	→ 8 Bit (A8–A15)
1	0	→ 8 bit (A0–A7)
0	0	→ idle

READY: used to activate low active signals & insert wait statements

RESET: when this signal is high:
CS: FFFF
IP: 0000
DS, ES, SS: 0000.
Instruction queue → empty
all flags are cleared.

CLK: clock signal
this pin is connected to clock generator (8284)

NMI : non-maskable interrupt

we cannot discard, when it comes to processor, it has to ~~exe~~ execute NMI first then goes to INTR.

INTR : interrupt.

(NMI has highest priority than INTR)

MN/$\overline{MX}$: minimum / maximum mode.

if MN/$\overline{MX}$ = 0 ⟶ maximum mode

MN/$\overline{MX}$ = 1 ⟶ minimum mode

GND : Ground pins (-ve terminal)

VCC : power supply (+ve terminal) 5 V ± 10%

ALE : address latch enable

when high multiplexed address / data bus contains address info.

(as we want chip to be small & efficient, so we provide multiple function to ~~pin~~ few pins)

(at one time – one function will be active)

(no pin can perform multiple function at same time).

$\overline{INTA}$: interrupt acknowledge

when interrupt is received processor should acknowledge that I am going to serve your interrupt

HOLD (hold) / direct memory access (b/w mem → I/O)

HLDA if we want to transfer data without involving CPU, then

hold acknowledge — System bus hand overs to DMA.

S6: logic 0

S5: indicates condition of IF flag bits

S4–S3: Indicates which segment is accessed during current bus cycle

S4	S3	
0	0	→ extra segment
0	1	→ stack segment
1	0	→ code / no segment
1	1	→ data segment

S7: always 1.

S2	S1	S0	
0	0	0	→ INTA
0	0	1	→ read I/O
0	1	0	→ write I/O
0	1	1	→ halt
1	0	0	→ code access
1	0	1	→ read memory
1	1	0	→ write memory
1	1	1	→ non-passive

Control Signals (minimum mode)

- $\overline{RD}$: read signal
- $\overline{WR}$: write signal
- $M/\overline{IO}$: memory or I/O
- $DT/\overline{R}$: data transmit / receive
- $\overline{DEN}$: data bus enable.

LOCK : used to lock peripherals off the system.
activated by using the LOCK: (prefix on any instruction)

$\overline{RQ}/\overline{GT_0}$
$RQ/\overline{GT_1}$: DMA request / Grant

QS_0
QS_1 : queue status

QS_1	QS_0	
0	0	→ queue is idle
0	1	→ first byte of opcode
1	0	→ queue is empty
1	1	→ subsequent byte of opcode

DOS Box.

```
.model small
.stack 64
.data                    data byte
A db 03h                 8 bit
.code

mov ax, @data
mov ds, ax
        data segment

lea si, A
mov bl, [si] - index
mov al, 0001h - immediate

repeat:
mul bl (3)
dec bl (2)
cmp bl, 00h - 2 == 0x
jnz repeat

mov ah, 4ch
int 21h
end
```

3 × 2 → 1 → 1 == 0x

6 × 1 → 0 → 0 == 0✓

`-d ds:000A` — will display value of A

17-01-2022

→ flag register 8086.

Q if AL = 7F, use instruction ADD AL, 01. What will be result? how the 6 status flag gets affected?

(7F) in decimal → $7 \times 16^1 + 15 \times 16^0 = 127$.
hex
↓ in binary
(0111) (1111) → $0\times2^7 + 1\times2^6 + 1\times2^5 + 1\times2^4 + 1\times2^3 + 1\times2^2 + 1\times2^1 + 1\times2^0$
in decimal
nibble (higher) nibble (lower)
$\Rightarrow 0 + 64 + 32 + 16 + 8 + 4 + 2 + 1 \Rightarrow 127$

4 bits — nibble
8 bits - word
16 bits - double word

7F + 01 ⇒ 0111 1111 (carry from lower nibble to higher nibble)
+ 0000 0001
then (AF=1).
SF (1) 0000 0000 → (80) in hex. (80H)
D7 D6 D5 D4 D3 D2 D1 D0

	R	C
0 - 0	0	0
0 - 1	1	0
1 - 0	0	0
1 - 1	0	1
1 - 1 - 1	1	1

Status flags: AF(1) ; ZF(0) , SF(1) , PF(1) , OF(1) , CF(0)
AF: auxillary; ZF: zero; SF: sign; PF: parity (1 assuming (even)); OF: over flow; CF: carry flag

if our ans exceeds $\pm\frac{2^n-1}{2}$ then overflow = 1, else overflow = 0.
here $\pm\frac{2^8-1}{2}$, (+127 / -127) range
but 80H → (-128) ∴ overflow = 1
1000 0000
2's complement: 0111 1111 + 1
1000 0000 = 128
∴ 1000 0000 is (-128)

Q MOV AL, 05
NEG AL

	D7 D6 D5 D4	D3 D2 D1 D0
→	0000	0000
⊖	0000	0101
①	1111	1011

= -5H.

	R	B
0 - 1	1	1
1 - 0	1	0
0 - 0	0	0
1 - 1	0	0

1 1111 1011
∴ 2's complement → 0000 0100 + 1
0000 0101 = 5
1111 1011 → (-5)

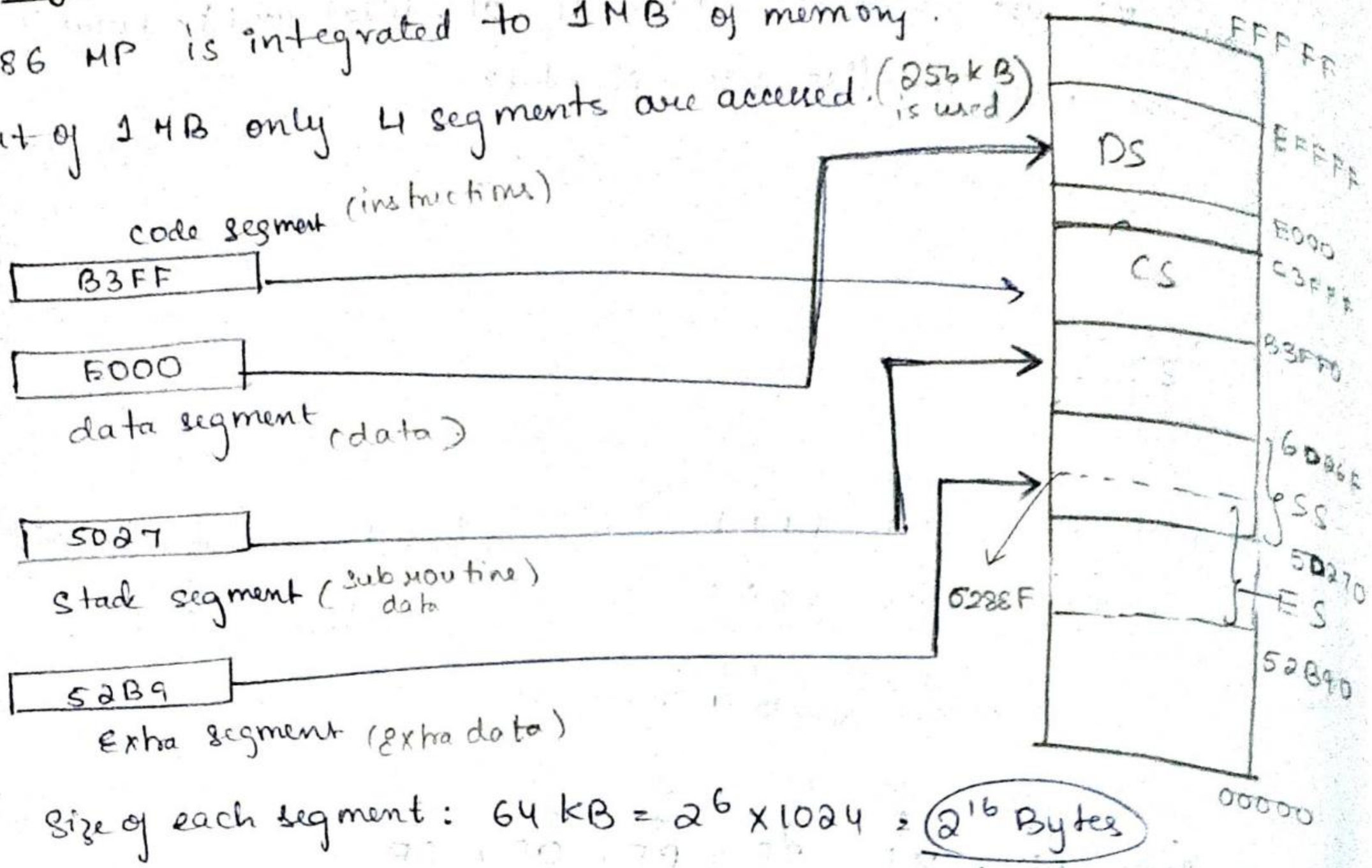
AF=1, CF=1, PF=1, SF=1 (even), ~~ZF~~ = 0, OF = 0

19-01-2022

Segment Registers.

8086 MP is integrated to 1 MB of memory.

out of 1 MB only 4 segments are accessed. (256 kB is used)



Size of each segment: $64\ kB = 2^6 \times 1024 = 2^{16}$ Bytes

to convert to
16 Bit add ⟶ 20 Bit address

(we need to append 0000)
LSB

top add of core segment → B3FF0 (2^{16}/FFFF) ⇒

```
  B3FF0
+ FFFF
-------
  C3FFF
```

Base location, displacement/logical/offset

```
  52B90
+ FFFF
-------
  628EF
```

0101 0010 1011 1001 0000
+ 1111 1111 1111 1111 1111
0110 0010 1011 1000 1111

6 2 8 F

Q If IP = 1000H CS = B3FFH

physical add → B3FF0
+ 1000

1011 0011 1111 1111 0000
+ 0001 0000 0000 0000 0000

B 4 F F 0

Q what physical mem. location is accessed by. instruction

MOVE [BP], AL if BP = 2C30H.

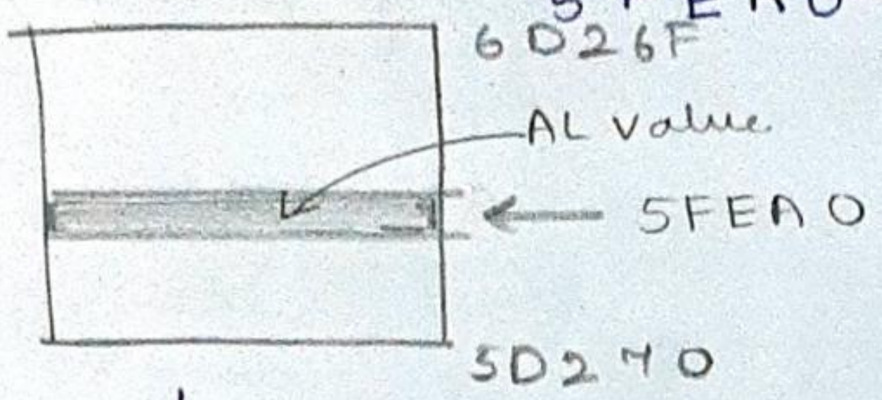
assume segment definition given above.

we have to store value in AL in BP / at location [BP].

default: segment register ⟶ STACK SEGMENT

Base — 5D270.

physical mem: 5D270
+ 2C30
5FEA0 → at this location, AL value is stored.



↓

stack segment

Table : Segment register assignments.

type of mem ref	default segment	alternet segment	offset (logical address)
instruction fetch	CS	none	IP
stack operation	SS	none	SP
General data	DS	CS, SS, ES	effective address
string source	DS	CS, SS, ES	SI
string destination	ES	none	DI
BX used as pointer	DS	CS, ES, SS	Effective address
BP used as pointer	SS	CS, ES, DS	Effective address

20-01-2022

Lab

→ write an ALP to find largest number in array

input → count (4) A[0] = 03 A[2] = 09
A[1] = 05 A[3] = 08

Code

```
.model small
.stack 64
.data
.code
start: Mov ax, @data
Mov ds, ax
Mov si, 3000h
Mov di, 3500h
Mov cl, [si]
Mov ch, 00h
Inc si
Mov al, [si]
dec cl
inc si
next: cmp al, [si]
jnc fit
Mov al, [si]
fit: Inc si
loop next
Mov [di], al
hlt
end.
```

base address

entering count value

(offset)

result

destination

count value

first element place

move first element

out of 4. 1 element is moved

2nd element place cheyadaniki

compare value in register & value in memory

if al > [si] else

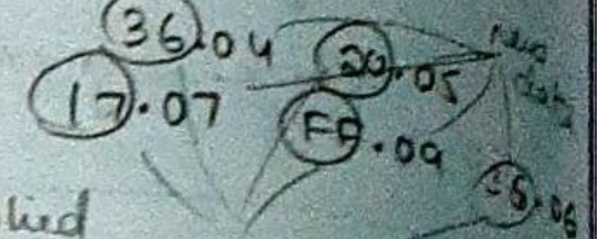
jc if we want smallest number

compare next element

final result

masm file.asm
link file.obj
debug file.exe
(-t) to trace
-e ds:3000

3000 n1 n2 n3 n4
space



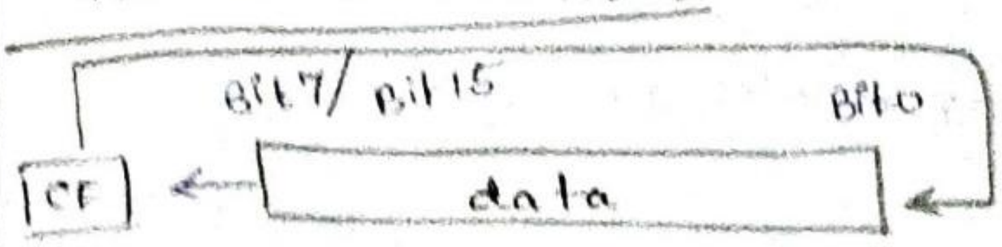
old data

-d ds:3000
(will show previous values stored)

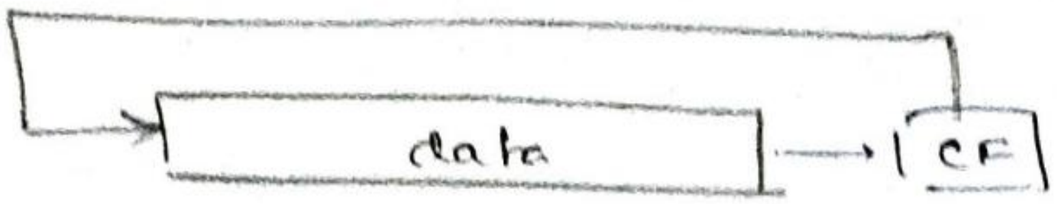
loop operations are applied on CX register or whole CH-00 to 0 until it will lead to diff value in count

24-01-2022.

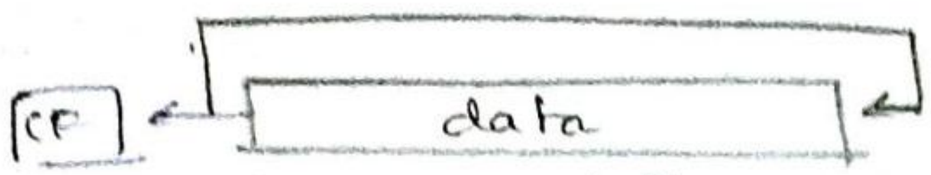
Rotate Instructions



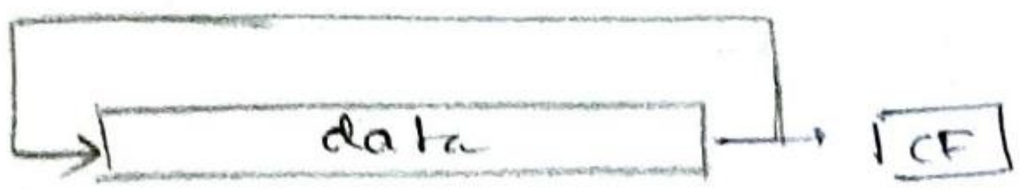
rotate left through carry (RCL



rotate right through carry (RCR



rotate left without carry (ROL



rotate right without carry (CF).

Q determine contents of registers AX, BX, CL after following program is run.

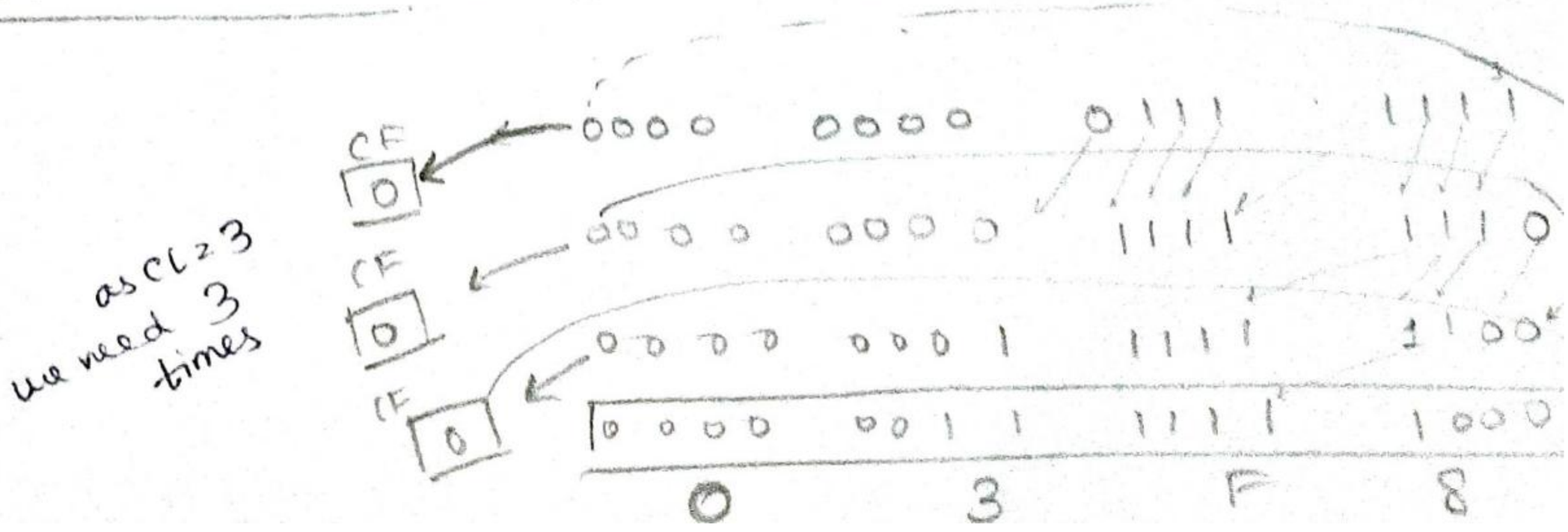
```
MOV CL,3
MOV AX,7FH
MOV BX,0505H
ROL AX,CL
AND AH,BH
OR BL,AX
```

→ MOV CL,3
AX, ???? BX. ???? CX = ??03

→ MOV AX,7FH
AX = 007F BX = ???? CX = ??03

→ MOV BX = 0505H
AX = 007F BX = 0505 CX = ??03.

→ ROL AX,CL



→ after ROL AX, CL

AX = 03F8 BX = 0505 CX = 0103

→ AND AH, BH

(higher order) first 8 bits

0000 0011
0000 0101

0000 0001

AND

0 1

∴ AX = 01F8 0505 = BX CX = ??03

→ OR BL, AL

1111 1000 — lower order (last 8 bits)
0000 0101

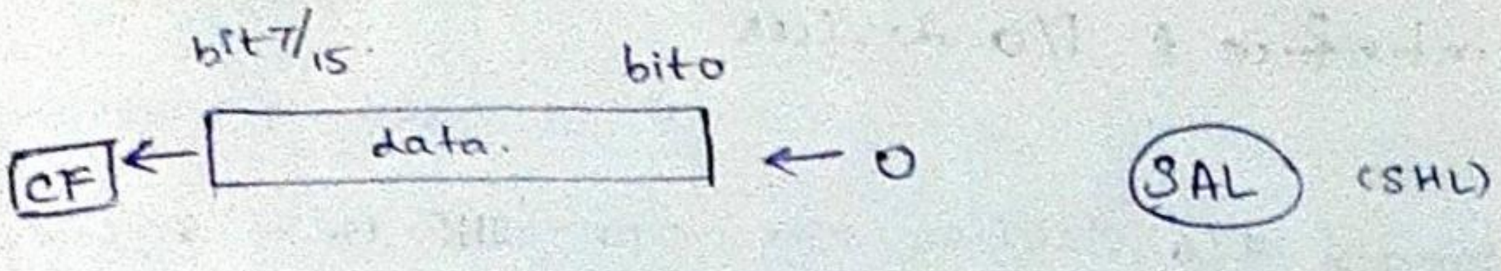
1111 1101

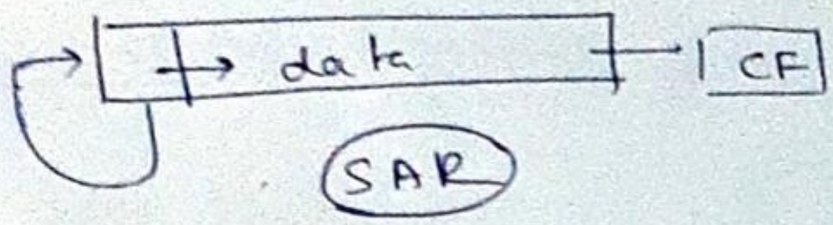
F D

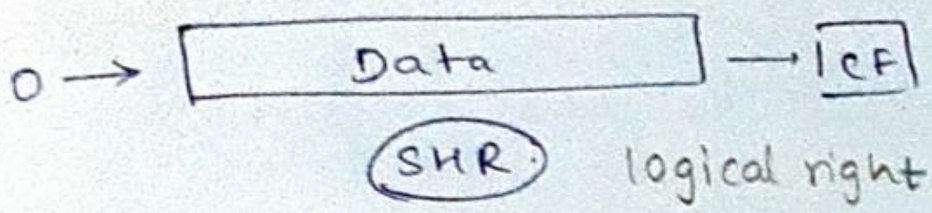
AX = 01F8 , BX = 05FD CX = ??03.

shift instruction (sign remains constant)

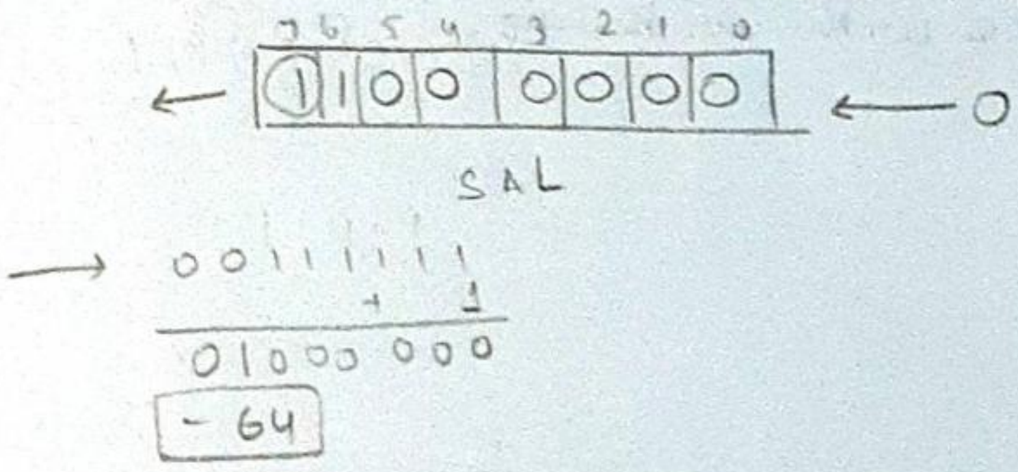
31-01-2022

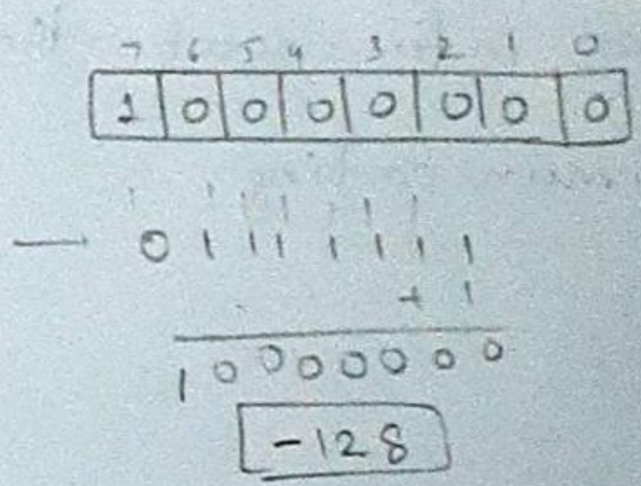


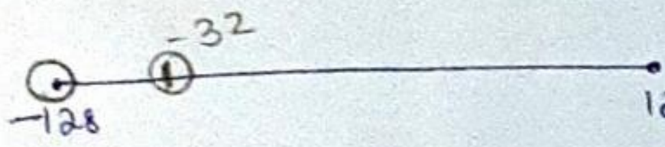




Consider a data byte C0H. Apply SAL.



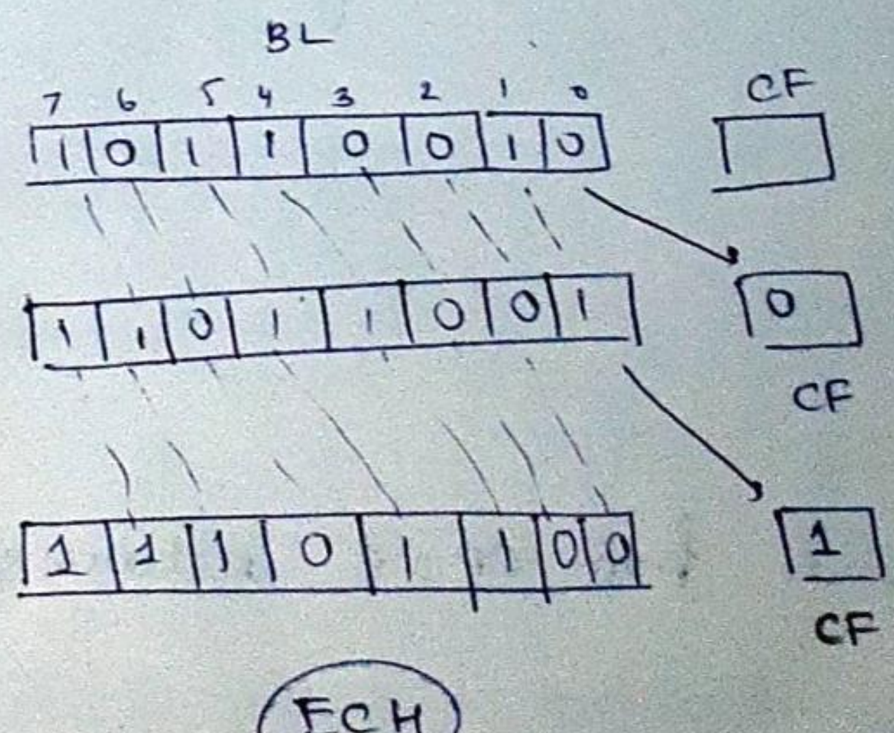




(out of range : overflow condition)

what is content of register BL after following instructions are executed?

```
MOV BL, B2H
MOV CL, 2
SAR BL, CL
```



ECH