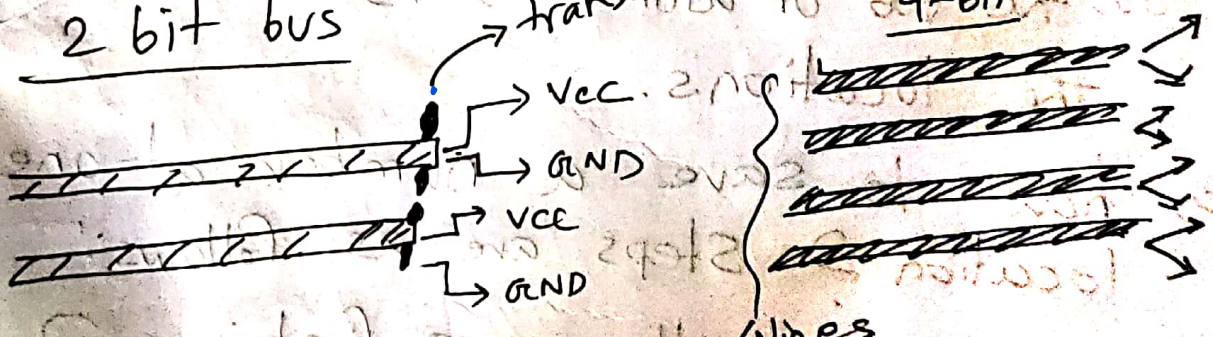
1. Memory ⇒ stores prog & data

2. MPU → fetch
→ decode
→ execute

3. I/O ⇒ give i/p
⇒ shows o/p

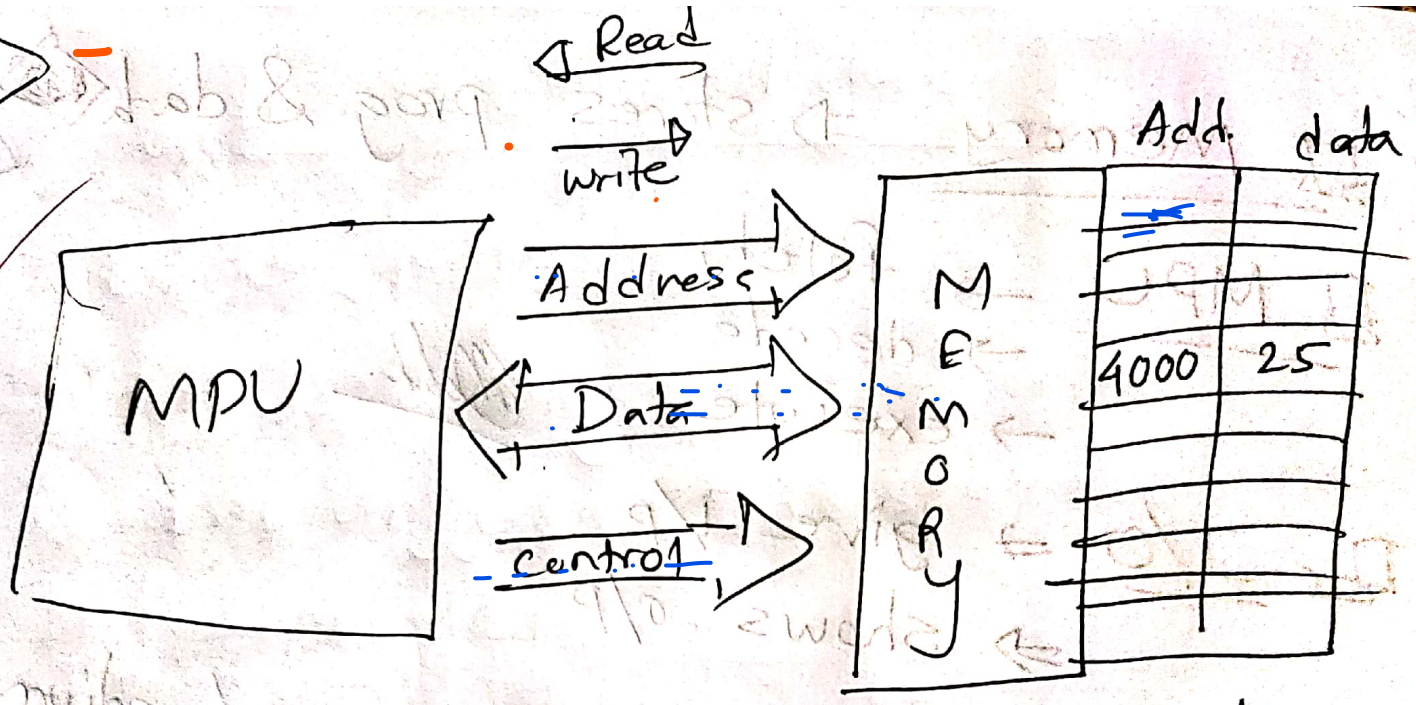
4. Buses ⇒ set of lines/mediums.
- Address
- data
- control

2 bit bus



It is always a trade off; since the more lines the more info will come, so the cost will be higher & also the space will be an issue.

12



Since there are millions of locations, every location has its own unique address. (during admission test you can remember).

So, the job of address bus is to identify the locations.

So, how to save a number at one location? steps are as follows:

1. Address bus will come first; will give the address - 4000
2. data bus will carry 35;
3. control bus will WRITE 35 in location 4000.

WRITE operation

⇒ Read & write always mention with respect to MPU. (permitted by MPU)

8086 Architecture

→ 16 bit MPU ; so always deals with 16-bit transmission. EU-2

→ two units

→ BIU & EU.

→ for every MPU, you need to find out : 3 things → fetch, decode & execute.

So, that you can trace out every MPU. since all MPU does the 3 above tasks.

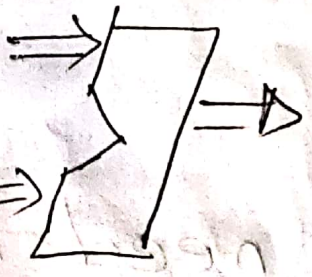
8086 Architecture

→ 8086 has 16-bit data bus.

→ 1st thing: every thing is rectangular in shape.

except: 

Arithmetic circuit



→ 

Physical address calculation

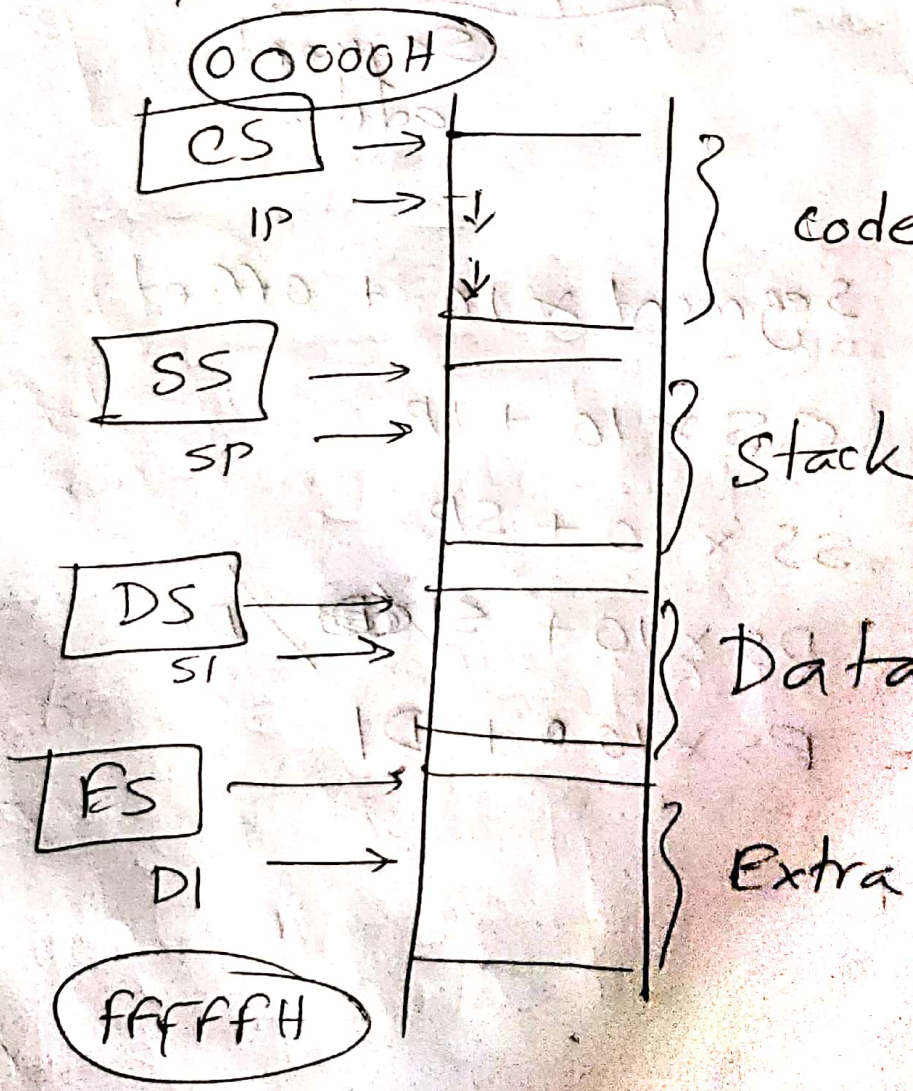
Segment x 10 + offset

= 20 bit address.

o] TB → inside HD

{ folder name
file name } virtual address

{ in reality there are 1 trillion location



Memory is divided into four segments.

Assume: you want to find out page 564.

So, if all the chapters are 100 page you will go to ~~chapter 5-0~~

pg - 64 of chapter 5 -
offset add. segment add.

Physical = Segment x 10 + offset

Calculated by MPU

= CS x 10 + IP
= SS x 10 + SP
= DS x 10 + SI / DI
= ES x 10 + DI

Who will fetch the instructions?

→ the BIU ; so the physical address calculation should be done by BIU. for fetching the next instruction, while EU is executing.

CS/DS/ES/SS → are not segments;

Segments are present in memory.

these are segment registers they contain the starting addresses of all the segments, (16-bit Reg)

After calculating the physical address, instructions gets fetched through the data bus

// Address bus → locates the location where to go.

// data bus → instructions/ data comes in/out.

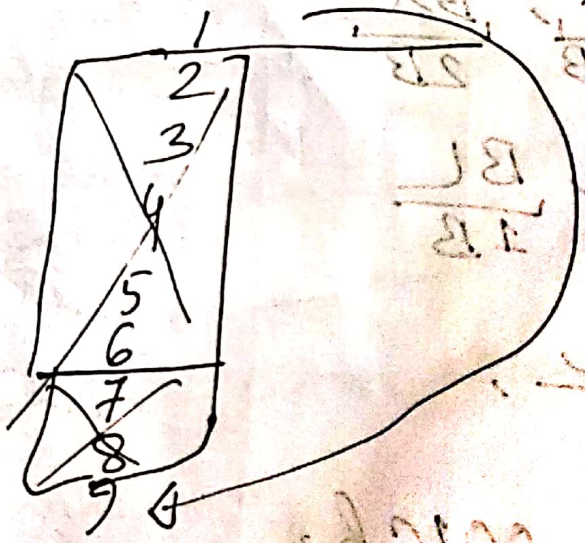
// control bus → gives $\overline{RD}$ = Read, $\overline{WR}$ = write signal.

// After fetching, we are ~~not~~ going to execute right now, bec. some execution is currently going on.

=> the next 6-bytes of the program will be fetched.

★ Pipelining fails when there is a branch, so at that time the 6-byte instructions, what had been fetched should flushed out.

→ MP assumes the program will go in a seqvencial manner.



Will the pipelining stops?

→ No, the pipeline will again start from instructions 2

What are functions?

1. Calculate physical address (20 bit)
2. prefetch instructions from memory
3. Manage the queue of 6 Byte.

MOV AX, BX
2B 2B 2B

MOV AL, BL
2B 1B 1B

MOV AL, 4Ch
2B 1B 1B

MOV BX, 001Ch
2B 2B 2B

CS / DS / ES / SS ⇒ segment registers.

IP, SP, SI, DI ⇒ holds the offset address of the next instruction.

Execution Unit

→ Control system = decodes the instructions

⇒ we write, AND BL, CL
but what has come is opcode of (AND BL, CL) ⇒ 0111101111

⇒ Control system releases the control signal.

So, steps are:

1. fetched
2. decoded
3. control signals RD/WR are released.
4. Execution.

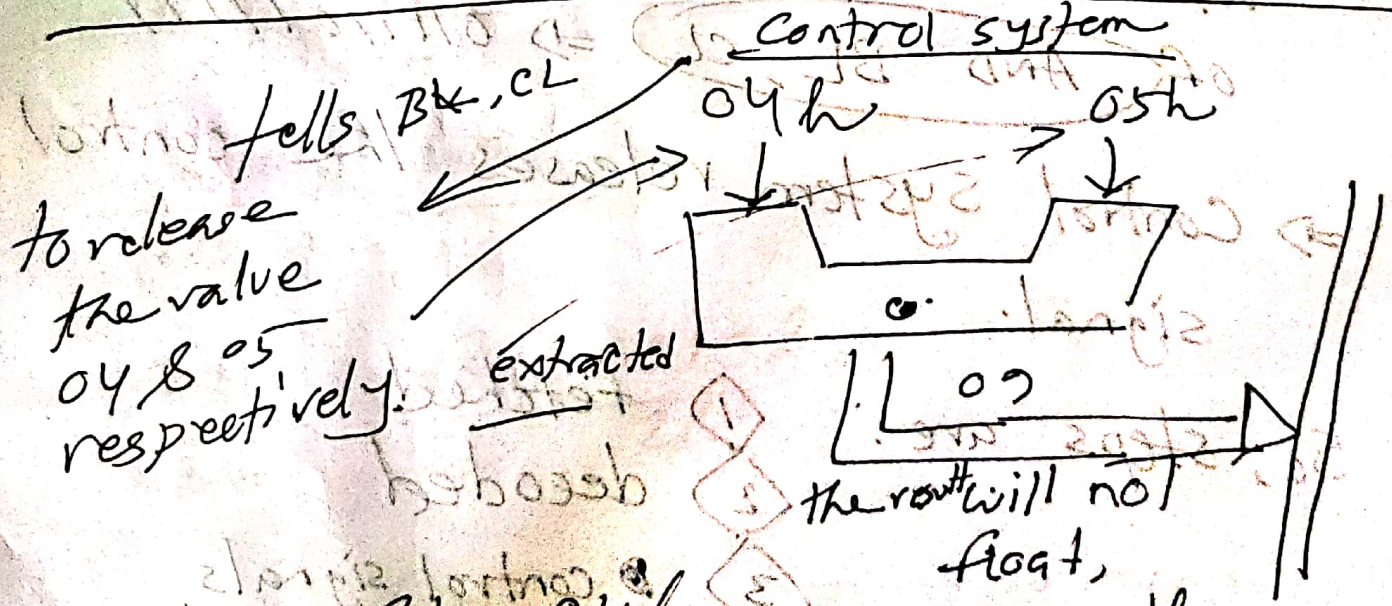
AX, BX, CX, DX

General purpose Register.

→ are assigned for a programmer.

X ⇒ means combination of two.

```
mov CL, 34H ;
mov CH, 12H ;
mov CX, 1234H ; → single instruction.
```

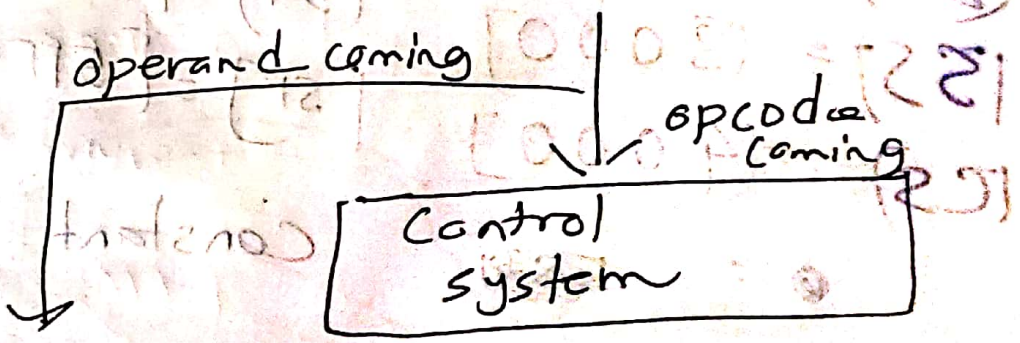



```
mov BL, 04h
mov CL, 05h
ADD BL, CL
```

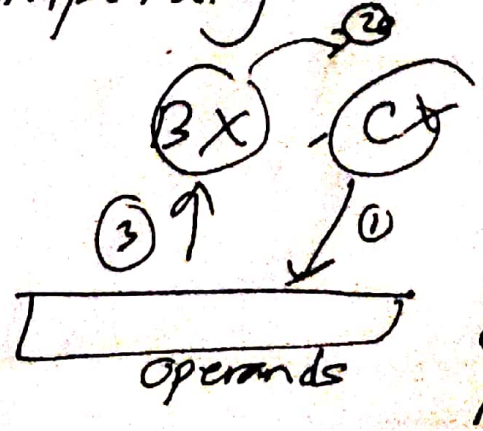
MOV BL, 04h
opcode ; operand

ADD BL, CL
opcode

- we decode the opcode
  we add the operand.



- operands ⇒ its a temporary register.
  → not available to the programmers
  → used by the up. only; for storing temporary values.



done by MP.