C O A

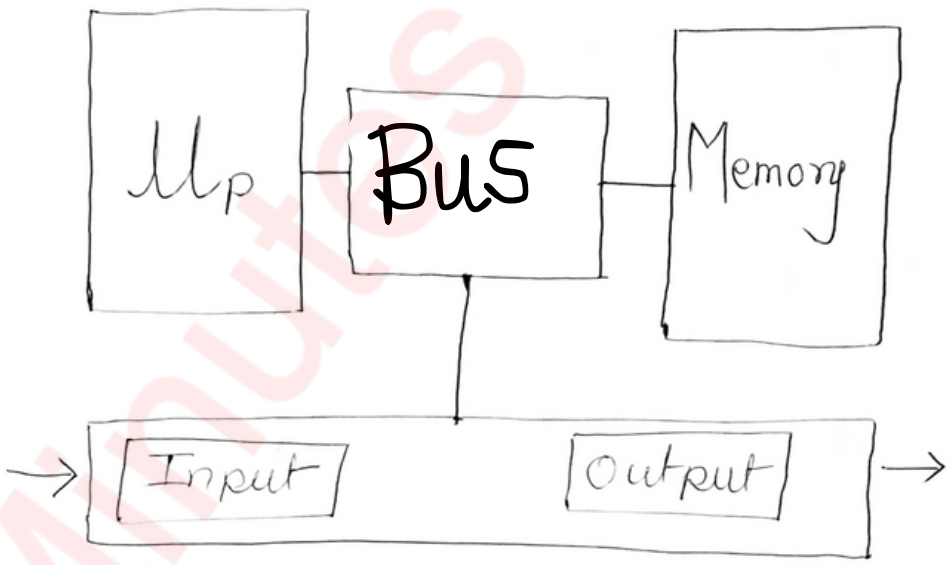
Computer — Organization — Architecture

"How" — "What"

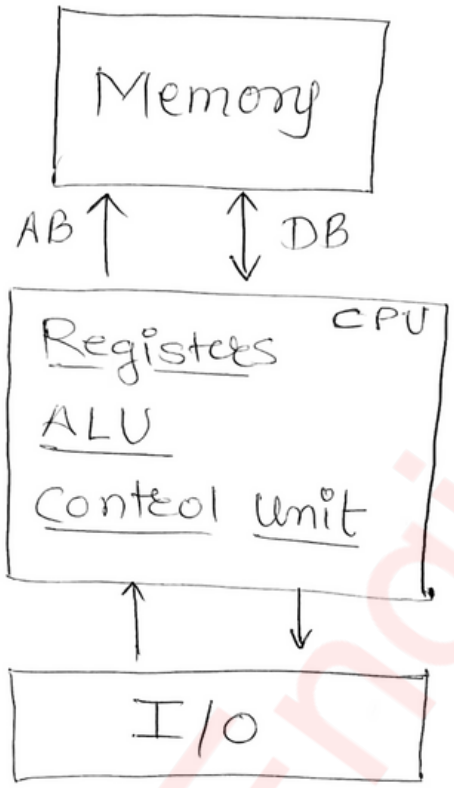
"Functional" ↔ "Structural"

"2nd" — "1st"

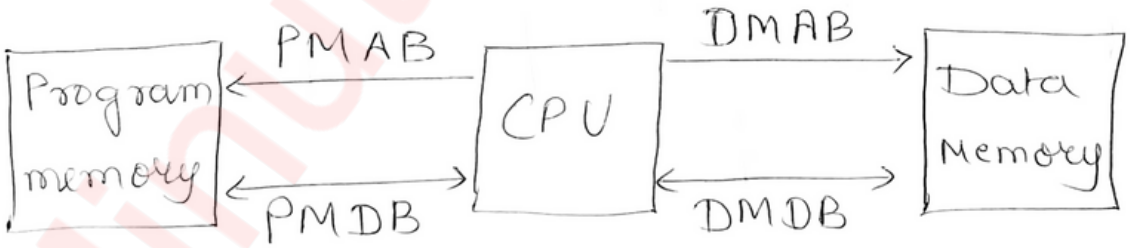
- Basic units of a computer:



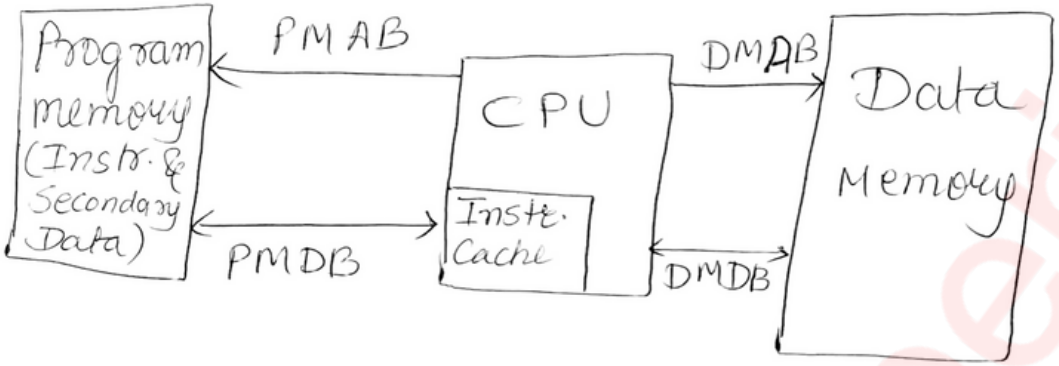
- Von Neumann's Architecture



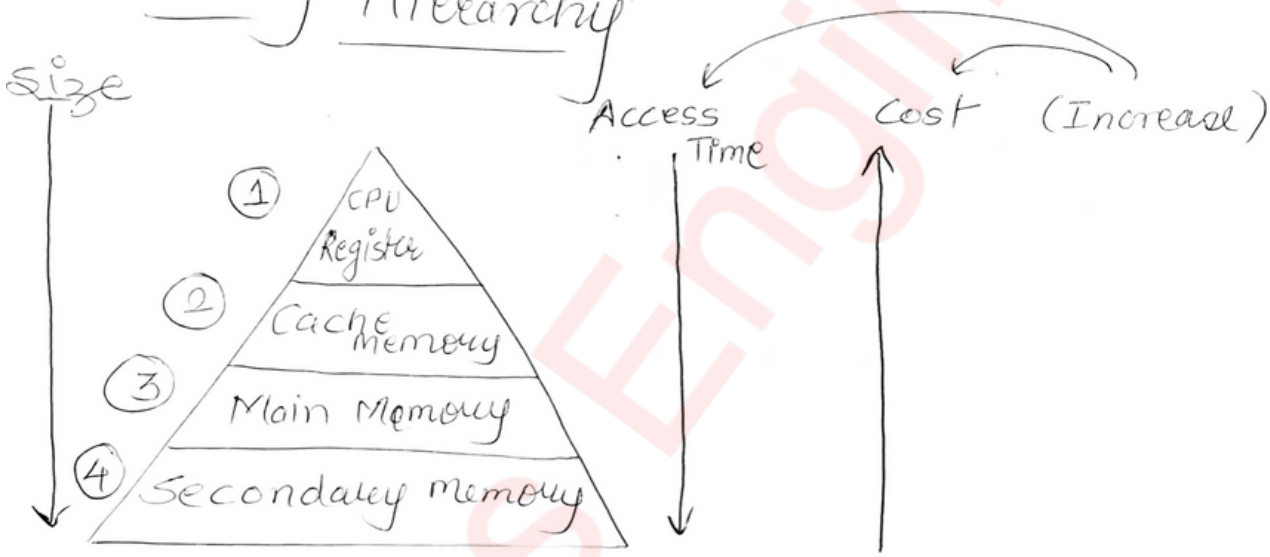
- Harvard Architecture



Super Harvard Architecture



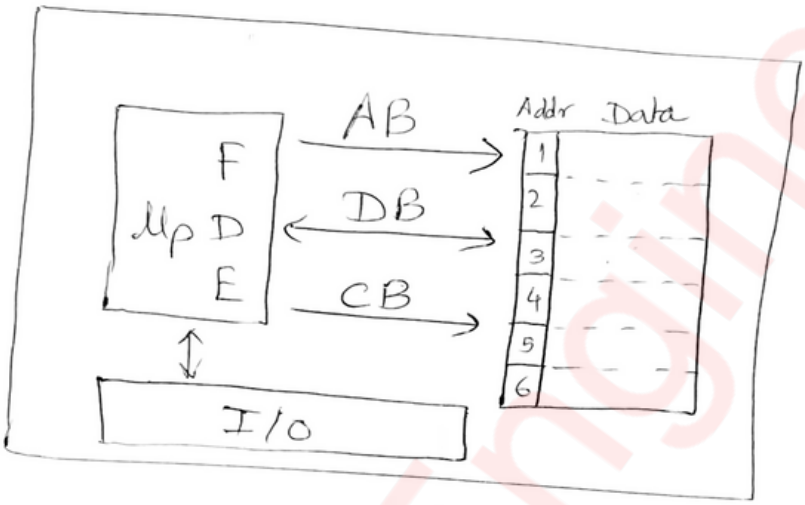
Memory Hierarchy

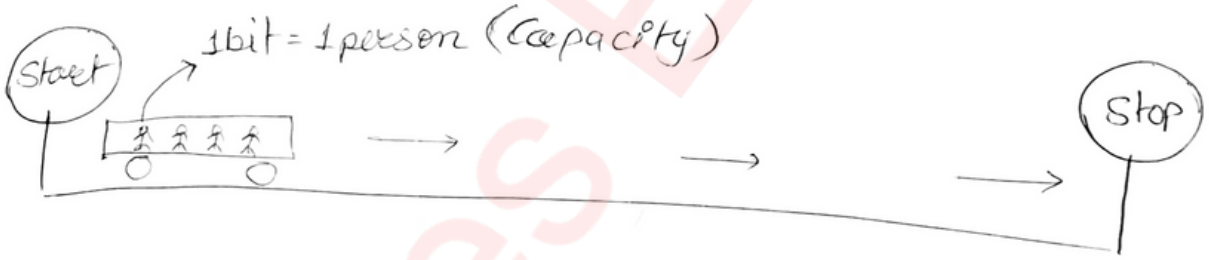


Registers

- Address
- Data
- Instruction
- Program Counter
- Accumulator
- Temporary
- Input / output

- System Buses
  - → Address
  - → Data
  - → Control





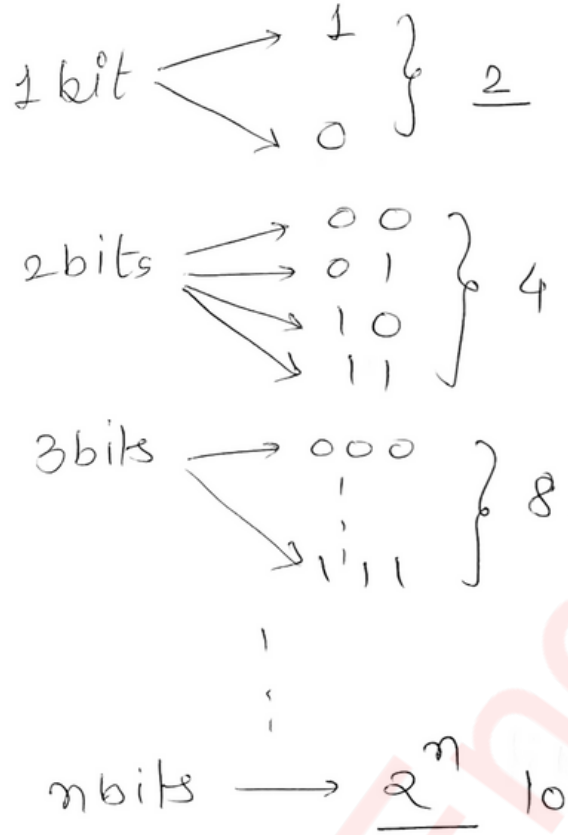
: Read

① Address Bus holds the Addr. of mem loc to read for. (Up → M)
② Control Bus gives read signal.
③ then that data is brought by (Up → M) (Up ↔ M) DB to Up.

: Write

⇒ Just a diff is CB gives write signal.

- Address Bus (unidirectional $\mu p \rightarrow M$)



$\Rightarrow$ 1 mem. location [1B data.]

So, total memory = 2^n B

for eg: AB = 16 bit for 8085 μp

locations = 2^{16}

Total memory = 2^{16} B

$\downarrow$

$2^{10} \times 2^6$ B

$\downarrow$

2^6 KB

$\downarrow$

64kB

Addr. Range

$\Rightarrow$ 0000 H

:

FFFF H

} 16bit

° Data bus (Bidirectional)

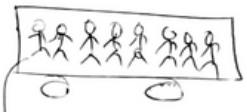
⇒ Read

Up ⟵ Memory

⇒ Write

Up ⟶ Memory

eg: 8085 (8 bit data bus)



1 person = 1 bit

8 bit data : 12 H

Range : 00 H

FF H

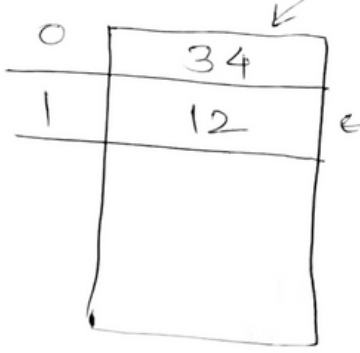
0001 0010

can be carried by DB at a time.

but incase data :

12 34 H

16 bit.



12 / H 34 / L

- Control Bus

$\mu p \longrightarrow$ Memory / I/O

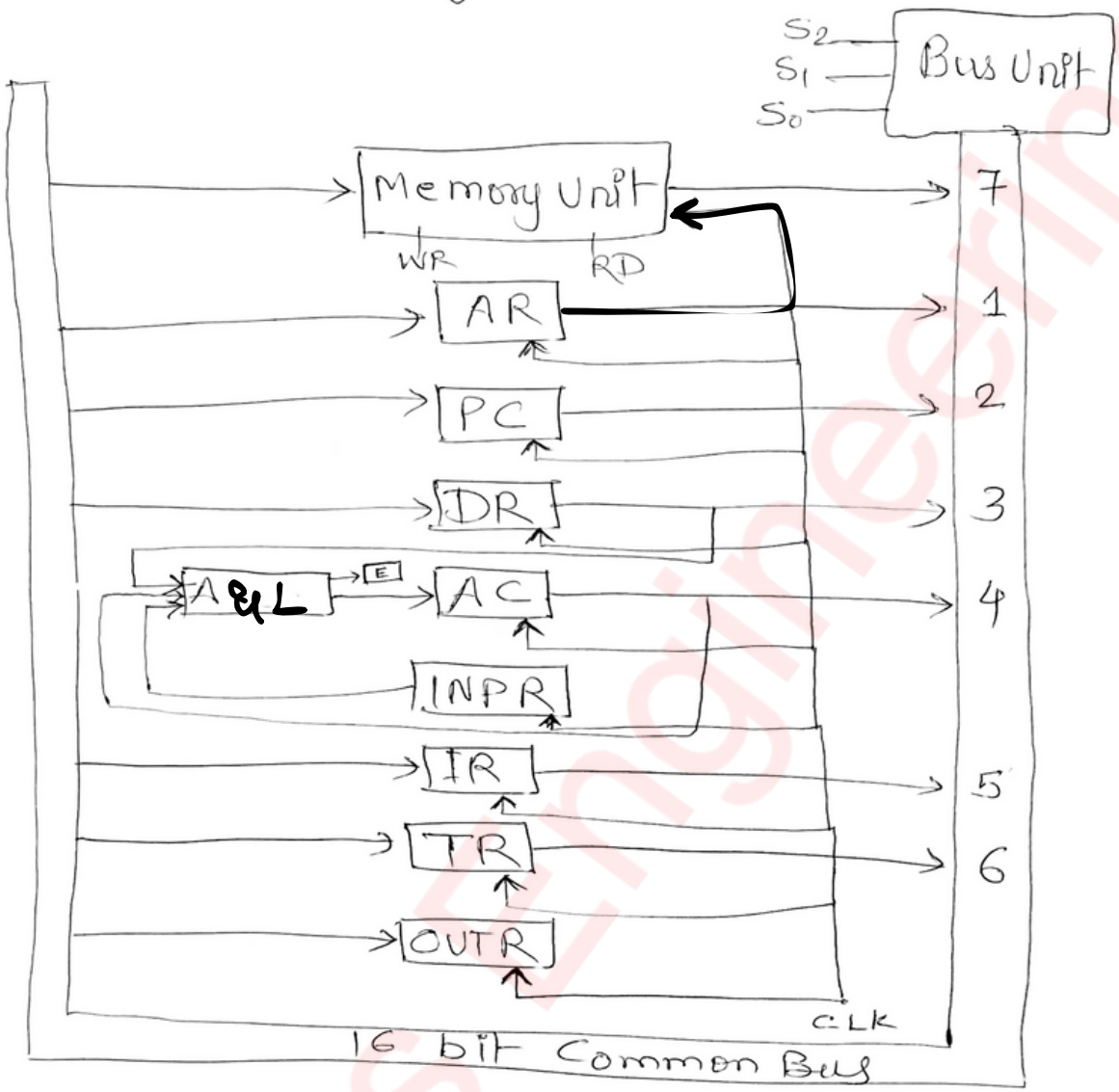
$IO/\overline{M}$ → 1 (I/O)
→ 0 (M)

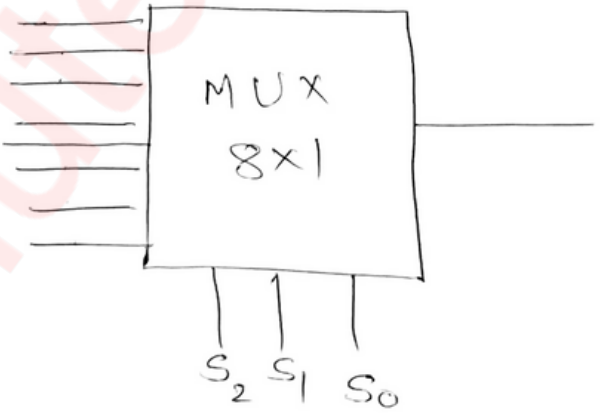
$\overline{RD}$ → 1 (write)
→ 0 (Read)

$\overline{WR}$ → 0 (write)
→ 1 (Read.

	$IO/\overline{M}$	$\overline{RD}$	$\overline{WR}$
MR	0	0	1
MW	0	1	0
IO R	1	0	1
IOW	1	1	0

- Common Bus System

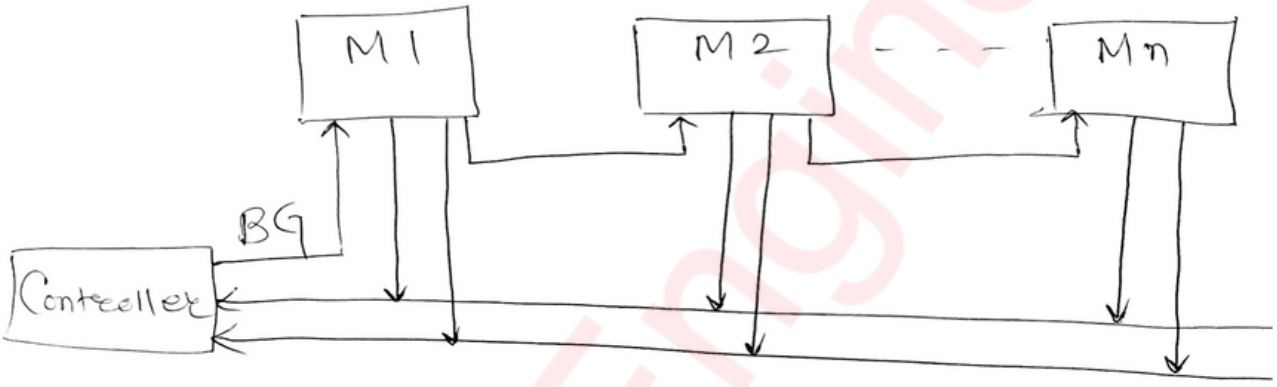




Bus Arbitration

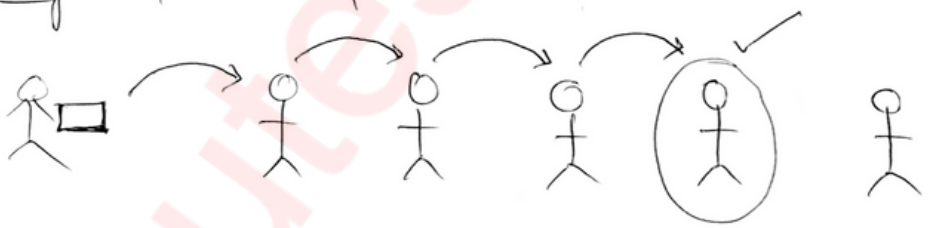
↳ Multiple Requests at the same time. without any malfunction.

→ Diasy chaining

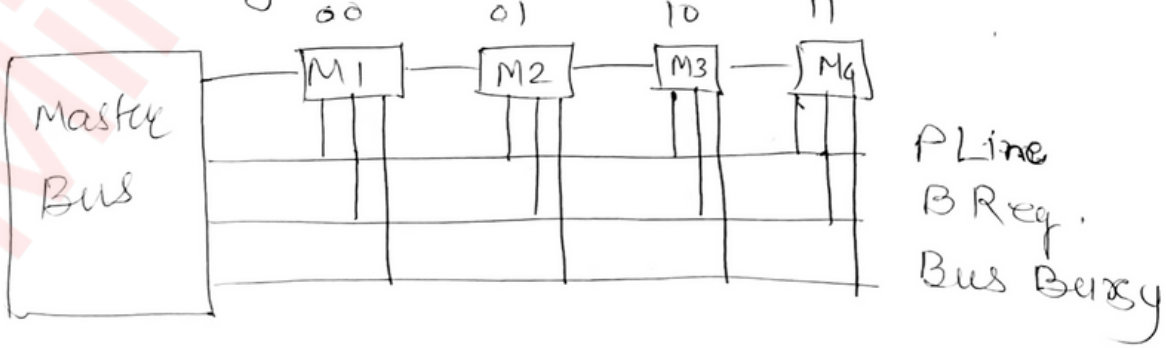


BG: Bus Grant signal has a serial propagation.

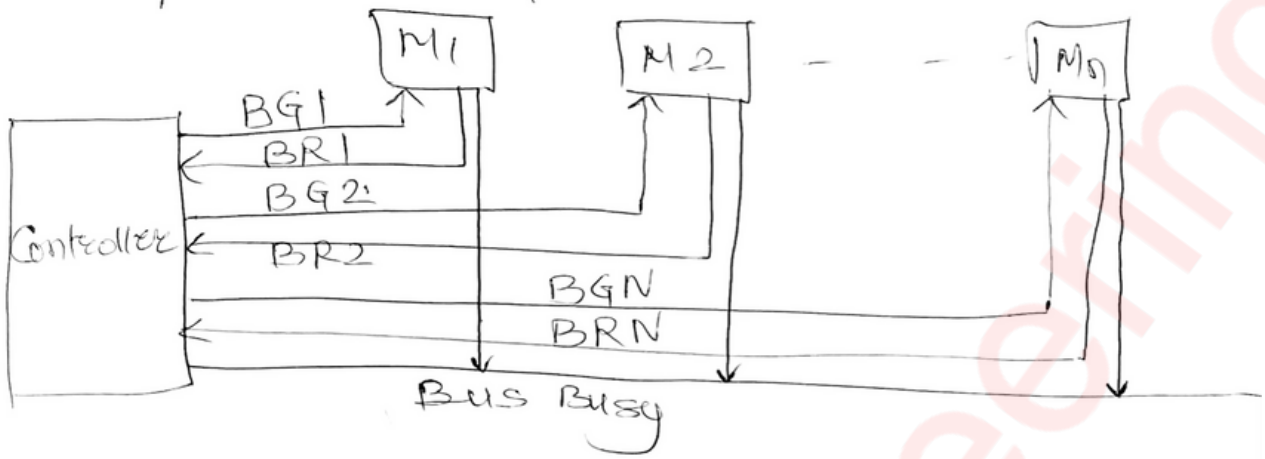
Eg: lost purse



→ Polling (can change Priority)



- Independent Request / device method.



(priority Based)

- Types of CPU organizations
  - Single AC organization
  - General Register organization
  - stack organization.

① Single AC
  - AC Register used for processing + storing Results Instructions
  - 1st operand is always in AC.
    eg: ADD B i.e AC + B
  - Result is stored in AC
    AC ← AC + B

- General Register

→ Use multiple General Purpose Registers

→ 2 or 3 addr. Instruction.

(eg:- ADD R1, A, B)

↑ destination, source

R1 ← A + B

→ Size increases (IR, DB)

Size of Instruction ↑ ↓ no. of Instruction.

eg:- A + B

3 addr : ADD R1, A, B

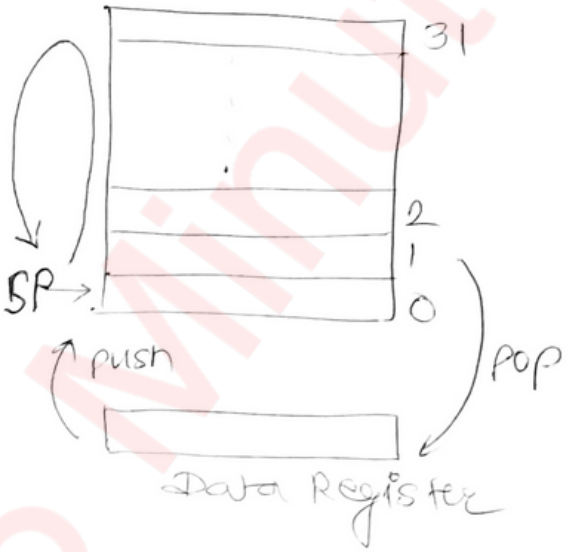
2 addr : MOV R1, A

ADD R1, B

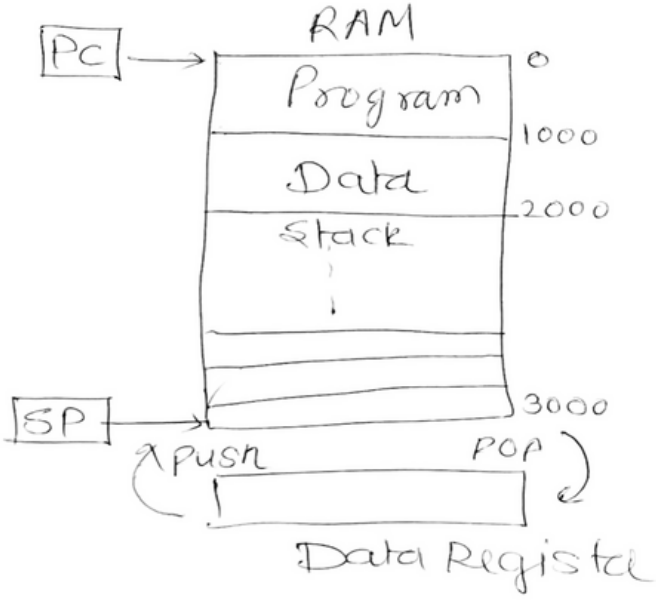
° Stack organization

- Stack structure
- LIFO / FILO
- SP : stack pointer holds the address of top of stack.
- Operations
  - PUSH : Insert on TOS
  - POP : Remove the TOS
- Types
  - Register stack
  - memory stack.

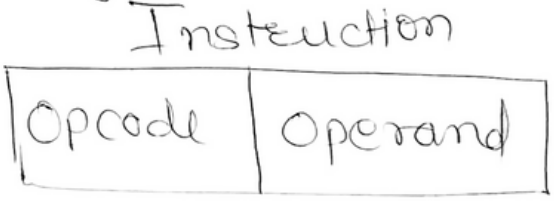
RS



MS



Addressing Modes



⟹ Ways in which one operand is specified.

Types:

① Immediate AM

- direct data is placed
- Constants. very fast
- No memory Refer. required.

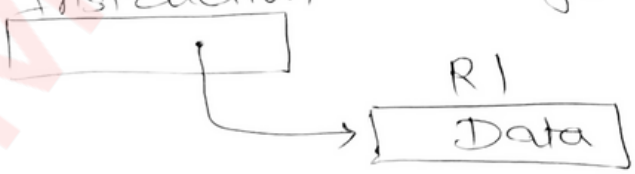
Eg: ADD 5H
MOV 50H

eg: in 8085

MVI D 50H } 2 Byte
LXI D 5000H } 3 Byte

② Register AM

- operand is placed in Registers

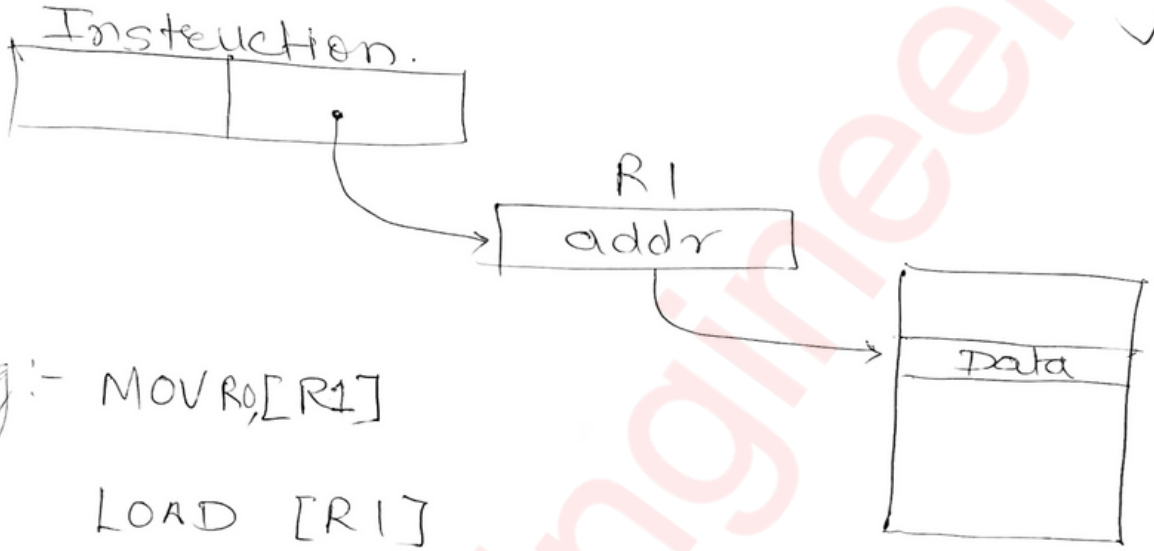


EA = Ri

eg: MOV R1, R2
LOAD R1

- Register Indirect Mode

  → Register holds the addr. of operand which is located in memory



eg:- MOV R0,[R1]

LOAD [R1]

ADD R1, [R2]

- Auto Increment / Decrement AM.

  → LOAD [R1]+

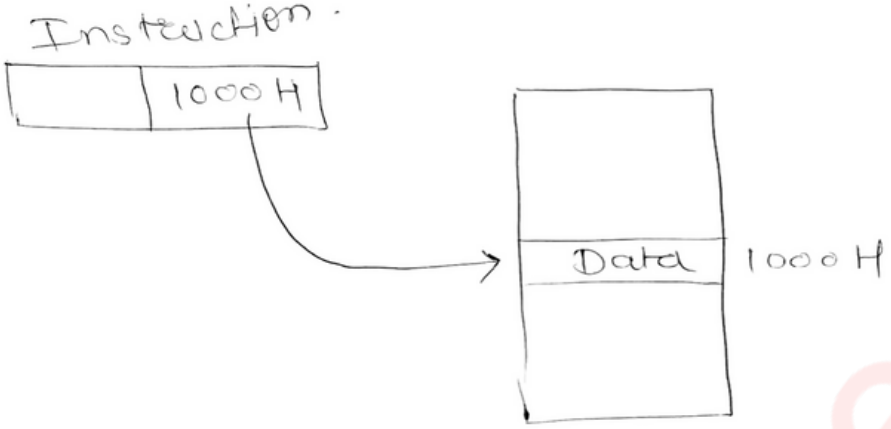
  AC ← M[(R1)]

  → Just mentioned starting address.

  eg: ADD R1, [R2]+

  R1 ← R1 + M([R2])
  R2 ← R2 + i

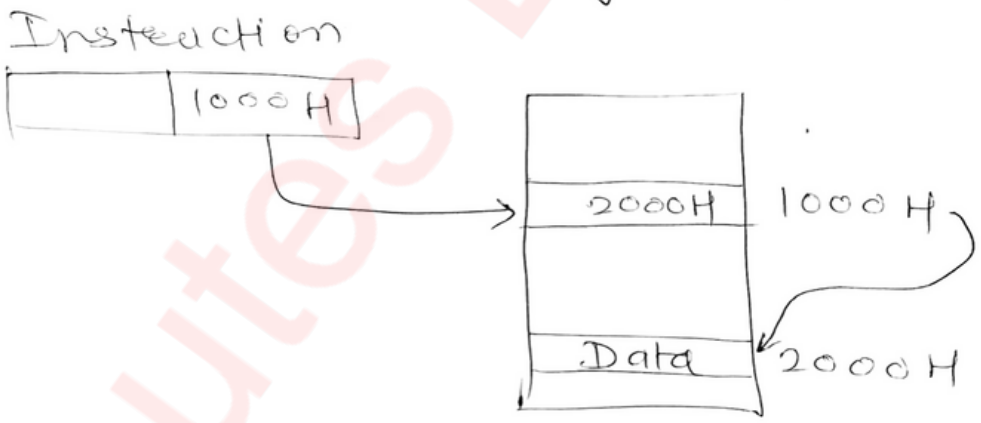
◦ Direct Addressing Mode



→ Not fixed / constant
→ for variables.

eg: ADD 1000H AC ← AC + M(1000H)
LOAD 1500H AC ← M(1500H)

◦ Indirect Addressing Mode

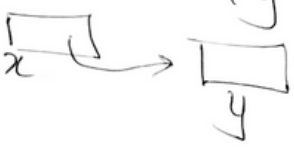


→ 2 memory access

eg: ADD X

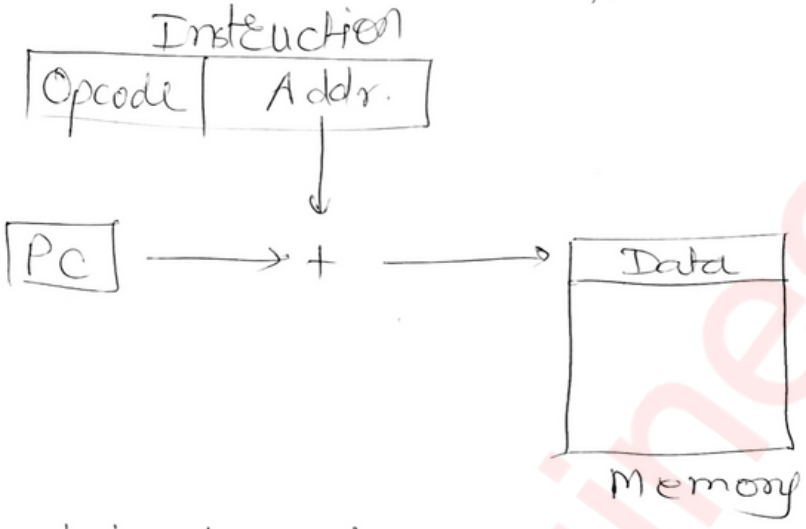
AC ← AC + [[X]]

→ Used in Pointers eg: *x = &y



◦ Relative AM

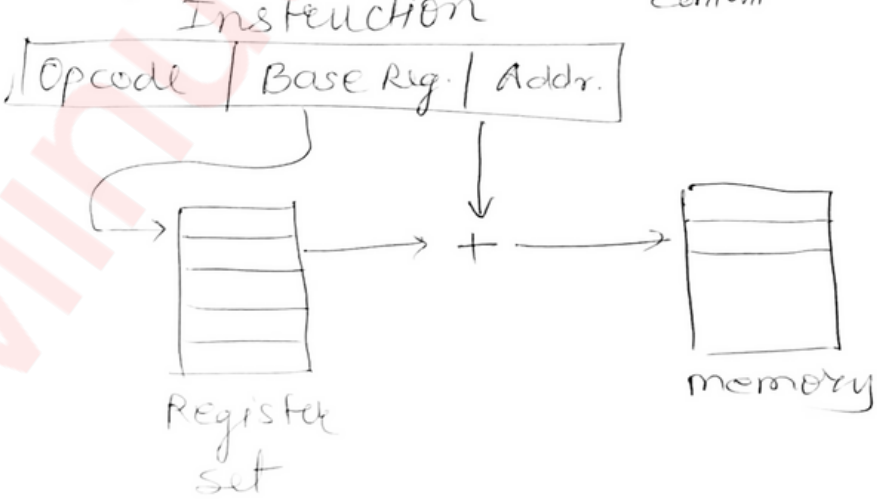
↳ EA = PC + Addr. in Inst. (offset)



⇒ Used in Jump / branch cases

◦ Base Register AM

EA = BR + Offset/Displacement
(Content)

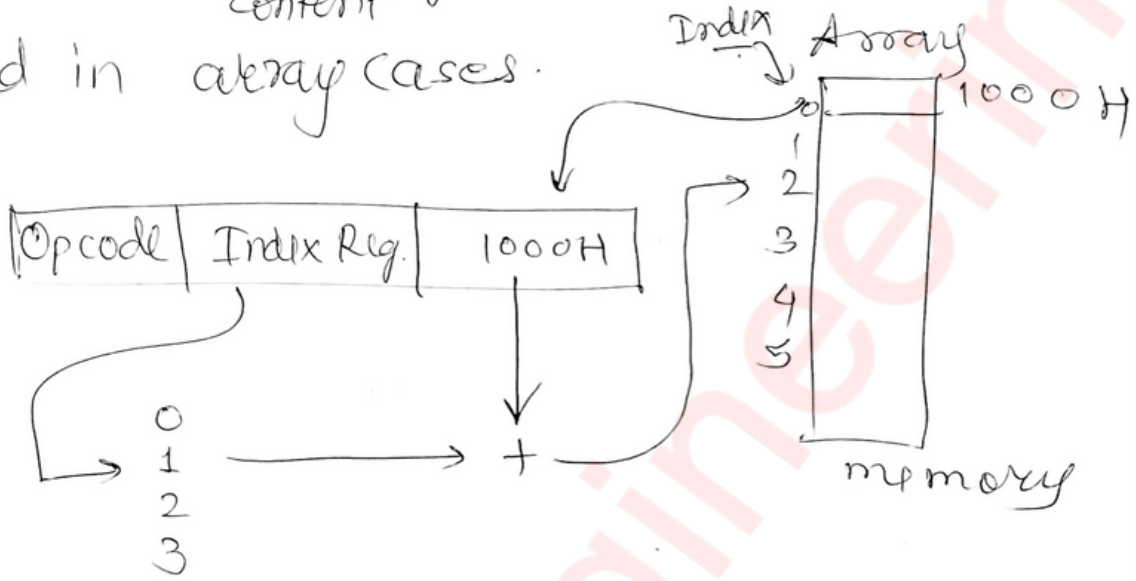


→ Program Relocation

◦ Indexed AM

EA = Index Rig. + Offset.
content

→ used in array cases.



◦ Implied AM

→ operands are implicitly defined.

→ पता है ही होते. पहले से.

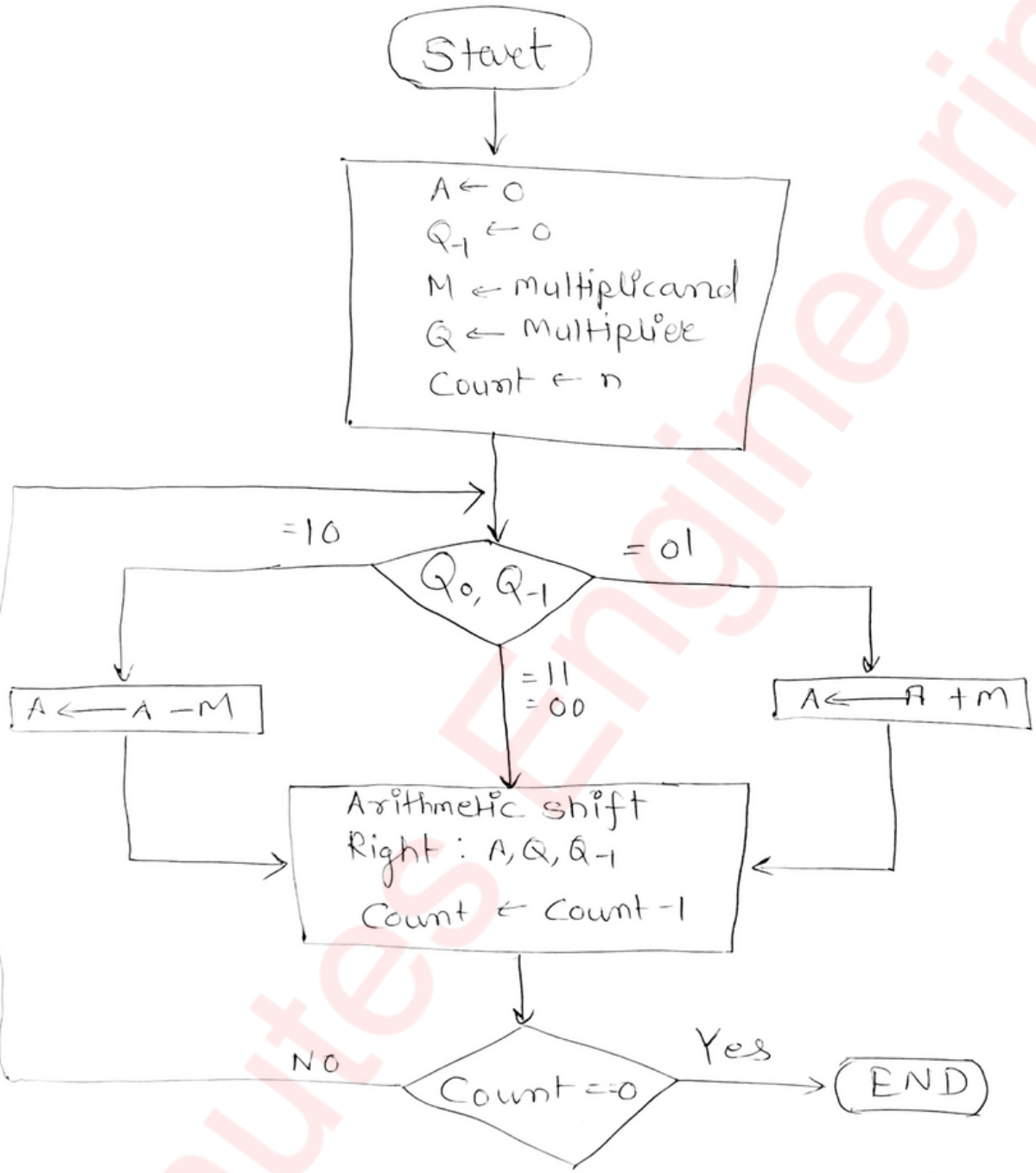
→ zero addr. Instructions

eg: stack operations

PUSH
POP
] on TOS of stack

Already know that.

Booth's Algorithm



A	Q	Q-1	M
0 0 0 0	0 0 1 0	0	0 1 0 1
0 0 0 0	0 0 0 1	0	0 1 0 1
+ 1 0 1 1			
1 0 1 1			
1 1 0 1	1 0 0 0	1	0 1 0 1
+ 0 1 0 1			
0 0 1 0			

(5 x 2)

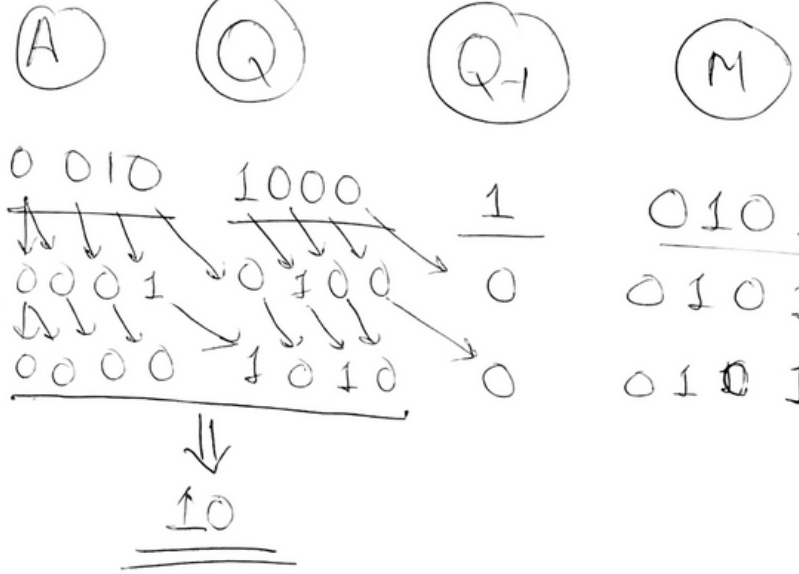
$Q_0, Q_{-1} = 00$

$Q_0, Q_{-1} = 10$

$(A \leftarrow A - M)$

↳ $A + (-M)$

$Q_0, Q_{-1} = 01$



- Array Multiplier

A1 A0
B1 B0
} 2 bit number

A1B0 A0B0
A1B1 A0B1

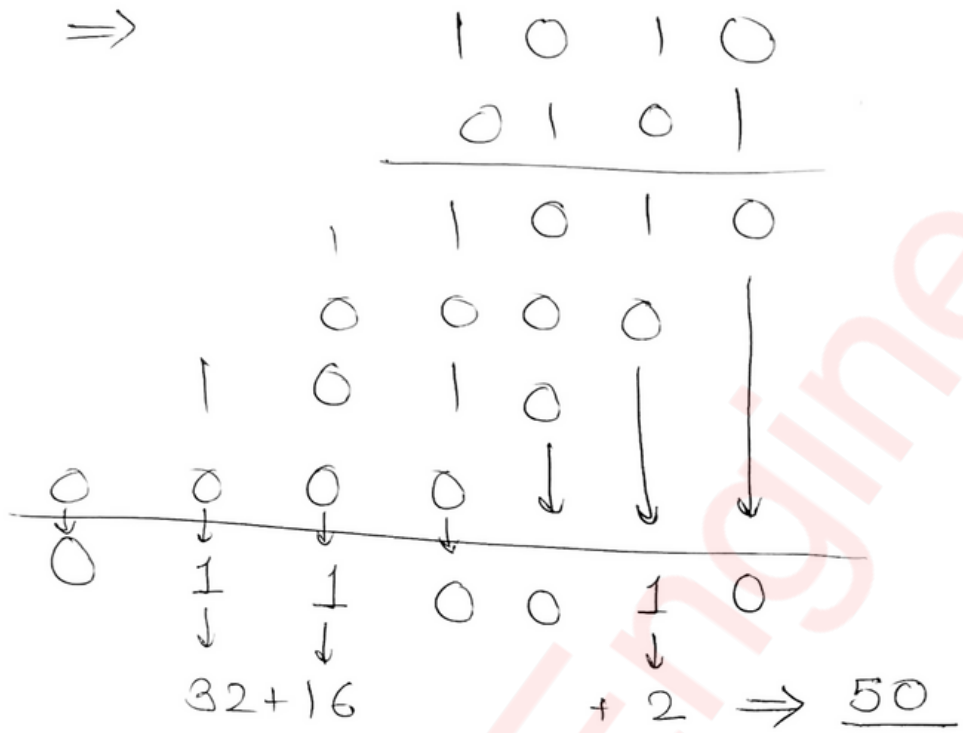
R3 R2 R1 R0

A = 10 (2)
B = 11 (3)

```
    1   1
    1   0
  -------
    0   0
1   1
-------
1   1   0
↓   ↓
4 + 2 = 6
```

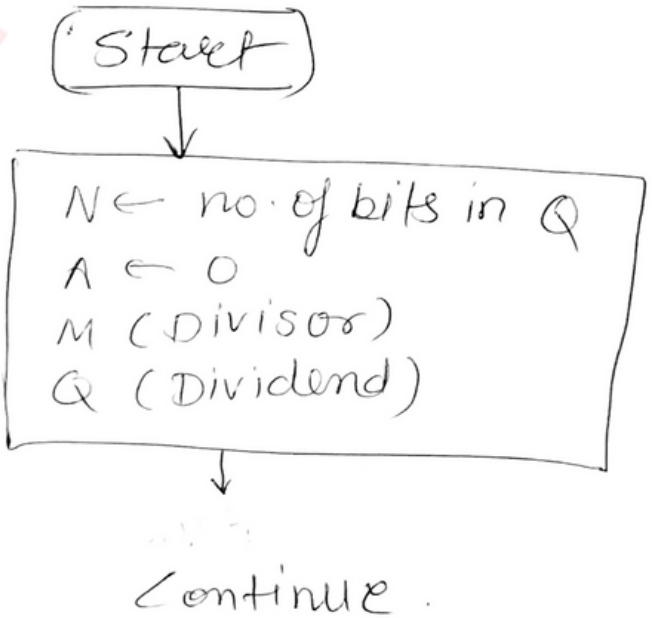
eg: A = 1010 (10)

B = 0101 (5)

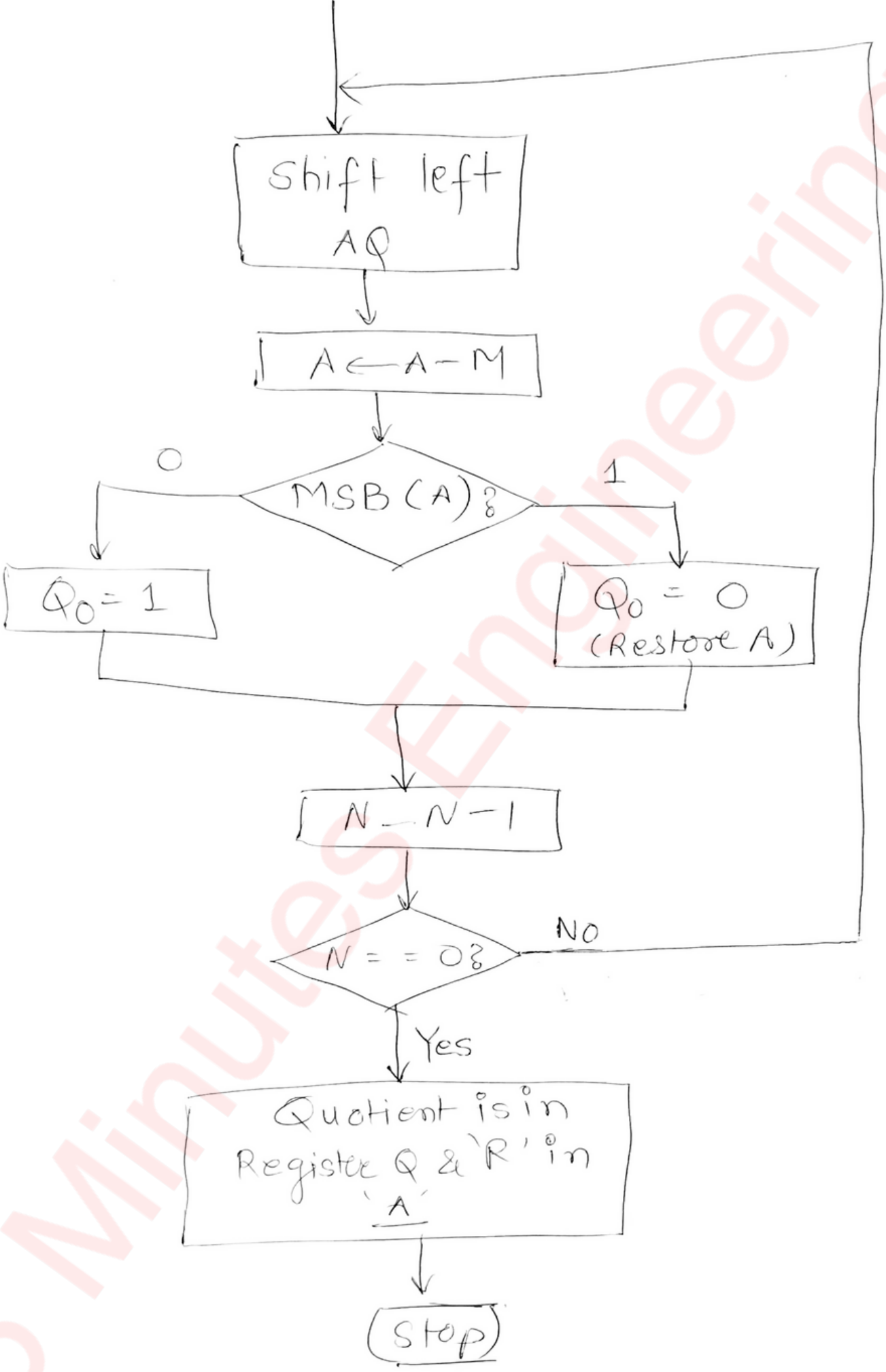


- Division Operation

⇒ Restoring Division Algorithm



Continue.



eg:- M (divisor) = 5 (0101)

Q (Dividend) = 10 (1010)

(-M)
1011

- for N = 4

A	Q
0000	1010
0001	010 x
+ 1011	
1100	

So, MSB(A) = 1 then x = 0
& Restore A.

A	Q
0001	0100

for N=3

A	Q
0001	0100
0010	100 x
+ 1011	
1101	

MSB(A) = 1 then x = 0 & Restore A.

A	Q
0010	1000

for N=2

A	Q
0 0 1 0	1 0 0 0
1 0 1 0 1	0 0 0 x
+ 1 0 1 1	
0 0 0 0	

$MSB(A) = 0 \quad \therefore Q_0 = 1$

A	Q
0 0 0 0	0 0 0 1

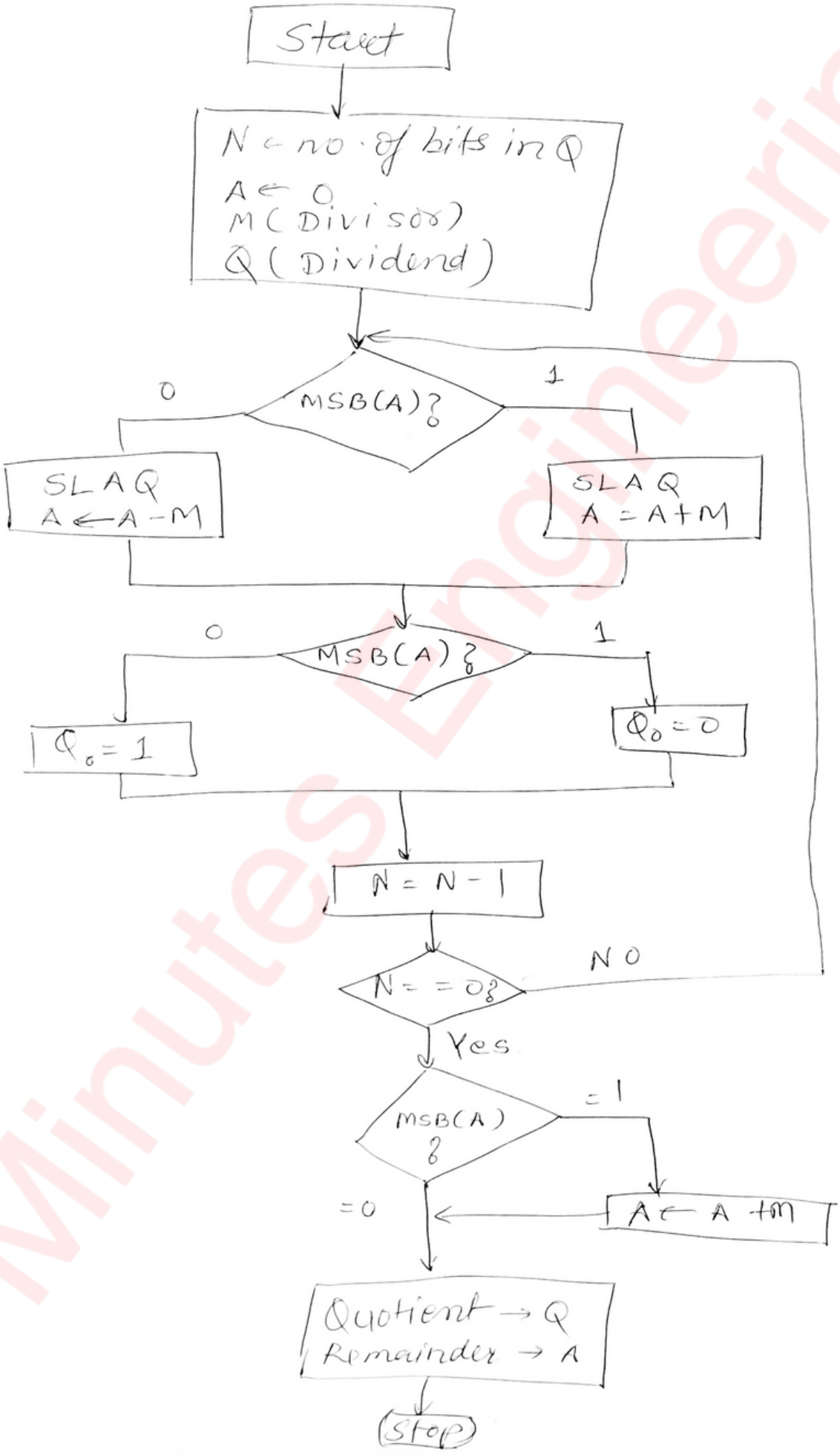
for N=1

A	Q
0 0 0 0	0 0 0 1
0 0 0 0	0 0 1 x
+ 1 0 1 1	
1 0 1 1	$\because Q_0 = 0$
MSB (A) = 1	
A	Q
0 0 0 0	0 0 1 0

for N=0

(0) Remainder Quotient (2)

Non-Restoring Division Algorithm



eg:- Q(Divident) = 3 (11)
M(Divisor) = 2 (10)

(-M)
1 0

for N = 2
⇒

A	Q
00	11

MSB(A) = 0
So, SLA Q
& A = A - M

A	Q
0 0	1 1
0 1	1 x
+ 1 0	↓ ↓
1 1	1 0

MSB(A) = 1 So

for N = 1

MSB(A) = 1

A	Q
1 1	1 0
1 1	0 x
+ 1 0	↓ ↓
0 1	0 1

MSB(A) = 0

for N = 0

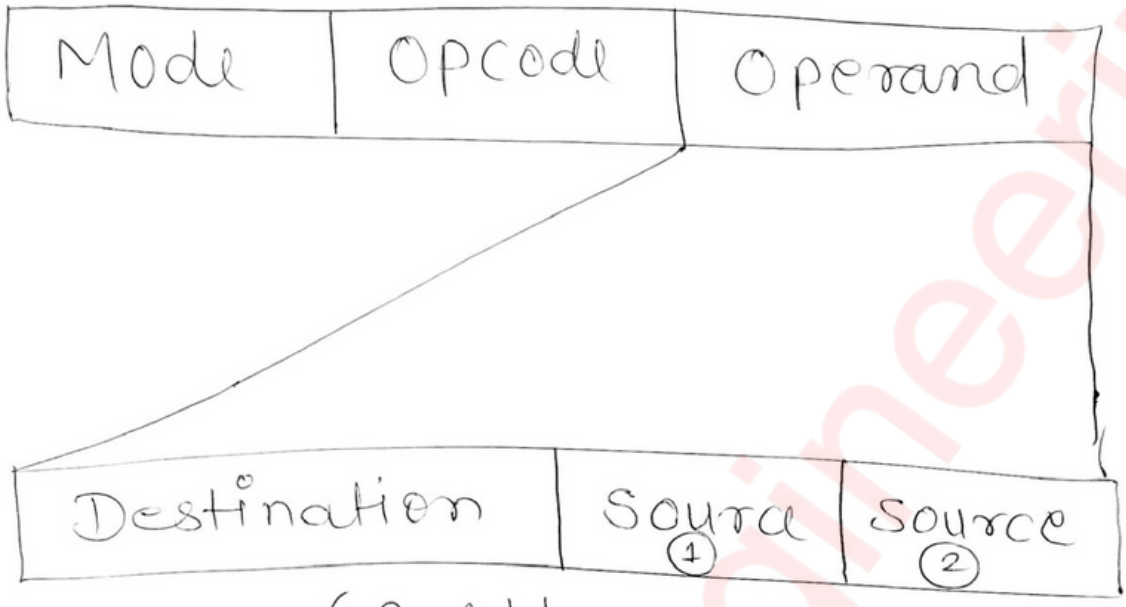
(msB(A)) = 0

Remainder = 01

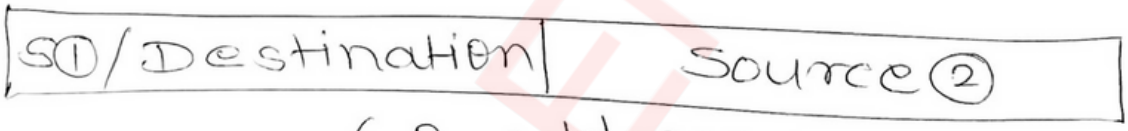
Quotient = 10

$$2\overline{)3}\ \begin{array}{r}1\\ 2\\ \hline 1\end{array}$$

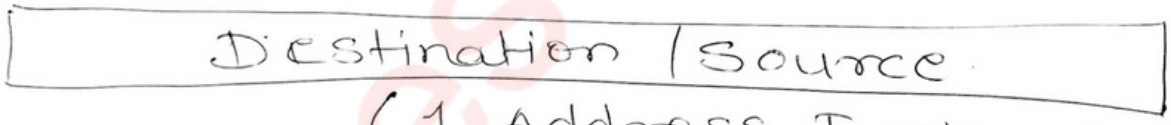
Instruction format



(3 Address Instruction)



(2 Address Instruction



(1 Address Instruction)

(0 Address Instruction)

Eg: $Z = (X + Y) \times (A + B)$

3 AI

```
Add R1, X, Y
Add R2, A, B
Mul Z, R1, R2
```

↓

(3)

2 AI

```
Mov R1, X
Add R1, Y
Mov R2, A
Add R2, B
Mul R1, R2
Mov Z, R1
```

↓

(6)

1 AI

```
LD X
Add Y
STR P
LD A
Add B
Mul P
STR Z
```

↓

(7)

0 AI

```
PUSH X
PUSH Y
Add
PUSH A
PUSH B
Add
Mul
POP Z
```

↓

(8)

⇒ Types of Instructions

- Data Transfer
- Arithmetic
- Logical
- Shift
- Control

⇒ Data Transfer:

- MOV
- LOAD (LDR)
- STORE (STR)
- INPUT (IN)
- OUTPUT (OUT)
- PUSH
- POP
- EXCHANGE (XCHG)

⇒ Arithmetic Instructions

- ADD
- SUB
- MUL
- DIV
- ADC
- SBB
- INC
- DEC

⇒ Logical Instructions

- AND
- OR
- NOT
- CLR
- XOR
- CLRC
- STC
- CMC
- TEST
- CMP

⇒ SHIFT Instructions

- SHL
- SHR
- ROL
- ROR
- SAL
- SAR
- RCL
- RCR

⇒ Control Instructions

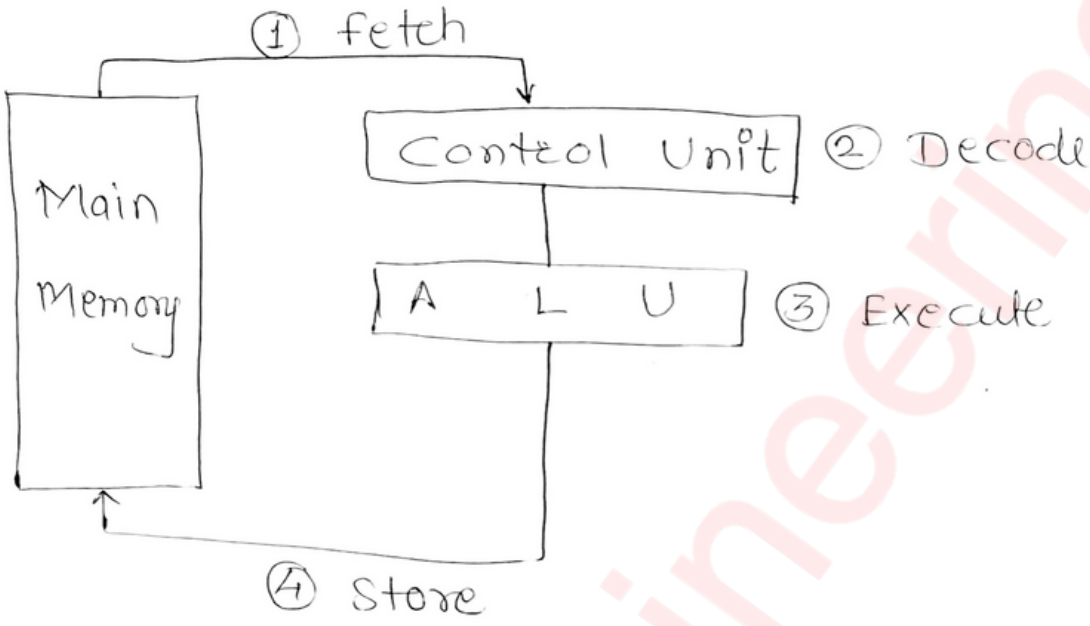
- NOP
- HLT
- EI
- DI
- SIM
- RIM

(NOP, HLT, EI, DI, SIM, RIM — Machine Control)

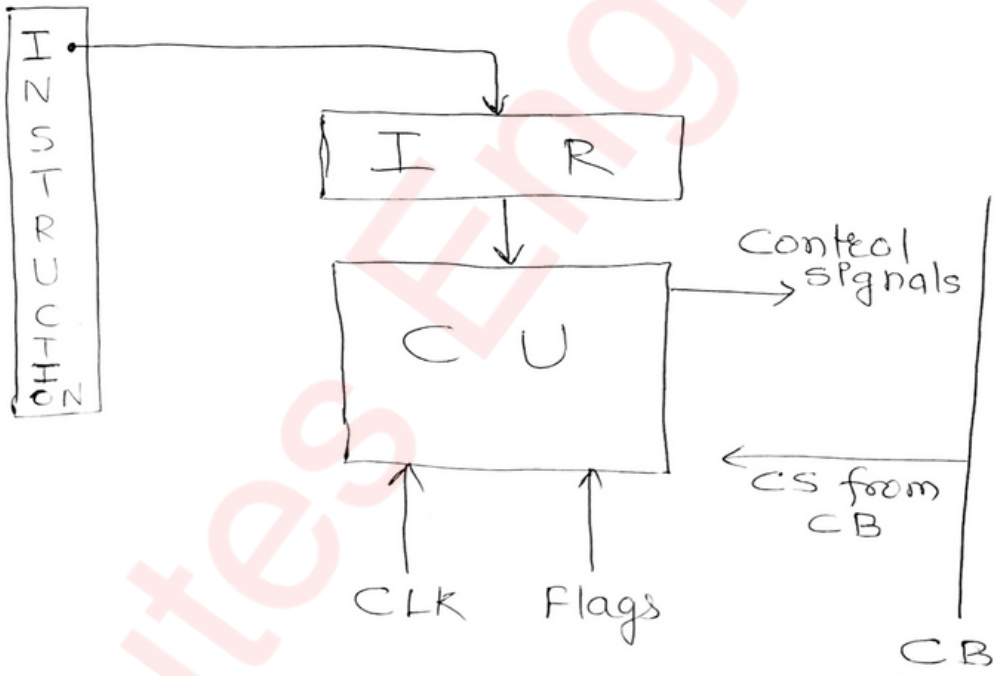
- JMP
- BR
- SKP
- CALL
- RET
- BNZ

(JMP, BR, SKP, CALL, RET, BNZ — Program Control)

• Control Unit

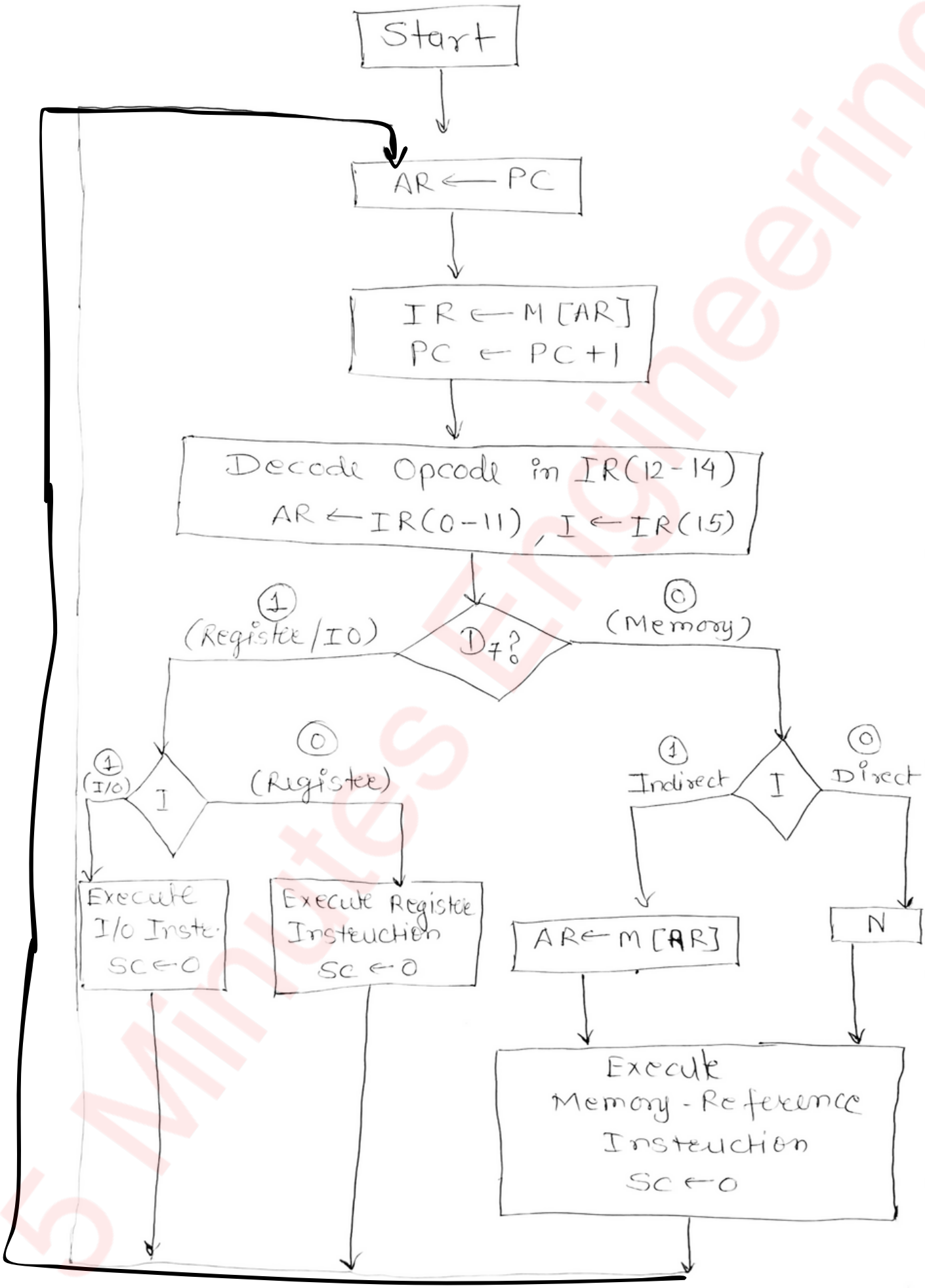


⇒



Types

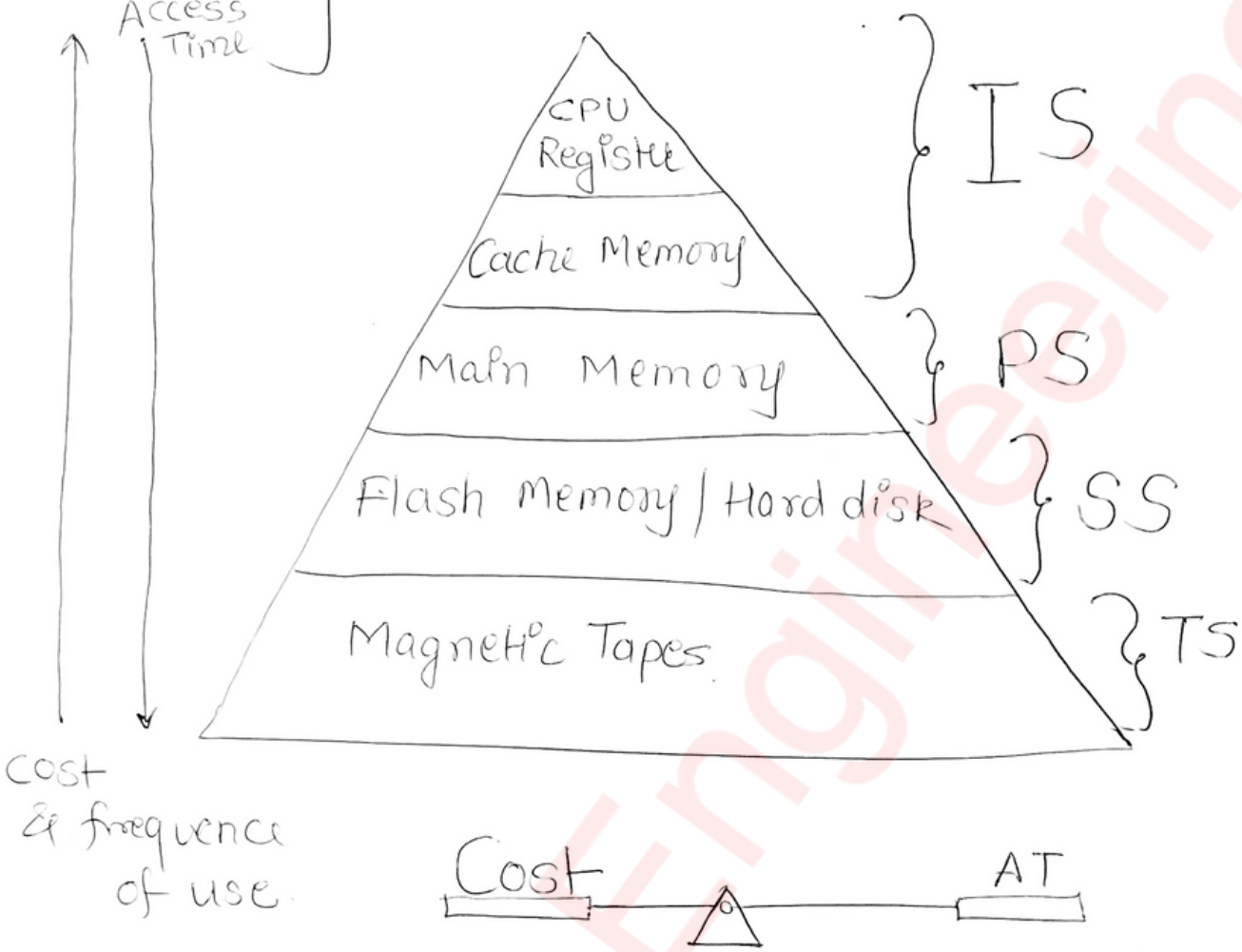
- Hardwired (Gates, FFs, decoders)
- Micro-Programmed (micro-instructions)



CISC Vs RISC

CISC	RISC
→ Complex ISC	→ Reduced ISC
→ Large Instruction set	→ Limited Number of Instructions
→ No. of Registers is very small	→ Large number of Register are Required.
→ S/w complexity ↓ Processor Architecture Complexity (↑)	→ S/w is complex Processor Architecture is simple
→ Pipelining is Difficult	→ Pipelining is Easy
→ Many Addressing Modes	→ few Addressing Modes
→ Instructions take more than 1 CLK.	→ A Instruction execute in 1 CLK.
→ Powerful	→ Relatively less powerful
→ Costly	→ Sasta hai.
→ Microprogrammed CU	→ Hardwired CU

◦ Memory



→ 2-Level Memory Organization

M1 Level 1

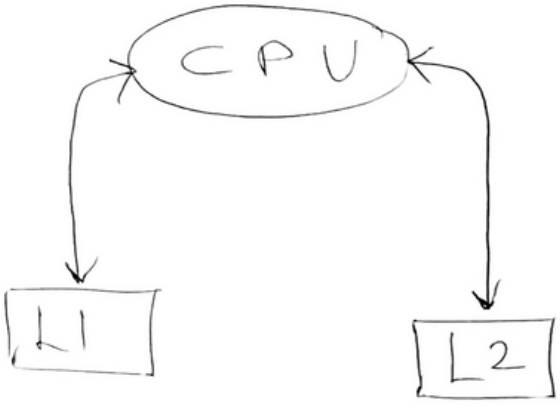
M2 Level 2

AT(L1) < AT(L2)

Cache MM

Size(L1) < Size(L2)

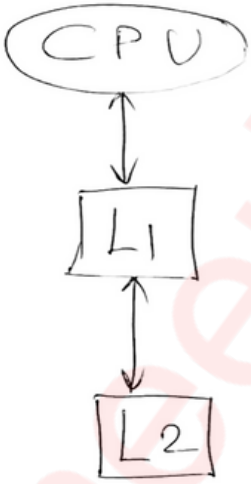
Independent



"Parallel"

$AT_{avg} = (H1)(AT1) + (1-H1)(AT2)$

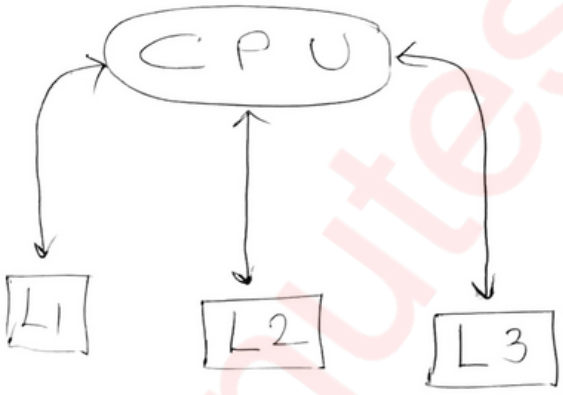
Heirarchical



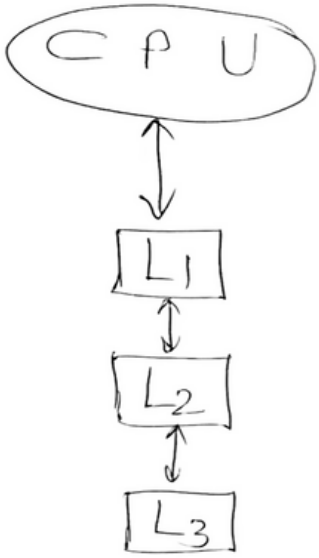
"Not Parallel"

$AT_{avg} = (H1)(AT1) + (1-H1)(AT1 + AT2)$

→ 3-Level memory Organization

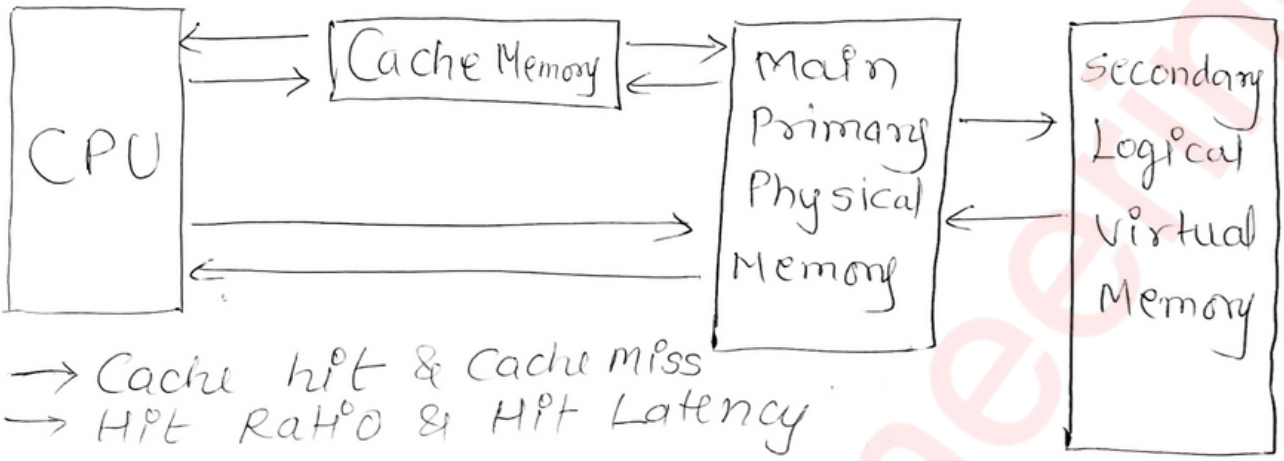


$T_{avg} = (H1)(T1) + (1-H1)(H2)(T2) + (1-H1)(1-H2)(T3)$



$T_{avg} = H1T1 + (1-H1)H_2(T1+T2) + (1-H1)(1-H_2)(T1+T2T3)$

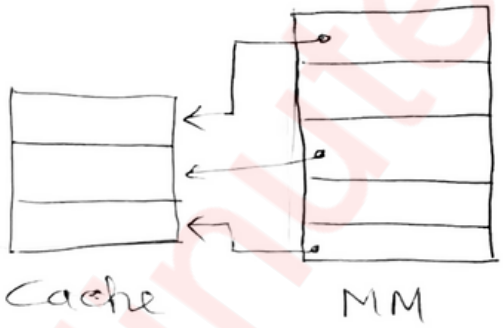
Cache Memory



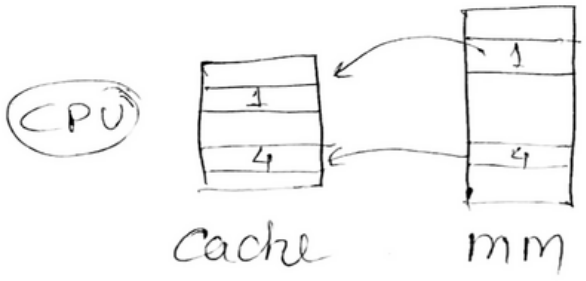
→ Cache hit & Cache miss
→ Hit Ratio & Hit Latency

- Locality of Reference (P) can access same MMB again & again
  - → Temporal Locality (Time)
  - → Spacial Locality (Space)

S L [Close Proximity]

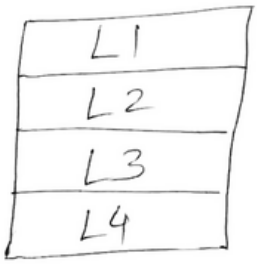


T L

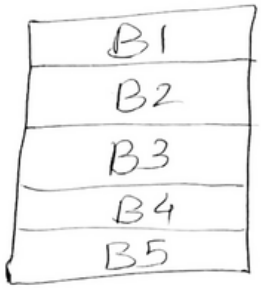


LRU

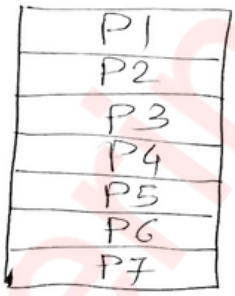
Basic Terms



Lines
(cache memory)



Blocks
(MM)



Pages
(SM)

Size of
(Line) = (Block) = (Page)

2^{10} → Kilo → 10^3

2^{20} → Mega → 10^6

2^{30} → Gega → 10^9

2^{40} → Tera → 10^{12}

10^{15} (Peta)

1 bit → 1, 0 } 2 location

n bit → 2^n location

2 bit → 00, 01, 10, 11 } 4 location

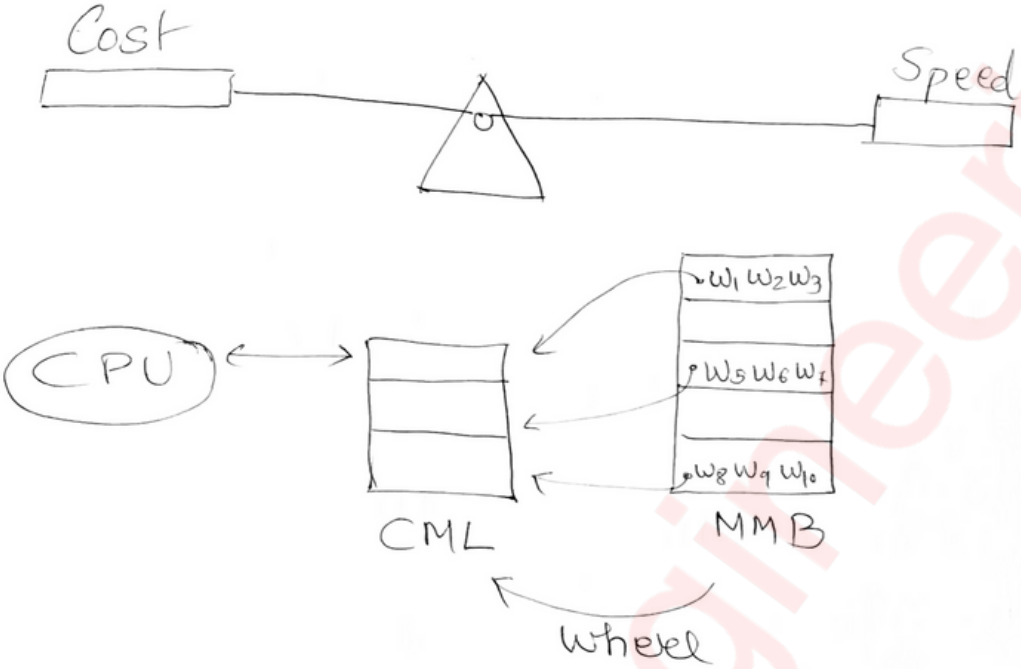
1 Byte | 1 Location

So, 2^n Bytes

3 bit → 000 ... 111 } 8 location

DRAM	Vs	SRAM
→ Dynamic		→ Static
→ Slower		→ Faster
→ Use of Capacitor		→ Use of Flip Flop.
→ Sasta hai		→ Mehnga hai
→ Refresh required		→ No need of Refresh.
→ Low Power Consumption		→ High Power consumption
→ size: larger		→ size: smaller.
→ Main Memory		→ Cache memory

• Cache Mapping



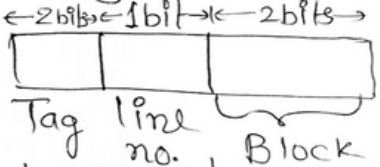
① Direct Mapping (a Block has only 1 Line in cache)

B0	0	1	2	3
B1	4	5	6	7
B2	8	9	10	11
B3	12	13	14	15
B4	16	17	18	19
B5	20	21	22	23
B6	24	25	26	27
B7	28	29	30	31

(32 words)

L0 → B0, 2, 4, 6
L1 → B1, 3, 5 7

(5 bits)
Physical Addr.



0 → 0 0 0 → L0
1 → 0 0 1
2 → 0 1 0 → L0
3 → 0 1 1
4 → 1 0 0 → L0
5 → 1 0 1
6 → 1 1 0 → L0
7 → 1 1 1

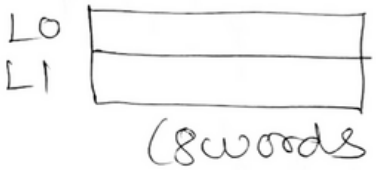
No. of Blocks
→ $\frac{32}{4}$ = 8 Blocks

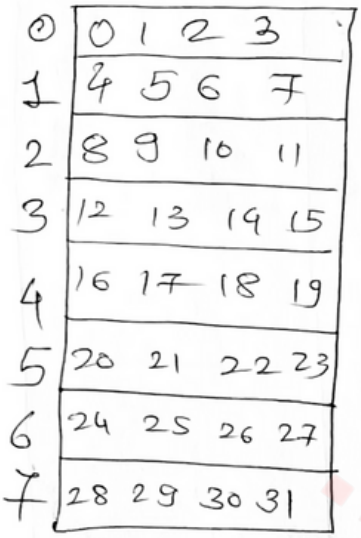
No. of lines
→ $\frac{8}{4}$ = 2 Lines

Block size = Line size = 4 words

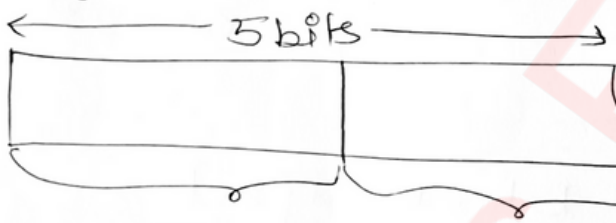
Cache Line number = (MM Block) Modulo (No. of lines in cache)

② Associative Mapping (Any Block can map to any Line)

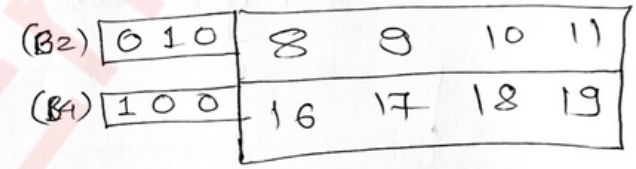


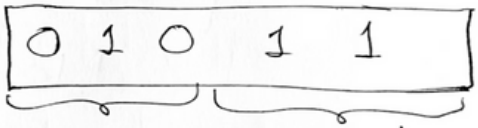


Physical Addr



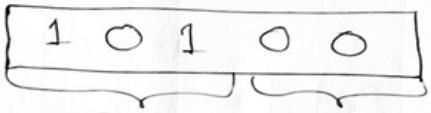
Total 8 blocks ③





B2 — 3rd word

(✓) cache hit



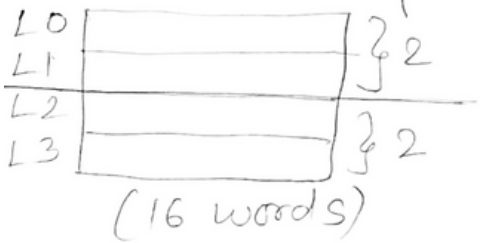
B5 — 0th word

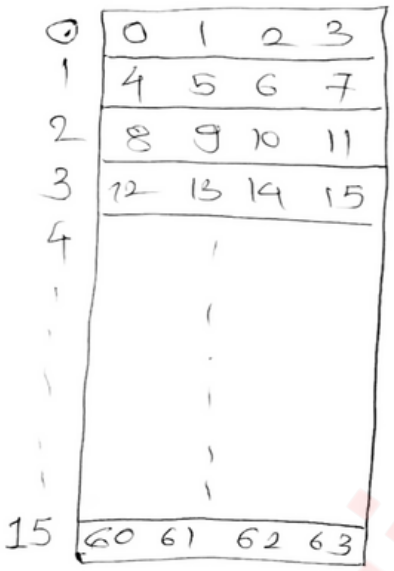
(✗) Cache Miss

* More comparisions Required (Total → lines)

③ Set Associative Mapping (Block is mapped to a (set))

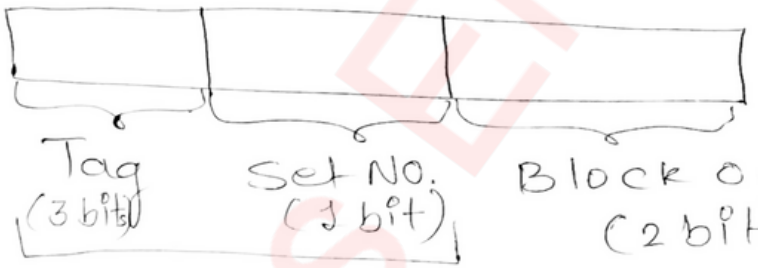
2 way set Associative

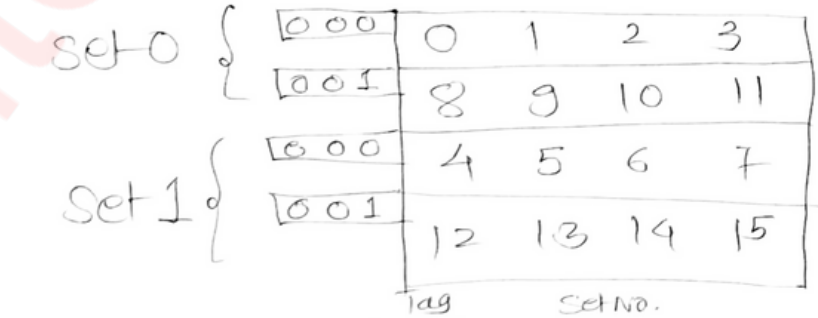


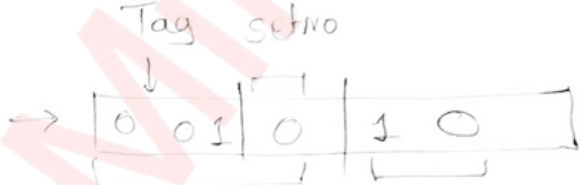


0 mod 2 = 0
1 mod 2 = 1
2 mod 2 = 0
3 mod 2 = 1
4 mod 2 = 0
15 mod 2 = 1
↑ Set No.

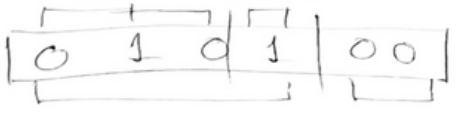
Physical Addr. (6 bits) ← $64 = 2^6$





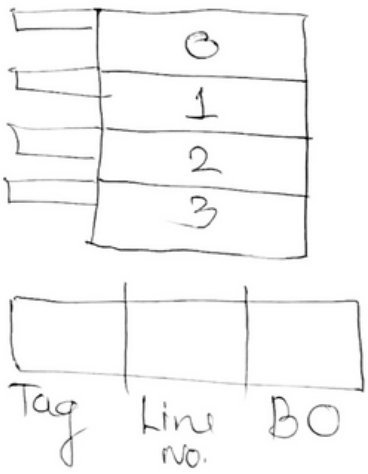


(Cache hit)



(Cache Miss)

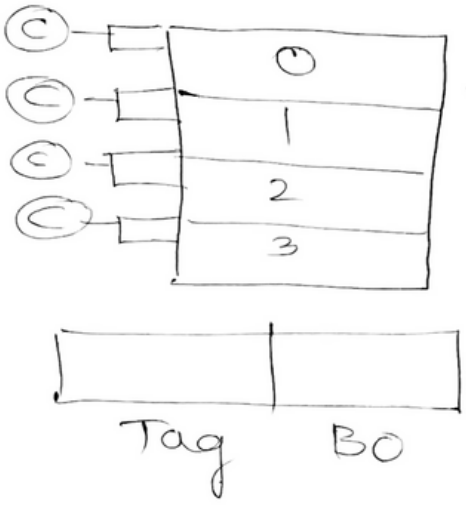
Direct



(1)

Comparator

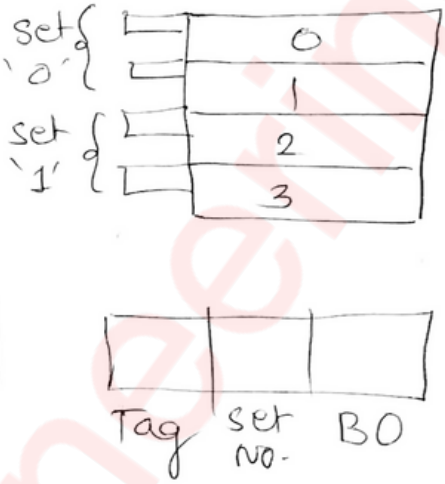
Associative



(N)

Comparators

set Associative



(2)

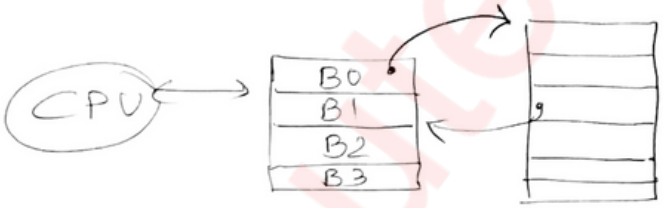
Comparators

2 way set-A

k way set-A

(k)

- Cache Replacement (Reduce Miss) or (Increase Hit)



→ Used for Associative & set Associative

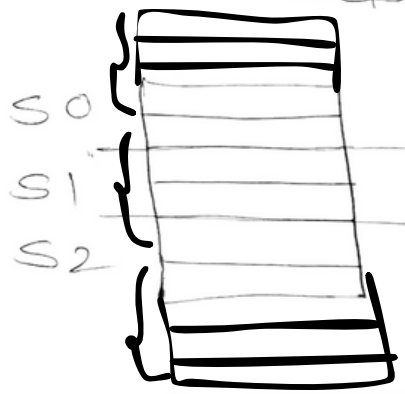
→ FIFO

→ LRU (Least Recently Used)

→ MRU (Most Recently Used).

① FIFO : First In First Out.

Eg: • 4 way set Associative
• 12 cache blocks
• No. sets = $\frac{12}{4} = 3$.



(Bno.) mod (no. of set).

S0	0, 6, 3, 9
S1	7, 10, 16, 19
S2	11, 14, 17, 20

Req: 0, 6, 7, 3, 9, 10, (7) Hits, 11, 14, 17, 20, 16, 19, (20) Hit, 5, 8, 13

5 mod 3 → S2 (FIFO) → ~~11~~, 14, ~~17~~, 20

13 mod 3 → S1 → ~~7~~, 10, 16, 19 → 10, 16, 19, 13

8 mod 3 → S2 → ~~14~~, 17, 20, 5 → 17, 20, 5, 8

② LRU (Least Recently Used)

Eg:- • 4 way set Associative
• 12 cache blocks
• No. of sets = $\frac{12}{4} = 3$

S0	0, 6, 3 9
S1	~~7~~, 10, 16, 19, 7
S2	~~11~~, 14, 17, 20, 11

Req:- 0, 6, 7, 3, 9, 10, 7, 11, 14, 17, 20, 16, 19, 20, 11, 5, 8, 13

5 mod 3 → S2
~~14~~ is LRU
⇓
17, 20, 11, 5

8 mod 3 → S2
17 is LRU
⇓
20, 11, 5, 8

13 mod 3 → S1
10 is LRU
⇓
16, 19, 7, 13

③ In MRU

5 mod 3 → S2
11 is MRU
14, 17, 20, 5

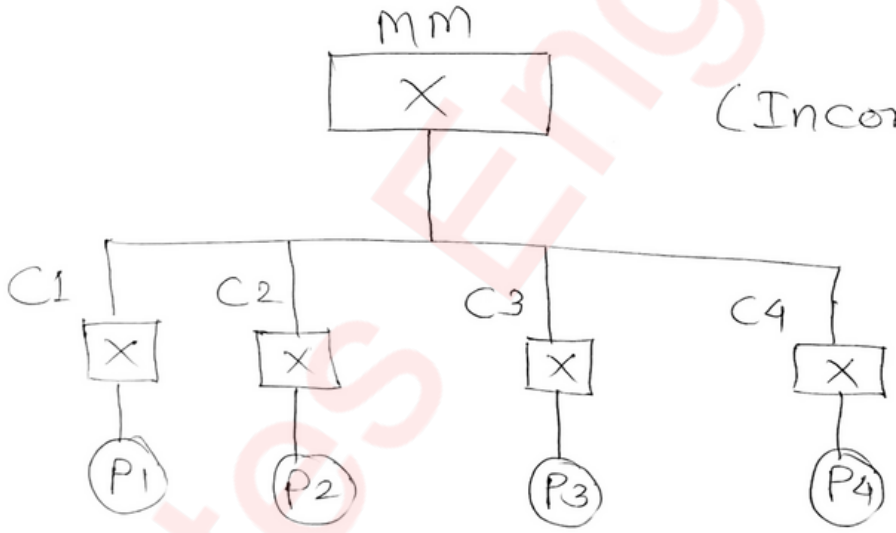
8 mod 3 → S2
5 is MRU
14, 17, 20, 8

13 mod 3 → S1
7 is MRU
10, 16, 19, 13

Miss

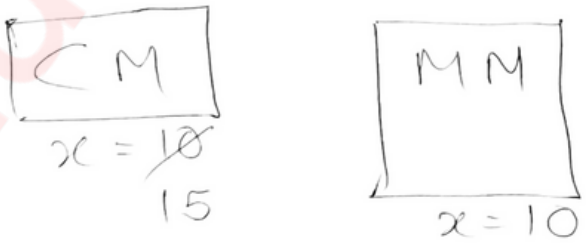
- **Compulsory** (0, 6, 7, 3, 9--)
- **Capacity**
- **Conflict**
  - Direct & set Associatin
  - ↳ many blocks mappid to a set / Line.

• Cache Coherence Problem



(Inconsistency)

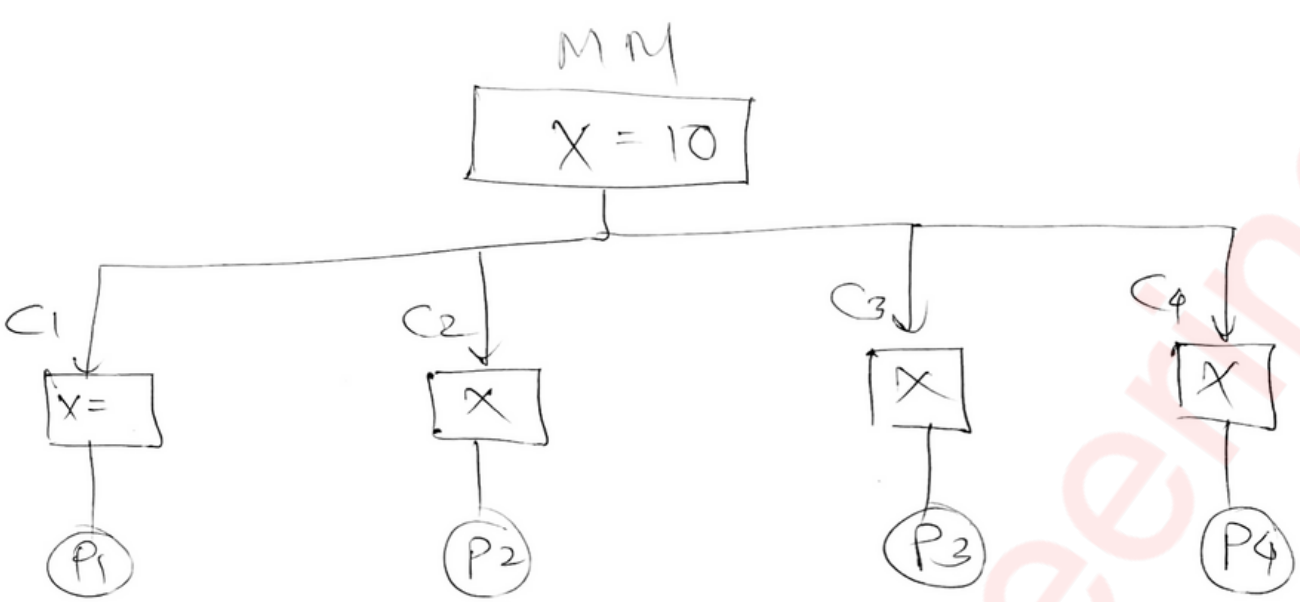
→



Solutions:

- WU - WT
- WU - WB
- WI - WT
- WI - WB

U ↓ Update
I ↓ Invalidate
T ↓ Through
B ↓ Back



WO, X = 11

P1 → update on all.

WT, X = 11

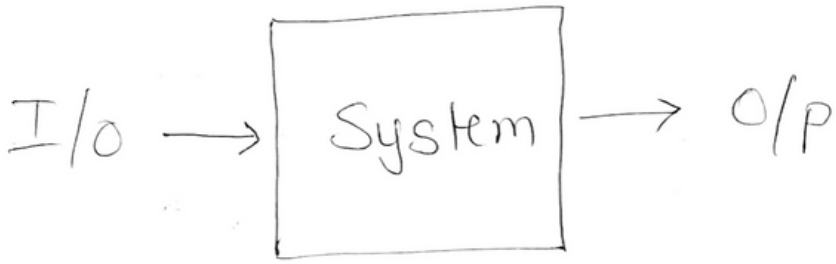
P1 → Update on MM as well.

WB, X = ~~11~~ ~~12~~ ~~13~~ ~~14~~ 15 → update it on MM as well

P1 after a Op/Job completes

WI, X = ~~11~~ 12 → Propagate message that X = 12 is Invalid value.

Input / Output (peripheral devices)



But → I/O devices cannot be directly connected to ip/CPU.

Speed difference

CPU ↑ I/O ↓

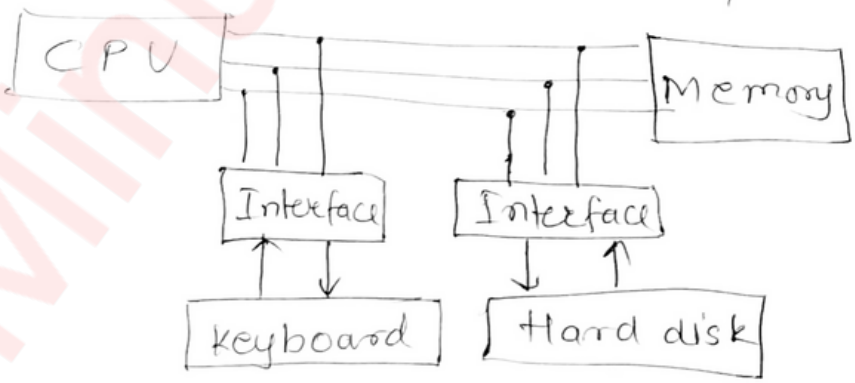
Nature I/O
- magnetic
- Optical
- EC

(heterogenous)

So, Interface is Required

↓

Transfer of info/data/signal bern internals & I/O.



→ Synchronization of operating speed of CPU w.r.t I/O devices.

Interrupt

(watching this video & you get a call in between)

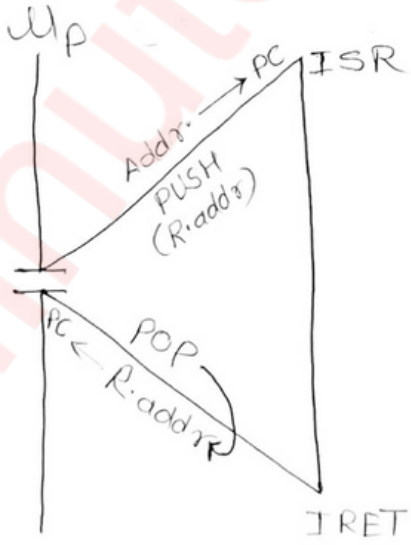
→ S/W : Instruction / System call / exception / error

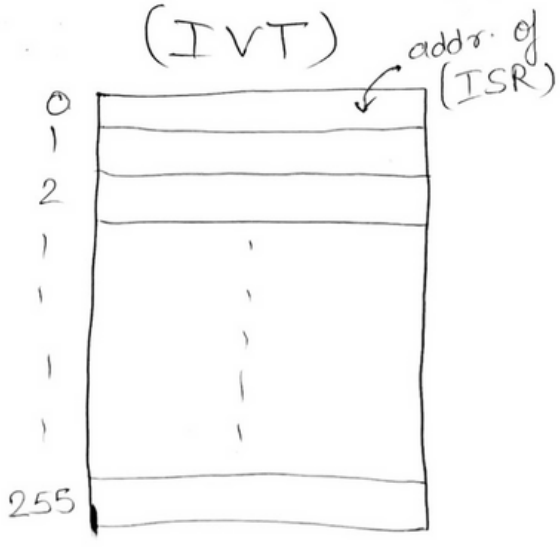
eg :- fork()
&
value error / divide by zero.

→ H/W : Outside / external
IRL (Interrupt Req. Line).

→ Maskable

→ Vectored
↳ ISR has fixed addr.





Memory Mapped I/O

→ Instruction: LDA, STA (same as memory type instructions for IO).

→ IO Accessed like any other memory location

→ Memory Read & Memory write

→ Any Register can communicate with IO devices

→ $IO/\overline{M} = 0$

→ Simple

→ 8085 μ_p

IO Mapped IO

→ Instructions: IN, OUT (Specific/different for I/O)

→ IO cannot be accessed as memory location.

→ IO read & IO write.

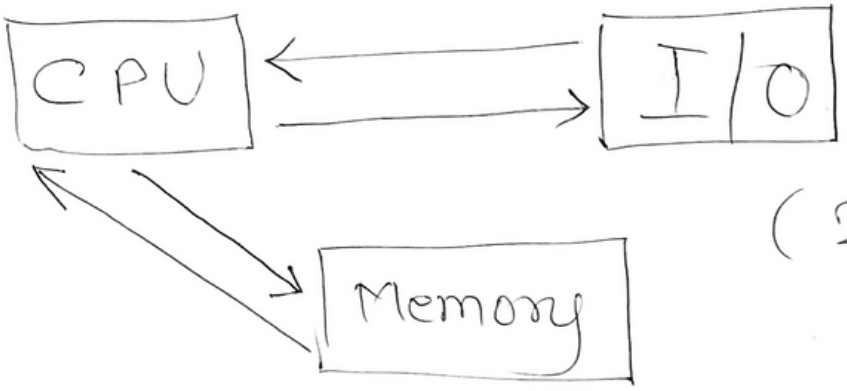
→ Only & only Accumulator.

→ $IO/\overline{M} = 1$

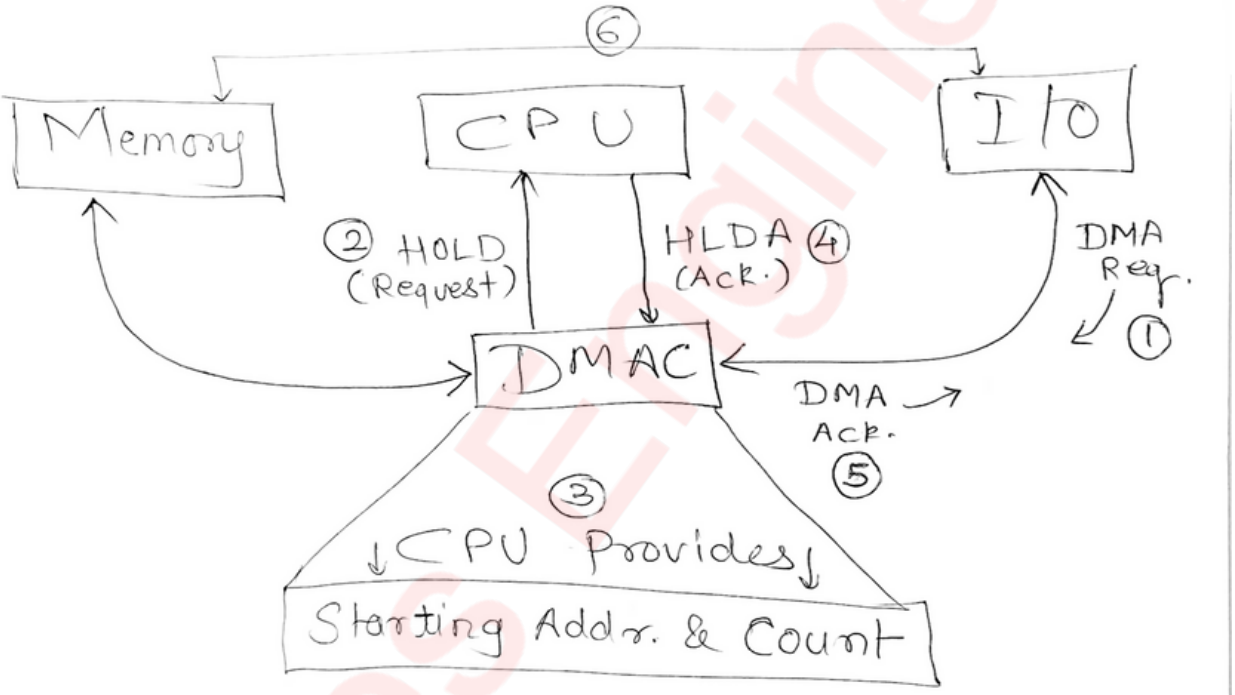
→ Complex

→ 8255 μ_p

Direct Memory Access (DMA)



(2 cycles)

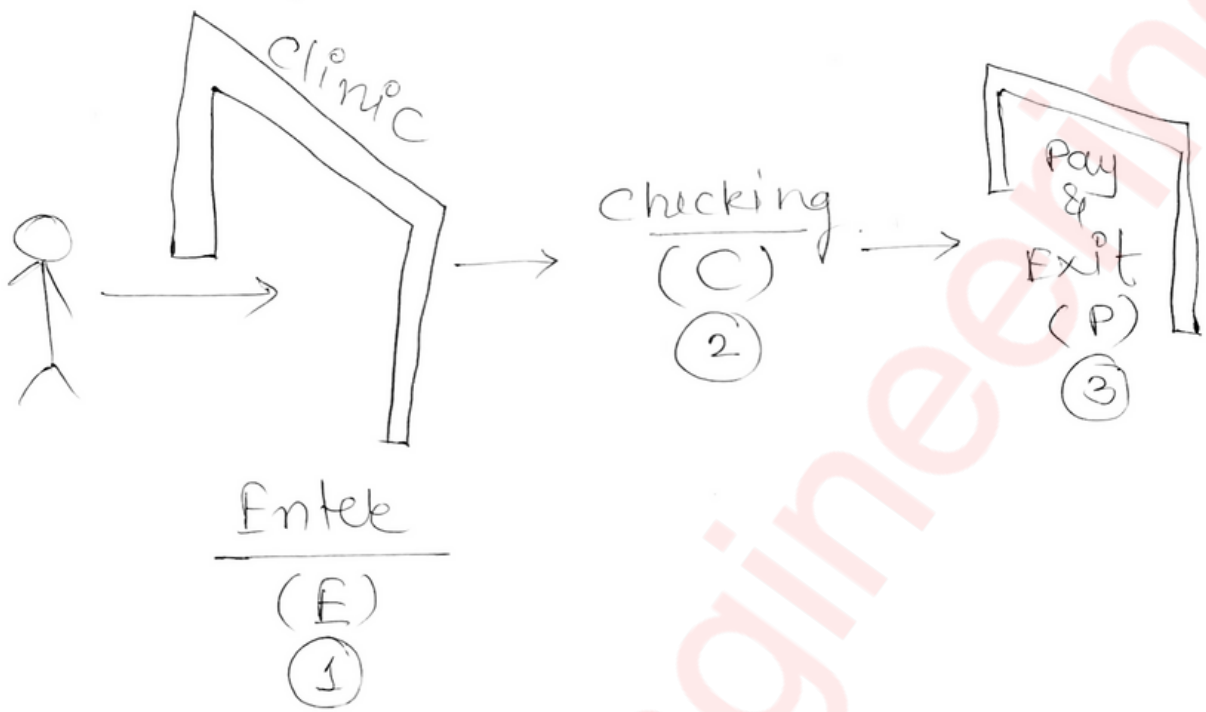


Modes:

- Burst (High speed devices)
  - whole I/O transfer is completed the given back control to CPU.
- Cycle stealing (steal cycle)

i.e while Instruction execution CPU will have ALU so No need of Buses.
Also Decode phase.

Pipelining



P1	E	C	P						
P2	×	×	×	E	C	P			
P3	×	×	×	×	×	×	E	C	P

P1	E	C	P						
P2	×	E	C	P					
P3	×	×	E	C	P				

* Simultaneous / overlapping of execution Instructions

- Instruction cycle → Fetch → Decode → Execute → Store

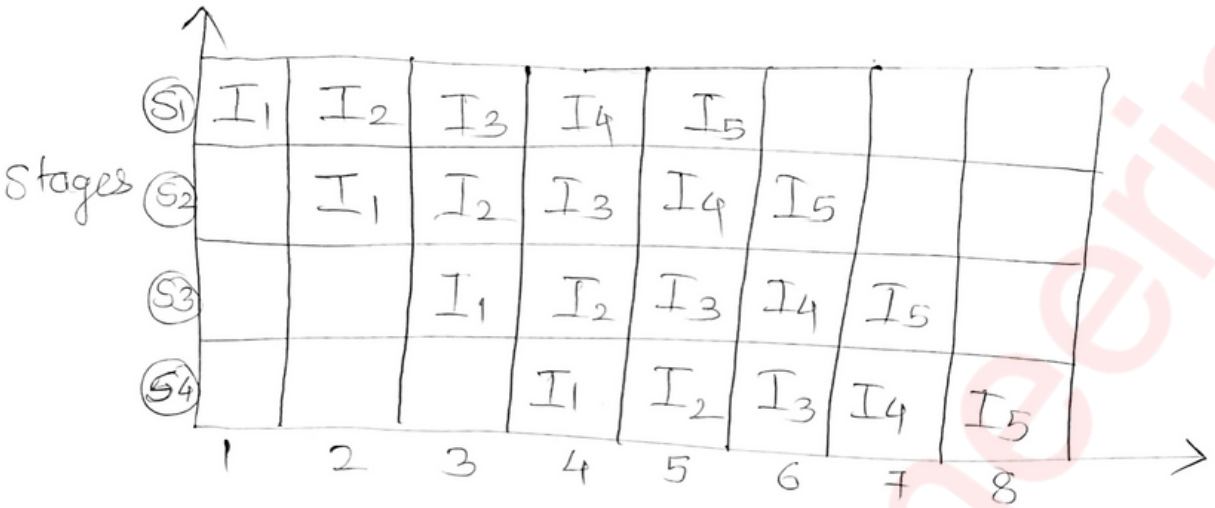
I_1	(F)	(D)	(E)	(S)				
I_2	x	x	x	x	(F)	(D)	(E)	(S)

I_1	(F)	(D)	(E)	(S)
I_2	x	(F)	(D)	(E)	(S)

- Multiprocessing

P1	I_1	(F)	(D)	(E)	(S)
P2	I_2	(F)	(D)	(E)	(S)

Space time diagram



(cc)

$4 \times 5 = 20$ clock cycles

⇒ Every stage can have

- Same delay
  - ⇒ 50ns, 50ns, 50ns, 50ns
- different delay
  - ⇒ 100ns, 50ns, 150ns, 60ns.
  - Take the max time. (150 ns)

eg:- 10 Instructions

$1 \times 4 \times 150 + 9 \times 150$

- $1 \times 4 \times 150$ → 1st Instruction
- 9×150 → Remaining 9 I.

⇒ 600 + 1350

⇒ 1950 ns

◦ Problems in Pipelining (cycle/Instr. = 1)

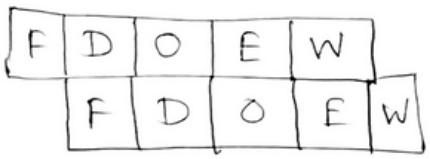
① Data Related

- → Read after write
- → Write after Read
- → write after write

RAW

$I_1 : R_1 \leftarrow R_2 + R_3$

$I_2 : R_5 \leftarrow R_1 + R_5$



R → W

WAR

$R_1 \leftarrow R_2 + R_3$

$R_3 \leftarrow R_4 + R_5$

(Rare) problem

W → R

WAW

$R_3 \leftarrow R_1 + R_2$

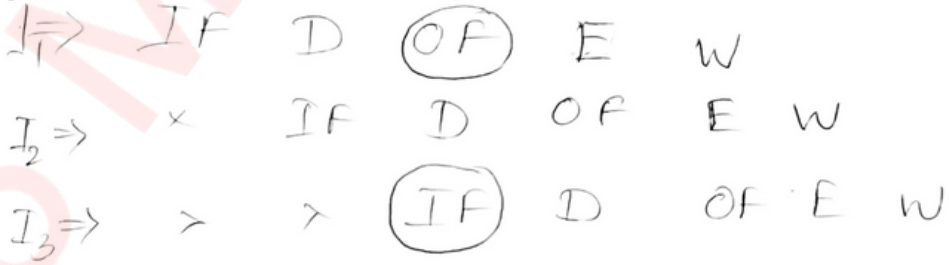
$R_3 \leftarrow R_4 + R_5$

parallel Execution Cases

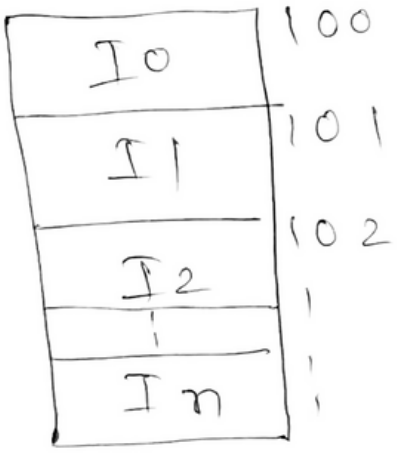
(Shared data) "R3"

② structure Related

- → Multiple 'I' require common same Resources.
- → Resources ↓ Instructions ↑



Control Related (Branch Hazards)



I_0 (F) (D) (E) (W)

I_1 (F) (D) (E) (W)

I_3 (F) (D) (E) (W)

$I_0 \rightarrow$ ADD R1, R2

$I_1 \rightarrow$ JMP 300

$I_2 \rightarrow$ ADD R3, R4 X flush/ stall.