

Computer Org. & Architecture

Syllabus:

~~Module I: Data Representation 2marks~~

Module II: Computer Architecture

Module III: Computer Organization

QRC 11.16.00

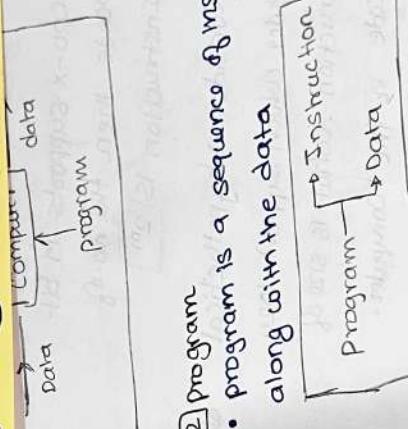
Faculty Name
Sagar . pingili
[01866765629]

- Keywords
 - [1] Computer
 - Computer is a computational machine used to process the data under the control of a application program which

- Initiated by the user.
- So computer system functionality is program execution
- ③ Instruction
 - Instruction is a binary code which is design inside the processor to perform some +

Binary - Bind - operation
with
Code

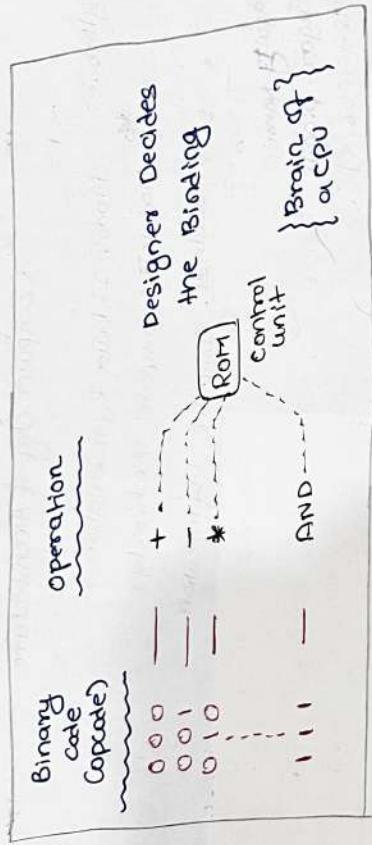
eg If CPU - x support 16 bit
 operation than
 $\frac{\text{opcode}}{\text{size}} = \frac{10^4}{16} = 10^3 \Rightarrow 2^4$



- program is a sequence of instruction along with the data

of bits required to represent the operation is called as "opcode".

46115



Prepare the insulation material

- If CPU-X supports 'N' different operations then the opcode size is $\log_2 N$ bit
 - If CPU-X supports M bit opcode then the no. of instruction is 2^m
 - Opcode size is varies with respect to the no. of operation possible in the CPU

Ques Consider a hypothetical Computer which support 1024 Instruction . what is size of opcode in the computer .

$$\text{order} = \log_2 n$$

[Encoding]

[Ques] consider a hypothetical

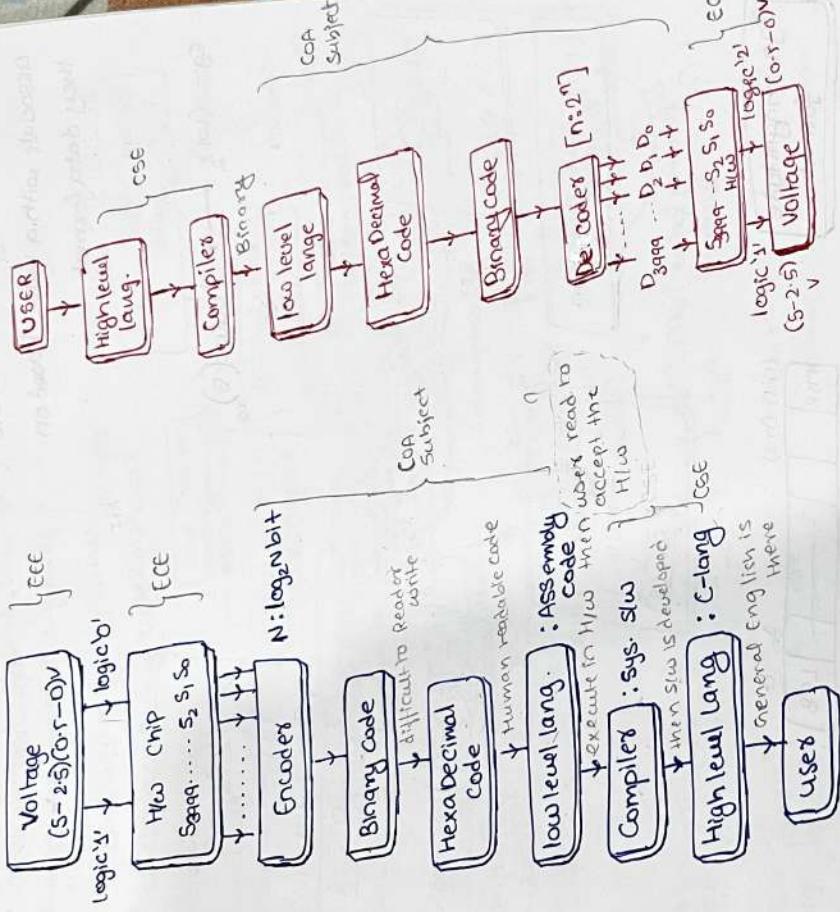
Computer which support 12 bit opcode
There many Instruction are possible
In the Computer [Coding]

$$\begin{aligned}
 \boxed{\text{Solve}} \quad \text{opcode size} &= 12 \text{bit} \\
 \# \text{ of math operation} &= 2^{\text{opcode}} \\
 \text{possible} &= 2^{12} \\
 &\approx 4096
 \end{aligned}$$

Wait | OPCODE

Designers view

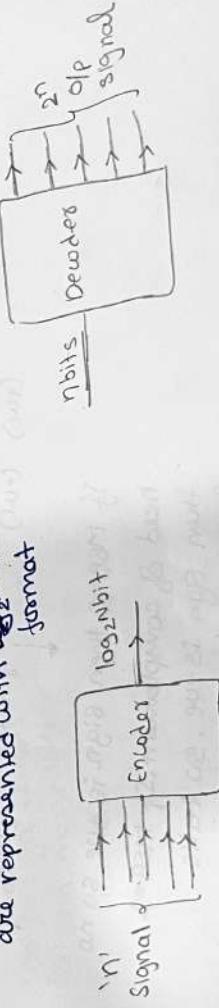
Users view



Decoding (at the time of execution)

- In this process, **n** signals produced by **n** bits

are represented with **log₂bit** format



Encoding (at the time of designing)

- In this process, **n** signals produced by **n** bits

are represented with **log₂bit** format



Q Data [is given by user]
 Data is a binary code which associate with a value based on their data format

[4] Data [is given by user]
 Data is a Binary code which is
 associate with a value based on
 they data format

$(ex:- 101)_2$ — ? operation

Depends on the
 Instruction man
 we can report
 operation

Designer

Designer Responsibility

Binary – Bind – value
code – with

Eg:- $(10)_2 \leftrightarrow ? \text{ value}(5)_{10}$

401 : 402

(10)₂ — fraction
floating point format

Sign magnitude format

- in base 2 System either 0 or 1

we can repeat the operation

$$\text{I.e. } \text{ClO}_2 - \frac{\text{Value}}{5}$$

$$= \frac{(10^2) + (0+2) + (1 \times 2^0)}{4} = 0$$

theologicis
signe
magnitude
done

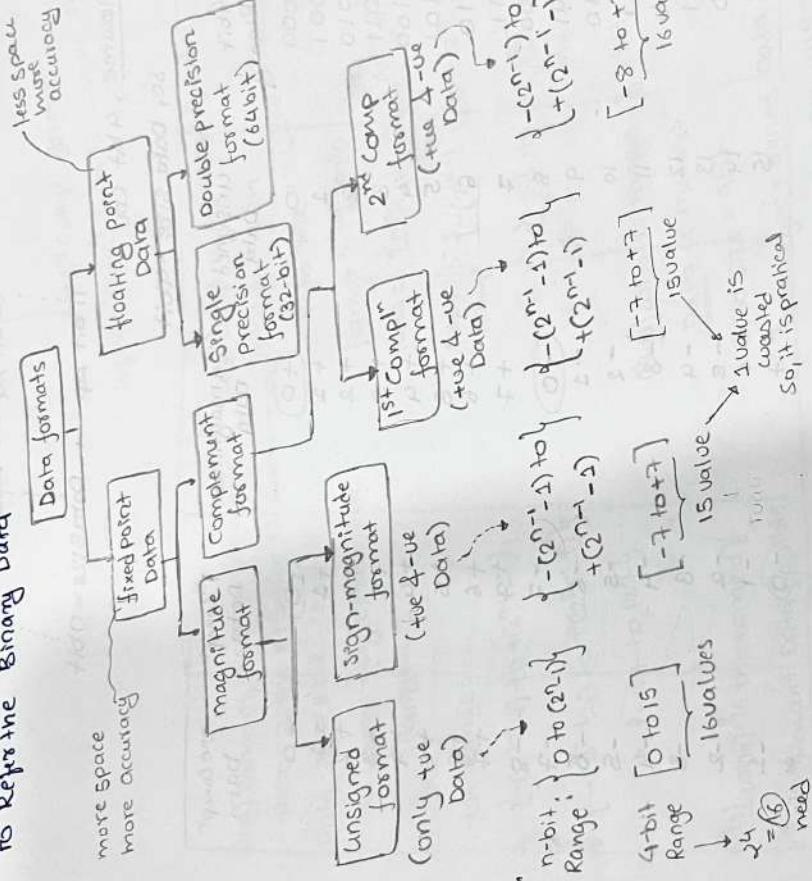
Complement: 1st comp
format: 2nd comp

If $M_{BB} = 0$ then sign is the sign
read of complement. If $M_{BB} \neq 0$
then sign is $-ve$. So take
complement to decide the sign.

Module 1 Data Representation

- In the Computer System, Data is always represented in a form of Binary. following format is used

To Represent Binary Data



- not all sign data is a sign magnitude format.

Card

- Fixed Point Data: [5bit] Data size varies w.r.t microprocessor used

Eg: 8bit MP \rightarrow Data size = 8bit

16bit MP \rightarrow Data size = 16bit

32bit MP \rightarrow Data size = 32bit

n bit MP \rightarrow Data size = n bit

Assume, 4bit CPU
So, Data size = 4bit

4bit Binary	unsigned Data	sign magnitude Data	1st Comp ⁿ Data	2nd Comp ⁿ Data
0000	0	+0	+0	+0
0001	1	+1	+1	+1
0010	2	+2	+2	+2
0011	3	+3	+3	+3
0100	4	+4	+4	+4
0101	5	+5	+5	+5
0110	6	+6	+6	+6
0111	7	+7	+7	+7
1000	-8	-0	-8	-8
1001	-7	-1	-7	-7
1010	-6	-2	-6	-6
1011	-5	-3	-5	-5
1100	-4	-4	-4	-4
1101	-3	-5	-3	-3
1110	-2	-6	-2	-2
1111	-1	-7	-1	-1

not in use b/c data redundancy / duplication

of zero

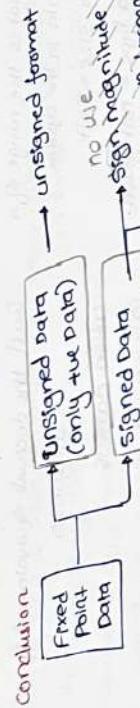
1st Comp
1000: -7

1000: 0111 (7)

2nd Comp
1000: -8

1000: 0111
1000: -8

in use
as two
code assign
for one
zero



∴ default signed Data is "2's comp" data

Ques] Range of a sign-magnitude Data in the 8-bit CPU is —
 Data size = 8 bit
 Data format = sign-magnitude format

n-bit Range:

$$\Rightarrow \{-(2^{n-1}-1) \text{ to } +2^{n-1}-1\}$$

$$\Rightarrow \{-(2^{8-1}-1) \text{ to } +2^{8-1}-1\}$$

$$\Rightarrow \{-(2^{127}-1) \text{ to } +2^{127}\}$$

Ques] what is the range of
 2's complement data possible
 in 16-bit CPU.

Data size = 16 bit

format: 2's complement

$$\text{Range} \{ -(2^{n-1}) \text{ to } +2^{n-1}-1 \}$$

$$16\text{-bit} \quad \{ -(2^{16}-1) \text{ to } +2^{16}-1 \}$$

$$\text{Range} \Rightarrow [-2^{15} \text{ to } +2^{15}-1]$$

Ques] what is the range of 10-bit

1's comp Data

Data size = 10 bit

format: 1's comp
 range: $\{ -(2^{n-1}-1) \text{ to } +(2^{n-1}-1)\}$

10-bit Range:
 $\{ -(2^{10-1}-1) \text{ to } +(2^{10-1}-1)\}$

$$\Rightarrow \{ -511 \text{ to } +511 \}$$

Ques] what is range of 8-bit signed
 Data in the size?

Data size = 8 bit

Data type = signed data

Data format = not given
 \Downarrow

B1 Default 2's comp

$$\text{Range} \{ -(2^{n-1}) \text{ to } +(2^{n-1}-1) \}$$

$$-2^{15} \text{ to } +2^{15}-1$$

Ques] what is the range of 10-bit

2's comp Data

range $\{ -(2^{10}-1) \text{ to } +2^{10}-1 \}$

$$\Rightarrow [-2^{9} \text{ to } +2^{9}-1]$$

Ques] what is the range of a 7-bit data in the system

Data size = 7-bit

Data type = not given

Default Data is unsigned

Range $\{0 \text{ to } (2^7 - 1)\}$

$$\begin{array}{r} 7 \text{ bit} \\ \hline 1000000 : (-110) \end{array}$$

$\Rightarrow 0 \text{ to } 127$

C) 2's comp

$$\begin{array}{r} 7 \text{ bit} \\ \hline 1000000 : (-110) \\ 1000000 \rightarrow 0111110 \\ 1000000 \rightarrow 01101110 (110) \end{array}$$

Ans] the decimal equivalent of

$$\begin{array}{r} 7 \text{ bit} \\ \hline 1000000 \\ 1000000 : (-110) \\ @ \text{ unsigned} \\ \hline 1000000 \\ \Rightarrow (46) \end{array}$$

Carry

the carry is 0

where data is

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

the carry is 0

where data is

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

the carry is 0

where data is

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

the carry is 0

where data is

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

the carry is 0

where data is

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

the carry is 0

where data is

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

the carry is 0

where data is

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

the carry is 0

where data is

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

the carry is 0

where data is

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

the carry is 0

where data is

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

the carry is 0

where data is

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

the carry is 0

where data is

0

Carry flag is used to handle the out-of-range data in they computers after processing unsigned no.

$$\begin{array}{r} 1 \quad 1 \quad 1 \\ 1 \quad 1 \quad 1 \\ + 1 \quad 1 \quad 0 \\ \hline 2 \quad 0 \end{array}$$

Note: $n\text{-bit}(n-1\text{-bit}) = (1+1)\text{-bit}$

18-bit extra space

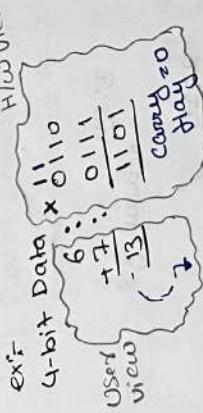
flip flop

flag

flag

carrying

used to hold the range exceeding status of a unsigned data



Range is not exceeding

Test with : $0 \rightarrow 2^{25-1}$
5Bit Range $\rightarrow 2^{10-1}$

- condition of carry flag.
- 1. Is there carry bit out of 16's after processing set (\rightarrow) carry
- 2. its conditional flag if true → Reset (0)
if false → Reset (0)
- 3. its conditional flag if false → Reset (0)
if true → Set (1)
- to handle overflow no.

- Is there an Extra bit out of MSB
- Is there an Extra bit out of MSB
- Is there an Extra bit required into the MSB

- carry flag is physical present to conduct... inside the flag
- conduct...
- Registers
- taking extra bit from flag register

Justify: ~~PSW~~(program status word)

flag register

Accum.

- Accum. is a register used to hold the ALU input or output
- Flag register hold the system supported flags

Carry flag = 0

$1101 \rightarrow 0111(13)$

Ans

ex:- 4bit Data
 Ex:- 8 : 1000
 $\begin{array}{r} + \\ 9 : 1001 \\ \hline 17 : 0001 \end{array}$
 out of range

Ex:- Data : 1001
 $\begin{array}{r} \times 1010 \\ \hline 1010 \end{array}$
 out of range

Ex:- Data : 1010
 $\begin{array}{r} \times 1010 \\ \hline 1010 \end{array}$

[Cry: 1]

Multiplication

- In multiplication process two action are performed.
 - generation of partial product
 - summation of the partial product
- After the generation of partial product, provide the summation to produce the final product.

Ex:- 4bit Data
 $\begin{array}{r} 8 : 1000 \\ + 9 : 1001 \\ \hline 17 : 0001 \end{array}$
 out of range

Ex:- Data : 1001
 $\begin{array}{r} \times 1010 \\ \hline 1010 \end{array}$

[Cry: 1]

In Decimal
no.

Borrow
10 (base)
 $\begin{array}{r} 28 \rightarrow 13 \\ - 17 \\ \hline 06 \end{array}$

Carry: 1

In Decimal
no.

Borrow
10 (base)
 $\begin{array}{r} 28 \rightarrow 13 \\ - 17 \\ \hline 06 \end{array}$

Carry: 1

- In the multiplication, Partial product and generated based on the multipliers that is when the multipliers bit is 1 then the partial product is then the partial product is multiplicant otherwise the partial product is "0"

Ex:- 4bit Data
 multiplicand multiplier
 $\begin{array}{r} 1100 \\ \times 1010 \\ \hline 1100 \\ 0000 \\ \hline 111000 \end{array}$
 3: 011 carry(1)
 2: 10 carry(0)
 1: 100 carry(2)
 0: 110 carry(0)
 5: 101 carry(2)

Final product
 41200001
 result.

- Software is the reason we don't use carry flag for multiplication.

$[n \text{-bit}] * [n \text{-bit}] = 2n \text{ bit}$

Registers pair is used in the CPU to hold the final product

Ques] Consider the following multiplication

$$(1010110)_2 * (110)_2 = (4010100)_2$$

What are the values of w, y & z variables?

$$\begin{array}{r} (1010110)_2 * (110)_2 \\ \hline 2 1 0 0 1 1 0 \\ 2 1 0 0 1 1 0 * \\ 2 1 0 0 1 1 0 * \\ 2 1 0 0 1 1 0 * \\ \hline 1 0 1 1 0 0 1 . (010100)_2 \end{array}$$

Ques] Consider the following partial product generated value

multiplication in a sequence
 $101101, 100000, 000000, 1101101, 101010000000$

what is the multiplier?

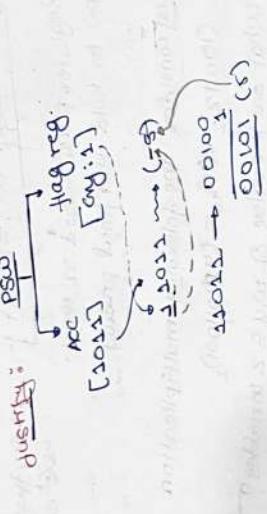
Ques] Consider the following partial product is multipicand : when multiple bits "w" partial product is 0 : when multiple bit is "0"

partial product is 0

$$\begin{array}{r} 101101 \\ \times 011001 \\ \hline 000000 \\ 000000 \\ 000000 \\ 000000 \\ 101101 \\ \hline 101101 \end{array}$$

$$101101 * 011001 \xrightarrow{45} (11125)_0$$

$$\begin{array}{r} \text{Eg. data} \\ \begin{array}{r} \text{5 : } 010 \\ \times 101 \\ \hline \text{5} \end{array} \end{array}$$



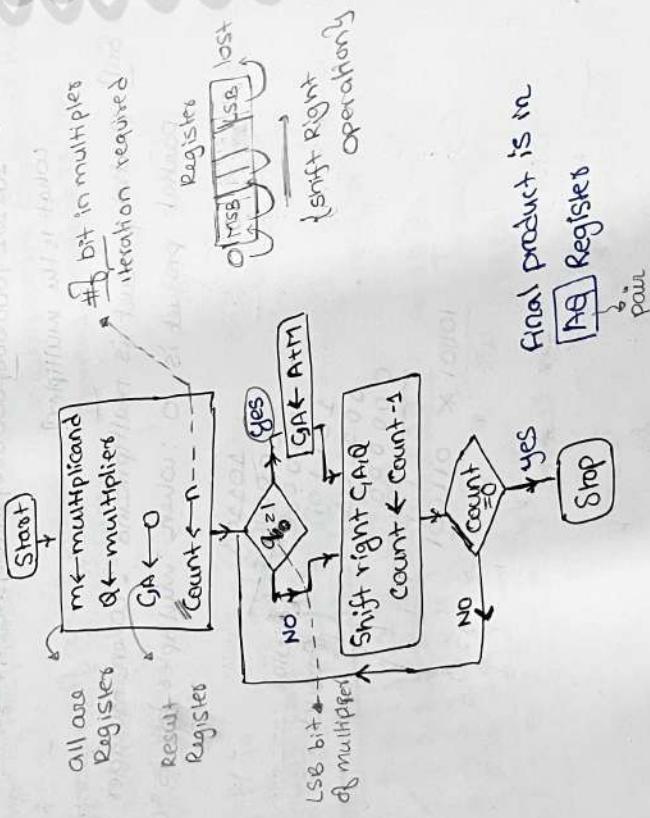
Note

- Limitation in the manual multiplication is,
- 1 Required more register to store the partial products.

- 2] Summation process become very complex in the H/W implementation
 ∵ Optimization is required in the multiplication implementation

1.e Accumulated Addition

- unsigned multiplication implementation in the H/W is describe using the following flow chart



卷之三

Eg: 4-Bit Data

$$\overbrace{1101}^{\text{multiplicand}}_2 \quad \overbrace{(101)}^{\text{multiplier}}_2$$

(43) 10

- # of Arithmetic operation required in the multiplication is equal to # of its multipliers

Final product

④ $\begin{array}{r} 0110 \\ \times 101 \\ \hline 0110 \end{array}$

二〇〇一、十一月三十日

0 0 1001 1110 : shift
0 0 1001 1110 : last count²

0 0100 11110 : sumfr $\text{G}_{\text{sumfr}} = 1$ (1)

A: 0100
B: 1100

Q: summarize
A: 1001 0001

16

$$n^{(k)} + \dots + n^{(m-1)} \rightarrow 4(2^{m-1}-1)$$

Range ~

Range: 2-8 rotary

discovery of data.

卷之三

०८३४२५७१०००

→ the dark flow

$$\left\{ \begin{array}{l} b=1 \\ a=1 \end{array} \right.$$

outflow data

卷之三

卷之三

Signed Data
Default is simple

The diagram illustrates a stack overflow attack. A horizontal bar represents memory. The left side shows a section labeled "true data" with a downward-pointing arrow. To its right is a section labeled "-ve data" with a downward-pointing arrow. A large, hatched area representing "overflow" extends beyond the boundaries of the "true data" section, overlapping into the "-ve data" section. An upward-pointing arrow from the hatched area indicates it is being written into memory. The boundary between the two sections is marked with a vertical line.

ex: $+5 : 011$
 $+5 : 011$
 \oplus

overflow $\left[\begin{array}{c} 16 \\ 15 \end{array} \right] - \left[\begin{array}{c} 16 \\ 15 \end{array} \right] = \left[\begin{array}{c} 0 \\ 0 \end{array} \right]$

卷之三

$$\begin{array}{r}
 -8 : 2's \text{ complement of } "8" \\
 0 : 2000 \\
 \downarrow \\
 2's \text{ complement} \\
 1111 \\
 \hline
 -8 : \underline{\underline{1000}}
 \end{array}$$

$$\begin{array}{r}
 \text{ex: } +6 : 0110 \\
 + -2 : 1110 \\
 \hline
 +4 : 0100 \\
 \text{Cry=0} \\
 \text{OV=0} \\
 \downarrow \\
 \text{Range is not} \\
 \text{exceeding} \\
 \{ -2^{25-1} \text{ to } +2^{25-1} \}
 \end{array}$$

- Carry flag is only suitable for unsigned no.
- flag used in signed num. is not carry flag
- These flag consist of some specific condition
- use overflow flag in signed no.

\rightarrow 16 bits

- use overflow to count the overflow can be covered

Ex: +7 : 0111
 + -7 : 1011
 \hline
 +4 : 1110
 Cry=0
 OV=0
 → so there is no true overflow
 Reason why we don't use carry with signed no.

$$\begin{array}{r}
 \text{no carry but overflow flag is there} \\
 \text{Cry=0 but OV=1} \\
 \downarrow \\
 \text{ex: } +7 : 0111 \\
 + -7 : 1011 \\
 \hline
 +4 : 1110 \\
 \text{Cry=0} \\
 \text{Overflow} \\
 \text{Flag = 1} \\
 \text{OV}
 \end{array}$$

- if two no. is added & the result contain -ve sign then overflow present.
- if two -ve no. is added & if these result sign is same then overflow

$$\begin{array}{r}
 \text{Out} \quad \text{In} \\
 \hline
 1110 \\
 1110 \\
 \hline
 1100
 \end{array}$$

- XOR operation has in carry & out carry with respect to MSB is the overflow to HSB

Ex

$$\begin{array}{r} +2 : 0010 \\ +3 : 0011 \\ \hline +5 : 0101 \end{array}$$

Range not exceeding

Note
 $(n-1) + (n-1) = (2^n) - 1$

$$1 \text{ bit extra space required}$$

1 flip-flop

Flag

Overflow flag

used in the signed data to hold the range exceeding condition

cond: "There is a carry-in to MSB & no carry out of MSB @ vice versa" $\rightarrow T \setminus \text{Set-Z} = \text{OV}$ (Overflow)
 $F \setminus \text{Reset-Y} = \text{NV}$ (NoOverflow)

Condition of overflow

View I:

cond: "There is a carry-in to MSB & no carry out of MSB @

vice versa" $\rightarrow T \setminus \text{Set-Z} = \text{OV}$ (Overflow)

$F \setminus \text{Reset-Y} = \text{NV}$ (NoOverflow)

View II:
"XOR" operation between In carry & out carry will MSB is a溢出 (overflow)

In carry	out-carry	Overflow flag status
0	-	0
0	-	0
0	-	0
1	-	0

$OV(x,y,z) = \overline{x}yz + xy\bar{z}$

$x = 1$
 $y = 1$
 $z = 1$

Two overflow conditions

x: MSB bit of operand 1

(sign)

y: MSB bit of operand 2

(sign)

z: MSB bit of a Result (sign)

View 3: when the same sign data is processed, Result contains two overflow conditions

Opposite sign than overflow present

$OV(x,y,z) = \overline{x}\bar{y}z + x\bar{y}z$

$x = 1$
 $y = 0$
 $z = 1$

Two overflow conditions

4-bit Data

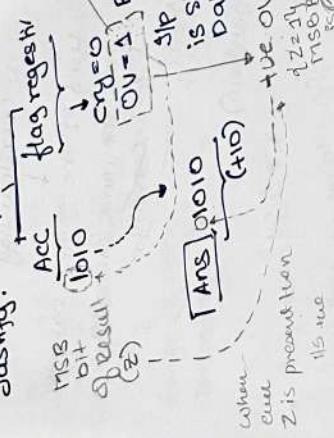
$$\begin{array}{r} \text{ex: } \\ \text{Acc: } \\ \begin{array}{r} +7 : 1111 \\ +3 : 1101 \\ \hline +10 : 0101 \\ \hline \text{Ans: } 0101 \\ \text{Cry: } 0 \\ \text{OV: } 1 \\ \text{Z: } 0 \end{array} \end{array}$$

$$\begin{array}{r} \text{ex: } \\ \text{Acc: } \\ \begin{array}{r} +7 : 1111 \\ +3 : 1101 \\ \hline +10 : 0101 \\ \hline \text{Ans: } 0101 \\ \text{Cry: } 0 \\ \text{OV: } 1 \\ \text{Z: } 0 \end{array} \end{array}$$

$$\begin{array}{r} \text{ex: } \\ \text{Acc: } \\ \begin{array}{r} -7 : 0011 \\ -3 : 1101 \\ \hline +4 : 0100 \\ \hline \text{Ans: } 0010 \\ \text{Cry: } 0 \\ \text{OV: } 1 \\ \text{Z: } 1 \end{array} \end{array}$$

no OV

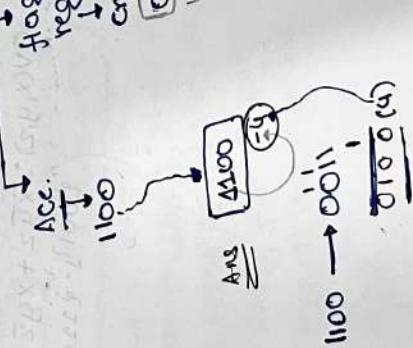
Justify:



$$\begin{array}{r} \text{ex: } \\ \text{Acc: } \\ \begin{array}{r} -7 : 0011 \\ -3 : 1101 \\ \hline +4 : 0100 \\ \hline \text{Ans: } 0010 \\ \text{Cry: } 0 \\ \text{OV: } 1 \\ \text{Z: } 1 \end{array} \end{array}$$

no OV

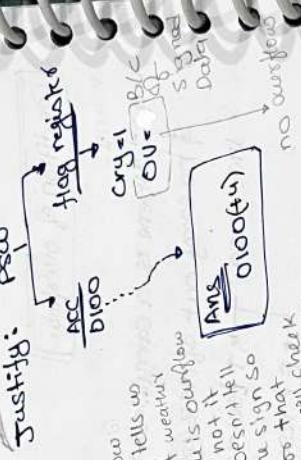
Justify:



$$\begin{array}{r} \text{ex: } \\ \text{Acc: } \\ \begin{array}{r} -7 : 0011 \\ -3 : 1101 \\ \hline +4 : 0100 \\ \hline \text{Ans: } 0010 \\ \text{Cry: } 0 \\ \text{OV: } 1 \\ \text{Z: } 1 \end{array} \end{array}$$

$$\begin{array}{r} \text{ex: } \\ \text{Acc: } \\ \begin{array}{r} -7 : 0011 \\ -3 : 1101 \\ \hline +4 : 0100 \\ \hline \text{Ans: } 0010 \\ \text{Cry: } 0 \\ \text{OV: } 1 \\ \text{Z: } 1 \end{array} \end{array}$$

Justify:



$$\begin{array}{r} \text{ex: } \\ \text{Acc: } \\ \begin{array}{r} -7 : 0011 \\ -3 : 1101 \\ \hline +4 : 0100 \\ \hline \text{Ans: } 0010 \\ \text{Cry: } 0 \\ \text{OV: } 1 \\ \text{Z: } 1 \end{array} \end{array}$$

no OV

Justify:



$$\begin{array}{r} \text{ex: } \\ \text{Acc: } \\ \begin{array}{r} -7 : 0011 \\ -3 : 1101 \\ \hline +4 : 0100 \\ \hline \text{Ans: } 0010 \\ \text{Cry: } 0 \\ \text{OV: } 1 \\ \text{Z: } 1 \end{array} \end{array}$$

$$\begin{array}{r} \text{ex: } \\ \text{Acc: } \\ \begin{array}{r} -7 : 0011 \\ -3 : 1101 \\ \hline +4 : 0100 \\ \hline \text{Ans: } 0010 \\ \text{Cry: } 0 \\ \text{OV: } 1 \\ \text{Z: } 1 \end{array} \end{array}$$

Note

- whenever overflow is set then bring the carry bit (carry bit) to the MSB bit position
ex: $\begin{array}{r} \text{Cry: } 0 \\ \text{OV: } 1 \end{array}$ or $\begin{array}{r} \text{Cry: } 1 \\ \text{OV: } 1 \end{array}$ then $\begin{array}{c} \text{if OV is} \\ \text{set} \end{array}$

Alternative logic to flush the Result is bring the carry flag status into a MSB position of a Result when overflow flag is set (1)

Sign Extension

- this process is used to store the smallest data into a large storage space to preserve the sign of a Data. by replicating (Duplicate) the MSB bit into a extra space

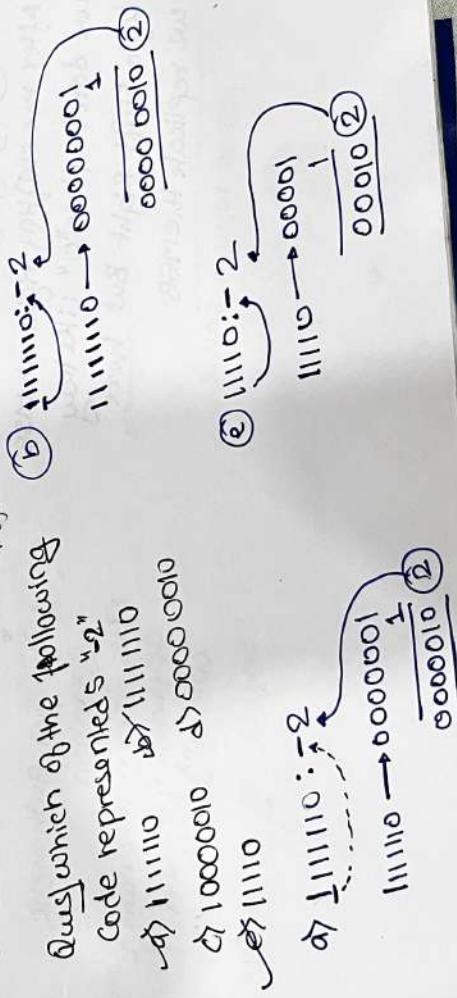
- sign extension is used in signed Data Representation only.

Note

- 0's padding is used in the unsigned data

- sign extension is used in the signed data

Default signed data is 2's complement



Ex: (-7): 1001 (4bit)

Store the Data into a 8bit Space

00001001 : +9
0's padding



00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000

00010
11110
00010
11110
00010
11110
00010
11110

00010
11110
00010
11110
00010
11110
00010
11110

c) $00000000 : +2$ ② $100000010 : -1^{126}$

Signed multiplication

- Booth's multiplication

[Q8] Bit pair multiplication

ex : 4 bit multipliers ASSUMPTION

Bit pairs:

The diagram illustrates a stack structure. It consists of a vertical rectangle divided into horizontal sections. The top section is labeled "last". Below it, several arrows point upwards from labels like "LSS" and "RSP" towards the "last" label, indicating the direction of the stack pointer. The bottom section is labeled "RSP".

After the shifting right at 1:00 we don't place "0" like they logic shift right but here we replicate them so it remain same

- Signed multiplication process is control by the multipliers
- Bit pair one formed based on bit pair one formed based on half multiplier. In these process one assumption is required to get the initial pair i.e. $[q_{n-1} = 0]$

- Different action are defined w.r.t. Bit pair status, to perform multiplication

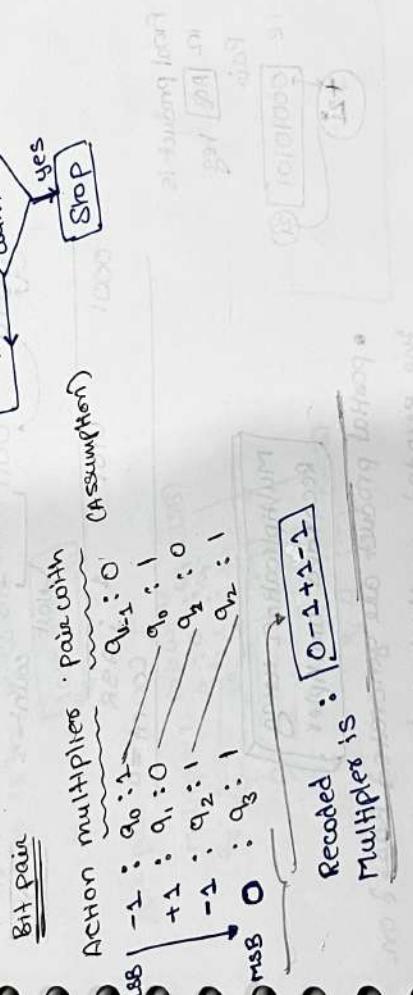
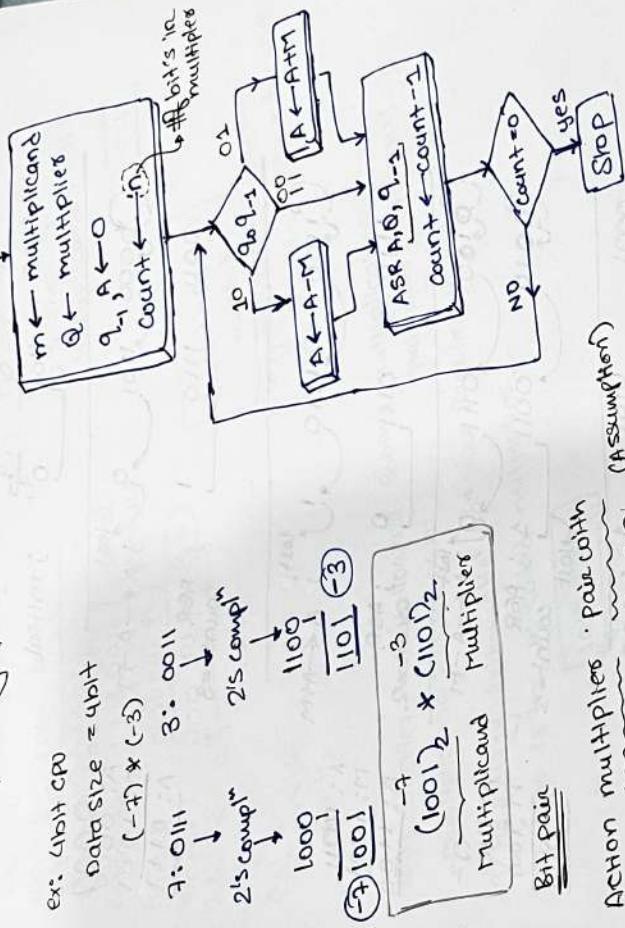
Re-colouring	$O \xrightarrow{+} O$
Action	$ASR + \frac{1}{2} ASR - \frac{1}{2} ASR ASR$
Bit pair	$00-00--$

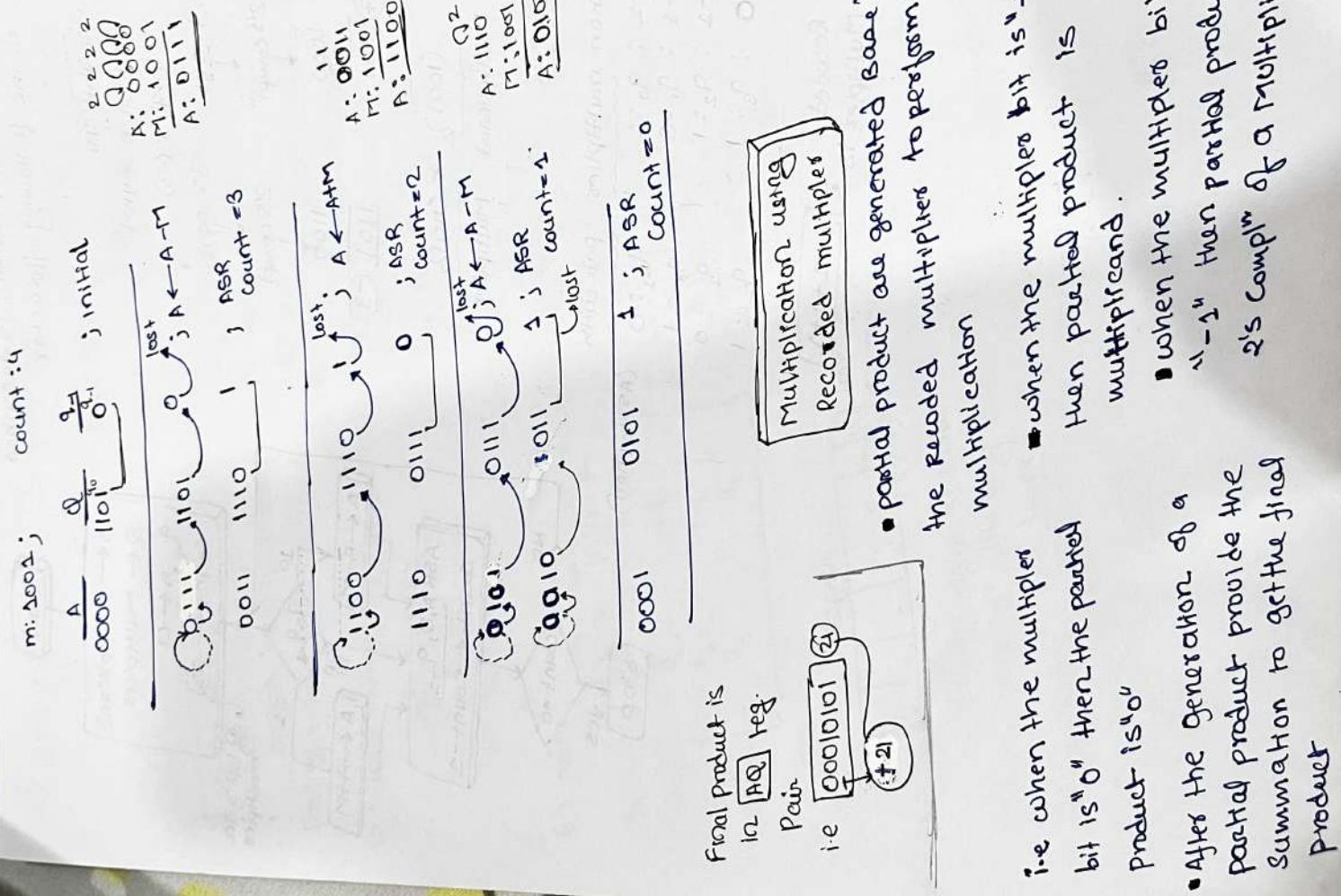
DSR: Arithmetic Shift Right

- This is logical operation
- It's data manipulation
 - Operation (logical)
 - Operation (physical)
- It's not Arithmetic

11

- Signed multiplication is described in the following flow chart.





$$\text{Ex: } (-7)^{10} \neq (-3)^{10}$$

$$\begin{array}{r} \text{multiplicand: } 001 \\ \times \text{ multiplier: } 011 \\ \hline \end{array}$$

$(n \text{ bits}) * (\text{m bits})$
 $= 2^n \cdot 2^m$ used by the two
 binary multipliers used for tree used
 recursive multipliers used for tree used

মানবিক বিদ্যা : ১, ০০০, ০০০।

How many times is it sometimes

La voulte si que re-couvre

Multiples of 1000000

Bit pair

A7 Antennae up.

A diagram showing a stack of cards. The top card is labeled "Top".

Both's multipliers: 10100101
multiplier: 11111111

卷之三

MATERIALS AND METHODS

multiplies 10100 : 01

Some consider the following signs multiply them.

$$\text{multiplied} \xrightarrow{(-58) * (-120)} \text{unmultiplied}$$

Banyak : 10002000

number : -120 Date : 20/04/2014

120 : 01111000
 10000111 4
 10001000 :
 -120

Bit pairs:

Paul Q₀ Q_t	000-000 000-000 000-000	100 ft 100 ft 100 ft
<u>Multiples</u> Q₀ Q_t	000-000 000-000 000-000	Ans. Ans. Ans.
Q₀ Q_t	000-000 000-000 000-000	1-1000 1-1000 1-1000

thus consider the following multiple possibility:

$(10110) * (110-10+1)$
What is the Binary multiplier.

⇒ what is the Binary wealth?

Recorded : $+10-10+1-1$
 Multiplexer : $\frac{q_5(5)}{q_5(0)}$: - - 1 0 0 1 0 1 0 1
 USB : $\frac{q_5(5)}{q_5(0)}$: - - 1 0 0 1 0 1 0 1
 MSB : $\frac{q_5(5)}{q_5(0)}$: - - 1 0 0 1 0 1 0 1

Braavy multiplex

100110

elcomes the most
product is decided

$$\begin{array}{r}
 (-10) * (+25) = \\
 (-10) * (011010) = \\
 -10 \xrightarrow{-10} 1011010 \rightarrow 010010 \xrightarrow{+25} 100111 \xrightarrow{+25} 110011 \text{ (Carry)} \\
 (01101) * (011001)
 \end{array}$$

10

Bit Pairs : Multiples of 1000-000
 1-1000 /Ans.
 pair with 1's
 $\frac{1000}{1000-000}$ - 000 1000 T (Ans)

ex considers the following bit pair in the multiplication and uses the value of

1. *Phytomyzidae*
2. *Homoptera*
3. *Collembola*
4. *Diptera*
5. *Orthoptera*
6. *Lepidoptera*
7. *Hymenoptera*
8. *Homoptera*
9. *Homoptera*

The data is the
① unsigned format
② signed format.

Digitized by srujanika@gmail.com

```

    graph TD
        PSW[PSW] --> Acc[Acc]
        Acc --> ORI_R1[ORI R1]
        R1[ORI R1] --> ORI_R1
        Acc --> ORI_R1
        Acc --> ORI_R1
        ORI_R1 --> Result["01011110"]
    
```

G1000: ④ 20002000

$10011100 \rightarrow 11000011$ 

 Mantissa Alignment process is used in the floating point representation, to adjust the decimal point. In this process Right Alignment → ↑(Increase) Exponent ← Exponent

- Align to Right Bitness
- Left Alignment → ↓ Exponent

ex. +1011.011 * 2⁺⁶

(1.0011011)

+1.011011 * 2⁺⁶⁺³

Diagram illustrating the state transitions of a memory cell:

```

graph TD
    S1["+ 1.01011 * 2^4"] -- Read --> S2["+ 1.01011 * 2^4"]
    S2 -- "By Default  
(Circumvents  
the loss)" --> S3["+ 1.01011 * 2^4"]
    S3 -- Write --> S4["+ 1.01011 * 2^4"]
    S4 -- "not stored  
(Loss)" --> S5["+ 1.01011 * 2^4"]
    S5 -- Write --> S6["+ 1.01011 * 2^4"]

```

The entire process is enclosed in a box with a label "A bit b...")".

When the Data is in normal format, then it's store into the memory using double (double 64bit) word by language

The IEEE formats. ... is tremendous present

- 9 A/c to the IEEE 754 standard
 - 10 single precision (32 bit) format
 - 11 double precision (64bit) format
 - 12 IEEE 754 standard

- floating point Data is stored in 32 bit format

=) 8abit joystick (double)

Biased Exponent (BE) file
used to represent the exponent [tex]

\rightarrow true if -ve Exponent

→ signed Exponent
→ 2's complement format
normalized

Required msbit to represent the sign of an Exponent

The diagram illustrates a 32-bit memory location. The bits are numbered from 0 at the bottom to 31 at the top. Bit 28 is highlighted as the most significant bit (MSB) and is labeled "28 bit (m)". A bracket above the bits indicates the range from bit 28 down to bit 0. The label "Data S" is positioned to the right of the bit 0 position.

This kind of structure not possible in the system

Two misses never possible in the Registers (i) Data 2's complement exponent is not possible in the system so, Alternative register i.e biased exponent

- So exponent are not repeated

Note

- normal Bias is max. possible
+ the exponent
- excess Bias is centre of the
Exponent range

4A
Page 20

$$\begin{aligned}
 \text{ex: } & 32\text{-bit format} \\
 \text{Data} & \boxed{+1.1011 * 2^7} \\
 BE & = AE + Bias \\
 & = +1.8 + (+1.27) \\
 & = +1.33 \quad \{BE
 \end{aligned}$$

$$\begin{aligned} \text{Data} &+ 11011 \times 2^6 \\ BE &\sim AE + BIQ \\ &= (-6) + 4 \\ &= +121 \end{aligned}$$

$$= +121 \{ \text{Be} < \text{Bash} \}$$

Note BE concept is used represent the sign of a AE with following relation i.e. ($BE > Bias$) Then AE is true ($AE > Bias$) Then AE -ve

To read the data from the memory and from the ROM port the AT
i.e. $NE = OE - \text{bias}$

ex: format:

5	BE	M
list	bit	list
1011	0011	1011

 Data: +1.1011*2¹³

$$\text{Bias} = \text{AE} + \text{BIAS}$$

$$\text{BIAS} = + (2^{n-1})$$

$$z + (2^{6-1})$$

卷二

$$\begin{aligned}AE &= 001101 \{13\} \\Bias &= 01111 \quad \{+31\} \\BE &= \underline{101100} \quad \{+41\}\end{aligned}$$

$$\begin{array}{l} \text{Exponent Read: } AE = BE - \\ \begin{array}{r} 101100 \\ + 1111 \\ \hline 100101 \end{array} \end{array}$$

$$\Delta E = BE - B_{\text{IAS}}$$

۷۰

$$4 \cdot 10^{11} * 2^{-13}$$

$$\begin{aligned} \text{Score: } & BE = AE + BIAS \\ & BIAS = +81 \\ & AE = -13 \\ & \text{option: } 1 \rightarrow 9^{\circ}\text{C/50m} \end{aligned}$$

13:00:00 19/11/13

1100 *Science*

卷之三

三三三

卷之八

三

$$\Delta E = BE - Bi_0 S$$

卷之三

۲۱۰

卷之三

LIBRARY

10011

四三

卷之三

101100 C(13)

1

$\Rightarrow +(\bar{S}_{\theta} - 1)$

1

Note: When the format is design with excess bias, then the bias is the centre of the exponent range

i.e. $\left(\frac{2^n}{2}\right) \text{ or } (2^{n-1})$ n: BE field size
 • In there format we can take the complement of a MSB bit
 ⚡ BE to get the AE rather than subtraction

$$\begin{array}{l} AE = BE - \text{Bias} \\ \text{or} \\ AE = \text{complement of MSB bit of BE} \end{array}$$

Ex: format:

S	BE	M
1bit	6bit	13bit

 ↓ Excess Bias

$$\begin{array}{l} \text{Data: } \underline{\underline{+1.10110}} \times 2^{+13} \\ AE: \underline{\underline{001101}} \end{array}$$

$$\begin{array}{l} \text{Expo. store: } BE = AE + \text{bias} \\ \text{Bias} = \text{Excess} = \underline{\underline{2^{(2^{n-1})}}} \\ = \underline{\underline{(2^{6-1})}} = \underline{\underline{32}} \end{array}$$

Expo Read:

1st method

$$AE = BE - \text{bias}$$

$$\begin{array}{l} BE: \underline{\underline{101101}} \quad \{ BE > \text{bias} \} \\ Bias: \underline{\underline{100000}} \quad \downarrow \\ AE: \underline{\underline{001101}} \quad [+13] \end{array}$$

$$\begin{array}{l} \text{2nd method} \\ AE = \text{complement of MSB bit of BE} \\ BE: \underline{\underline{101101}} \quad \{ BE > \text{bias} \} \\ \text{Complement of MSB} \\ \downarrow \\ AE: \underline{\underline{001101}} \quad [+13] \end{array}$$

Normal Biased.
 Max. possible +ve exponent

Centre of the exponent range

Suppose format is given as

S	BE	M
1bit	6bit	13bit

Default Bias = normal Bias
 $\{ +2^{n-1} \}$ +ve expo. -ve

$$\rightarrow +(\underline{\underline{2^{6-1}}}) \quad \rightarrow \underline{\underline{31}}$$

$$\begin{array}{l} \text{Bias} = \text{excess bias } (2^{n-2}) \\ \text{Bias} = \underline{\underline{\left(\frac{2^6}{2}\right)}} = \underline{\underline{32}} \quad \text{+ve Exponent} \\ \text{more } \uparrow -\downarrow \quad \text{+ve Exponent} \end{array}$$

S	BE	M
1bit	6bit	13bit

$$\begin{array}{l} \text{Bias} = \text{excess bias } (2^{n-2}) \\ \text{Bias} = \underline{\underline{\left(\frac{2^6}{2}\right)}} = \underline{\underline{32}} \quad \text{-ve Exponent} \\ \text{+ve Exponent} \end{array}$$

Note • Default Bias used in the IEEE format is format bias

- Excess Bias is not in use

③ Mantissa field:

- Used to represent the mantissa part of a data (fraction)

- Data is always stored in the memory in a form of normalized mantissa format

- De-normalization from the memory

$$\pm \text{bbb...} * 2^{\text{e}}$$

De-normalization

Align to right, 1 more

$$\pm 0.\text{bbb...} * 2^{\text{e+1}}$$

Implicit

final Data

Data: -6.25

Ex: format: S | BE | M
1 bit 4 bits 5 bits

Given Data: -6.25

→ Stored into memory

(1) bbbb...)

Align to right, 2 more

S | BE | M
1 bit 4 bits 5 bits

1st digit 5th digit

① sign = -ve

= 1

② BE = AE + Bias
Bias = normal

$\Rightarrow +(2^{n-1})$

i.e. bbb...

fraction

Implicit (no need)
to store

Note

• To read the data from memory
de-normalization concept is used.

means align the data to right one time, to get the implicit digit into a fraction

Note
Store: 1.M
Read: 0.1M

Note
Store: 1.M
Read: 0.1M

De-normalization

Align to right, 1 more

$$\pm 0.\text{bbb...} * 2^{\text{e+1}}$$

Implicit

final Data

Data: -6.25

Ex: format: S | BE | M
1 bit 4 bits 5 bits

1st digit 5th digit

① sign = -ve

= 1

② BE = AE + Bias
Bias = normal

$\Rightarrow +(2^{n-1})$

$\Delta E: 0010$
 Bias: 011
 $BE: 1001$

Read:

Memory: $\boxed{1 \ 1001 \ 10010}$

$$\textcircled{1} \text{ sign} = 1 (-ve)$$

$$\begin{aligned} \textcircled{2} \text{ AE} &= BE - Bias. \\ &= 9 - (4+7) \\ &= +2 \end{aligned}$$

(3) Mantissa

$\boxed{M: 10010}$

$$\text{Data: } \frac{1}{\pm M} * \frac{S}{2^E}$$

De-normalization

Align to right of the binary point

$$\begin{aligned} &\rightarrow 0.10010 * 2^{+2} \\ &\text{Simplifies to} \\ &-0.110010 * 2^{-3} \end{aligned}$$

To repeat the decimal value, make the exponent value 0 by aligning the data to left of the binary point i.e.

$$\boxed{-110.010 * 2^{+3-3}} \Rightarrow \boxed{-6.25}$$

$\Delta E: 0110$
 Bias: 011
 $BE: 1001$

"0" padding at MSB

"0" padding at LSB

C-bit fraction

from the memory

Word size 32 bit

Byte mis

Sign bit

Exponent bits

Mantissa bits

Unused bits

Q1) consider the following hypothetical format used to represent the data

Format $\boxed{6 \ | \ BE \ | \ M}$
16 bit 3bit 12bit

- i) without normalization
- ii) with normalization

Storage	① sign: -ve		
② w/o normalization	② BE = AC + Bias Bias = normal $= +(\frac{2^{n-1}}{2})$ $= +(\frac{2^7-1}{2})$ $= +63$		
Data: $-17.75 * 2^{+12}$ This part is in Decimal $-10001.11 * 2^{+12} \rightarrow$ w/o normalization	AC : 0001100 Bias: 0111111 BE: 1001011		
Store the data	(3) M: $(000111)_H$		
	<table border="1"> <tr> <td>S BE M</td> </tr> <tr> <td>1bit 7bit 12bit</td> </tr> </table>	S BE M	1bit 7bit 12bit
S BE M			
1bit 7bit 12bit			

④ without normalization	Stored into memory		
Given data: $-17.75 * 2^{+12}$	<table border="1"> <tr> <td>S BE M</td> </tr> <tr> <td>1bit 7bit 12bit</td> </tr> </table>	S BE M	1bit 7bit 12bit
S BE M			
1bit 7bit 12bit			
	① sign = -ve		
	② BE = AC + Bias		
	(1. blob ...)		
	Align to right where		
	$-1.000111 * 2^{+12} \rightarrow$		
	$-1.000111 * 2^{+16}$		

Q4) Consider the following
format used to represent the
data format:

S BE M
1bit 7bit 12bit

Excess 2's complement

Data:- $-14.25 * 2^{+14}$

What is the hexadeclinal equivalent
when it is stored in the memory
using above format?

Soln

stored into a memory

Default storage
is with normalization

Storage: $-1.1101 \times 2^{+14}$

$(1.1101 \dots)$

$-1.1101 \times 2^{+14}$

$(1.1101 \dots)$

Align to right aligns

$-1.1101 \times 2^{+14} + 3$

$-1.1101 \times 2^{+14}$

$-1.1101 \times 2^{+14}$

$$\begin{array}{r} AE : 0000100001 \\ Bias : 1000100000 \\ BE : 1000100001 \end{array}$$

$\boxed{1|0000100001|1000100000}$

S BE M

C4A720)H

Ques] Consider the following format

used to represent the data

format: $\boxed{S|BE|M}$

1 bit 10 bit 1 bit

memory: data in the format

is $(\text{0x})_{16}$ hexa decimal

what is its decimal equivalent?

① Sign: true

→ 0 → positive/odd bits = 2^11 = 2048

Memory address = 1110101010000000

$AE = 66 - (+63)$

$= 3$

③ Manissa
 $\boxed{1.11001}$

① $Sign = -ve$

$= 1$

② $AE = AE + Bias$

Bias = excess

$= 2^{n-1}$

$= 2^{14-1}$

$= 2^{13}$

$= 8192$

$= 256$

AE : 0000100001

Bias : 1000100000

BE : 1000100001

$\boxed{1|0000100001|1000100000}$

S BE M

C4A720)H

memory (42.D0D)H or original data

↓

$\boxed{0|000010111010100000}$

S BE M

C12B01)H

(12B01)H

$S|BE|M$

(1bit)

BE (1bit)

M (12bit)

② $AE = BE - Bias$

Bias = normal

$= + (2^{n-1}-1) = + (2^{14-1}-1) = +63$

$= 3$

$\boxed{110010111010100000}$

↓

-159.01

(3) M: 11010100 0000

from the memory
 $\pm m * 2^{e}$
 $+ (-)$
 $(\frac{1}{2})^3 \times (\frac{1}{2}^2)^3 = 0.625$

Data: $[+1.110101000000 * 2^{-3}]$

To repeat decimal value make
 the exponent "0" by align
 the data to left shifts.

$$i.e. +110.10100 \cdot -$$

0.625

Ques] Consider the following memory data stored in the format given

format: S BE M
 1bit 7bit 11bit
 Bias
 Excess Bias

memory Data: (8CC00)_H

1	0	1111100	1100000000000000	M
S	BE	(7abit)	(12bit)	

Excess-Bias
 $= 60 - 64$

$$\text{Bias} = (2^{n-1}) \Rightarrow (2^{7-1})$$

(64)

(1) Sign: 1(-ve)

$$\begin{aligned} (2) & E = BE - Bias \\ & = 60 - 64 \\ & = -4 \end{aligned}$$

(3) M: 1100...-

To repeat decimal value make the exponent
 from memory

"0" Align to right 11times
 $i.e. -0.0000000000000000000$

-0.0000000000000000000	M
110000000000000000000	

final data
 -0.110000000000000000000

Value
 $-[(1 * 2^{-4}) + ((1 * 2^{-5}) + ((1 * 2^{-6})$

ans

$= [-0.110000000000000000000]$

- Range of floating point data
 - { min to max }
 - { All 0's to All 1's }
- to Analyze the range of the floating point Data. let us consider 32-bit floating point format as a reference b/c floating point data range is depend on floating format.

64bit	<table border="1"> <tr> <td>S</td><td>8E</td><td>M</td></tr> </table>	S	8E	M
S	8E	M		
32bit	<table border="1"> <tr> <td>S</td><td>8E</td><td>M</td></tr> </table>	S	8E	M
S	8E	M		
16bit	<table border="1"> <tr> <td>S</td><td>8E</td><td>M</td></tr> </table>	S	8E	M
S	8E	M		

Design: Scott T. O'Neil

32bit
② Exponent Range: Depend on "BE"

format as a reference. b/c floating point data range is depend on floating format.

6461

○
N

$$\therefore AE = BE - Bias$$

N-1227

Soil Exponent is 1.0
Range -1023 to +1024

$$\min_{\mathbf{B} \in \mathcal{E}} = \text{All } \mathbf{B}'s$$

$$\begin{aligned} &= 0 \\ \therefore AE &= BC - BH + BS \\ &= 0 - (-1023) \\ &= 1023 \end{aligned}$$

N

$$\Delta E = BC - B_{\text{IAS}} \\ = Q_{\text{OUT}} - (A1023) \\ \boxed{= +1024}$$

1

Suppose Excess Bias is used in the system

32bit	$S \backslash BE \quad \quad M$
16bit	8bit + 32bit

↳ Excess bias

Bias at 2^{n-1}

$\Rightarrow 2^{15-1} = 128$

Excess bias = All digital BE = $0000\ 0000\ BE$

$BE = 0 - (128)$

$BE = -128$

$\therefore BE = -128$

Note: In normal bias (+ve), Exponent and Bias (-ve) Exponent are more

Exponent are more but in excess

$\therefore BE = 255 - (-128)$

$\therefore BE = 255 + 128$

Note

- Smallest data size accuracy

Ka liya Excess Bias

• Images! Data size occupancy

Analysis
user view

• i.e Date:0.2375623

i.e Data: 0.0000002365
≈ 0.237

אלה נמלטו מארצם ב- 10.11.1947

Data: $2^{23} \text{ m/s} * 2^{-22.7}$
 $2^{23} \text{ m/s} * 2^{-126}$
 S.O./
 $2^{23} \text{ m/s} * 2^{+128}$

64bit format
 $\{ -1023 \text{ to } +1024 \}$ Range
 Biased exponent
 Data: $+0.00 \dots (1022)0s11$

$\{ 0 \dots 1023 \}$ position
 C. bias...
 Align to left : $+1.011 * 2^{-1023}$
 upto 1023 times

$$2^{52} M's * 2^{1023}$$

$$2^{52} M's * 2^{-1022}$$

$$\vdots$$

$$2^{52} M's * 2^{1024}$$

so high value of ms can't be represented.

- In 2's compl' range are not present bias exponent are stored in BE format so range of exponent depends on "BE".

$$1.000 \dots 0s11$$

$$2^{52} M's$$

$$2^{1023}$$

$$1.000 \dots$$

$$2^{1023}$$

Align to left

out of range
 so normal
 bias can't
 be applied

source can apply
 excess bias,
 we can adjust
 half

Biased exponent
 Biased exponent
 Biased exponent

Biased exponent

③ Mantissa Range

$$\text{if } \max'M' = \text{All } 1's \text{ in } M' \\ \text{if } \min'M' = \text{All } 0's \text{ in } M'$$

$$\Rightarrow 000 \dots (2^8)0s \Rightarrow 000 \dots (2^8)1s$$

$$\therefore \text{normal } M' = 1.M$$

$$= 1.0$$

Biased exponent

Align the data
 to left 123 times : $\underbrace{000 \dots (2^4)1s * 2^{-23}}$

to convert the fraction into
 $\Rightarrow (2^{n-1})$
 $\Rightarrow (2^{n-2})$

Generalize:
 min to max
 $\rightarrow (2^{-2} - \frac{\text{mantissa}}{\text{size}})$

$$(2^{-2} - \frac{\text{mantissa}}{\text{size}}) \Rightarrow (2^{24-1}) * 2^{-23}$$

$$(2^{24-1}) * 2^{-23} \Rightarrow (2^{24} * 2^{-23}) - (1 * 2^{-23})$$

$$(2^{24} * 2^{-23}) - (1 * 2^{-23})$$

$$(2^{24} * 2^{-23}) - (1 * 2^{-23})$$

(4) Range of a 32-bit floating point $\pm m \times 8^{\pm e}$

32bit	$S BE M$
32bit	8bit 23bit

plus Data Range is

$+ (1) * 2^{127}$ to $+ (2 - 2^{-23}) * 2^{+1023}$

-ve Data Range is,

$-(1) * 2^{127}$ to $-(2 - 2^{-23}) * 2^{+1023}$

64bit	$S BE M$
64bit	11bit 52bit

plus Data Range is

$+ (1) * 2^{-1023}$ to $+ (2 - 2^{-52}) * 2^{+1024}$

-ve Data Range is

$-(1) * 2^{-1023}$ to $-(2 - 2^{-52}) * 2^{+1024}$

Exponent underflow

- these condition will occur when the exponent is exceeding the maximum possible "the exponent. it becomes true when the data approach to 0".

How to handle the overflow & underflow?

$$32\text{bit} + 8\text{bit} = 32\text{bit} + \text{split}$$

so we set boundaries

32bit

Data :
size
So, flagane of not present required
not used by floating point
format IEEE754

1. To handle the overflow & underflow condition 2 exponent are reserved in the floating point format named as:-
 - ↳ min Exponent All 0's in BE
 - ↳ max Exponent All 1's in BE

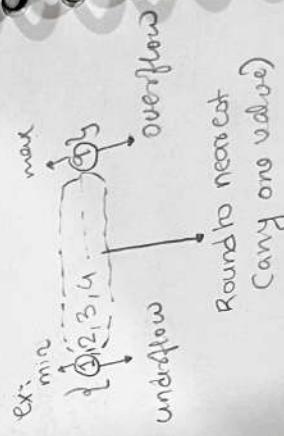
3 Rounding techniques are implement by using the special value to represent they Data. namely

- ① Round to "0" → underflow
- ② Round to "+∞" → the overflow
- ③ Round to "-∞" → the underflow
- ④ Round to "nearest" → Representable Data

Exponent overflow

- these condition will occur when the exponent is exceeding the maximum possible "the maximum possible" - the exponent. it becomes true when data is approach to "+∞" or "-∞".

2. Special value are defined by using that above exponent, to handle the overflow & underflow condition of 0, +∞ & -∞



ex: 32-bit format

S	BE	M
1bit	8bit	23bit

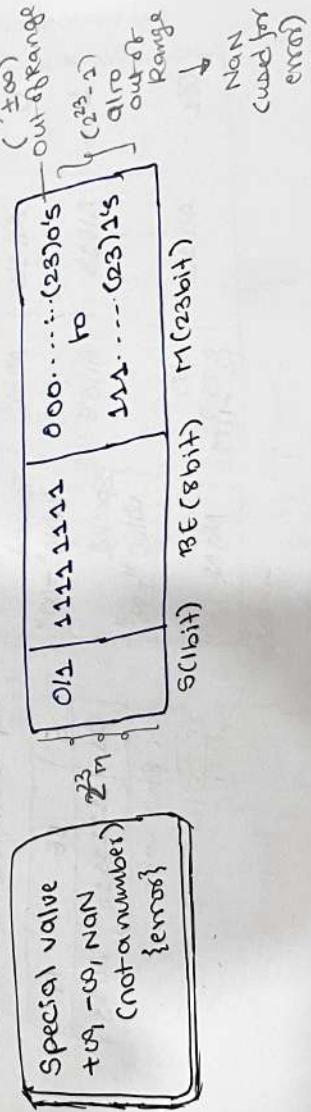
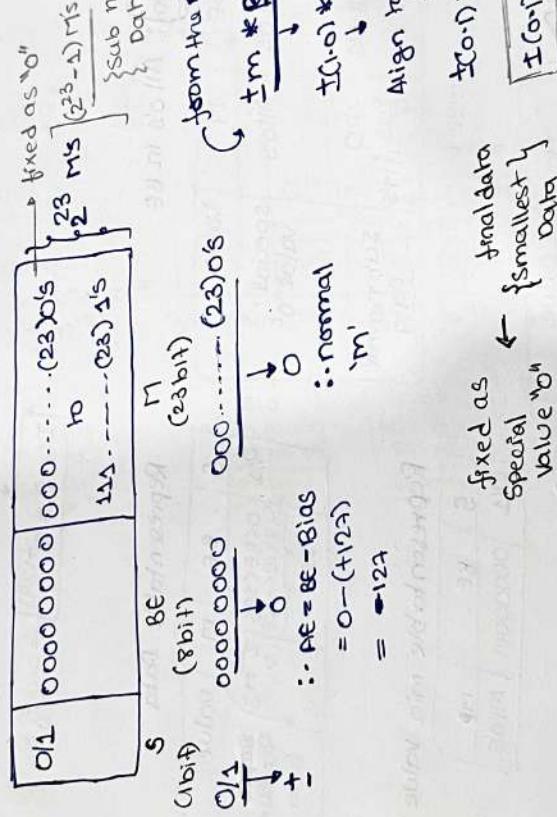
2⁸ Exponents

-127 to +128

Special Value "0"
 254 exponents
 are representable
 Data exponents

Special value "0"

{Min expo: All 0's in BE} Reserved for special value as a special exponent

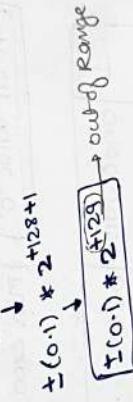


Sign	Exponent	Mantissa
0/1	11111111	0000...-0231011

265

$AE = BE - Bias$
 $= 265 - (127)$
 $= +128$

From the memory
 $\pm m * 2^{BE} \rightarrow \pm (1.0) * 2^{+128} \rightarrow$ De-normalization



Special expo: All 0's in BE

S	BE	M	Value
0/1	All 0's	All 0's	special value '0'
0/1	All 0's	#0 11111111	Sub normal data

means data which is having all 0's in exponent [all 0's in BE]

All 1's in BE

S	BE	M	Value
0	All 1's	All 0's	Special value "+∞"
1	All 1's	All 0's	Special value "-∞"
0/1	All 1's	#0 11111111	Nan.

Representable Data

S	BE	M	Value
0/1	0 < BE < 255	2 ^{3M-3} (All 0's)	Representable data
	254	2 ^{3M-3} (All 1's)	

Representable min value

S	BE	M	Value
0/1	00000000	All 0's	

Representable max val. ie

S	BE	M
0/1	11111111	All 1's

ex 8 bit code

000 special

001 smallest

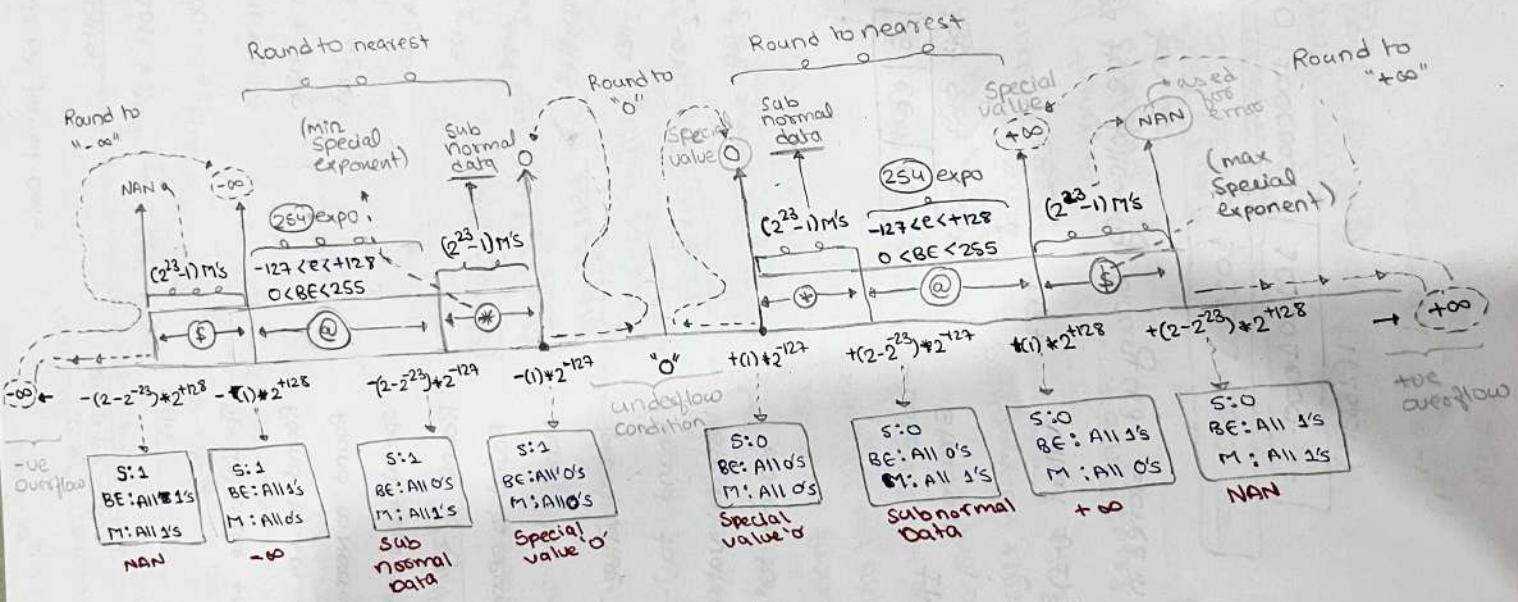
010 largest

111 special

④ Indicate min Exponent Data with all possible 1's

⑤ Indicate max Exponent Data with all possible 0's

⑥ Indicate the data existed b/w the min & max exponent with all possible M's

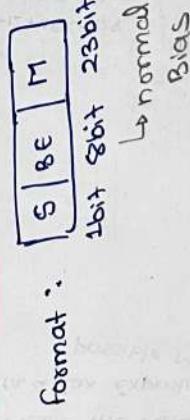


Eg: 32 bit format Data

Representative Value

Data	Representative Value
① +1.1011 * 2^{128}	---
② +1.000... * 2^{128}	---
③ +1.1011 * 2^{130}	Special value +∞
④ +1.1011 * 2^{120}	Round to +∞
⑤ +1.000... * 2^{-127}	Round to Nearest
⑥ +1.011 * 2^{-129}	Special value "0"
⑦ +1.1011 * 2^{-127}	Round to nearest (subnormal data)
⑧ -1.1011 * 2^{128}	---
⑨ -1.000... * 2^{128}	Special value "-∞"
⑩ -1.1011 * 2^{132}	Round to "-∞"
⑪ -1.000... * 2^{-127}	Special value "0"
⑫ -1.1011 * 2^{-130}	Round to "0"

Conclusion.



Data : $\pm M * 2^{\pm e}$
Structure
↳ normal = $(-1)^S (1.M) * 2^{BE-Bias}$

Ques] considers the following binary code stored in the memory using 32 bit format

Binary: 0 10000011 101101 000 ...
S(1bit) BE (8bit) M (23bit)

$$\begin{aligned} \hookrightarrow \text{normal}_2 &+ (2^{8-1}-1) \\ \text{Bias}_2 &= +127 \end{aligned}$$

Value: $(-1)^S (1.m) * 2^{E-510}$

$$(-1)^S (1.101101000\ldots) * 2^{131-(510)}$$

Decimal value required so make the exponent "0" by right align to left shift

$$16 \cdot 8 \cdot 2 \cdot 1 \cdot 0 \cdot 25 + 11011 \cdot 01000\ldots * 2^{131-4}$$

$$16 \cdot 8 + 2 + 1 + 0 \cdot 25 \Rightarrow 27 \cdot 25$$

Ans: $[+27.25]$

Floating Point Arithmetic

① Addition / subtraction

Step 1: Take 2 no. x, y

Step 2: Equalize the Exponents

Take the diff b/w the 2 exponents if diff is "0" then go to step 3 else align the smaller exponent mantissa to right upto the difference to equalize the exponents

later goto step 3
Step 3: Add / subtract the mantissa
Step 4: Normalize the result
Step 5: Store the result into a memory

② Multiplication

Step 1: Take two numbers x, y

Step 2: Add the exponents

Step 3: multiply the mantissa
Step 4: normalize the result
Step 5: store the result into a memory.

③ Division

Step 1: take two no. x, y

Step 2: subtract the divisor

Step 3: divide the dividend exponent from the divisor

Step 4: divide the mantissa

Step 5: normalize the result.

Step 6: store the result into a memory.

Ques: Consider the following 32bit floating point. Data & perform the multiplication

$$x: 2.5 * 2^{+8} \quad \text{Step 1: } x = 2.5 * 2^{+8}$$

$$y: 3.5 * 2^{+6}$$

Want result final product in memory in Hexa decimal.

$$\begin{aligned} \text{Step 2: } & (1.8) + (4.6) \rightarrow 14 \\ \text{Step 3: } & (2.5)_{10} * (3.5)_{10} = (8.75)_{10} \\ & (1000.11)_2 \end{aligned}$$

Step 4:

$$\begin{aligned} \text{final product: } & 1000.21 * 2^{+14} \\ & \downarrow \text{normalize} \quad (1.111\ldots) \text{ Align to right} \\ & \text{by 3 more.} \\ & 1.000111 * 2^{+14} \end{aligned}$$

Steps: Store the result into memory
format: 32bit format

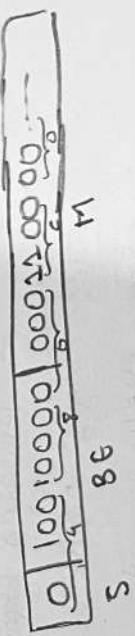
$$\begin{array}{|c|c|c|} \hline S & \text{Ex} & M \\ \hline \end{array} \quad \begin{array}{l} \text{8bit} \\ \text{23bit} \\ \text{Normal Bias = 127} \end{array}$$

(3) $M: 1.000111$

(1) sign = true
= 0

(2) $BE = AE + Bias$
 $AE: 0001000$
 $Bias: 0111111$

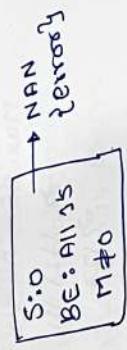
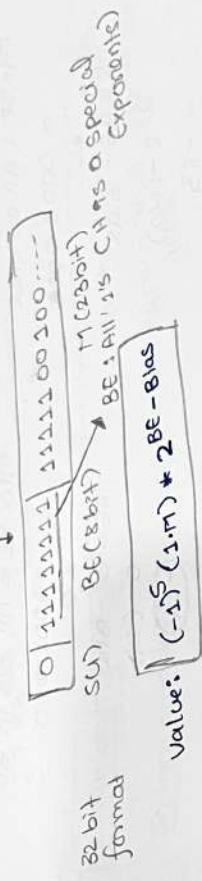
$BE: 10010000$



$$\text{Ans: } ((480C0000)H) //$$

Ques] Consider the following memory data stored in the memory using 32-bit format what is the value reported by the system

memory data : 0x7FFC8000
memory data: (7FFC8000)H



Ques] Consider the following data present in the memory using the given format . what is its value



$$\begin{aligned} & \text{Value: } (-1)^S \times (1.M) \times 2^{BE - Bias} \\ & \Rightarrow (-1)^1 \times (1.2) \times 2^{10111100 - 512} \\ & \text{Excess Bias} = (2^{\frac{10}{2}}) - \frac{510}{2} = 512 \end{aligned}$$

Ques] Consider the following format used to represent the data

- ① what is the range of exponents possible when the format is designed with



- ② normal bias
③ excess bias
④ report the mantissa (fraction) range
⑤ report the mantissa (fraction) range
⑥ report the data in normal bias
⑦ report the data in excess bias.

- Sol] ① a) normal Bias ② Excess Bias
- Bias = $(2^{n-1} - 1)$ Bias = $\frac{2^n}{2} \approx 2^{n-1}$
 $= + (2^7 - 1 - 1)$ $\Rightarrow (2^7 - 1) \Rightarrow 63//$
- $= +\boxed{63}$ Min BE = All 0's in BE
MinBE : All 0's in BE
 $= 0000000$ $= 0000000$
 $= 0$ AE = BE - Bias
AE = BE - Bias
 $= 0 - (+\boxed{63})$ $= 0 - (+63)$
 $\Rightarrow -63$ Max. BE = All 1's in BE
max : All 1's in BE
 $= 1111111$ $\Rightarrow 1111111$
 $= 127$ $\Rightarrow 127$
 $= 127 - (+63)$ $\therefore AE = BE - Bias$
 $= \boxed{+64}$ $= 127 - 64$
 $\Rightarrow +64$ $= \boxed{+63}$
so, Expo. Range : $\{-63 \text{ to } +64\}$ so, Expo Range : $\{-64 \text{ to } +63\}$
Generalization : -Bias to +Bias +1
Generalization : $2^{-Bias} \text{ to } + (Bias + 1)$
- ③ Mantissa Range (fraction)
- generalization :
Min 'M' = $2^{-2^{-2^{-\text{mantissa size}}}}$
Max 'M' = $2^{2^{2^{2^{\text{mantissa size}}}}}$
in float format
Mantissa size = 12 bit
Mantissa size = 39 / 21
- ∴ Range of fraction = $[1 \text{ to } 2^{-2^{12}}]$

(iii) Normal Bias Data

$$\pm(1) * 2^{-63} \text{ to } \pm(2 - 2^{-12}) * 2^{+64}$$

$$\pm M * B^{\pm e}$$

(ii) Excess-Bias Data:

$$\pm(1) * 2^{-64} \text{ to } \pm(2 - 2^{-12}) * 2^{+65}$$

Conclusion ↗

In the computer:

- only IEEE format present
- 32-bit format (float)
- 64-bit format (double)

Braille: 16

i.e.

S	BE	M
1bit	8 bit	6 bit

Excess -16

i.e.

S	BE	M
1bit	8 bit	6 bit

: Excess Bias

i.e.

S	BE	M
1bit	8 bit	6 bit

: Excess Bias

i.e.

S	BE	M
1bit	8 bit	6 bit

: Excess Bias

↓
S bit (default normal bias)

↓
M bit (excess bias)

In the exam Question:
• Hypothetical format are given

Default : Normal Bias

S	BE	M
1bit	8 bit	6 bit

Excess Bias

S	BE	M
1bit	8 bit	6 bit

Customized Bias

S	BE	M
1bit	8 bit	6 bit

not a normal

not excess

Module : 2

Based on the program storage system Arch.

Computer Architecture

2nd

Von-Neumann Architecture
Single memory Archⁿ

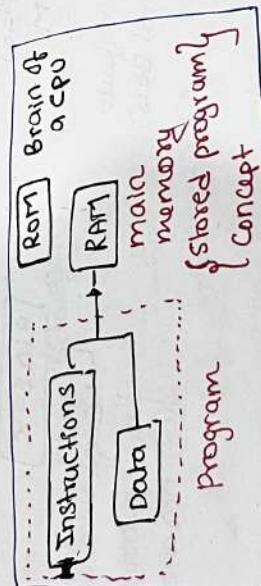
main memory is RAM

2) Harvard Arch
Multimemory Archⁿ

Von-Neumann Architecture • In these Architecture, application program is always required in Main memory to execute

- main memory is RAM, it is a volatile memory
- It allows read & write operation. so we can change the application program in Main memory, if required
- In these architecture CPU always execute main memory part of a program

In these architecture, we can run different Application programs in the same H/w (within same H/w you can see multiple behaviour)



used in the personal computers Design

so, $2^{10} \times 2^{10}$ cells

64 KB cells

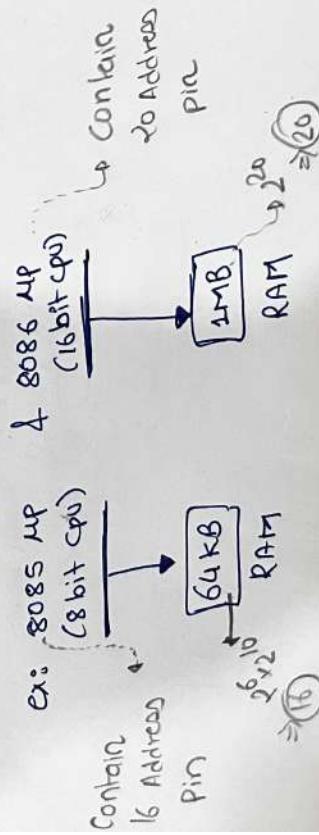
4 each cells contain

≈ 8 byte (say default)

so, 64 KB

cell size = 8 bit

• When you change
program in main
memory after
execution
behaviour
change in the H/w



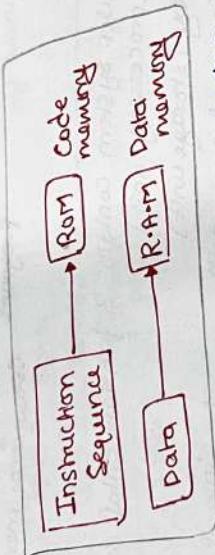
Harvard Architecture

not in syllabus

- In these Architecture program logic & its operational data both are present in separate memory called as code memory & data memory respectively

Code memory is permanent memory (R.O.M) & Data memory is volatile

- memory that is (R.A.M)
- In These architecture CPU always execute predefined program in the code memory on a different data element [Hence with a fixed customized behaviour]



- when application doesn't change the requirement application requirements is static then use Static Hardward
- Implementation as a microcontrollers design like ROM 128B RAM
- ex: 8051 microcontroller

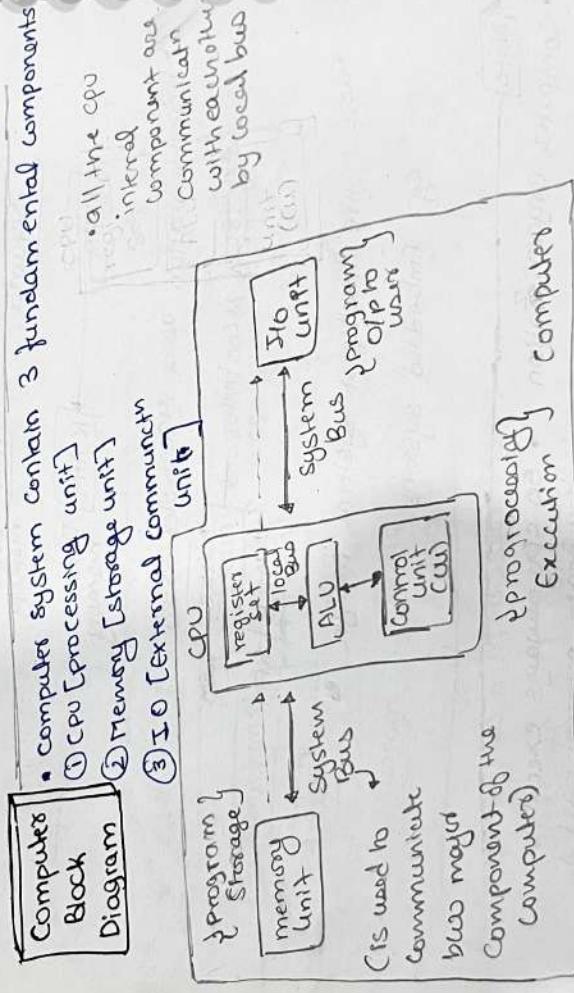
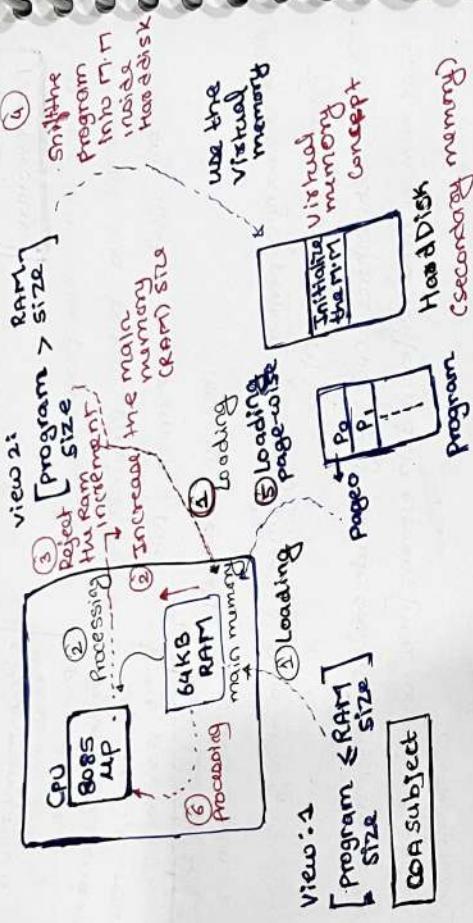
used in the intelligent systems design

e.g: Embedded systems

- Note**
- So CPU always executes the main memory part of a program by generating the physical Address
 - Computer Organization
 - subject describes the implementation of a von neumann Architecture

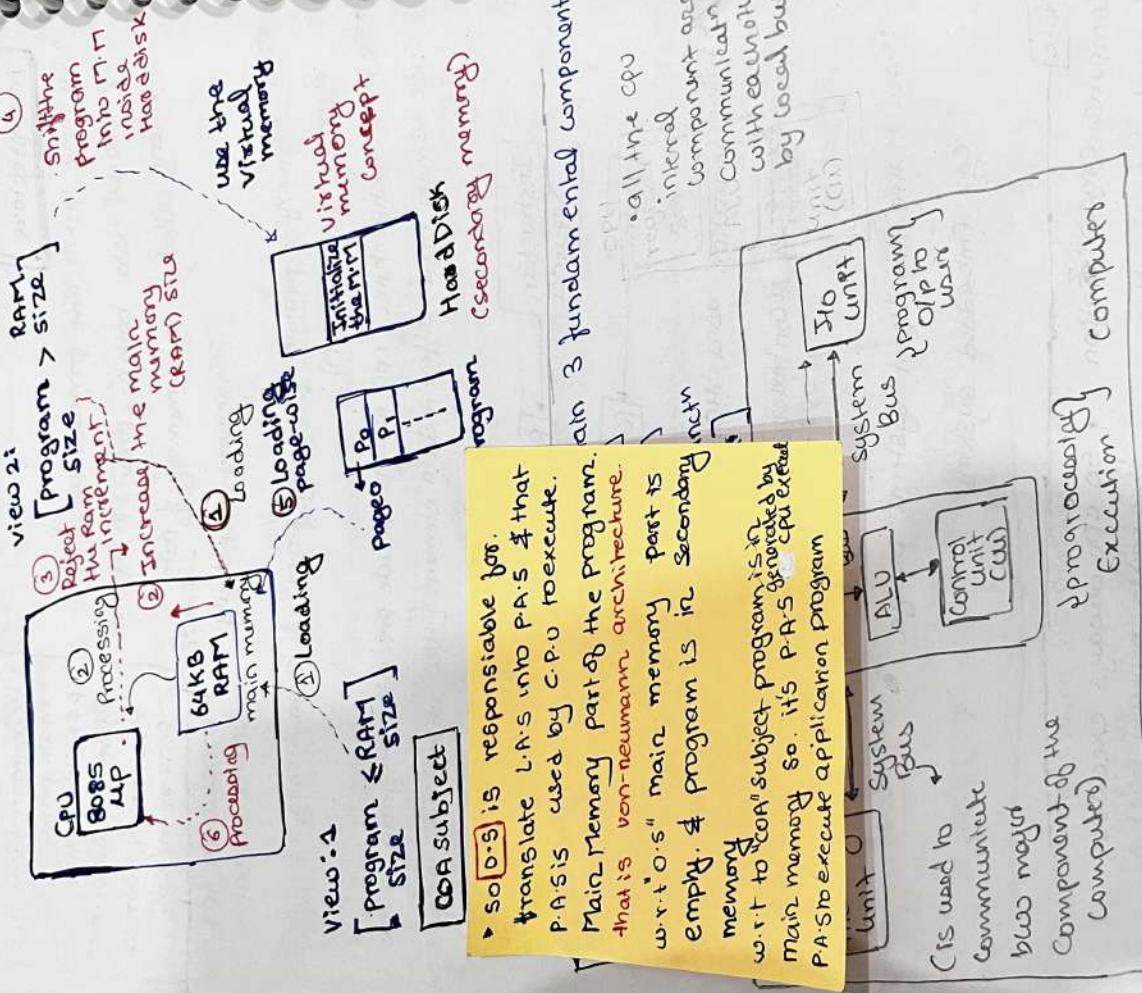
Overview of a System

OS Subject



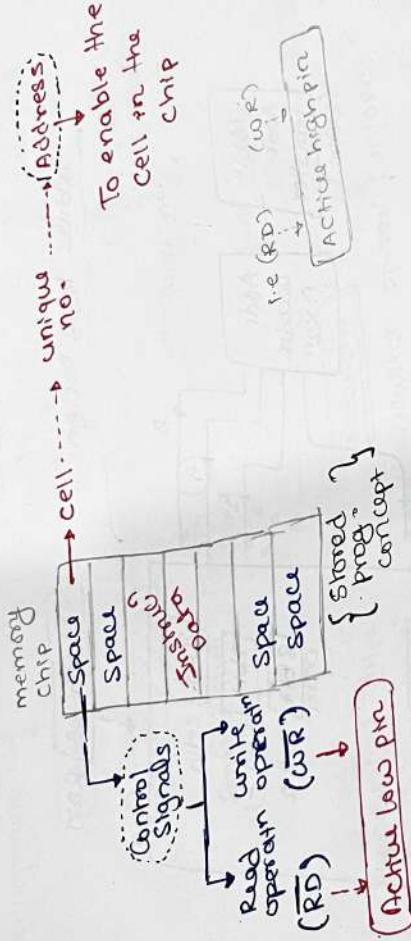
Overview of a System

OS Subject

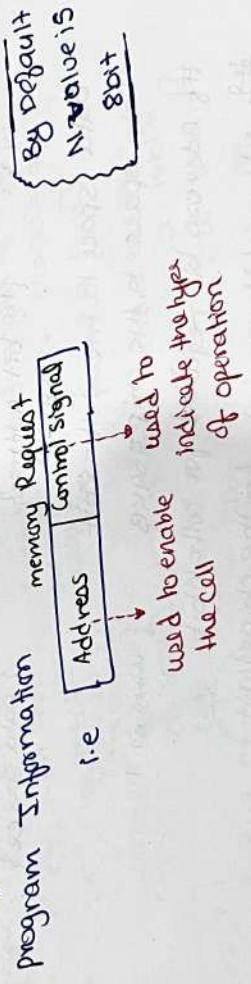


Memory Organization

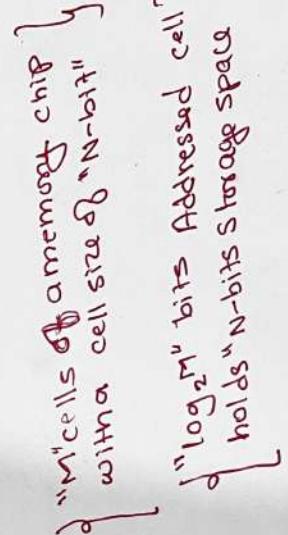
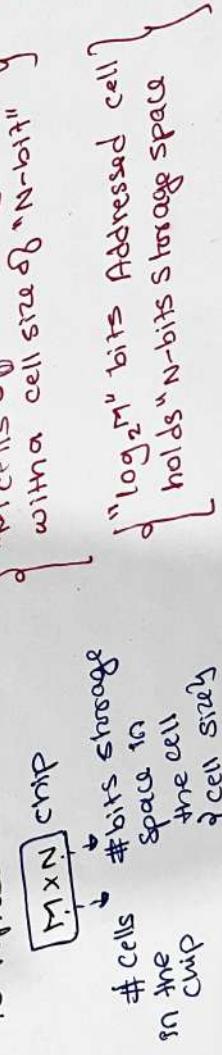
- memory is storage component in the computer, used to store program
- memory chip is divided into equal size part called cells.
- memory cell has identity with unique no. called address
- memory chip recognises the control signal to indicate the type of operation i.e. read or operation & write or operation

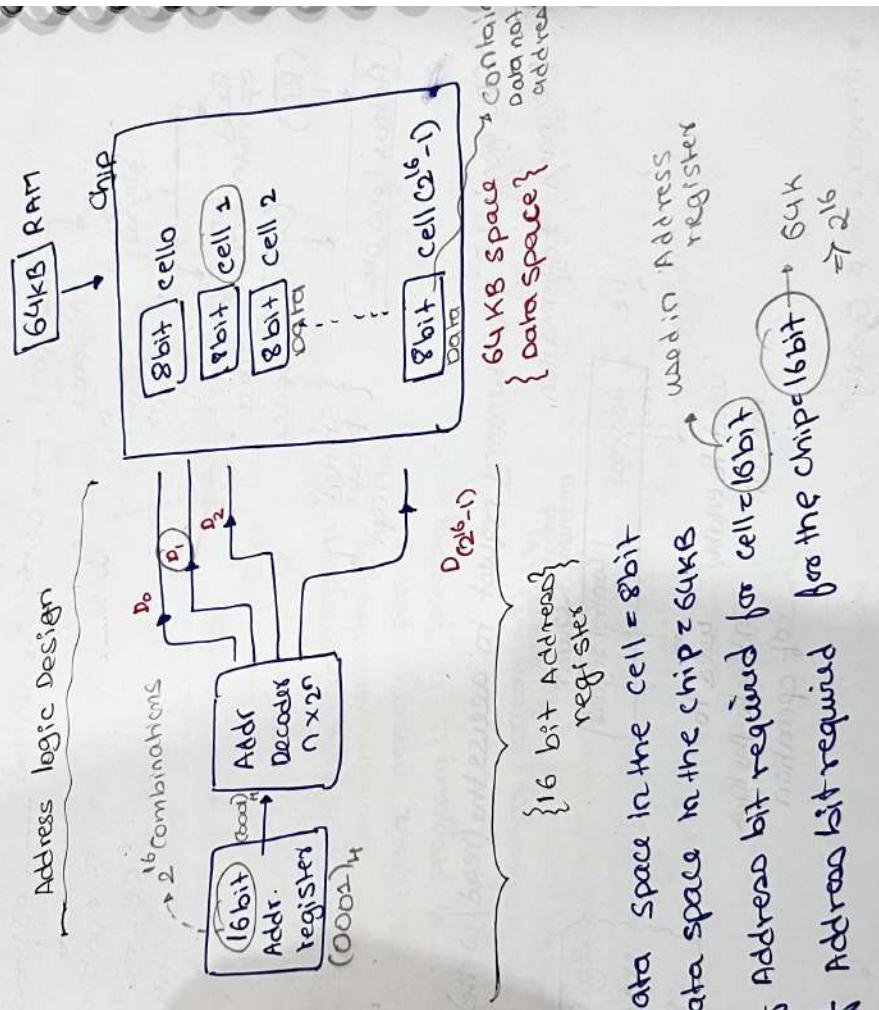
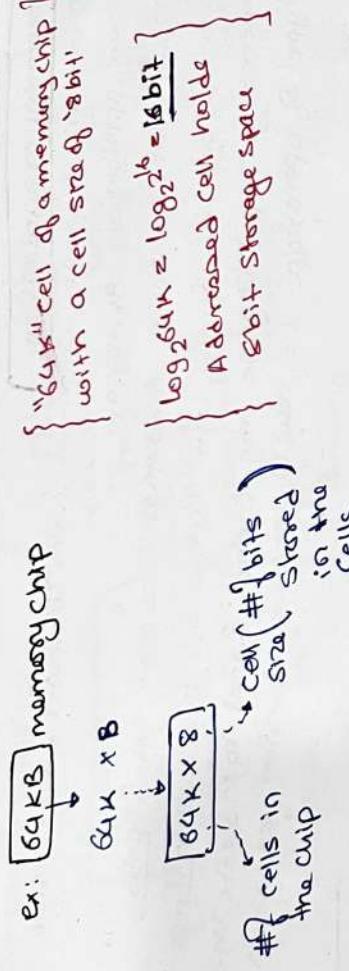


- CPU generates the memory request to access the (read/write) the program information

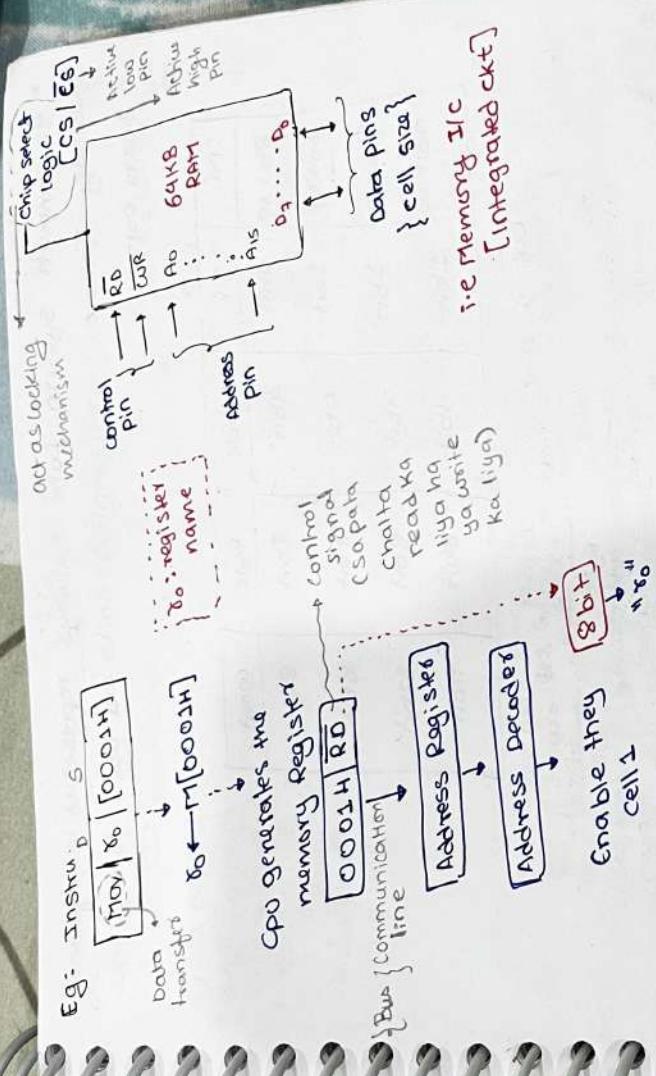


- memory chip Contd





- Data space in the cell = 8bit
 - Data space in the chip = 64KB
 - # of address bit required for cell = 1bit
 - # of address bit required for the chip = 16bit
- $\Rightarrow 2^{16}$



Notations

Hypothetical chips

- 4K \times 4; 12bit Addr
- 256M \times 6; 28bit Addr
- 4G \times 8; 32bit Addr
- $2^n \times 1$; x bit Addr
- $M \times N$; 108bit

depend on the cell size

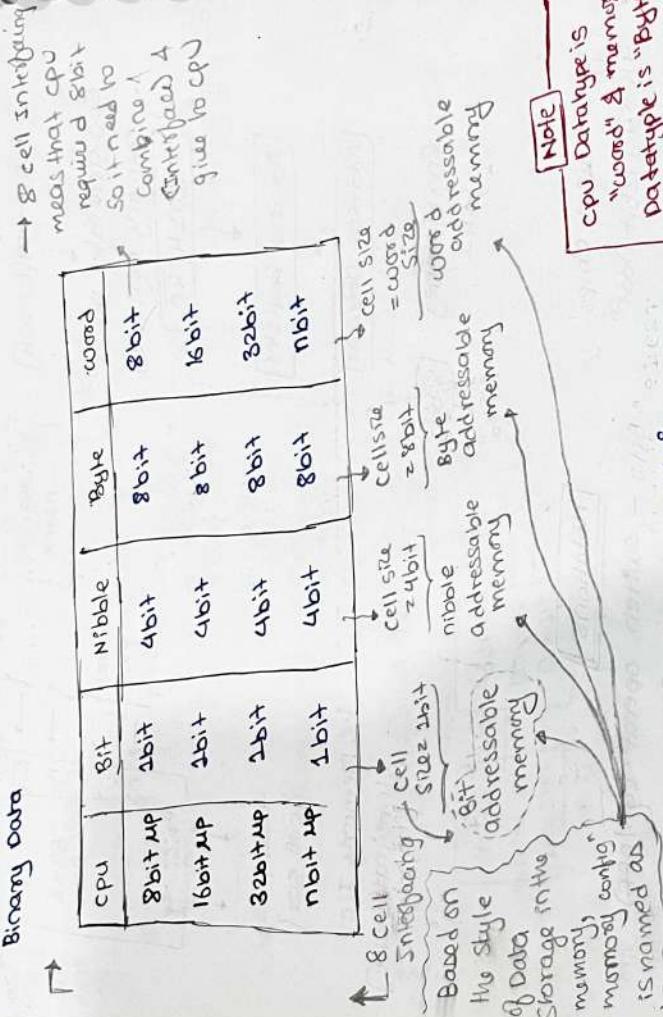
$$\begin{aligned} 1K &= 2^{10} \\ 1M &= 2^{20} \\ 1G &= 2^{30} \\ 1T &= 2^{40} \\ 1P &= 2^{50} \end{aligned}$$

Notations

- Bits - switch operate (on/off)
- Nibble - 4bit code
- Byte - 8bit code
- word length of a "CPU"
- word - word processed in the CPU
- byte - 8bit bit processed in the CPU
- cell - cell in the RAM
- chip - chip of a RAM

8bit CPU: word size = 8bit
16bit CPU: word size = 16bit
32bit CPU: word size = 32bit

- A computer system data is always represented in a format binary following referential units and used to refer the binary data



Note

CPU Datatype is "word" & memory
datatype is "Byte"
both are synchronised
using the concept
Interfacing

ex: 4K x 1 ; 12-bit Address → Bit Address
4K x 4 ; 12-bit Address → nibble Address
4K x 8 ; " → Byte Address
4K x 10 ; " → word Address

Ques] Consider the following memory chip design used in 32bit

hypothetical CPU

- 256K x 8
- 64M x 4
- 32G x 32
- 128M x 1

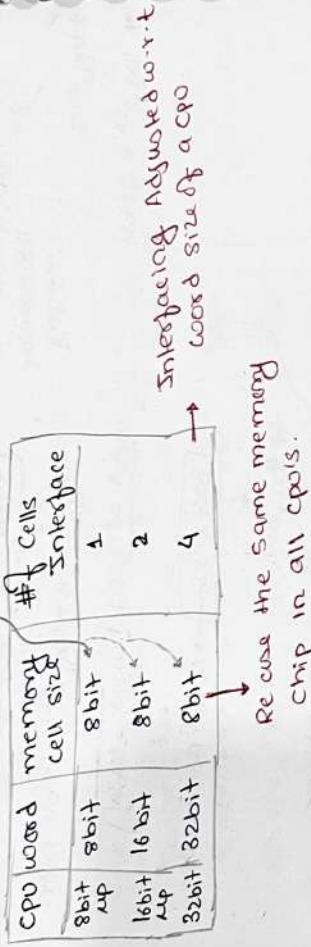
what is the size of a
bit Address, nibble Address
Byte Address, & word Address
w.r.t. the above chips.

Bit Address

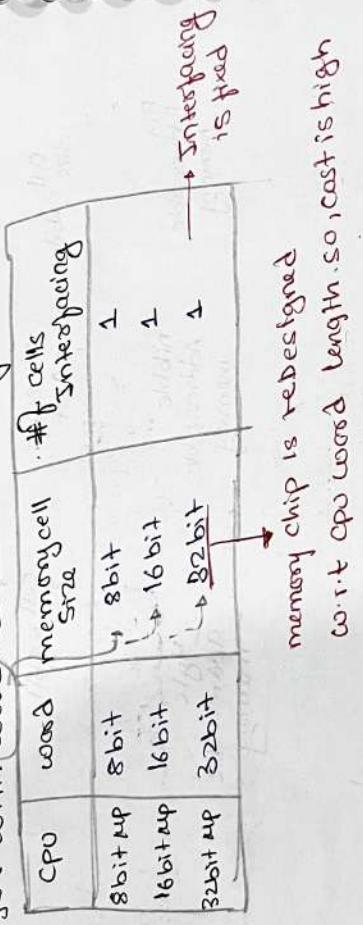
128M x (1) Bit Addressable
memory

$\log_2 128M \Rightarrow \log_2^{2^9} \neq 27$ bit Address

Sys. design with Byte Addressable memory



Sys. with word Addressable Memory



Note:

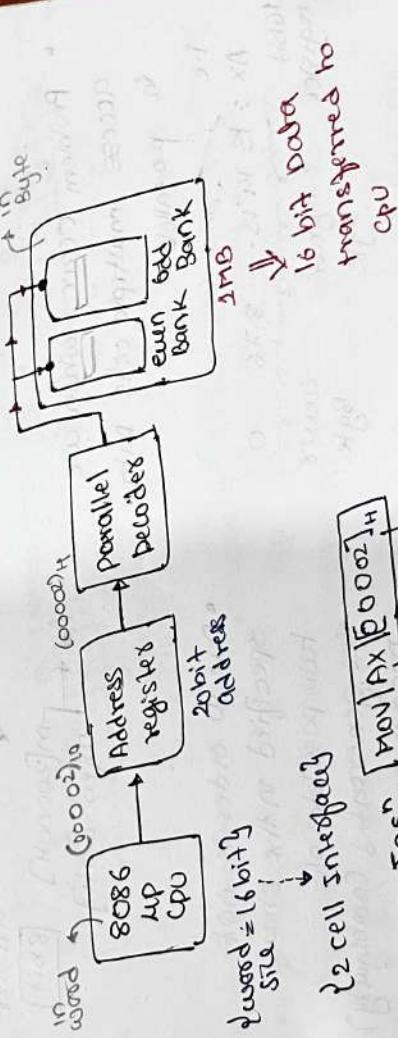
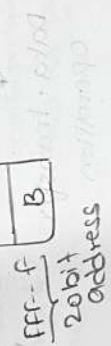
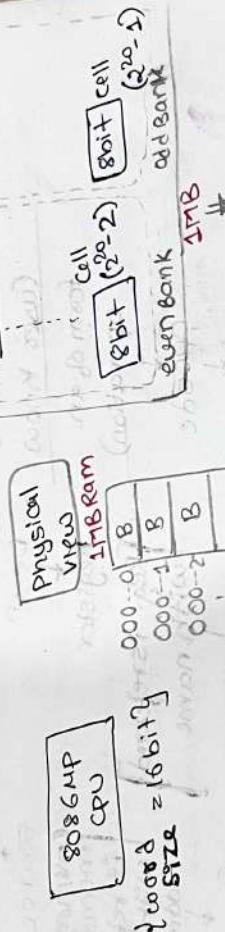
1. Default memory control in the system design is Byte Addressable means data is stored in the memory in byte-wise sequence So, memory data type is **Byte**

2. CPU always perform the operation on a word formatted data so, CPU data type is **word**
3. CPU data type is synchronized with a memory data type by implementing interfacing block them i.e. multiple memory cell are interfaced to CPU w.r.t word size of a CPU



Q Storage : Bytewise
Accessing : word-wise

To analyze the above concept
Let us take 16-bit MP
16 bit MP supports 1MB RAM



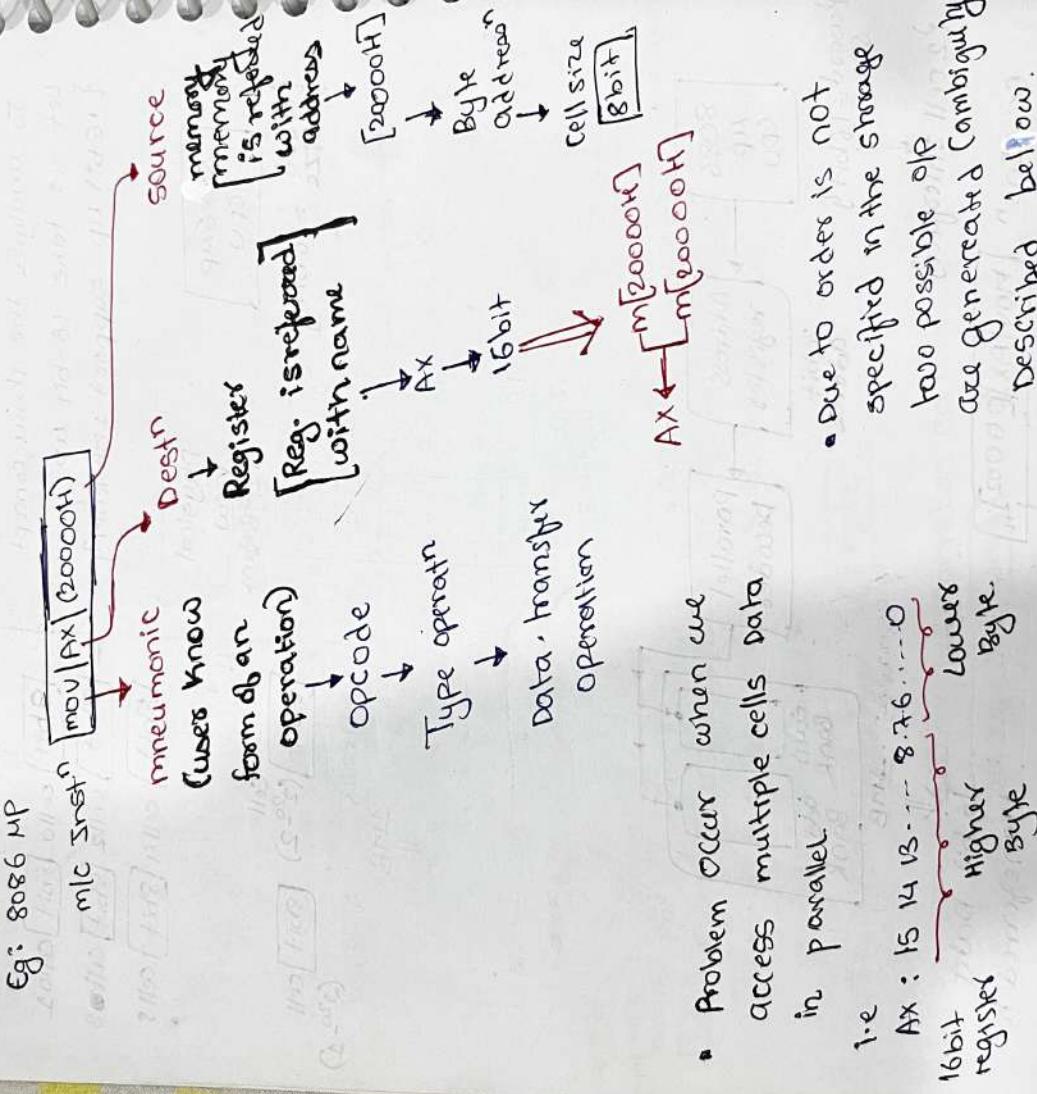
Ex: **Ins** \rightarrow **M[0000002H]**
Register name \rightarrow **AX**

\downarrow
2 cell transfer
 \downarrow
16 bit data
 \rightarrow **M[0000003H]**

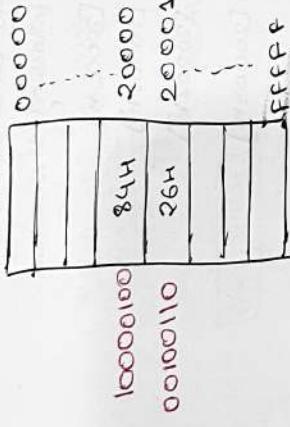
the two unit words in
Byte is synchronized
using pipeline

Are no the word
length no. of cell
are accessed

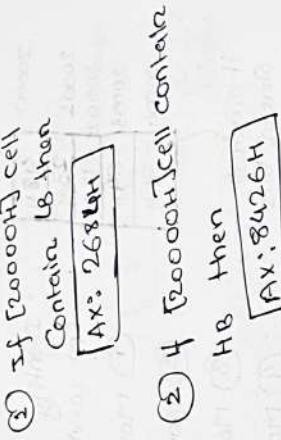
Eg: 8086 MP



Assume,
memory
Contents.



Ex:-



- to handle the above ambiguity, memory address interpretation (Endian-ness) mechanisms are used.

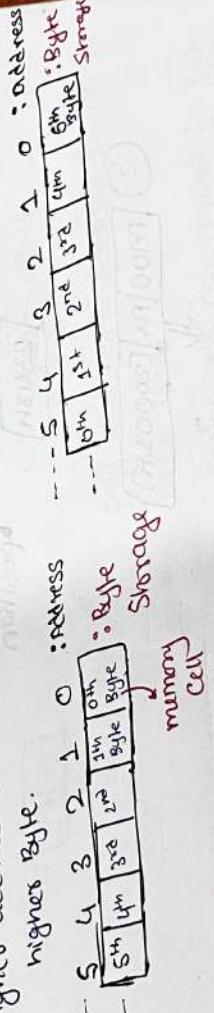
These mechanisms shows order of data storage in the memory.
These are of 2 kind

→ Little-Endian

→ Big-Endian

- In the Little-Endian lower address contains lower byte & higher address contains higher byte.

Higher Byte.



Instruction produces the following:

above instruction produces the following
by default address DIP

c.i.e. $\text{AX: } 26H \quad 84H$ → data
is Little-Endian

above instruction produces the following
by default address DIP

c.i.e. $\text{AX: } 26H \quad 84H$ → data
is Little-Endian

Lower
Byte

HEX

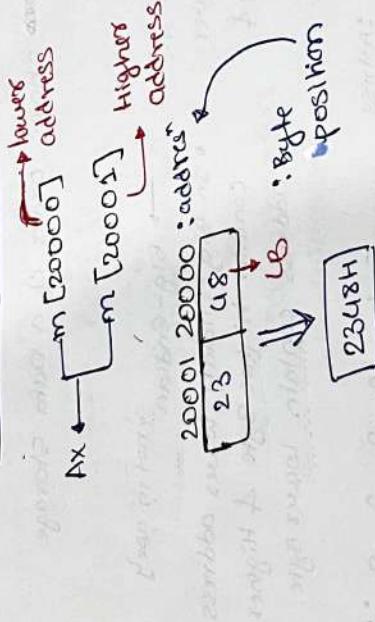
Ques] consider the following memory contents.

20000	48H
20001	23H
20002	92H
20003	64H
20004	71H

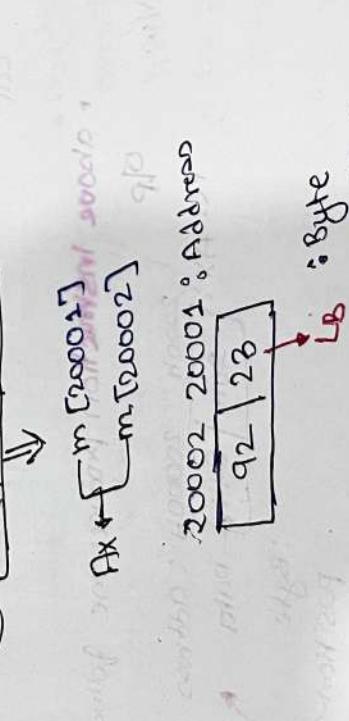
It is actual
Store in bits
Store in [01000111]

16bit CPU: 2 cells accessing required

① $\boxed{\text{MOV AX, [20000H]}}$

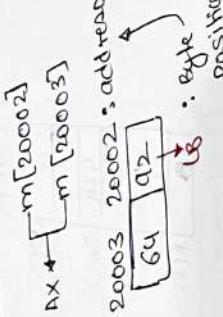


② $\boxed{\text{MOV AX, [20001H]}}$

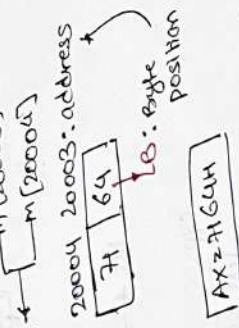


AX: $\boxed{9223H}$

(3) $\text{mov}[\text{ax}] \rightarrow [2000024]$



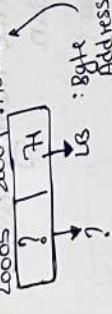
(4) $\text{mov}[\text{ax}] \rightarrow [2000035]$



In 8085 up, Endian-ness Tech
register @ not

8085 up → split up
8 bit copy

word size = 8bit



memory config: default
Byte Addressable
CellSize = 8bit

only one cell is accessed. so, no

order issue
∴ Endian-ness not required for accessing.

- ∴ Data accessing may not required little endian but string required.
- But, By default every memory chip is designed with a Little - Endian.

Ex: 8085 M/P, (8 bit CPU)

M/C instruction :

LDA	[2000H]
-----	---------

load

accumulates.

ACC \leftarrow [2000H]

Op:

ACC = 48H

\Rightarrow so 3 cell required

Ex: Instr: [LDA | 2000H] 3 byte instr stored in the memory with a

Starting Address of [1000H] show the storage sequence

Storage memory

1000	LDA
1001	40
1002	20
1003	
1004	

In the instruction storage opcode is stored information
first & rest of the information
is stored in the memory with
Little-endian
opcode storing at

1002 1001 1000 : Address

20	40	LDA
----	----	-----

: Byte
and 1000

position more addressable
bit number right to left
, starting from
bottom address of instruction
1000 1001 1002
1003 1004
1005 1006 1007
1008 1009 1010
1011 1012 1013
1014 1015 1016
1017 1018 1019
101A 101B 101C
101D 101E 101F

sys old

- word size = 64bit
- memory size = 256 kB

cell size → byte addressable

$$\text{Address size} = \log_2 \# \text{cells}$$
$$= \log_2 256 \Rightarrow 8 \text{ bit}$$

(CPU interface with 8 cells of a memory chip)

Enhance memory

sys new

To design the memory

- word size \neq 64bit
- memory config = word addressable
- (cell size = word)
size
- 64bit

cells in = 256K cells $\Rightarrow 2^5$ cells

Memory chip 8

$$\text{Address size} = \log_2 2^5 = 5 \text{ bit}$$

∴ no of bits saved = (18 - 15) bit
 $\Rightarrow 3 \text{ bit} //$

Q1 consider 32bit hypothetical CPU which support $128M$ word memory.
 System is enhanced with a byte addressable memory. How many address bit are to address the complete chip in the enhanced system?

Sys new

- word size = 32bit
- memory config = byte Addressable

(cell size = 8bit)

To design the memory

cell space of 8bit, split the old memory chip

Cell into 4 sub cell

Sys old

- word size = 32bit
- memory size = $128M$

[cell size = word size]

$$\# \text{Address bit} = \log_2 \# \text{cells}$$

$$= \log_2 128M$$

$$= 27 \text{bit}$$

CPU interfaced with only
 1 cell of a memory

$$\# \text{cells} = 128M \text{ cells} * 4 \text{ cells}$$

$$= 2^{29} \text{ cells}$$

$$\text{So, Address size} = \log_2 2^{29} = 29 \text{bit}$$

CPU interfaced with 4 cells of a memory

chip

Address pins in the new system

$$= 29 \text{bit}$$

$$\frac{\text{Address size}}{\text{Address size}} = \frac{29}{27}$$

Complexity reduction of 27%
 Complexity of 11.5% less

$$11.5\% = 20\%$$

$$f(24-8) = f(16) = 2^{16}$$

Wish

- Q) consider 16 bit hypothetical CPU which support 64MB memory space.
- System is enhanced with a CPU word length of 64 bit
- mem. containing 2 addressable bytes
- word size 2⁶⁴ bits
- mem. capacity 2⁶⁴ * 2⁶⁴ bytes
- 16 bit addressable memory
- 8 cells per memory cell
- To design memory cell
- size of 8 bit, merge the 8 cell into 1 cell
- 2⁶ = 64M
- Address = $\log_2 64M$
- $\log_2 2^6 \Rightarrow 26$ bit

Adjustments in the Enhanced System

~~sys~~ new system

- word size 64 bit
- mem. containing 2 addressable bytes

~~mem.~~ Good size
Good size

~~cell size~~ = 8 bit
cell size = 8 bit

(CPU Interfaced with 2 cells)
of a memory chip

$$\# \text{cells} / \text{function} = \frac{2^6}{8 \text{ cells}} = 2^3$$

$$\# \text{cells} / \text{function} = \frac{2^6}{8 \text{ cells}} = 2^3$$

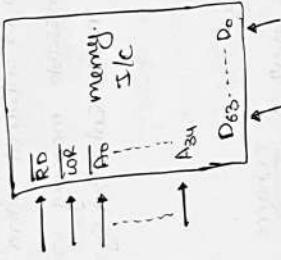
CPU interfaced with only 1 cell
(CPU interfaced with only 1 cell)

so, in the new system

- CPU data pin \downarrow from 64 bit to 32 bit
- Address pins are \uparrow from 26 bit to 23 bit
- memory interfacing \uparrow from 2 cells to 1 cell

old system
new system
enhanced system

Q) Consider the following memory chip configuration and answer the following questions:
 a) What is the memory capacity? (in MB)?
 b) How many bits are present?



Address pin
in the memory = $\frac{33}{35}$ pins
Chip

Data pins in memory = $\underbrace{D_{63} \dots D_0}_{64\text{bit}}$
chip (cell size)

: memory size = 2¹⁶ address cells

ମୁଦ୍ରଣ

$$\begin{aligned} & \because 2^{35} * 88 \\ & \Rightarrow 2^{38} 88 \\ & \Rightarrow \boxed{2^{35} * 64 \text{ bit}} \\ & \Rightarrow \boxed{2^{35} * 25600} \end{aligned}$$

⑥ considers the following memo

25c Design

what is the memory capacity in

Byle

Address pins: A_{35-A_0}
36 bits

Data pins in
the mem. chip: not given.

10

$$\Rightarrow 2^{36} \text{ cells} = 2^{36} 3 \Rightarrow \underbrace{2^6 \times 2^{30}}_{64,576} 3$$

8bit

Q1) consider the following memory chip used in the 64bit multiplexed CPU.

How much memory possible in the CPU in bytes of how many memory cells are interfacing b/w the CPU & memory.

$$\text{So, } \text{Address pin} = \underbrace{\text{A}_{31} \dots \text{A}_0}_{32\text{bit}} \quad \text{Data bus} = \underbrace{\text{D}_7 \dots \text{D}_0}_{8\text{bit}}$$

in memory
chip (cell size)

memory size = 2 address cells

$$\Rightarrow 2^{32} \text{ cells.} \Rightarrow 2^{32} \times 8 \text{ bit}$$

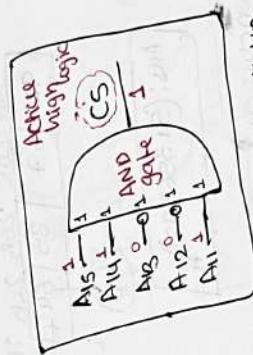
$\Rightarrow 4 \text{ GB}$

So, memory is byte addressable

$$\text{CPU word size} = 64 \text{ bit} \quad \text{8 cells}$$

Interfacing:

Q2) consider the following chip select logic used in the 64KB memory. What is the range of addresses in hexdecimal addressed by the line above chip select logic?



memory size = 64KB

$$\text{A}_5 \text{ A}_4 \text{ A}_3 \text{ A}_2 \text{ A}_1 \text{ A}_0 \text{ A}_7 \text{ A}_6 \text{ A}_5 \text{ A}_4 \text{ A}_3 \text{ A}_2 \text{ A}_1 \text{ A}_0$$

chip select logic

$$\begin{matrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

$$\begin{matrix} 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{matrix}$$

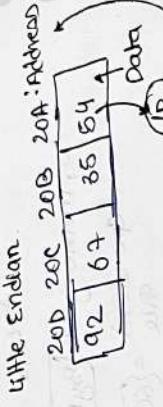
So Range $\left(\begin{matrix} \text{C} & \text{FF} \\ \text{0} & \text{0} \end{matrix} \right)_H$

- a) consider 32bit hypothetical CPU which is accessing the data from the starting Address (20A)_H. memory contents is as follows:

208	26H
209	48H
20A	3AH
20B	85H
20C	67H
20D	02H

- (a) Little - Endian
- (b) Big - Endian

- Starting address: (20A)_H



position

Ans: (02678548)_H

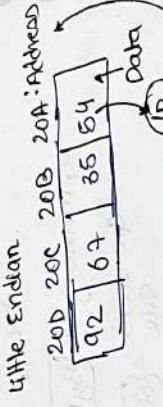
- 32bit CPU

$$\text{Word size} = 32 \text{bit}$$

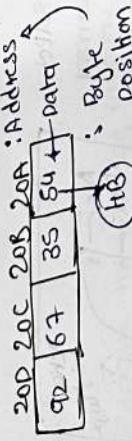
$$(\text{Data size})$$

- 4 cells accessing required

B/C Byte Addressable memory



Big-Endian



Ans: (3A856702)_H

Conclusion

- So, physical information is stored in memory Byte By Byte
- But CPU accessing information from memory in word by word.

- System design with Little-Endian, anyone.
- If cell you can access but system produce only 1 word.

Pin structure of 8085
↳ 40 pins available

CPU Pin Structure

[CPU chip structure]
CPU contain set of pins used to perform the operation on a

External components i.e. memory & I/O.

Externally connected components divide into 3-groups

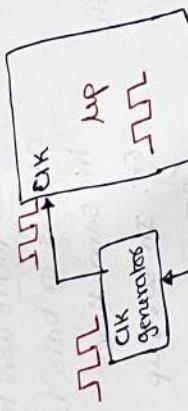
Address pins

Active high pin

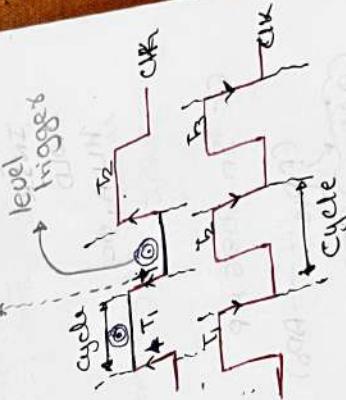
Time multiplexed pins.

Time multiplexed pins with a constant frequency clock to control the various operation.

CPU is operating with a constant frequency clock to control the various operation.



constant frequency



clock cycle time means, time

required to transfer

the clock pulse from rising edge to

rising edge to falling edge to

falling edge. It depends on CLK frequency

clk cycle [CPU cycle]
Time state [clk period].

Cycles $\propto \frac{1}{\text{clk frequency}}$

Ex: 1 GHz CLK is used in the CPU.

$$\text{Cycle Time} = \frac{1}{1\text{GHz}} \text{ sec} = \frac{1}{10^9} \text{ sec} \approx 10^{-9} \text{ sec}$$
$$\approx 1 \text{ ns}$$

• Active low pin: these pins enable when the CLK pulsation is low state.

[Pin name]

Ex: $\overline{\text{RD}}$

$\overline{\text{WR}}$

IN1A, etc.

- True multiplexed pin
- These pins work to connect the address & data but not at the same time

• Active high pin:

These pins is enable when the CLK state is in high state.

[Pin Name]

ALE

INIR

HOLD

HLDA, etc.

These pins is enable when the CLK state is in high state.

[Pin Name]

(AD₀ --- AD₇)

Address

Data

"8" Data pins are time

multiplexed with

Address pins

• Advantage is that no. of pins are reduced

in the CPU.

(AD₀ --- AD₇)

Address

DATA

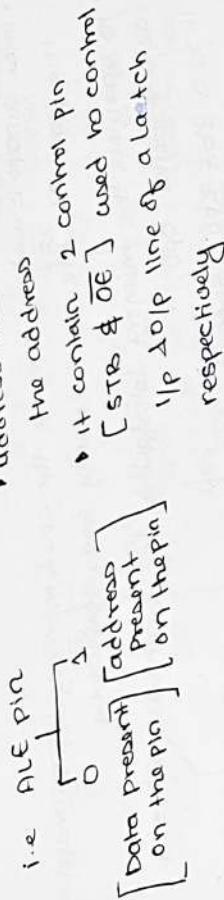
Then how to separate?

using ALE

(Address Latch enable)

we manage the addressing pins on the time multiplexed pins.

address latch is used to lock



i.e (STB) pin → 0 ; OLP line are disable
1 ; OLP line are enabled.

strobe

Address high
Pin i.e (STB) pin → 0 ; OLP line are disable
1 ; OLP line are enabled.

Address low
Pin i.e \overline{OE} pin → 0 ; OLP lines are enabled
1 ; OLP line are disabled.
OLP enable



SRB
Address
latch.

OLP lines.

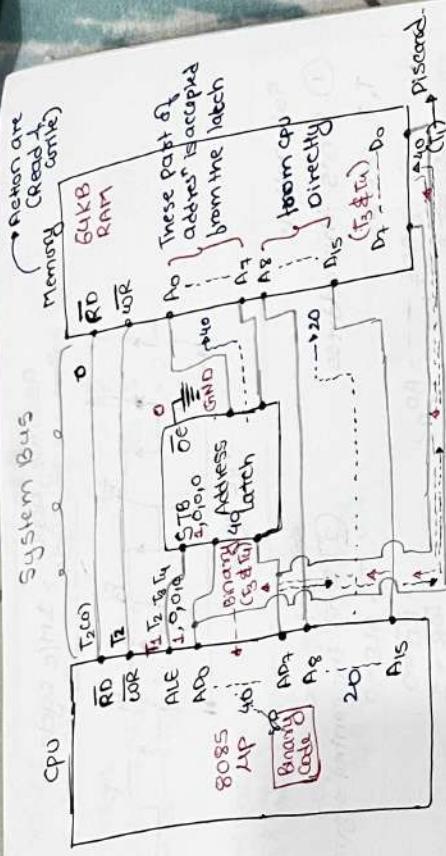
IOP lines.

connection of both the pin from CPU & memory is called as interfacing.

- Memory Interfacing
- these concept is used to integrate the CPU & memory chip by mapping the pins b/w the CPU & memory body respectively
- to analysis the memory interfacing, let us consider 8085MP as a reference CPU.
- It is a 8 bit CPU, support 64KB main memory



- Difference
 - CPU contains time multiplexed concept to reduce the no. of pins in CPU chip. word size will be fix
 - memory doesn't have time multiplexed. concept b/w memory chip will be used in any CPU so you can't fix time multiplex so memory will be reuse.
- these connection is known as interfacing
 - we just connect line but in real computer these line are present in system bus.
 - Connection b/w CPU & memory chip.



- why \bar{DE} is made to ground (GND)?
b/c to make op line enable (active) always.
[to make info "always active"]
- why \bar{DE} is made to ground (GND)?
b/c any address that comes to latch should always go to memory.
- why are we making pipeline always active?
b/c any address that comes to latch should always go to memory.

"no use of Data if latch is not enable."

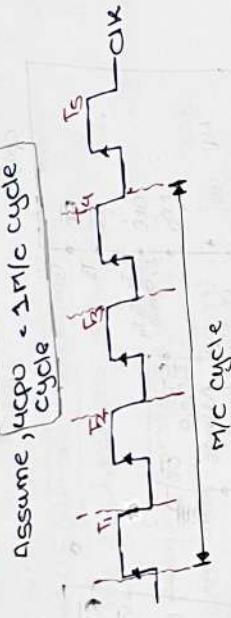
Action
Read operation →

mlc instruction: [LDA | BWD] ; load
Acc ← n[2040] →
Acc accumulates

CPU generates the memory request

[2040H | RD]

71c cycle required [#cpu cycles required for memory I/O operation]



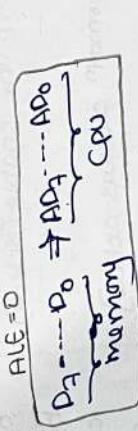
Sub actions
① Sends the address

T_1 : $ALE = 1$
 $40 \Rightarrow AD_7 \dots AD_0$
 $20 \Rightarrow A15 \dots A8$

② Send the control signal.

T_2 : $ALE = 0$
 $RD = 0$
 $\overline{WR} = 1$

③ Read the data.
 $(T_3 \& T_4)$:
 $AD \dots D_0 \Rightarrow AD_7 \dots AD_0$
 memory



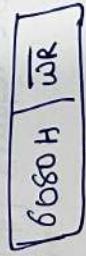
(ROM) control unit
 (RAM) memory array

Action : acc site operator
 M/C instruction: $STA[6080]$: store accumulate.



$M[6080] \leftarrow ACC$

→ CPU generates
 the memory required



M/C cycle required (T_1, T_2, T_3, T_4)

- Sub actions
- ① Send the address
 - ② send the control signal

T₁: ALE=1
 80 → AD₇ → AD₀
 60 → A₅ → A₈

(Rom control unit)

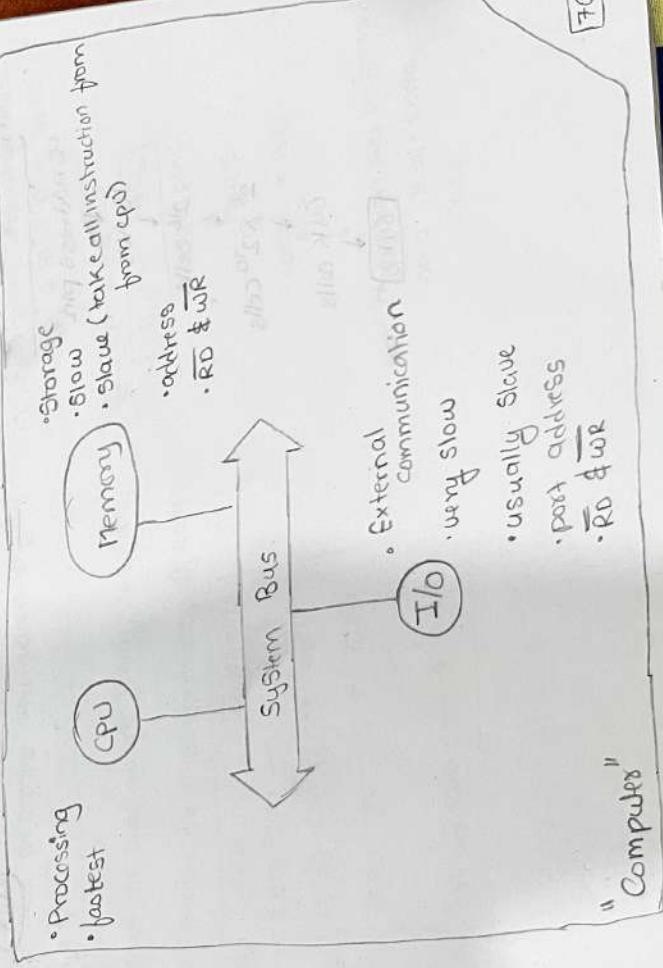
T₂: ALE=0
 RD=1
 WR=0

③ write the data
 (CB & T₄) ALE=0
 AD₃ → AD₀ → D₇ → D₀
 CPU → memory

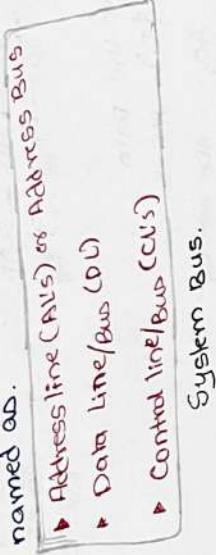
▪ Bus Structure

- Bus is a communication channel, used to provide communication between different component

- Characteristics of Bus is shared Transmission media
- limitation of a bus is only one transmission at a time
- A Bus which is used to provide a communication between major component of computer (CPU, memory, I/O) is known as system Bus.

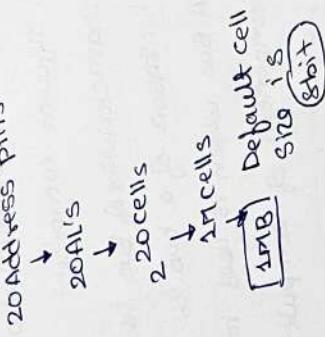


- System bus contain 3 category of line , used to provide the communication b/w the major component of the computer (CPU, memory & I/O) named ad.



1. **A₁₅:** used to carry the address to memory & I/O

Ex: In 8086 MP
(A₁₅.....A₀)



• Uni-Directional

* Based on the width of a Address Bus(# Address pins/lines) we can determines the capacity of a memory system.

Ex: In 8085 MP

(A₁₅.....A₈A₇'.....A₀)

16 Address Pin

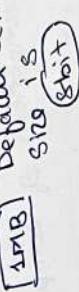
16 A15

2¹⁶ cells

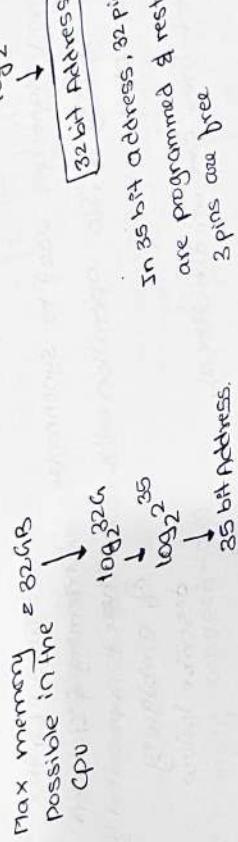
2¹⁶ X 2¹⁰ cells

64K cells

64KB



Ex: Consider a system with
4GB RAM expandable upto
32GB. How many Address
Pin are present in the
CPU.



- ∴ CPU contain **35** Address Pin
- Bait Current memory size = 4GB
 \log_2^{4G}
 \log_2^{32}
- 32 bit address
- In 35 bit address, 32 pin are programmed & rest 3 are free.
3 pins are free.
- Ex: In 8085 MP
- | | |
|-------------------------------------|-------------|
| AD ₇ ... AD ₀ | 8 Data Pins |
|-------------------------------------|-------------|
- 8 DPLS
→ word size = 8bit
- Operation are performed on a 8bit data code
- Ex: In 8086 MP
- | | |
|--------------------------------------|--------------|
| AD ₁₅ ... AD ₀ | 16 Data Pins |
|--------------------------------------|--------------|
- 16 DPLS
→ word size = 16bit
- Operation are performed on a 16bit data code.
2. **DLS**
- used to carry the Binary code between the CPU, memory & I/O
 - Bi-directional
 - Based on the width of the Data Bus we can determine the word length of a CPU
- Based on the word length we can determine the performance of a CPU
- Ex:
- | | |
|-----|-----------|
| CPU | word size |
|-----|-----------|
- 8 bit CPU → 8bit
16 bit CPU → 16bit
32bit CPU → 32bit
"n" bit CPU → "n" bit

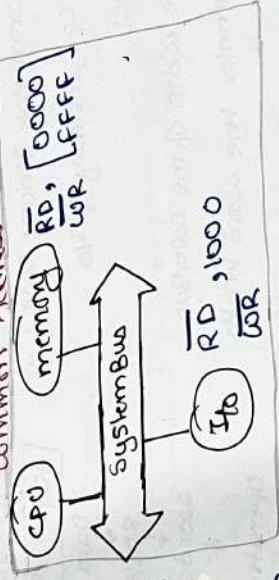
3. CL's

- Used to carry the control signal & time signal
- Control signal indicate the type of operation Eg: \overline{RD} , \overline{WR} ... etc.
- Timing signals are used to synchronise the memory & I/O operation with a CPU clock
- Control lines are unidirectional.

Ex: Sys. with Common Bus

Common CS

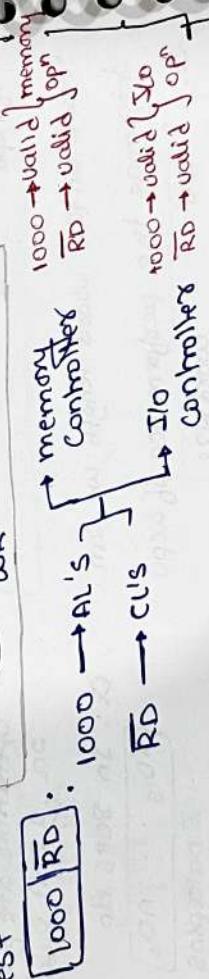
Common Address



CPU generates the memory Request
 $1000 \mid \overline{RD}$

$1000 \rightarrow AL's$

$\overline{RD} \rightarrow$ CS



These is
 Conflict
 [Ambiguity]

- To handle the above situation

Bus configurations is used in the

System design.

There are 3 types:

- ① Isolated - I/O mapped - SO || Just separate Bus
- ② Isolated - I/O [I/O mapped - SO] || Separate common signal
- ③ memory - mapped - SO || Separate address

Note:
 when there is
 Common Bus
 Common Control
 Signal
 Common Address

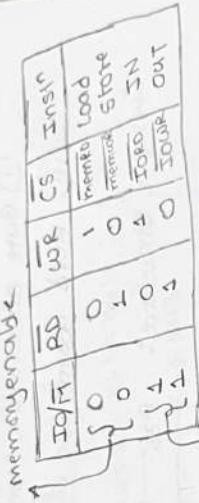
are used bus are
 memory & SO then
 there is a possibility
 of ambiguity
 describe below.

& take the
 remaining
 2 out of
 3
 [Bus CS,
 Address]

I/O

- These configurations uses they common control signal & common Address bus different Buses for both memory & I/O
- I/Os are expensive config so it is suitable in the high performance system design
- Memory-mapped-I/O
 - This config uses the common bus & common CS's but different address for both memory & I/O
 - Unique Address space for memory (I/O)

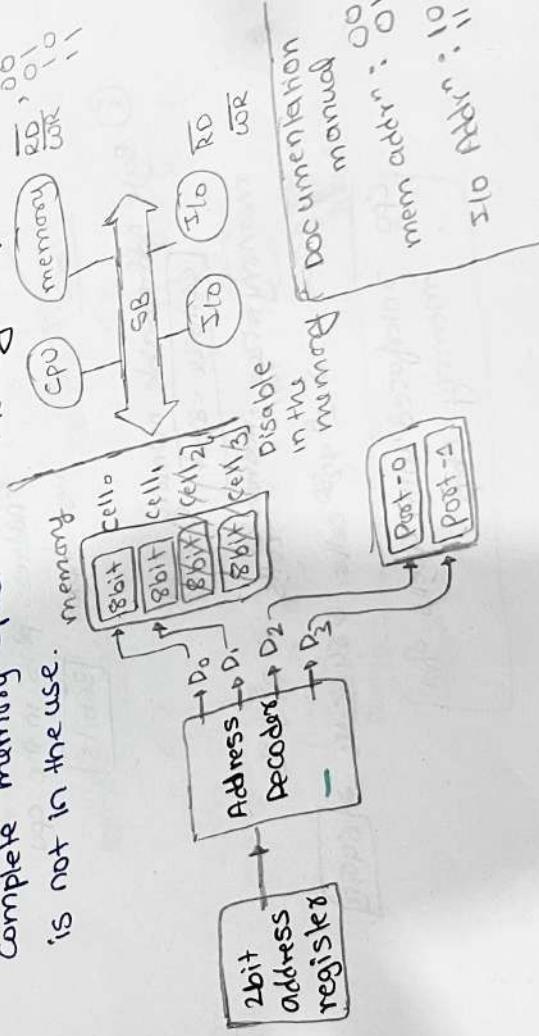
- I/O
 - I/O is used in the CPU to implement these config



- This config is used in I/O enable the general purpose computers.

- In these design memory address space is shared no I/O ports. So limitation is complete memory space is not in the use.

- Consider a system with 4 cells of a memory & 2 I/O port designed with a memory-mapped-I/O



- These config was the common bus & common Address bus uses different Control signals for memory & I/O

- I/O
 - I/O is used in the CPU to implement these config

- (1) Consider a hypothetical CPU which contains 32-bit multiplexed pins used to carry the address & data. Also contains 4 more address pins to identify the word length of a CPU.
- (2) Capacity of a CPU in bytes when the system supports:
- ① Byte addressable memory
 - ② word addressable memory.

CPU pin structure:

```

    graph LR
      A[Address Bus] --- B[Data Bus]
      A --- C[Address Pins]
      B --- D[Data Pins]
      C --- E[36 pins]
      D --- F[32 bits]
  
```

- (3) word length of CPU: width of a data bus
- CPU
- # Data pins in the CPU
- i.e. 32 pins
- 32 bits
- (4) memory size: depends on width of address bus.
- # Address pins in the CPU
- i.e. 36 pins.
- 36 bits

- (5) Byte Addressable memory
- Cells size = 8 bit
- memory size = 2^{36} cells $\Rightarrow 2^{36}$ cells
- $\Rightarrow 2^6 = 64$ cells $\Rightarrow 64$ cells \Rightarrow 64KB

CPU interfaced with cells of a memory chip.

iii) word addressable memory

$$\begin{aligned} \text{Cell size} &= \text{word size} \\ &= 32\text{bit} \end{aligned}$$

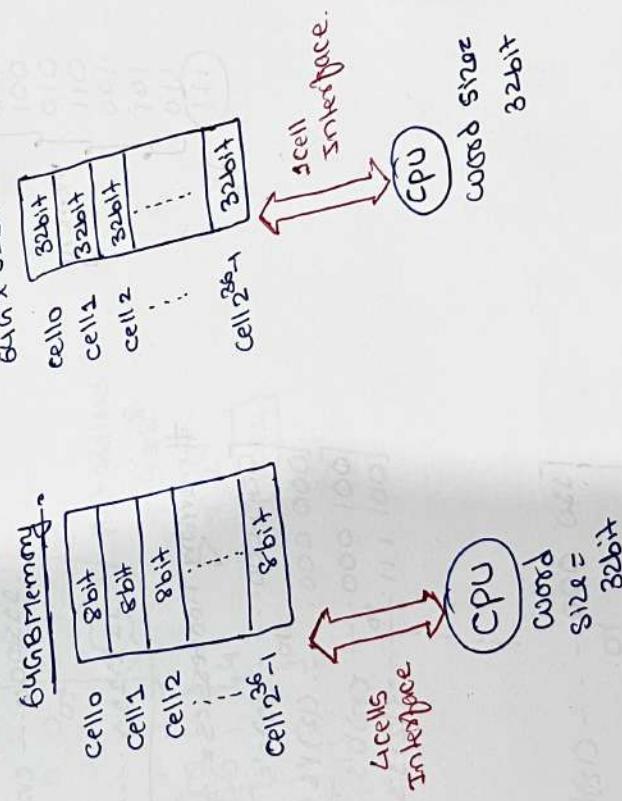
memory = 2 addressed cells.

$$\Rightarrow 2^{36} \text{ cells.} \rightarrow 2^{36} \times 32\text{bit}$$

**CPU interfaced with 1 cell
of a memory chip**

A/c to question the answer expect in byte

$$\begin{aligned} \therefore 64\text{G} \times 32\text{bit} &\neq 64\text{G} \times 4\text{B} \\ &\Rightarrow 2^{36} \times 2^2\text{B} \\ &\Rightarrow 2^{38}\text{B} \\ &\Rightarrow 256\text{GB} \end{aligned}$$

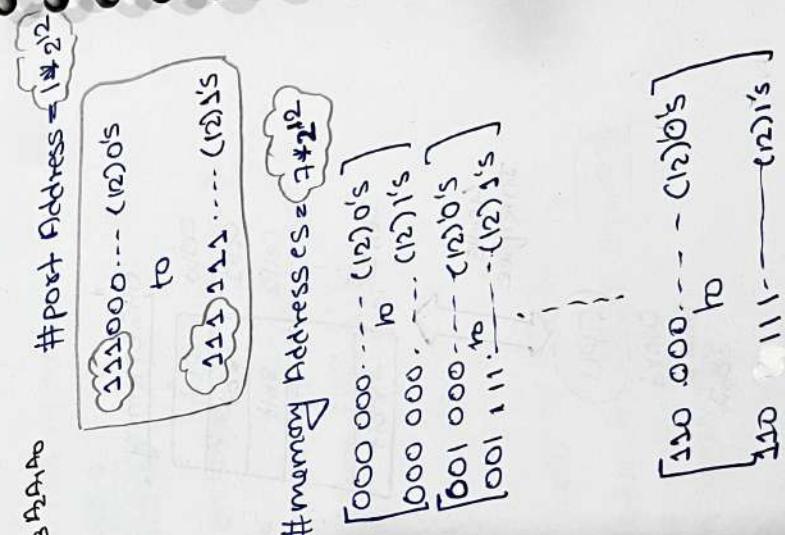
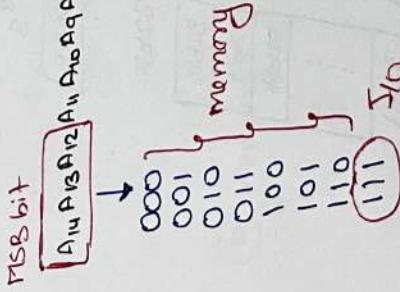


Q) Consider a 32 bit hypothetical CPU which supports 32KB memory system is designed with a memory mapped I/O config in which when the 3MSB bit of Address status is set then user that address for I/O. How many port addresses & memory addresses are possible in the system?

Soln : 32bit CPU
 word size = 32bit Extra parameters

• memory size = 32 KB

$$\therefore \text{Address size} = \log_2 32K \\ = \log_2 15 \\ \Rightarrow 15 \text{ bit}$$

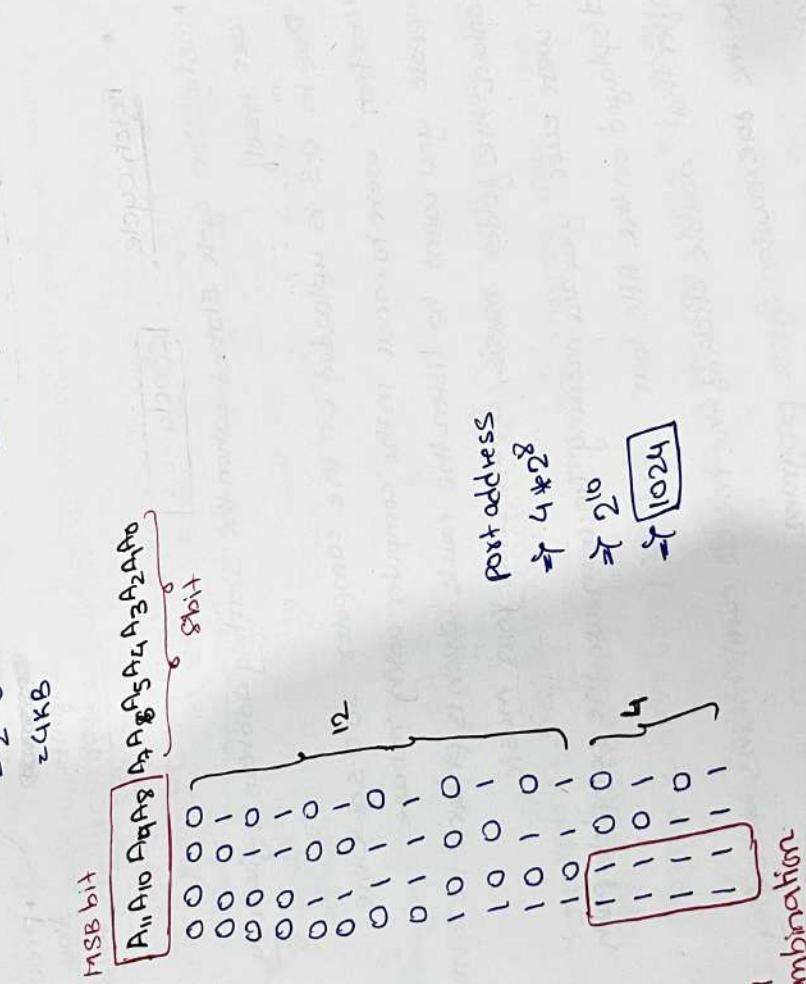


Q1) Consider a hypothetical system, designed with a memory management unit.
In this design, 2^{12} address space is mapped to 10 ports. In these designs, shared 10 ports in which 4 bits of the address & 10 port i.e. 3 bits of the address below the memory address are mapped to 10 port.

How many port addresses are possible in the system?

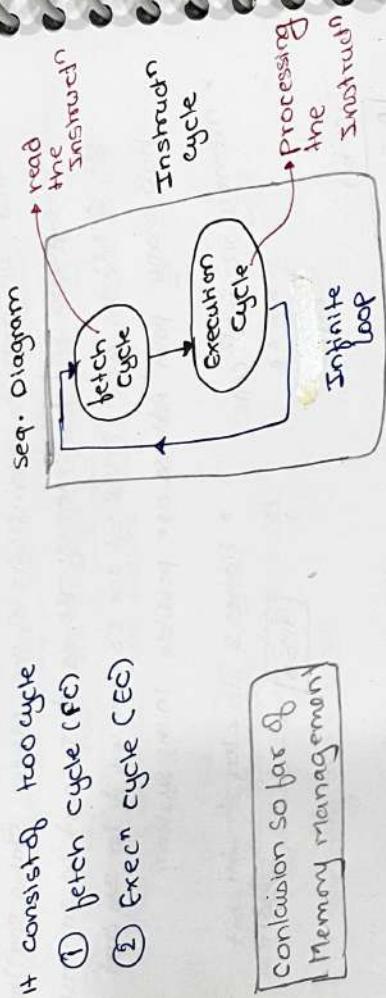
• Address size = 10×2^2 + 12 bit address
= $2^{12} \times 8$

$$\text{Memory size} = 2^{12} \text{ cells}$$
$$= 2^{12} \times 8$$
$$= 4 \text{ KB}$$



Instruction cycle

- It describes the execution sequence of a program or an instruction.
- It consists of two cycles:
 - I Fetch cycle (FC)
 - II Exec cycle (EC)



Conclusion

- Instruction cycle starts when the starting address is given by user itself.
- Due to O.S is uploaded on the computer, so O.S give the direction where to move in the computer using mouse.
- Suppose you want to listen the music which is present in D drive contain the folder music, & then select your music file.
- So when you are moving your mouse the complete path is displayed on the title bar.
- So, after that double click of the mouse event means executing indicate the execution of the program.
- So that path is converted as address By O.S & that address is called as logical address b/c your file is present in 2nd memory (Hard disk).

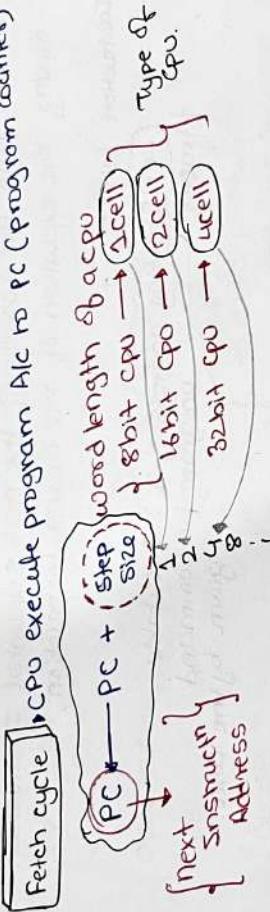
- van-neumann architecture. point of view program is always required in the M.M.
- But my program not available in main memory. it is in H/W so O.S. is responsible for transferring program from H/W to M
- H/C to your path.

- So, M.M part of the program is executed by the CPU by using.

Instruction cycle

- without the address of the program CPU doesn't start the execution. So compulsory program address supply by the user.

CPU execute program Alc to PC (Program counter)



Program counters act as
eye to CPU to compulsory
it should present

1. objective of the fetch cycle is instruction fetch. In these process.

1. objective of the fetch cycle is instruction fetch. In these process.
2. PC is a compulsory register in the CPU, used to hold the instruction address

3. starting address of the program is supplied by the user &
- the next instruction address is automatically generated by the CPU, by incrementing the PC value by step size i.e. $PC \leftarrow PC + Step\ Size$

- Step size is a fixed constant in the CPU, depend on the word length of a CPU

Step size → fixed constant

CPU	word Size	# cells Interlacing	Step Size
8 bit MP	8 bit	1	1
16 bit MP	16 bit	2	2
32 bit MP	32 bit	4	4
64 bit MP	64 bit	8	8
...

- Total PC hold the address of the next sequential instruction during the execution of the current instruction.

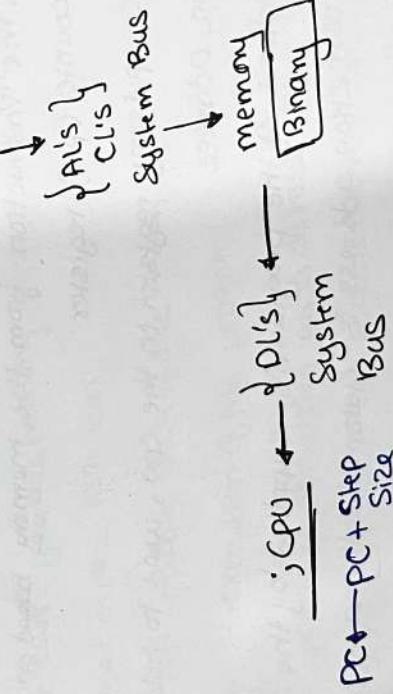
Instruction fetch:

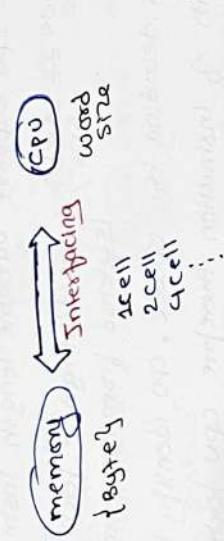
+ Starting address of a program
+ tracey

means instruction by instruction execution

PC → CPU generates the memory registers

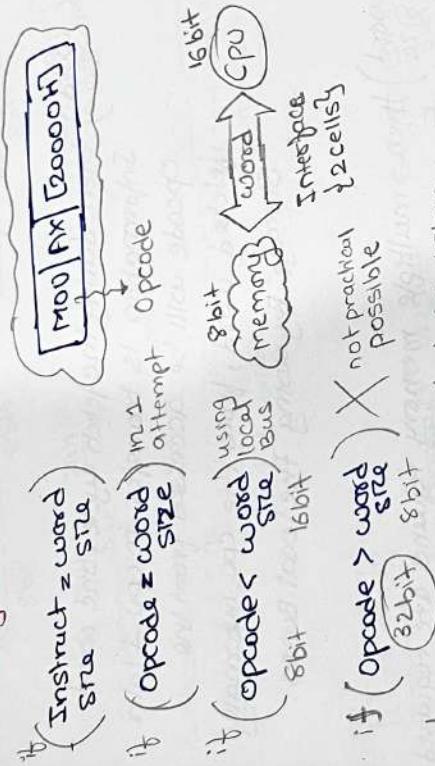
Conclusion of about point





After decoding
the opcode
CPU knows
the length
of instruction

Analysis: Instruction :



So, 4 attempt are required so when no of attempt increase, memory reference increase time increase. CPU is faster but memory slow. So when multiple time memory is referred by the CPU the time increases & performance decrease.

where instruction

Analysis:

1. when the CPU support fixed length instruction then during the fetch cycle. if the size equal to word size then the complete instruction is fetched from memory.

2. when the CPU supports variable length instruction

(a) $(\text{opcode} = \frac{\text{word}}{\text{size}})$ then during the fetch cycle, opcode is fetched from the memory

- after decoding the opcode, CPU identifies the actual length of instruction therefore CPU fetch remaining part of the instruction from the memory after the decoding of the opcode

(b) $(\text{opcode} < \frac{\text{word}}{\text{size}})$ then during the fetch cycle, one word information is transferred to CPU & later opcode will be accessed from the fetched word, from the CPU internally storage by using the local buses.

(c) $(\text{opcode} > \frac{\text{word}}{\text{size}})$ then multiple memory references are required to fetch the opcode itself. This concept is not in the practice b/c performance is decreased when the access time of instruction increase.

Designer view

Ex: 8085 CPU

- 8bit CPU (word size = 8bit)
- Opcode size = 8bit
- Address size 216bit (64KB)

(2) 28(2¹⁶) long instruction



Eg: MUL 24H

move immediate

∴ opcode size need not be same as word size

ROM control unit

User view

Read the instruction manual

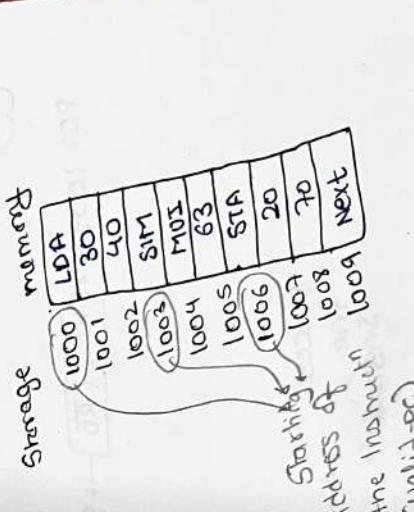
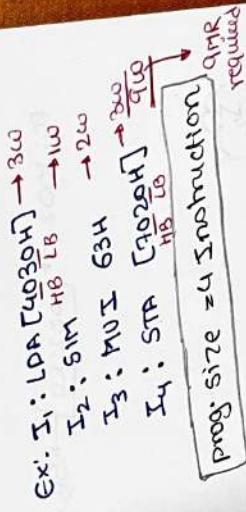
→ write the source code

i.e. org 1000 → it means
origin after 1000
Start storing
instruction in
memory

If not given then
By default its 0000

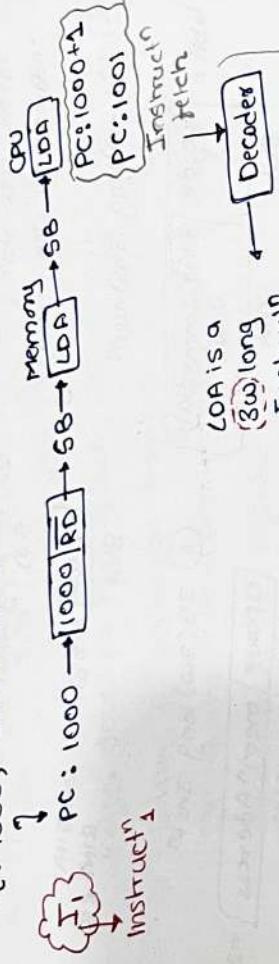
[set the starting address]
of a program

Starting
addresses of
the instruction
in memory

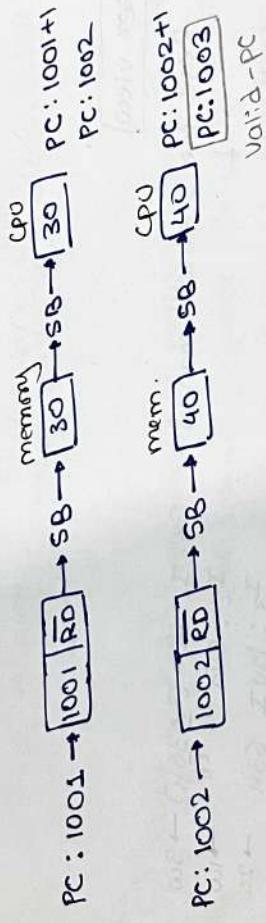


88

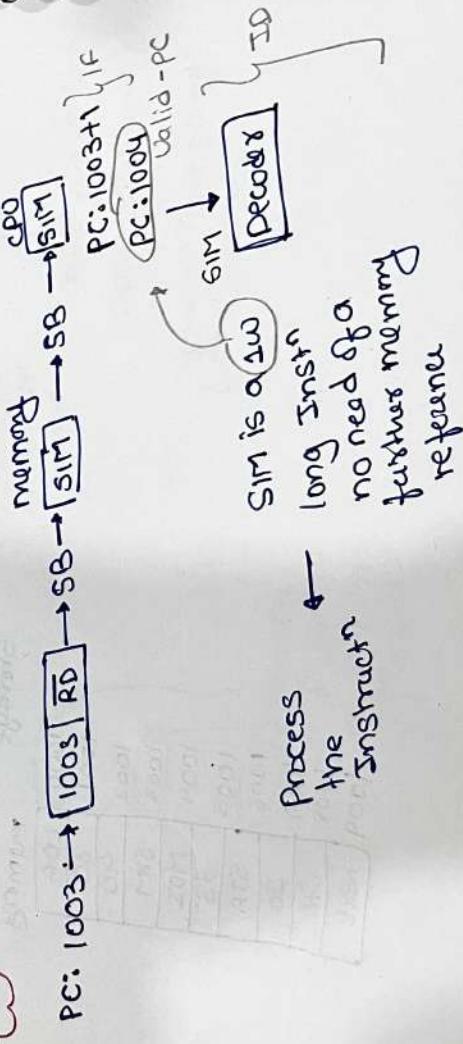
-t: 1000; executable statement



so, 2 more memory reference required to fetch the remaining part of an instruction



processes the
information



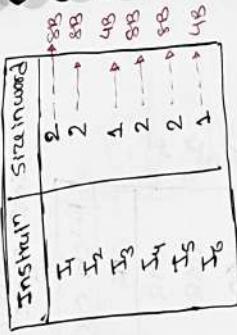
Instn cycle:

Fetch cycle	Execution cycle
IF	ID
IMR	300 500 ?
IMR	200 400 ?
IMR	200 300 ?
IMR	300 400 ?
IMR	200 300 ?

qmr required
to fetch the
program from
the memory

MR: memory
reference

IF: instn fetch
ID: instn decode



Q] Considered 32bit hypothetical CPU used to execute the following code program is stored in the memory with a starting address of 11000000.

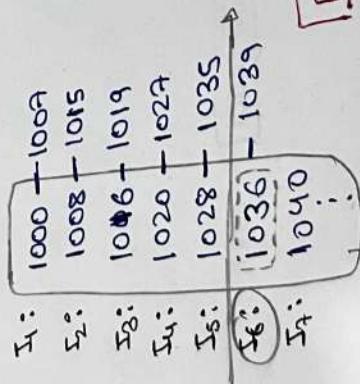
- During the execution of "SS" instr what could be the value present in the PC when the system supports.

- Byte addressable memory
- Word addressable memory.

so! PC hold the address of next instr during the execution of current instr

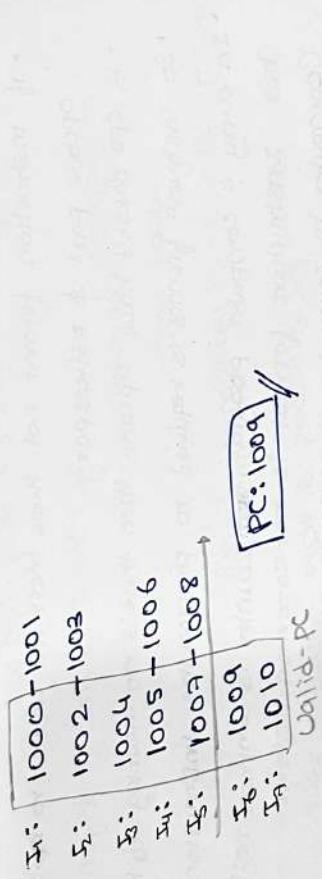
A. Byte addressable memory (cell size = 8bit)

- word size = 32bit
= 4B
- 4 cell of a memory space



valid - PC

- Q. word addressable memory
 Cell size = word size
 So, two instructions take one cell of memory space.



Q) consider 2ubit hypothetical CPU which support 2 word long instruction. Program is stored in the memory starting address of 800, which one of the following is a valid - PC?

- A. 400 B. 600 C. 1200 D. 700.

memory config = not quad

so, 1 word = 2 byte
 word size = 2ubit
 storage size = 2w
 $\Rightarrow 24 \text{ bit}$
 $\approx 3B$

so, 3 cells storage space is required for instruction.

Storage:

I ₁ :	300 - 301
I ₂ :	303 - 305
I ₃ :	306 - 308
I ₄ :	309 - 311
I ₅ :	312 - 314
I ₆ :	315 - 317
I ₇ :	318 - 320
I ₈ :	321 - 323

value of PC is a multiplication factor of 3.

Execution cycle

- for processing we need instruction format which tells us detail information about opcodes & other field
- with the help of instruction format CPU processes the instruction
- if instruction format not there then its unable to separate opcode part & address part.
- if CPU doesn't have opcode then there is no meaning of processing
- so, instruction format is required to process the instructions
- In every computer program, common register is common register but instruction format is not common register but computer to computes it values)
- So 1st select the computer type. (CPU type)
- there are 3 type of CPU. (Based on the availability of the ALU)
(Operands)
- 1. Data transfer instruction
- 2. Data manipulation instr
- 3. Transfer of control instr
(JMP Instruction)

2 Arguments

(D) (S)

Ex: $(MOV) (R1) (R2)$

Opnd D S

$R1 \leftarrow R2$

• Data transfer instrn

• while execution of these instruction

Data will be transfer

• to transfer the data you need

Source & destination

• so no need of ALU to transfer

Data instruction b/c Data

is just copying from one place to another

so no modification required

- Data manipulation instr : (alu instruction)
 - to modify the data you need ALU

ex: $D \quad S_1 \quad S_2$

ALU Data path Counter wait type of CPU

- Transfer of control instr: D , destination (Δ argument)

ex: $JMP \quad 2000$

// Just transferring the control to 2000

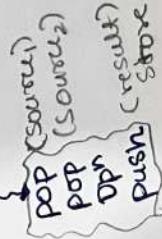
objectives of the execution cycle is processing it currently fetched instruction

- Instruction format is required to process the instruction
- Instruction format describe the layout design (internal structure) of an instruction
- Instruction format is divide into **3 types**. based on the CPU organisation
- CPU organisation is classified into 3 type based on the availability of the ALU operands (ALU data path) name as
 - stack CPU
 - accumulators CPU
 - general purpose CPU
- why address is not present?
 - b/c default location is stack, operand and stack, operand operand.
 - By default program in the stack. So no need of operand address
 - of operand address only opcodes is enough

• stack CPU:

ALU : $D \quad S_1 \quad S_2$
operator : $TOS \quad TOS \quad TOS$

Opcode ; 0-address instr [opcode is there without address format.]



why address is not present?

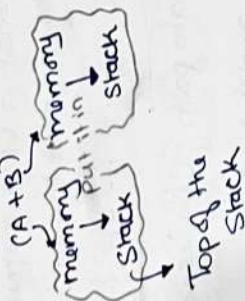
101,2,3,4 = address in stack

classification of "ISA" (Instruction set Arch)

stack is a part of main memory

if A & B are not ex:
present in stack then
we need to bring them
in the stack & then
perform operation

Data transfer
Op



- when stack is the default location the compulsory memory address required so in these computers data transfer instruction & data manipulation instruction are address instruction.
- required zero address instruction.

Ex: I₁: push \$i [2000] 23H

Variable	Address	cell
A	2000	23H
B	3000	42H

$A+B = 23+42 = (65)H$

variable is replaced by
memory address
(so variable itself
a memory address)

3 M/C
Instruction
required

I₂ : push \$j [3000] 23H

I₃: add A+B [(A+B) (65)H]

[pop
pop + push]

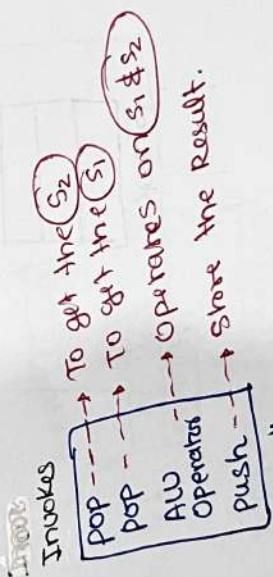
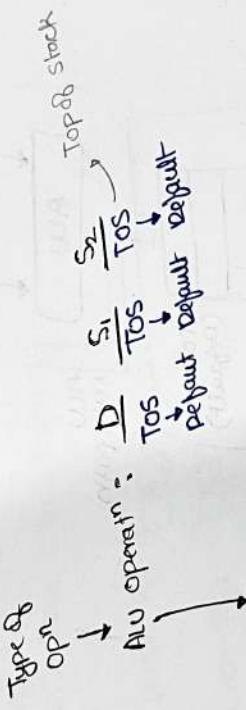
Stack CPU (0 - odd address)
In these organisation ALU operation are performed only on the stack data. means both ALU operand are always required in the stack

- After the data processing, result is also placed into the stack
- Stack is a last in first out data structure so insert & delete operation take place at the same end known as Top of Stack (TOS)
- It become a default location so its address need not be required in the instruction
- Stack is initialized in the main memory using stack pointer (SP)

Registers

compatible instruction format is

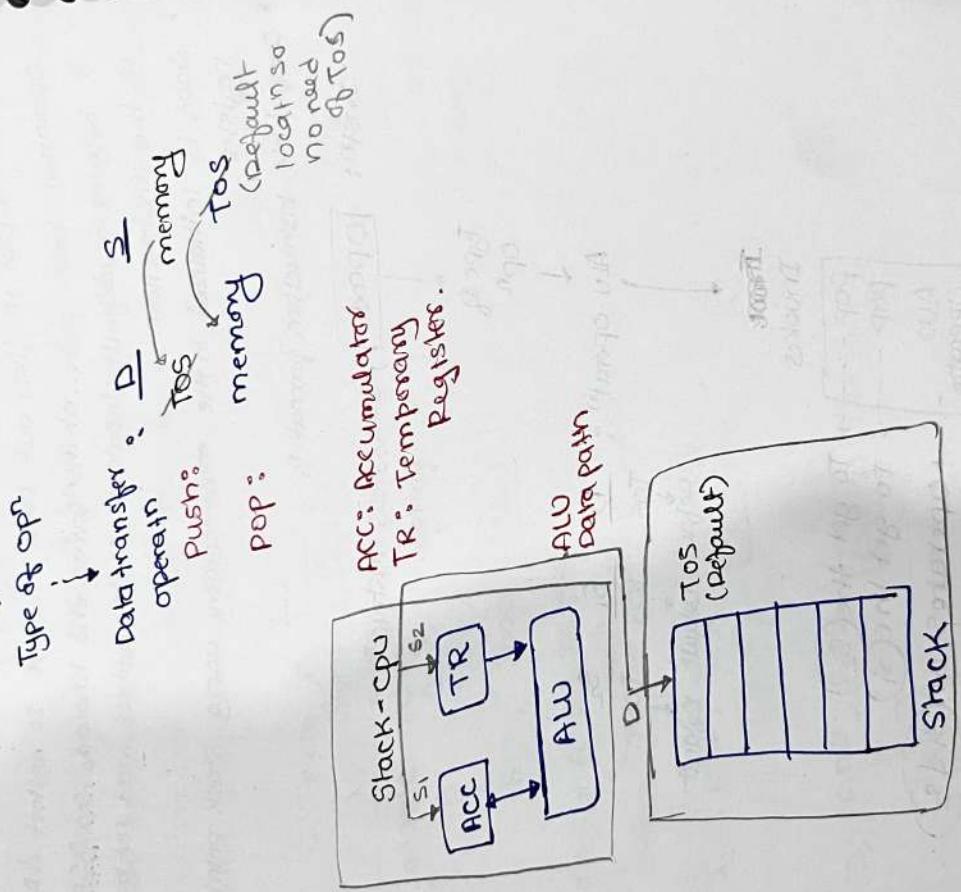
Instn: **[OpCode]**; D - Address instruction format.

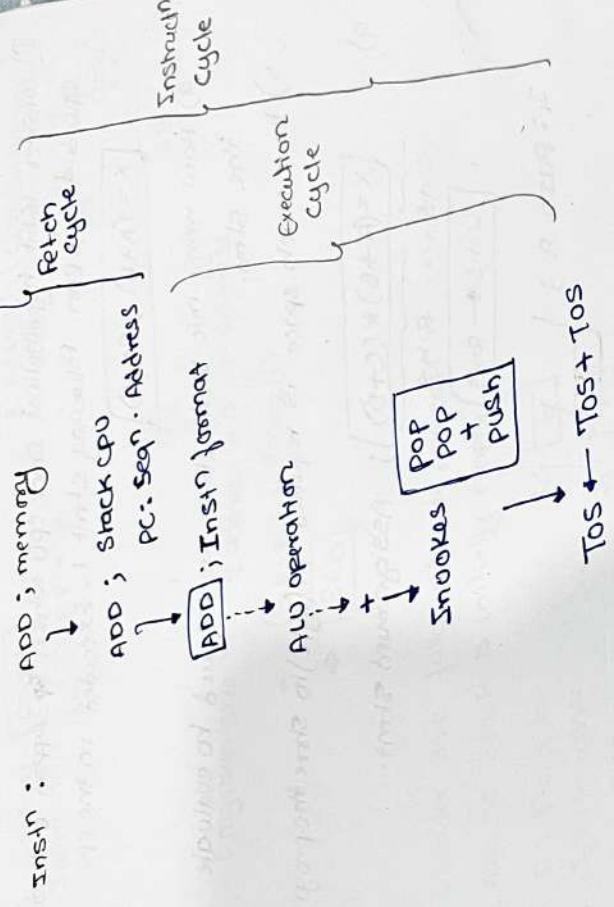


$$\text{TOS} \leftarrow \text{TOS} (\text{ALU op}) \text{TOS}$$

- In these organisation Data transfer operation [push & pop] or design task one address instruction

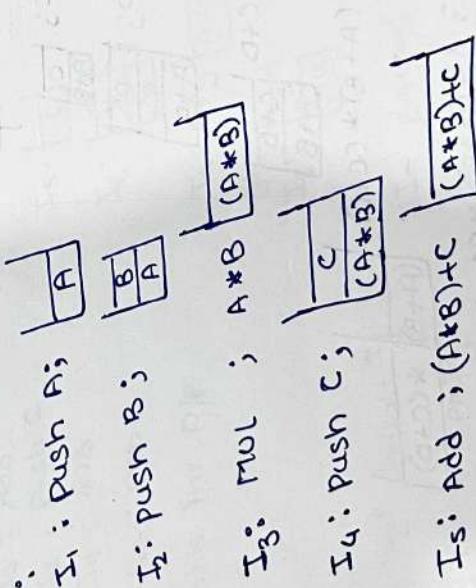
Instruction : opcode | Address; 1-address insin format.





Ex: $(A * B) + C$
Variable are in the memory

Code:



Q) Consider 16 bit hypothetical stack CPU which supports 8 word opcodes & 4GB RAM. Following Stmt is Executed in the CPU.

$$X = (A+B) * (C+D)$$

- a) How many m/c instruction are required to evaluate the Stmt.
- b) How much space is required in (Byte) to store the program

a).

$$\boxed{X = \underbrace{(A+B)}_{\text{L.H.S}} \times \underbrace{(C+D)}_{\text{R.H.S}}}; \quad \text{Assignment Stmt.}$$

2-addresses
 $I_1: \text{push } A ; \boxed{A}$

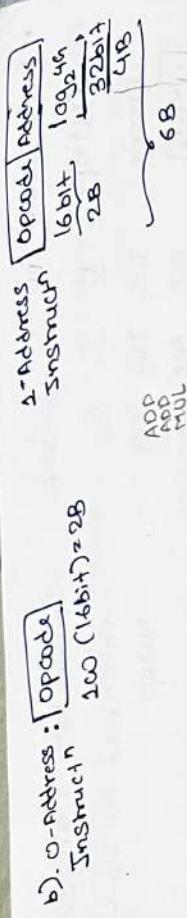
$I_2: \text{push } B ; \boxed{B}$

0-address
 $I_3: \text{ADD} ; \boxed{A+B}$

1-address
 $I_4: \text{push } C ; \boxed{C}$
 $I_5: \text{push } D ; \boxed{D}$

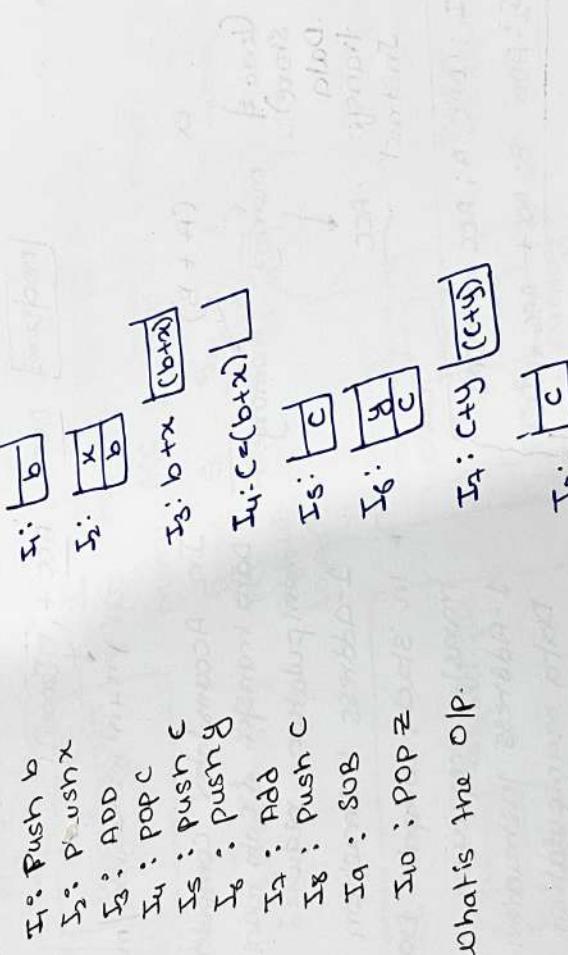
0-address
 $I_6: \text{ADD} ; \boxed{C+D}$
 $I_7: \text{mul} ; \boxed{(A+B) * (C+D)}$

1-address
 $I_8: \text{POP } X ; \boxed{(A+B) * (C+D)}$



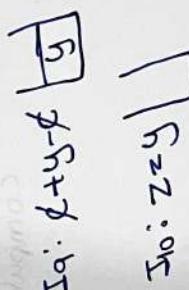
$$\begin{aligned}
 \text{Program size} &= "5" \text{ 1-address} + "2" \text{ 0-address} \\
 &\Rightarrow (5 * 6B) + (3 * 2B) \\
 &\Rightarrow \boxed{36B}
 \end{aligned}$$

Q1 Consider the following code, execute on a stack CPU
 Assume stack is initially empty.



What is the value of R?

Op : "z" contains "y"
 \$ stack is empty



- Accumulator-cpu (1-address format)
 - All opn : $D_1 S_1 S_2$
 - Opcode $TOS TOS$
 - $ACC ACC$ **Register memory** // so in the entire computer only one accumulator is present
 - Address** // 2nd operand may be present in Register memory
 - // So there no need of address memory

In the accn &cpu 1-address instruction is used

ex: $ADD[re] ACC \leftarrow ACC + r_0$ $ACC \leftarrow ACC + M[2000]$ $ACC \leftarrow ACC + M[2000]$ \downarrow present in Register/memory

• In Accumulator Computer

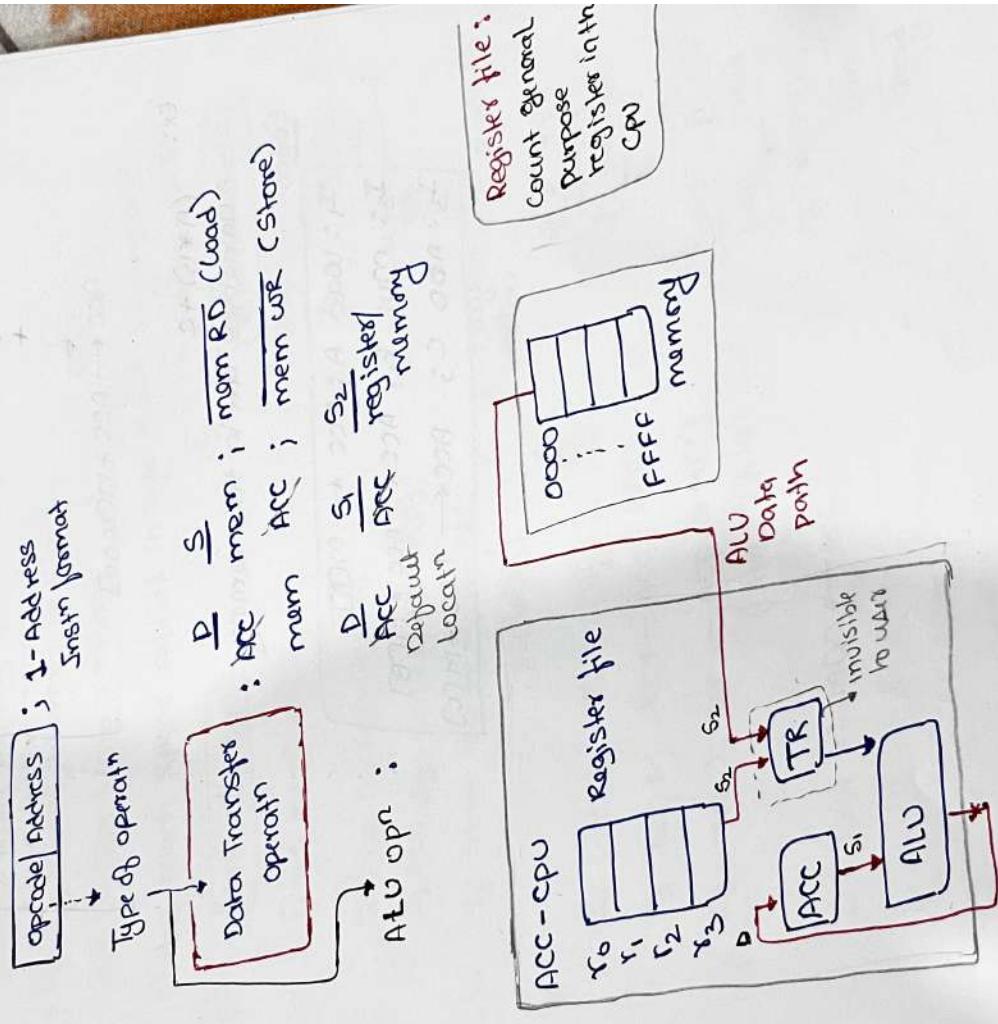
 - Data transfer & Data manipulation required is 1-address instruction
 - 1-address instruction requires 1-Address instruction & Data manipulation requires 0-address instruction.

ex $(A + B)$ \downarrow memory
 (Load & store) \downarrow Data transfer Instruction
 $ACC \downarrow$

$I_1: LOAD A; ACC \leftarrow M[A]$
 $I_2: ADD B; ACC \leftarrow ACC + M[B]$

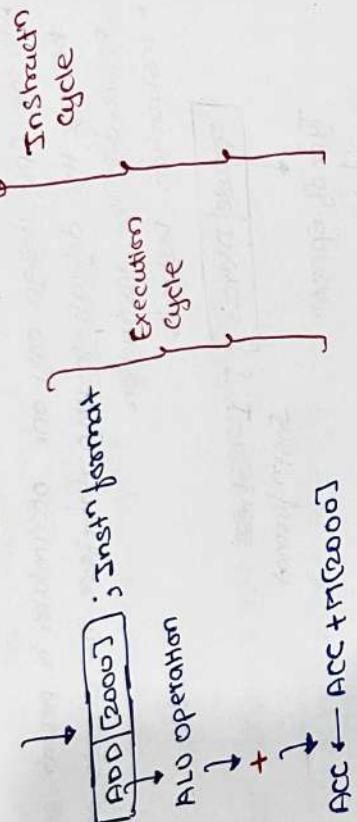
$\boxed{I_1: PUSH A}$ $\boxed{I_2: PUSH B}$ \boxed{ADD} \rightarrow Stack computer

- In these organisation ALU & operand is always required in the accumulator & the 2nd operand is present in either ~~either~~ in the register or in memory . after the Data processing , result will be placed into the accumulator
 - In the CPU design only one accumulator is present . so accumulator become the default location . therefore its address need not be required in the instruction required in the instruction
 - Instruction Design is



Instn: ADD [2000]: memory

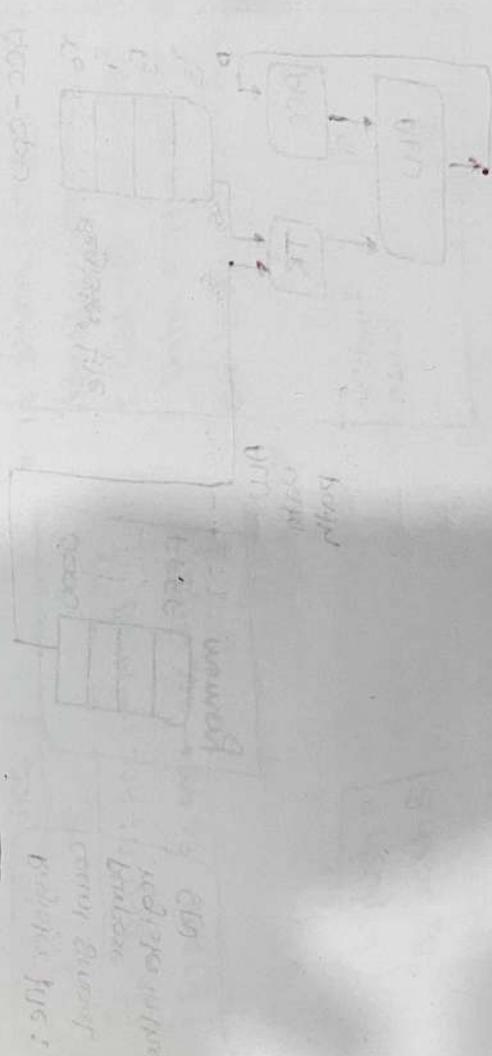
ADD [2000]; ACC - CPU
PC: sequential address



Ex: $(A * B) + C$
Variables are in the memory

Code:

```
I1: load A; ACC ← m[A]  
I2: mul B; ACC ← ACC * m[B]  
I3: add C; ACC ← ACC + m[C]
```



Q) Consider the following stmt execute on a Acc-CPU, CPU contain 8bit opcode & 4KB RAM.

$$X = (A + B) * (C + D)$$

- Variables are in the memory
- no internal data for carrying in the CPU (no registers)
- Registers Transfers operation

- How many 'mc' instr are required? $\lceil \frac{16}{8} \rceil = 2$
- How many memory spills required? $\lceil \frac{16}{8} \rceil = 2$

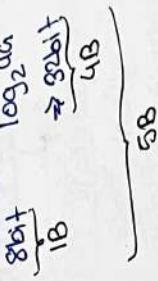
Spilling means intermediate result will be stored into a memory

- How much space required in Reg to store the program?
- 5-address instrn

C

$$X = (A+B)* (C+D)$$

[L.H.S \rightarrow R.H.S]



Program = "x" → address
STa

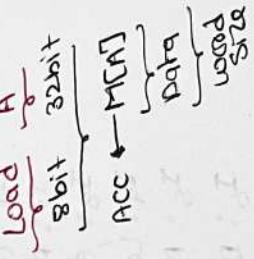
Spilling

$$X = 358 //$$

I₁: LOAD A; ACC $\leftarrow M[CS]$

I₂: ADD B; ACC $\leftarrow ACC + M[DS]$

I₃: MUL T; ACC $\leftarrow ACC * M[FS]$



I₄: Store X; M[DS] $\leftarrow ACC$

word size

20bit \rightarrow 16bit \rightarrow 8bit \rightarrow 4bit

20bit \rightarrow 16bit \rightarrow 8bit \rightarrow 4bit

20bit \rightarrow 16bit \rightarrow 8bit \rightarrow 4bit

Ans: $X = (A+B) / (CC+DD)$

Code:

I₁: LOAD A; ACC $\leftarrow M[A]$
I₂: ADD B; ACC $\leftarrow ACC + M[B]$

I₃: STORE T; M[T] $\leftarrow ACC$ {Spilling}

I₄: LOAD C; ACC $\leftarrow M[C]$
I₅: ADD D; ACC $\leftarrow ACC + M[D]$

I₆: STORE Y; M[Y] $\leftarrow ACC$ {Spilling}

I₇: LOAD T; ACC $\leftarrow M[T]$

I₈: DIV Y; ACC $\leftarrow ACC / M[Y]$
($A+B$) / ($CC+DD$)

I₉: STORE X; M[X] $\leftarrow ACC$

not all sharing is
spilling (only
intermediate we
can say)

Instruction # not optimized
ans
2 spills X wrong
ans.

i.e. To write the optimized code follow the order of evaluation

i.e. evaluate CC+D first

Code:
I₁: LOAD C; ACC $\leftarrow M[C]$
I₂: ADD D; ACC $\leftarrow ACC + M[D]$
I₃: STORE T; M[T] $\leftarrow ACC$ {Spilling}
I₄: LOAD A; ACC $\leftarrow M[A]$
I₅: ADD B; ACC $\leftarrow ACC + M[B]$
I₆: DIV T; ACC $\leftarrow M[T]$
I₇: STORE X; M[X] $\leftarrow ACC$

Ans: Instruction # 18 spills

Correct ans.

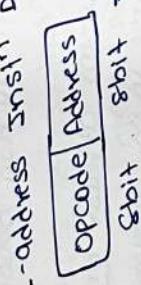
Analysis: How to design different instr format w/ same op.

- ①. variable length instrn support cpo (CISC)
- ②. fixed length instr supported cpo (RISC)

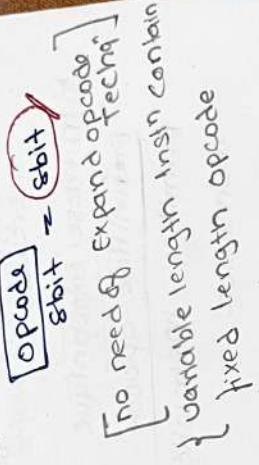
1. Variable length instrn

Assume Opcode = 8bit, Address = 8bit

- a) 1-address Instrn Design



- b) 0-address Instrn Design

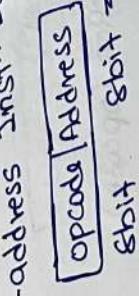


2. Fixed length instrn

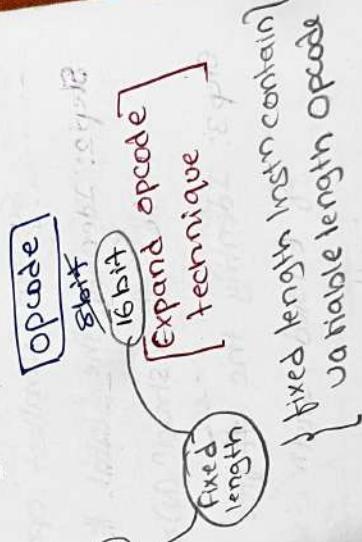
Assume Opcode = 8bit

• Address = 8bit

- a) 1-address Instrn Design



- b) 0-address Instrn Design



Expand opcode Technique

- These technique is used in nine fixed length instruction support op codes, to implement the different instruction with different format.
- In these technique, address field will be appended to the free opcodes, to design the derived instruction, therefore derived instruction opcode size is increases pre-requiment atleast one primitive opcode must be free in the system.
- In these technique primitive instruction means smallest opcode primitive instruction means instruction

Expand opcode Technique Procedure:

Step 1: Identify the primitive instruction format in the System (smallest opcode instr)

Step 2: Identify the total # instruction (opcode)
i.e $\# \text{instruction} (N) = 2^{\# \text{opcode}}$

Step 3: Identify the # free opcode after the implementation of existed instr
i.e $\# \text{free opcodes} = N - \# \text{existed instr}$

Step 4: calculate the # of derived instruction possible in the system by multiply the # free opcode with a devided value of the address field.

$$\text{i.e } \# \text{derived} = \# \text{free } * 2^{\# \text{address}} \\ \text{Instr}$$

10A

Q1 consider a hypothetical system/cps which support 6bit instrn & 4bit address. If there exist "3" 1-address instrns then

how many 0-address instrns can be possible in the cps

Instruction size : 6bit \rightarrow 1-address instrn

Instruction format length \rightarrow 0-address instrn

instruction format

of address format

Opcode, Derived.

(6bit)

Primitive : [Opcode | Address]

2bit, 4bit

Want appendng

6bit
we don't want
address

Step 2: # instrn (operations)

Possible = 2^{Opcode}

Only 00 address = 2²

01 address = 2¹

02 address = 2⁰

10 address = 2¹

11 address = 2⁰

Step 3: # instrn (operations)

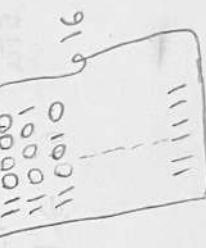
possible

instrn impossible

+ # free addresses

Code

* 2ⁿ



→ 4-5
→ 1
Opcode is free

Train set

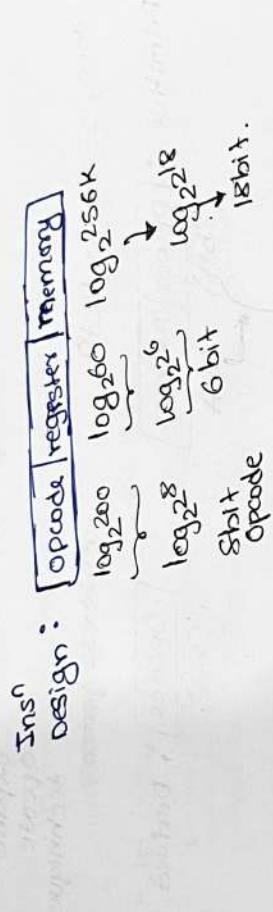
1-address instrn = 3

0-address instrn = 16

Total # instrn = 19

Q1 Consider a hypothetical CPU which supports 1 instruction with regⁿ operands & memory operands system supports 200 instructions, 60 registers & 256 KB RAM. How many bits are required to encode the instruction (instruction size)?

Sol:

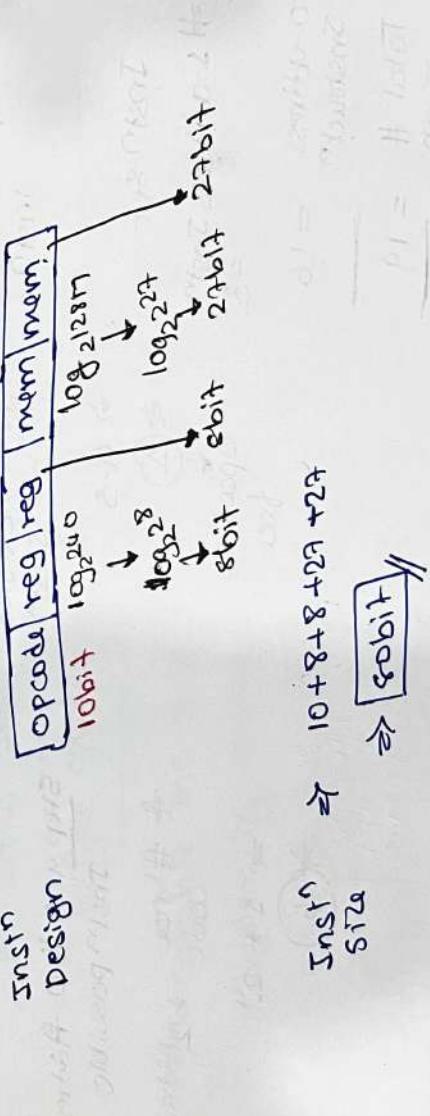


Instruction size: $(8 + 6 + 18) \text{ bits}$

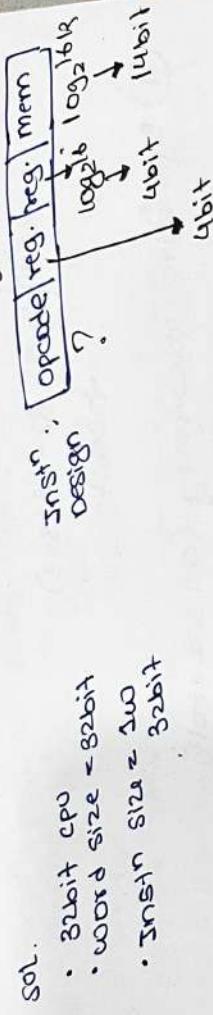


Q1 Consider a hypothetical CPU which supports 1 instruction with 10 bit opcode, 2 registers & 2 memory operands. What is the size of an instruction (in bits)?

Sol:



- Q) Consider a hypothetical 32-bit CPU which support word long instruction with 2 registers & memory operands. Register file size is 16 bytes. Suppose 16-bit memory. How many instructions are possible in this CPU.



$$\Rightarrow \text{so, opcode} = [32 - (4 + 4 + 4)]$$

$$= 16 \text{ bit}$$

$$\# \text{ Instructions} = 2^{16}$$

- Q) Consider 16-bit hypothetical CPU which support word long instruction with 6-bit opcode, 2 registers operands & memory. Register file size in the CPU is "8". What is the capacity of a memory in byte possible in the system when the system supports:
- Byte addressable memory
 - Word

Sol.

16-bit CPU
word size = 16-bit
Instruction size = 2 word
 $2 * 16 \text{ bit}$
= 32-bit.

Instruction format :	<table border="1"> <tr> <td>Opcode</td><td>Reg.</td><td>Reg.</td><td>memory</td></tr> <tr> <td>6bit</td><td>\log_2</td><td>?</td><td>?</td></tr> <tr> <td>3bit</td><td>3bit</td><td>3bit</td><td>3bit</td></tr> </table>	Opcode	Reg.	Reg.	memory	6bit	\log_2	?	?	3bit	3bit	3bit	3bit
Opcode	Reg.	Reg.	memory										
6bit	\log_2	?	?										
3bit	3bit	3bit	3bit										
memory address =	$32 - (6+3+3)$												
	$\Rightarrow 20$ bit.												

$$\text{memory address} = 32 - (6+3+3) \\ \Rightarrow 20 \text{ bit.}$$

Q) Byte addressable memory (Cell size 28bit).

$$\begin{aligned} &\Rightarrow 2^{20} \text{ cells.} \\ &\Rightarrow 1\text{TB cells} \\ &\Rightarrow 1\text{MB} \end{aligned}$$

Q) word addressable memory (Cell size = word size)

memory size = 2 address cells

$$\begin{aligned} \text{word size} &\Rightarrow 2^{20} * 2 \text{ bytes} \\ \text{memory size} &\Rightarrow 2^{20} * 16 \text{bit.} \end{aligned}$$

A/c to Ques " the size in Bytes.

$$\begin{aligned} \therefore \text{Memory size} &= 2^{20} * 2^3 \text{ with word size } 2 \\ &\Rightarrow 2^{23} \text{ bytes} \\ &\Rightarrow \boxed{2\text{MB}} \end{aligned}$$

- * Q) Consider 32bit hypothetical CPU which support no long instruction, placed in 256 MB memory space. Instruction field has 7bit opcode, 15 registers. What is the largest unsigned sys support 16 registers. Is it possible in the instr?

Ans: 819111

Instn
design



$$\text{Immediate} = \frac{32 - (7 + 4 + 4)}{16} = 13 \text{ bit}$$

represents
the constant

$$\rightarrow \text{unsigned Data : } \begin{cases} 0 \text{ to } (2^{13}-1) \\ 0 \text{ to } 81911 \end{cases}$$

Ans: 819111

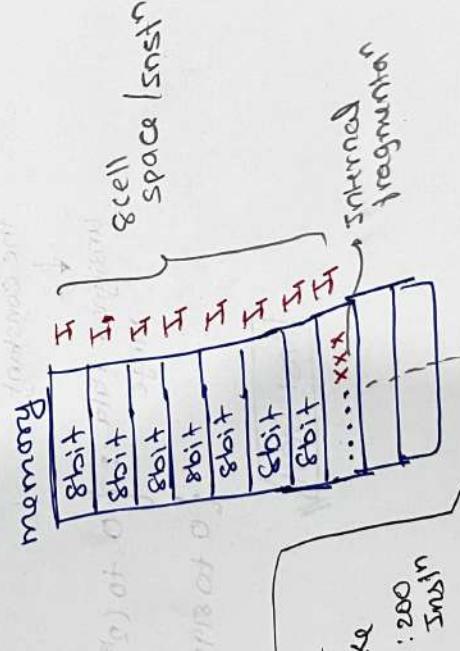
Q) consider a hypothetical CPU which support instruction with
2 registers operands, memory operands & 17bit immediate.
Constant field Instn size of a CPU is 60. System support
256MB ram & 32 registers. Program size is 200 Instn.
How much storage space is required in byte to store
the program in memory

Instn Design :	opcode	reg	reg	memory	immediate
	10bit	10bit	10bit	10bit	17bit

So, $Instn\ Size = (6 + 5 + 5 + 28 + 17) \text{ bit}$
 $= 61 \text{ bit}$.

memory size = 2^{56} MB cell size = 8bit

Note
 If there is internal fragmentation, the entire remain space you can't allocate much. So that cell is entirely taken by one Instn.



So, Program size \approx
 $200 \text{ Instn} * 8 \text{ cells}$
 $\Rightarrow 1600 \text{ cells}$
 $\Rightarrow 1600 \text{ B}$

* Ques1 consider the following code executed on a hypothetical CPU. System supports 60 registers & 64 memory

Code

```

T1 : MOV R0,A
T2 : ADD R1,B
T3 : MUL R2,C
T4 : MOU R3,B
T5 : DDQ R4,D
T6 : DIV R2,A
T7 : MUL R5,B
T8 : DIV R3,C

```

what is the size of an opcode
if size of an instrn in the

equi, where the variable are
memory addresses.

Instn Set = # unique opcodes
in the program

{MOV, ADD, MUL, ONLY}

Control
unit

ROM

$$\text{Opcode size} = \log_2 \# \text{Instrns}$$

$$= \log_2 8 = 3 \text{ bit}$$

~~wrong approach~~

So, Opcode size

$$\Rightarrow \log_2 4$$

$$\Rightarrow 2 \text{ bit}$$

opcode | reg | memory

$$2 \text{ bit} \quad \log_2 60 \quad \log_2 64 \text{ K}$$

6 bit 16 bit

$$\text{Instn Size} = (2 + 6 + 16) \text{ bit}$$

$$= 24 \text{ bit}$$

$$\text{program size} = 8 \cdot \text{Instn}$$

$$= 8 * 24 \text{ bit}$$

$$= 8 * 38$$

= 240B space required
to store the program

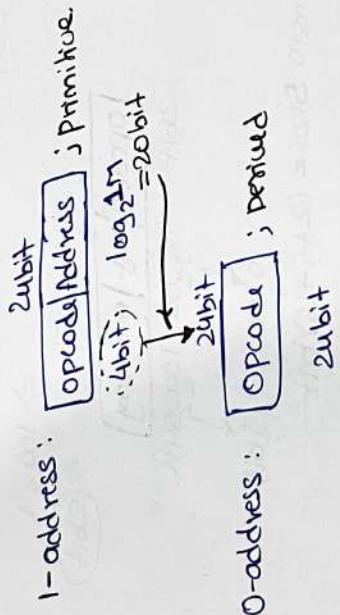
- Q1) consider 2ubit hypothetical CPU which support 2w long instruction placed in the 1MB memory. If there exist "12" 1-address instrn than how many 0-address instrn are formulated?

- word size = 24bit
- instr size = 16bit

\Rightarrow 2ubit; fixed size

Format:
 1-address }
 0-address }

expand
opcode
Technique.



① prioritise format : $\boxed{\text{opcode}} \boxed{\text{address}}$

1bit 20bit

- ② # opn possible $= 2^4 = 16$.
- ③ # free opcodes after 1-address. $\text{Instrn} = (16 - 12)$
 $= 4$

④ # 0-Address instrn $= 4 \times 2^{20}$

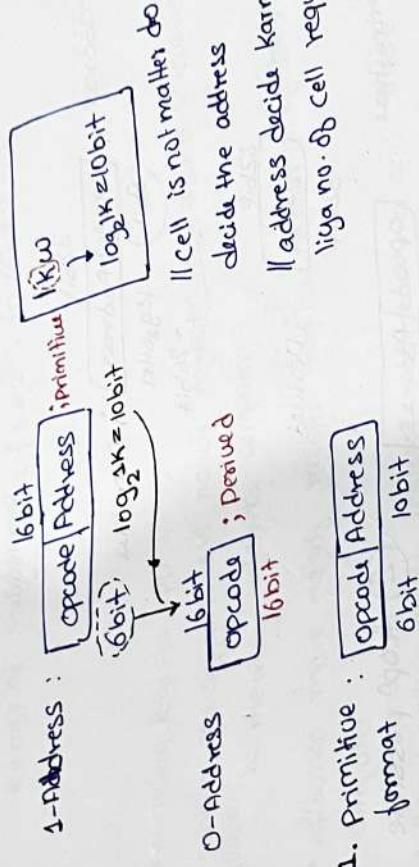
$\Rightarrow 2^{22} //$

Q) Consider 16-bit hypothetical CPU which support word long
 Instn placed in 1Kb memory || There exist "8" 1-address
 Instn then how many 0-address Instn are formulated?
 Instn

Format :
 1-address 1bit
 0-address

Sol: 16-bit CPU
 word = 16bit

Instn size = 2¹⁰
 16 bit length
 expand code technique



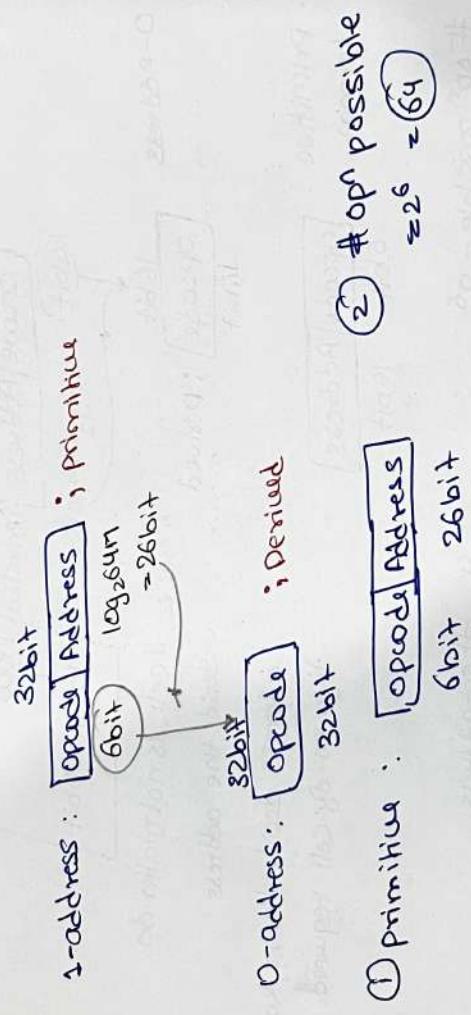
1. primitive : **opcode** | **Address**

format 6bit 10bit

2. # opⁿ possible = $2^6 = 64$
3. # free opcodes after 1-address = $(64 - 8)$ instruction = 56
4. # 0-address instn possible = $56 * 2^{10} = 56K$

(Q) consider 32 bit hypothetical CPU which support 2 word long instruction, placed in 32bit memory.
Support both 1-address & 0-address Instn what is the range of 1-address & 0-address Instn possible in the system.

- 32 bit CPU
- word = 32 bit
- Instruction size = $2^{32} = 32 \text{ bit}$ fixed length
 - Format: { 1 - address } { 0 - address }
 - Expand op code technique



- (3) If no of existing address instn no given then follow min & max min - address, instn = 2 & free address Instn is 63
 & max address is 63 & free address Instn is 1.

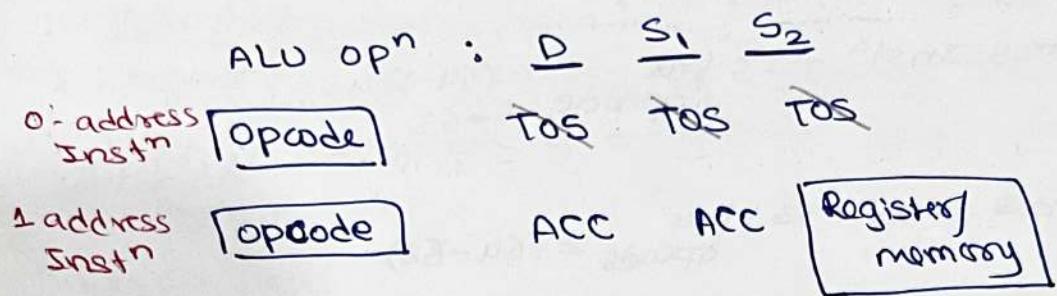
- share the opcode in sharing min value is 1 not 0.
- word range between & max has we are sharing bytes 1 address
- min address instr = 1, free = $(64 - 1)$
- opcode = 63
- = 63
- max 1-address instr = 63, free = $(64 - 63)$
- opcode = $(64 - 63)$
- = 1

so, Range of 1-address : 1 to 634

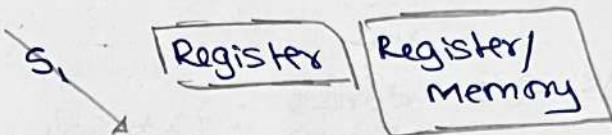
Range of 0-address : 1×2^6 to $(63 \times 2^6)^3$

- General Register CPU**
 - when CPU supports less no. of registers, then that is known as register to memory reference computer
 - If when more no. of registers than its register to register reference computer
 - Based on the no. of registers present in the CPU these organisation is further classified into 2 types
- ① Register - memory reference CPU
(CPU with less registers)
- ② Register - register reference CPU
(CPU with more registers)

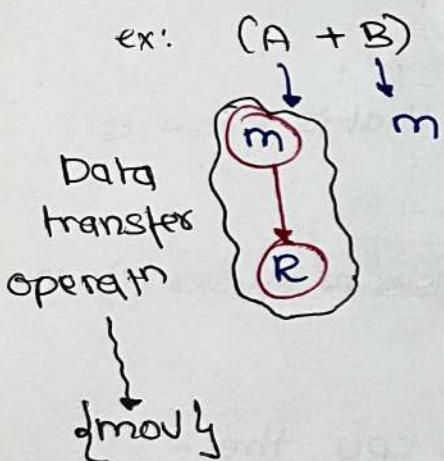
• Register - memory reference CPU (2 address format)



2 address Instn



Source itself act as destination so not need of giving the destination address



{general purpose}
data transfer
used for
Reading & writing

|| where as store is used only for writing & load is only used for reading in Register - memory reference.

I₁: MOV R₀, A ; R₀ ← M[A]

I₂: ADD R₀, B ; R₀ ← R₀ + M[B]

In Accumulator computer

I₁: load A

I₂: ADD B

In Stack computer

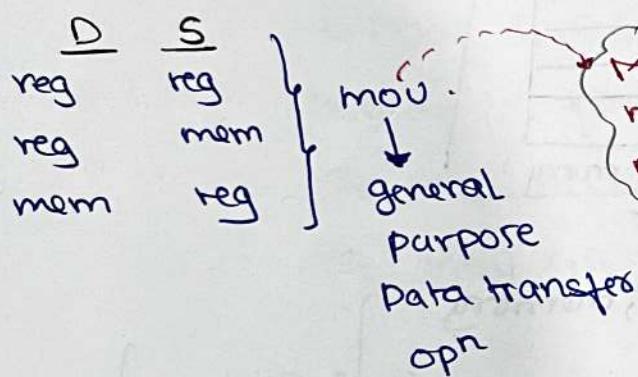
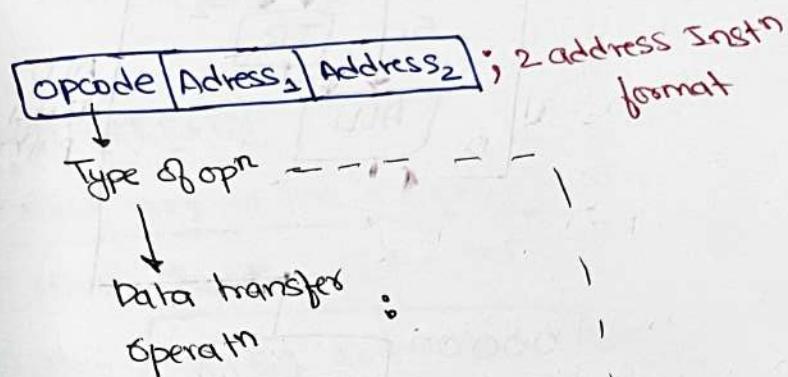
I₁: PUSH A

I₂: PUSH B

I₃: ADD

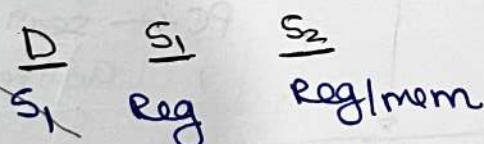
- In these organisation ALU 1st operand is always required in the register & the 2nd operand is either present in register or memory
- after the data processing, result will be placed into a source, so source is acting as destination, so destination address is not required in the instruction.

Instructⁿ Design :



MOV r₀, r₁
MOV r₀, [2000]
MOV [2000], r₀

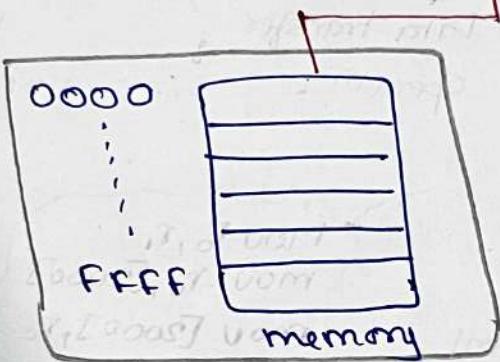
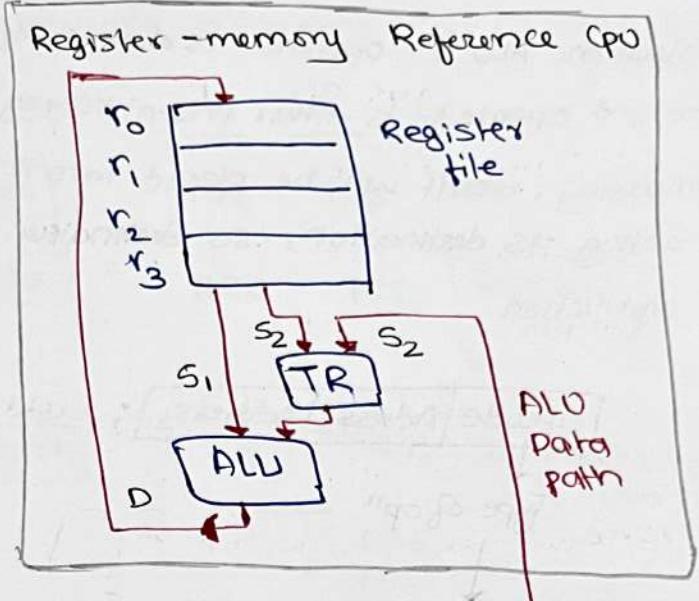
ALU operatn :



Add r₀, r₁
Add r₀, [2000]
Add [2000], r₀: X

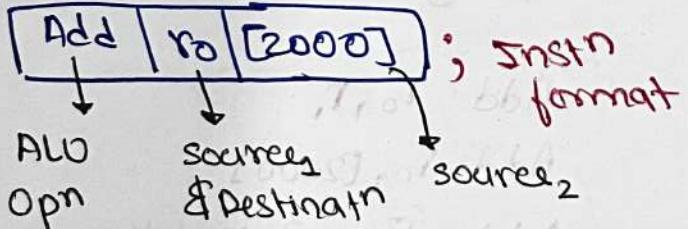
Invalid

1st operand is not present in the register.



Instⁿ : Add r₀, [2000]; memory

Add r₀, [2000]: R-m CPU



$$r_0 \leftarrow r_0 + M[2000]$$

fetch Cycle

Instⁿ cycle

Execution cycle

Ex: $(A * B) + C$

- Variable are in the memory

Code: $I_1: MOU r_0, A; r_0 \leftarrow M[A]$

$I_2: MUL r_0, B; r_0 \leftarrow r_0 * M[B]$

$I_3: ADD r_0, C; r_0 \leftarrow r_0 + M[C]$

Ex: $\boxed{X = (A+B) * (C+D)}$

- Variable are in the memory

$I_1: MOU r_0, A; r_0 \leftarrow M[A]$

$I_2: ADD r_0, B; r_0 \leftarrow r_0 + M[B]$

$I_3: MOU r_1, C; r_1 \leftarrow M[C]$

$I_4: ADD r_1, D; r_1 \leftarrow r_1 + M[D]$

$I_5: MUL r_0, r_1; r_0 \leftarrow r_0 * r_1$

$I_6: MOU X, r_0; M[X] \leftarrow r_0$

workBook chapter-3
Solution

(3)

$$(113. + -111.) + 7.51$$

2

9.51

$$113. + (-111. + 7.51)$$

-103.49
(round off)

-103

10

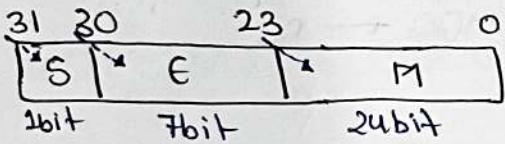
[Rounding can be done if the fraction part is below .5 but if it is above .5 then we can't round it]

ex: 9.49 ≈ 9

but

9.5 or 9.51 ≈ 9.5 or 9.51

(4)

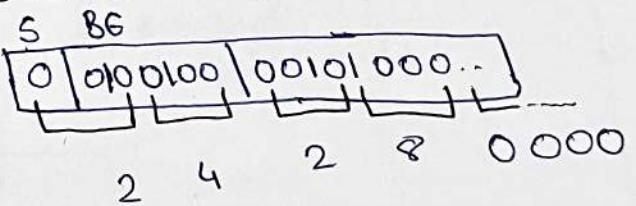


Bias = 31

the decimal no. : 0.15625×2^5

Hexadecimal representn

without normalizatn

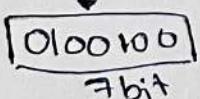


① Sign = true(0)

② BE = BE + Bias

$$= 5 + 31$$

$$= 36$$



③ $0.15625 \rightarrow 0.00101$

⑤ Hexadecimal format with normalized.

Data: 0.00101×2^5

(1.00101)

align to left upto

3 time

$1.01 \times 2^{5-3}$

1.01×2^2

① sign = +ve (0).

② BE = AE + Bias

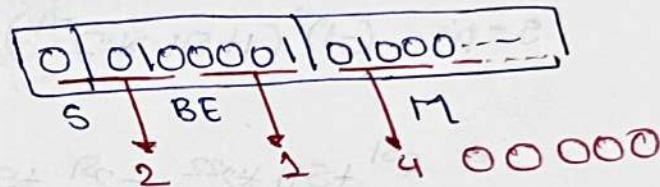
$$= 2 + 31$$

$$= 33$$

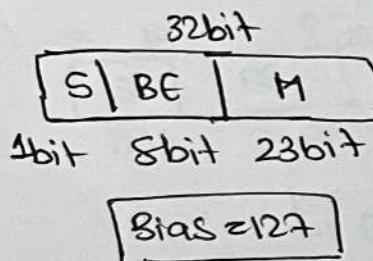
0100001

7bit

③ M: 1.01



⑥



Data: $1|0111111|101000\dots$

Value $(-1)^S (1.M) \times 2^{BE-Bias}$

$(-1)^1 (1.101) \times 2^{127-127}$

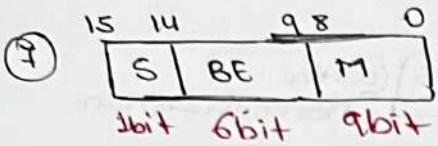
$-(1.101) \times 2^0$

-1.625

01100011

$11100011.F0000000$

$1.00110011000000000000000$



Data: $(-1)^S (1 + m \cdot 2^{-9}) 2^{E-31}$
 E: 111111 reserved for special.

- max. difference b/w two successive real no. represent^n

E: 111110 → 62

$$(S) \text{ Successive: } (-1)^S (1 + 511 \cdot 2^{-9}) 2^{62-31}$$

$$(P) \text{ Predecessor: } (-1)^S (1 + 510 \cdot 2^{-9}) 2^{62-31}$$

$$S-P: (-1)^S (1 + 511 \cdot 2^{-9}) \cdot 2^{31} - (-1)^S (1 + 510 \cdot 2^{-9}) \cdot 2^{31}$$

$$\Rightarrow 2^{31} + 511 \cdot 2^{22} - 2^{31} + 510 \cdot 2^{22}$$

$$\Rightarrow (511 \cdot 2^{22}) - (510 \cdot 2^{22})$$

$$\Rightarrow (511 - 510) \cdot 2^{22}$$

2^{22}

- ⑧ using Booth algorithm for multiplication the multipliers -57

$$57: 00111001$$

\downarrow
 2^5 comp

$$\begin{array}{r} 11000110 \\ \hline -57: 11000111 \end{array}$$

Bit pairs

Multiples

$q_0: 1$	$q_{-1}: 0$	$; -1 (\text{LSB})$
$q_1: 1$	$q_0: 1$	$; 0$
1	1	$; 0$
0	1	$; +1$
0	0	$; 0$
0	0	$; 0$
1	0	$; -1$
1	0	$; 0 (\text{MSB})$

pair with

q₋₁: 0

q₀: 1

1

0

0

0

1

0

0

0

1

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

⑨ n-bit data
 $x = \boxed{\text{integer} \mid \text{fraction}}$
 Integer fraction

ex:
 $x = \boxed{\begin{matrix} \text{5bit} \\ \text{2bit} \end{matrix} \mid \text{3bit}}$
 min: 00.000 (0)
 to
 max: 11.111 (3.875)

⑩ 0 to $(2^f - 2^{-f})$
 0 to 3.875

⑪

01001101	11101001	00110110
↓		MSB bit

Carry: 1
 overflow: 0 (reset)
 sign: 0

⑫

S	BE	M	
1bit	7bit	8bit	

Excess-64 Bias
 Expo.

Sol: ① Sign = true(0)

② BE = AE + Bias

$$= 13 + 64$$

$$= 77$$

$$= \boxed{1001101}$$

0.239×2^{13} w/o normalization

0	1001101	00111101	
4	0	3	D

$\Rightarrow \boxed{403D}$

③ $0.239 \rightarrow 0.\boxed{0011101}000101$

(12)

$$0.239 * 2^{13}$$

with
normalization

$$+0.0011101000101 * 2^{13}$$



(1.bbbb---)

Align to left '3' time.



$$+1.11101000101 * 2^{13-3}$$

$$+1.11101000101 * 2^{10}$$

$$\textcircled{1} \text{ Sign} = \text{true}(0)$$

$$\textcircled{2} \text{ BE} = \text{AE} + \text{Bias}$$

$$= 10 + 64$$

$$= 74$$

$$\textcircled{3} \text{ M: } 1.\boxed{11101000}101$$

0	1001010	11101000	
4	A	E	8

4AE8
//

(13)

S	BE	M
---	----	---

1bit 8bit 23bit.

Data:

3F800000

0	1111111	000 ---
S	BE	M

Value: $(-1)^S (1.M) * 2^{BE - \text{Bias}}$

$$(-1)^0 \cdot (1.0) * 2^{127 - 127}$$

$$+(1.0) * 2^0$$

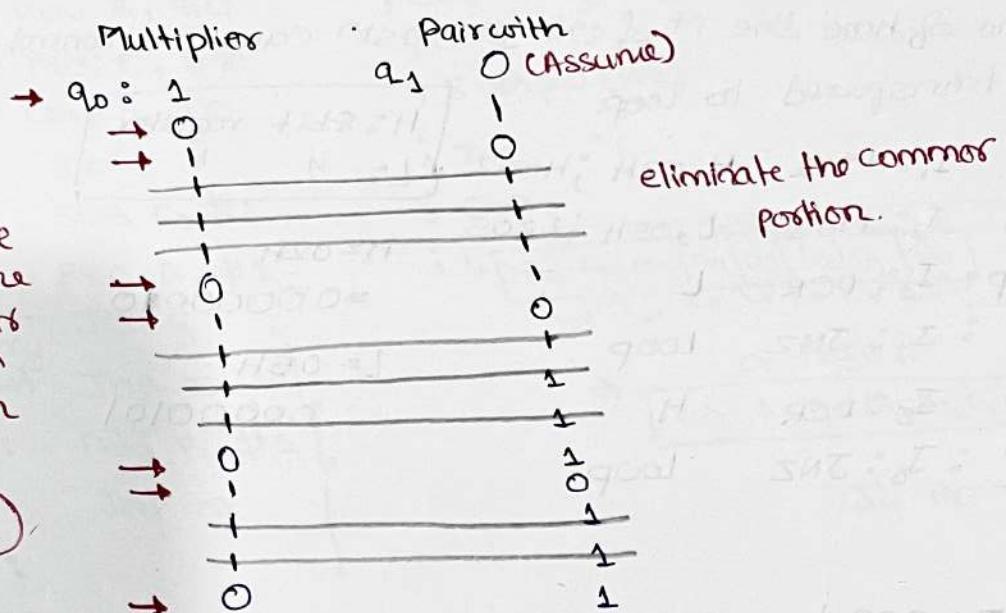
$\textcircled{1}$
//

Q14

Multiples to prepare the Bit pair

Opposite
Pair are
used for
arithm
operaotr

8 pair



(P33) 11111111

(G4) 225 : 1

(T) 5N

length

sd 1100

bit for work

G1000H

(T) -2A

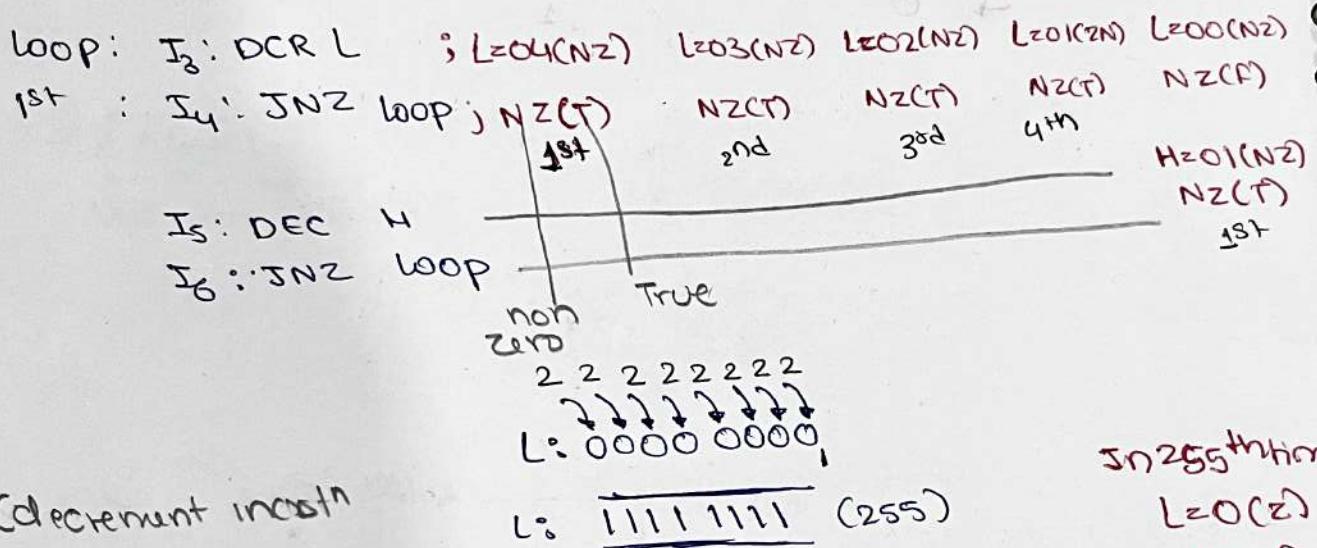
length of init pos is 17

min 13 bitsанс

Chapter 2

① no. of time the 1st & 2nd JNZ instr cause the control to be transferred to loop.

		H: 8bit register
I ₁ : MUL	H, 02H ; H=02	L: 00H
I ₂ : MUL	L, 05H ; L=05	H=02H =00000000
Loop: I ₃ : DCR	L	L=05H
1st : I ₄ : JNZ	loop	00000101
I ₅ : DCR	H	
2nd : I ₆ : JNZ	loop	



(decrement instrn
 invoke the
 subtraction operation
 means constant s is
 subtract from the
 register / memory
 contained, when register
 satisfied the min limit
 then value roll back to
 upper limit)

In 255th time
 L=0(Z)
 NZ(F)

L: 255(NZ)
 NZ(T) 255 time
 control will be transferred.

H=00(Z)
 NZ=F

1st JNZ = 259 time
 2nd JNZ = 1 time

given A, B, & C 8bit register.

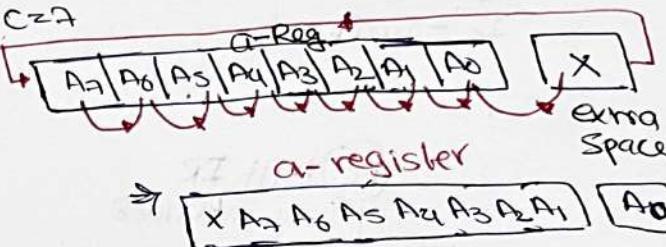
4
5
MOV B, #0
MOV C, #8
Z: CMP C, #0
JZ X
SUB C, #1
RRC A, #1
JC Y
JMP Z
Y: ADD B, #1
JMP Z
X:

Only true +
when carry
flag is set

; B=0
; C=8
; 8 compo (NZ)
; Z---F
; C=7
; LSB Bit

When both are same 0
(set)

not same 1
(reset)



a-register

X A7 A6 A5 A4 A3 A2 A1 A0

If A0 → 0 (no carry)
→ 1 (carry)

next iteration

JZ CMP0 (Z)

JZ Z --- T

Z = COMP C, #0 ; 1CMP0 (NZ)

; Z---F

; C=0

; A7 A6 A5 A4 A3 A2 A1 A0 X

If A7 → 0 (NC)
→ 1 (C)

to get the initial value

X: RRC A #1

In the A-register

• So objective of these
program is, reading the
A register data bit wise
after every bit reading
B-register will be increm'
when the A-register bit
status is 1. So at the end of
program execution B-register
contains the count of a no. of
1's in the A register.

17, 8 & 9

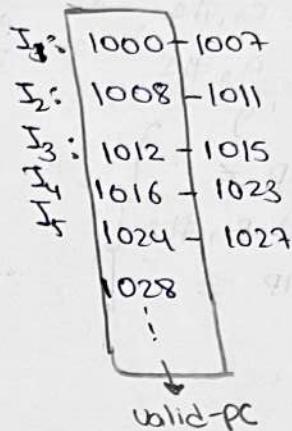
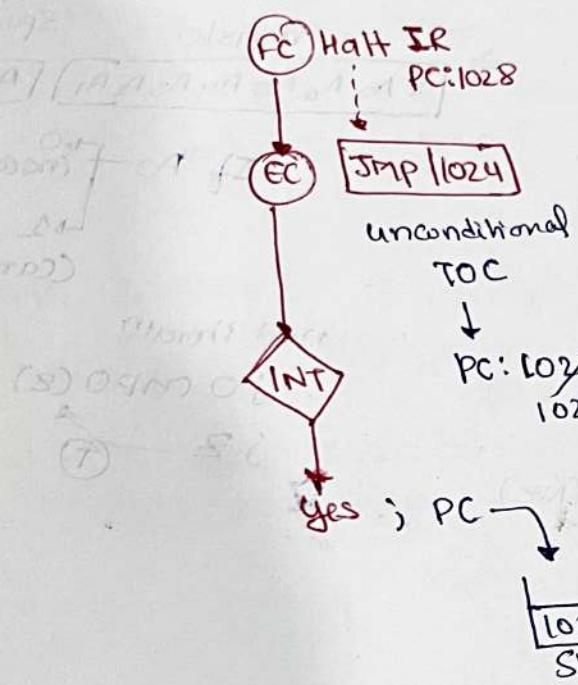
given: Byte Addressable

{cell size = 8bit}

word size = 32bit (4B)

Storage: Starting location 1000

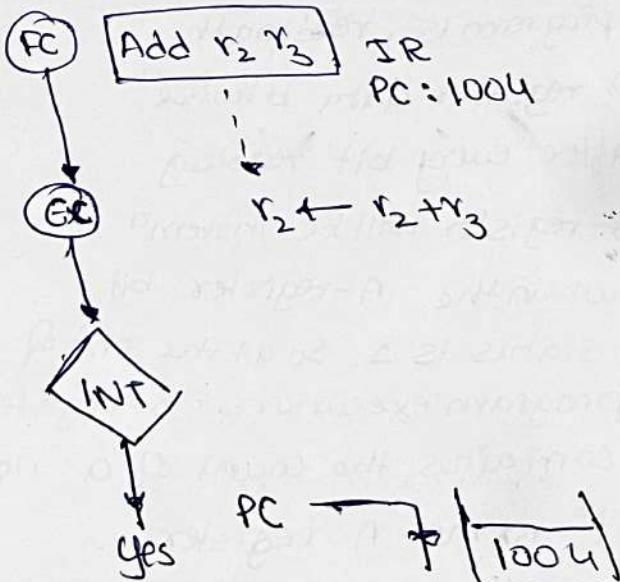
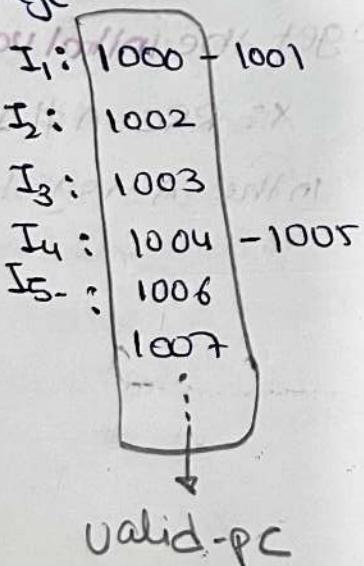
- 7) $I_1 = 200$
 $I_2 = 100$
 $I_3 = 100$
 $I_4 = 200$
 $I_5 = 100$ (halt)



∴ Halt is Self loop

- 8) word addressable
cell size = word

Storage



- when the instruction is an ALU instrⁿ, then during the execution cycle program counter is not updated
- But JMP instructn update the PC.

Q) IF + ID Processing based on the Instruct we are performing at the given cycle for that.

$I_1: 2W * 2C$	3cycle
$I_2: 1W * 2C$	3cycle
$I_3: 1W * 2C$	1cycle
$I_4: 2W * 2C$	3cycle
$I_5: 1W * 2C$	—
	<u>10cycle</u>
	14110 ⇒ 24cycle

13 & 14 Expand opcode Technique

16bit Instⁿ (fixed)
 2-address format
 1-add^r format

expand
opcode
Techⁿ

① primitive Instⁿ

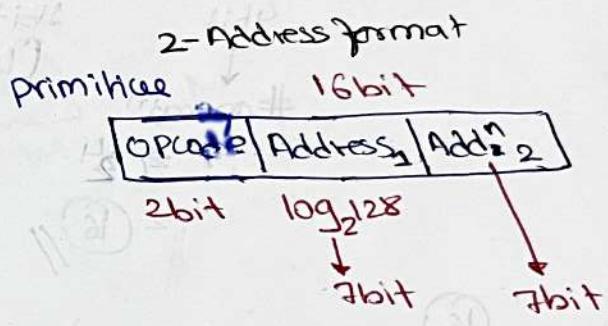
Opcode	Add ₁	Add ₂
2bit	7bit	7bit

② # opⁿ = $2^2 = 4$

we find range as existing of any address is not given.

so, Range of 2-address Instⁿ = 2^1 to 3^2

Range of 1-address Instⁿ = $\{ (1 * 2^1) \text{ to } (3 * 2^1) \}$



1-address format.
 Derived ;

Opcode	Address
9bit	7bit

Share the opcodes

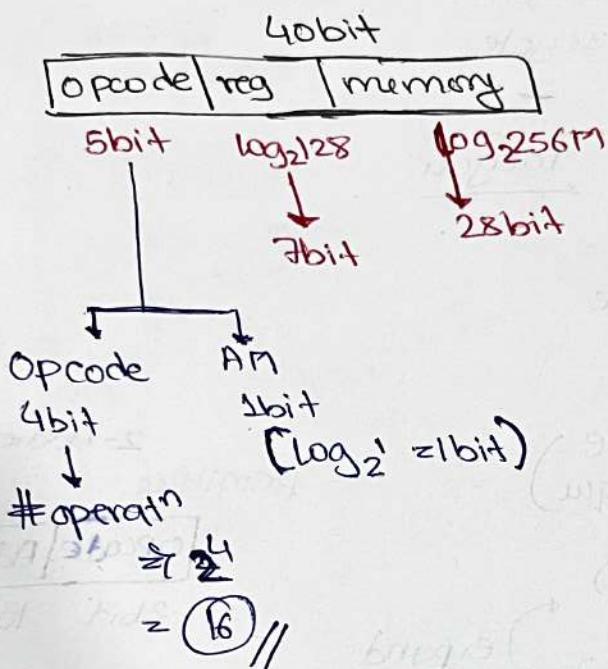
2ⁿ address 1 address

1	---	3
3	---	1

$$\# \text{ 1-address} \\ \text{instn} = 2^k \\ \text{possible} \\ = 256$$

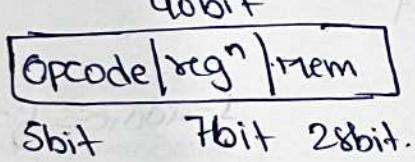
15, 16 & 17

15.

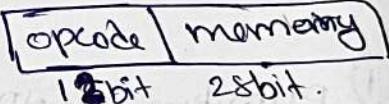


(16.)

2-address Instructn



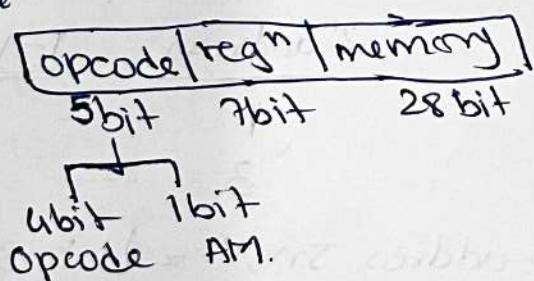
i-address
Instr



perived

6

primitive



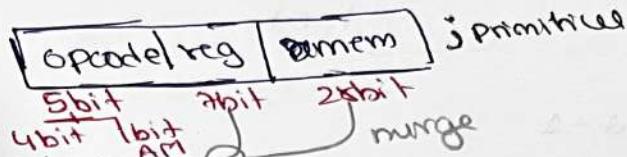
② # opⁿ possible : 2^4

③ no. of free opcodes
after 2-address $\Rightarrow 2^{4-n}$

④ # 1-address instrⁿ $\Rightarrow (2^{4-n}) * 2^7$ $\xrightarrow{\text{merging register}}$

⑦

2-address
instr



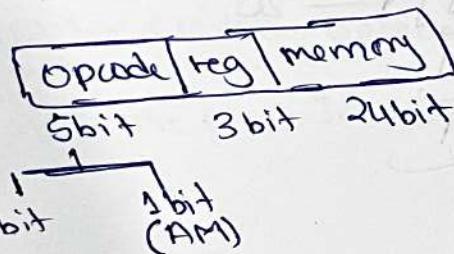
1-address
instr



0 address:



① primitive:



② # opⁿ possible = 2^4

③ # free opcode
after 2-address $\Rightarrow 2^{4-n}$

④ # 1-address instrⁿ possible

$$\xrightarrow{(2^{4-n}) * 2^{28}}$$

⑤ # free opcodes

after 1-address instrⁿ $\Rightarrow ((2^{4-n}) * 2^{28} - m)$

⑥ # 0 address instrⁿ
possible $\Rightarrow ((2^{4-n}) * 2^{28} - m) * 2^7$

(18) $M[1000] = 18$
 $M[1001] = 1$
 $M[1020] = 16$

$I_1: R_S = 1$
 $I_2: R_d \leftarrow M[1000 + [R_S]]$

$\underbrace{M[1000]}_{1001} + \underbrace{[R_S]}_{1}$
 $\underbrace{\qquad\qquad\qquad}_{1}$

$R_d = 1$

$I_3: R_d \leftarrow \underbrace{R_d}_{1} + 1000$

$\underbrace{\qquad\qquad\qquad}_{1001}$

$\boxed{R_d = 1001}$

$I_4: M[0 + [R_d]] \leftarrow 20$

$\underbrace{M[0]}_{1001} + \underbrace{[R_d]}_{1001} \leftarrow \underbrace{20}_{X_{20}}$

So, $M[1001] = 20$

(19) $R_1 \rightarrow M[100]$
 $M[100] \rightarrow R_2$
 $M[100] \rightarrow R_3$

$R_1 \rightarrow R_2$
 $R_2 \rightarrow R_3$
 $R_1 \rightarrow M[100]$

What is the necessity of
accessing the memory
3 times when R_1, R_2, R_3

- All are present in the CPU
- So 1st transfer the data internally, later update the memory

} which reduce memory reference & speedup performance of CPU.

(24) RISC processor has 12 registers windows
 Total no. of registers in the processor.
 16 global register.
 Each window has 8 I/P 16 local & 8 O/P register.

$$(\text{Register file}) = w(L+C) + G$$

$$\Rightarrow 12(16+8)H16$$

$$\Rightarrow 304.$$

(27) RTE (Return from Exception) is equivalent to RFI

(Return from Interrupt) / Interrupt return

So every interrupt return sub-program must end with
 interrupt return. It must be privileged instruction
 & there should be a valid opcode required for IRET instruction

while execution of the IRET instruction no other interrupt are entertain

(30) Program

```

1
2
3
4
5 : x(new)
6 : x(I/P)
7 : x(I/P)
8 : x(new)
9 : x(I/P)
10
11
12 : x(new)
13
14
15
16 : x(I/P)
17 : x(I/P)
18
19
20
21
22
23
24
25

```

x(new) Fresh value

x(I/P) x is used as I/P

// whatever variable is used by the
 x variable the same variable
 register is used by how many other
 variable / Stmt.

stmt	<u>Reg 1</u>	<u>Reg 2</u>
6	(X)	7 Stmt
7 Stmt	8 Stmt	13
8 Stmt	9 Stmt	14
9 Stmt	X	15
X	10	16
10	H	
H	12 : X	
12 : X	13 Stmt	
13 Stmt		
		25

These all
 Instn
 use same
 registers.

// before reusing
 the register
 take the existing
 data in the
 registers
 whether
 that data is
 used in
 future or not
 if not required
 then replace.

Q33 | Q34 | Q35

given: $M[3000] = 10$
 $M[2000] = 100\ 110$

$M[2009] = 100$

$M[2010] = 100$

Program Address = 1000

Program:

I ₁ :	MOV r ₁ , [2000]; r ₁ = 10	1MR
loop : I ₂ :	MOV r ₂ , (r ₃) 1MR ; r ₂ $\leftarrow M[r_3]$; r ₂ = 100 2000 100
	I ₃ : Add r ₂ , r ₁	; r ₂ = 110
10 times	I ₄ : MOV (r ₃), r ₂ 1MR ; M[r ₃] $\leftarrow r_2$; M[2000] = 100 110 100 110
	I ₅ : INC r ₃ ; r ₃ = 2001	
	I ₆ : DEC r ₁ ; r ₁ = 9(NZ)	
	I ₇ : (BNZ loop) - NZ	
	I ₈ : (HALT).	

So, Increment the address by 1 & Increment the value by 20. So, from 2000 to 2009 all values are 110.

• Total memory reference to access the data in the program = 1MR + (2MR * 10 times)
 $= 21MR$

• In the program execution 10 memory cells are accessed i.e [2000] to [2009] ∵ no change in [2010] cell
∴ So, [2010] = 100.

35. Byte Addressable
(cell size = 8bit)
word size = 32bit (4B)

Storage:

I ₁ :	1000 - 1007
I ₂ :	1008 - 1011
I ₃ :	1012 - 1015
I ₄ :	1016 - 1019
I ₅ :	1020 - 1023
I ₆ :	1024 - 1027
I ₇ :	1028 - 1035
I ₈ :	1036 - 1039
	1040

Instructn
size

2

1

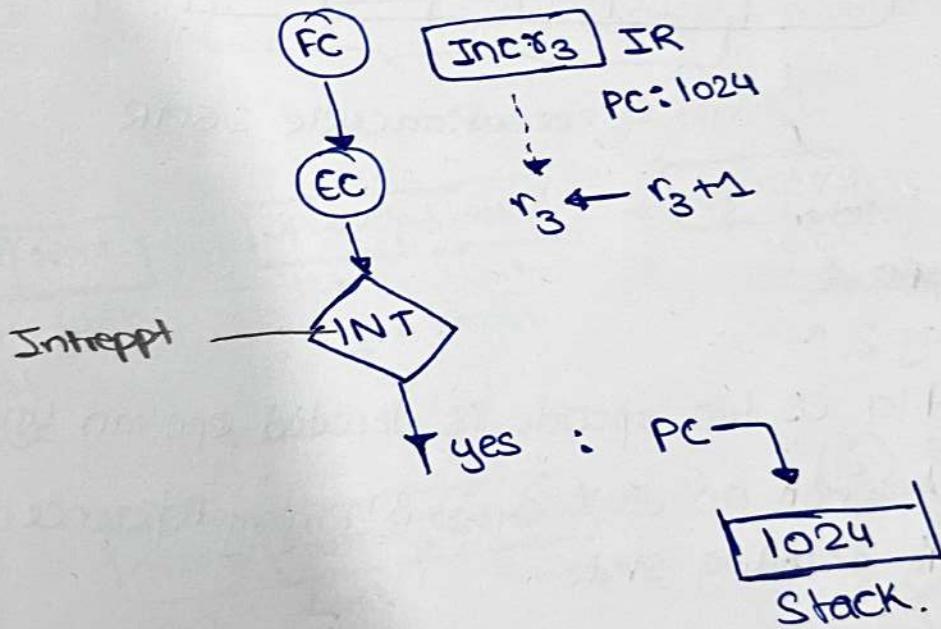
1

1

Halt at
Incrnt
Instruction

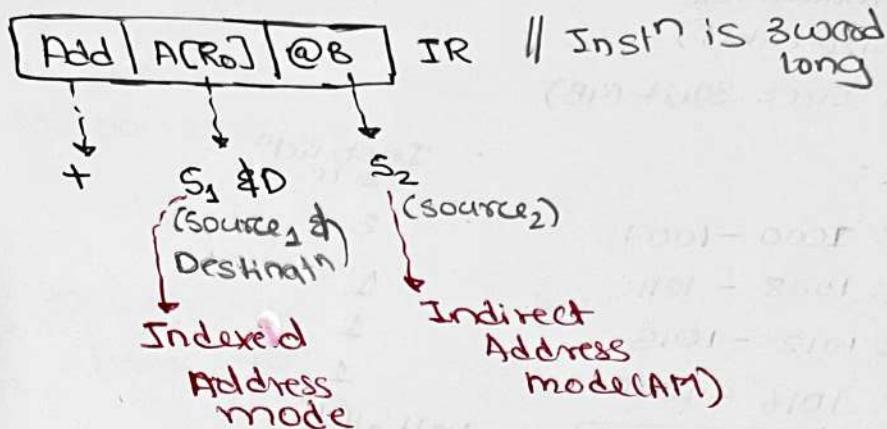
2

1



(36)

Given:



FC	EC				
	IF	ID	OF	PD	WB
1MR	2MR	S ₁ 1RR IAU 1MR	S ₂ 2MR	1ALU	D IRR IAU 1MR

no. of execution cycle : 6MR

1word is fetched
in the FC &
remaining 2 are

fetched in EC b/c opcode is decoded CPU can understand length
of instruction as 3word \therefore 3 Memory Reference (MR) are counting
under the decoding state

(37) Instruction size = 16bit hence total encoding = 2^{16} . will be

size of address field is = 4bit. so encoding will be 2^4

this machine use 0, 1, 2 address instruction.

34 2-address Instⁿ will take = $34 * 2^4 * 2^4$ encoding

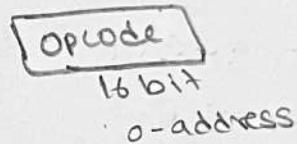
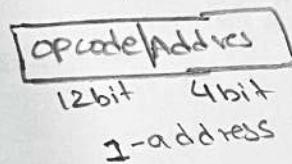
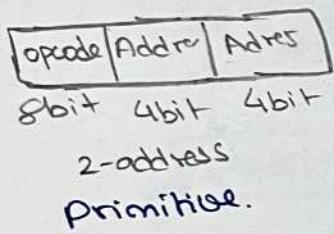
100 1-address Instⁿ will take = $100 * 2^4$ encoding.

N 0-address Instⁿ will take =

$$(2^{16} - (34 * 2^4 * 2^4 + 100 * 2^4))$$

encoding = 55232 which will also be the total 0-address instruction

Expand
Opcode
method



① primitive : Instⁿ

opcode	Address	Address
8bit	4bit	4bit

⑥ # 0-address Instⁿ possible $\nexists 2^4 * 3452$

55232

② # OPⁿ possible = $2^8 = 256$

③ # free opcodes after 2nd address = $(256 - 34) \Rightarrow 222$

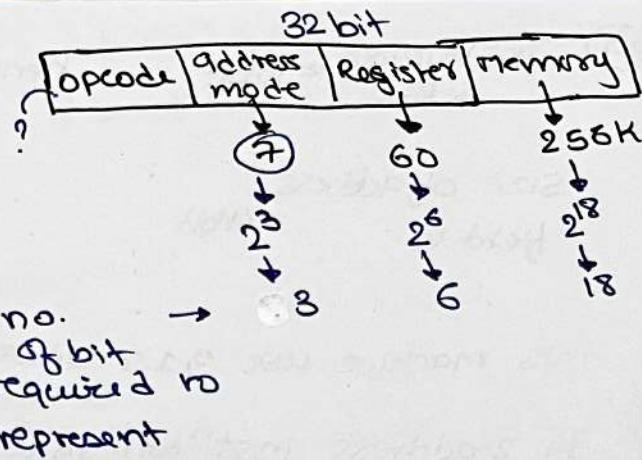
④ # 1-address Instⁿ $\nexists 2^4 * 222 \Rightarrow 3552$
possible

⑤ # free Opcode after 1-address Instⁿ = $(3552 - 100)$
 $\Rightarrow 3452$

(38) 32-bit Instruction.
the instruction format
has 4 field.

$$80, \text{ opcode} = 32 - (3 + 6 + 8) \\ \Rightarrow 5.$$

$$80, 2^5 = \underline{\underline{32}} //$$



20101 and address mode [0000] - problems
minimum 24 bits

Instruction (8)

address mode

register & P

~~00000000~~

Instruction word | 00000000

0000 0000 0000 0000

00000000

0000 0000 0000 0000

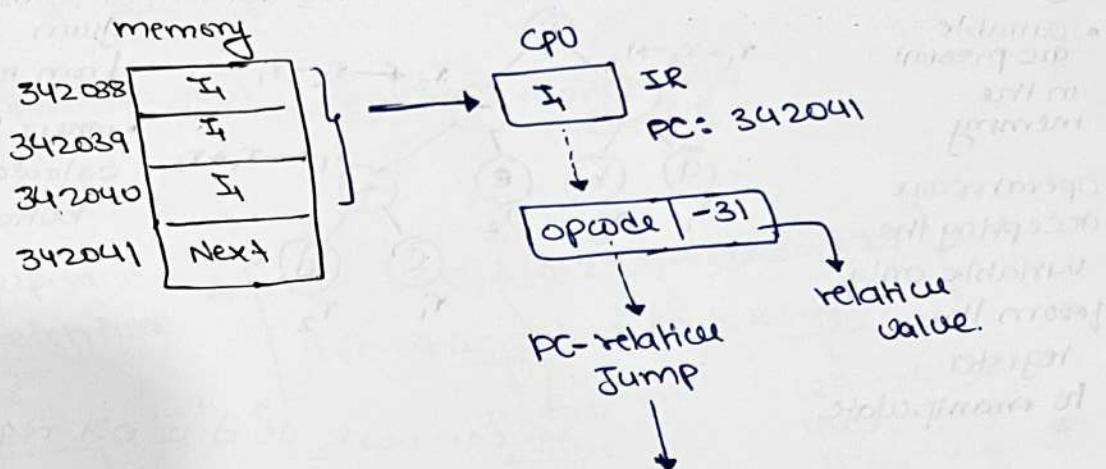
0000 0000 0000 0000

0000 0000 0000 0000

0000 0000 0000 0000

0000 0000 0000 0000

39
40

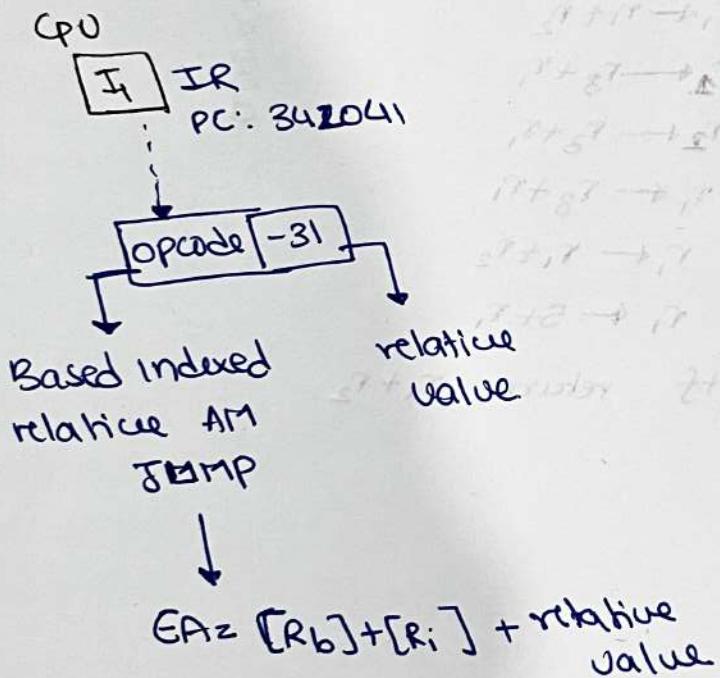


$$\text{Effective address} = \text{PC} + \text{relative value}$$

$$\rightarrow 342041 + (-31)$$

$$\rightarrow 342010.$$

40.



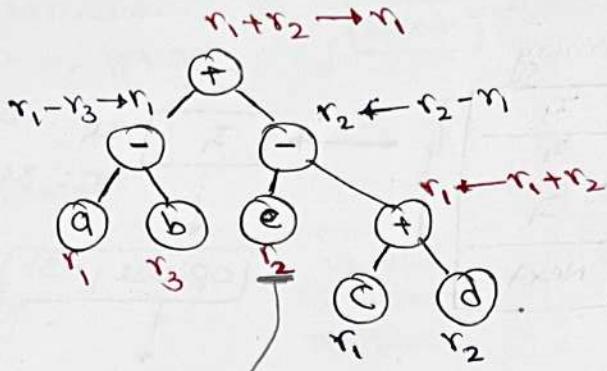
$$EA = [R_b] + [R_i] + \text{relative value}$$

$$\approx 480220 + 9 + (-31)$$

$$\approx 480198$$

(41)

- Variable are present in the memory
- Operators are accepting the variable only from the register to manipulate.

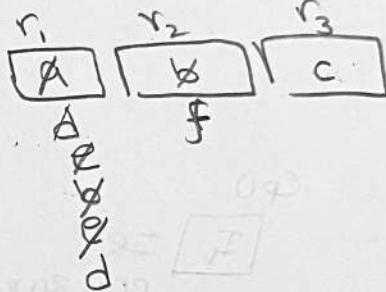


- when program given calculate from top to bottom
- when tree given calculate from bottom to top

we can use r_2 as "d" is not required
in the future.

(42)

- $I_1 : a = 1 \quad r_1$
 $I_2 : b = 10 \quad r_2$
 $I_3 : c = 20 \quad r_3$
 $I_4 : d = a + b \quad r_1 \leftarrow r_1 + r_2$
 $I_5 : e = c + d \quad r_1 \leftarrow r_3 + r_1$
 $I_6 : f = d + e \quad r_2 \leftarrow r_2 + r_1$
 $I_7 : b = c * e \quad r_1 \leftarrow r_3 * r_1$
 $I_8 : e = b * f \quad r_1 \leftarrow r_1 * r_2$
 $I_9 : d = 5 * e \quad r_1 \leftarrow 5 * r_1$
 $I_{10} : \text{return } d + f \quad \text{return } r_1 + r_2$



so $addr + [addr + [addr]] = 143$
etc etc

(10) i. Processor

8P103N3

(43)

- Avg = $\frac{\text{Sum}}{\text{Count}}$
- information given $= \sum \text{all \%} = \text{relative value}$
in % (100%)

Avg. Operand fetch time \Rightarrow

$$\left[(0.3 * 0) + (0.2 * 0) + (0.22 * 2) \right]$$

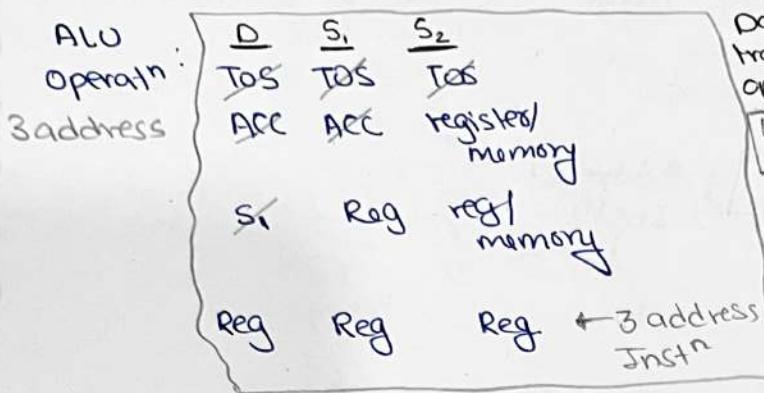
$$+ (0.19 * \frac{4 \text{cycle}}{\downarrow}) + (0.11 * \frac{3 \text{cycle}}{\downarrow})$$

↓
 1MR take
 2cycle
 +
 2MR take
 4cycle

IRR - 0cycle
 IALU - 1cycle
 IMR - 2cycle

$\boxed{1.45}$ //

• Reg - Reg reference CPU (3-address format)



reg-reg computer

I₁: Load r₀, A ; r₀ ← M[A]

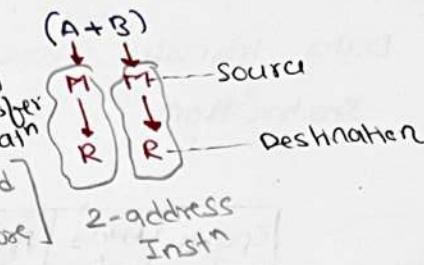
I₂: Load r₁, B ; r₁ ← M[B]

I₃: Add r₂, r₀, r₁; r₂ ← r₀ + r₁

Accumulator computer

I₁: Load A

I₂: ADD B



reg-memory computer

I₁: MOV.r0, A

I₂: ADD.r0, B

Stack computer.

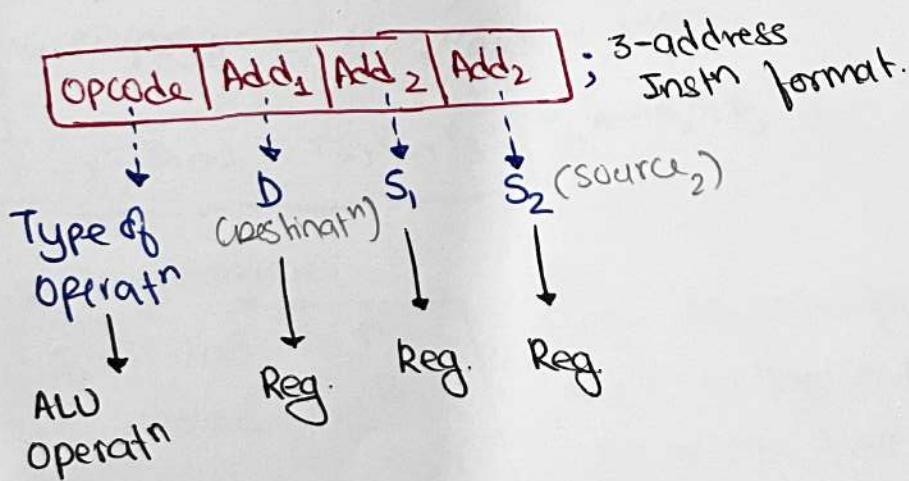
I₁: push A

I₂: push B

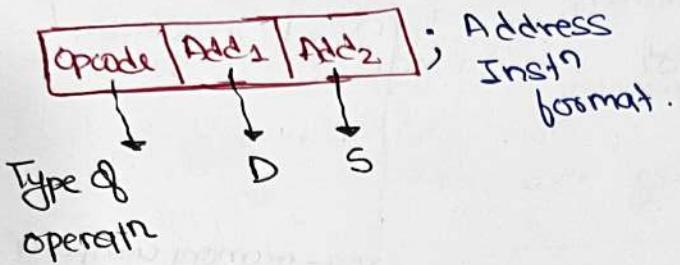
I₃: ADD

- In these organisation ALU operations are performed only on a register. Data means both of them operands are always required in the register. After the data processing, result is also placed in the register.

• Instruction Design:

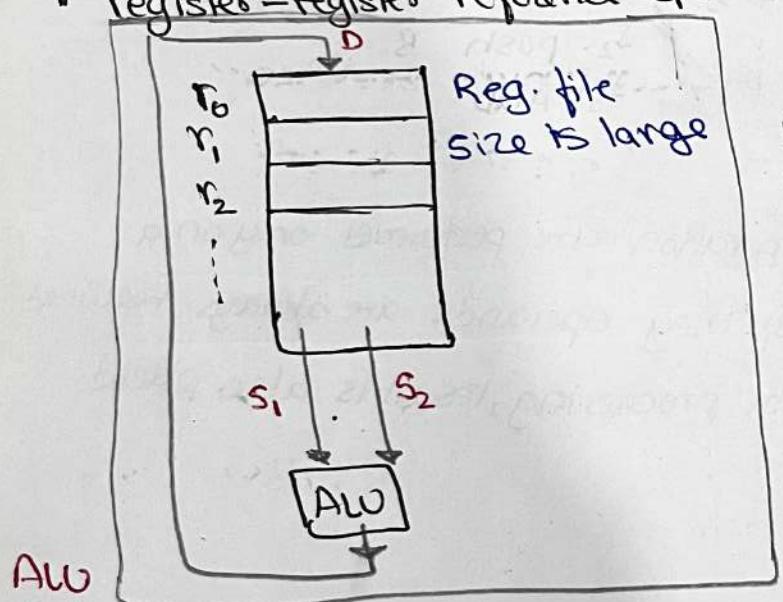


③ In the organisation Load & Store instructions are used as a Data transfer instruction these are design with 2-address Instruction.



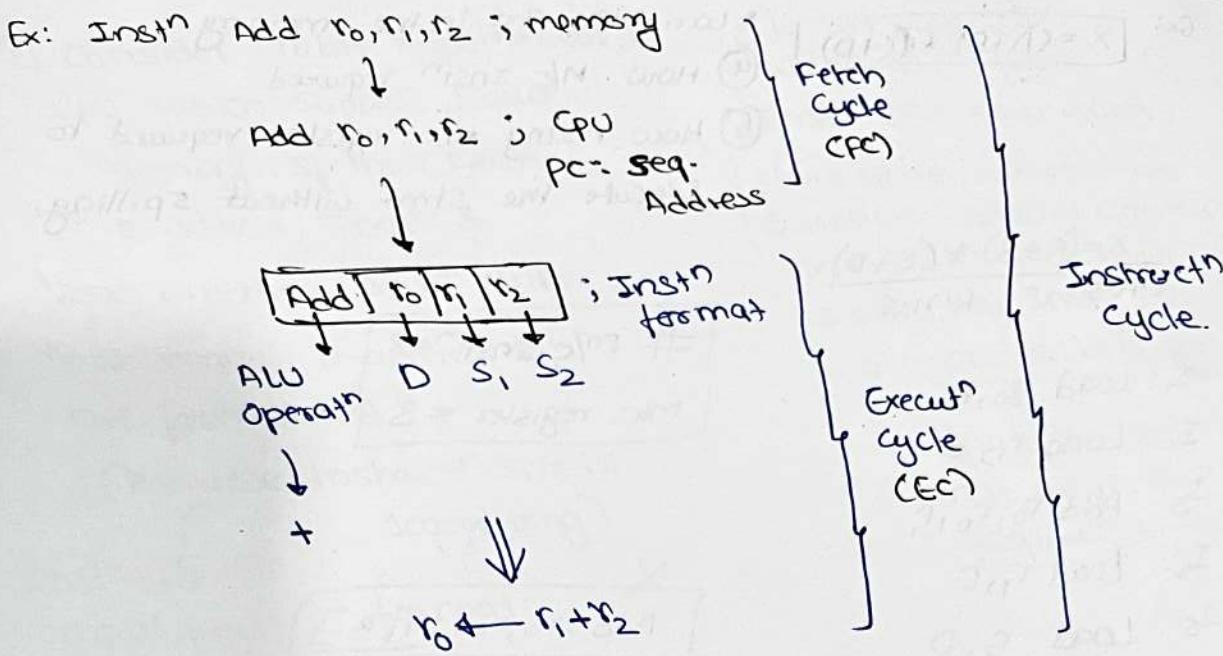
Data transfer :
 Operatn Reg \leftarrow memory (load)
 memory \leftarrow Reg (store)

register-register reference CPU (more registers)



ALU
Data path

\therefore no need of TR as ALU is not accepting data from memory.
 According to Von-neumann Architecture program will be present in memory so 1st we need to perform Data transfer operation from memory to register & then we can perform data manipulation when it is present in register.



II Instruction format is a reference element in the EC &
PC is reference element in the FC

III variable is name of memory cell (high level langn)

IV in low level langn memory cell are address with address.

Ex: (A * B) + C • variables are in the memory.

I₁: Load r₀, A ; r₀ ← M[A]
 I₂: Load r₁, B ; r₁ ← M[B]
 I₃: MUL r₂; r₀, r₁ ← r₀ * r₁
 I₄: Load r₃, C ; r₃ ← M[C]
 I₅: Add r₄, r₂, r₃; r₄ ← r₂ + r₃

II we can also reuse the registers

reuse the registers.

I₁: Load r₀, A
 I₂: Load r₁, B
 I₃: MUL, r₀, r₁, r₀
 I₄: Load, r₁, C
 I₅: Add , r₀, r₀, r₁

Reg. reusability
used in the code
to mini. the
register.

Ex:

$$X = (A+B) * (C+D)$$

• variables are in the memory.

④ How M/C instr required

⑤ How many reg. required to execute the stmt without spilling

$$\frac{X = (A+B) * (C+D)}{\text{L.H.S} \quad \quad \quad \text{R.H.S}}$$

I₁: Load r₀, A

I₂: Load r₁, B

I₃: Add r₀, r₀, r₁

I₄: Load r₁, C

I₅: Load r₂, D

I₆: Add r₁, r₁, r₂

I₇: MUL r₀, r₀, r₁

I₈: Store X, r₀

M/C instrⁿ = 8

min register = 3

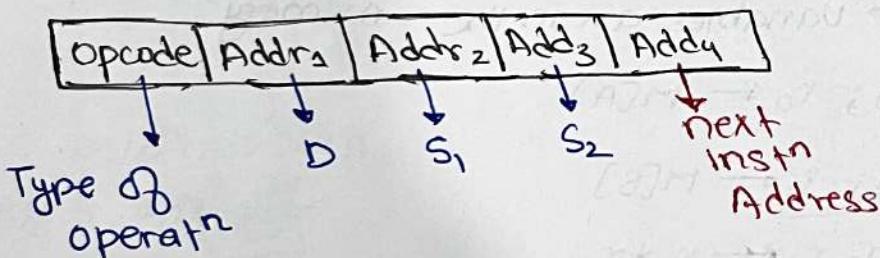
~~Add. r₀, r₀, MCB]~~

not present in register.

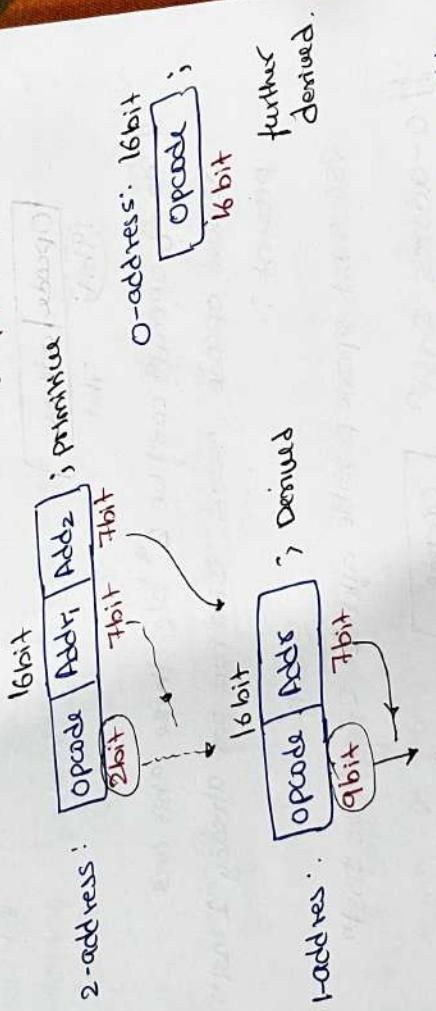
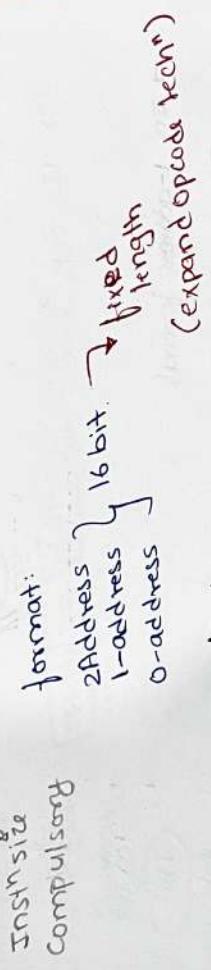
note

• PC is a compulsory register in the CPU. Used to hold the address of next instr during the execn of a current instr.

If 4-address format is not in use.



- ① Considers 16-bit hypothetical CPU which support 12800 memory. If there exists "12" 2-address Instn & "1252" 1-address Instn then How many 0-address Instn are formulated?
 - Assume Instruction size is word long
 - Instruction format:
- Compulsory



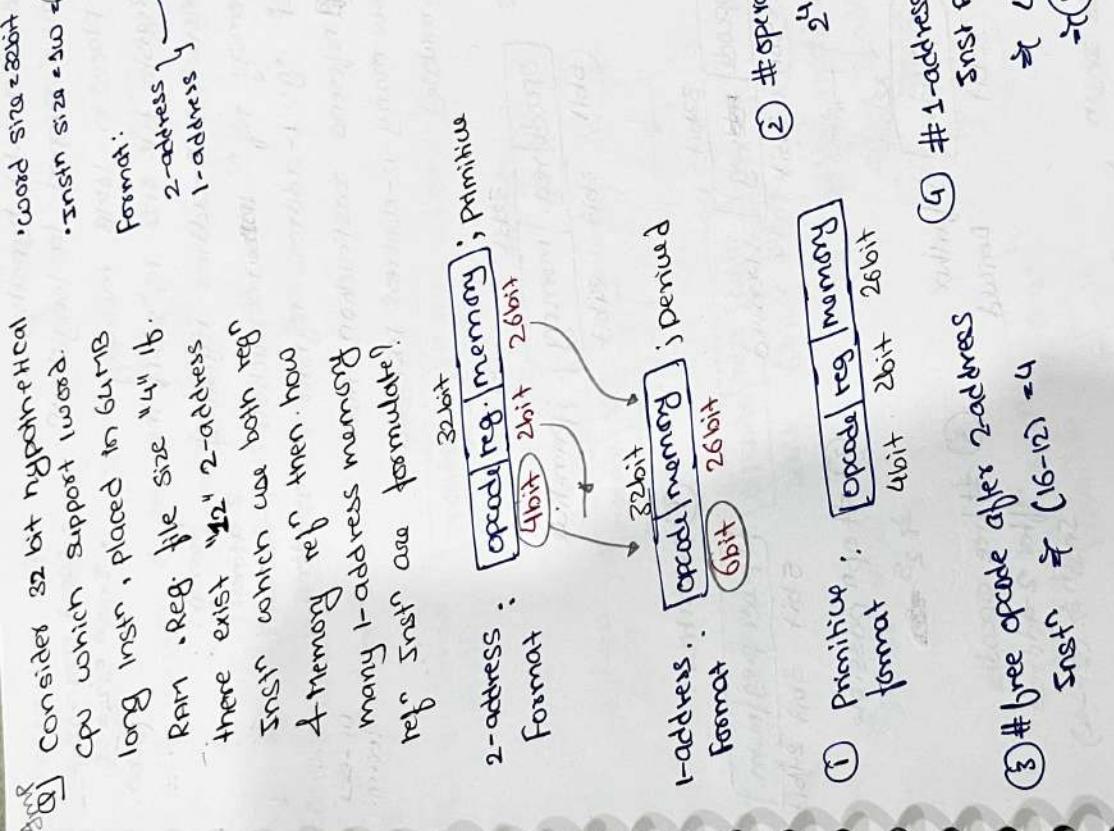
$$\# \text{ op} = 2^4$$



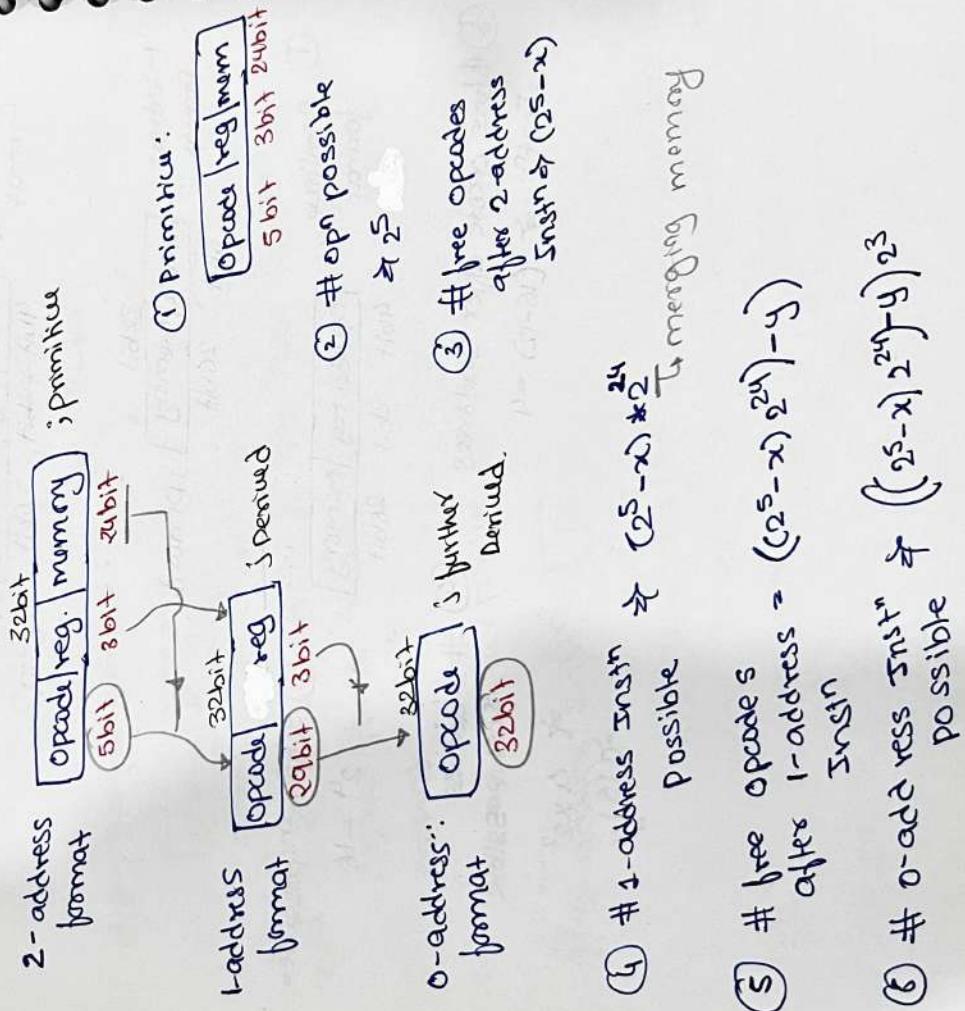
① Primitive format

- ② # free opcode after : $(4-2) = 2^2$.
- ③ SNS
2-address

- (4) # 1 address instr : 2^{*2^7}
 $\Rightarrow 2^8 = 256$
- (5) # free - opcode after 1-address instr = $(256 - 252)$
 $\Rightarrow 4$
- (6) # 0-address $\Rightarrow 4 * 2^7$
 $\Rightarrow 2^9 = 512$ //
- 11 so 128 cells & each cell contain 512
 \rightarrow in 1-address format
- Opcode | Address
7-bit 7-bit
- 11 no. of opcodes can't be 2^9 b/c there about 512
 derived opcode means is hard to per already 1 instr
 present
- 256 8-bit opcode possible after "2" 2-address instr
- || 0-address instr Opcode 6-bit 5-bit
5-bit 5-bit
- 512 16 bit opcodes
 one possible diff
 i.e. 2-address instr
 2^{52} 1-address instr
- 5-bit
 # 2-addrs instr : 2
 # 1- " " : 252
 # 0- " " : 512



Q1 Consider 32bit hypothetical CPU which support two long instn
Placed in 16MB memory
Registers file size is "8". If there exists "x" 2-address registers & memory ref instruction & "y" 1-address registers reference instruction then how many 0-address instrns are formulated.



Sol:
- word size = 32bit
• Instn size = 1024bit = 32bit.

fixed length

expand
opcode
Technique

format:

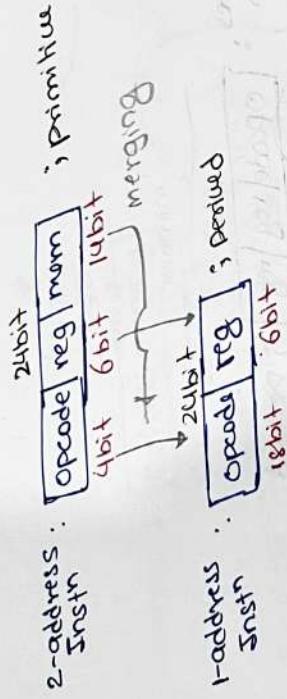
2-address

1-address

0-address

(Q1) considers 2-bit hypothetical CPU which supports no long branch placed in 16KB Ram System support 8 registers & the exist "12" 2-address registers of memory instn then how many 1-address register references instn are formulated?

word size = 24bit
instn size = 24bit
fixed length
Format:
2-address
1-address
expand
opcode
Techn

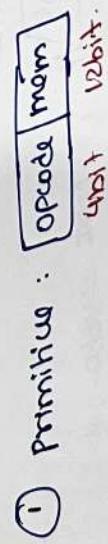
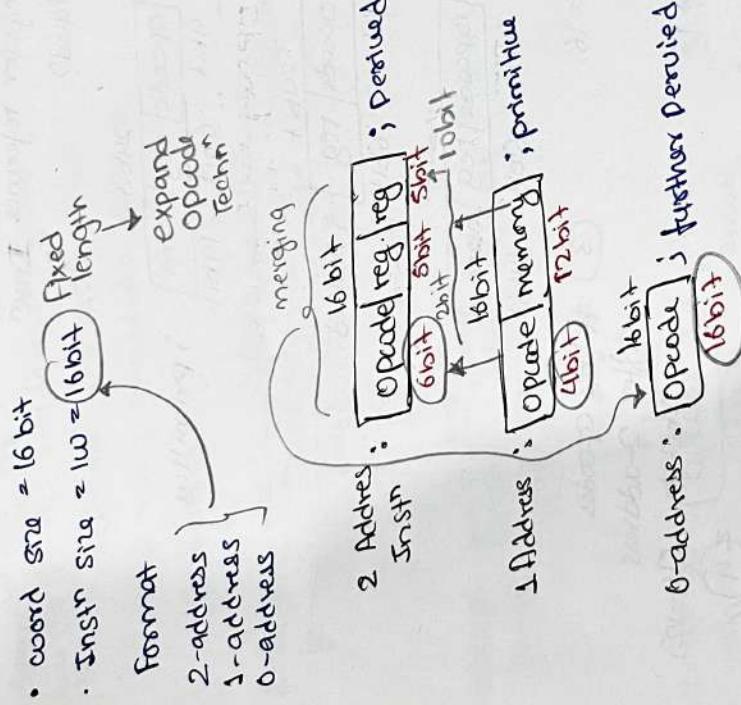


- (1) primitive

4bit	OpCode	reg mem
Instn	6bit	18bit

; derived
 - (2) # opns = $2^4 = 16$
 - (3) # free opcodes after 2-addrs instn - 1
 - (4) //
- Q4 # 1 address instn $\Rightarrow 4 * 2^4$ possible

Assume consider 16 bit hypothetical CPU which supports 16 long
 words, placed in 4KB RAM. Reg file size in the CPU is 32
 bits. If there exist "16" 2-address registers reference such
 "16" 1-address memory references than how many 0-address
 can be formulated?



$$\text{② } \# \text{ opn possible} = 2^4 = 16$$

$$\begin{aligned} \text{③ } \# \text{ free opnd} &= (16 - 16) \\ \text{after 1-address} &- 2 \end{aligned}$$

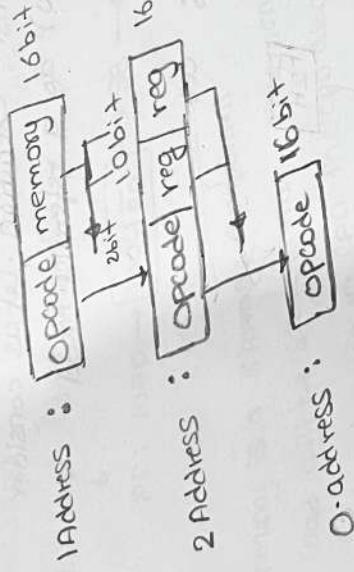
$$Q. \# \text{ 2 address} = 2 * 2^2 = 8$$

instn

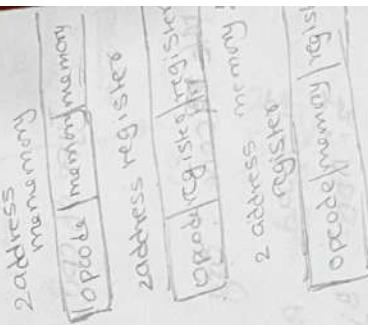
$$Q. \# \text{ free opode} = 2^{(8-4)} \\ \text{ after 2 address} = 4$$

instn

$$Q. \# 0\text{-addr, instn} = 4 * 2^0 \\ \text{possible} \rightarrow 2^0 //$$



O-address :



1000
000 - 200
1000 - 2000
1000 - 2000
1000 - 2000

1000
000 - 200
1000 - 2000
1000 - 2000
1000 - 2000

1000
000 - 200
1000 - 2000
1000 - 2000
1000 - 2000

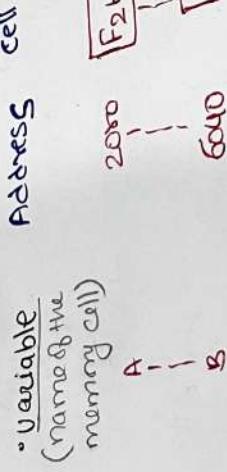
1000
000 - 200
1000 - 2000
1000 - 2000
1000 - 2000

- Instruction Execution sequence
 - it contains Fetch cycle & Execution cycle → processing of the instruction & to process the instruction we need
 - the instrn from the memory & its what PC (Program counter) required.

- 8085 CPU is an Accumulator CPU & 8085 has an register to memory CPU.

- to analyze the instrn execution let us consider 8085 MP (ACC-CPU) as a reference model

- Expression : $(A+B)$

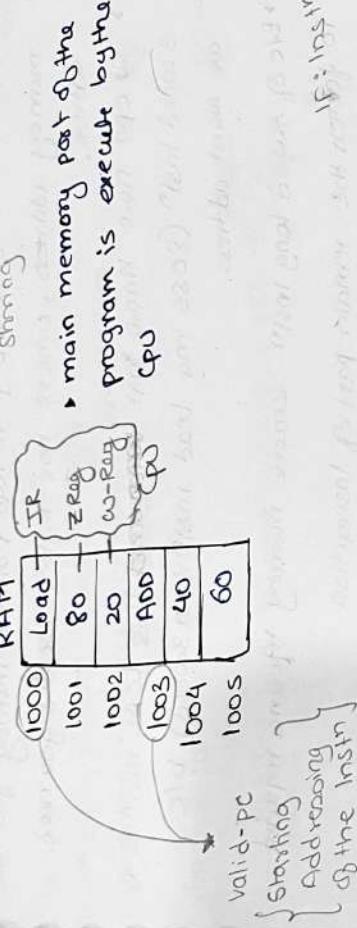


Type of CPU: ACC - CPU
{8085 MP}

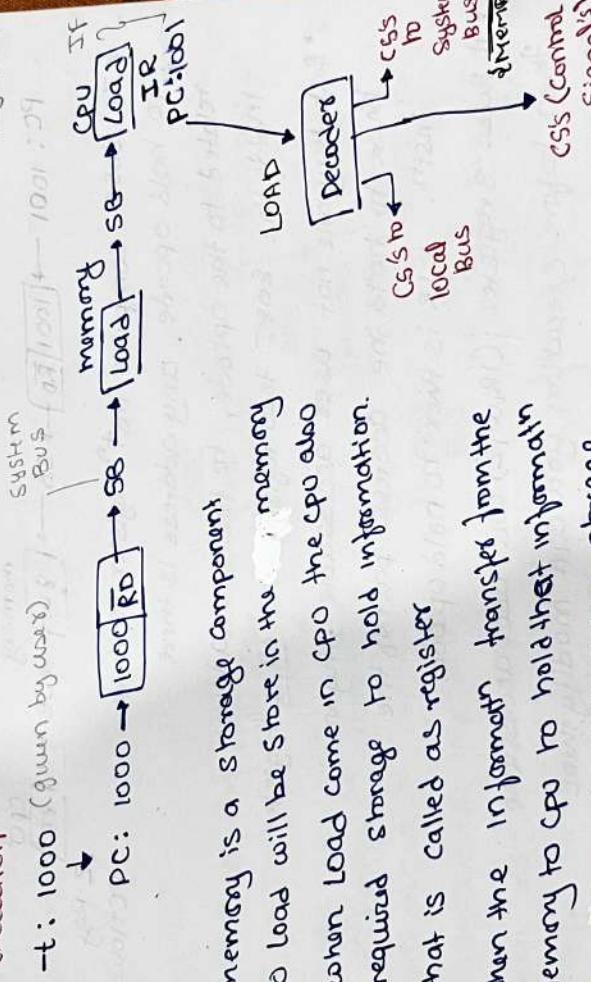
M/C code : org 1000

I_1 : load Rj ACC $\leftarrow M[R_j]$
 I_2 : ADD Rj ACC $\leftarrow ACC + M[R_j]$

Storage



Execution



- memory is a storage component
so load will be stored in the memory
& when load come in CPU the CPU also required storage to hold information.
that is called as register
- when the information transfer from the memory to CPU to hold that information
CPU compulsory have the storage component [register]. that any CPU contain set of registers.
- IR are designated to hold opcode what ever the opcode fetched from the memory that is compulsory given to memory.

2MR more
required

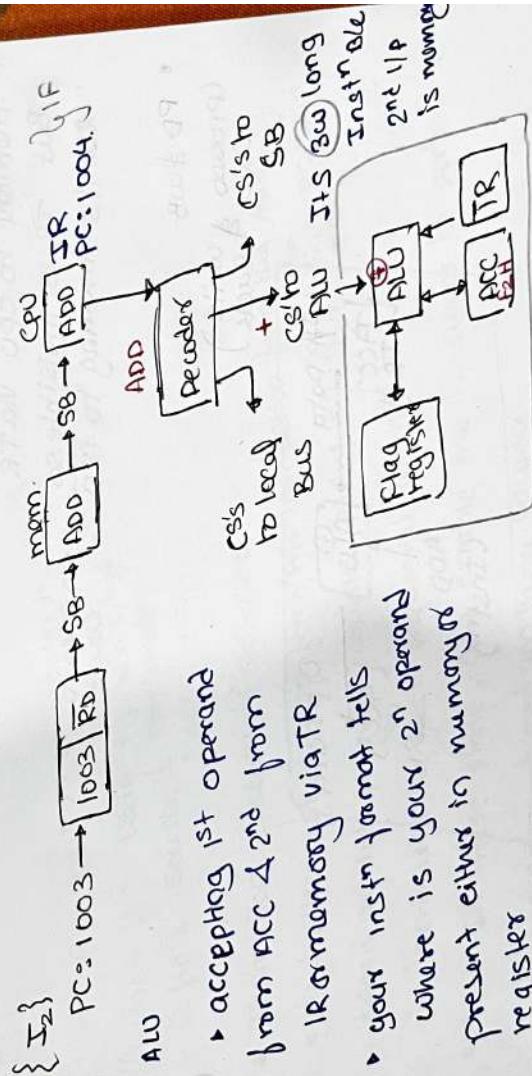
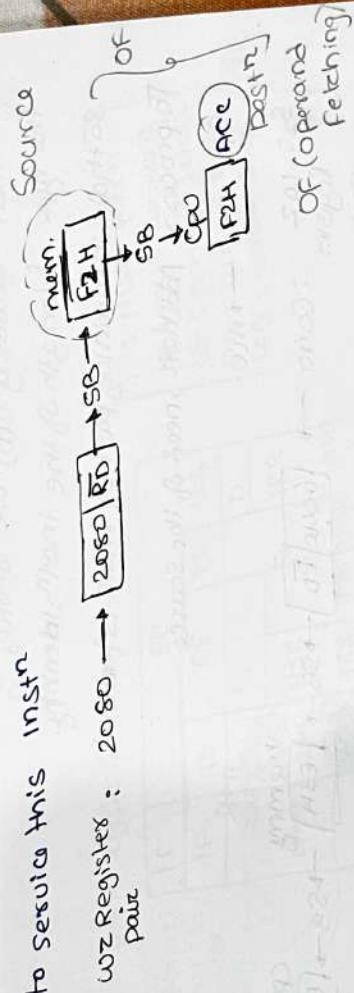
It's a 32 long
instr

- so IR size is opcode size
 - to service memroy to read from memory you need memory address, where the data is located you need address.
 - so CPU now know that this load instr is a 3 word instr (8085 ma load instr is a 3 word) b/c we want address.
 - B/C of second long instr 2 more memory reference required to fetch the remain part of instruction.
- memory → SS → SP → BP → CPU
 PC : 1001 → BP → PC : 1002
- In 8085 computer Instⁿ register is there to hold opcode. any address is there related to the opcode, to hold that address inside the 8085 two register are "w₁₂" register. But they are not user accessible, they are here to hold the address part of the instr. & IR is there to hold opcode.
 - If these 3 registers (IR,w₁₂) are user accessible then before execution you can modify them so initial instr never execute. i.e. these register are not user accessible.
 - where "w" is higher byte & "z" is lower byte

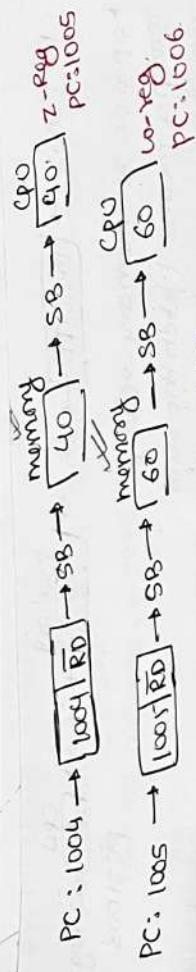
PC : 1002 → [1002 | RD] → SB → [Memory] → 200 → [200] w/o reg.
PC : 1003

- op code is memory read,
- reading means "fetch the data from the memory & address in "wz register"

To service this instr

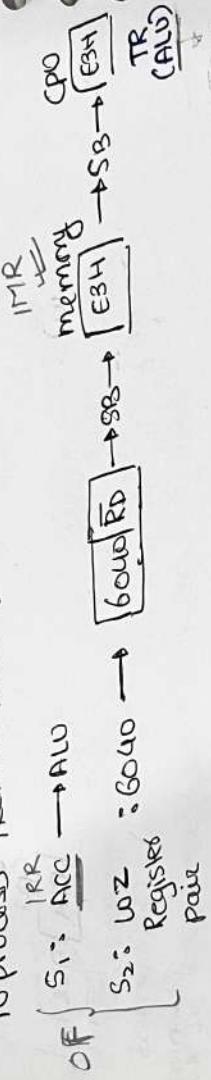


- accepting 1st operand from ACC & 2nd from memory via TR
- your instr format tells you where is your 2ⁿ operand present either in memory or registers



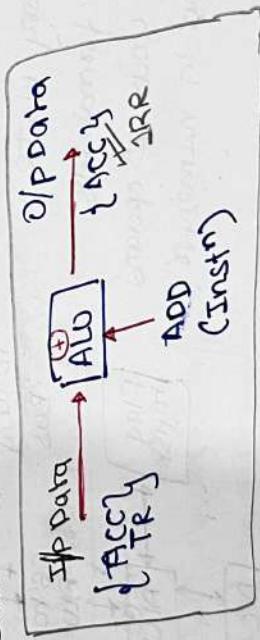
These last 3 steps is shown during decoding
 In instruction decoding (ID) we identify
 the type & length of the instr. Identify
 so that remaining part of instr is fetched

To proceed addition, need of the source



- memory to CPU via TR
 But TR is invisible so no need to count
 And its moving to ALU

- PD & WB
 (Procedure & write)
 Data & back



$$\begin{array}{l}
 \text{ID:} \\
 \text{ACC: } \begin{array}{cccccc} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 \end{array} \\
 \text{TR: } \begin{array}{c} \oplus \\ \hline 0 & 0 & 1 \end{array} \\
 \text{ACC: } \begin{array}{cccccc} 1 & 1 & 0 & 1 & 0 & 1 \end{array}
 \end{array}$$

cy: 1
on: 0

PSW : ACC & flag register

Instn cycle :

a) EC		EC		OF		PD		WB	
FC	IF	ID	OF	S ₁	S ₂	D	IRR	IRR	IRR
Wait for bus	IF	ID	OF	3MR	3MR	—	IRR	IRR	IRR
Wait for bus	IF	ID	OF	2MR	2MR	—	IRR	IRR	IRR
Wait for bus	IF	ID	OF	3MR	3MR	IRR	IRR	IRR	IRR
Wait for bus	IF	ID	OF	2MR	2MR	IRR	IRR	IRR	IRR

Fetch
Add

Citizen All Instn all parameter required?

If when it's Data transfer we only required
2 field source & destination

- Instruction execution state are
 - IF: In these state, CPU fetch the instruction from the memory based on the program counter, at the end of fetching PC will be incremented to next sequential location
 - ID: In these state, CPU enable the hardware to perform the operation based on the opcode
 - OF: In this state, CPU fetch the data based on the addressing mode
 - PD: In these state, CPU process the data based on the enabled hardware.

IF : Instn fetch
 ID : Instn decoding
 OF : Operand fetch
 PD : Process data
 WB : write back

- e.g.: In these state, result will be stored into the destination based on the Addressing mode.

PSW (Program Status Word)

PSW: ACC & flag, register pair

- ALU O/P contain the set of flags

→ flag registers contain 2 kinds of flags name as

① conditional flags (its CPU voice)

- when condition true the value is set else reset

conditional flag are predefined

with a condition . so these condition will be evaluated based on the result nature of the ALU . when condition & result nature matching then its set

Conditional flags

- these flags are predefined with a condition , this condition is evaluated based on the result nature of the ALU.
- when the condition is true flag is set otherwise reset
- ALC 8086 CPS flag register, 6 conditional flags are present
 - carry flag
 - parity flag
 - zero flag
 - sign flag
 - overflow flag
 - auxiliary carry flag

10A

① carry flag:
condition: "Is there an extra bit out of MSB?"
"Is borrow required into the MSB?"

T (1)	Set carrying
F (0)	reset (no carry)

Condition: "Is the floor carpeted?"
[Partly Yes] → $\frac{1}{2}$ $\left\{ \begin{array}{l} \text{Floor is carpeted} \\ \text{Floor is not carpeted} \end{array} \right.$

Auxiliary carry flags:

Cond: "Is there an extra bit from the lower nibble (3rd position) to higher nibble (4th position)"

$T(1) = \text{set}$ (Auxilliary carry)
 $F(0) = \text{reset}$ (no Auxilliary)

ex: 8bit data: 7 6 5 4 3 2 1 0
 Higher lower
 nibble nibble

12bit Data: 11 10 9 8 7 6 5 4 3 2 1 0

If there were required 2 binary xillary carry flags BCD arithmetics to adjust truly data.

cond: "IS TRUE ALSO OF CURRENT VALUE '0'" → TRUE-EQUAL (TRUE)

ex. ACC value ~~exists~~

10
个 个
0 0

5. sign flag
Cond: "Is the MSB bit of a MU OLP (ACC) contain '1'"

$$\begin{array}{c} \overline{T(C)} = \text{set} \\ T(C) = \text{res et} \\ \text{NL} (-\text{us logic}) \quad \text{PL (+us logic)} \end{array}$$

6. overflow flag

Cond: $ON(x,y,z) = \overline{x}\bar{y}z + x\bar{y}\bar{z}$

$$\begin{array}{c} \overline{x} \quad \bar{y} \quad z \\ \overline{x} \quad y \quad \bar{z} \\ \hline \overline{x}\bar{y}z + x\bar{y}\bar{z} \end{array}$$

- Control Flag
- These flags are used by the user, to convey the information to the CPU
 - Based on the status of these flags, some of the operation are controlled in the hardware
 - 3 control flags are present
 - ALC to 8086 CPU flag register, 3 control flags are present
 - Trap Flag :
 - At a result
 - single step program execute; trace (-t)
 - One instruction
 - || -t: will show the last execute instruction by instruction
 - || -g: will show the last execute
 - || -i: will show all instructions by instruction
 - || -l: will show all instructions by instruction

2. interrupt flag
- 0: disable the interrupt; DI
 - 1: enable the interrupt; EI
 - (by default)
 - Only interruptable interrupts (low priority)
- Interrupt is a signal which is generated by the low speed component to distributed the normal flow of execution.
- Component is used to take decision whether CPU respond to the interrupt or not.

// while execution of the program if you don't want to allow any interrupt then compulsory make sure that 1st instruction of your program is DI. Then it make interrupt flag off your program is DS.

- which make it disable [Interrupt Flag]
- Low priority interrupt are maskable interrupt may or may not service by CPU
- High priority interrupt are non-maskable interrupt can never disable

③ Direction Flag
↑ 0 : Auto increment ; CLD (clear direction)
↓ 1 : Auto decrement ; STD (set direction)

Ques] consider the following signed data & perform the addition operation

$$\begin{array}{r} 1010 \\ - 1011 \\ \hline 1001 \end{array}$$

→ carry: 1
→ zero: 0
→ sign: 0
→ overflow: 1
→ flag: 1
After the computation.

④ what is the result in decimal.

① 1010
+ 1011

0110.0011

Auxilliary carry
Flag = 1(AC)

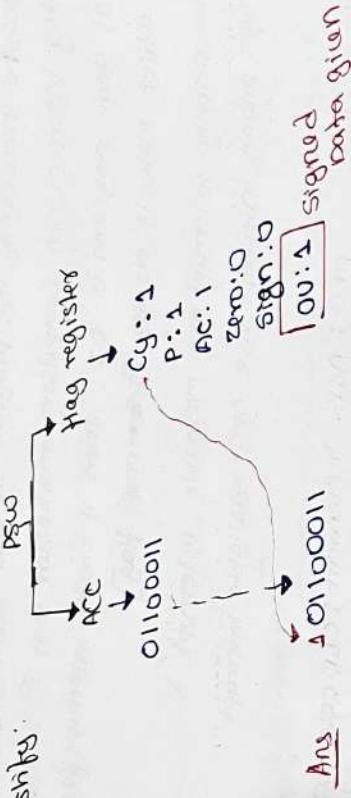
Cy: 1 (C)
Parity: 1 (PE)

overflow: 1 (OV)

zero flag: 0 (NZ)

sign : 0 (PL)

→ +ve logic溢出和进位标志
溢出和进位标志



$[R15] = 101100011$

Ans

$10110\ 0011$

$0100\ 1100$

0100011101

$\begin{array}{r} 128 \\ \hline 168 \end{array}$

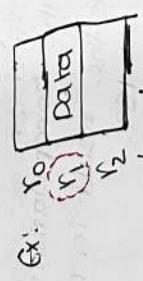
$\begin{array}{r} 4 \\ \hline 1 \end{array}$

Addressing mode ▶ O/P of the Addressing mode is effective Address (EA)



$EA=2000$

$[EA]= [2000] = \text{data}$



$EA=r_1$

$[EA]=[r_1]=\text{data}$

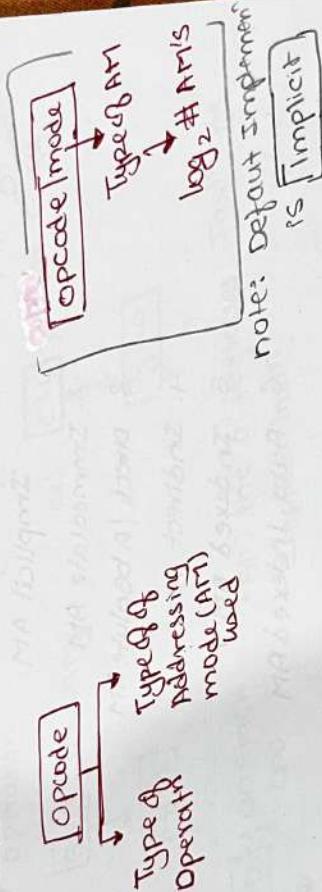
Content of

- Addressing mode shows the location of the data or next instruction in the computer
- EA is the actual address of a data or next instruction

ex: `name [ea]` ↗ content of

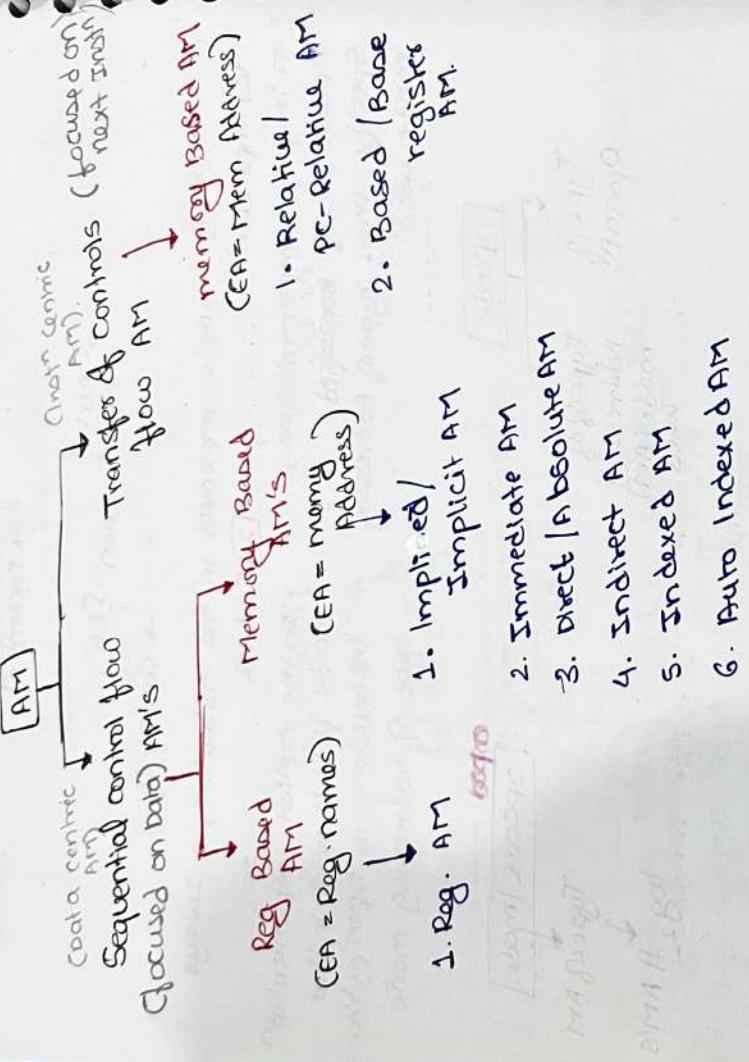
↳ `symbolname = [ea]` ↗ content of

- Addressing models implement in the instruction, in 2 ways
 - Implicit
 - Explicit
- In implicit implementation,
 - the explicit implementation mode field is used while instruction to specify the mode word
 - Specified through type of addressing mode word



- Different symbols are used in the user program to specify how varies AM
 - i.e. $\#$ | I — Immediate AM
 - R — Register AM
 - $[I]$ — Direct AM
 - (R) — Indirect AM
 - $\#$ — Indexed AM
 - $R[I]$ — Base register + indexed AM.
 - $(+/-)$ — Auto indexed AM.

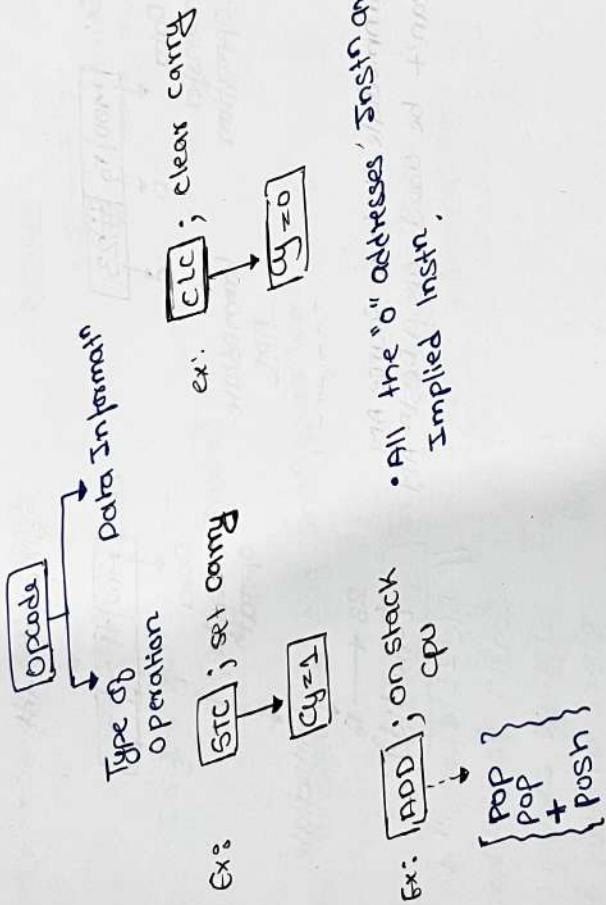
- Different AM's used in the system design is as follows



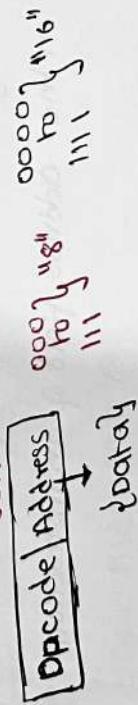
- Instruction is always present in the memory
- Data is either present in register or memory based AM.
- Data is always present memory in the memory based AM.

- Sequential control flow ARM's
- these mode concentrate on they location of a data.
- therefore Data transfer & Data manipulation both of the instruction are design within these ARM's.
- Different ARM's are employed under these category.

- ① Implied ARM
- In these mode data information is present in the opcode itself



- ② Immediate ARM
- All the "0" addresses' trash one
 - these mode is used to access they constant. Int here mode data is present in the address field of the instruction 3bit



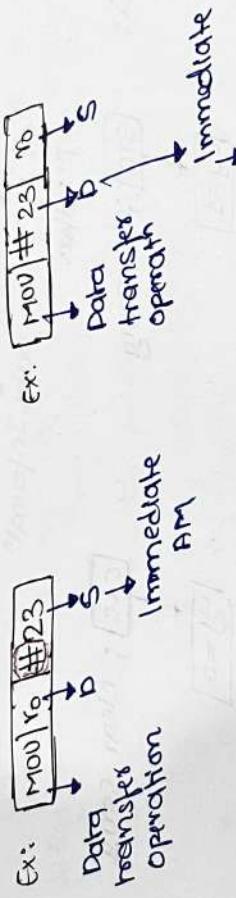
- Limitation is a that range of constant are delimited by the size of a Address Field.

i.e n-bit address field support 2^0 to 2^{n-1}

unsigned constant

$2^{0-n} \text{ to } (2^{n-1}-1)$

Signed constant.



∴ Immediate AM is a source AM
It can't be used as a Destin AM

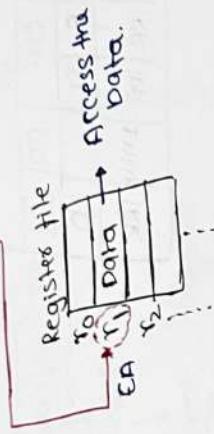
23 → r0
↓
Data transfer operation
immediate
AM

23 → r0
↓
Data transfer operation
immediate
AM

③ Registers AM

- These Mode is used to access the local variable.
- In these mode Data is present in the register . so effective address is register name. The effective address will be maintained in the address field of an instruction

Inst: [opcode | Address]



• EA = Address field name

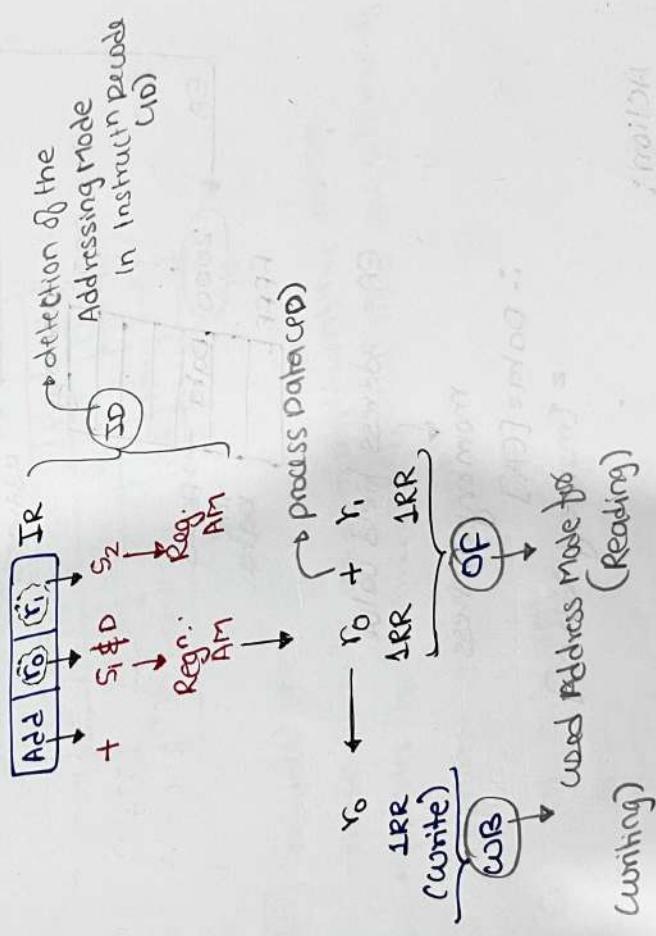
→ Registers name

• Data = [EA]

= [Registers name]

Actions:
AR → To
read/write
the Data

Ex: Consider two long instructions below!



Instruction cycle

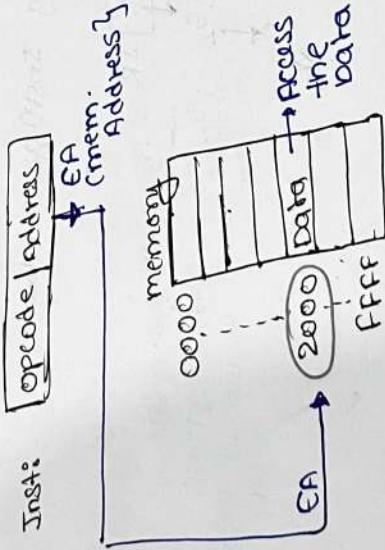
// IRR is negotiable

but IMR not
DIR Registers is a
faster component
it doesn't required
cycle, whereas
memory is slow
component it requires
machine cycle
so IMR countable

4 Direct Am. (Absolute AM)

- these mode is used to access they static variable.
- In these mode data is present in the memory, so Effective address is a memory address. these Effective address will be maintained in the address field of an instruction.

FC	EE					
IF	ID	OF	PD	CW		
	400	S1	S2	D		
IMR	3MR	IRR	IRR	1AWU	IRR	



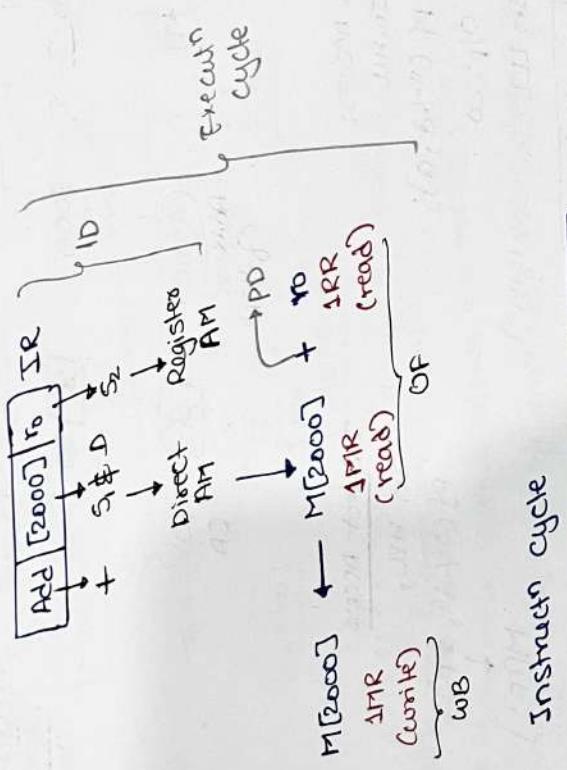
EA = Address field value

↓
memory address

$$\therefore \text{Data} = [\text{EA}] \\ = [\text{Memory Address}]$$

Action:
IMR → to read / write
The Data.

Eg: consider 600 long 2-address hypothetical system with
direct & registers AM. given below.



Instruction cycle

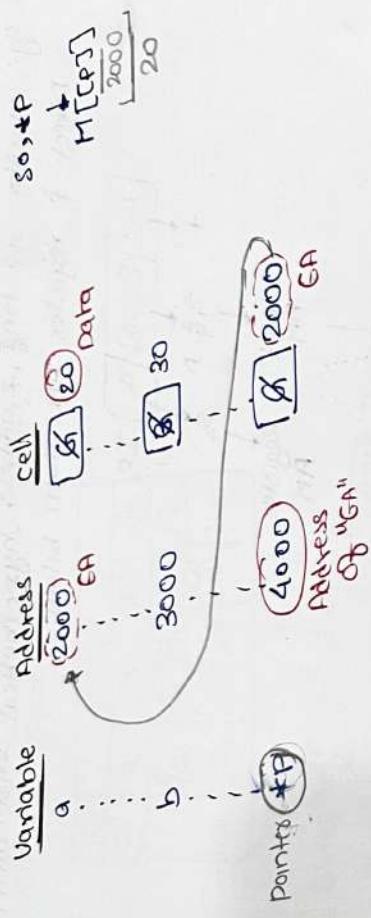
FC	EC				
IF	ID	OF	PD	WB	
		S ₁	S ₂	O	
		M ₁	M ₂	I ₁ R	I ₂ R
		I ₁ R	I ₂ R	I ₁ W	I ₂ W

5] Indirect AM

- these mode is used to implement the pointers.
- pointer is a variable which hold the address of another variable in a program

Ex: int a, b, *p;
a=20j
b=30j
p("1.d", a); // direct accessing
p="2d";
p("1.d", *p); // indirect accessing

pfc("1.d", *p); // indirect accessing



Direct Access:

in HL,

pt ("0..d", a);

012:20

in LR (word long)

Mou[r0]@2000

EA is in Address field
 $r_0 \rightarrow M[2000]$

{r0: 20}

AMR required

Indirect Access

in HL,

pt ("0..d", *p);

M[r0]

OP: 20

Mou[r0]@4000

address
of r0

→

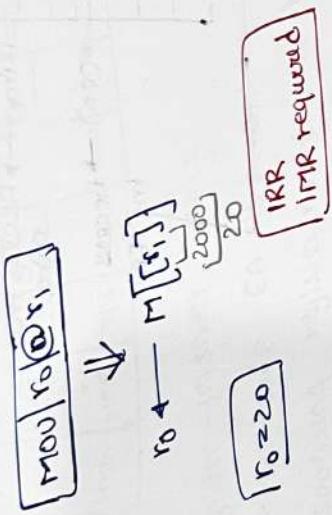
M[r000]

r0 = 20

2MR required

Enhanced indirect [register-indirect]
[store the "EA" in the register]

MOV R1, #20000; R1 = 2000



- ① Memory Indirect EA's (default)
- In these mode Effective address is in memory, the correspond memory address will be maintained in the Address field of the instruction as address of a "EA"

Effective Address (EA) = [Address field value]
→ memory address

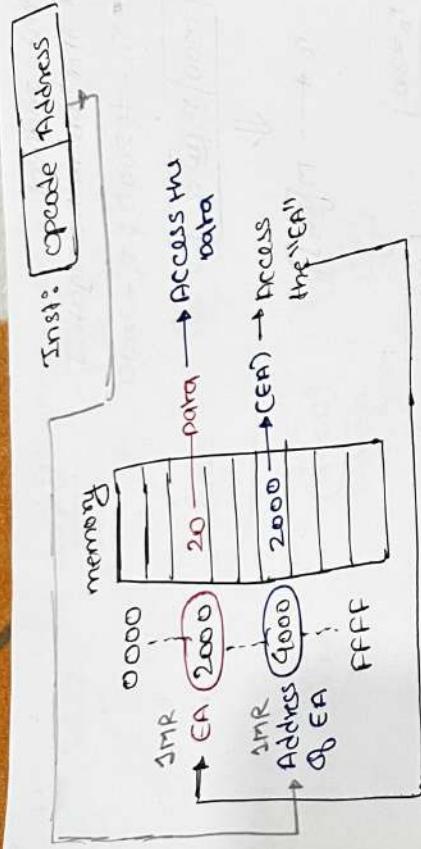
$$\begin{aligned} \text{Data} &= [\text{EA}] \\ &= [\text{[memory address]}] \end{aligned}$$

EA
Data

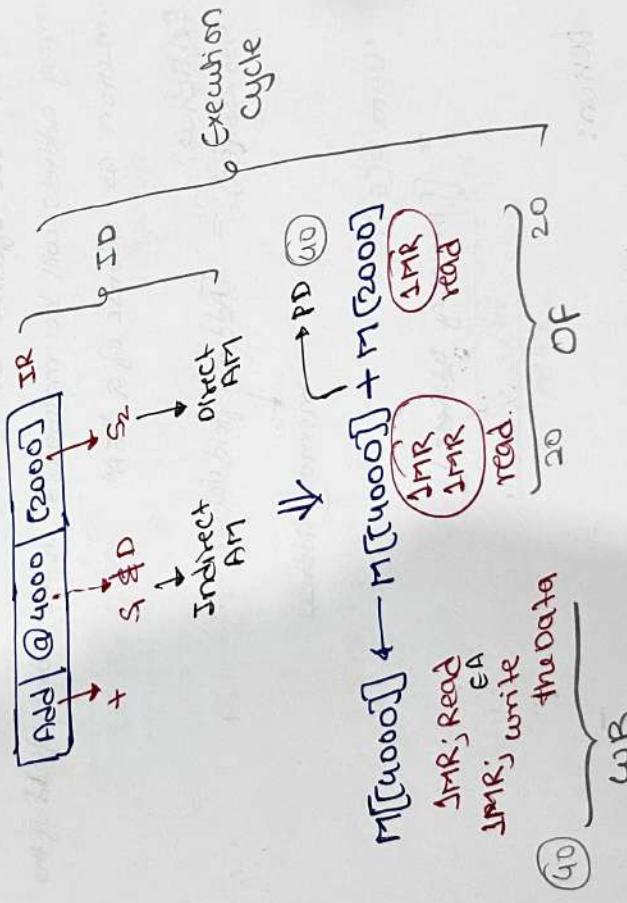
Action:

IMR → To get the "EA"

IMR → To access the "data"



Q) Consider the following ~~two~~ long 2-address Instn with Indirect & Direct AM's



Instn cycle:



(ii) Register - indirect AM's.

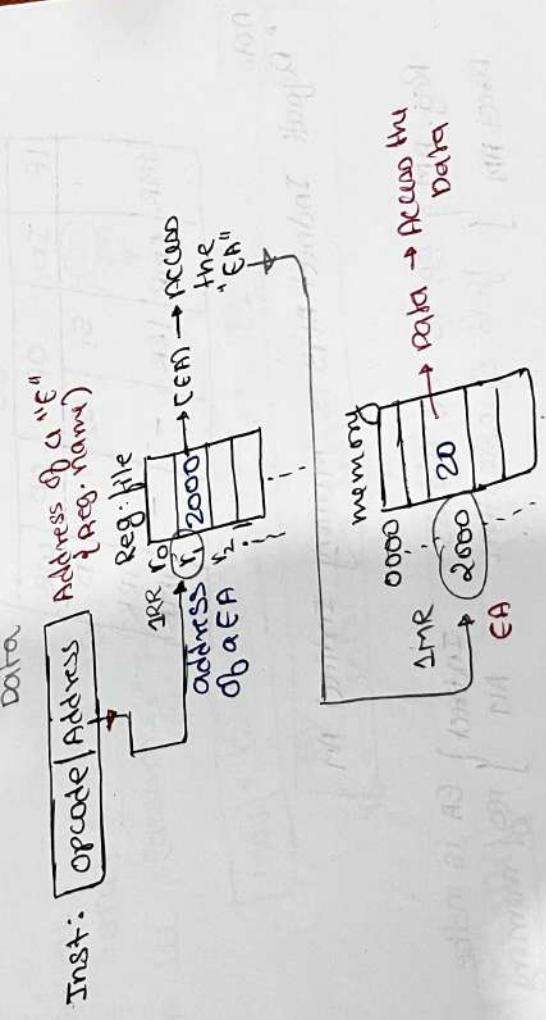
- In these mode EA is present in the registers, the corresponding register name will be maintained in the address field of the instruction as a address of a EA field

$$\therefore EA = [\text{Address field value}]$$

Registers Name.

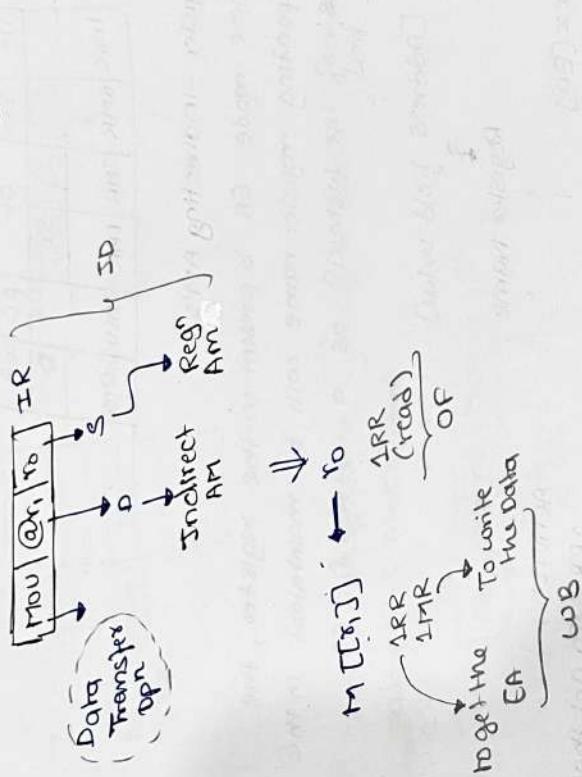
$$\begin{aligned} \text{Data} &= [EA] \\ &= [[\text{Registers name}]] \end{aligned}$$

Action:
IRR: To get the EA
IMR: To get the Data



Ex: Consider 6w long two address instr with one register
Indirect & reg. AM's

@ r : registers indirect
@ 2000: memory indirect



Instn cycle:

FC	EC	OP	PD	WB
IF	SD	S1	S2	D
	6w	IMR	SMR	IRR

Note
Default Indirect AM's is memory indirect AM.

Reg. AM } CA is the Address
Indirect AM } field of instruction
Direct AM }

CA is in the
AM } reg memory

Note

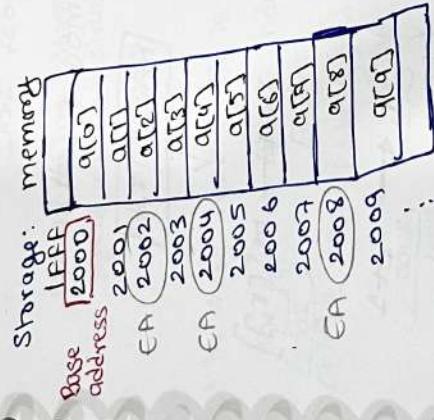
- In the following AM, EA is computed by adding the constant to content of some registers.

- ① Indexed AM
- ② Auto indexed AM
- ③ PC-relative AM
- ④ Base register AM

Analysis: why EA is calculated by adding const?

Consider char array

i.e. `char a[10]`



Accessing of a data.

- ① Random element Accessing
 - [Base Address + Index value req]
 - varies
 - stored in the index Reg (CR)
- ② Fixed Constant

Index values : [0 to 9]

In HLL,
 $a[0]$
 $a[1]$
 \dots
 $a[9]$

In LIL, assume, "r1" is R1

`MOV r0 [2000+r1]`

Constant &
 Index register
 name.

$r0 \rightarrow M[2000+r1]$

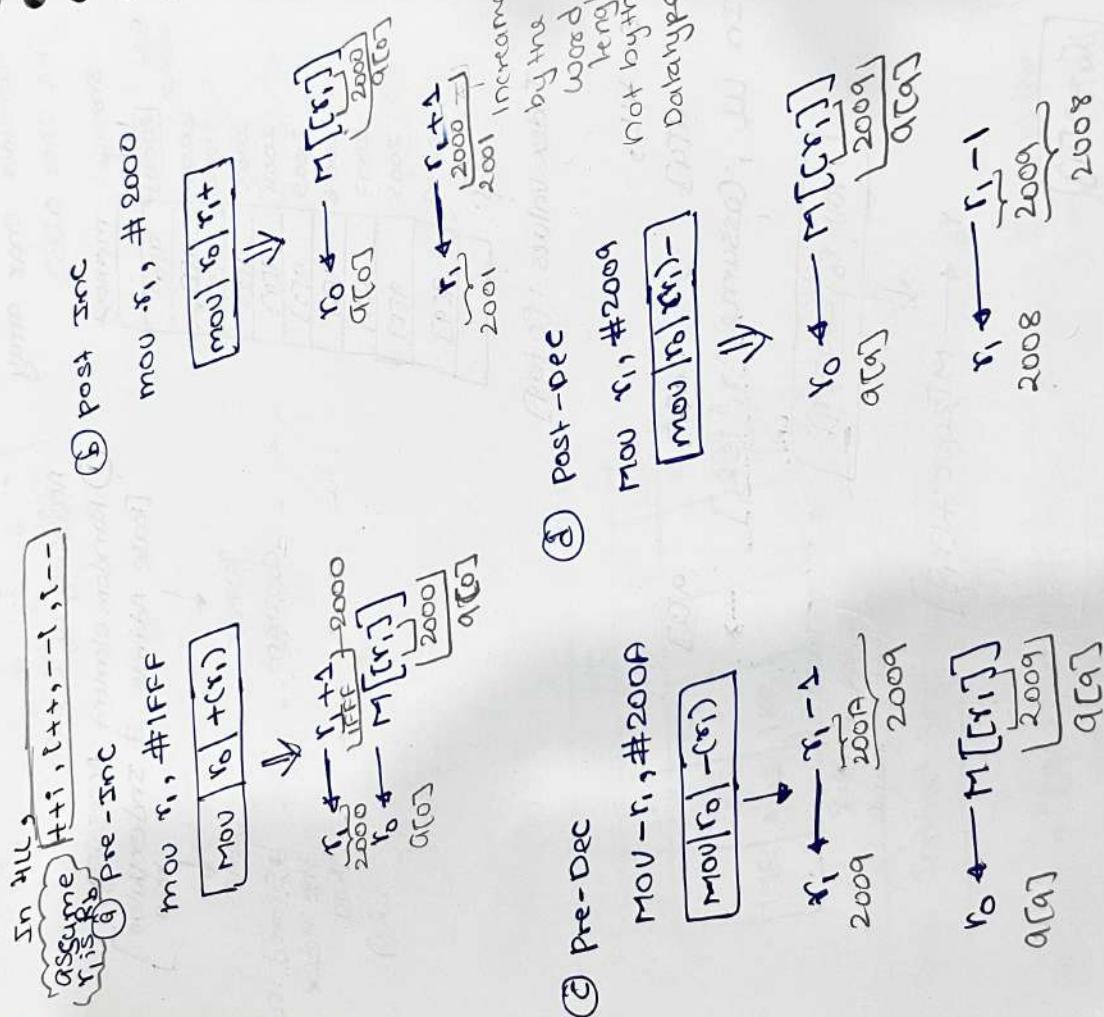
$a[4] \rightarrow M[2004]$

$r0 = a[4]$

$a[4] \& Data$

② Linear processing using Base Address is required
(Auto indexed A/R)

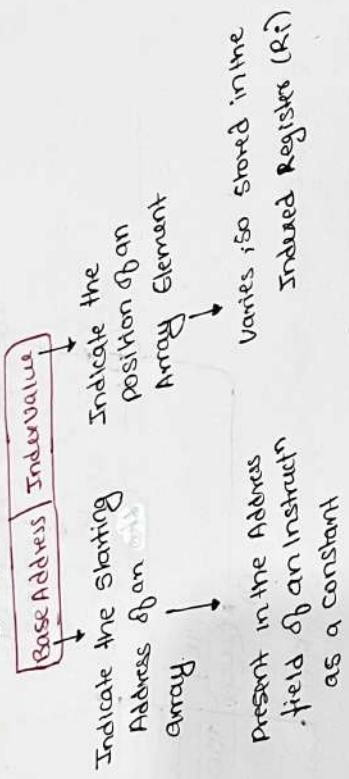
→ Continuously updated
use the Base register (R0) holds the Base Address.



Indexed FM or Base Indexed FM

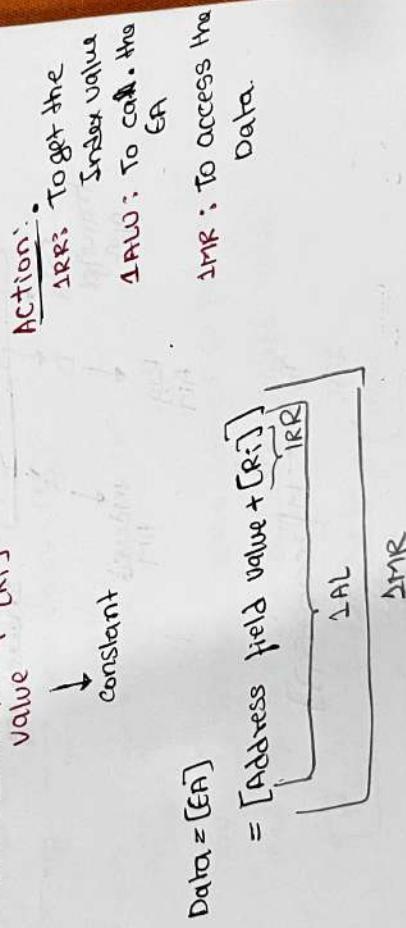
- these mode is used to access the random element in array from the memory

In these style of accessing, 2 parameters are required

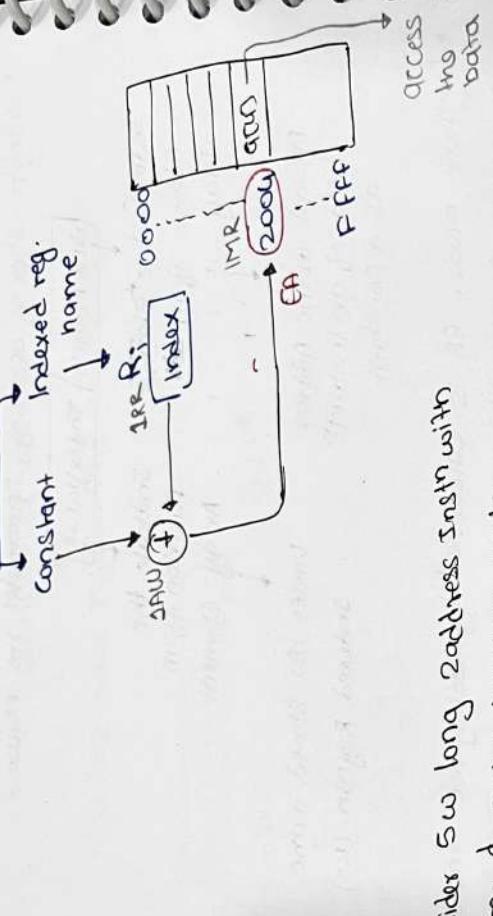


- In these mode, EA is calculated by adding the constant value to content of the index register as a constant

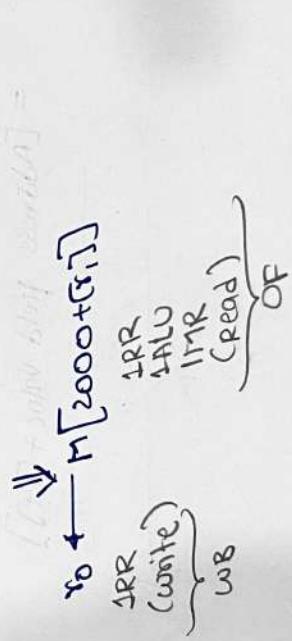
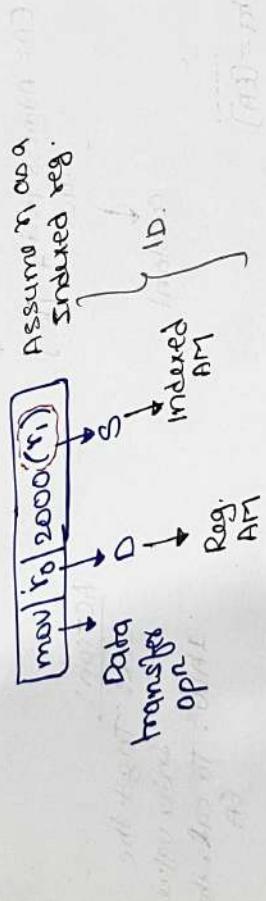
$$\text{i.e. } \text{EA} = \text{Address field} + [\text{Ri}]$$



Instruction: [opcode] [address]



Q) Consider 5 word long address instrn with reg of indexed Address mode.



Instruction cycle:



To calculate EA

Note

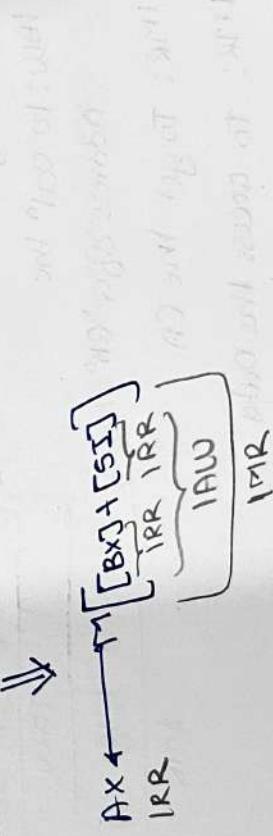
- when the system support both Base & Index registers. Then we can store Base address & indexed value into a respective registers, later refers the index registers & base register name in the address field of the instruction.

Ex: In 8086 MP
 $bx \cdot reg. \rightarrow \text{base register } (R_1)$

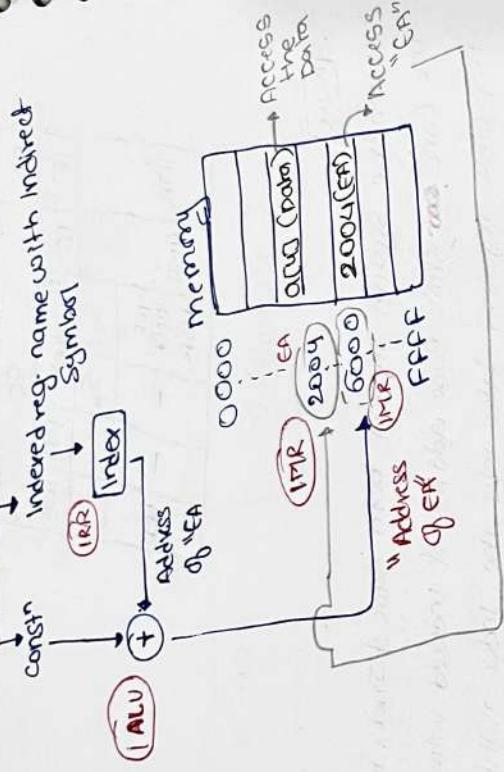
(source SI IDI
 \downarrow
 index IX IDI
 \downarrow
 (destn IX))

MOV BX, #2000 ; Base address shifted to \boxed{BX}
 MOV SI, #4 ; Index value shifted to \boxed{SI}

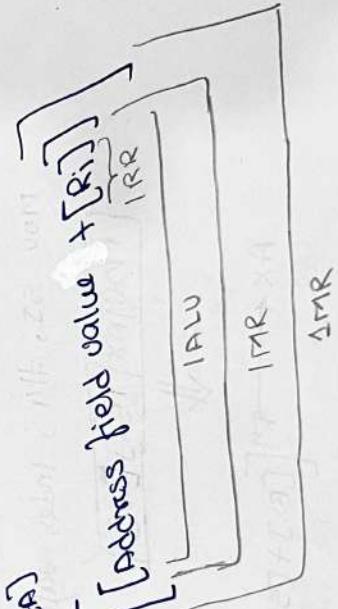
$\boxed{\text{MOV AX} [BX][SI]}$



Instruction: [Opcode] Address



- Indexed indirect ARM or indirect indexed ARM
- In these mode EA is in the memory
- In these mode EA is calculated by adding two
- Inherent mode address of the EA is calculated by adding registers
- Constant value to the content of a indexed register
- i.e. $EA = [Address\ field\ value + [R_i]]$



Action:

IRR: To get the index

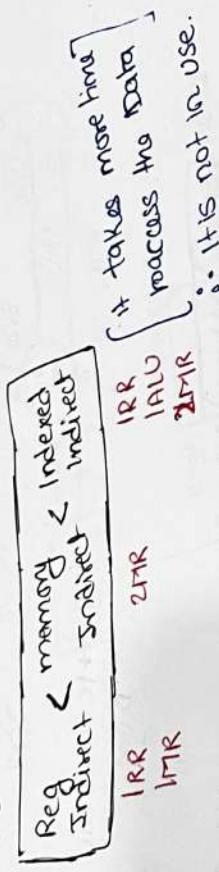
IAR: To call the address of "EA"

IRR: To get the EA

IRR: To access the data

Note

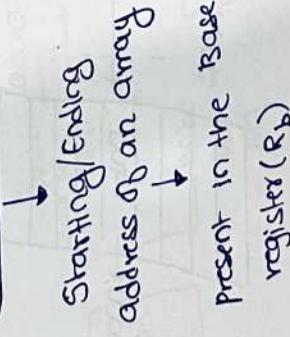
Accessing or due to the data Accessing using Indirect AM is,



Auto - Indexed AM

- These mode is used to access the linear array element from the memory
- In these style accessing only one parameter is required

Base Address

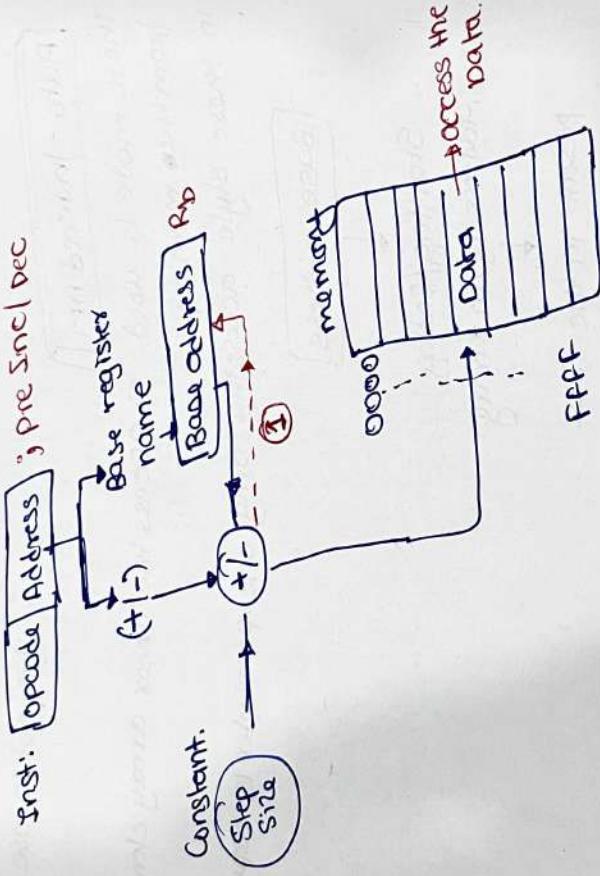


- In these mode, EA is calculated either by incrementing or by decrementing the Base register address based on the Step size.
- Step size is a fixed constant depend on the word length of a CPU.

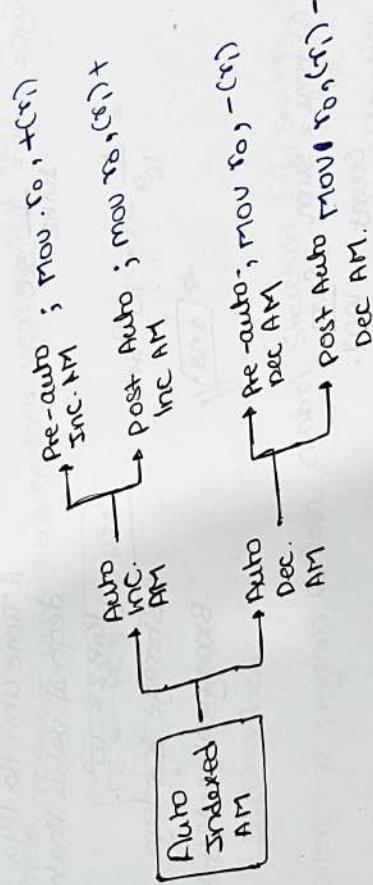
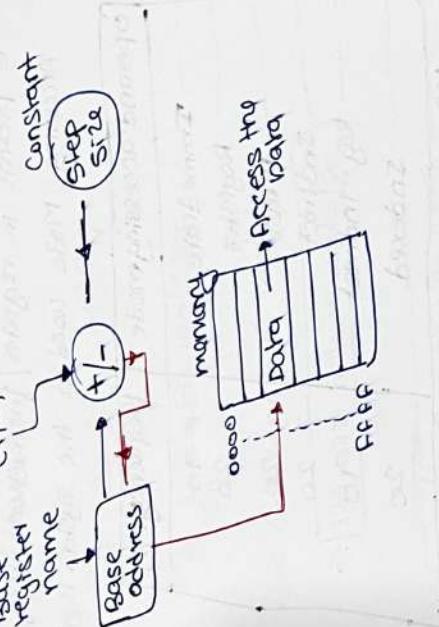
- $EA = [R_B] (+1) \text{ Step size}$
- $Data = EA$

$$= \left[\begin{matrix} R_B \\ IRR \\ IALU \end{matrix} \right] (+1) \text{ Step size}$$

- Action:
- "IRR": To get the Base address
 - "IALU": To calculate the "EA"
 - "ALU" → To access the "Data"

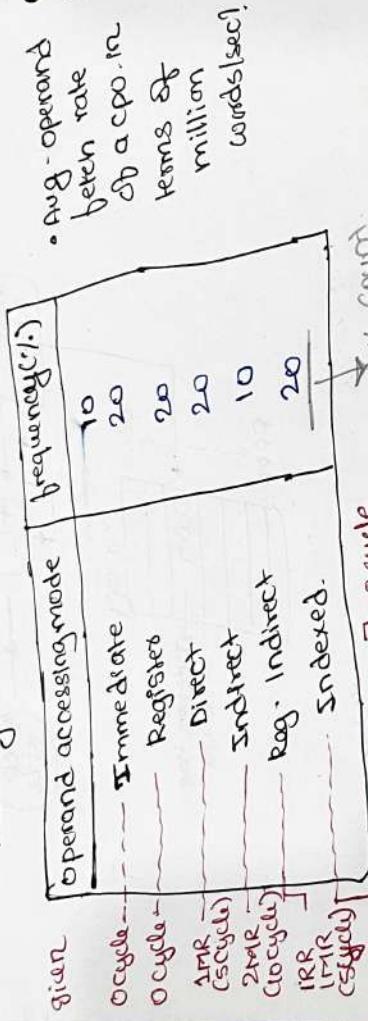


Inst: [opcode] address ; post index



where "r" is a Base Register.

- * [Ques] consider 1GHz. CLK frequency CPU, which consume 5 cycle for all computation & "0" cycle when the data is present in registers/instruction itself. Different operand is given below.
- Accessing mode used in the system is given below.



• Avg. - operand fetch rate of a CPU in terms of million words/sec!

100% count

100% time unit ka liya decimal value liya

$\text{Avg. time} = \frac{1}{10} \text{ sec} = 10^{-9} \text{ sec}$

Sol.: $\text{Cycle time} = \frac{1}{10 \text{ GHz}}$

$\Rightarrow \boxed{10^{-9} \text{ sec}}$

Storage ka liya

Boring value $10^{12} = 2^{30}$

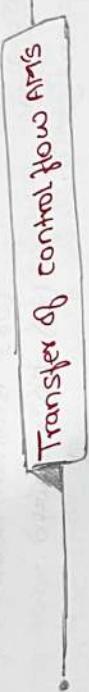
$$\text{Avg. time} = \frac{\text{sum}}{\text{count}} = \frac{\text{sum}}{100!}$$

$$\text{Total required} \rightarrow \left[(0.1 * 0) + (0.2 * 0) + (0.2 * 5) + (0.2 * 8) \right] * 100! \\ + (0.2 * 10) + (0.1 * 5) + (0.2 * 8)$$

$\Rightarrow 5.1 \text{ ns}$

1 opend → 5.1ns
Circuit (Circuit) X 1sec
Process → 1sec

$$\begin{aligned} \text{1 word} &\rightarrow \frac{1}{5.1 \times 10^9} \text{ words/sec} \\ &\rightarrow 5.1 \rightarrow 10^9 \text{ words/sec} \\ &\rightarrow 0.196078 \times 10^9 \text{ words/sec} \\ &\Rightarrow 196.078 \text{ millions words/sec} \end{aligned}$$



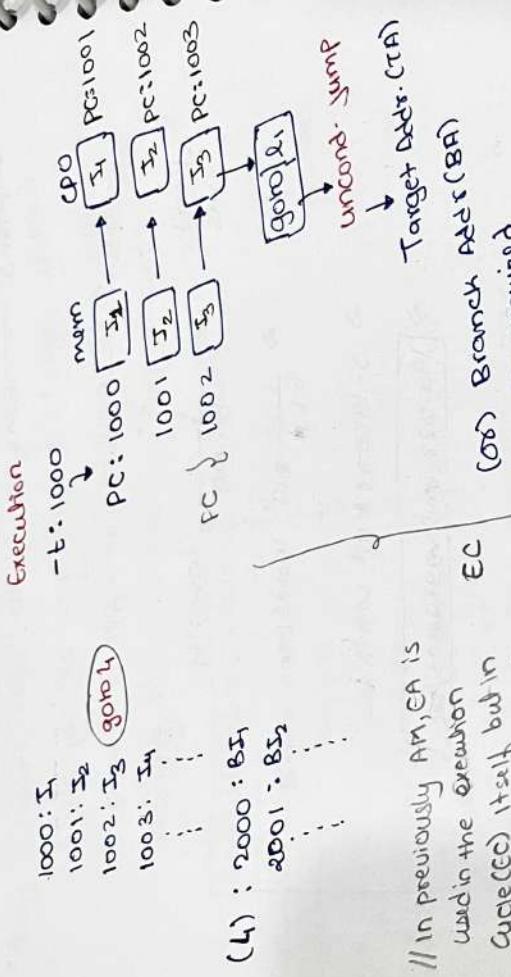
These mode are concentrate on they location of a next instruction

$$\boxed{\text{PC} \rightarrow \text{PC} + \text{step size}}$$

Point to sequential

Instr address

- when the program contain control structure (if, switch, goto, loop & function)
- then during the program execution control will be transfer from one place to another place. described below



// In previously AM, EA is used in the execution cycle(EE) itself but in these ER is calculated but used in fetch cycle

- // in previously run we need delta
- // but in these mode we need trackback

So Data execution cycle ka address change but main are fetched from the fetch

we are just calculating
treeea not using "rec"
we will use it in next fetch

```

graph TD
    PC2002[PC-2002] --> PC2001[PC-2001]
    PC2001 --> PC2000[PC-2000]
    PC2000 --> PC2001

```

The flowchart illustrates the relationships between three systems: PC-2002, PC-2001, and PC-2000. PC-2002 is connected to PC-2001. PC-2001 is connected to both PC-2000 and PC-2002.

Q1 P 2nd sum: $I_1 - I_2 - I_3 - I_4 - \dots$ go to

Data address

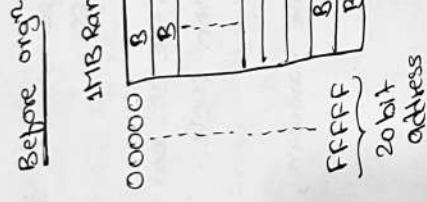
Data Centric AMs : EA is identified in the EC & also used in the EC next instruction address.

Instn centric : EA is identified in the EC but used AMs in the next fetch cycle.

- to implement the control structure in the base hardware, transfer of control(TOC) instruction are used.
i.e Branch instr | jmp instr | skip instr
- In HLL, where you have a control structure at the time of compilation Compiler substitute the TOC opcode in place of control structure
- At the time of designing the processor the designer decide which TOC instr to be used out of 3 instr.
- Transfer of control implemented/design with the following AM, used to calculate the TAC (target address) | BAC(Branch address) | EA(effectice Address)

1. PC-relative AM
2. Base register AM.

to analysis the above AM, let us consider 8086 memory organisation as reference model
i.e 8086 Support 1MB RAM, Organized into 16 logical segments with a segment size of 64KB

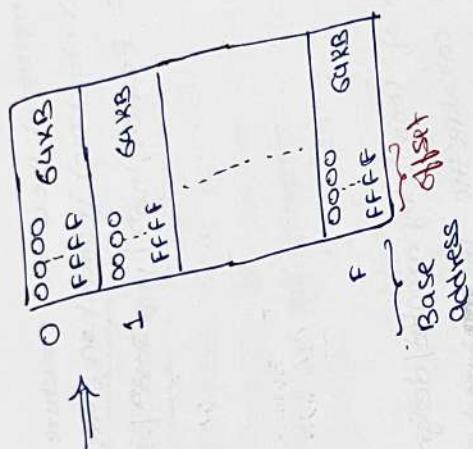


After org

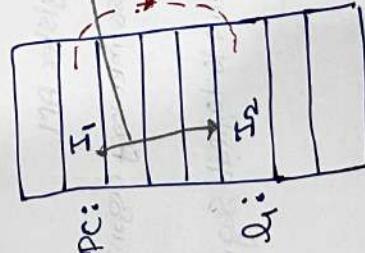
$$\Delta M_8 = 16 \times 64KB$$

$$= 2^{24} - 2^0$$

$$= 2^{20}B$$



Intra segment
TOC



Other
segment

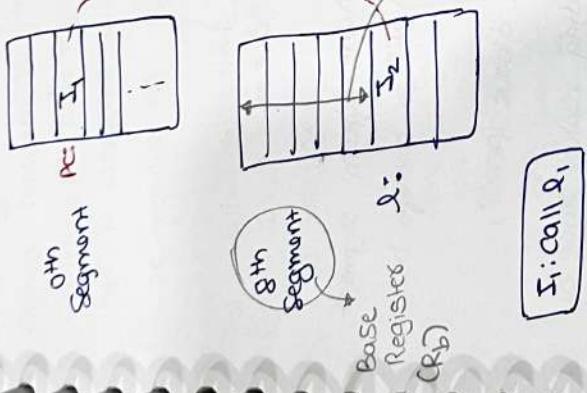
Displacement
(The step count
reqd to reach \$2)
Segment
TOC
Control
will be
transferred

I: jmp &

TA/BALERA: PC + Relative
value

Intra segment TOC

|| when memory doesn't support then only intra segment & ib support then both intra and supported



$$TA/BP/EA = [R_0] + \text{relative value.}$$

PC-relative AM

when the target instruction is present in the same segment than during the program execution, control will be transfer within that segment called as intra segment TOC.

- these AM is used to implement the above operation
- in there mode, EA is calculated by adding the relative value to the "PC"
- relative value means distance b/w the current location to target location
- it is a signed constant, present in the address field of instruction

call -> the jump is made by changing the PC

$$EA = PC + \text{Address field value}$$

↓

current relative
location value

↓

next instr address

↓

$PC \leftarrow PC + \text{relative value}$

Based Base registers ARM

- when the target instruction is present in the different segment than during the program execution, control will be transfer b/w the segments called as inter segment TOC
- these mode is used to implement the above operation,
- In these mode EA is calculated by adding relative value of the content of the base registers

$$EA = [R_0] + \text{Address field value}$$

↓

next instr Address

↓

Target segment base address

$$PC \leftarrow [R_0] + \text{relative value}$$

Note

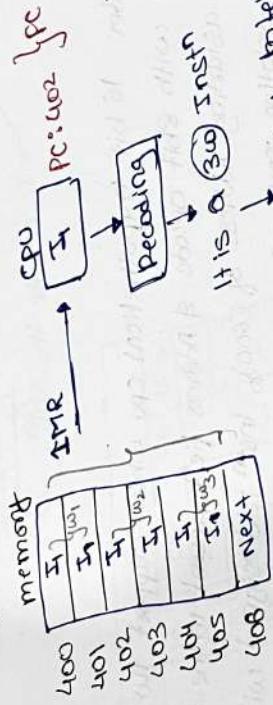
- PC relative & Base register both of the addressing mode are suitable for program re-location at run-time [and one used in compiler design]
- Base registers addressing mode is best suit for position independent codes.

Q) Consider the 6B long Imm present in memory with a starting address of 4000 word length of a CPU is 2B. A-63 signed constant is present in the address field of an instruction with a base register is 800. calculate the EA when the instruction is designed with a

- PC-relative Address
- base register AM

- (a) word size = 2B
- (b) Imm = 6B (3B long)

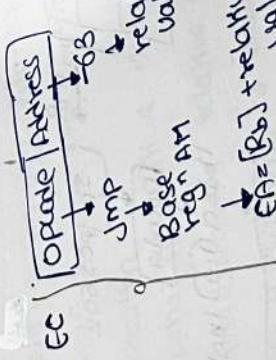
storage



It is a 3B instruction
2B R required to refer

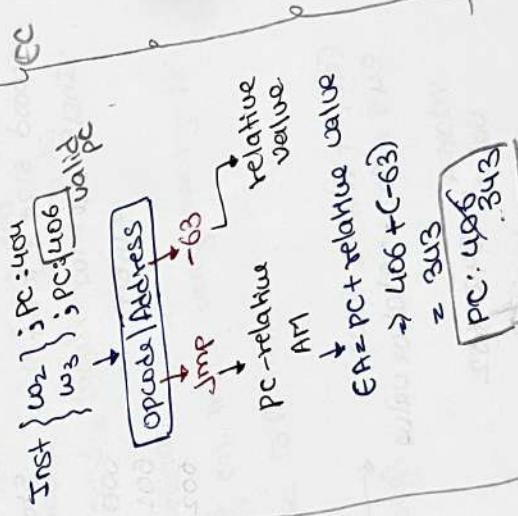
to complete SNSQ

same as previous question



$$\begin{aligned} EA &= [R0] + \text{relative value} \\ &= 800 + (-63) \\ &= 737 \end{aligned}$$

PC: 4006
737



PC: 4063
3933



- To get the current "PC" location from mem at R₀
- To call the "CF"
- To update the "PC".

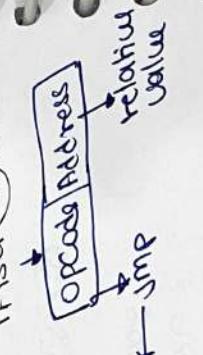
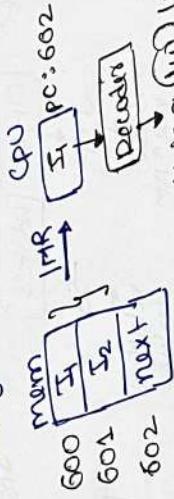
? Consider 16 bit hypothetical CPU which supports twobyting

Instruction with eight opcodes & Address field present in the memory with starting Address of C600h. Instruction is designed with a PC-relative JMP operation to transfer the control to sub location

During it Execution what is the relative value in Hexa decimal usage?

word size = 16 bit

· Insdr 548 = 1001000000000000₂
≈ 16 bit



→ PC-relative → EM

$$FA = PC + \text{relative value}$$

$$548 = 602 + \text{relative value}$$

$$\text{relative} = 548 - 602$$

$$\Rightarrow -54$$

* Relative Value size = Address field = 8bit

- 8bit code

$54 \rightarrow 00110110$

↓
2's complement

\downarrow

11001001

- 8bit:
 $\frac{11001001}{(C\ A)^{16}}$

Ans: $(CA)^{16}$

FC	EC					
IF	ID	OF	PD	WB	D	O
11	01	S1 S2				
IR	IR	IR	-	-	-	-

IR
IRU
IRR

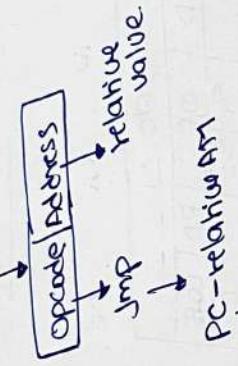
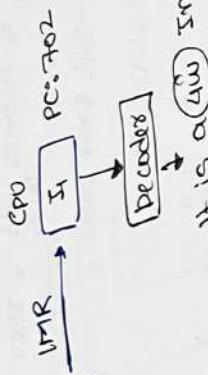
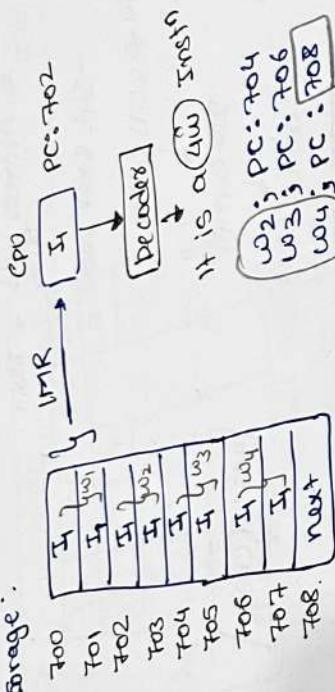
Q1) Consider 8 bit long PC, relative jump sign is stored in the mem' with a starting address of $(7000)^{16}$ in a 16bit CPU. This instruction transfers the control to $(6220)^{16}$ location during its execution.

- What is relative value in decimal?
- Use the above relative value & calc. the EA using base reg. Ans with a content of the base register is $(2000)^{16}$.

Sol: word size = 16bit (2B)

Instruction size = 8B (4w long)

Storage:



$$\begin{aligned} EA &= PC + \text{relative value} \\ 620 &= 708 + \text{relative value.} \end{aligned}$$

$$\begin{aligned} \text{relative value} &= (620 - 708) \\ &= -88 \end{aligned}$$

Now after decoding operation, we get
EA = [R_b] + relative value
= 200 + (-88)

Base register AM.

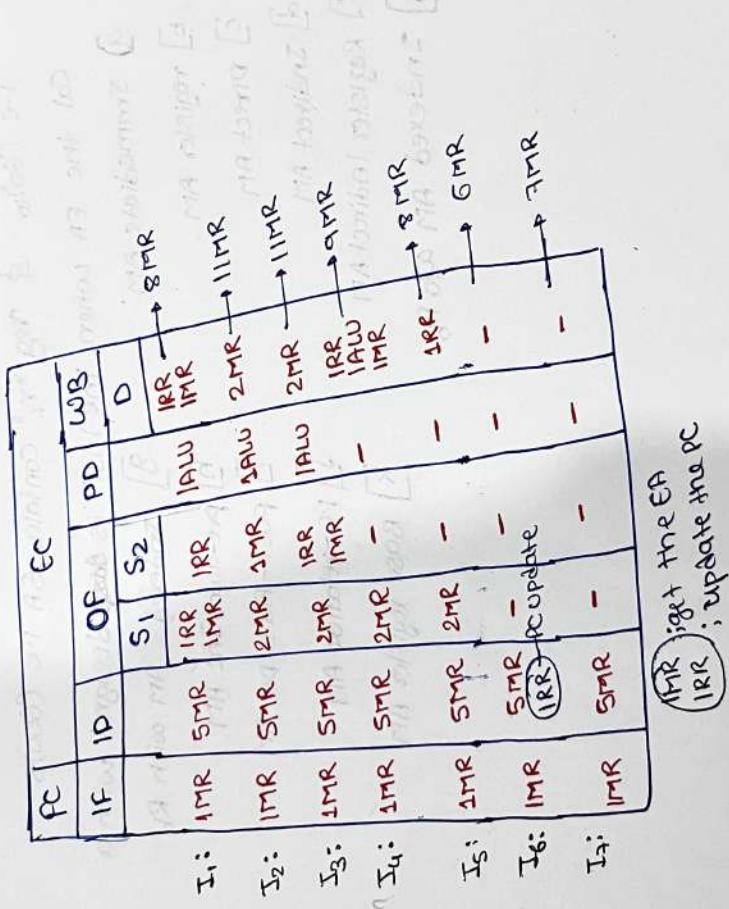
$$\begin{aligned} EA &= [R_b] + \text{relative value} \\ &\approx 200 + (-88) \\ &\approx 112. \end{aligned}$$

Now after getting EA, we can find the address of memory location.
EA = 112
112 is the address of memory location.

From this we can find the value.

Q) consider the following 6w long 2-dress instr uses different AM calculate the no. of memory reference (memory cycles) required to complete the instr processing

1. ADD @ R0, R1
2. MUL @ 2000[PC+0000]
3. MOV @4000@10
4. MOV R0@4000
5. MOV R0@6000
6. JMP 2000; Direct AM
7. JMP @ 2000; Indirect AM



- Q) Consider 16bit hypothetical CPU which support 4096 long
 instr with 8bit opcode & Address field placed in the memory
 with a starting Address of (500)₁₀. Address field contain
 $(200)₁₀. memory content of 100 is (000)₁₀. Reg file size in the
 System is 8 (R₀ to R₇). R₆ & R₇ registers are designated as index
 & base registers respectively, contain (40)₁₀ & (240)₁₀ values
 respectively. Reg "R₀" contain the data i.e (60)₁₀ & reg "R₂"
 contain EA i.e (60)₁₀ & reg "R₂" contain the data.
 i.e (60)₁₀ & reg "R₂" contain EA i.e (600)₁₀$
- Cal the EA where the instr is designed with a
- a) Immediate AM indexed AM with R₆
 - b) register AM pre-auto Inc AM
 - c) Direct AM post - Auto Dec AM
 - d) Indirect AM PC-relative AM
 - e) Register Indirect AM base registers AM
 - f) Indexed AM w/o R₆

- Type of instruction / Type of operation / Instruction set.
 - In the CPO design 3 category of instr are present
 - i.e. Data transfer instr (mov, load, store, push, pop, in, out, ..etc)

↳ Shift & Rotate Sums

i

- while execution of increment function, constant memory content. when it satisfies the max. limit then count will be rolled back to min. limit

1-2 INSTN INC TO



10

ex:

INC	RD
-----	----

; $\text{GO}:(\text{FF})\text{H}$

its status during the execution of decrement function const . it is satisfied when it reaches zero . when it reaches zero , the value will be rolled back to max limit . from the registers or memory will be rolled back to max limit than the count min . limit

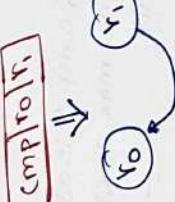
doc 360

10

ex: `dec | r0; r0 : (000)H`

`r0: 0000 0000`

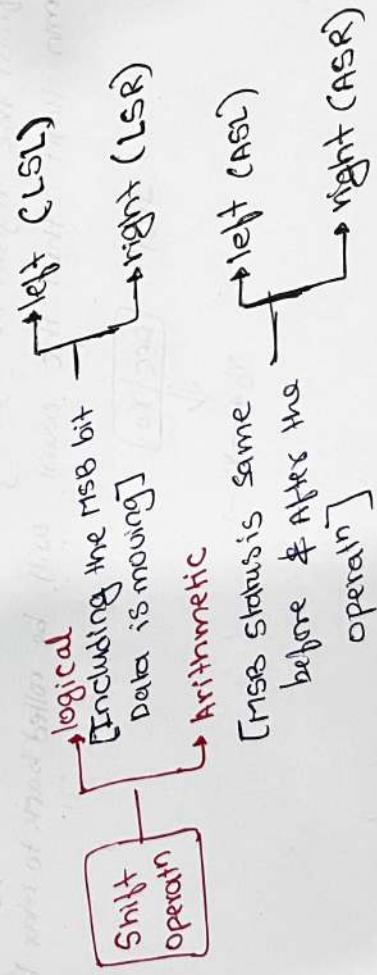
`r0: 1111 1111 CF: 1`

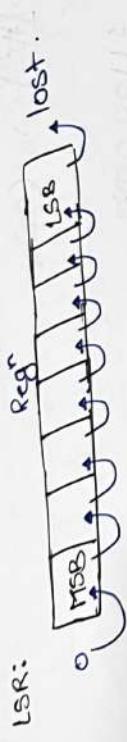
- cmp function
 - will execution of compare (cmp) instn 2nd argument compared with the 1st argument
- 
- "r1" Data is compared with "r0" Data

- these instruction execution status is present in the zero & carry flag

shift operation

- while execution these operation data is moving to left or right direction in a bit wise sequence with loss



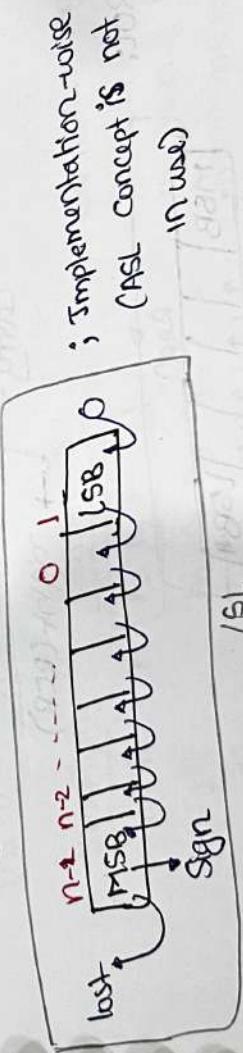
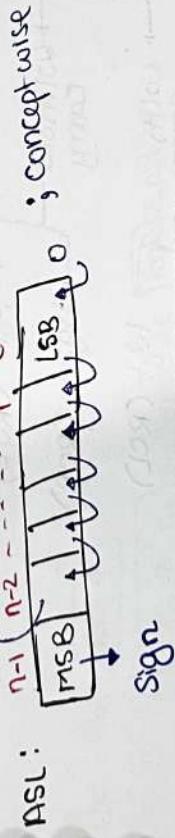


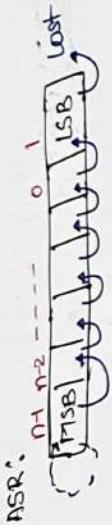
Ex: LSL r0 #3 ; r0 : (F2)H
not given default "1"

Implementation note

r0: 1111 0010 0000 0

\Rightarrow r0: 1001 0000 0000 0 ; lost (bit 3 want sign bit 0)





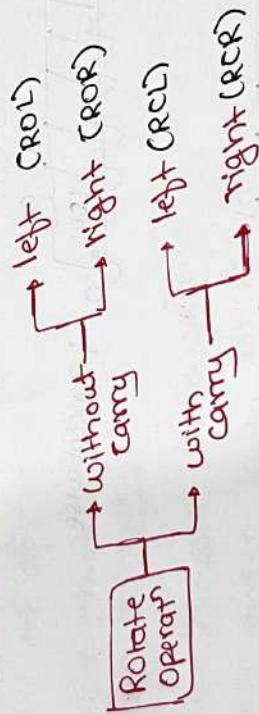
Ex: $[RS1|r0|#2]; r0: (CC2)H$

$$r0 = \begin{array}{c} \downarrow \\ \overline{\overline{1100\ 0000}} \end{array}$$

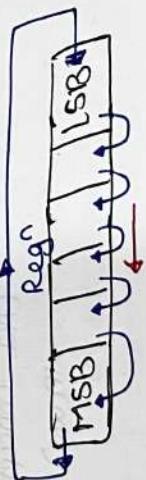
$$r0 = \begin{array}{c} \overline{\overline{1111\ 0000}} \\ | \\ (CF0)H \end{array}$$

Rotate Operatn

- while execution of these instr data is moving to left or right direction in a bit wise sequence without loss
- Circular shift operatn is a Rotate operatn

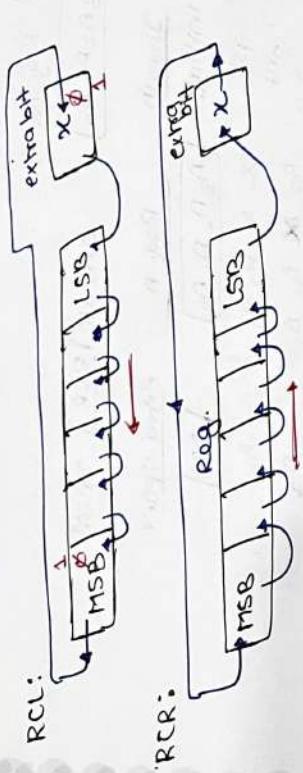


ROL:



ROR:





Ex: The top of RCL Instrm is given below

Information carry status. Data in the what is the IP register

Set → LSR Status
Reset → LSR Status

IP Data: 11100110

ex: ubit Reg-A execute ROR I n\l

<u>Iteration</u>	<u>Registers - A</u>	<u>In n-bit Register</u>
Initial :	$\boxed{A_3 \ A_2 \ A_1 \ A_0}$	Data in nth iteration of Reg/Ram
1st :	$A_0 \ A_3 \ A_2 \ A_1$	instr. terminated
2nd :	$A_1 \ A_0 \ A_3 \ A_2$	Data will be back to reg.
3rd :	$A_2 \ A_1 \ A_0 \ A_3$	

卷之三

Ex: 4-bit Register A

	A ₃ A ₂ A ₁ A ₀	execute RCR	shift
Initial	<u>A₃ A₂ A₁ A₀</u>		
1st	X A ₃ A ₂ A ₁	A ₀	<u>extra space</u>
2nd	A ₀ X A ₃ A ₂	A ₁	
3rd	A ₁ A ₀ X A ₃	A ₂	
4th	A ₂ A ₁ A ₀ X	A ₃	
5th	X A ₃ A ₂ A ₁ A ₀		

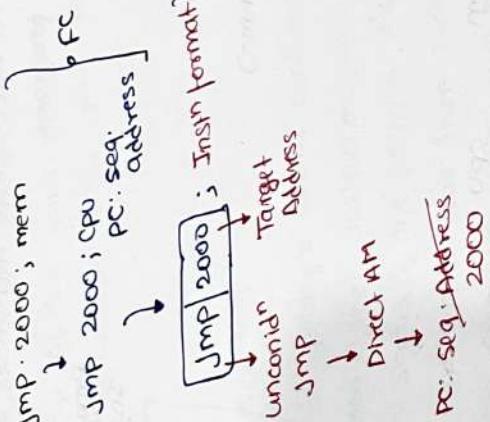
- ∴ In n-bit Registers Data, in $(n+1)^{th}$ iteration of RCR | RCL instruction the initial value will be back to register.

Transfers of control (TOC) instr

- conditional TOC instr
- unconditional TOC instr

- unconditional TOC instr
- these instr design without condition.
- while execution of these instr control will be transfer to target location without checking any condition.

|| label are illegal
in the Hlw part



Note

- These instrns is used to implement the unconditional selection Stmt (goto) & Mlc control instr (HALT) in the Hlw

HALT.

- It is a m/c control instr, used to control the functioning of machine &a device
- while execution of these instr invokes the unconditional jmp with starting address of halt as a target address
- $\text{re } [L_1 : \text{jmp } L_1]$ self control loop
- while execution of these instr CPU enters into a infinite loop means program control is transferred to same location again & again without change the data . so were point of view program execution is completed but CPU point of view, same instr is executing upto infinite time

- when CPU enters into a HALT state then reset the MP to execute the new program

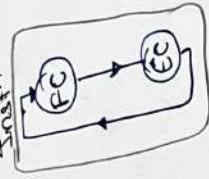
Code:

```

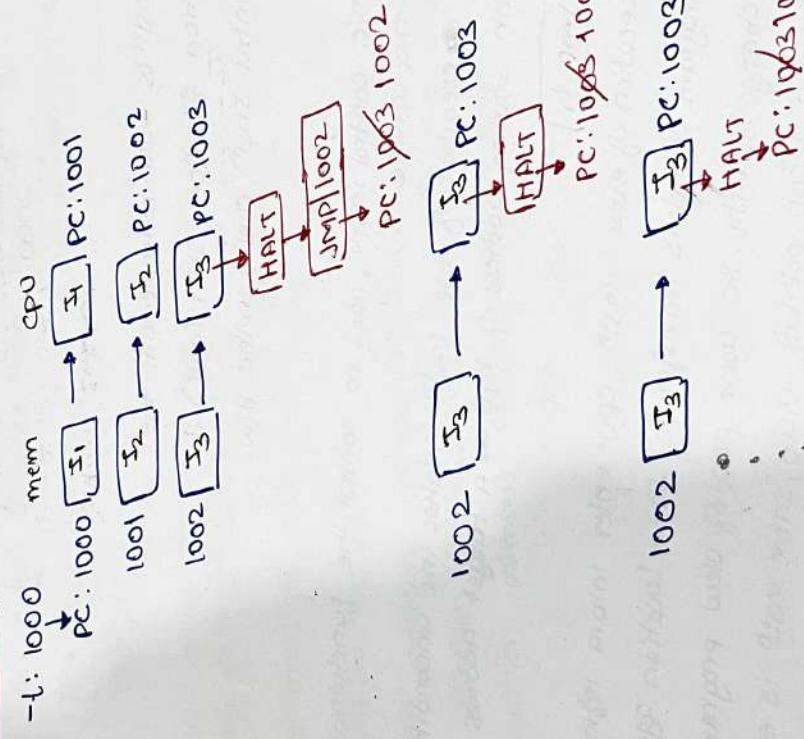
1000: T1
1001: T2
1002: T3 (HALT)
1003: T4
:

```

Program load
Instruction
is instruction



Execution flow.



O/P
seqn.

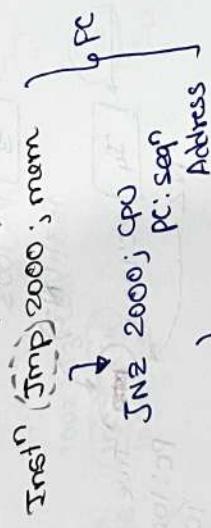
T₁ - T₂ - (T₃) - T₃ - T₃ - T₃ - ...
Halt

// until the user reset the up box it will be in the infinite loop. So wait. When you reset up box then default value is load into PC, now CPU is ready to execute new program.

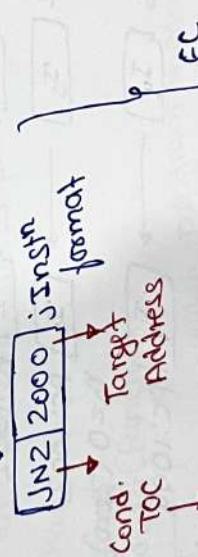
Conditional TOC Instr

- these instr is predefn with a condition while execution of the instruction, condition will be evaluate based on the status of the previous instr [Flags].
- when the condition is true then control will be transferred to target location, otherwise control will transfer to sequential location.

Instr (Jmp) non-zero [condition]



Instruction cycle



$NZ = \text{Status of zero flag}$
 $\text{Previous instr flag}$

True
(PC: seg Address)
2000

False
(no change
in PC)

Note:

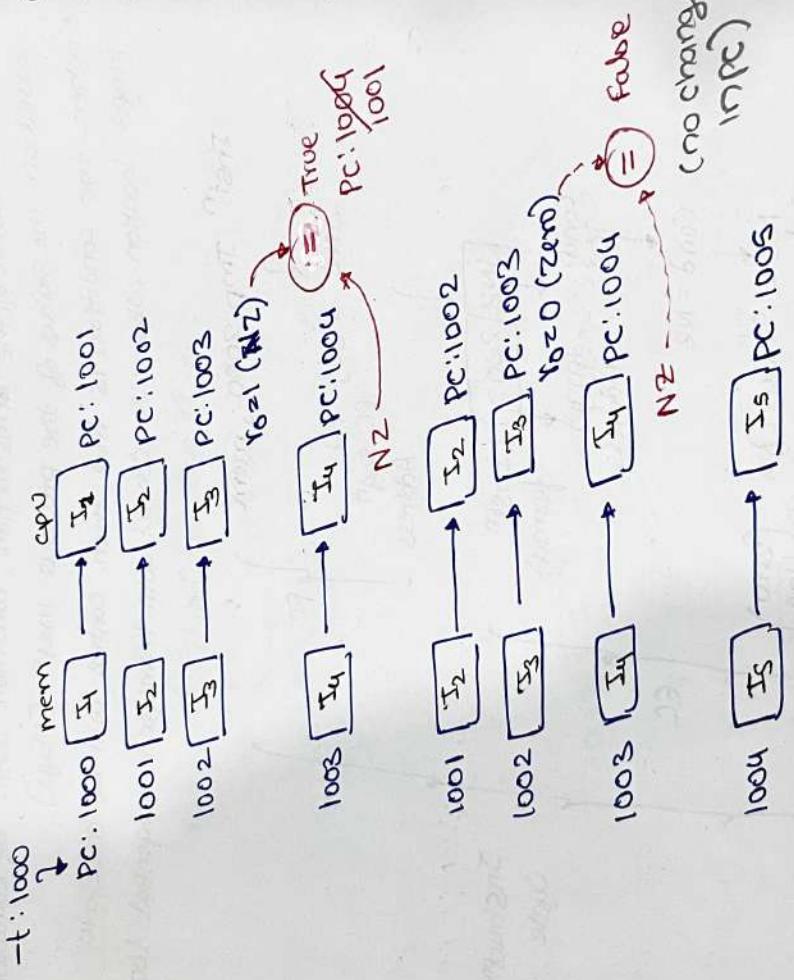
- This instruction is used to implement a question for while-do stmt in the base W/0
- Conditional TOC instn

ex: Code:

```

1000 : I1
1001 : I2
1002 : I3 CDEC 101; assuming I6=23
1003 : I4 CJNE 1001
1004 : I5
1005 : I6
    :
  
```

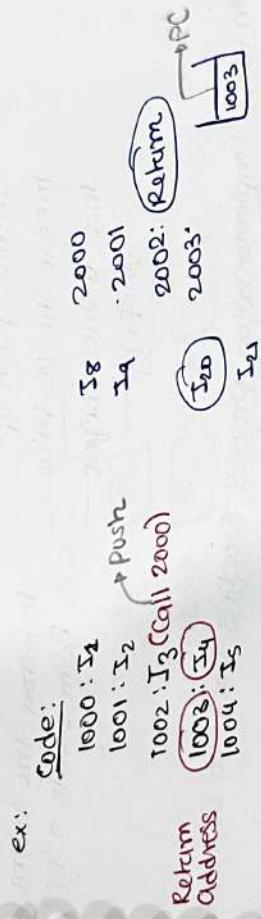
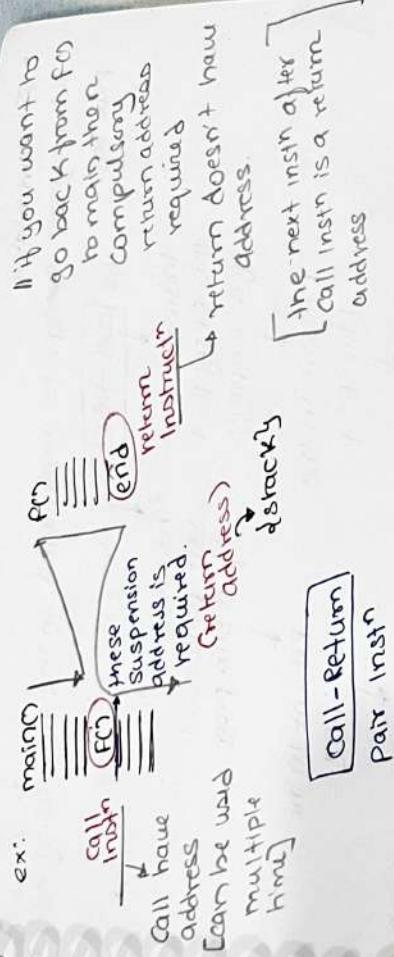
Execution:



O/p seqn: I₁-I₂-I₃-I₄-I₂-I₃-I₄-I₅-I₆-...

True False

looking like do-while Stmt.



So, call & return instrn are used as pair instrn to implement function in the H/W.

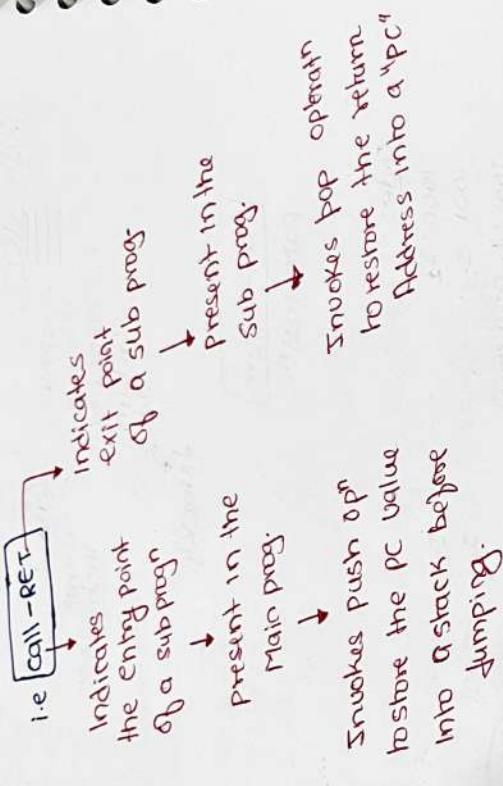
Sub program (function)

- Sub program is a reusable program means repeatedly occurred functionality in the application is developed as a sub program only once, later use it in the application many times [write once - Read many times]

Characteristics.

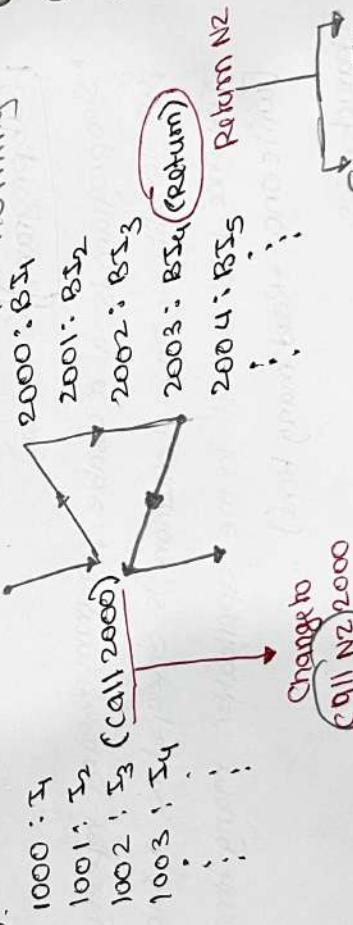
- single entry point - single exit point
- main program is suspended during the execution of a subprogram
- control will be transferred back to main program after the completion of subprogram

- Implementation
- Sub program concept is implemented in the HLLs using the pair instr i.e $\boxed{\text{call - RET}}$

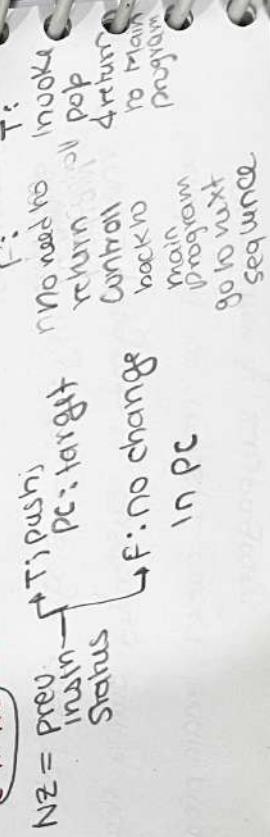


- In these implementation process stack is used to store the return address.
- return address is a next insin address after the call instr.

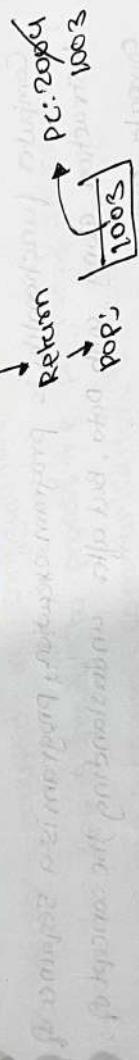
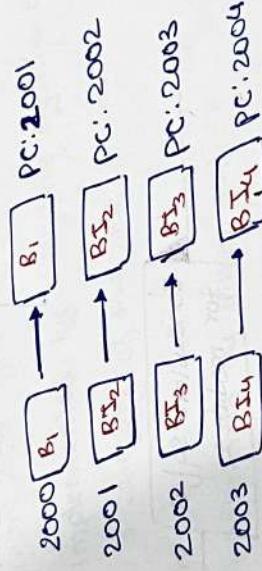
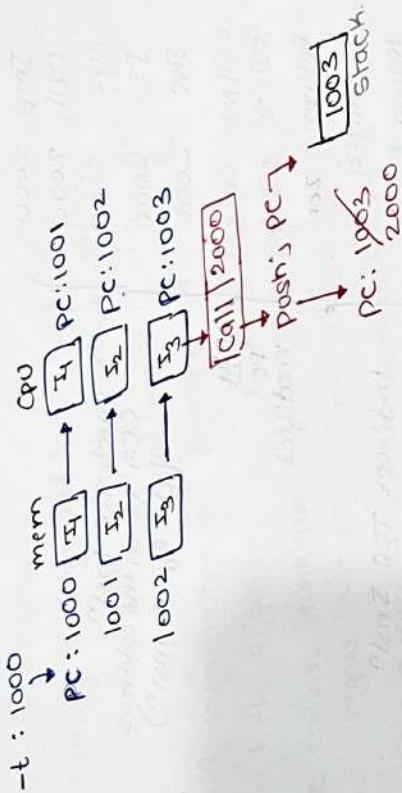
↑ B Standard code:



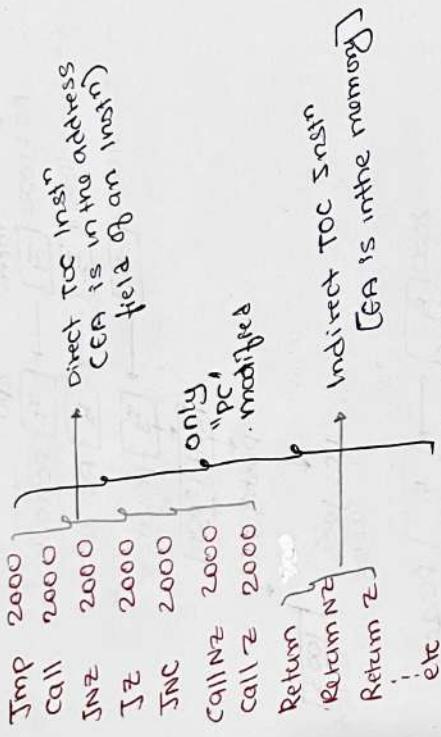
Change to
 $(call \boxed{N2} 2000)$



execution



TOC Inst



Conclusion so far
for module 2

Computer functionality is Program execution, program is a sequence of instruction along with data. But after understanding the concept of computer system we realised that Data is not an entity. Data information present in the instruction itself by using AT

entity. Data manipulation present in the instruction, program is a combination of data transfers, data manipulation & transfers of control instruction. Data transfer & data manipulation are designed with data centre address mode implies immediate, Register, Direct, Indirect, etc. Transfer of control instruction are design with a instruction centre addressing mode (AM) i.e PC-relative & base register.

- user developed program always present in the main memory to execute b/c Computer is designed with von-neumann architecture

so that application program always resides in the main memory to execute.

Main memory is organized into cells, cells are addressable unit, default cell size is 8 bit so that memory interfacing will be adjusted according to the word length of a CPU so multiple cells of information is transferred from CPU to memory interfacing technique.

little endian technique means low address contain lower byte & higher address contain higher byte. so when multiple cells interface into CPU, then data will will be accessed by the CPU & memory in a little endian order.

These interface implemented by using the **System Bus**.

System bus contain 3 category of bus line, Address line, Data line & control line.

System bus is implemented by either IOP or isolated IO or memory mapped or configuration when the CPU is ready to process the program & memory is ready with a program, communication is ready with a system bus. then we are going to execute the program. that program execution sequence is describe by the instruction cycle.

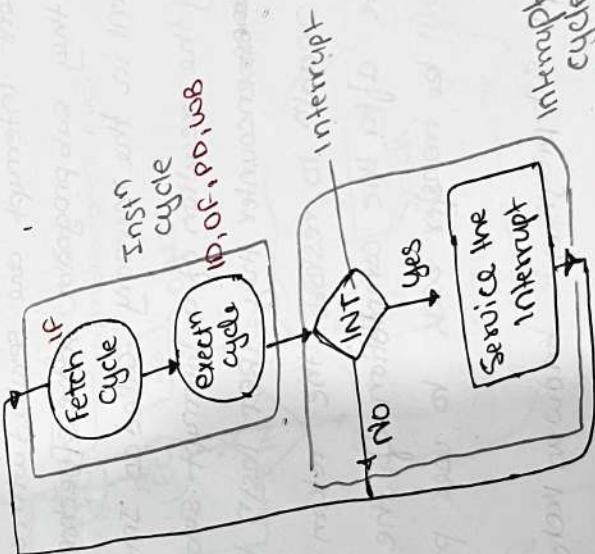
Instruction cycle contain fetch cycle & execution cycle. Fetch cycle objective is to fetch the instruction from the main memory, referen. I/O unit is program counter. so by using program counter CPU fetch the instruction from the memory.

- objective of execution cycle is processing, processing non instruction to produce an instruction. needs an instruction format
- instruction format depend on the type of CPU. CPU organised into stack CPU, accumulator CPU & register CPU. It is classified into memory CPU, register CPU, memory CPU, register CPU, memory CPU, register CPU.
- ALU to CPU instruction format are zero address, one address, two address & 3 address. Address format not in use. CPU program counter is common register in all the CPU. PC. PC is used to implement the displacement format.
- In the instruction execution, system supported instruction design if we want to use expand opcode technique a fixed length instruction than expand opcode technique when system support variable length instruction than no need of the expand opcode technique.
- So all together in the instruction life cycle, states are present 1F, 1D, 1C, 1B, 1A
- offset execution of time instruction the result is presented the condition flag used for CPU voice. So CPU convey the msg to user using conditional flag.
 1. conditional flag
 2. control flag

- control flag are flag user voice, user convey they msg to CPU using control flag
 - conditional flag are carry, parity, auxiliary carry, zero, sign...etc
 - control flag are trap flag, interrupt flag, direction flag
 - control flag are trap flag, interrupt flag, direction flag
 - so now computer is busy with program execution. If su already so device generated the "interrupt". so when interrupt occur to how CPU respond to true interrupt & how it handle the interrupt.

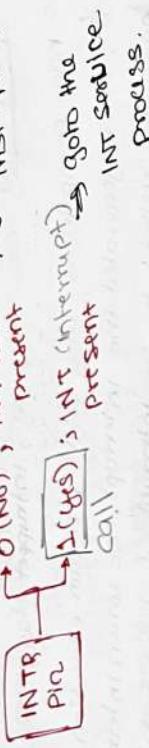
Interrupt cycle

- Interrupt is a signal which is used to handle the process
 - Interrupt cycle describes the instruction cycle at the interrupt cycle is connected to instruction cycle at the end of the execution cycle. So CPU will respond to the interrupt only after the completion of current instruction execution



Initialization
Interrupt Subprogram
Interrupt cycle is
objective of the

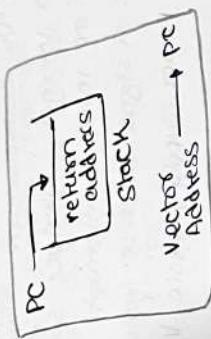
- After completion of current instruction, CPU reads the status of the interrupt pin to detect the interrupt that is.



- When CPU detects the interrupt, then it saves the "PC" value into a stack, then loads the vector address into "PC" to execute the interrupt sub program.
 - Interrupt sub program is referred with a vector address, and with a IRET instruction from exception
 - When IRET is executed, it returns to the interrupt return from interrupt.
 - Different interrupt are serviced with a different sub program.
 - So all they ^{CPU Interrupt Subprogram} sub program address [vector address] are present in the memory called as Interrupt Vector Table [IVT].
 - during the execution of a interrupt sub program, when the CPU encounter this IRET instn then it invoke the "POP" operation to restore the return address into "PC".
 - Therefore after the completion of the interrupt sub program, control will be transfer back to user program.
- \therefore Interrupt sub program is known as vector address

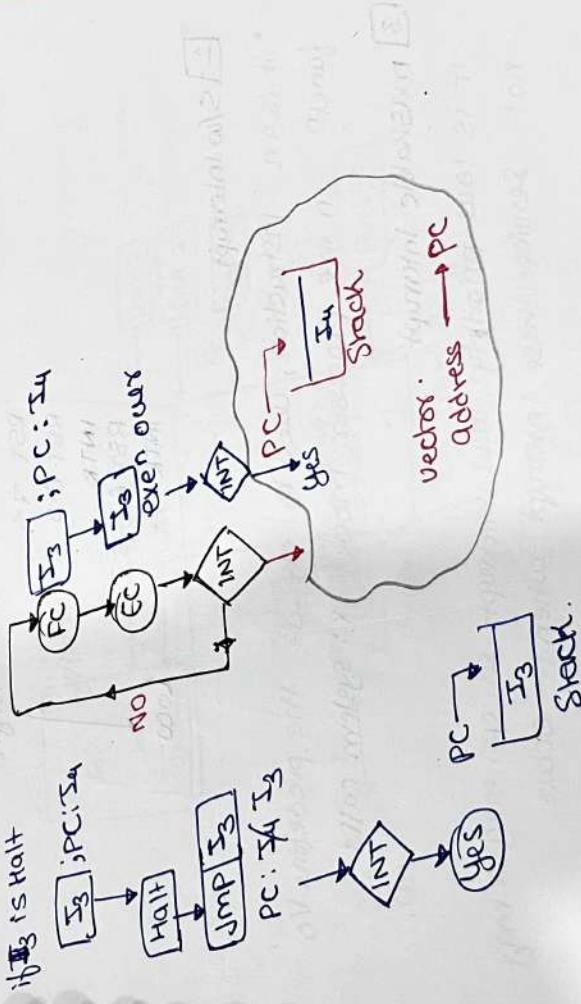
Note: Objectiva of interrupt cycle is, interrupt subroutine initialization.

means save the PC value into the stack & load the vector address into a "PC".



prog.

If I_3 is halt
 I_3 occurred during I_3 execution what will be
over occur
the return address pushed into a stack

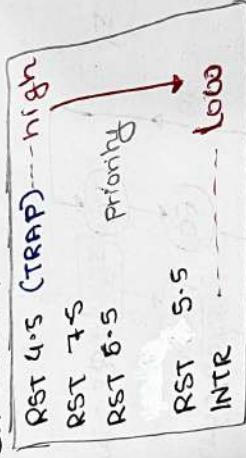


Type of Interrupts

1 H/w Interrupt

- It is a signal generated by the H/w component connected to the CPU using external or internal interface. So H/w interrupt are external interrupt or internal interrupt.
- External interrupt is a signal which is generated by the external H/w i.e. Power supply, basic I/O device (Keyboard, printer, etc).
- Internal interrupt is a signal which is generated by the internal H/w present in the motherboard.
ex: Temperature sensor, timer, some critical sensors, Invalid opcode, Divide-by-zero, stack overflow, etc.
- In the CPU design, H/w pins are reserved to hold them

H/w interrupt ex: INT goes to IP



2 S/w interrupt

- It is an instruction used to execute the predefined I/O function in the processor mode ex: system calls

3 maskable interrupt

- It is low priority H/w interrupt so CPU may or may not services those interrupt when it occurs.

- ④ non-maskable interrupt.
- It is a high priority M/W interrupt so can compulsory service where interrupt won't occur

- ⑤ vectored interrupt
- these interrupt contain interrupt vectors to specify their location of a vector address e.g. RST (4.5) Interrupt vector.

RST (4.5) Interrupt

vectors

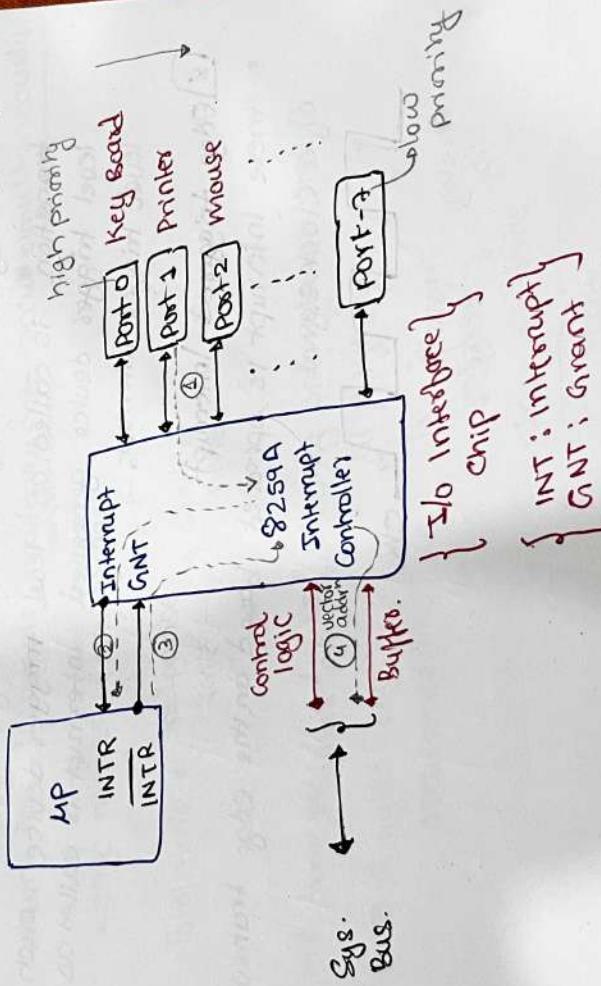
Show the vector address location

1) So interrupt with interrupt vector which supply the vector address information

⑥ non-vectorized interrupt.

these interrupt doesn't contain the interrupt vectors. so CPU send the ACK (Acknowledgment) & waiting until the interrupt source supply the vector address. ex: INT

(fixed priority)



mode of 8259 A

- 1). single buffer mode
- 2). cascading mode:
 - max 8x I/O device are possible to connect using master & slave approach.

Priority assignment:

- ① fixed priority: lower part having higher priority after service the request.
- ② variable priority: rotate the priority.

⑦ Level Triggered Interrupt: operated based on the level transition

- These interrupt is triggered by a clock signal.

of a clock signal.



- the signal that operate b/w high level & low level transition is called as level trigger device. such transition is called interrupt is called as level trigger device generated interrupt

⑧ Edge triggered Interrupt

- These interrupt is operated based on the edge transition

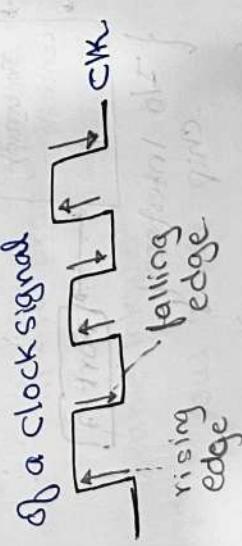
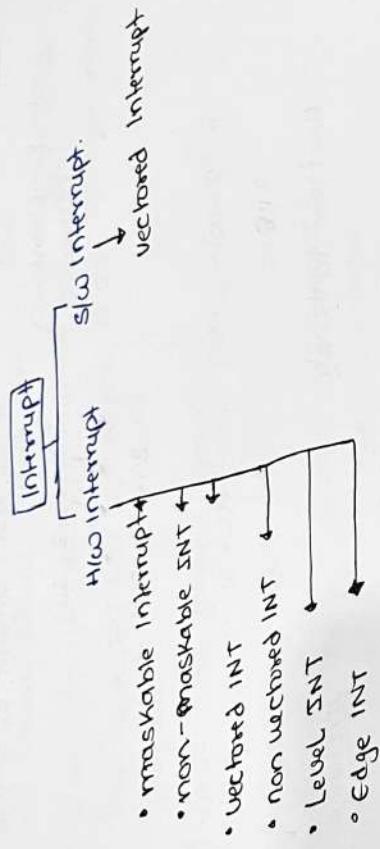


Diagram: 8259
mode, INT

Interrupt structure.



Q) Consider a hypothetical CPU which supports 4 interrupts (I_1, I_2, I_3, I_4) with a respective service time of 2ns, 5ns, 3ns, 4ns. Response time of an interrupt is 2ns. I_1 has the highest priority & I_4 has the least priority. What is the range time required to service "I₄" interrupt when the interrupts may/may not occur simultaneously.

Sol: Range means & min to max

$$\text{Total time} = \text{Response time} + \text{Service time}$$

min time required for "I₄": only one interrupt present at a time, hence 2ns
(w/o simultaneous occurrence) \rightarrow [2ns]
i.e. I₄ will occur after 2ns

$$2\text{ns} + 10\text{ns} = 12\text{ns}$$

max. time req. for "I₄" \Rightarrow All 4's are present at a time

(with simultaneous) (I_1, I_2, I_3, I_4)



2ns **2ns** 2ns 2ns

$$\Rightarrow (2+2)+ (2+5)+ (2+3)+ (2+1)$$

$\Rightarrow 18 \text{ ns}$

Ans 21ns to 18ns

- Q) consider 2ns clk cycle CPU which consume 6 cycle for memory reference, 4cycle for ALU op & 0 cycle for registers. reference used to execute the following code stored in the memory with a starting Address of (AAA)₁₆. CPU word length is 16bit & support word addressable memory. Design.

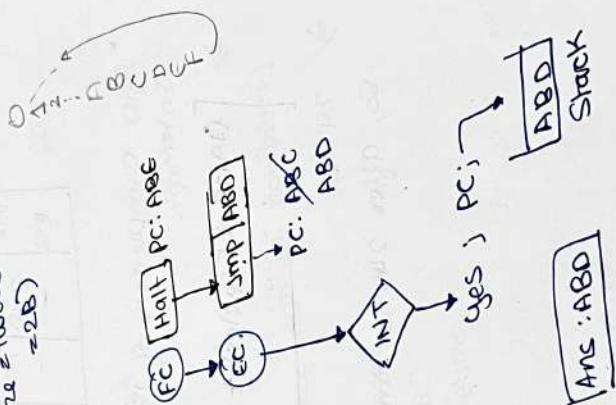
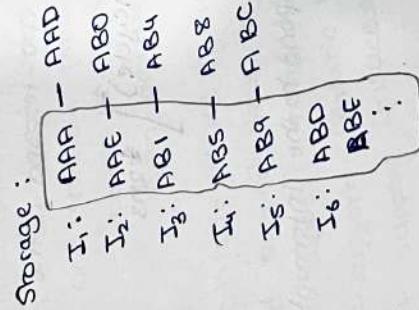
Instr	meaning	size in bytes.	words
Mov r ₀ , @2000	r ₀ \leftarrow M[2000]	8	1
Mov r ₁ , 3(r ₀)	r ₁ \leftarrow M[8+r ₀]	6	1
Add @3000, r ₂₀₀₀	M[8000] \leftarrow M[8000] + M[2000]	8	1
MUL r ₂ (r ₀), @2000	M[3+r ₀] \leftarrow M[3+r ₀] * M[2000]	8	1
Mov @4000, r ₀	M[4000] \leftarrow r ₀	8	1
Halt.	M[4000] \leftarrow No	2	1

- (a) If an interrupt occurred during halt inst. what will be the return address pushed into a stack
- (b) If an interrupt occurred during ADD inst. what will be the return address pushed into a stack
- (c) If an interrupt occurred during HALT inst. how much time can CPU respond to the interrupt before program execution?

(d) What is the program execution?

Ans:

word size = 16bit
 memory config = word addressable
 $\text{CCW size} = \text{word}$



(b)

(c) execution

FC	EC	OF	PD	WB	
IF	ID	S ₁	S ₂	D	
I ₁	1MR	3MR	2MR	-	IRR
I ₂	1MR	2MR	IRR	-	IRR
I ₃	1MR	3MR	2MR	1MR	2MR
I ₄	1MR	3MR	IRR	1ALU	IRR
I ₅	1MR	3MR	1ALU	1ALU	1ALU
I ₆	1MR	3MR	1MR	1MR	2MR

$$\Rightarrow \boxed{144 \text{ ns}}$$

• After completion of 1st blodin CPU respond to the interrupt.

$$\Rightarrow \boxed{(1MR * 6 cycles) + (2ALU * 4 cycles)} \quad 24ns$$

→ 24ns.
So, after 24ns time, CPU respond to the interrupt

RISC vs CISC

Characteristics of RISC

- [Reduced Instn set computer]
- It supports more registers.
- It supports less addressing mode.
- It supports fixed length instr.
- It supports sequential pipeline.

CPI = 1

- It is a super computer.
- It is used in real time application.
- It is an expensive processor.
- It contains smaller instr set.
- + use hard-wired control unit
- ex: Motorola processor.
Power PC processor.

ARM processor.

Characteristics of CISC [Complex Instn set computer]

- It supports less registers.
- It supports more addressing mode.
- It supports variable length instr.
- It supports unsuccessful pipeline.
- It supports

(cycle per instr)

CPI ≠ 1

- It is a general purpose computer.
- It is used in personal applications.
- It is a less expensive processor.
- It contains larger instr set.
- It uses microprogrammed control unit.
- e.g.: Pentium processors

- Register organisation in "RISC" CPU.
 - "RISC" CPU support more registers, catalogued into 4 groups
 - global register (G)
 - local register (L)
 - IN register (I)
 - OUT register (O)
 - In the "RISC" CPU registers window is formed by grouping the IN, OUT & local registers.
 - Global registers are accessible by all three windows therefore $Windows = L + 2C + G$
 - In the "RISC" CPU, register windows are organised into overlapping order. That is one window's OUT Registers are used as IN Registers in the other's window.
 - overlapping order.
- Reg. organ in RISC CPU
-

- Q: # of registers in the CPU
 A: $W = L + C + G$
 Q: Registers file size
 A: $W = (L + C + G)$

$$\# \text{ of registers in the CPU} = W = L + C + G$$

- Q: no. of windows in the CPU
 A: $W = 32$
 Q: # local registers
 A: $L = 16$
 Q: no. of global registers
 A: $G = 5$

Q: consider a hypothetical computer which contains 32 global registers, 16 local registers, 16 in-registers & 16 out-registers in the CPU, 42 registers. windows are overlapped. what is the size of a window & registers in the CPU?

$$\begin{aligned} \text{Window size} &= L + C + G \\ &= 16 + 2 * 16 + 32 \end{aligned}$$

$$= 88$$

$$\begin{aligned} \text{Registers file size} &= W(L+C)+G \\ &\Rightarrow 32(88+16) + 32 \\ &\Rightarrow 1712 \end{aligned}$$

Conclusion

In these module we learned or analysed only the program execution sequence in that computer.

Module: 3

Computer organization

Computer system contain 3 fundamental components



1. CPU
2. Memory
3. I/O

CPU organization

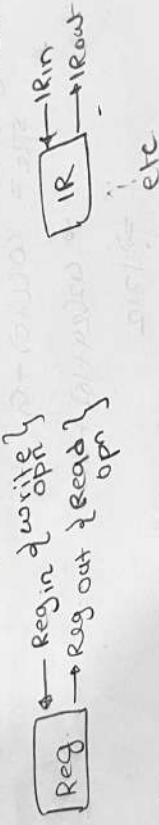
cpu is a processing unit in the computer.

It contains 3 internal component

- ① Registers
- ② ALU
- ③ Control unit (CU)

Registers

It is a internal storage component of a CPU



every CPU contain set of mandatory registers.

list as

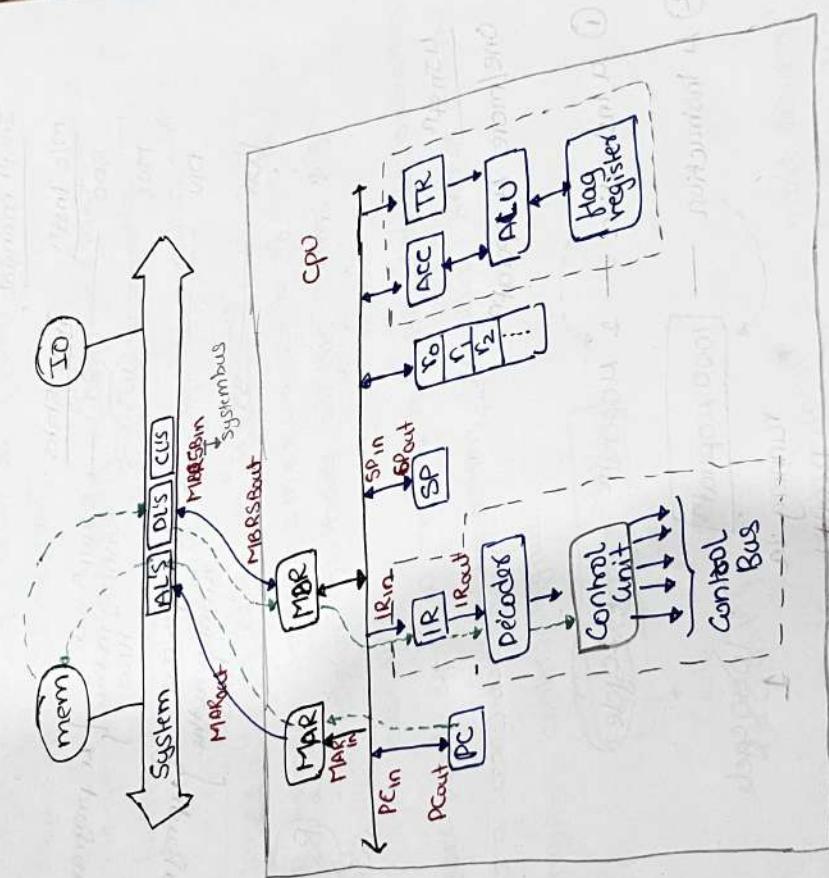
- ① PC: Holds the instr address
(Instruction Register)
- ② IR: Hold the opcode
- ③ ACC: Hold the ALU IP & ALU OIP
- ④ TR: Holds the ALU 2nd operand
- ⑤ MAR(Memory address Register): Holds the memory address connected to Address Bus.

⑥ **memory** : Hold the memory content connected to Data bus
 (memory Buffers/ Data registers)

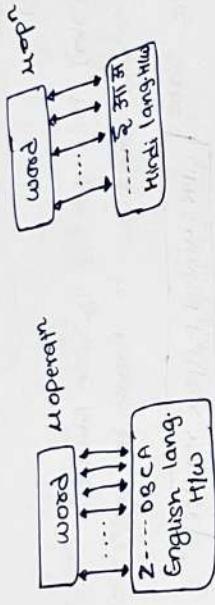
⑦ **SP (stack pointer)** : Hold the top of stack Address.

⑧ **Flag register** : Hold the flags to handle the instruction status.

⑨ **GPR**: Hold the Data GPR: general purpose registers



Micro-operation



Instn manual.

mic Instn Time stakes

ADD	(8T)
MUL	(10T)
DIV	(12T)
LOD	(BT)
...	



μInstn

One/more μ operatn

cycle

① μ instruction → μ operatn

② μ instruction → 1000 μoperatn
running in 1000 cycles

CPU with 6-bit opcode — 64 operation/instruction

CPU with 8-bit opcode — 320 opn / instrn

CPU with 7-bit opcode — 128 operation / instrn

μoperation is a elementary operation in the Base Hilo (skelton) also called as [primitive operation]
Atomic operdn in the Hilo]

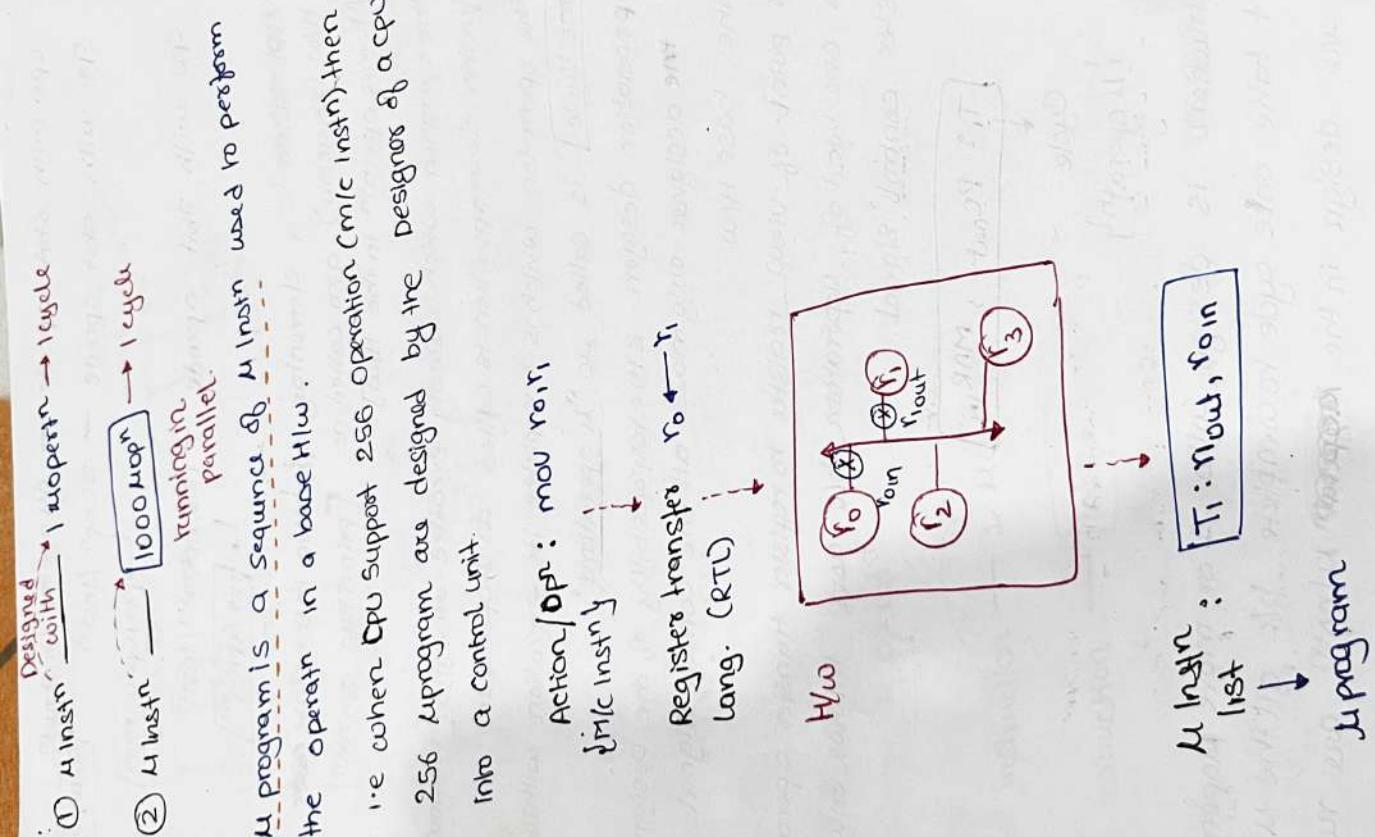
- the operation which is directly executed on a Hilo without further decomposition is called as μ operation.
- the operation which is completed as μ operation within the operation which is completes its execution
- One cycle is called as "μ operations"
- In operation design is the responsibility of the designer b/c designer only knows all the control signal

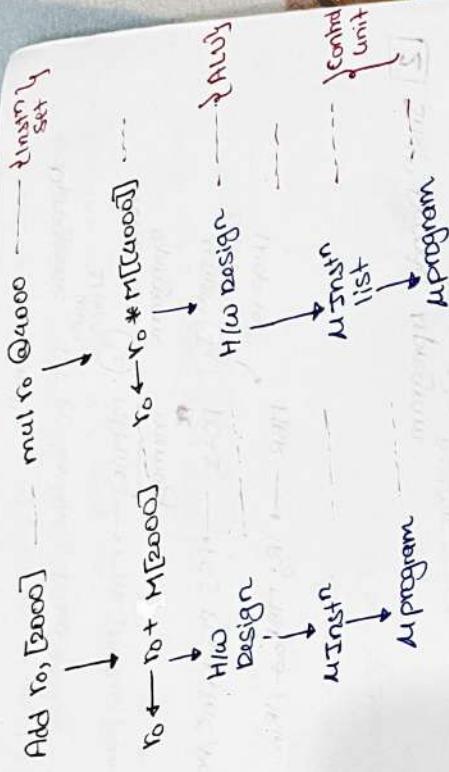
In the base Hilo user point of view registers no register transfers operation is a one word of operation . b/c user knows only the register control signal.

ex:
 $T_1 : PCout , MARin$

μ operation
Cycle

μ instruction is design with 2 or more μ operation still it take one cycle to complete b/c all the operation which are design in the parallel.



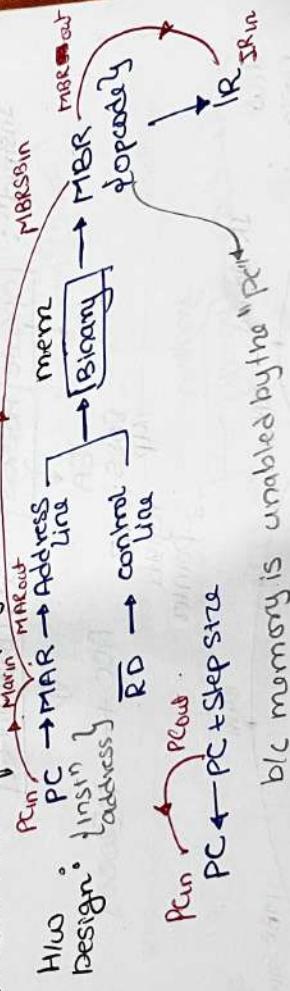


Microprogram (Microprogram)

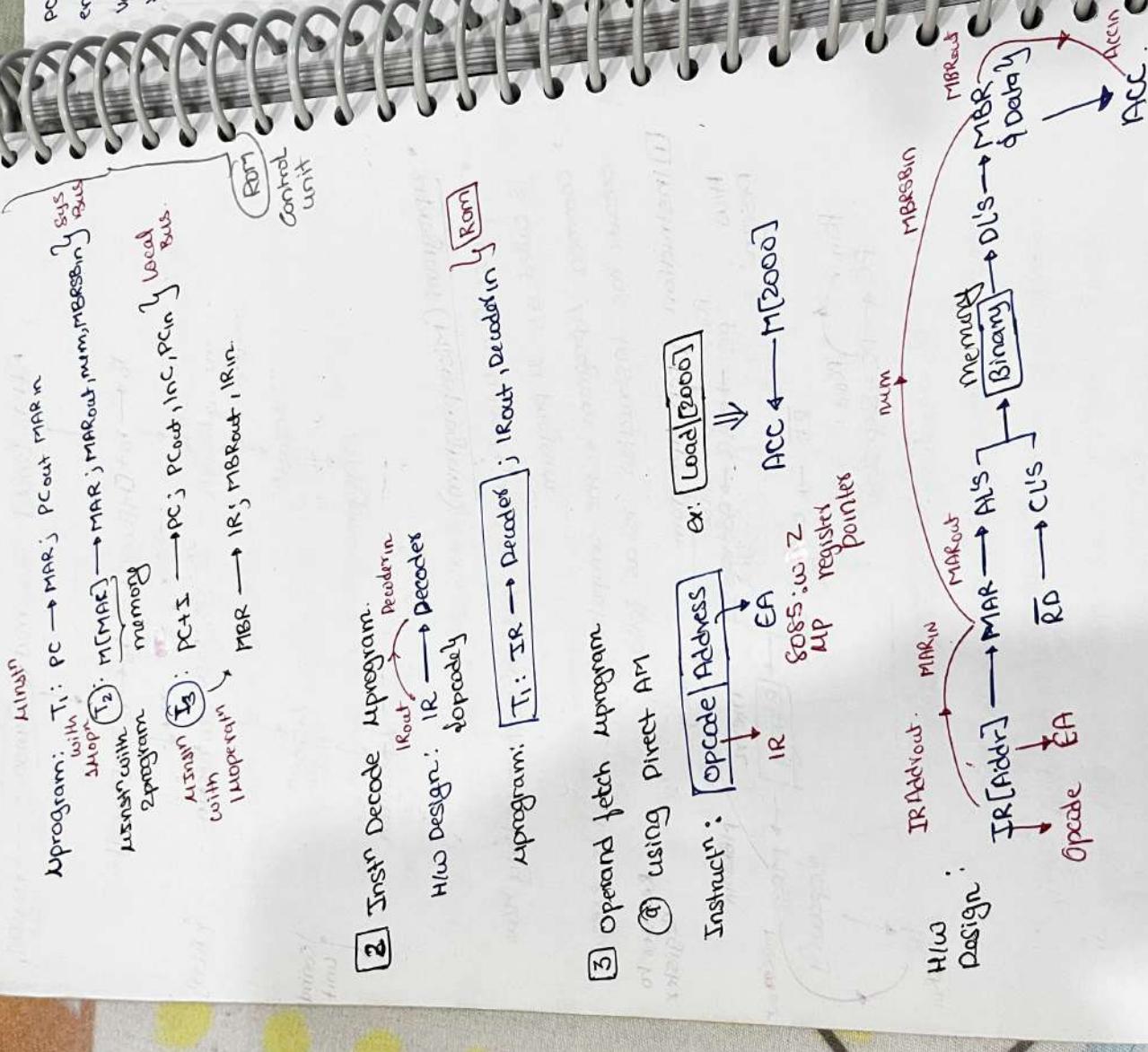
Sequence of a microinstr., used to perform action in they H/W is called as micro program

common microprogram in the computer design , used to execute execute the instruction is as follow

① Instruction fetch microprogram.



blk memory is enabled by the "PC".
when you enable it with 1st time PC then
compulsory it is an op code . after decoding again
you are enabling based on pc that is not op code
that is address part of instrn.



∴ PC is not involved in the of "PC" increment process stop at the end of "ID". At the end of "ID" PC became valid PC that valid PC is used in the next fetch cycle. So PC is not involved in the OF, PO, WRS

So, IR_{out} & IR_{Address}_{out} → along with OPCODE of you want address then use IR_{Address}_{out}.

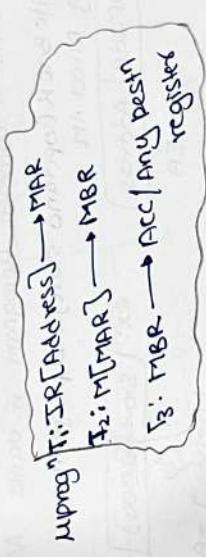
Inside the IR

Information out

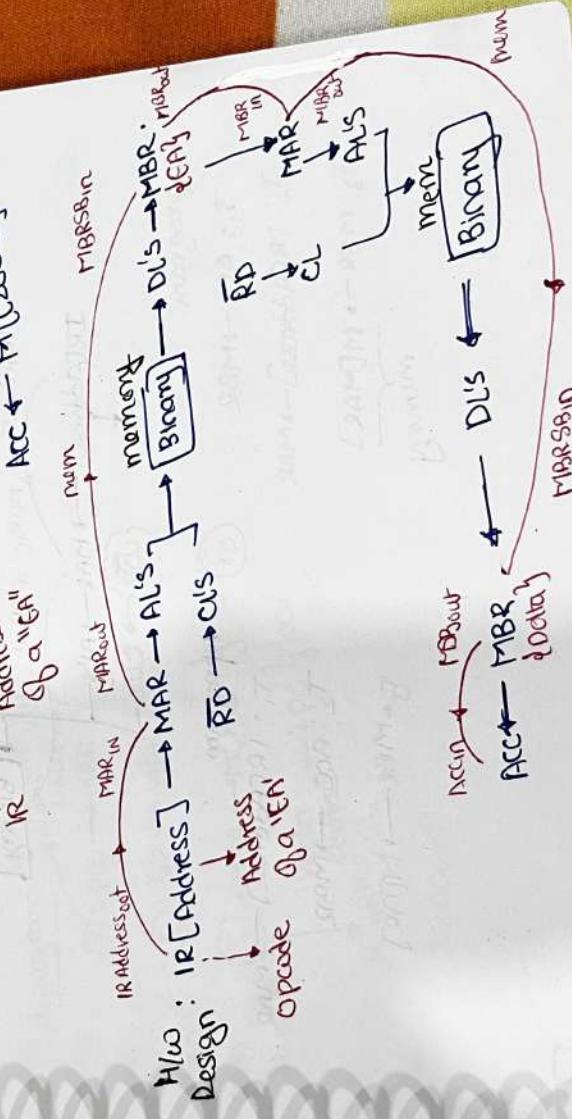
(Inside IR is OPCODE)

that OPCODE get out to

Decoder.



⑥ By using indirect AR.





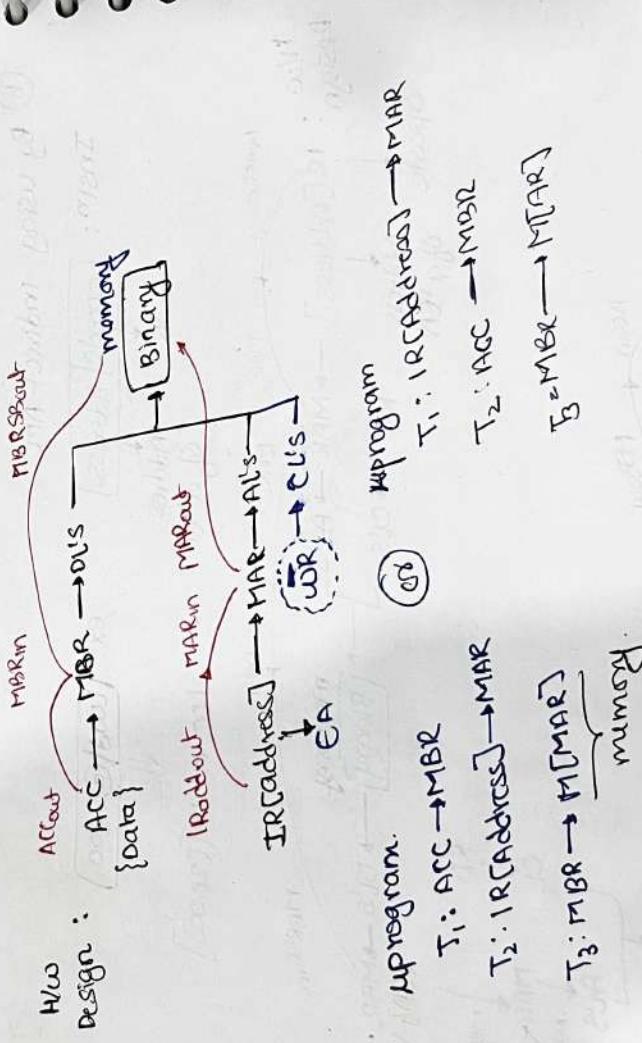
[4] Process Data Upgradem

It is not a common programming language.

5 write back (operand store) \leftarrow

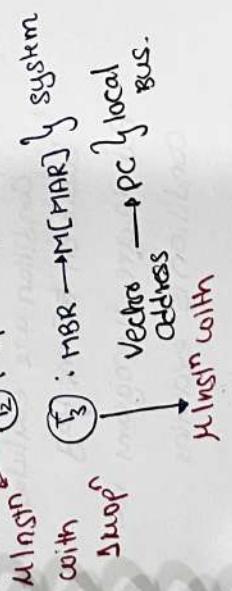
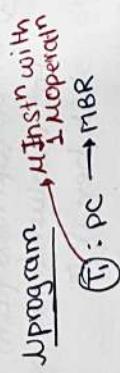
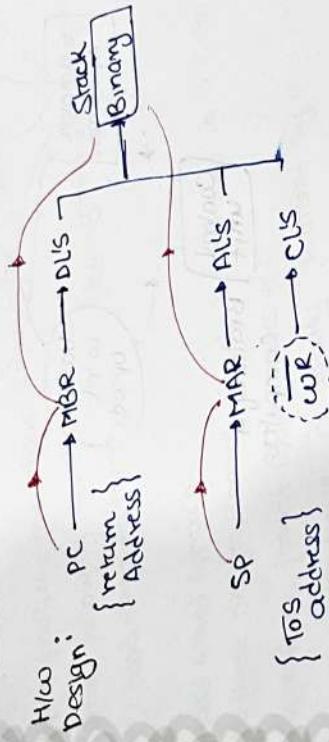
using direct address

$M[2000] = ACC$



158

[6] Interrupt subprogram initialization program



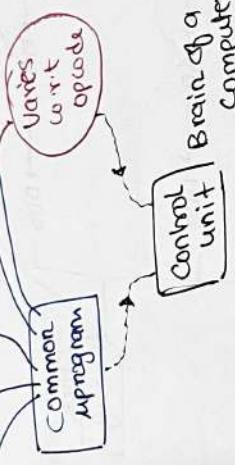
μInsr with μOpnd

μInsr with μOpnd

μInsr with μOpnd

μInsr with μOpnd

Instn cycle		Execution cycle		Interrupt cycle	
Fetch cycle	IF	ID	OF	PD	WB
Instruction address					
Memory					



"llsubit" option one by one

Consider the following program
 $T_1: PC \rightarrow MBR$
 $T_2: MBR \rightarrow MAR$
 $T_3: MBR \rightarrow \text{memory}$

which of the following

Op is satisfied with the above

program

IF

OF

conditional JMP

conditional JMP
 ② Interrupt subroutine
 Initialization.

assignment
~~to~~ MAR not to
~~if~~ PC \rightarrow MAR
~~in~~ MBR in "instn"
~~in~~ "OF" (operand fetch)
~~in~~ "PC" (instruction)
 but here it is

conditional JMP here
~~condition~~ are evaluate
~~based on~~ some flag

But we are program no
 conditions examining

Control Unit Design

Pre-Requirements

- [1] how many control signals are present in the base H100 (Skeleton)
- [2] how many instructions implemented in the base H100 (Skeleton)
- [3] how many micro ops required for each instruction for each instruction
- [4] what are the control signals required for each micro op

After finalizing the above requirement, control unit implemented using

- After finalizing the above requirement, control unit implemented using
 - ① Hard-wired approach
 - ② Programmed Approach

• Hard-wired approach

• In these designs, control signal is expressed in a sum of product expression format (SOP expression)

• Control expression is directly realized by the independent hardware called as Hard-wired control unit

• It is a fastest control unit

Value	Value	Value
000	001	010
011	100	101
110	111	011

• It is used in the real time application & reconnection of a

- even a minor modification required redesign & redesign
- Control signal. Hence it is not flexible
- It is not suitable in the design & testing phases
- It is not suitable in the design & testing phases
- RISC control unit is a Hardwired control unit.

Programmed logic design for combinational

$$\begin{aligned} & (X+Z)T + (Z+Y)T + (Y+X)T = 903 \text{ of } \\ & (X+Y)T + (Z+X)T + (Y+Z)T + (X+Z)T = 1000 \text{ of } \\ & X = 1000 - 903 = 97 \text{ of } \\ & Y = 1000 - 903 = 97 \text{ of } \\ & Z = 1000 - 903 = 97 \text{ of } \end{aligned}$$

- Sample cu Data

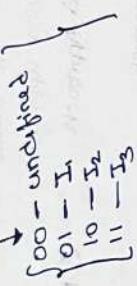
Pre-requirements

$$\textcircled{1} \# \text{CS's in the CU: } \{S_0, S_1, S_2, S_3\}$$

$$\textcircled{2} \# \text{Ins'n in the CU: } \{T_1, T_2, T_3, T_4\}$$

ADD
SUB
MUL

$$S_{72} = \log_2 3 \rightarrow 2 \text{ bit}$$



$$\textcircled{3} \# \text{Instruction/Instn: } \{T_1, T_2, T_3, T_4\}$$

- Database of the CU.

Minsty/ Inst	T ₁	T ₂	T ₃
T ₁	S ₀₁ S ₁₁ S ₂₂	S ₀₁ S ₁₁	S ₁₁ S ₂₂
T ₂	S ₀₁ S ₁₁	S ₂₂ S ₃₃	S ₀₁ S ₂₂
T ₃	S ₀₁ S ₂₂	S ₂₂ S ₁₁	S ₂₂ S ₀₁ S ₁₁
T ₄	S ₀₁ S ₃₃	S ₂₂ S ₁₁	S ₀₁ S ₂₂ S ₁₁

Hard-wired CU Design [CS: SOP expression]

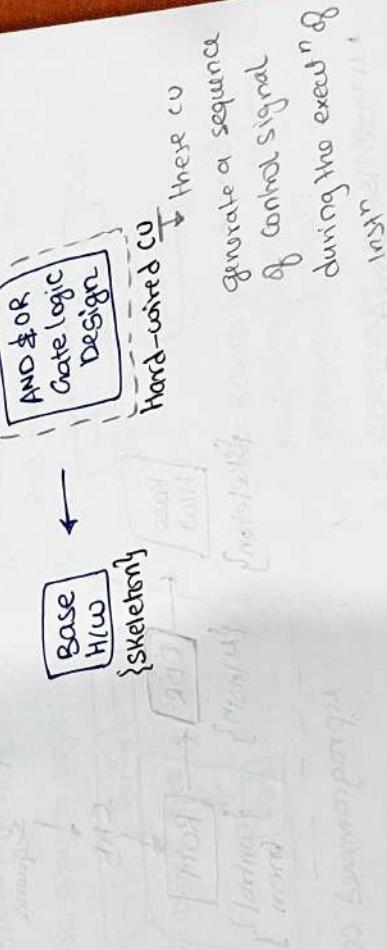
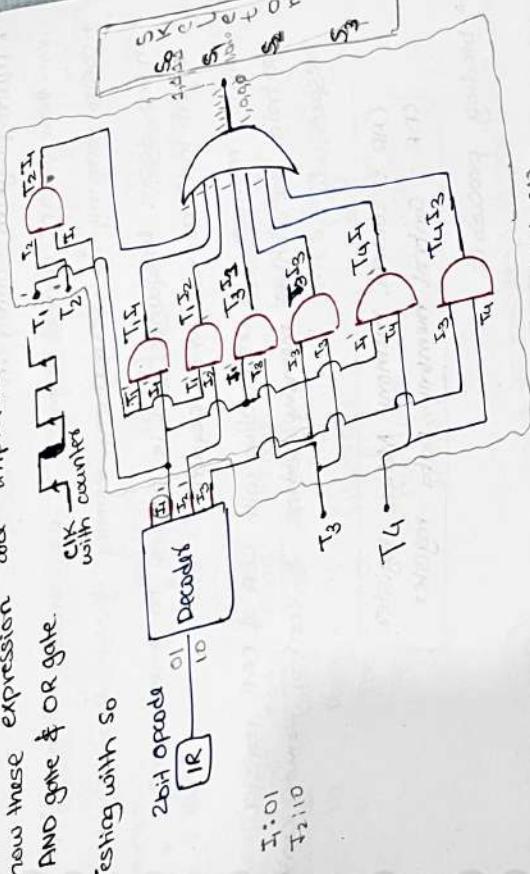
$$S_0 = T_1(T_1 + T_2) + T_2T_1 + T_3(T_1 + T_2) + T_4(T_1 + T_2)$$

$$S_1 = \underbrace{T_1(T_1 + T_2 + T_3)}_{T_1 \text{ can be written unmanually required}} + T_2(T_1 + T_3) + T_3(T_2 + T_3) + T_4(T_2 + T_3)$$

$$S_2 = T_1(T_1 + T_3) + T_2(T_2 + T_3) + T_3 + T_4T_2$$

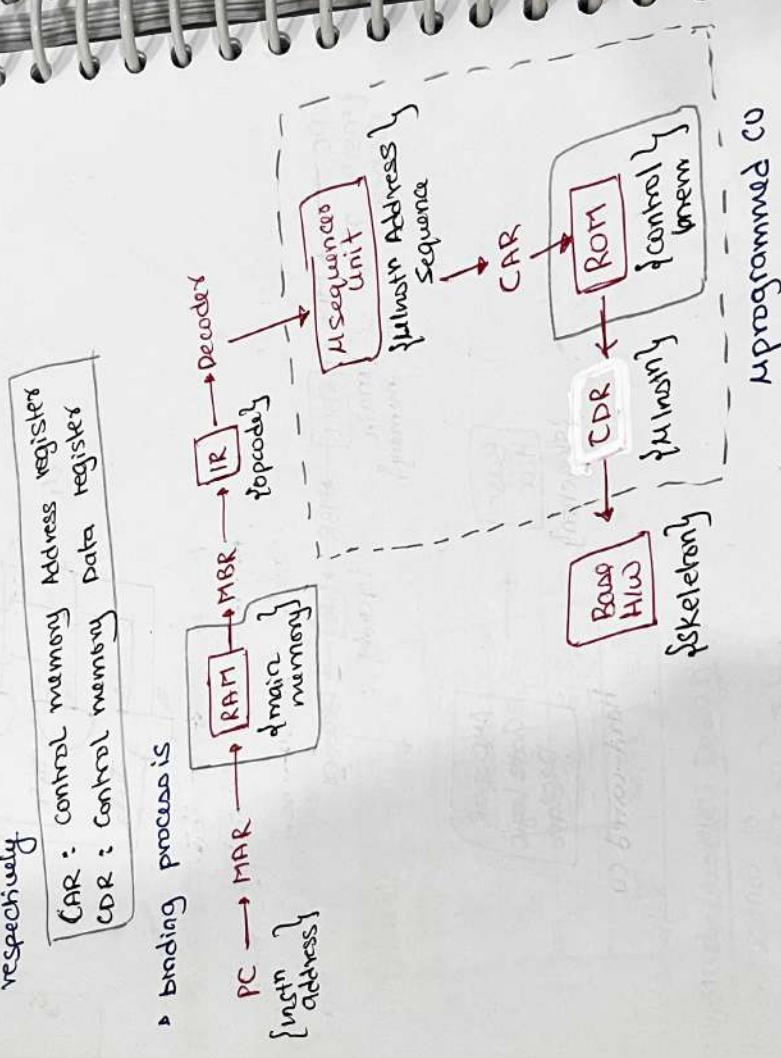
$$S_3 = T_1T_3 + T_2(T_2 + T_3) + T_3 + T_4T_2 + T_4$$

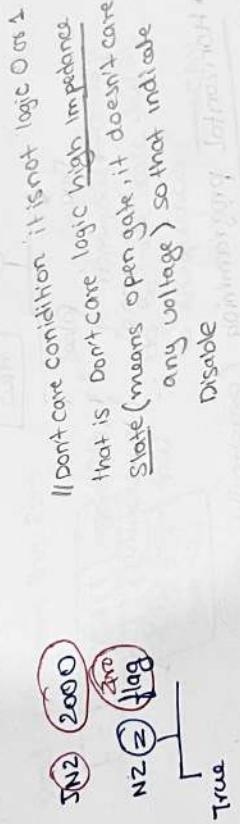
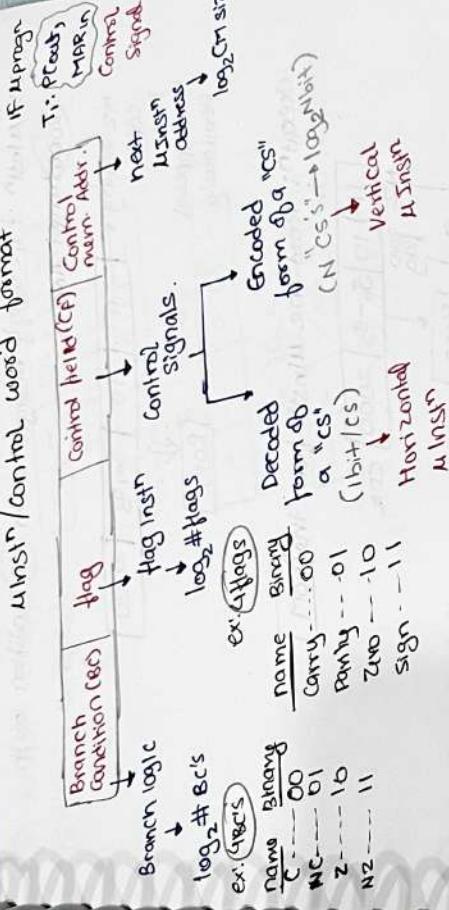
Know these expression
AND gate & OR gate
1e. Testing with so



16A

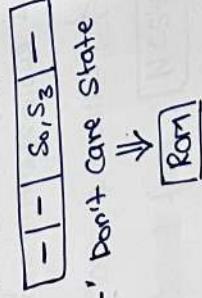
- micro program control unit
 - in these designs, control memory is programmed with a micro program
 - in these designs, control memory is programmed with a micro program
 - in these designs, control memory is permanent memory that is read only memory
 - control memory is permanent memory that is read only memory
 - in the design sequencer unit is present to generate the micro instrn address in a sequence
 - in the design sequencer unit is present to generate the micro instrn address in a sequence
 - control memory is associative with CCR & CAR register, used to hold the control memory address & control memory content respectively





$\mu\text{Instn} \{ T_1 : S_0, S_3 \}$ design

with Branch logic



|| Don't care condition it is not logic 0 or 1
 that is, Don't care logic high impedance
 State (means open gate, it doesn't care
 any voltage) so that indicate
 disable

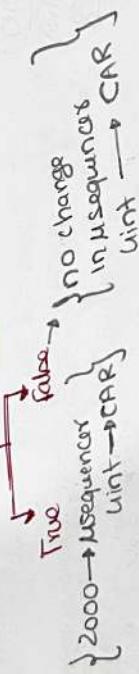
i. sequencer \rightarrow CAR
 {unison Address}

Ex: When tr_1 ; so why design with $\text{N}2$ condition with

2000 Target Address
 $\text{N}2$ Binary $\leftarrow \begin{array}{|c|c|c|c|c|c|} \hline & 11 & 10 & S_0, S_3 & 2000 \\ \hline \end{array}$ \downarrow
 code
 that
 we supposed
 previously

execute: Read the MInst from the ROM

11	10	S ₀ -S ₃	2000
N2	2000	Hw	



Horizontal programming (practically not possible)

[Refers the sample CS data given in the previous section]

① #CS's in Hw: {S₀, S₁, S₂, S₃}

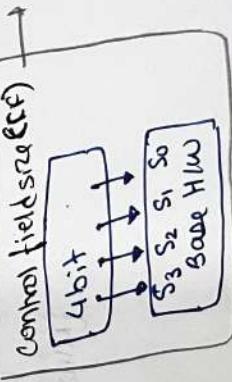
② Decode from : $\begin{cases} \text{Hw}/\text{CS} \\ 0 \text{ or } \text{CS} \end{cases}$ \rightarrow 0 (possible)
 \rightarrow 1 (enable).

So, to represent 4CS's in

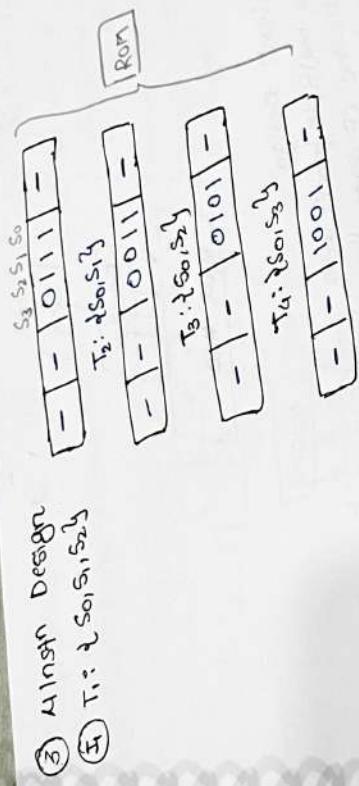
|| we should store all the Hw. 4 bit required
 to connect all the

connection to
 program the
 Hw

|| we should store all the Hw. 4 bit required
 to represent 4CS's in

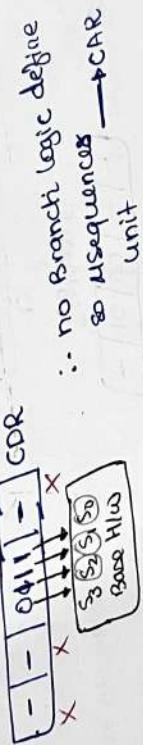


These concept is not in
 practical use
 because all the
 connection which is
 not possible.



④ Operational state.

(I) Read " T_i " from the ROM

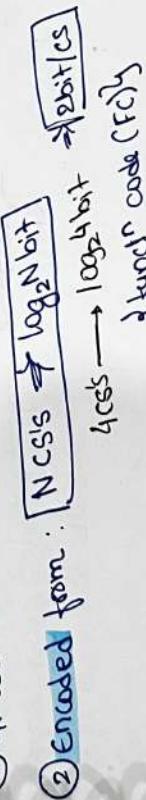


∴ no Branch logic define
so sequences → CAR unit

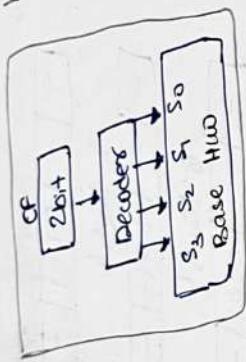
Vertical programming (practically possible)

[Refer the sample CU data given in the previous section]

① #CS's in the H/W : $\{S_0, S_1, S_2\}$



here no need to remember
the connections to
program the Minin.



③ Minin design:
use multiple functn codes
in the CF when the
Minin required multiple CS's

④ $T_1: \{S_0, S_1, S_2\}$

-	-	00	01	10	-
---	---	----	----	----	---

$T_2: \{S_0, S_1\}$

-	-	00	01	-
---	---	----	----	---

$T_3: \{S_0\}$

-	-	100	10	-
---	---	-----	----	---

$T_4: \{S_0, S_3\}$

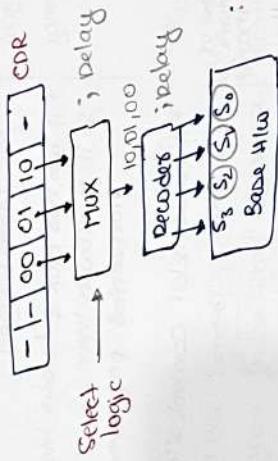
-	-	00	11	-
---	---	----	----	---

-	-	00	11	-
---	---	----	----	---

ROM

Programmed ROM

- (G) Operational State
 (H) Reading "1" from the ROM



// whenever multiple
Code are there.
many to one
multiplexers
(multiple IPDS passing
through O/P channel
one by one).

.. it is the slowest control unit
of the delay of mux &
Decodes but easy to implement

(in the general purpose
register in "cise" computers
using these)

- // general purpose computers are not super computers; it's normal computer // intra normal computers Vertical microprogrammed CU is present
- // in super computers Handwired CU is present.

its default unprogrammed CU

Vertical programming.

Difference b/w horizontal & vertical programming.
Decodes format is 1 bit/cs
of a "CS"

- You take decodes format signal, no. of signal in the H/W equal to no. of bits in the Control field.

So horizontal supports longer
Control field/words

- * In these designs the "control" in these designs, CU are represented in binary
represented in decade binary format.

.. it is the slowest control unit
of the delay of mux &
Decodes but easy to implement

(in the general purpose
register in "cise" computers
using these)

- Its default unprogrammed CU
- encodes form of NCSS's log2Nbit
- a "CS"
- here Decodes format signal.
- No. of signal in the H/W is equal to the no. of log of the bits in the control field.

So vertical supports shorter
Control field.

- * In these designs the "control" in these designs, CU are represented in binary
represented in decade binary format.

CIV is not in use

- no need of the decoder to generate the control signal
- so mux & decs dev latency are not more so it is faster than vertical

here, degree of parallelism can be

↳ This means if we have more than 4 bit control signal



so degree of parallelism means that how many signal can be activated at a time

so it support high degree of parallelism.

- If you enhance the width "C0" to 6bit. (or extra) then you

2 more combination (here less investment less profit)
↳ a bit flexible

- Difficult to implement
- more expensive as more bit are required. so more signal Kaliya more bit

(practically available) used in Computer

- needs the decoders to generate the control signal
- has mux & decoders latency so it is slower than horizontal programming

it have 2 bit control signal

then

00	—	00
01	—	01
10	—	10
11	—	11

XX — don't care state

so only one signal is active at any time for a 2bit "C0" if you don't want then you can put don't care state

so it support low degree of parallelism (none/one)

- if you enhance the width to just by one bit then you will witness more. combination b/c 2ⁿ bit = 2ⁿ so you are have no. of combination now (so less investment more profit)

- more flexible
- easy to implement
- less expensive
- its slow

- In booting process, RAM & ROM are different
 - RAM { main memory } & control memory
 - Inside it → user program is present & that user program is written by threads
 - It is in they control of they user
- so at the time of the execution of the "main memory" part of the program these "ROM" invoke the corresponding program internally
 - It is in ROM
 - Control of the ROM
 - Control of the User
- so user developed source code is present in the main memory & whatever program required for the hardware area is stored in the control memory permanently
- Conclusion
 - in the uprogram, control unit, control memory is present that is ROM (Read only memory), in that ROM all program are permanent
 - there are 2 types of storage RAM & ROM
 - so main memory part of the program address is pointing the program counter (PC) → it is responsible to generate the sequence of the instruction address
- similarly in control memory program are present so, program sequence unit
 - instr sequence is generated by they PC is incremented to point the next how the PC is implemented
 - insn sequence similarly functionality is implemented in sequence unit.

- In the RM 2 registers are associated MAR & MBR such that it goes from "PC" → MAR → RAM → MBR
 - Similarity in control memory there are 2 registers memory addressed registers
 - Similarity in control memory there are 2 registers CAR & CDR
 - control memory
 - Address & data registers
 - So sequence unit generated address is loaded to CAR. So CAR enable register is "ROM". Inside the "ROM" cell in "ROM" M50th is shifted to CDR present. That now CDR information passing to skeleton (Base = 4100) then it shows 0P.

Q) consider a hypothetical CPU which support 8K CPU memory
 1120 contains 120 CS's & 8 branch condition
 what is the size of a CS in bits & CS in bytes using
 vertical programming

Horizontal
 control word

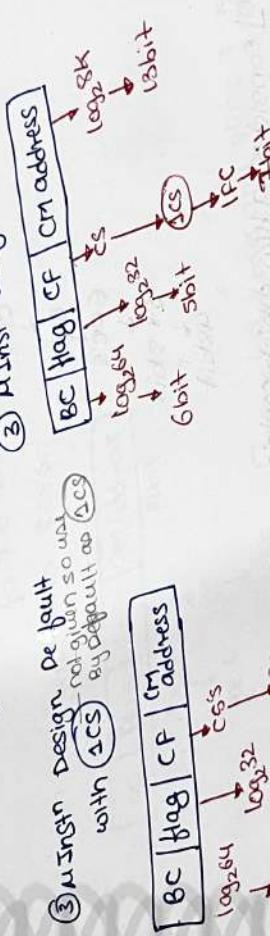
Vertical

① #CS in 1120 = 120

$$\text{② Encoded form} = \log_2 120 \\ \text{of a 'CS'} \\ = 7 \text{ bits/CS}$$

so 120 bits
 for 120 CS's.

③ Minsin design for fault with CS
 not given so we will
 by default use CS



④ Minsin design for fault with CS

BC HAG CF CM Address

loop bit

CS

slot bit

CC

slot bit

Q) Consider a programmed CS support
is designed with 6CS what is the size of a minst?

Sol: Default program. CS is vertical

$$\# \text{CS's in the H/w} = 200 \\ \text{Encoded form of "CS" } = \log_2 200 \\ = 8 \text{ bits/CS} \\ \text{FCY}$$

Minst design with 6CS

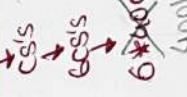
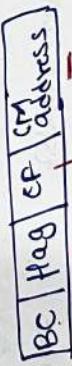


* Q) Consider a H/w which contain
400 CS's designed with a decoded
Binary format. Minst is designed.
With a 6 CS's what is the
size of a minst?

Sol: Horizontal CS
CS's in H/w = 400
Decoded form = 11 bits/CS

of acc.

(400 bit required for)
400 CS



In the 400 bit code
Any 6 bits status
is 1's & rest of the
bit are 0's

* It considers a hypothetical CPU which contains 2000 bits.
 16 flags & 32 branch conditions. When the condition is true, then target address is loading into a 32-bit binary format signal and implemented using 8 CS's.

- M main is designed with
 a) what is the size of a bus in main design when
 b) how many bits are saved in the main design when we use vertical over horizontal?

a) Horizontal CO
 # CS's in HCU = 300
 Decoders form 80 "Cs" = 1 bit / Cs
 300 bit required
 for 300 CS's

M main design with 8 CS's.

BC	Flag	CF	CM ADDRESS	Wait
log ₂ 32	log ₂ 6			CS
				wait
				CS

In 300 bit code any 8 bits
 status is 1's &
 rest of the bits are 0's.

$$\text{HCU size} = (S + 4 + 300 + 1) \text{bit} = 323 \text{bits.}$$



It has 32 bit output. It is justified because
 if first will happen then all the others
 will not happen. So it is
 parallel to the other
 and it is a bus

Not valid

① 111100

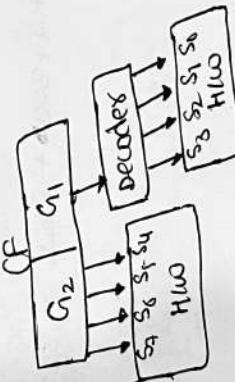
② 110100

③ 110100

④ 101011

With the following cases in the (P)

- When we control one memory unit.
- C_i support vertical & a support horizontal.



Q) Consider the following design used to implement the

control in two groups

Q) Consider the following design used to implement the

$$\therefore \text{VCW} = 172 - (S + 4(8 * 9) + 14) = 95bit$$

$$\# bits save = (3223 - 95) \\ = 2228 bits.$$

8 bit bus

BC	Host	CF	CM Address
10 ₂ 32	10 ₂ 16	C _{CS}	Units

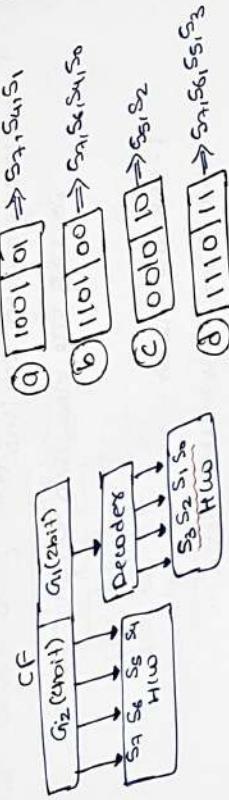
• Main design with CCS's

(PC)

CCS's in H16 = 200

CCS's in H16 = 200

- ⑤ Vertical Cn
CCS's in H16 = 200
• Encoded format = log₂200
= 7 bits



- ⑦ consider a hypothetical CPU which support 2 address buses.
Each instruction takes 10 cycles to compute. Main CPU contains 16 flags & uses encoded format with 32 branch conditions. Branch instruction is designed with 32-bit address format to control the branch logic. What is the size of a CAR & CDR register?

卷之三

卷之三

CSS LRU LRU CLASS

OEH-BO 2008

1

Zabit (Cs)

ECONOMIC
POLICY

ମନ୍ଦିର ପ୍ରେସ୍‌ରୁ

BC Tag CF CM Address

卷之三

卷之三

unit 1

卷之三

三

卷之二

100

Individuals

$$\therefore \text{CM Size} = 2560 \text{ CUs} / \text{Unit}$$

$$\therefore \text{CAR Size} = 1024 \times 2560 = 12 \text{ bit}$$

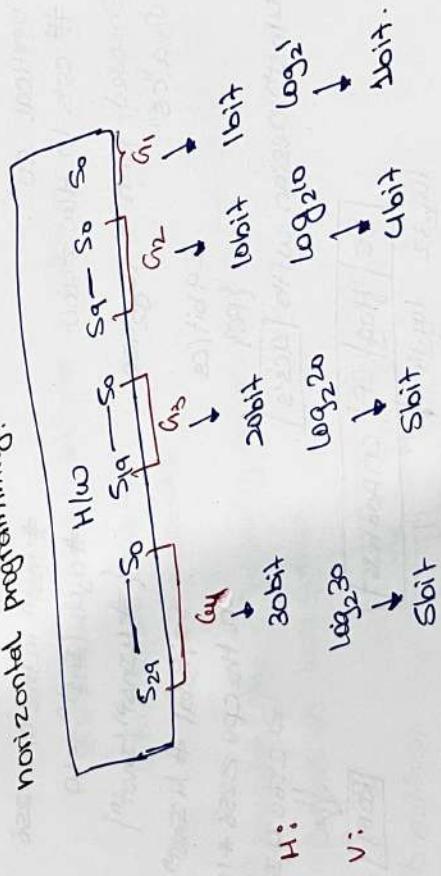
$\therefore \text{CAR Size} = 12 \text{ bit}$
 $\text{CAR Size} = S_4 + (U * q) + 12$
 $= 6 \text{ bit}$. which support 4 groups

Q] consider a hypothetical CU which supports 4 groups
of a mutually exclusive control signal given below

group	G ₁	G ₂	G ₃	G ₄
CS	1	10	20	30

and saved using vertical over

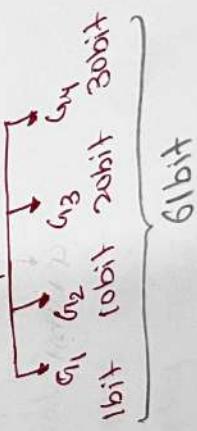
How many bit are required for horizontal programming?



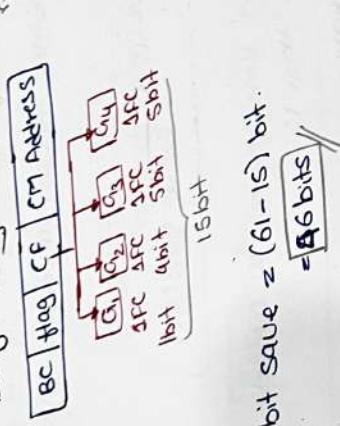
H:
V:

ultram design using Horizontal (max. no. of bytes)

RC	Flag	CP	CM Address
G ₁	G ₂	G ₃	G ₄



Universal design using vertical (min. bit required is 15bit)



$$\# \text{bit save} = (61 - 15) \text{ bit}$$

-**46 bits**

- Performance evaluation of a CPU
- Performance is an indirect measurement, depend on the execution time.

$$\text{Performance (P)} \propto \frac{1}{\text{Execution Time (ET)}}$$

- Execution time means time required to complete the program execution that is.

$$\boxed{\text{Execution time} = \text{no. of second program (CPU time)}}$$

$$\begin{aligned} \# \text{Instr} / \text{Program} * \text{no. of cycle} / \text{Instr} * \text{no. seconds} / \text{cycle} \\ \rightarrow \text{Instn count (IC)} \quad \rightarrow \text{CPU time} \end{aligned}$$

$$\therefore ET = IC * CPU * Cycle Time$$

- In the programs, different instruction are taking different cycle to complete

$$\therefore \boxed{\text{Program Execn} = \sum (\text{IC}_i * \text{CPU}_i) * \text{Cycle Time}}$$

1. Type of Instruction

- Speed up (S) factor is used to measure the performance gain of a CPU by compare the system performance level

12A

System "x" is compared with

System "y"

$$\frac{S = \frac{\text{performance}_x}{\text{performance}_y}}{\frac{ET_x}{ET_y}} = \frac{\frac{1}{2}}{\frac{1}{4}} = \frac{ET_y}{ET_x}$$

Same - constant System "x" runs in hours

- // which ever system performance gain you want to measure
- that system in execution must be in the denominator
- // so denominators system is "y". Here faster than "x".
- numerators

ex.: System X

{Task}

ET_x = 10s

System Y

{Task}

ET_y = 5s

- performance gain of a system "x" is
- System "x" is assume value

$$S = \frac{ET_y}{ET_x} \Rightarrow S = \frac{10}{5} \Rightarrow \boxed{S=2}$$

so, Sys "x" runs 2 times

faster than system "y".

Reasoning Subject:

Ex: Statement: "x" is 2 hours
faster than "y"

$$\text{Coq subject: } S = \frac{y}{x} - \boxed{y=2x}$$

$$ET_{old}$$

$$ET_r$$

Q)

consider 2 cycle cycle CPU which consumes a cycle for Data transfer, 7 cycle for AND & 5 cycles for Branch Instruction. Relative frequency of those instructions are 40%, 30% & 30%, respectively.

$$\text{Sol: Avg. ins. } ET_r = \frac{\text{Time required/prog}}{\# \text{Instrns / prog}}$$

$$= \frac{\sum (C_i * CPS_i) \text{ cycle/time}}{\# \text{instrns}}$$

④ Performance of a CPU in terms of MIPS

$$\begin{aligned} \text{⑤ If system is enhanced with an} \\ \text{avg. CPS of 1. In this process} \\ \text{cycle time is increased upto} \\ \text{0.9ns. what is the performance} \\ \text{gain of a system.} \end{aligned}$$

$$\begin{aligned} \text{Avg. ins.} &= 14.4 \text{ ns} \\ \# \text{Instrns} &\rightarrow 1 \text{ sec} \Rightarrow \frac{1 \text{ sec}}{14.4 \times 10^9 \text{ ns}} \text{ instrns/sec} \\ &\Rightarrow \frac{1}{14.4 \times 10^9} \text{ instrns/sec} \Rightarrow 0.06944 \times 10^9 \text{ instrns/sec} \\ &\Rightarrow \boxed{6.944 \text{ MIPS}} \end{aligned}$$

$$\begin{aligned} \text{Avg. ins.} &= 14.4 \text{ ns} \\ \# \text{Instrns} &\rightarrow 1 \text{ sec} \Rightarrow \frac{1 \text{ sec}}{14.4 \times 10^9 \text{ ns}} \text{ instrns/sec} \\ &\Rightarrow \boxed{6.944 \text{ MIPS}} \end{aligned}$$

$$\textcircled{5} \quad ET_{\text{old}} = 14.4 \text{ units}$$

$$ET_{\text{new}} = \frac{[(0.4 * 1) + (0.3 * 1)] + (0.3 * 2)}{(0.4 * 0.3 + 0.3)}$$

ET_{new} = 2.9 units.

$$S = \frac{ET_{\text{old}}}{ET_{\text{new}}} = \frac{14.4}{2.9} = \boxed{4.96}$$

Amthal's Law

- Amthal laws focus on performance gain of system by make the common case fast & frequently used components modify them
- Common law concentrate on the performance gain of a system
- Amthal law concentrate on the system functionality.

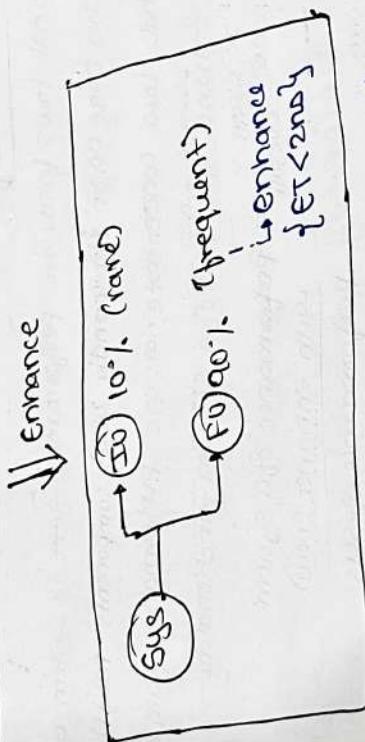
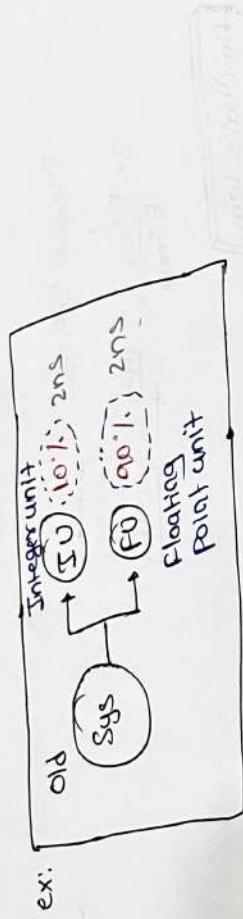
After enhance the part of a system
 i.e. $S_{\text{overall}} = \frac{\text{Performance of a system with enhance part}}{\text{Performance of sys. without enhance part}}$

$$S_{\text{overall}} = \frac{\Delta ET_{\text{new}}}{\Delta ET_{\text{old}}} \quad \text{Directions:} \quad \begin{array}{l} \textcircled{1} \text{ (enhanced part)} \\ \textcircled{2} \text{ (original part)} \end{array}$$

$$S_{\text{overall}} = \frac{ET_{\text{old}}}{ET_{\text{new}}} \quad \textcircled{1}$$

- in the enhancement process only the part of the system is modified so new system contain 2 portion
- enhanced portion
 - enhanced portion

$E\tau_{new} = E\tau$ of unenhanced portion +
 $E\tau_Q$ enhanced portion



to calculate $E\tau_{new}$ two parameters are required

- ① Fraction enhancement (F)
- ② Speed enhancement (S)

indicate that how much part of the fraction enhanced

System is modified

i.e. F : enhanced part
 $(1-F)$: unenhanced part

indicate the performance gain of the enhanced part

$$S = \frac{\text{Performance of new system}}{\text{Performance of old system}}$$

$$S = \frac{ET_{\text{new}}(1-F)}{ET_{\text{old}}(1-F)} \Rightarrow \frac{ET_{\text{new}}}{ET_{\text{old}}} = \frac{(1-F)}{S}$$

$$ET_{\text{new}} = ET_{\text{old}} \frac{(1-F)}{S}$$

Substitute the above value in eq -②

$$\text{i.e. } ET_{\text{new}} = ET_{\text{old}}(1-F) + ET_{\text{old}} \frac{(1-F)}{S}$$

Substitute the ET_{new} in eq -①

$$\text{i.e. } S_{\text{overall}} = \frac{ET_{\text{old}}}{ET_{\text{old}}(1-F) + ET_{\text{old}} \frac{(1-F)}{S}}$$

→ To cover the complete system take the relative data.

$$\text{i.e. } \text{System} = 100\%$$

$$S_{\text{overall}} = \frac{100\%}{(100\% - F) + \frac{F}{S}}$$

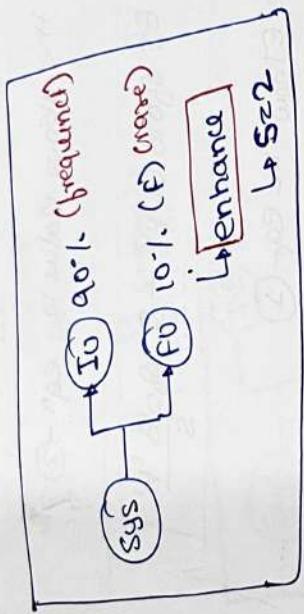
$$S_{\text{overall}} = \left[\frac{1}{(1-F) + \frac{F}{S}} \right] \text{ or } \left[\frac{(1-F) + \frac{F}{S}}{1} \right]$$

- when the system contain multiple enhancement then performance gain is calculate as

$$S_{\text{overall}} = \left[(1 - \sum F_i) + \sum \frac{F_i}{S_i} \right]^{-1}$$

i.e. # enhancements

Q] consider a hypothetical system used in the scientific Appr domain. Appr program refers the integers & floating point unit during its execution. Floating point unit is enhanced then it runs 2 times ~~faster~~^{speed up} but only the 10% of instr are floating point. what is the speed up



$$f = 10^{-1} \cdot \begin{cases} \text{Amplify} \\ S = 2 \end{cases} \rightarrow S_{\text{overall}} = ?$$

state that performance is improved by enhance law.

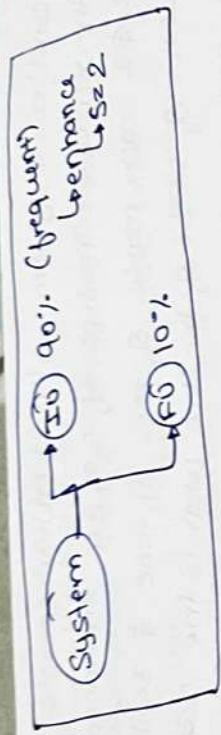
The frequency used component not rarely used component.

$$S_{\text{overall}} = \left[(1 - 0.1) + 0.1 \right]^{-1}$$

$$= \boxed{1.05}$$

So here the enhancement is done on the rare component so the enhanced component is negligible \rightarrow performance gain

∴ correct enhancement is frequently used component modification



$$F = 90\% ; S = 2$$

$S_{overall} = ?$

$$S_{overall} = \left[(1-F) + \frac{F}{S} \right]^{-1} = \left[(1-0.90) + \frac{0.90}{2} \right]^{-1} = \boxed{1.81}$$

Q1 consider a cache memory which is 13 times faster than main memory & it is referred by the CPU 63%. How much performance do we gain by using this cache? Here enhancement is introduction of cache memory into computer

$S = 13$	Amthal's laws
$F = 63\%$	
$S_{overall} = ?$	

$$S_{overall} = \left[(1-F) + \frac{F}{S} \right]^{-1} = \left[(1-0.63) + \frac{0.63}{13} \right]^{-1} = \boxed{2.39}$$

- Q1 Consider a system with enhancements having the respective spreads of s_1, s_2, s_3 .
 ▷ If each usable of 35%. So how & will be unusable of 25% of the time. what is the overall spread

$$F_1: 35\%, \quad S_1 = 5 \quad \text{Overall} = ?$$

$$F_2: 35\%, \quad S_2 = 8 \\ F_3: 25\%, \quad S_3 = 4$$

$$\text{Overall} = \sqrt{1 - (F_1 + F_2 + F_3)} + \frac{F_1}{S_1} + \frac{F_2}{S_2} + \frac{F_3}{S_3}$$

$$\text{So, } \sqrt{1 - (0.35 + 0.35 + 0.25)} + \frac{0.35}{8} + \frac{0.35}{4} + \frac{0.25}{5} \\ \Rightarrow [0.05 + 0.07 + 0.04375 + 0.0625]^{-1}$$

$$\Rightarrow [0.22625]^{-1} \\ \Rightarrow 4.41988$$

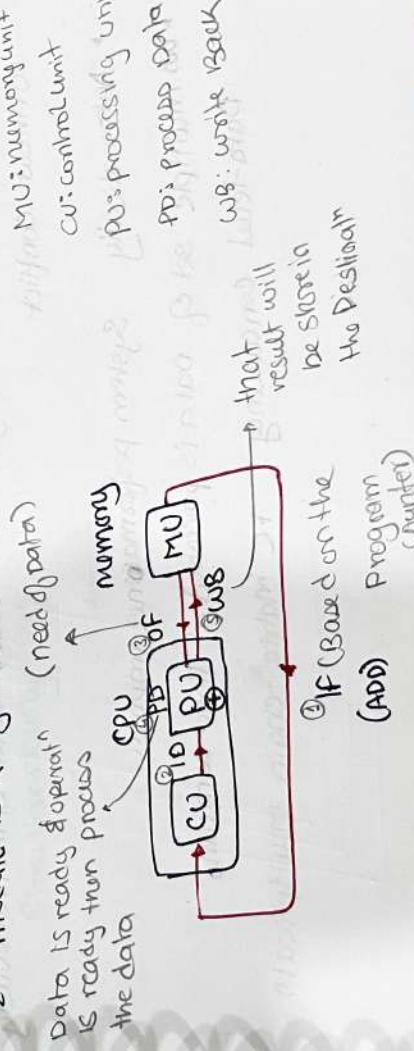
- High performance CPU design
- High performance Processors exhibit the concurrency
- Concurrency means 2 or more instruction execution at a time
- According to Flynn's classification, computers

- is of 4 kind
1. SISD [single instruction & single data stream]
 2. SIMD [multiple " & multiple "]
 3. MISD [multiple " & single "]
 4. MIMD [multiple " & multiple "]

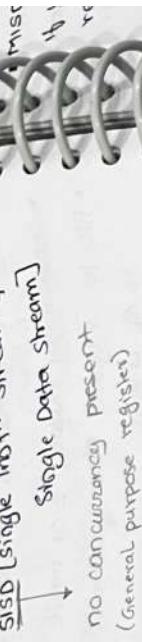
Short cut

- CU : control unit
- PU : processing unit
- MU : memory unit
- IS : Instruction Stream
- DS : Data Stream.

- In 2nd module whenever the functionality we learn about computer for instruction cycle, what we realise is that CPU execute the one instruction after another so concurrency is not there when system doesn't satisfy the concurrency that is called as general purpose computer. So what ever we discuss in the 2nd module is a general purpose computer.



- ∴ Only one instruction is running at a time, therefore there
- Architecture is known as SISD [Single Instruction Stream & Single Data Stream]



SIMD (Single Instruction multiple Data Stream)

- If you want to process multiple sets of data then you need extra hardware in the computer.

↳ means multiple ALU are needed, all the ALU are controlled by single control unit.

- So all one instruction is decided by the control unit that one instruction is distributed to all the ALU so all the ALU are ready to perform the same operation but not on the same data, on different - different set of data processed on the same operation provided multiplex memory.
- If single port memory is there then conflict is there, so when you introduce multiplex ALU to access the memory, when multiple set of data then multiplex is required. When multiplex memory is there then you can access data simultaneously without conflict

- Automatically system performance improves
- multiple set of data is processed at time ~~at same time~~
- Data-level concurrency i.e. matrix-chain multiplication

- MISO (multiple instr stream & single data stream)
 - ↳ If you want to process multiple instr then multiple brain required, multiple brain means multiple control unit required & cu present in the cos . if you want multiple required & cu present in the cos then system contain multiple cpu [multiple cu & hu].
 - ↳ All the cpu are sharing a common memory.
 - ↳ (single port memory), multiple cpu are present but only one get the access of memory & remaining 3 cpu are waiting.
 - ↳ So system physically have 3 cpu but logical one cpu is In the processing (operation)
 - ∴ look wise multi processor computer but behaviour wise single processor computer.
- Practical not possible
- SIMD (multiple instr stream & multiple data stream)
 - ↳ To use multiple cpu we provide multiple memory then multiple cpu are accessing their own port of memory.
 - ∴ So without conflict they may access simultaneously

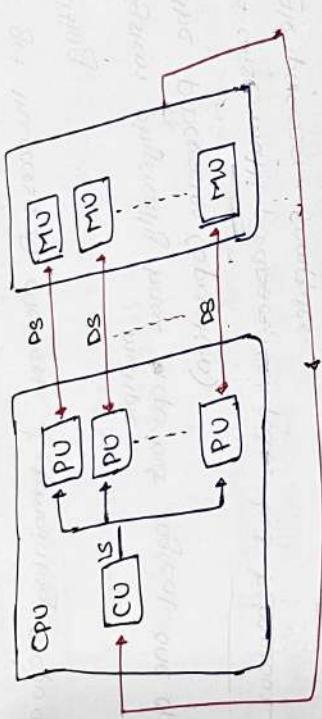
SISD [no concurrency]

Implemented as an uniprocessor system
e.g.: 8085, 8086



not in use b/c
X SISD [data level concurrency]

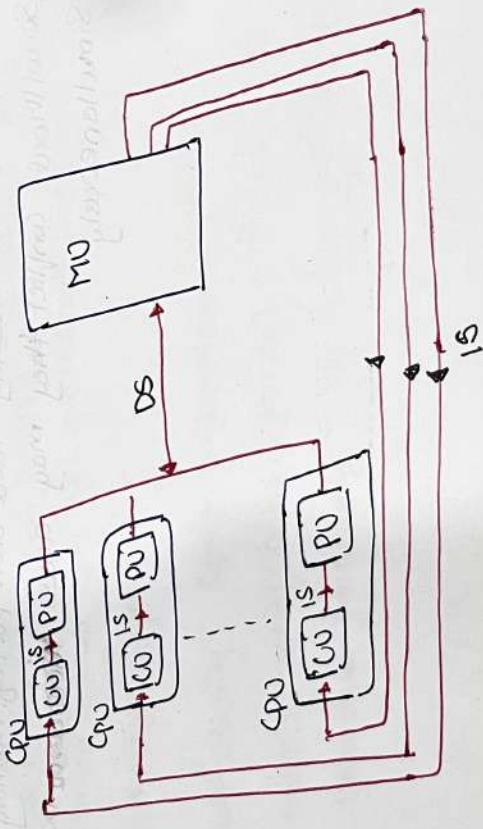
Implemented as used in super computers
Supr. appln not in personal appln



Implemented as an array
(vector) processor

ex: PEPE, Processor
System Processor

MISD [not in use]



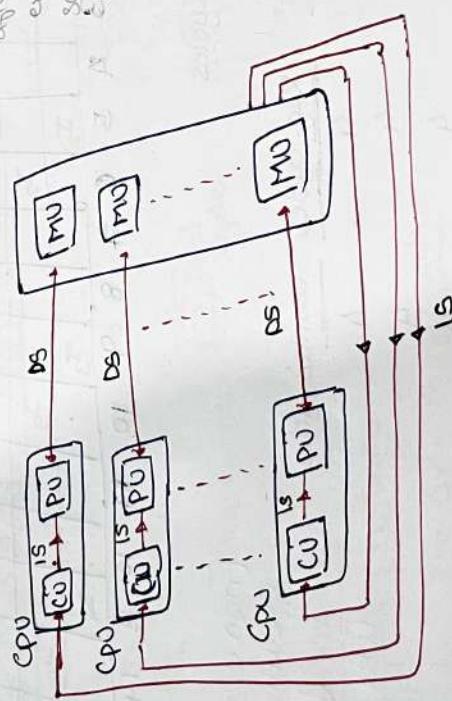
|| ॥ नमामि तेजो मूर्खा अक्षया धूम विदुष, प्रभु, रुद्र, शशि
नुलीपि विदुष एव अक्षया धूम विदुष, प्रभु, रुद्र, शशि

↳ means instr format change means multiple opcode area design in the same instr that become a very long instr word [VLIW] . so within 1 instruction opcode area design . once the instr is fetched from the memory later opcode distributed internal . when opcode distributed then different action take place that means VLIW

\therefore this nich contain multiprocessors but only one processor is in the use at a time.

This much is not yet implemented

X nation wide
X mind [instn level concurrency]



∴ This Arch.
 is implemented as a multiprocessor sys.
Ex: Cray processor
 cyber processor.

- Using pipelining we are solving the issue of ~~concurrency~~ [no concurrency] without pipeline
- ~~ISSD~~ [no concurrency] & without pipeline

Objectives : Instruction Execution

Execution : IF, ID, EX, WB
 Status : IF, ID, EX, WB
 Registers
 mem. address

Execution sequence of instruction (I_1, I_2, I_3, I_4)
 Assuming instrns are fixed length equal to the length
 of CPU word.

So second instrn, attempt to fetch notes na
 & reading hana ka bala execution na
 & reading completed



[Non overlapping
 execution]

→ Cycles
 → Cycles

Instn	Reg. cycles	Counted cycles
I_1	4	4
I_2	4	4
I_3	4	4
I_4	4	4

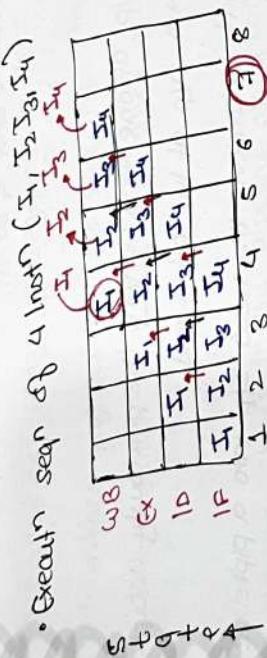
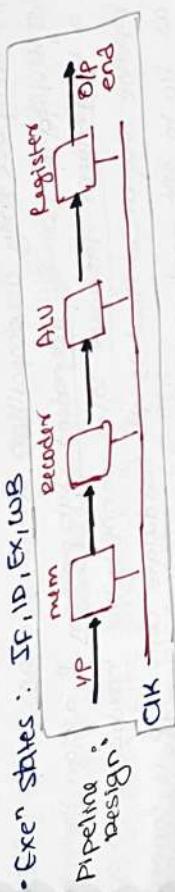
$$ET = 16 \text{ cycles.}$$

(Execution Time)

$ET_{\text{non-pipeline}} = 16 \text{ cycles}$

178

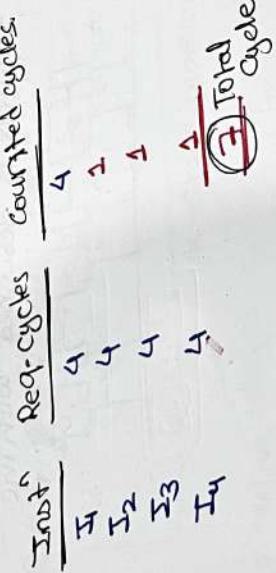
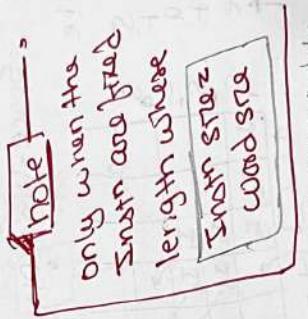
- Also with pipeline
- Objectives: Task Execution
- Core stages: IF, ID, EX, WB
- Pipeline design:
 - Pipelining
 - overlapping execution



$n = 0 + 0 + 4$

- I_2 latency cycles
- of I_4 uses extra cycle
- extra cycle

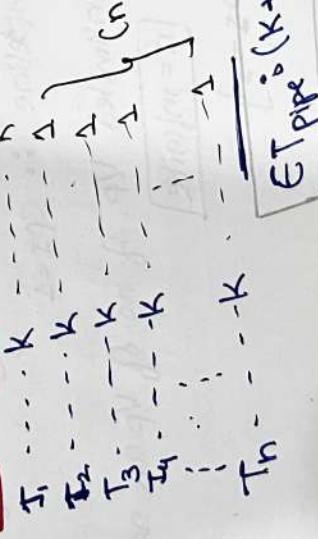
- Pipelining
- concentrate on SLP not WOP



Generalization:

- K-stages, n-tasks

$$\text{Task Req. Cycles} = \frac{\text{Counted cycles}}{K}$$



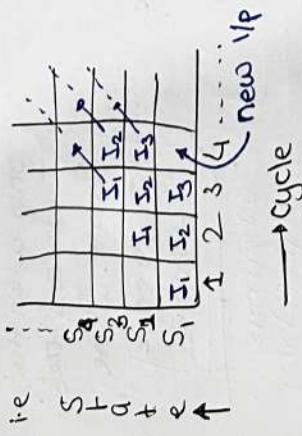
Pipelining

Function:

- Pipelining uses the decomposition technique means problem statement is divided into set of independent sub problems & assign them to a independent hardware units connect the hardware in a pipeline sequence that is 1st hardware unit OIP is connected to 2nd hardware unit & so on.

Definition:

- Accepting the new VP at one end before the previously accepted VP is appears as an OIP at the other end
- Definition states that, insert the new VP into a pipeline before completion of a OIP VP. So new VP over execution with old VP called as overlapping execution
- Overlapping execution sequence is described with the Space-time diagram.



- Successful characteristic of a pipeline is, in every new cycle new VP must be inserted into pipeline $\therefore \text{CPI} = 1$

- Pipeline is designed for unlimited VP. If no. of VP are not given then assume as $n = \text{infinite}$

$$\text{So, } \frac{1}{\text{CPI}} = \frac{1}{1} = 1$$

18A

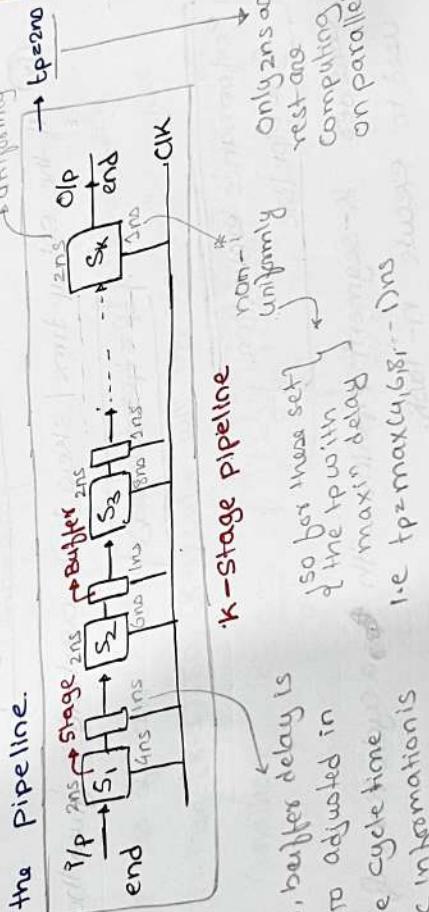
Design

1. Pipelines has two end called as Input & Output. At these end, multiple pipe are interconnected to satisfies the functionality of a pipeline. Each pipe in the pipeline is called as stage or segment.
 2. Interface register is used b/w intermediate O/P. It is also called

四〇二

Non overlapping boundaries are called as Old or called as ~~old~~ the complete boundary of a region is called outline.

- 3) all the stages in the pipeline along with a buffer is connected to a common clock. so clock adjustment is very important in the pipeline.



Side so Buttered
moulding through
Butter the shape to

is also considered

[Together max no. individuals max]

$$\Delta p = \rho g + \rho gh_{\text{max}}$$

卷之三

Note

- Cycle time of the pipeline is adjusted Nc to the stage delay + buffer delay in the pipeline.
- ① If uniform delay pipeline given then

$$\boxed{Cycle\ time = t_p = \text{stage delay}}$$

- ② If nonuniform delay pipeline given then

$$\boxed{t_p = \max(\text{stage delay})}$$

- ③ If buffer delay present in the pipeline then

$$\boxed{t_p = \max(\text{stage} + \text{buffer delay})}$$

- ④ If the set up time / skew time overhead is given then

$$\boxed{t_p = t_p + \text{overhead}}$$

Performance evaluation of a pipeline

- considers K-segment pipeline with a cycle time of t_p , used to execute n-Task.
- In the Pipeline, very 1st task is execute in a non-overlapping order. So it take K-cycle to complete. So the remaining $(n-1)$ emerged from the pipe at the rate of 1-task per cycle so $(n-1)$ task take $(n-1)$ cycle to complete.
- ∴ total time required to complete n-task in a K-segment Pipeline is

$$ET_{\text{Pipe}} = (K+n-1) \text{cycle}$$
$$ET_{\text{Pipe}} \Rightarrow (K+n-1)t_p$$

- consider a non-pipeline system used to execute n-task in which each task take " t_n " time to complete the execution therefore total time required to complete n-task in a non-pipeline system is $ET_{non-pipe} = n \cdot t_n$; t_n = one task ET in non-pipeline.

- performance gain of a pipeline is,

$$S = \frac{\text{Performance}_{\text{pipe}}}{\text{Performance}_{\text{non-pipe}}} \Rightarrow \frac{\frac{1}{ET_{\text{pipe}}}}{\frac{1}{ET_{\text{non-pipe}}}}$$

$$S = \frac{ET_{\text{non-pipe}}}{ET_{\text{pipe}}}$$

$$\Rightarrow S = \frac{n \cdot t_n}{(K+n-1)t_p}; n \rightarrow \infty$$

- when no. of task are increased than "n" becomes much larger than $(K-1)$. so, $(K-1)$ value will approach to "n". under these condition.

$$S = \frac{n \cdot t_n}{n \cdot t_p} \Rightarrow S = \frac{t_n}{t_p}$$

one task ET
cycle time

- when the pipeline stage are perfectly balanced then one task execution time in they non-pipeline is also equal to no. of stages in they pipeline under this condition; $S \approx \frac{K \cdot t_p}{K+1}$

$$\begin{aligned} i.e. t_n &= K \cdot t_p \\ &= K \cdot t_p \end{aligned}$$

$$\begin{aligned} S &\approx K ; n \approx 100 \\ \# \text{ stages} &\downarrow \\ \text{Pipeline depth} & \end{aligned}$$

- when the system is operated with 100% efficiency then max. speedup is possible that is also equal to no. of stages in the pipeline

$$\eta = 100\% \quad \frac{S_{\text{max}}}{S} \quad S_{\text{?}}$$

$$\eta_{\text{pipe}} = \frac{S}{S_{\text{max}}} = \frac{C}{K}$$

throughput of a pipeline

throughput pipe = # tasks processed / total time taken to process the tasks of work done

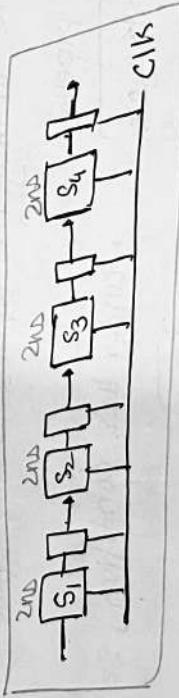
$$= \frac{n}{(k+n-1)t_p}$$

Ex: $T_{\text{pipe}} = 8/\text{tions}$

"3" Tasks are completed in a "tions" time unit

? Note

1. when the pipeline stage's are perfectly balanced [uniform delay] than one task execution time in the pipeline is also equal to one tasks execution time in the non-pipeline.



① 1-task execution time (ET) in Pipeline

$$ET_{\text{Pipe}} = (K+n-1)t_p + \left\{ \begin{array}{l} K=4 \\ t_p=2 \text{ ns} \\ n=1 (\text{finite}) \end{array} \right\}$$

$$\Rightarrow (4+1-1)2 \text{ ns}$$

$$\Rightarrow 8 \text{ ns}$$

② 1-Task ET in non pipeline (t_n)

$$t_n = S_1 + S_2 + S_3 + S_4$$

$$\approx (2+2+2+2) \text{ ns}$$

$$\Rightarrow 8 \text{ ns} \quad \{ = K+t_p \}$$

$$S = \frac{t_n}{t_p} \rightarrow 8$$

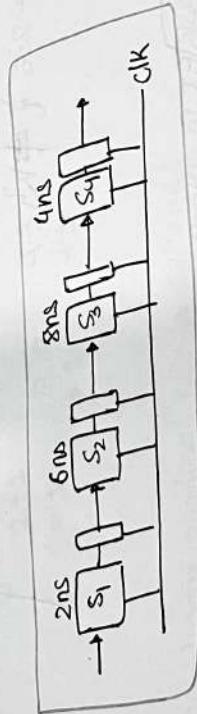
$$S = \frac{8}{2}$$

$$n = \frac{S}{K}$$

$$n = \frac{4}{2}$$

$$n = 1 \quad \{ = \text{finite} \}$$

2. when the Pipeline stages are not perfectly balanced in [non-uniform delay] then one task ET in they pipeline is always greater than one task execution time in non-pipeline.



① 1-task ET in Pipeline

$$K=4$$

$$t_p = \max(2, 6, 8, 4)$$

$$= 8 \text{ ns}.$$

$$n = 1 \quad \{ \text{finite} \}$$

$$ET_{\text{Pipe}} = (K+n-1)t_p$$

$$= (4+1-1)8 \text{ ns}$$

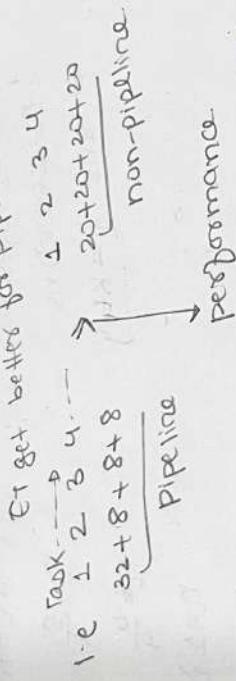
$$= 8 \times 8 \text{ ns.}$$

(2) 1-task ET in non-pipeline (t_n)

$$\begin{aligned} & \Rightarrow t_n = S_1 + S_2 + S_3 + S_4 \\ & \Rightarrow 2+6+8+4 \Rightarrow \boxed{20\text{ns}} \quad \{ \neq K_p \} \\ & \Rightarrow \eta \neq 100\% \end{aligned}$$

due to **Task**
ET pipeline
 \leftarrow ET non-pipeline

But as the task no. increase the



performance

$$\begin{aligned} & \eta = \frac{s}{k} \Rightarrow \frac{2 \cdot s}{4} \\ & \eta = 62.5\%. \quad \boxed{\downarrow} \\ & S = \frac{20\text{ns}}{8\text{ns}} \\ & S = 2 \cdot s \quad \{ \neq K_p \} \end{aligned}$$

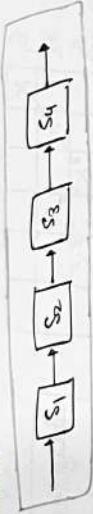
3. Let,
 'T₁' is **one task** in pipeline
 'T₂' is **one task** in non-pipeline

thus relation is

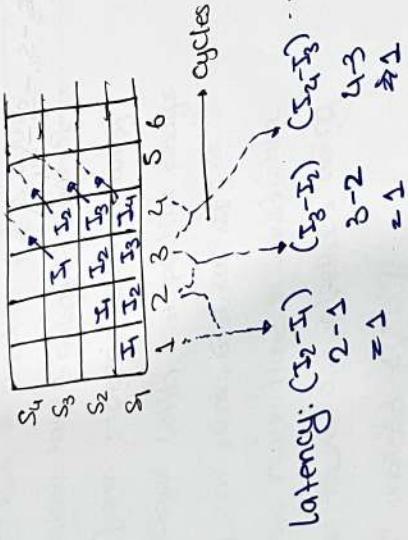
$$\boxed{T_1 \geq T_2}$$

Type of Pipelines

- ① Linear Pipeline
 - These pipeline contain only the feed forward connection.
 - Linear Pipeline are synchronous pipeline
 - Latency of a linear pipeline is always "1"
 - Latency means CLK cycle difference b/w 2 successive initiation. in the pipeline

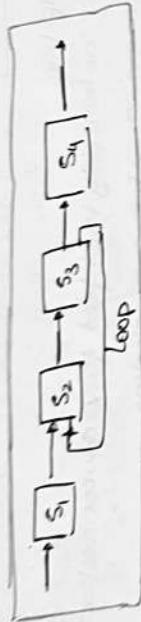


Exctrn path: $S_1 - S_2 - S_3 - S_4$

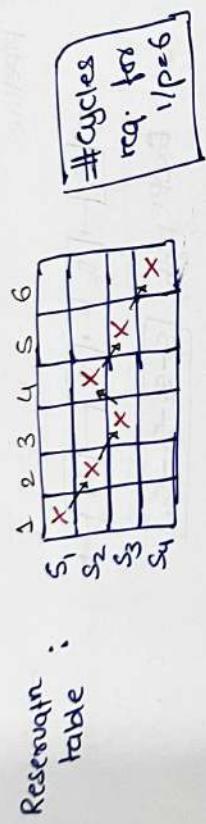


- ② non-linear Pipeline [not in syllabus]

- These pipeline contain both forward & backward connection
- Non-linear pipeline are asynchronous pipeline
- In these pipeline reservation table is required to proceed they IP.
- So latency of the pipeline is depend on they reservation table



Execution path : $S_1 - S_2 - S_3 - ?$



Exeⁿ:
Path : $S_1 - S_2 - S_3 - S_4 - S_5$



③ 1000 ms logic
dimensions 100x100 \rightarrow 1000000 pixels \rightarrow 1000000 bits
Writing 2 monolithic DRAM \rightarrow 1000000 bits \rightarrow 1000000 bytes
Will need to connect 1000000 bits to 1000000 bytes

Question

- A. Consider an instruction pipeline which has a speed up factor 14 while operating with 75% efficiency. what could be the no. of stages in the pipeline?

$$\text{Sol: } S = K \cdot n \Rightarrow n = \frac{S}{K} = \frac{14}{0.75} = \frac{112}{3} = 37.33 \Rightarrow K = \frac{14}{n} = \frac{14}{37.33} = 0.375 \quad \begin{matrix} \text{next stage} \\ \text{requires} \end{matrix}$$

K=20

- 2) Consider 2 pipeline A & B where Pipeline 'A' is having 9 stage of uniform pipeline 'B' is having 5 stage with a respective delays of 2ns, 14ns, 6ns & 8ns.

How much time is saved when 200 tasks are pipelined using "A" instead of "B"?

$$\begin{array}{l} \text{Pipe A} \\ K=9 \\ t_{fp}=3\text{ns} \\ n=200(\text{finite}) \end{array}$$

$$\begin{aligned} ET_A &= (K+n-1)t_{fp} \\ &= (9+200-1)3\text{ns} \\ &= 624\text{ns} \end{aligned}$$

Pipe B.

$$\begin{array}{l} K=5 \\ t_{fp}=\max(2, 4, 6, 8, 1) \\ = 8\text{ns.} \end{array}$$

$$n=200(\text{finite})$$

$$ET_B = (K+n-1)t_{fp}$$

$$= (5+200-1)8\text{ns}$$

$$= 1632\text{ns}$$

$$\therefore \text{Time saved} = (1632\text{ns} - 624\text{ns})$$

$$\Rightarrow 1008\text{ns.}$$

Q) Consider 4ns CLK cycle non-pipeline CPU execute the program which contains 400 instructions with an Avg. CPS of 5. Same program is executed in the 6-stage pipeline with a respective delay of 2ns, 3ns, 1ns, 1ns, 6ns & 4ns. What is the performance gain of a pipeline.

Pipeline.

Sol: non-pipeline:

$$\begin{aligned} t_m &= \text{one task ET} & k &= 6 \\ &= CPS * \text{cycle time} & t_p &= \max(2, 3, 1, 6, 1) \\ &= 5 * 4ns & &= 6ns \\ &= 20ns. & n &= 400 (\text{instructions}) \\ & & & \\ n &= 400 & & \end{aligned}$$

$$S = \frac{n \cdot t_m}{(k + n - 1)t_p}; n \rightarrow \text{finite}$$

Performance gain

$$S = \frac{400 * 20ns}{(6 + 400 - 1)6ns} = 3.29$$

$$S = 3.29$$

Q) Consider 4-stage pipeline with a uniform delay of 1ns used to execute the program which contains 600 instructions. What is the program execution time? Also calculate performance gain of a pipeline when program is executed in a non-pipelined.

18B

Sol: Pipeline [Program ET_{PIPE}]

$$K=4$$

$$t_P = 8 \text{ ns}$$

$$n = 600 \text{ (front)}$$

non-pipeline : [Program ET_{PIPE}]

$$ET_{\text{non-pipe}} = n \cdot t_n$$

for one task ET in non-pipe

$$ET_{\text{pipe}} = (K+n-1)t_P$$

$$= (4+600-1)8 \text{ ns}$$

$$= 4824 \text{ ns}$$

$$\Rightarrow ET_{\text{non-pipe}} = 600 \cdot 32 \text{ ns}$$

$$= 19200 \text{ ns}$$

$$S = \frac{ET_{\text{non-pipe}}}{ET_{\text{PIPE}}} \Rightarrow \frac{19200}{4824} = \boxed{3.98}$$

[5] consider 4-stage pipeline with a respective delays of 20ns, 40ns, 30ns & 50ns. what is the speed up & efficiency of a pipeline?

So:

$$K=4$$

$$t_P = \max(20, 40, 30, 50)$$

$$= 50 \text{ ns}$$

$$n = \text{not given}$$

$$S = \frac{t_n}{t_P} = \frac{140}{50} = 2.8$$

$$S = 2.8$$

assume as 'x'

$$S = \frac{n}{K} = \frac{2.8}{4}$$

$$\eta = 70\%$$

- Q) consider 4-stage pipeline with a respectively delays of 4ns, 2ns, 5ns, & 6ns. Interface registers used the stage have a delay of 2ns. what is the performance gain of pipeline when very large no. of IP are processed?

So:

student mistake

$$K=4 \\ t_p = \max(4 \text{ ns}, 2 \text{ ns}, 5 \text{ ns}, 6 \text{ ns}) \\ 6 \text{ ns}$$

no buffers at the end of last stage.

$\Rightarrow 6 \text{ ns}$

$n = \text{not given (very large)}$



Assume as "∞"

→ *number of stages*

no need of the
buffer b/c
non-overlapping
even

$$\left[\frac{S_1 t_n}{t_p} \right]$$

$$S = \frac{t_n}{t_p} = \frac{120}{60} \Rightarrow S = 2.833$$

- Q) Consider 4-stage pipeline with a respective delays of 6ns, 10ns, 20ns, 30ns. Interface registers used the stages & also connected at the end of last stage contain the delay of 4ns. what is the speedup of a pipeline?

Sol:

$$K=4$$

$$t_p = \max\left(\frac{60+4}{64}, \frac{10+4}{14}, \frac{30+4}{34}, \frac{70+4}{74}\right)$$

$$\Rightarrow t_{pns}$$

n not given \rightarrow assume $n = \infty$ \rightarrow $\boxed{S = \frac{t_p}{t_p}} ; n \rightarrow \infty$

$$t_n = S_1 + S_2 + S_3 + S_4 \\ = 60 + 10 + 30 + 70 \\ = 140 \text{ ns}$$

$$\Rightarrow S = \frac{140}{74}$$

$$\boxed{S = 2.29}$$

Q1. Consider 3ns沉 cycle non-pipeline processes which consume

8 cycles for data transfers. A cycle time of 4 cycles for Jmp instruction. Relative frequencies of these transf are 35%, 40% & 25%. respectively. System is enhanced with a pipeline, in the process 0.8ns setup overhead is present in the system. What is performance gain of enhanced system.

Sol: Standard formula

$$Et_{prog} = \sum (Tc_i * Cps_i) \text{ cycle time}$$

non-pipe:

$$Et_{non-pipe} = \sum (Tc_i * Cps_i) \text{ cycle time} \\ \Rightarrow \left[(0.35 * 8) + (0.4 * 7) + (0.25 * 5) \right] \text{ ans.} \\ = (2.8 + 2.8 + 1.25) \text{ ns} \\ \Rightarrow 20.55 \text{ ns.}$$

Pipe:

$$\left\{ \begin{array}{l} \text{ET Pipe} = \overline{[0.25 * 2]} + (0.4 * 2) + (0.25 * 2) \\ \text{ETI} = 12 \end{array} \right\}$$

$$\Rightarrow 3.8 \text{ ns}$$

$$S = \frac{\text{ET non-pipe}}{\text{ET pipe}} \Rightarrow \frac{20.55}{3.8}$$

S=5.4

- Q Consider 4-stage pipeline where different main and spending different cycles at different stage given below.

	S ₁	S ₂	S ₃	S ₄	
T ₁	1	3	2	1	= 7
T ₂	1	1	3	1	= 6
T ₃	2	1	1	2	= 6
T ₄	1	1	3	1	= $\frac{6}{25}$ cycle

(15)

- Q How many cycles are required to complete the instruction

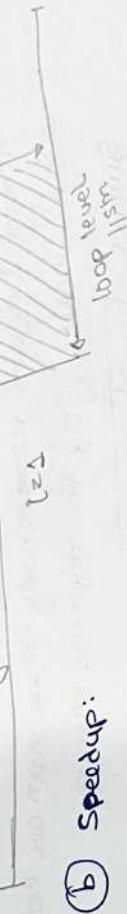
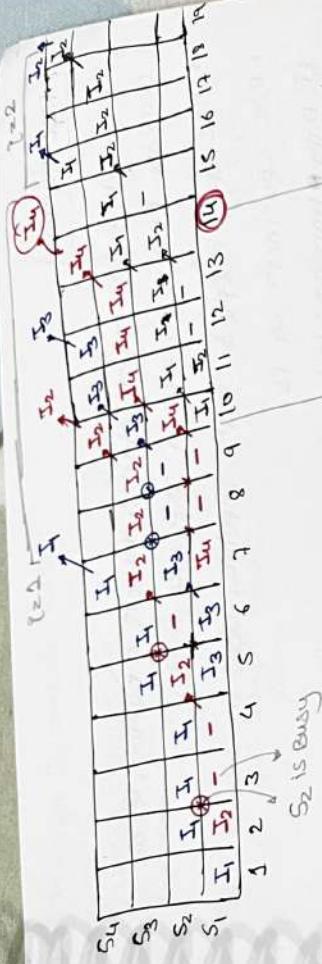
B What is the speed up (1.78)

- C Following loop is executed in the Pipeline

for (i=1; i<=1000; i++) {

S₁, S₂, S₃, T₄}

Op of "S₂" will be available by (i=2)
after _____ cycle.



⑥ Speedup:

$$S = \frac{ET_{\text{non-pipe}}}{ET_{\text{pipe}}} \Rightarrow S = \frac{25}{14}$$

S21.78

⑦

without loop level lsm means start the 2nd iteration after the completion of a 1st instruction

with loop level lsm means start the 2nd iteration before Completion of a 1st iteration.

Note: Default is with loop level lsm.

→ If option does not match
then not without
loop level lsm

RISC Pipeline

- To analyse the implementation issue of a pipeline, let us consider RISC pipeline as a reference model
- RISC CPU supports instruction pipeline, used to execute instruction
- RISC CPU instrn set is
 - ① Data transfer: In the RISC CPU load & store instrns are used to data transfer instrn, to transfer the data between memory & registers respectively. These instrns are designed with a indexed AM, to refer the memory

Syntax: $\boxed{\text{Load } | r_0 | 3(r_1)}$



$r_0 \leftarrow m[3+r_1]$

$\boxed{\text{Store } | 3(r_1) | r_0}$



$m[3+r_1] \leftarrow r_0$

(2) Data manipulation instrn: In the RISC CPU, ALU operation are performed only on a registered data (Registers - Registers ref. CPU)

Syntax: $\boxed{\text{Add } | r_0 | r_1 | r_2}$



$r_0 \leftarrow r_1 + r_2$

(3) Transfer of control instr: In 2nd module

(4) Unconditional Jump

Syntax: $\boxed{\text{jmp}} \boxed{2000}$

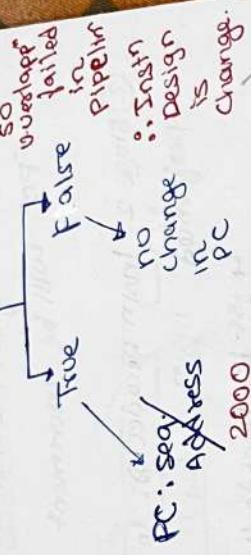
PC: ~~seq/~~ ~~2000~~

(5) Condition jump.

Syntax: $\boxed{\text{jnz}} \boxed{2000}$

PC: ~~seq/~~ ~~2000~~

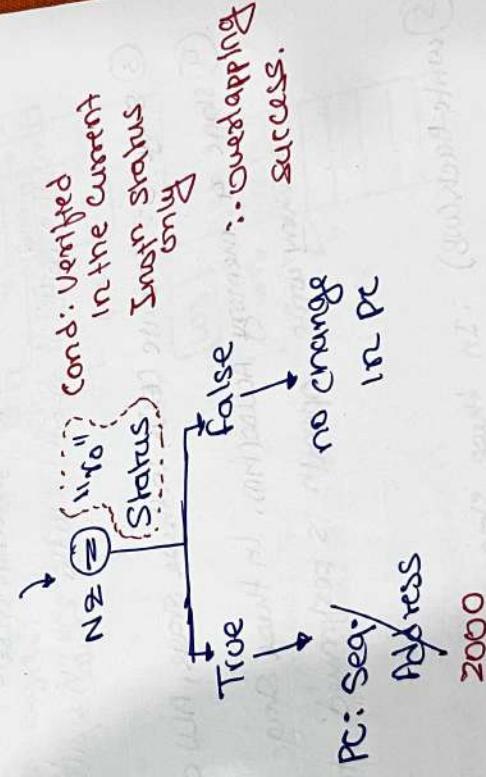
In 2nd module



In 3rd module:

Syntax: $\boxed{\text{djnz}} \boxed{r0} \boxed{2000}$; decrement sum if $\boxed{\text{Nz}}$

Dec r0



2000

(3)

(4)

(5)

- To execute the above instruction set, 5-stage pipeline is used in the RISC CPU called as 5-stage pipeline.
 - Different stages present in the pipeline are follows
- Stage 1: Instruction fetch (IF): In this stage, CPU determine address of the memory based on the "PC" simultaneously.
 - Instruction from the memory based on the sequential address.
 - "PC" will be increment to next sequential address.

② Stage 2: Instruction decoding. In this stage 2 actions are performed

(Instruction Access)

- ↳ 1st is Decode the instruction
- ↳ 2nd is Operand fetch from the register file.

Comparators (Ckt's)

- This stage also contains comparators (Ckt's), to evaluate the Branch instruction.
- Condition will be updated in this 2nd stage also.

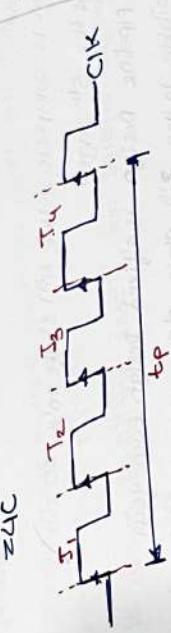
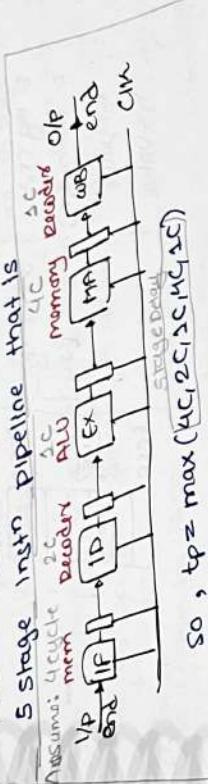
③ Stage 3: Execute (EX): In this stage, ALU operation is performed

- ④ Stage 4: Memory Access (MA): In this stage memory read & memory write operation is performed to access the data.

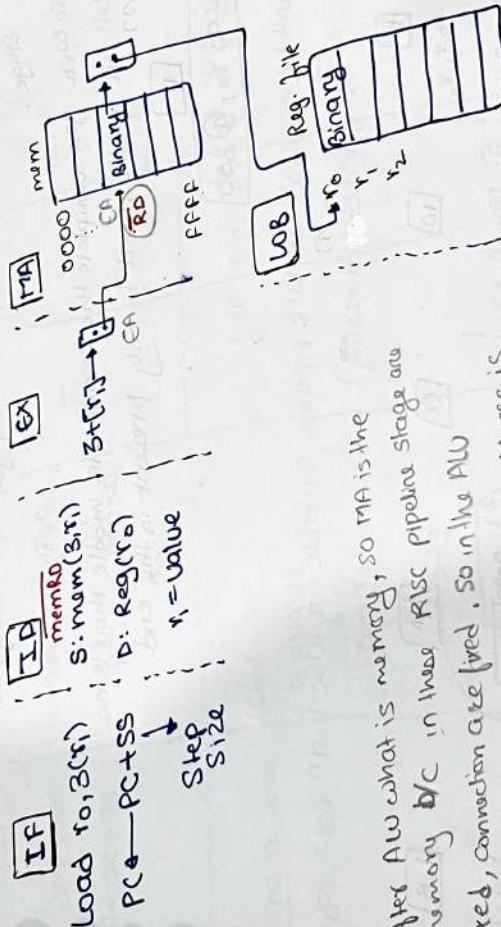
(Data Access)

- ⑤ Write Back (WB): In this stage registered write operation is performed

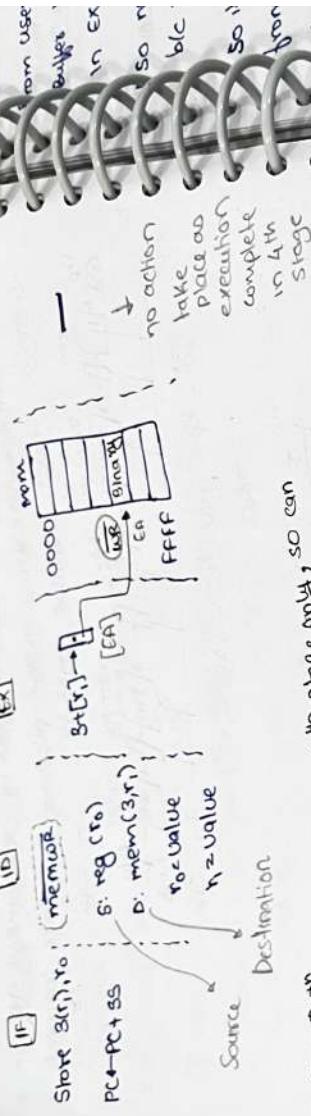
- above stage operan are assigned to independent H/w later connect the H/w in the pipeline



5-stage instrn pipeline.



- After AW what is memory, so MA is the memory b/c in these RISC pipeline stage are fixed, connection are fixed. So in the PLU effective address is calculated. that address is to read the Data or write the Data.
- when instrn is load then read the data when instrn is store then write the data

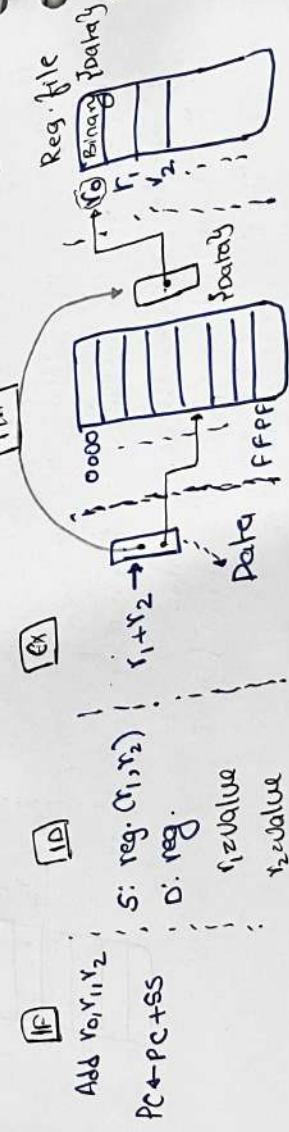


Store Instn
Execution is complete in the 4th stage only, so can we remove the 5th stage?
• Pipeline Design is a fixed design even your no, b/c pipeline is available in the 4th stage still you are moving op's available in the 4th stage b/c of the pipeline linear connection to all the stages b/c of the pipeline linear connection once you enles pipeline then only you exit through last stage.

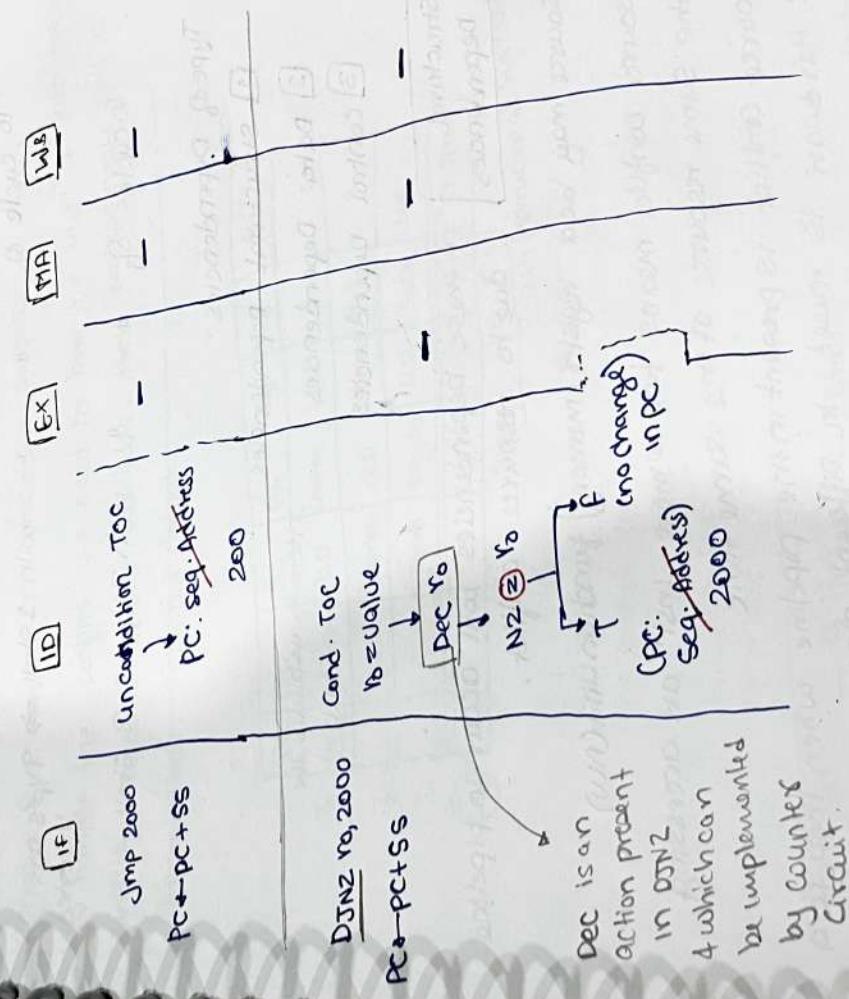
- so even you complete the execution in middle there is no exit in middle, exit is only present in the end



Can't be executed as RISC Pipeline suitable for RISC Instn not for all [due to indirect addressing]



- From user point of view, data will be transfer from ALU buffers to MA buffer. So whatever the value present in EX buffer that will be transfer from MA Buffer. [Internally]
- so no need of memory b/c of memory operation not required b/c its an ALU operation. so these memory is not involved
- so if memory is not involve then can't remove these memory b/c it is a fixed pipeline.
- so pipeline. NO, we can't remove b/c it is a fixed pipeline.
- b/c if we remove these memory then we can't perform Load & store in they Pipeline.
- so pipeline is design for all the instr, to satisfy all the want not individual instr



Imp:

Dependencies in the Pipeline

- [1] major problem in the pipeline is dependencies, it causes extra cycles in the pipeline

- [2] a cycle in a pipeline without new VP initiation is called as extra cycle. It is also known as Stall / Hazard
- [3] when stall is present in the pipeline then $CPI + 1$

Ex: 10 cycle of pipeline op required
with 2 stall $\Rightarrow 8$ VPs are inserted

10 cycle of " " with 1 stall $\Rightarrow 9$ VPs are inserted.
" with 3 stall $\Rightarrow 3$ stalls

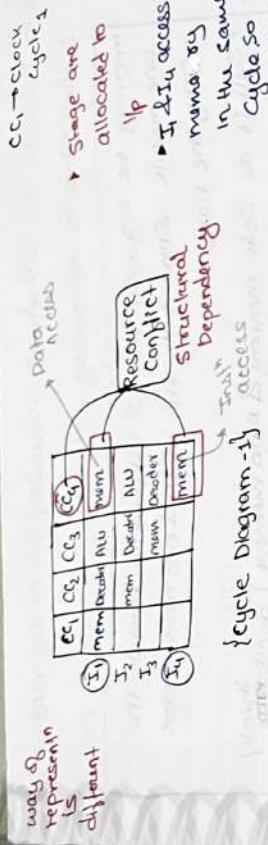
Type of Dependencies

- [1] Structural Dependencies
- [2] Data Dependencies
- [3] Control Dependencies

Structural Dependencies

- these dependencies will occur in pipeline due to resource conflict.
- resource may be registers / memory / function units (FUs)
- resource conflict means two or more instr are accessing the same resource at the same time
- resource conflict is present in the pipeline when there is a same hardware is reused in different stages. that is described in the following diagram

How to minimize the stall



- In the above execution sequence, 5 & 7th instruction are trying to access the same resource memory in the same cycle "CC". This situation in the pipeline is called as Resource conflict.
 - Conflict is an unsuccessful operation means that no one gets the control of a memory so processing suspended.
 - To handle the conflict we need to keep the "instn" in wait until the resource becomes available. These waiting create stall in the pipeline. Described below.



$\Delta_1 \Delta_2 \Delta_3$ are concentrated on
the existed V/P , it
they are already
present in the
pipeline

$$7 \text{ cycles of operation} - 4 \text{ hits} = 3 \text{ stalls}$$

- To minimize the Structural Hazards, Hardware mechanism

is used that is Re-naming.

~~memory be organised into 2 independent module in one module store only the instruction & one module store only the data~~

* Call it as Code memory & data memory [don't think]

Bring the Harvard architecture

In this architecture

Code memory - ROM

Data memory - RAM

that is different

- But here what we are doing? Same main memory
- That is organised into different module
- Where code memory is used in 1st stage & data memory in the 4th stage

- Re-naming
- In the cycle Diagram - 1, Instn "I" refers the memory
 - In 4th stage to access the data simultaneously Instn "I" refers the memory. In 1st stage to access the instruction, Instn "I" refers the memory.
 - In the same cycle "C4"
 - * When the Instn & data are both present in same memory then the above situation create conflict
 - * Re-naming mechanism state that organise the memory into 2 independent module's, used to store the Instn & Data Separately, called as code memory of data memory (DM) respectively

- Refer the code memory in 1st stage & data memory in the 4th stage so that accessing of these two memory in the same cycle doesn't create the conflict. Described below

Code memory (CM)
Data memory (DM)
Decoder (DEC)

CC ₁	CC ₂	CC ₃	CC ₄	CC ₅	CC ₆	CC ₇	CC ₈
CM	DEC	ALU	(DM)	WB			
T ₁	CM	DEC	ALU	DM	WB		
T ₂	TM	CM	DEC	ALU	DM	WB	
T ₃	TM	CM	(CM)	DEC	ALU	DM	
T ₄	TM	CM	CM	DEC	ALU	DM	
T ₅	TM	CM	CM	CM	DEC	ALU	
T ₆	TM	CM	CM	CM	CM	DEC	
T ₇	TM	CM	CM	CM	CM	CM	

7 cycles - 7 IP = stalls
& opn inserted

Amp Data Dependency

- Consider the program segment where instr "J" follows instr "I". In the program order that is

i.e. prog:

I: inst
J: inst
.....

After getting clock

clock

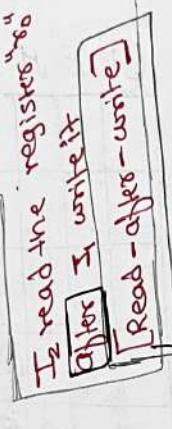
106

- Data dependency will be occur when the instn "I" tries to read the data **before** Instn "I" writes it.

Read-Before write

ex: $I_1: \text{add } r_0, r_1, r_2, r_0 \leftarrow r_1 + r_2$
 $I_2: \text{sub } r_3, r_0, r_2, r_3 \leftarrow r_0 - r_2$

I_3 is data dependent on I_1



not a
data dependency
condition

- when the above instruction Executed on a Pipeline then Data dependency condition will be occurs. b/c instruction " I_2 " is executed along with " I_1 ". so I_2 reads the data from the registers " r_0 " **before** I_1 writes its

Read Before write

- So " I_2 " incorrectly access the old value from the register " r_0 "

Cumulant Data Access

→ Data loss.

Note

- when the above instruction and executed in the non-pipeline system then data dependency doesn't occur b/c instruction I_2 will be

I_2 will be
executing after
completion of
Instn I_1 . so



not a
data dependency
condition

CC	CC ₁	CC ₂	CC ₃	CC ₄
CC ₁	0, 1, 2	1, 2, 3	2, 3, 4	3, 4, 5
CC ₂	1, 2, 3	2, 3, 4	3, 4, 5	4, 5, 6
CC ₃	2, 3, 4	3, 4, 5	4, 5, 6	5, 6, 7
CC ₄	3, 4, 5	4, 5, 6	5, 6, 7	6, 7, 8

H

it is infected b/c
Read - before - write
Data Access

DATA
ISSUE

- to detect the data dependency condition, some logic will be implemented in the "state"
- detect the data dependency condition, some logic will be implemented in the "state"

ID State

a) decide the issue

(TRANSLATION Algorithm)

Sno	Branch	Push	Independent	Dependent	Co	Each	Each	0
I ₁	ADD	r ₁	-	-	-	-	-	0
I ₂	SUB	r ₂	-	-	-	-	-	0
I ₃	MUL	r ₃	-	-	-	-	-	0
I ₄	DIV	r ₄	-	-	-	-	-	0

Ex. Stack

1) "OF" from the file
[access and the
independent
values]

r₁ value
r₂ value

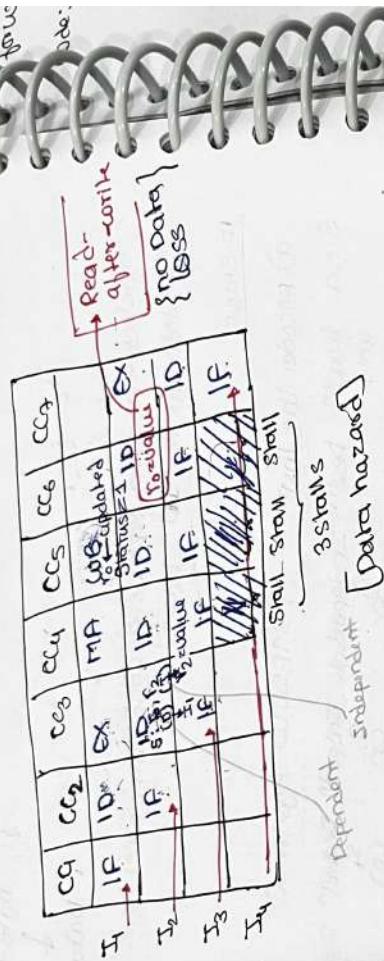
r₃ value
r₄ value

T
(Ex-stages
allocation)
Ex. Stack is not
allocated

TD₂

- By using the above logic we can stop the accuring of unwanted data so that instruction will be waiting on "ID" stage until the data become available. These waiting create stall in the pipeline.

Described below

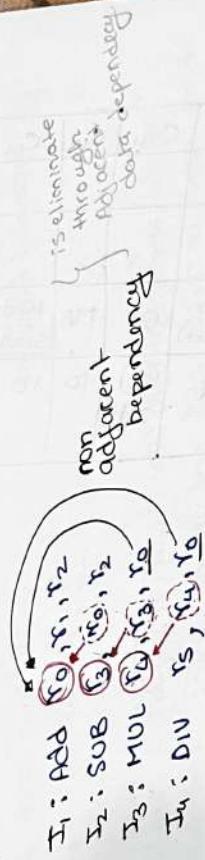


- to minimize the data hazard , hardware mechanism is used **Operand forward** [also name as **By-passing** | short - circuiting]
- these technique status that use the buffer between stages, to load the intermediate OLP : so that new data will be accrued from the buffers before updating the register file. In these technique "WB" stage is modified to perform Write. Read & write operation means . the updated data in the registers is immediately available for read . so no buffer is required at the end of last stage.

- Buffer delay is only consider in pipeline not consider in non-pipeline

- considers the following code, execute with operand forwarding & w/o [without] operand forwarding

Code:



$T_2 - T_1(r_0)$ Adjacent
 $T_3 - T_2(r_3)$ Data dependency
 $T_4 - T_3(r_6)$ Is called as True data dependency

Execution sequence with operand forwarding

	CC ₁	CC ₂	CC ₃	CC ₄	CC ₅	CC ₆	CC ₇	CC ₈	CC ₉	CC ₁₀	CC ₁₁	CC ₁₂	CC ₁₃	CC ₁₄	CC ₁₅	CC ₁₆	CC ₁₇	CC ₁₈	CC ₁₉	CC ₂₀	
T_1	IF	ID	EX	WB	WB																
T_2			IF	ID	EX	WB															
T_3				IF	ID	EX	WB														
T_4					IF	ID	EX	WB													

Cycles required

Call the ALU units start from "EX" stage. So all forwarding start from next stage.

|| use one forward from the buffer as it's same. The buffer is not full instruction not load instruction

Execution will demand forward-looking

- 1) Consider RISC Pipeline used to execute the following where all instruction suspending 1 cycle on all the stage but load. Then takes 3 cycles on unstage

I₁: Load r₀, 3(r₁)
 I₂: Add r₃, r₀, r₁
 I₃: Load r₂, 5(r₁)
 I₄: Sub r₄, r₂, r₁

How many cycles are required to complete the code with & without operand forwarding.

- 2) Consider 4 stage (IF, ID, EX, MA) pipeline where all instructions are spending 1 cycle on all the stage. But mul takes 3 cycles & div takes 4 cycles on EX-stage. Following code is executed in the pipeline operand forwarding is used in the pipeline
- In the pipeline operand forwarding is used in the pipeline
- a) How many cycles are required to complete the code.
 b) How many cycles are saved using operand forwarding over w/o operand forwarding?

I₁: mul r₀, r₁, r₂
 I₂: DIV r₃, r₁, r₄
 I₃: ADD r₅, r₃, r₁
 I₄: SUB r₆, r₅, r₃

① Goal: RISC Pipeline

5-stage

IF, ID, EX, MA, WB

1	1	3	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

in the program
 there is a load instruction
 so load instruction of P
 is present in the
 unstage MA, so

QDA

பெப்ரவரி 81

⑥ 10/10 operando forwarding

Happy 15th birthday

Si us
purda

40

With operated by operator

ပြန်လည် ၈၁

$$\# \text{cycle} \text{ scaled} = (16 - 12) \Rightarrow \# \text{cycles} //$$

(b) of the Operated forward by

182

With regard to the following

comes there is a date b/c all and ALD

250 4-Sage

Jmp Control Dependency

- Control dependency will occur in the pipeline due to a branch instruction execution in the pipeline

Conditional function [Transfer of control signal]

Jump

code: $T_1 - S_1 - T_2 - B_{S1} - B_{T2}$

$T_1: T_2$

$T_2: T_3$ (Jump 2000)

$T_3: T_4$

$T_4: T_5$

$T_5: T_6$

$T_6: T_7$

$T_7: T_8$

$T_8: T_9$

$T_9: T_{10}$

$T_{10}: T_{11}$

$T_{11}: T_{12}$

$T_{12}: T_{13}$

$T_{13}: T_{14}$

$T_{14}: T_{15}$

$T_{15}: T_{16}$

$T_{16}: T_{17}$

$T_{17}: T_{18}$

$T_{18}: T_{19}$

$T_{19}: T_{20}$

$T_{20}: T_{21}$

$T_{21}: T_{22}$

$T_{22}: T_{23}$

$T_{23}: T_{24}$

$T_{24}: T_{25}$

$T_{25}: T_{26}$

falling edge path (execution from the start address)

execute from target address onwards

$T_2: T_3$ taken path

Expected : $T_1 - S_1 - T_2 - B_{S1} - B_{T2}$

OIP

$T_1: PC \rightarrow MAR$

Instn.fetch: $T_2: M(MAR) \rightarrow MDR$

$PC + 1 \rightarrow PC$

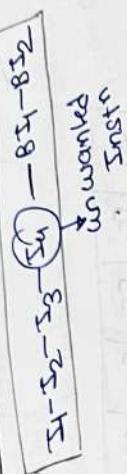
$T_3: MDR \rightarrow (IR)$

$T_4: MBR \rightarrow$

In the middle of fetching the PC is updated not at the end. So the address of the next PC is updated to sequential address. So in the middle of fetching the PC value is changed.



Actual OIP seq: T1-T2-T3-T4-T5-T6-T7-T8-T9-T10-T11-T12-T13



Instructions

- in the above execution sequence, unwanted sequence is executed in the pipeline fetched into pipeline. when it is executed in the situation then the program oip is missing.
- then the program is called as control dependency.
- in the pipeline is called as flush operation.
- to handle the above situation, flush operation is used, to suspend the unwanted instruction after the jump substituted the NOP. It is also called as substitute in the program.
- in the program.

freeze operation consuming the input is means consuming the output.

- NOP instruction is a delay instruction.
- cycle without changing the output.
- cycle is inserted after a jump.
- delay instruction never enters into pipeline.
- unwanted

Code with flush op

1000: $\text{JZ } \text{I}_1$
 1001: $\text{JZ } \text{I}_2$
 1002: $\text{JZ } \text{I}_3 \text{ cmp } \text{R}2, \text{R}0$
 1003: NOP
 1004: $\text{JZ } \text{I}_4$

2000: $\text{BT } \text{I}_1$
 2000: $\text{BT } \text{I}_2$

	$t=1000$											
	PC:1000											
$\text{JZ } \text{I}_1$												
$\text{JZ } \text{I}_2$												
$\text{JZ } \text{I}_3 \text{ cmp } \text{R}2, \text{R}0$												
NOP												
$\text{JZ } \text{I}_4$												
$\text{BT } \text{I}_1$												
$\text{BT } \text{I}_2$												

Actual
Old : $\text{JZ } \text{I}_1 - \text{JZ } \text{I}_2 - \text{NOP}$
Des.

$\boxed{\text{JZ } \text{I}_1 - \text{JZ } \text{I}_2 - \text{NOP}}$

unwanted instr
 \rightarrow delay instr
 \rightarrow stall
 \rightarrow hazard

Ex seq is
 not same
 wrt expected
 \oplus program but
 program op
 is SAME

- we inserted only one NOP block only one unwanted branch in the pipeline [the target address present in the 2nd stage] so, only one stage behind the target stage.
- so only one unwanted branch enters into RISC pipeline
- so only one unwanted branch present in the 2nd stage.
- b/c target address present in the 2nd stage.
- so no suspend when one unwanted branch is used only one NOP required.
- so if the target address is in stage two
 - "NOP" required
 - so how many NOP is inserted after the jump
 - so how many target address availability will depend on the target address availability.
- in-line RISC pipeline Target is present in 2nd stage pipeline
- no. of stall [NOP] created in the pipeline due to Branch instn is called as **Branch penalty**.
- Branch penalty is depend on target availability of a branch address in the pipeline

i.e. Branch = At what stage -
penalty = Target address is available

- Target address is available in the 2nd stage
- in RISC pipeline Branch penalty is always \leq BIC

Note • For some time now, we have been working on the problem of how to make pipelines target address availability

• For hypothetical pipelines target address availability
given as,

① Stage no. given then,

$$\text{Penalty} = \text{given stage} - 1$$

② Stage name given then,

$$\text{Penalty} = \text{corresponding name stage} - 1$$

name numbers

③ Until the instr is complete at all the instn and proceed through all the stages then

$$\text{Penalty} = \text{last stage} - 1$$

stalls from the Branches during the program execution = Branch * freq * penalty

Branches / Instn / prog.
stalls / Branch

note:-
to minimize the control hazard, hardware mechanism

is used called as Branch prediction buffers also called as [Branch target buffer] loop buffer

it is a high speed buffer, present in the 1st stage of

the pipeline to hold the predicted target address.
when the target address is available in the 1st stage
then no stall is present. this concept is working when
the program contains loop

note:-
when the question contains pipeline with branch predictor
then we can assume that target address is present
in the 1st stage means no stall otherwise [without
branch prediction] assume that target address is not
available in the 1st stage. no stall present
(I/O technique)

note:
to minimize the control hazard software mechanisms

used i.e. Delayed Branch

Delayed Branch (S/W technique)

• Delayed Branch.
• If is a Compiler rearranges the code if possible, to avoid
the stall. otherwise substitute the "nop" instruction after the
Jump instr. to preserve the execution path if not
possible to rearrange

"compiles"

a) Rearrangement
[swap the Jmp Instn with
the prev. Instn]

User code:

1000 : I_1
 1001 : I_2
 1002 : $\text{I}_3(\text{Jmp} \quad 2000)$
 1003 : I_4
 :
 2000 : B_{I_1}
 2001 : B_{I_2}

Expected : $\text{I}_1-\text{I}_2-\text{I}_3-\text{B}_{\text{I}_1}-\text{B}_{\text{I}_2}$
 O/P

Actual O/P seqn : $\text{I}_1-\text{I}_2-\text{I}_3+\text{I}_4-\text{B}_{\text{I}_1}-\text{B}_{\text{I}_2}$
 unwanted Instn

Code:
 micro generated
 Address: $\text{I}_1-\text{I}_3(\text{Jmp } \text{B}_{\text{I}_1})$
 use have I_2
 rearrange I_3
 them

B_{I_1}
 B_{I_2}

PC: I_1	CC_1	CC_2	CC_3	CC_4	WB	MA
PC: I_2	IF	IO	EX	MA		
PC: I_3	IF	IO	EX	MA		
PC: I_4	IF	IO	EX	MA		
I_3	IF	IO	EX	MA		
I_2	IF	IO	EX	MA		
B_{I_1}				IO		
B_{I_2}				IO		

↓
PC STORE
address
of I_1

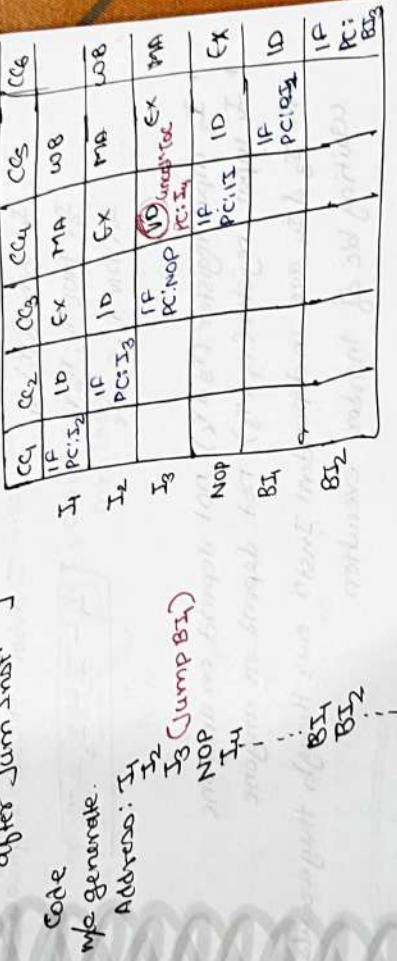
[we know the
sequence of
Instns]
we don't know
the sequential
addresses
here.

Actual O/P - seq : $\text{I}_1-\text{I}_3-\text{I}_2-\text{B}_{\text{I}_1}-\text{B}_{\text{I}_2}$
 Jump

- I_3 is just a jump instruction so sequence of that doesn't matter
- but the rest of them can't be rearranged
- Execn seqn is different wrt expected seqn but program o/p is Same

③ NOP substitution

[Insert NOP before
after Jump instr]



Actual opseq: $T_1 - T_2 - T_3 - \text{NOP} - \text{BT} - \text{BT}_2$

unwanted

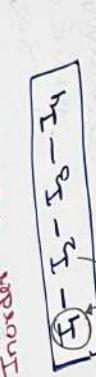
execn seqn is diff wrt expected
O/p seqn but program

- Suppose there is another pipeline where the target address is present in the 4th stage then 3 stall possible. If you want to implement delayed branch concept then rearrangement of the three step above in the program if that is not possible then insert 3 NOP after jump inst.

Intra Scheduling

- CPU always execute the program in a sequence from top to bottom called as Inorder execution.
- In the Inorder execution sequence, if any instr is dependent then the remaining instr are also starting the stall cycle even they are independent.

Ex: $I_1: \text{Add } R_1, R_2, R_3, R_4$



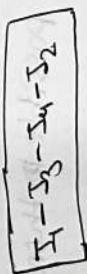
$I_2: \text{Sub } R_3(R_4) R_4$

$I_3: \text{Mul } R_4, R_5, R_6$

$I_4: \text{Div } R_3, R_7, R_8$

- I_3 input register (R_4, R_6) not depend on anyone.
- I_4 input registers (R_4, R_8) not depend on anyone.
- $\therefore I_3$ & I_4 are independent instr even though they are also waiting bc of Inorder execution
- I_2 depends on I_1 . So,
- In the above execution sequence, I_2 will be waiting until the data become available. So stall will be created. These stall are also shared by the I_3 & I_4 instr even they are independent

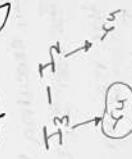
- To handle the above situation, Inorder scheduling concept is used
- Scheduling concept states that execute the independent instr first called as Out-of-order execution [re-order executing].



208

- out of order execution create 2 more dependency in the pipeline

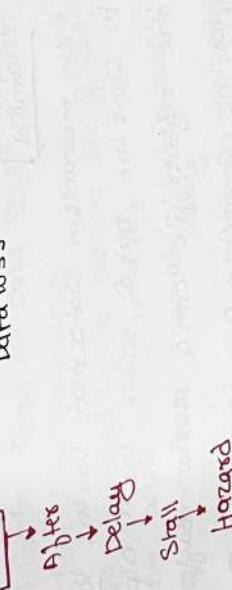
1 Anti data dependency



Read-Before-write is True
Data loss

Write-Before-Read is Anti
Data loss

2 O/P Data dependency



Anti-Data Dependency

- These dependency will occur in the pipeline when the instruction "I₂" try to write the data before instruction "I₃" read it

Write-Before-Read

Ex: I₃ executing before I₂. so I₃ update the register I₄ before I₂ read it

- before I₂ read it new value from the register I₄
- I₂ incorrectly reads the unwanted data [Read]

Old Data Dependency

- these dependency will occur in the pipeline when instr I_2 tries to write the data before instr "5" write it.

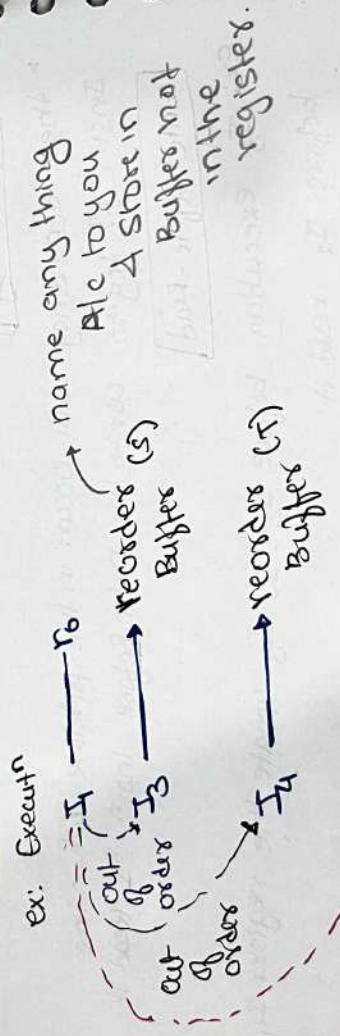
write-before-write

ex: I_4 executing before I_2 . So I_4 update the register I_3 before I_2 writes it
 unwanted data write}

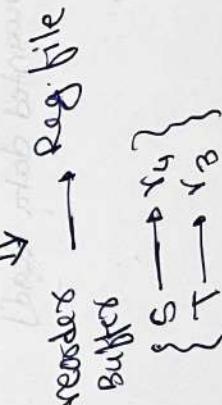
Note → to minimize the anti and old dependency stall
 Hardware mechanisms used that is register renaming

- these mechanism states that use the Re-order buffer to store the out of order instr old later update the registers file with a reorder buffer contents after the completion of a dependent instr execution

∴ data is safe



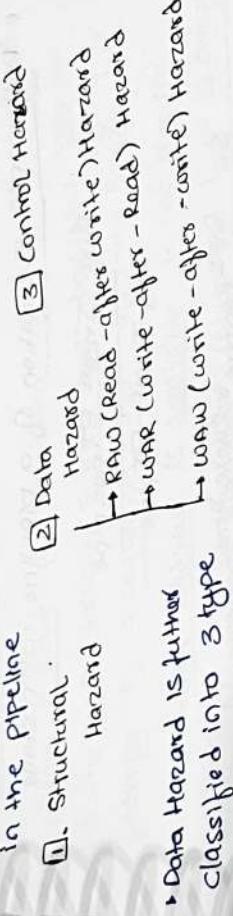
In-order $\rightarrow I_2 \rightarrow v_3$ (store to)



$v_3 \rightarrow v_4$

Hazards

- Hazard is a delay . delay is present in a pipeline due to a dependency condition . so 3 kind of hazard are possible in the pipeline



RAW Hazard : The Hazard is created into a pipeline when Instr "J" tries to Read the Data before Instr "I" write it {True Data Dependency}

[RAW Hazard is created in the pipeline b/c of Instr "I"]
Reading data before Instr "I" write it
Due to RAW the Raw Hazard / delay is present

WAR Hazard : These Hazard is created in the pipeline when the Instr "J" tries to write the data before Instr "I"

Read it {Anti - Data Dependency}
Due to WAR the WAR Hazard / delay is present

WAW Hazard: These Hazard will be created in the pipeline when the Instr "J" tries to Write the data before Instr "I" write it {Op data Dependency}
Due to WAW the WAW Hazard / delay is present

[Before - problem After - delay]

Instr J	Instr I	Dependency	Hazard
O/P reg. == O/P reg.		True Data	RAW Hazard
C/O reg. == I/P reg.		ANTI Data	WAR Hazard
C/O reg. == O/P reg.		O/P Data	WAW Hazard

note: RPR is not a hazard
B/C RPR is not a dependency

- Performance evaluation of a pipeline with stalls

$$S = \frac{\text{Avg. Instn ET nonpipe}}{\text{Avg. Instn ET pipe}}$$

$$S = \frac{\text{CPS nonpipe} * \text{cycle time nonpipe}}{\text{CPS pipe} * \text{cycle time pipe}}$$

- Ideal CPS of pipeline is always ① but due to the dependency stalls are present in the Pipeline

$$\Rightarrow S = \frac{\text{CPS nonpipe} * \text{cycle time nonpipe}}{(1 + \# stalls) \cdot \text{instn ET}}$$

one task execution time (tn)

$$S = \frac{tn}{(1 + \# stall) \cdot tn}; \text{non uniform delay pipeline}$$

trunk ET

$$S = \frac{K \cdot tp}{(1 + \# stall) \cdot tp}$$

In the uniform delay pipeline

$$S = \frac{K}{(1 + \# stall) \cdot instn}$$

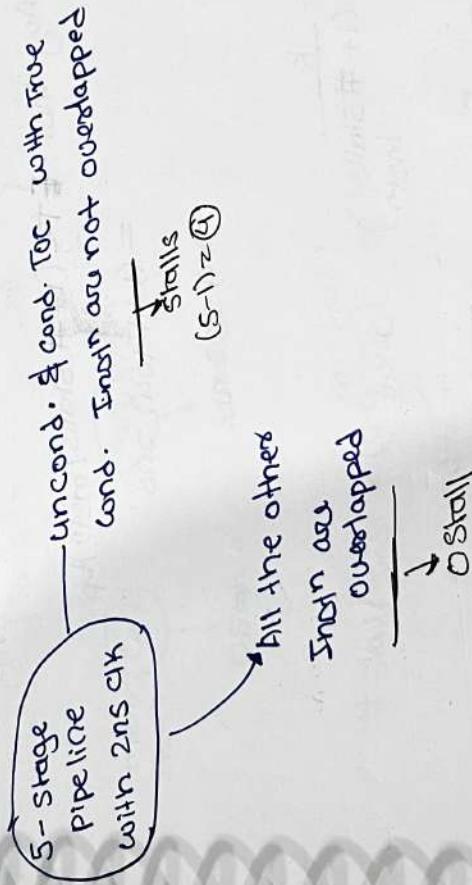
uniform delay pipeline
no of stalls per instn

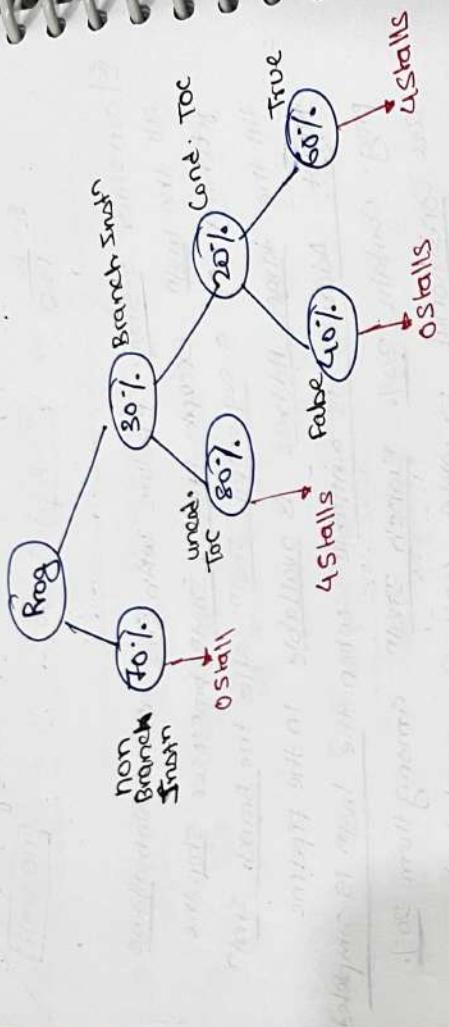
- when the system is operating with $100' \cdot n$ then nosall

$$S = \frac{k}{1+O} \Rightarrow \boxed{S=K} \quad n=100'.$$

Q) consider 5-stage pipeline with 2ns CLK which allows all the instr except Branch instr processes. Stop the fetching of a sequential instr after the Branch instr till the target Address is available in the pipeline. target Address is available when the instr is completed. Program contains 30'. Branch instr among them 40'. Q) instr doesn't are conditional in which 40'. When cond is false then satisfy the condition. when the cond is false then the following instr are overstapped

- Avg instr ET
- Speed UP





$$\#(\text{no. stable instances}) = (0.740) + (0.348 * 0.8 * u) + (0.3 * 0.2 * 0.4 * u) + (0.3 * 0.2 * 0.6 * u)$$

Aug 2019 $\#T = (1 + \# smalls / max) * \#P$

$$\Rightarrow 4.208 \text{ m.}$$

$$⑥ S = \frac{K}{(1 + \# \text{ stalls})} ; \text{ uniform delay}$$

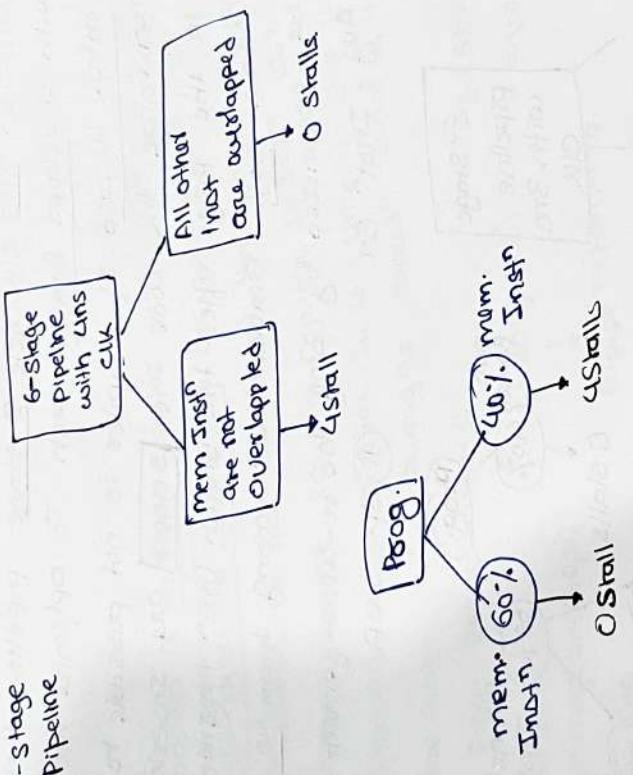
$$S = \frac{5}{(1+1 \cdot 10\%)}$$

Q) Consider 6 stage pipeline with a respective delays of 2ns.
Ans: 2ns, which allows all the loads except memory to complete their execution in 6 stages. The memory reference load will take 7 stages.

⑨ Aug. Inst'n Execution Time

(g) Speedups

Sol: 6-stage Pipeline



#S'all / S'us'ya = (0.4 * 4)

6
N

Aug. 2019 = 147 stalls (initial) to

$$\Rightarrow (1+1.5) \text{ units} = 10.4 \text{ ns} //$$

214

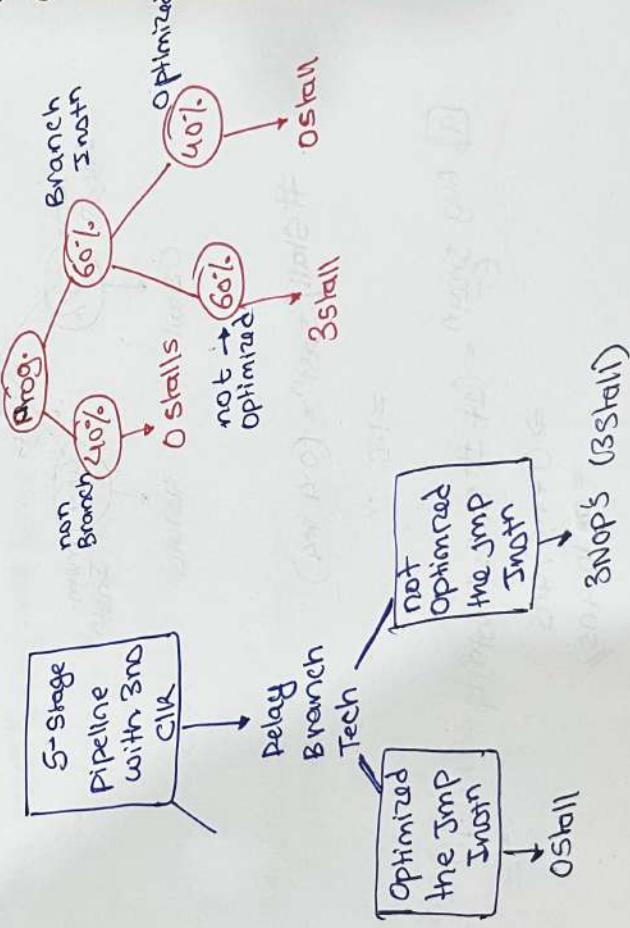
$$b) S = \frac{t_D}{C_1 + \# \text{stalls} / \text{Instn}^n t_D} ; \text{ non uniform delay}$$

$$S = \frac{(2t_1 + t_4 + 2t_3 + 2)}{10 \cdot \text{uns}}$$

$$S = \frac{14}{10 \cdot \text{uns}} \Rightarrow S = 1.34 \text{ ns}$$

(as individual delay is next given)
uniform delay

- Q) consider 3ns clk cycle 5-stage pipeline designed with a delayed branch tech. To determine the control hazard. If compiler is not possible to rearrange the code the SNOP's are inserted in the program after the Branch instr. program contain 60%. Branch instr among them 40% are optimized by using the re-arrangement
 (a) Aug. Instn Err (b) Speed up



$$\# \text{ std}(15) / \text{ std} = \frac{1}{\sqrt{0.6}} = 1.154$$

$$S = \frac{k}{(1 + \# \text{ stalls}) / \text{work}}$$

uniform delay

a) Aug. instr EPIPE
 $\rightarrow C_1 + \# stalls (instrn) + P$
 $\rightarrow (1.1 \cdot 0.8) \cdot 3ns$
 $\rightarrow 6.24ns$

Sz 2.4

$$S = \frac{5}{(1+0.08)}$$

(Q) consider a 5-stage pipeline with a respective delays of
 2ns, 14ns, 3ns, 6ns & 5ns designed w/o branch prediction. All Insts are proceed through all the stages in the pipeline program contain 20 Insts (5 to 520) executed on a pipeline. It is a uncond. Jmp exists on a pipeline. In what is the control to JI6. During its execution what is the Prog. Executn Timer?

$$t_0 = \max(2, 4, 3, 6, 5)$$

4926

$$G_{\text{Tripe}} \equiv (k+n-1)!$$

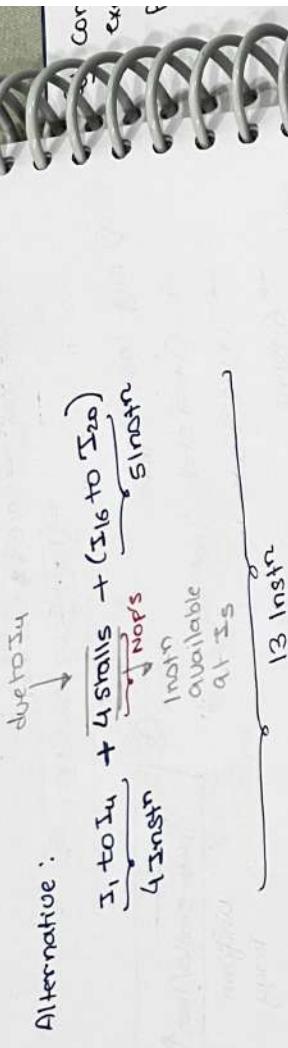
$\text{N}_2\text{O}_5(\text{ginit})$

22/10205.

$\Rightarrow (5 + \sqrt{5})^{-1}$ ends



Alternative :



∴ Insns are executing in overlapping orders & T_{q1} is unclear
jump, In RISC pipeline unconditional jump target address present in 2nd stage but for question all insns processed through all stages so every target op will be present in last stage.

- ⑥ How much time is saved to execute the program with prediction in the pipeline over no prediction.

Sol: with prediction means Target Address is available in 1st stage

∴ no stalls

$$\text{Prog: } (\overbrace{I_1 \text{ to } I_4}^4 + 0 \text{ stalls} + \overbrace{(I_{16} \text{ to } I_{20})}^5) \cdot q \cdot \text{ipscn}$$

$$\begin{aligned} ET &= (k+n-1) \cdot t_p \\ &= (5+9-1) \cdot 6 \text{ ns} \\ &= 78 \text{ ns} \end{aligned}$$

$\Rightarrow \boxed{2 \text{ ns}}$

Q1 Consider 5-stage pipeline with 2ns clk used to execute the program, which contains 150 ns. In the program 3 cat-of instruction is present with a (consecutive) respective stalls of $\frac{3}{2} + \frac{1}{2}$. In the prog. 3 cat-of stalls are equally distributed i.e. (so 150). What is the program execution time?

$$\begin{aligned}
 & \cdot K = 5 \\
 & \cdot t_p = 2 \text{ ns} \\
 & \cdot n = 150 \text{ (initial)} \downarrow \\
 & \quad C_{\text{Pipeline}} = (K+1)t_p \\
 & \quad (5+450-1)2 \text{ ns} \\
 & \quad \rightarrow 908 \text{ ns}
 \end{aligned}$$

$n = 150 + 300$
= 450

→ 300 stalls

Q1 Consider the following program code, execute on a pipeline system

System
 How many RAW, WAR & WAW
 Hazard possible in the prog.

$I_1: r_0 \leftarrow r_1 + r_2$
 $I_2: r_1 \leftarrow r_0 * r_2$
 $I_3: r_2 \leftarrow r_1 * r_0$
 $I_4: r_0 \leftarrow r_2 - r_1$
 $I_5: r_1 \leftarrow m[2000]$
 $I_6: M[2000] \leftarrow r_1$

Raw [un-ordered]
order
[top reg. == 1/preg.]

$(T_2 \rightarrow T)$ T_2 -dependent registered

con
adjacent

data dependency

T₀ - TS(C₁)

4RAW

$$\begin{array}{l} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array}$$

800

$$\begin{array}{l} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array}$$

800

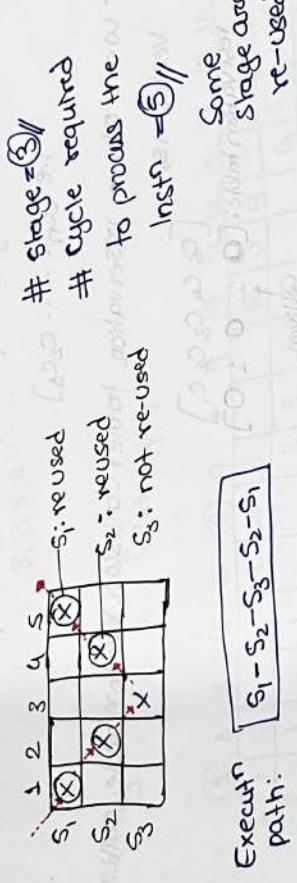
$$\begin{array}{l} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array}$$

800

Computer Organization & Architecture

Non linear Pipeline

- These pipeline contain forward & backward connection.
- So reservation table is used to processes the IP.
- Consider the following reservation table, to process the IP in a hypothetical non linear Pipeline



Latency Analysis

- Latency means CLK cycle difference b/w 2 successive initiation

In the pipeline in the above diagram, some of the latency causes the collision in non-linear Pipeline, some of the latency causes the collision [Stage conflict] called as **forbidden latency** (non permissible latency)

If some of the latency doesn't create the conflict called as **non-forbidden latency** (permissible latency)

- To identify the forbidden latency, we need to take the CLK cycle difference b/w any two check mark in a row with respect to above reservation Table

$\text{row}_1(s_1) : (S-2) = 4$ latency & mean insert a new VP

$\text{row}_2(s_2) : (C_1-2) = 2$
in strcycle then legal

$\text{row}_3(s_3) : \underline{(C_1-2) = 2}$
(used only once)

• collision vector is reported based on the reservation table

n: #cycle req. to process the VP

i.e. $[c_n, \dots, c_2, c_1]$

w.r.t above reservation table, collision vector table, collision

vectors is
 $[c_8 \ c_4 \ c_3 \ c_2 \ c_1]$
reservation table: $[0 \ 1 \ 0 \ 1 \ 0]$

collision

• latency sequence is generated based on the collision vectors.

$(2-1) = \underline{\text{latency}}$ w.r.t "q"
 $(1,2) = 3$ 4 5 6 7 8 9 10 11 12 13

s_1	1	2	2	1	2	3	4	3	4	5	
s_2	1	2	1	2	3	4	3	4	5		
s_3	1	2	1	2	3	4	3	4	5		
s_4	1	2	1	2	3	4	3	4	5		

Note: latency cycle from 3rd sequence use $(2-1)$ will decide & the $"2"$ sequence

$(2-1) = 1$ $(7-2) = 5$ $(1+5) = 6$ $\Rightarrow (1+5)/2 = 3$

sequence given by $(3-8) = 5$ $(8-7) = 1$ Collision vector $(1,5)(1,5)(1,5) \dots$

9/10/16

9/10/16

Latency seq w.r.t "C"				Latency 3	$(4-1)=3$	Latency 3	$(4-1)=3$
①	2	3	④	5	6	7	8
s_1	1	1	2	1	3	2	4
s_2	1	-	1	2	3	3	4
s_3	-	1	2	3	3	4	4

Latency cycle: $(4-1)=3$
 $(7-4)=3$
 $(10-7)=3$

$3, 3, 3; 3 \dots$

Latency seq w.r.t " C_S "
Latency of 5

①	2	3	4	5	⑥	7	8	9	10	11	12	13	14	15	16	17
s_1	-	1	1	2	3	1	2	3	4	5	4	5	4	5	4	5
s_2	-	-	1	-	2	3	2	3	3	4	5	4	5	4	5	4
s_3	-	-	-	-	-	1	2	3	2	3	4	5	4	5	4	5

Latency cycle $(5-1)=5$
 $(7-6)=1$
 $(12-7)=1$
 $(13-12)=1$

Aug. latency $\approx \frac{(5+1)}{2} = 3$
 $(7-6)=1$
 $(12-7)=1$
 $(13-12)=1$

$(s_1)(s_3)(s_1) \dots$

So use only 1 among all three $[c_1, c_3, c_s]$ don't use all or 2
blocks having same performance + Aug. latency

In min there is more period
& max there is less performance

ML (min. Aug. latency) $\Rightarrow \min(3, 3)$
 $= 3 //$

• Memory organization

$$\text{Performance (CP)} \propto \frac{\text{access (AT)}}{\text{Time}}$$

Access Time Execution Time

AT < ET

$$\text{Total Time} = \text{Hit / miss latency} + \text{access time} + \text{transfer time}$$

negligible

depends on the Bandwidth

not given in Question

take "0" not given

Type of memory org

- ① simultaneous Access memory org [not in use]
- ② Hierarchical Access memory org

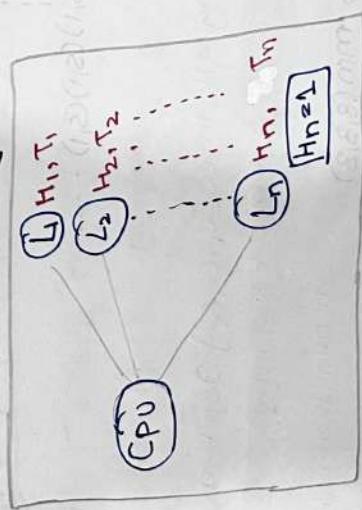
Hit/miss latency: time taken to respond that is it hit / miss

Hit/miss latency \neq compulsory latency \neq compulsory damage had occurred

it should be "0".

Access Time: Data Preparation time [after receiving the Address & control signal, then how much time memory is taking to prepare the data to transfer.

④ simultaneous Access memory organization [why should I change?]



Here, L₁, L₂, ..., L_n: level of the memory

T₁, T₂, ..., T_n: Access time of memories

218

Relation is,

$$T_1 < T_2 < T_3 < T_4 \dots -$$

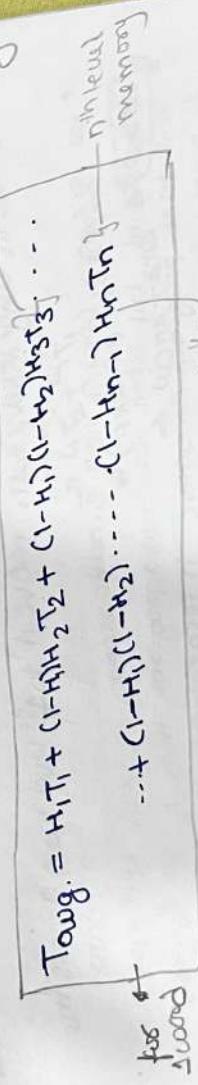
$H_1, H_2, H_3, \dots, H_m \rightarrow$ Hit Ratio's
of a
memories

$$\text{Hit Ratio}(H) = \frac{\# \text{ Hits}}{\text{Total } \# \text{ Access}}$$

$$\boxed{\text{Hit Ratio} + \text{Miss Ratio} = 100\%}$$

- in those organization, CPU directly connected to the all the level of a memory, but accessing the memory in a sequence that is CPU refers the level 1 memory to access the data. when data is present in level 1 memory then you directly access the data from level 1 memory otherwise request will be forwarded to level 2 memory
- when operation is hit in level 2 memory then CPU directly access the data from level 2 memory without copying the data in level 1 memory. otherwise request will be forwarded to level 3 memory
- when hit in level 3 memory then CPU directly access the data from the level 3 memory without copying the data into level 1 & level 2 memory otherwise request will be forwarded to level 4 memory & so... on.

- time required to access one word data from the memory is called as Avg. memory Access Time [Tavg.]



Hit ratio of last level is always ≤ 1

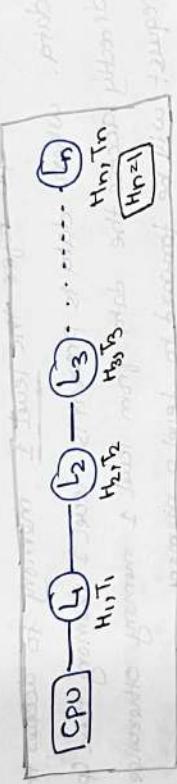
1 word — T_{avg}

? # words — 1 sec

$$n \left(\frac{\text{Throughput}}{\text{word}} \right) = \frac{n \text{ word/sec}}{T_{avg}}$$

[2] Hierarchical Access memory organization [How Should I change]

Ansatz 1: for better performance add: L_1 memory & L_2 / L_3 / L_4



In these organisation CPU always access the data from level 1 memory. When there is miss in level 1 memory then the corresponding data will transferred from that higher level to L_1 memory later CPU will be accessing the data only from level 1 memory.

$$\begin{aligned} T_{avg} &= H_1 T_1 + (1-H_1) H_2 (T_2 + T_1) + (1-H_1) H_3 (T_3 + T_2 + T_1) \\ &\quad + (1-H_1) (1-H_2) (1-H_3) \dots - (1-H_n) (T_n + T_{n-1} + \dots + T_1) \end{aligned}$$

If miss in any level, then move it to 1^{st} level then access the data.

For one word point of view, simultaneous access is local but not hierarchical but when we run a program there is set of word so in that program execution hierarchical is best b/c of principle of locality of reference & to minimize the access time.

ex: Code with reusability

Simultaneous

I. Data (Assume, Data)

Time: $t_{T_2} \geq t_{T_1}$ ~~Time~~

the FT_t Data

[Change]

三

Ex. Code with linear access of data [simultaneous possible write
program can cause race condition] \rightarrow Simultaneous

T_1 : Data (w₁) Assume all words
 T_2 : Data (w₂) are in the
 T_3 : Data (w₃)
 T_4 : Data (w₄)
 T_5 : Data (w₅) [L] memory

Assume, $T \approx \Delta_{\text{vibrations}}$

$\Rightarrow \text{Lions} + \text{Tigers} = \text{Lions}$

Hierarchical orgⁿ into n blocks.

Assum, Block size = 8 ns

$T_1 : \tau_1, T_2 + \tau_1$

$T_2 : \tau_1$ Spatial locality
 $T_3 : \tau_1$ (adjacent data words)
 $T_4 : \tau_1$ (accessing in a sequence)

Note

Time: $T_2 + 4T_1$

Block access + (4*2 ns)

None (Gions)

$\Rightarrow \geq 18 \text{ ns}$

due to these access time reduce

It is 2 type

1 Temporal

- to statisfy the above locality of reference; organize the data in memory in the form of block
- block contain multiple words. Block size is decided by the designer of the system
- Hierarchical orgⁿ is used in the computer design.
- Simultaneous orgⁿ is not in use.

Note: when the question contain hierarchy word /

Hierarchy meaning / cache memory word than use

Hierarchical orgⁿ [Otherwise use simultaneous orgⁿ]

Q1 In a 2 level memory org L₁ memory is q times faster than L₂ mem & its access time is sams less than the avg. memory access time. Let L₁ mem. access time q times what is the hit ratio?

Sol:

- Simultaneous org (idle time given)
- L₁: H₁, T₁
- L₂: H₂, T₂

which ever faster that should be denominator

$$\frac{H_2 T_1}{H_2 + T_1} \Rightarrow \frac{q T_2}{T_1 + q T_2}$$

$$so, T_2 = T_{avg} - 50$$

$$\therefore T_{avg} = T_1 + 50$$

Let T₁ = 40 ms ; Then, T₂ = 9 * 40 ms

$$T_{avg} = 40 + 50 = 90 ms.$$

$$T_{avg} = H_1 T_1 + (1-H_1) H_2 T_2 = 85 + 85 = 170 ms$$

$$q0 = (H_1 + 40) + ((1-H_1) * 360)$$

$$q0 = 40 H_1 + 360 - 360 H_1$$

$$320 H_1 = 270$$

$$H_1 = \frac{270}{320} = 0.84375$$

Q2 If H₁ = 0.5, H₂ = 0.1, T₁ = 100 ns, T₂ = 300 ns

$$T_{avg} = 100 * 0.5 + 300 * 0.1 = 130 ns$$

Q1 3 level mem org

has the following specification.

If block size not given then assume

level	Access time/word	Block size in word	Access block size	
			Hit Ratio	Time/block
1	40ns	1	0.7	$T_1 = 40 \times 0.7 = 28$ ns
2	200ns	2	0.9	$T_2 = 200 \times 0.9 = 180$ ns
3	100ns	4	1	$T_3 = 100 \times 1 = 100$ ns

If the referred data is not in L₁ main then copy
the block from L₂ to L₁. If not in L₂ then
copy the block from L₃ to L₂ & L₂ to L₁.

what is the Avg. memory Access time T_{Avg}?

Hierarchical org

$$T_{Avg} = H_1 T_1 + (1-H_1) H_2 (T_2 + T_1) + (1-H_1)(1-H_2) H_3 (T_3 + T_2 + T_1)$$

$$\Rightarrow (0.7 \times 40) + (1 - 0.7) 0.9 (400 + 40) + (1 - 0.7) ((1 - 0.9) / (1600 + 400 + 40))$$

$$\begin{aligned} &\Rightarrow 28 + (0.27 \times 440) + (0.03 \times 2040) \\ &\Rightarrow 28 + 118.2 + 61.2 \\ &\Rightarrow 208 \text{ ns} \end{aligned}$$

Q1 Consider cache memory having the hit ratio of 80%. If access time of main memory is 300 ns.

what is the Avg. memory access time

- Cache memory given
- So memory organisation is hierarchical

$$\begin{aligned} \text{CM: } H_C &= 80\% \\ T_m &= 300 \text{ ns} \\ T_C &= 80 \text{ ns} \end{aligned}$$

Tavg. = ?

$$\text{Tavg.} = H_c T_c + (1-H_c) H_m (T_m + T_c)$$

$$= (0.8 * 80) + (1 - 0.8)(300 + 80)$$

⇒ 140ns

Memory hierarchy design.

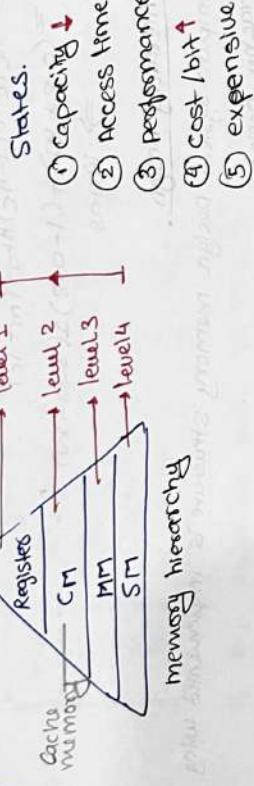
- In the computer system design memory structure is implemented using the hierarchy design
- A/C to memory hierarchy design system supported memory Standard is as follow

Mem. Standard.

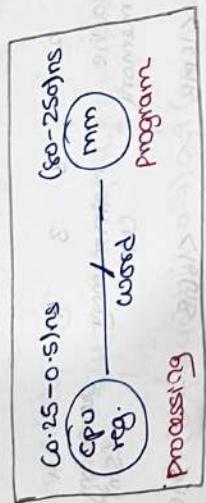
Level:	1	2	3	4
name:	Register memory	Cache memory	main memory	secondary memory
Typical size:	<1KB	<16MB	<16GB	>100GB
Implemented:	Customized	SRAM	DRAM	magnetic
Access time:	(0.25-0.5) ns	(0.5-25) ns	(10-250) ns	(20-150) ns
Band width:	(20000- 100,000) MB/sec	(5000- 10,000) MB/sec	(1000- 5000) MB/sec	OS
Managed by:	Compiler	HW	SW	compact disc (CD)
Backed by:	CM	MM	200GB hard disk	CD

Bottom-Top approach

Span memory hierarchy

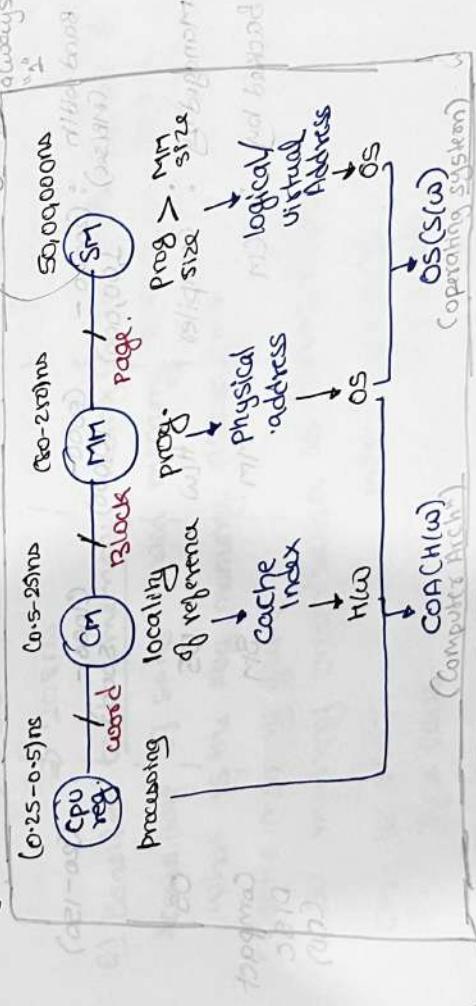


- System design w/o hierarchy



{ speed gap \uparrow b/w the CPU & MM }

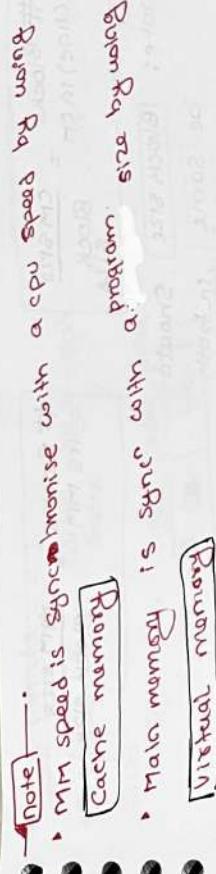
System design with hierarchy



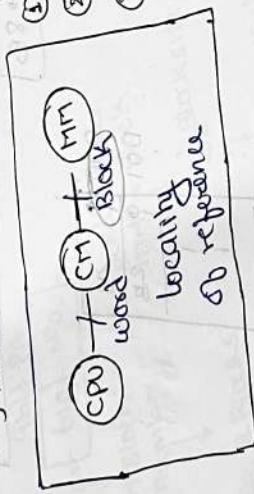
$$T_{avg} = H_1 t_c + (1-H_1) H_m (T_m + T_c) + (1-H_2)(1-H_m) H_S (T_S + T_m + T_c)$$

cache memory data access from cache memory

secondary memory data access from cache



Using the cache



- ① memory organisation
 - ② mapping mechanism
 - ③ Replacements Algo.
 - ④ updating techn.
 - ⑤ multi level cache org

- www.yourdomain.com

- ① Memory organisation : describe cache organisation
 - ALU to memory hierarchy design, Data will be transferred from main memory to cache memory in the form of blocks

so both of the
curve block size

Address	CM SR	Cache	main
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			
26			
27			
28			
29			
30			
31			
32			
33			
34			
35			
36			
37			
38			
39			
40			
41			
42			
43			
44			
45			
46			
47			
48			
49			
50			
51			
52			
53			
54			
55			
56			
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			
69			
70			
71			
72			
73			
74			
75			
76			
77			
78			
79			
80			
81			
82			
83			
84			
85			
86			
87			
88			
89			
90			
91			
92			
93			
94			
95			
96			
97			
98			
99			
100			
101			
102			
103			
104			
105			
106			
107			
108			
109			
110			
111			
112			
113			
114			
115			
116			
117			
118			
119			
120			
121			
122			
123			
124			
125			
126			
127			
128			
129			
130			
131			
132			
133			
134			
135			
136			
137			
138			
139			
140			
141			
142			
143			
144			
145			
146			
147			
148			
149			
150			
151			
152			
153			
154			
155			
156			
157			
158			
159			
160			
161			
162			
163			
164			
165			
166			
167			
168			
169			
170			
171			
172			
173			
174			
175			
176			
177			
178			
179			
180			
181			
182			
183			
184			
185			
186			
187			
188			
189			
190			
191			
192			
193			
194			
195			
196			
197			
198			
199			
200			
201			
202			
203			
204			
205			
206			
207			
208			
209			
210			
211			
212			
213			
214			
215			
216			
217			
218			
219			
220			
221			
222			
223			
224			
225			
226			
227			
228			
229			
230			
231			
232			
233			
234			
235			
236			
237			
238			
239			
240			
241			
242			
243			
244			
245			
246			
247			
248			
249			
250			
251			
252			
253			
254			
255			
256			
257			
258			
259			
260			
261			
262			
263			
264			
265			
266			
267			
268			
269			
270			
271			
272			
273			
274			
275			
276			
277			
278			
279			
280			
281			
282			
283			
284			
285			
286			
287			
288			
289			
290			
291			
292			
293			
294			
295			
296			
297			
298			
299			
300			
301			
302			
303			
304			
305			
306			
307			
308			
309			
310			
311			
312			
313			
314			
315			
316			
317			
318			
319			
320			
321			
322			
323			
324			
325			
326			
327			
328			
329			
330			
331			
332			
333			
334			
335			
336			
337			
338			
339			
340			
341			
342			
343			
344			
345			
346			
347			
348			
349			
350			
351			
352			
353			
354			
355			
356			
357			
358			
359			
360			
361			
362			
363			
364			
365			
366			
367			
368			
369			
370			
371			
372			
373			
374			
375			
376			
377			
378			
379			
380			
381			
382			
383			
384			
385			
386			
387			
388			
389			
390			
391			
392			
393			
394			
395			
396			
397			
398			
399			
400			
401			
402			
403			
404			
405			
406			
407			
408			
409			
410			
411			
412			
413			
414			
415			
416			
417			
418			
419			
420			
421			
422			
423			
424			
425			
426			
427			
428			
429			
430			
431			
432			
433			
434			
435			
436			
437			
438			
439			
440			
441			
442			
443			
444			
445			
446			
447			
448			
449			
450			
451			
452			
453			
454			
455			
456			
457			
458			
459			</td

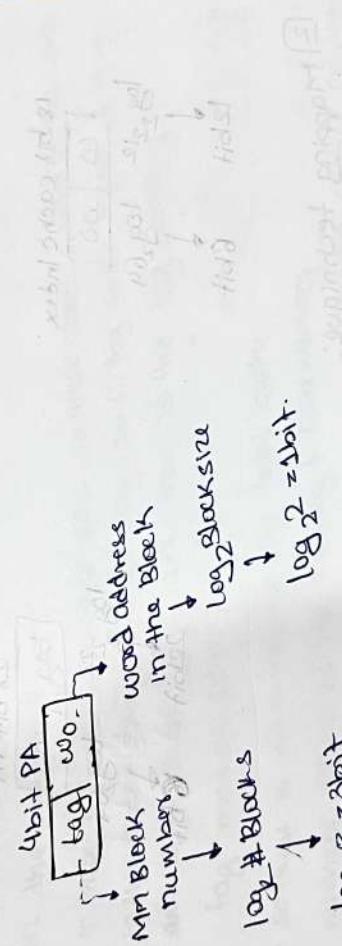
Address formats

CM size = 88 cache = $\log_2 8 = 3$ bit
Sindex



$$\log_{10} \# \text{lines} = 2.64 \quad \log_{10} \text{stack size} = 2.61$$

mm 818 - 819 Dusitakorn (พญาดุสิต) = 102616271627



Q) consider a hypothetical CPU which contains 256 KB cache & 256 MB main memory. memory system is organised into a 64B blocks.

(a) How many line & blocks are present in the respective memories.

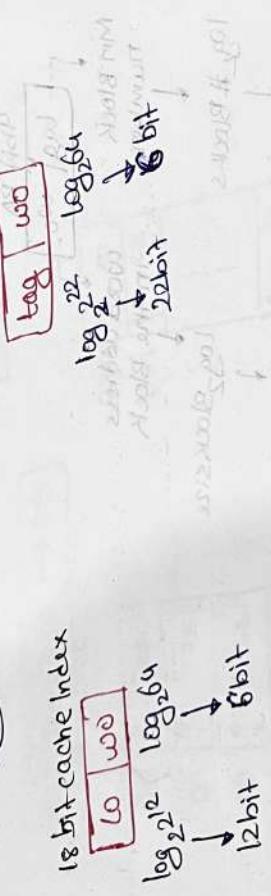
(b) Show the address formats.

$$\text{Sol: } \# \text{line} = \frac{\text{main size}}{\text{block size}}$$

$$= \frac{256 \text{ K}}{64} = \frac{2^{18}}{2^6} = 2^{12}$$

$$\# \text{block} = \frac{\text{main size}}{\text{block size}}$$

$$= \frac{256 \text{ M}}{64} = \frac{2^{28}}{2^6} = 2^{22}$$



2) Mapping technique:

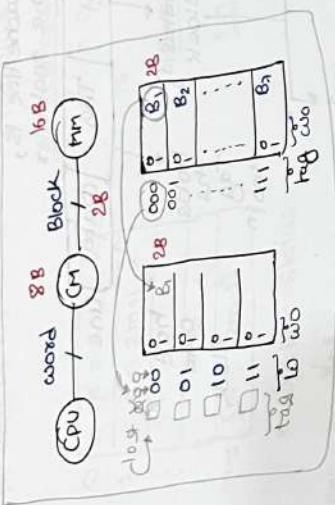
Process of copying the data from main memory to cache memory

Is called as mapping

In the mapping process data block is copied into cache memory along with tag.

- to hold the tag information, extra storage space is required in memory line called as tag-space.

Overview



- along with a block tag is also moving in cache (Data moves along with address)
- address less data use less data
- But here tag bit is 30bit & line offset is only 10bit
- so 10 bits is lost

therefore in cache memory design there is a need of extra storage space in every line how many extra space required depend

on the tag size

- In there we need 1bit more space, so these spaces is called as tag space
- In the cache memory . so only 1bit tag space require & there are 4 such line . so $4 \times 1 = 4$ bit is the tag directory
- tag memory size
- software is extra memory, so total cache memory equal to tag memory + Data memory
- these address (LO) is implemented using decoders circuit. (Bit logical)
- so tag & wo are physical space.

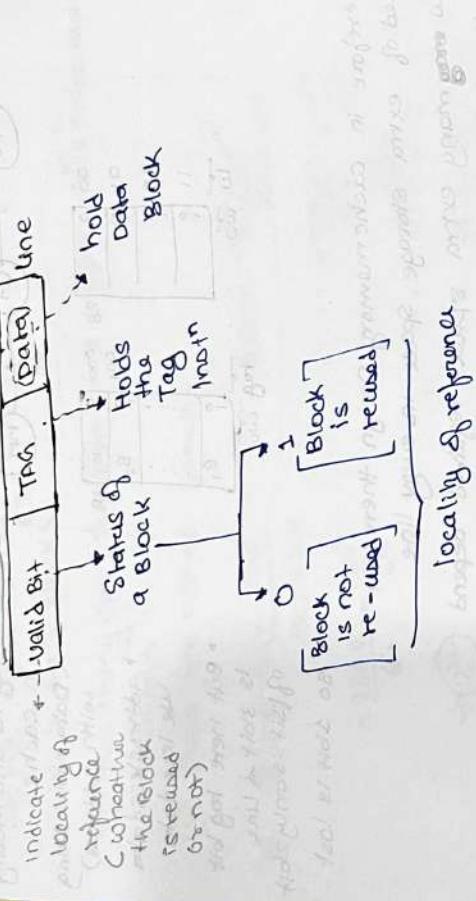
tag mem. size = #lines * tag spaces in CM in the line

- Data mem. size = #line * Block size in CM

$$\text{Total} = \frac{\text{Tag memory}}{\text{Cache size}} + \frac{\text{Data memory}}{\text{mem. size}}$$

fixed
various depends on type of mapping used

- contents of a cache line is, cache controller



- 3 kind of mapping technique is used in cache design
 - ① Direct mapping
 - ② Associative mapping
 - ③ Set-Associative memory

Direct mapping Associative mapping Content addressable memory

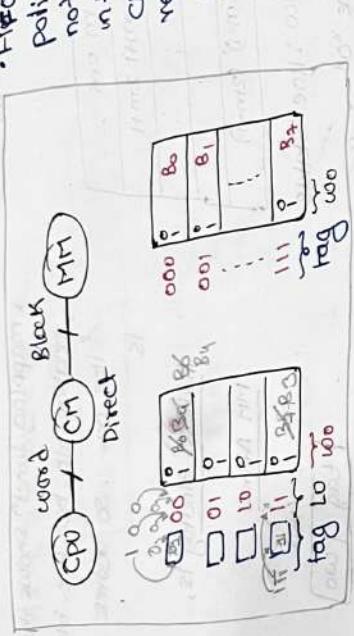
• In these techniques, most functions are

Monsoon (मौसम) → Rainy season
→ Monsoon → Rainy season
→ Monsoon → Rainy season
→ Monsoon → Rainy season

- ~~• $\text{mod}_4 = 0$~~
- ~~• $\text{mod}_4 = 1$~~
- ~~• $\text{mod}_4 = 2$~~
- ~~• $\text{mod}_4 = 3$~~
- ~~• $\text{mod}_4 = 0$~~
- ~~• $\text{mod}_4 = 1$~~
- ~~• $\text{mod}_4 = 2$~~
- ~~• $\text{mod}_4 = 3$~~

↳ 16 combinations

- **FIFO & LRU**
Policies are
not required
in the Direct
cache to
replace the
data block



Assume,
 CPU generate the following MM block reference during
 the prog. exec.
 i.e. MM block ref.: $B_0, B_1, B_2, B_3, B_4, B_5, B_6, B_7$
 . CM is empty
 Accessing

Conflict miss

Unique block reference w/
 CM is empty

Block reference w/
 CM is empty

unique

(the first time when
 the block is
 first referred)

Compulsory miss (the first time when
 the block is
 first referred)

Chromophore Conflict →
Technique Conflict miss

CH MOD N = 3
0 mod 4 = 0 ;
t mod 4 = 2 ;
B₀ - miss ;
B₂ - miss .

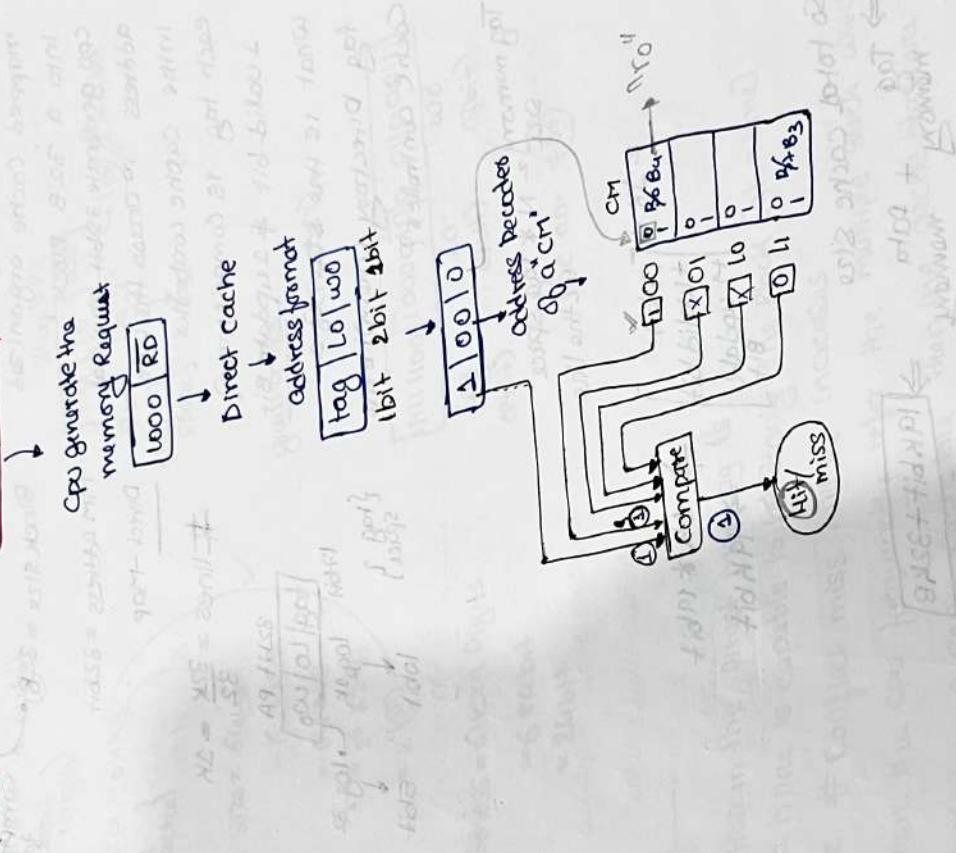
$B_0 - \text{Hit}$	$B_1 - \text{Hit}$	$B_2 - \text{miss}$	$B_3 - \text{miss}$	$B_4 - \text{miss}$	$B_5 - \text{miss}$
$0 \bmod 4 = 0$	$1 \bmod 4 = 1$	$2 \bmod 4 = 2$	$3 \bmod 4 = 3$	$4 \bmod 4 = 0$	$5 \bmod 4 = 1$

- (adjective) **mapping** itself is a **mapping** replacement no need of replacement policies in Direct mapping

2100
日十三

188

Accessing : We know: $\text{mod} \leftarrow [0 \dots 5000] ; r_0 \leftarrow M[000]$



- Sol: Given CM:Size = 82KB both are same
 Cache Block size = 32B
 MM Address = 32bit
 Direct-MAP
- Quesn: Consider 32KB direct mapped cache organized into a 32B block. CPU generates 32bit physical address to access the data.
- In the cache controller each tag is comprising of a valid bit + 1 update bit what is the size of a tag directory (number of cache controllers)
- $\therefore \text{Tag memory size} = N * \text{tag space in the line}$
- $\Rightarrow 1K * 10bit + 1\text{ update bit} \Rightarrow 19bit$
- So total cache size $\Rightarrow \text{Tag memory} + \text{data memory} \Rightarrow 19bit + 32KB$
- Q2 consider a 64bit hypothetical CPU with 64KB direct mapped Cache organized into a 64 word block. To which cache line in decimal the following memory address is mapped
 $0xFC8463$.

Sol:

Given
word size = 64 bit
CPU control length

Cache
Data
Cache

$$\begin{aligned} \text{Block size} &= 64 \text{ bit} \\ \text{Cache size} &= 64 \text{ KB} \\ \text{Cache address} &= 6 * 64 \\ &= 64 * 64 \\ &= 2^6 * 2^6 \\ &= 2^{12} \end{aligned}$$

MM Address = Hexa Digits

$$= 6 * 64$$

$$= 24 \text{ bit}$$

$$= 24 / 6 = 4 \text{ blocks}$$

$$\# \text{line} = \frac{\text{blk}}{2^6} \rightarrow \frac{2^6}{2^6} = 2^0$$

$$\begin{array}{|c|c|c|} \hline \text{tag} & 10 & \text{wo} \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline \text{tag} & 10 & \text{wo} \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline \text{tag} & 10 & \text{wo} \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline \text{tag} & 10 & \text{wo} \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline \text{tag} & 10 & \text{wo} \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline \text{tag} & 10 & \text{wo} \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline \text{tag} & 10 & \text{wo} \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline \text{tag} & 10 & \text{wo} \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline \text{tag} & 10 & \text{wo} \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline \text{tag} & 10 & \text{wo} \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline \text{tag} & 10 & \text{wo} \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline \text{tag} & 10 & \text{wo} \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline \text{tag} & 10 & \text{wo} \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline \text{tag} & 10 & \text{wo} \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline \text{tag} & 10 & \text{wo} \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline \text{tag} & 10 & \text{wo} \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline \text{tag} & 10 & \text{wo} \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline \text{tag} & 10 & \text{wo} \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline \text{tag} & 10 & \text{wo} \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline \text{tag} & 10 & \text{wo} \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline \text{tag} & 10 & \text{wo} \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline \text{tag} & 10 & \text{wo} \\ \hline \end{array}$$

[Note]

Limitation in the direct mapping is

1. each line is capable to hold only one block [at a time]
i.e. # conflict miss will be increases.
2. when the CPU frequency refer the multiple block which are mapped into a same cache line then the cache block area continuously blocking so hit ratio will be dropping down. So these phenomena is known as thrashing.

Ex: Assume,
MM block ref: B₀, B₁, B₀, B₁, ...

mod function
give line no.

cache empty

Accessing



Accessing B₀-miss, 0 mod 4 = 0
B₁-miss, 1 mod 4 = 1
B₀-miss, 0 mod 4 = 0
B₁-miss, 1 mod 4 = 1
B₀-miss, 0 mod 4 = 0

- due to those mod function thrashing is present.

so when cache memory is design with address (line no.) then compounding we need to use mod function. so thrashing is possible.

- so to avoid it, Design the cache without address

In direct mapping there is no need of mapping policies b/c of mod function.

mod function clear say the relationship So now we need 3 bit for each block b/w block no. & line no. so no need of replacement policy in Direct mapping.

[log space increase]

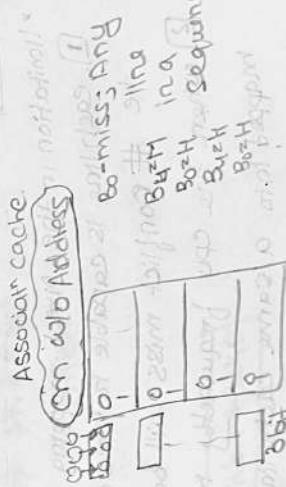
so cost increase

so Alternative required
[Set - Associative cache]

Hit ratio

so, Alternative required

re Associative cache



so cost increase

so Alternative required

[Set - Associative cache]

Associative caching

- These cache is design without address called as Content addressable memory
- In these cache design any memory block can be mapped into any cache memory line in a sequential manner logic is used to map the data

so address breaking is

MM address : $\boxed{\text{Tag}} \boxed{\text{100}}$

$\boxed{\text{101}}$

$\boxed{\text{102}}$

$\boxed{\text{103}}$

$\boxed{\text{104}}$

$\boxed{\text{105}}$

$\boxed{\text{106}}$

$\boxed{\text{107}}$

$\boxed{\text{108}}$

$\boxed{\text{109}}$

$\boxed{\text{110}}$

$\boxed{\text{111}}$

$\boxed{\text{112}}$

$\boxed{\text{113}}$

$\boxed{\text{114}}$

$\boxed{\text{115}}$

$\boxed{\text{116}}$

$\boxed{\text{117}}$

$\boxed{\text{118}}$

$\boxed{\text{119}}$

$\boxed{\text{120}}$

$\boxed{\text{121}}$

$\boxed{\text{122}}$

$\boxed{\text{123}}$

$\boxed{\text{124}}$

$\boxed{\text{125}}$

$\boxed{\text{126}}$

$\boxed{\text{127}}$

Tag memory size

$N * \text{tag space}$

so, total cache size

= tag memory + Data

$\Rightarrow 12\text{bit} + 88$

associatn :

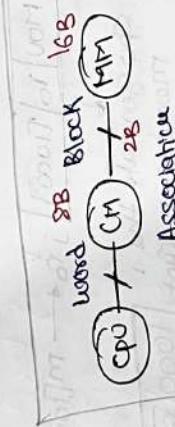
$\boxed{\text{CM}}$

{tag}

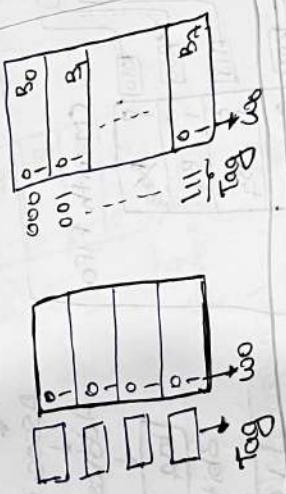
space

$\Rightarrow 4 * 3bit$

$\Rightarrow 12bit$



Associatn



- FIFO & LRU Policies are required to replace the data block when the cache is full.

Advantage:

Fastest cache Block Address Decoders
Latency not present

so, Alternative required

i.e. Set-Associative cache

Set Associative Cache

- these cache designs is used to compromise the disadvantages in they Direct & Associate cache design.
- In these cache designs, line are grouped into set, to accommodate multiple block in a set

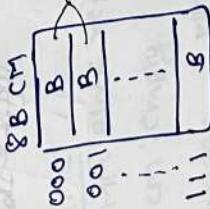
$$\# \text{sets}(\Sigma) = \frac{N}{P \text{ways}}$$

N: # lines in CM
P: # lines in the set

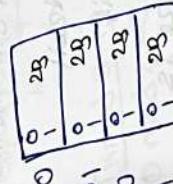
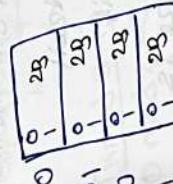
Disadvantage:
Expensive cache Block, Tag
memory size ↑

Consider 2-way set
Before orgⁿ

After organized
into arrays



$$\# \text{lines}(N) = \frac{E}{2} = 4$$



[correct CM]

After organized
into a set

$$\# \text{sets}(\Sigma) = \frac{N}{P \text{ways}} = \frac{4}{2} = 2$$

Consider 2-way set
After orgⁿ

cm. w/o Address



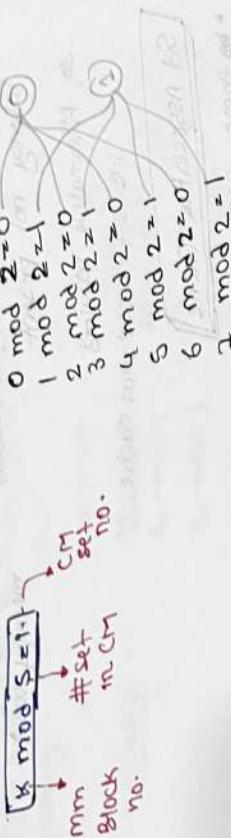
{ Asso.
cache }



{ S. rows
P. cols }

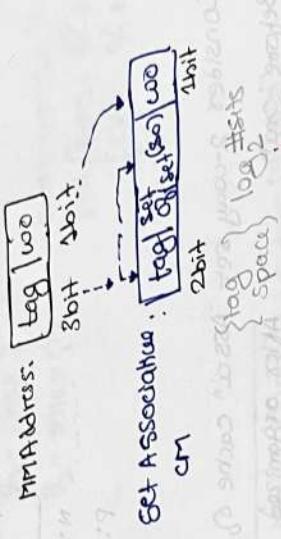
{ S. rows
P. cols }

In the cache design mod function is used to map the data



mapping function shows the memory block no. & cache memory set no.

i.e Address binding is



so, total cache size is

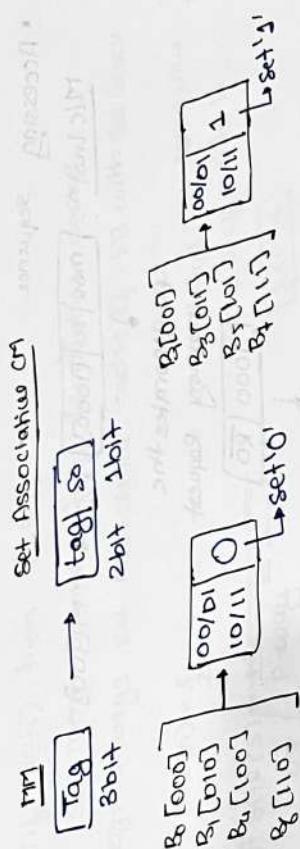
Tag + memory + data memory

8bit + 8bit

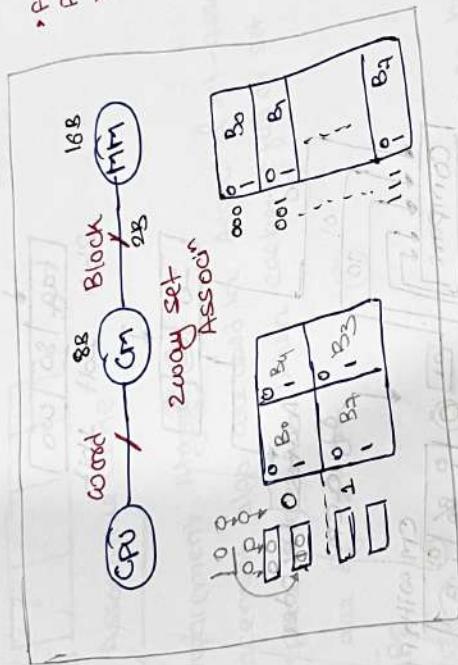
$$\frac{1}{2} \times 2^{32} \#$$

$$= \frac{1}{2} \times 2^{32}$$

2^32 #



- FIFO LRU policies are required to replace the data block when the set is full.



Assume current block ref.: B0, B4, B0, B4, B3, B2

cm : empty
Accessing:

min block mod size :
B0 - miss ; 0 mod 2 = 0 ; set '0'
B4 - miss ; 1 mod 2 = 1 ; set '1'

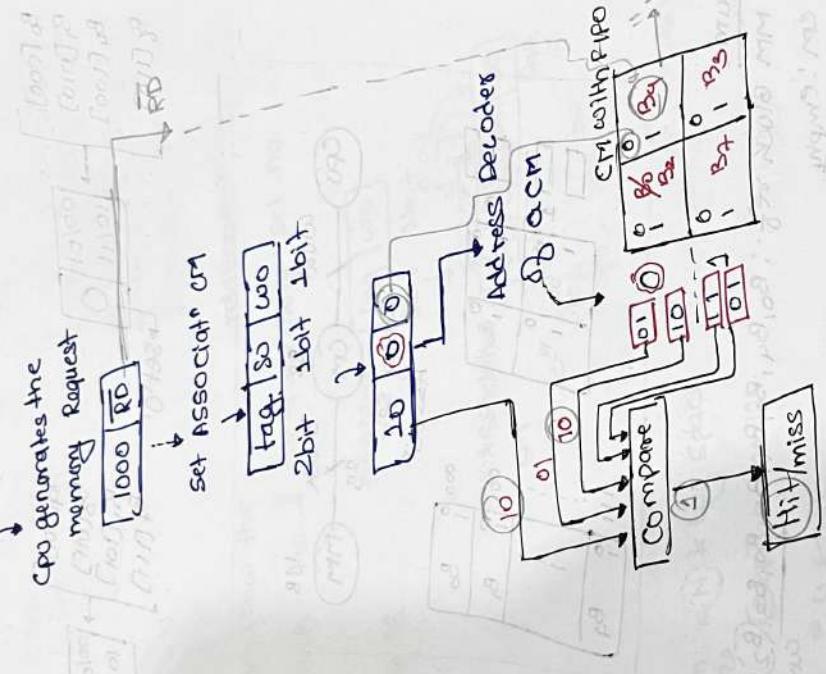
B0 - hit ; 0 mod 2 = 0 ; set '0'
B4 - hit ; 1 mod 2 = 1 ; set '1'
B0 - miss ; 0 mod 2 = 0 ; set '0'
B4 - miss ; 1 mod 2 = 1 ; set '1'

[FIFO : B6 B2]
LRU : B6 B2

- Accessing Sequence

MIC history: $\text{move}_0[0000]$; $v_0 \rightarrow m[1000]$

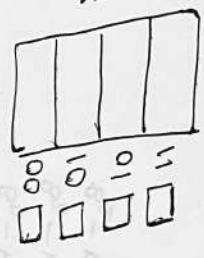
→ CDO Governmental type



Note → If $P \geq 1$ then $S \in N$
So, memory became direct cache

Ex:- 1-way set associative cache of 8B with 2B Block

$$\# \sin(\theta) = \frac{N}{P - \cos\theta} = \frac{4}{1}$$



23A

② If $(P \in N)$ then $(S \leq 1)$

so mem. becomes fully associative cache
e.g.: 4-way set associative cache of 8B with 3B block

$$\# \text{line}(N) = \frac{S}{2} = 4$$

sets $(S) = \frac{N}{B} = \frac{8}{4} = 2$

$$P-\text{assoc} = \frac{B}{L} = 1$$



Associative Cache

3 Replacement Algorithms

- Replacement algo. are used in "Assoc" & "set-assoc" designs,
to replace the data when cache is full or set is full.
these are of 2 kind.

① FIFO

② LRU

longest time

- In FIFO, replace the block in cache which is having longest time stamp
- In LRU, replace the block which is present in the cache longest time without reference



③

2nd Set 1 11010101 11010100 11010100

3rd Set 1 11010101 11010100 11010100

4th Set 1 11010101 11010100 11010100

→ 3rd Set 1 11010101 11010100 11010100

→ 2nd Set 1 11010101 11010100 11010100

→ 1st Set 1 11010101 11010100 11010100

→ 0th Set 1 11010101 11010100 11010100

- Q) Consider 8-way set associative Cache of 64x8B organized into a 512-B blocks cache main memory

Data is a subset of 2²¹ bytes of a memory space. In the cache Controller each tag is comprising of a valid bit, 1 update bit & 2 replacement bit

⑥ What is the size of memory? 50KB

⑦ What is the size of a total cache?

⑧ What is the size of a total cache?

⑨ What is the size of a total cache?

Sol: given: cm: $\text{GK}(\vec{B})$

Block S120 = 32(B)

$MM \approx 2^{34} B$ [Cache memory is a subset of main mem.]

$$\text{Physical} = \log_{256}$$

• secondary stage - more
of older individuals -
more social support

१०८२४३८५१॥

$\text{states}(S) = \frac{N}{\text{present.}} = \frac{2^{\frac{N}{2}}}{3} \Rightarrow \frac{2^{\frac{N}{2}}}{2^3} \Rightarrow (2^{\frac{N}{2}})^3$



- (9) Tag mom. = N * tag space in the line

$$= 2^{21} * (21 \text{ bit} + 1 \text{ valid bit} + 1 \text{ update bit} + 2 \text{ replacement bit})$$

$$\Rightarrow 2^{21} * 25 \text{ bit}$$

→ 50 Kbits

Q) Consider a block cache initially empty with the following min block reference.

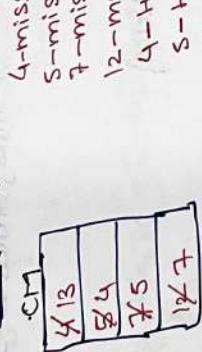
4, 5, 7, 12, 4, 5, 13, 4, 5, 7.

Identify the hit ratio using.

- (a) FIFO
- (b) LRU
- (c) Direct mapped cache
- (d) 2-way set assoc. cache with LRU.

Sol:

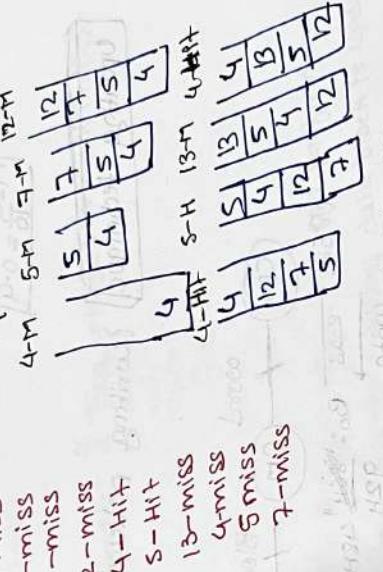
(b) FIFO. [Sequence]



$$H = \frac{2}{10} = 0.2$$

Hit ratio.

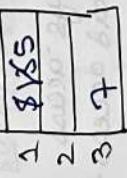
(c) Top.



13-miss.



4-miss
5-Hit
7-miss.

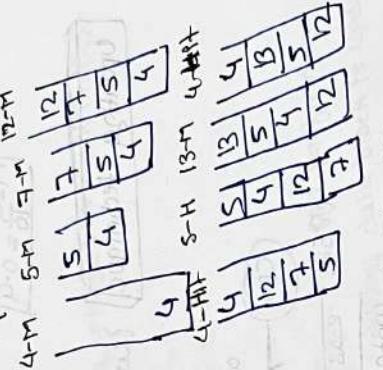


K mod N=1

$$\begin{aligned} & 13 \mod 4 = 1 \\ & 4 - \text{Hit} ; 5 \mod 4 = 1 \\ & 12 - \text{miss} \mod 4 = 0 \\ & 4 - \text{Hit} \\ & 5 - \text{Hit} \\ & 7 - \text{Hit} \end{aligned}$$

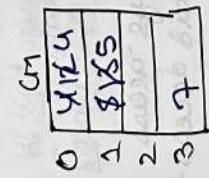
Hit ratio = 2/10 = 0.2

(d) recent references in the cache.

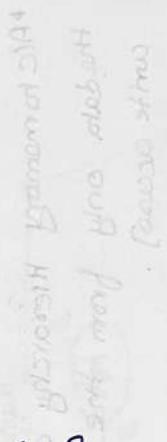


13-miss.

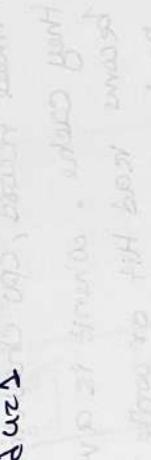
Direct cache



4-miss
5-miss
7-miss
12-miss
4-Hit
5-Hit
13-miss
4-miss
5-miss
7-miss



Recent references



Hit ratio = 2/10 = 0.2

2-way set - associative $4 \times 4 \text{ mod } 2 = 0$
 $S = \frac{N}{2}$ ways $\rightarrow \frac{N}{2} = 2$

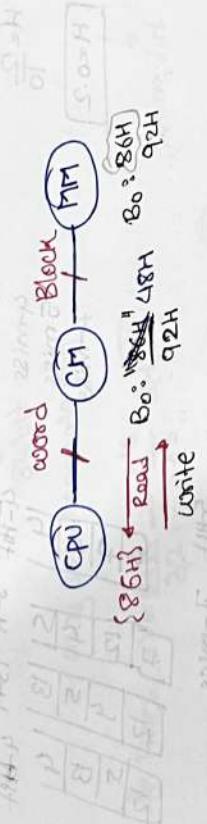
0	4 12
1	5 13

$4 \times 4 \text{ mod } 2 = 1$
 $7 - 1 ; 9 \text{ mod } 2 = 1$
 $12 - 1 ; 12 \text{ mod } 2 = 0$
 UNIT
 SHIT

$13 - \text{Miss} ; 13 \text{ mod } 2 = 1$
 $7 - \text{miss} ; 7 \text{ mod } 2 = 1$

$\frac{10}{10} = 0.1$

Updating technique { writing policies }



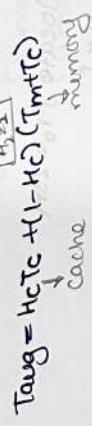
Some address contain different
 value at different place that is
 data inconsistency [data coherence]

- Alchromony hierarchy design, cache always access data only from the cache memory [read access] write access]

In future research, can check the availability of data block in
 they cache. When it is available in cache than the operation
 becomes read hit or write otherwise operation becomes
 read-miss or write miss

- when the miss occurs then the respective blocks are copied from the main memory to cache memory called as read allocate or write allocate.
- after the allocation, we perform the Read & write operation only on a cache memory.

$$T_{avg} = H_1 T_1 + (1-H_1) (T_2 + \tau_1) \rightarrow \text{so memory change}$$

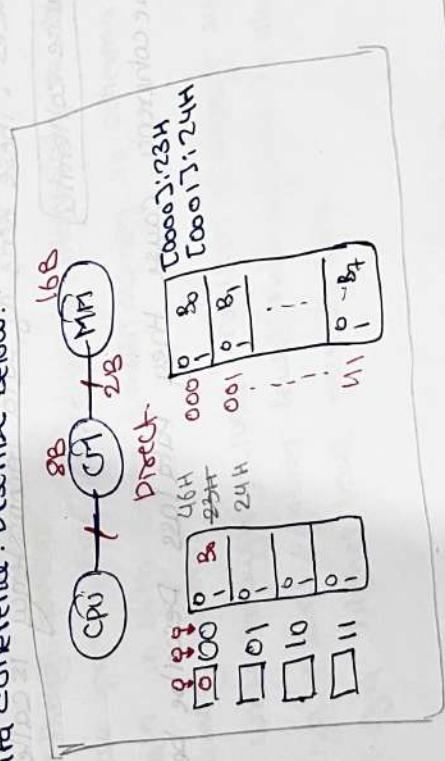


$$T_{avg} = H_1 T_C + (1-H_1) (T_M + T_C) \quad \& \quad T_{avg} = H_1 T_C + (1-H_1) (T_M + \tau_1)$$

write cycle time

read cycle time

- in the memory hierarchy design, when the CPU performs the write operation then only the cache memory data block is updated while others than only the cache memory data block is updated in the main memory.
- the corresponding blocks are not updated in the main memory, i.e. same address contains different value at different places, thus memory is known as data coherence. Describe below.



Code:

I₁: mov r₀, [B₀ 0000] ; R₀ - 0th Byte → r₀

I₂: Add r₀, #23H ; r₀+23 → r₀

I₃: mov [B₀ 0000] ; r₀; r₀ → (R₀ - 0th byte)

Write operation

Blocks aligned on cache boundary

Execution

I₁: R₀ - Read miss; Read allocate'; r₀=23H

$$K \bmod N = 1$$

$$0 \bmod 4 = 0$$

(Memory location) + offset = address = 23H + 0 = 23H

I₂: r₀+23H → r₀; r₀=46H

I₃: R₀ - write hit ; 46H

[0000]: 23H

Set memory location 0000 to value 23H. (Memory content 23H)

- During the execution of I₃ "I₀" instruction update the memory only the corresponding block is not updated in the mem.
- Same Address contain diff value at different places. These kind of data inconsistency is called as Cache inconsistency

Cache coherence cause they data loss described below.



In: Mov R0, [0000] ; R0 - 0th byte \rightarrow R0
Is: R0 / B4

Ts:

Tc:

Ts: Mov R0, [0000]; R0 - 0th byte \rightarrow R0

Exn: T0: R0 - Read ; Read allocated memory
Is: miss $[k \bmod N = 1]$
 $\{ k \bmod 4 = 0 \}$

Cn: R0 : 464
R0 28A

- So, updated data is lost :: Is incorrectly accessed.
the old value from the block "R0" (data loss).

• To handle the above situation, updating techniques are used in the cache design. These are of 2 types.

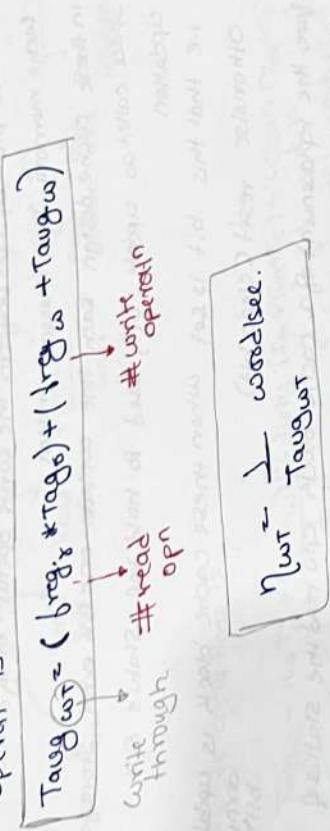
- ① Write through protocol.
- ② Write back protocol

Simultaneously update in both memory & cache
In both memory & cache, RAM write bit is set.

- ④ Write through protocol.
- In these protocol, CPU performs the simultaneous write operation in both cache & main memory. So coherence in the memory system.
- In the write through protocol, inclusion is always successful.
- In the write back protocol, inclusion means lower level memory data is always included in higher level memory data.

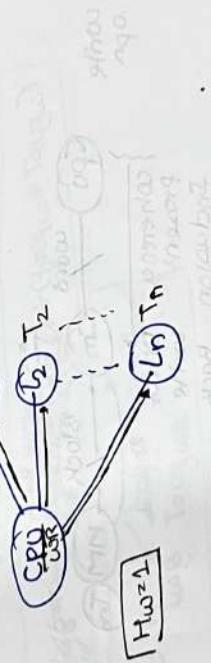
avg. access time when considering both read & write

Operations



Note:

- When write through protocols is implemented in the system, $H_w=1$.
- Simultaneous access memory organisation then always become 1. block size need not be required.
- In the 1st level cache memory.



$$Ta_{avg} = H_1 T_1 + (1-H_1) H_2 T_2 + \dots$$

$$Ta_{avg} = T_m$$

Note: When the question contains cache miss, write through when the question contains simultaneous access.

when $H_w=1$ then use protocol

otherwise ($H_w \neq 1$) use memory org formula

Hierarchical org!

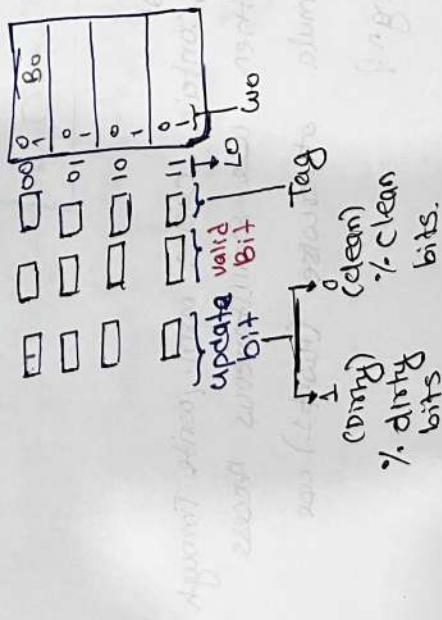
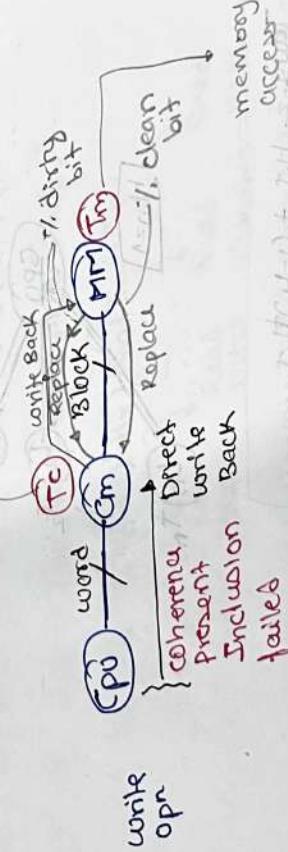
Read

Write Back (word) protocol.

- In these protocol, CPU performs the write operation only in the cache memory. So coherency present.
- In these cache design each line contains one bit extra storage space called as update bit, used to hold the status of a update bit is set when the cache block is updated i.e. that this bit is set when the cache block is updated otherwise reset (clean bit)

- Before the replacement of a cache block CPU reads the status of update bit to perform the write back operation i.e. when update bit is "1" then CPU writes back the cache block into main memory. later replace the cache block with new block otherwise simple replace the cache block with new block.
The cache block :: data is safe.

Cache memory (in word wise)



238

• Read cycle time is,

$$T_{avg} = H \cdot T_C + (1-H) \left[\text{dirty bit} (T_m + T_m + T_C) + \text{not dirty bit} (T_m + T_m + T_C + 1 \cdot \text{clean bit}) \right]$$

• write cycle time is,

$$T_{avg} = H \cdot T_C + (1-H) \left[\text{dirty bit} (T_m + T_m + T_C) + \text{not dirty bit} (T_m + T_m + T_C + 1 \cdot \text{clean bit}) \right]$$

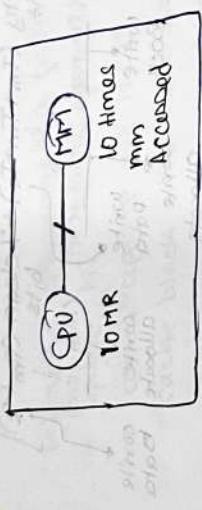
• Avg mem. Access time
when consider both read & write ops is,

$$T_{avg_avg} = (\text{frequency} * T_{avg_r}) + (\text{frequency} * T_{avg_w})$$

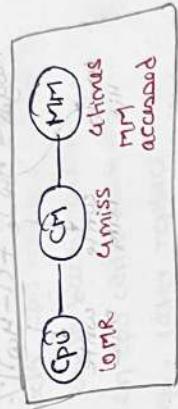
$$\eta_{avg} = \frac{1}{T_{avg_avg}} \text{ word/sec}$$

- 5) Multi-level cache org
- multi-level cache organization is used in the system design to reduce miss penalty
 - miss penalty means time required to transfer the data from higher level to lower level when there is a miss in lower level memory

Sys. Design w/o cache

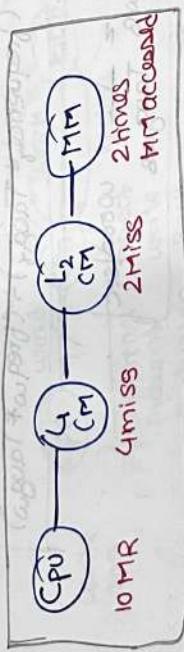


System design with cache



②

- Sys. Design with multilevel caches.
- used for more hits & reduce main memory access.



- In the multi-level cache orgn L₁cm size < L₂ cm size

L₁cm Access time < L₂ cm access time

CPU to L₁ > L₂ cm to CPU
Association > Association

① go
miss
ac

Two kind of miss rate calculated in multilevel cache orgn

$$\text{① Global miss rate (GMR)} = \frac{\# \text{misses in cache}}{\text{Total # references generated by the CPU}}$$

$$\text{② Local miss rate (LMR)} = \frac{\# \text{misses in cache}}{\text{Total # accesses to the cache.}}$$

wrt above sys.

$$\begin{aligned} \text{GMR}_L &= \frac{4}{10} = 0.4 \\ \text{GMR}_M &= \frac{2}{10} = 0.2 \\ \text{LMR}_L &= \frac{\# \text{misses in LCM}}{\# \text{misses in LCM}} \\ &= \frac{2}{4} = 0.5 \end{aligned}$$

$$\text{GMR}_L = \text{LMR}_L$$

Avg. memory access time is calculate in term of hit time,
local miss rate \neq miss penalty is as follows.

Hit ratio these is relative value

2-level caches,

$$\text{Avg.} = \text{hit time}_L + (\text{miss rate}_L \times \text{miss penalty}_L)$$

$$\text{miss penalty}_L = \text{hit time}_L + (\text{miss rate}_2 \times \text{miss penalty}_L)$$

$$\text{miss penalty}_L = \text{TM access time.}$$

Avg. memory stall / instrn is calculated as:

$$\frac{\text{no. misses}}{\text{instrn}} \times \left[\frac{\text{# misses in L1/miss} + \text{# misses in L2}}{\text{no. misses per instrn}} \right] \times \text{extra time}$$

$$+ (\text{# misses in L2 per miss}) \times \text{instrn}$$

↓
misses
per instrn

(extra time)

Type of cache misses

① Compulsory miss: [cold-start miss / first ref. miss]

- these miss will occur when very 1st line referenced block in the program is not in the cache memory. these misses can be minimized by increasing the block size

② Capacity miss

- these miss will occur when the cache is full, these misses can be minimized by increasing the cache size

③ Conflict miss [Collision miss @ Inference miss]

- these miss will occur when too many block are mapped into a same cache line or same cache set these miss can be minimized by doubling the associativity of cache.

Question.

- ① Consider cache memory designed with write through protocol having the access time of 60ns main memory access time is 400ns. Hit ratio of a read request is 70%. System generates 80% read requests & remaining are used for write opn. What is the Avg memory access time when considers both Read & write operatn?

Sol: given

$$\begin{aligned} \cdot CM \\ \cdot CM \\ \cdot T_c = 60ns \\ \cdot T_m = 400ns \\ \cdot H_r = 70\% \\ \cdot F_r = 80\% \\ \cdot F_w = 20\% \\ \cdot H_w = \text{not given} \end{aligned}$$

$$T_{avg} = H_r T_c + (1-H_r) T_m \text{ simultaneous mem. org.}$$

$$\begin{aligned} &= (0.7 * 60) + (1-0.7) 400 \\ &\Rightarrow 42 + 120 \\ &\Rightarrow 162 ns \end{aligned}$$

$$T_{avg} = (F_r * T_c) + (F_w * T_m)$$

$$\begin{aligned} &= (0.8 * 162) + (0.2 * 400) \\ &= 129.6 + 80 \\ &= 209.6 ns \end{aligned}$$

num. orgn become
Simultaneous Access orgn

- ② Consider a cache memory having hit. ratio for read & write operatn is 70% & 80%. respectively cache main access time is 40ns. Main memory access time is 80ns. When miss occurs in the cache the LRU block is copied from MM to CM. System generates 40% write request & remaining are used for read. What is the average access time?

what is the throughput (Q) bandwidth (B) of a memory
 in terms of million words/sec when the cache
 is designed with

- (a) write through
- (b) write back.

Sol:

$$H_w = 70\%$$

$$H_m = 80\% \cdot (H_w + 1)$$

~~200ns/word~~ Hierarchical
 80ns/word main
 org

$$T_c = 400ns \left\{ \begin{array}{l} \text{block access} \\ \text{time} \end{array} \right\}$$

$$T_m = 800ns \left\{ \begin{array}{l} \text{word} \\ \text{access} \end{array} \right\}$$

$$\text{Block size} = 4KB$$

$$br = 60\% \cdot (0\% \text{ clean bits})$$

$$br = 40\% \cdot (1\% \text{ dirty bit})$$

$$T_{avg,w} = 160ns$$

$$T_{avg,m} = 200ns$$

$$\Rightarrow 200ns$$

$$T_{avg,w} = H_w T_w + (1-H_w)(T_m + T_w)$$

$$= (0.8 \cdot 200) + (1 - 0.8)(800 + 200)$$

$$= 160 + 200 \Rightarrow 360ns$$

$$T_{avg,w} = (f_w * T_w) + (f_m * T_m)$$

$$= (0.6 * 280) + (0.4 * 360)$$

$$\Rightarrow 168 + 144 = 312ns$$

$$n_{cut} = \frac{1}{T_{avg,w}}$$

$$\Rightarrow \frac{1}{312} = 0.003205$$

T_{avg}

$$Tau_{avg} = 1.6T_c + (1 - 14) \left[0.4 \cdot \text{dirty}(\tau_m + \tau_m + \tau_c) + 0.1 \cdot \text{clean}(\tau_m + \tau_c) \right]$$

$$\Rightarrow (0.7 * 40) + (1 - 0.7) \left[0.4 \cdot (800 + 800 + 40) + 0.1 \cdot (800 + 40) \right]$$

$$\Rightarrow 28 + 0.3 [656 + 504]$$

$$\Rightarrow 28 + 348 \Rightarrow \boxed{376}$$

$$Tau_{avg} = 1.6T_c + (1 - 14) \left[0.4 \cdot \text{dirty}(\tau_m + \tau_m + \tau_c) + 0.1 \cdot \text{clean}(\tau_m + \tau_c) \right]$$

$$\Rightarrow (0.8 * 40) + (1 - 0.8) \left[0.4 \cdot (800 + 800 + 40) + 0.1 \cdot (800 + 40) \right]$$

$$\Rightarrow 32 + 0.2 [656 + 504] \Rightarrow \frac{82 + 282}{\boxed{264}}$$

$$Tau_{avg} = (f_r * Tau_{avg}) + (f_w * Tau_{avg})$$

$$= (0.6 * 376) + (0.4 * 264)$$

$$= 225.6 + 105.6$$
~~$$= 331.2$$~~

$$\Rightarrow \boxed{331.2}$$

$$\frac{1}{10^6} \text{ word/sec.} \Rightarrow 0.003312 \text{ sec}$$

- Q) Consider system with 2 level cache organization having the hit time of the cache lines 40ns & 40ns respectively. Miss penalty of L₂ cache is 200ns. System generates 60 memory references. So misses in L₁ cache & 30 misses in L₂ cache present. What is the Avg. memory access time.

the

Hit time₁₂ = Hit time₁₂ + Miss time₁₂

miss penalty₁₂ = 200 ns

$$LNR_4 = \frac{50}{600} = 0.0833$$

$$T_{avg} = \text{Hit Time}_4 + \text{Miss Rate}_4 \times [\text{Hit Time}_4 \text{ (miss & penalty)}]$$

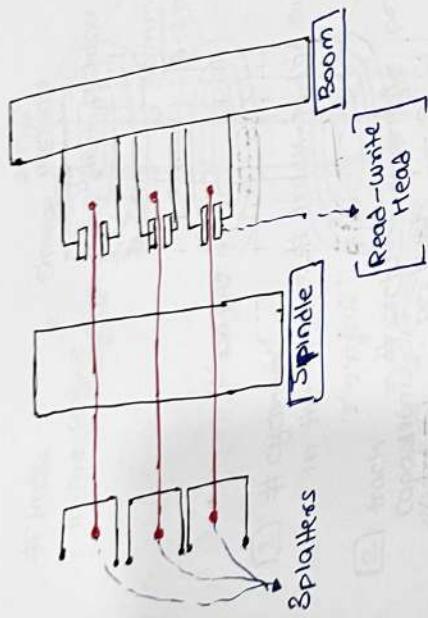
$$\Rightarrow 10ns + 0.0833 \left[\text{miss} \times \frac{0.6 * 200}{100} \right]$$

$$\Rightarrow \frac{10ns + 13.328}{160} = 23.328 \text{ ns.}$$

Harddisk Structure

- It is a direct access electromagnetic storage component.
- Harddisk contain a bunch of magnetic coated platters.
- To store the data
- each platter contain 2 recorded surface.
- each surface contain set of concentric circle called as tracks
- each track contain set of sectors.
- Sectors hold tiny data.
- in the Harddisk surface, tracks & sectors are the addressable unit [0 to N-1]
- In the harddisk each surface has its own Read-write head to access the data directly from the harddisk.

- In the harddisk cylinder is logical Address by connecting the same track no. in all the surface



Surface



Storage

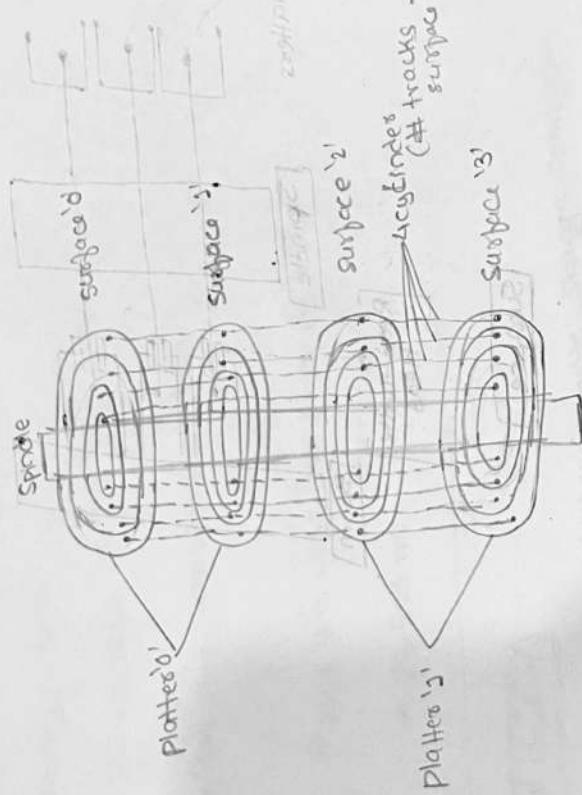
rock
in ft

(1)

(2)

(3)

(4)



locations (# tracks)

Surface 3

Surface 2

2011/16/22

2011/16/22

2011/16/22

2011/16/22

2011/16/22

2011/16/22

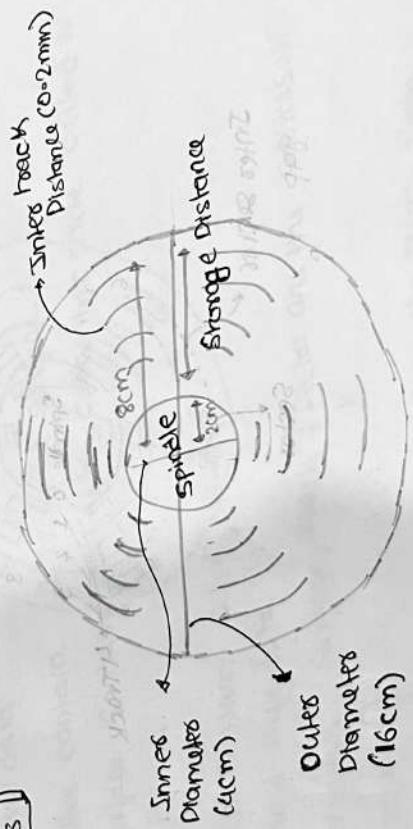
2011/16/22

2011/16/22

2011/16/22

2011/16/22

Surface Analysis



$$\text{Storage Distance} = \left[\text{Outer radius} - \text{Inner radius} \right] \Rightarrow (8-2) \text{ cm}$$

$\approx 6 \text{ cm}$ and 1.2 cm

$$\# \text{ tracks} = \frac{\text{Storage Distance}}{\text{Inter track distance}}$$

$$\Rightarrow \frac{6 \text{ cm}}{0.2 \text{ mm}} \approx \frac{60 \text{ mm}}{0.2 \text{ mm}} \Rightarrow 300 \# \text{ tracks}$$

① $\# \text{ cylinders}$ in the disk = $\# \text{ tracks}$ in the surface

② $\# \text{ tracks} = \# \text{ sectors} * \# \text{ bytes per sector}$

③ cylinders capacity = # surface * track capacity in disk

$$④ \text{Disk capacity} = \# \text{ cylinders} * \text{cylinder capacity in disk}$$

Alternative:
• Disk capacity

$$= \# \text{ surfaces} * \# \text{ tracks} * \# \text{ sectors per track} * \# \text{ bytes per sector}$$

Seek Time: time required from initial position to desired track position (A, 1, 7) = $\#$ of cylinders

Rotational latency: time taken to adjust the head to desired head position it is "0" b/c will adjust seek time worst case it is one complete rotation. Avg. case: it is half (default).

- Different adjustment are required
 - ① seek time
 - time required to move the head point from the initial position to desired track in the surface
 - time required to move the head to the beginning of the desire sector in the track
 - ② rotational latency
 - in the best-case Rotational latency is "0" b/c head is "0" sec head
 - in the worst case, one complete revolution is required to adjust the head point.
 - ③ transfer time
 - time required to access the data from the disk.
 - ④ overhead
 - additional delay in the disk operation
 - so total time required to access the data from hard disk is avg.

$$T_{avg} = \frac{\text{seek} + \text{avg. Rotational latency} + \text{transfer} + \text{overhead}}{\text{time}}$$

Note → when the sector address is expressed in 3-decimal format there sectors no. in the ~~block~~ block is calculated by using the following formula.

Sector # i : cylinders # i
 Sector no. in the entire Hard disk
 Surface # j : sectors # j
 Sector # k : tracks # k
 Sector # $i+j \cdot n + k$ = $K + S(j \cdot n + k)$
 in the disk

Q1] Consider Harddisk with a rotational rate of 7200 rpm

Avg. seek time of a disk is 4.2 ms. 64ms data file is stored in the disk. Disk contain 32 platters. Surface contain 128 tracks. Track contains 256 sectors. Sector holds 512B data.

- ① Disk capacity?
- ② Time required to access the file?
- ③ Time required to access 100 random sectors?
- ④ Disk data transfer rate in Mbps?

⑤ Disk capacity : $(32 \times 2) \times 128 \times 256 \times 512B$
surface

$$\Rightarrow 2^6 + 2^7 \times 2^8 + 2^9 \times 2^{10} \Rightarrow 2^{20}B \rightarrow \boxed{128GB}$$

⑥ file accessing time

$$\boxed{\text{File size} = 64B}$$

1 sector \rightarrow 512B

1 # sectors \rightarrow File (64B)

$$\Rightarrow \frac{64B}{512B} \Rightarrow \frac{1}{8} \rightarrow 2^3$$

\rightarrow

① Avg. seek time $\approx 4.2\text{ms}$

② Rotational latency :

7200 revolution $\rightarrow 2\text{ms}$
 $\rightarrow (60\text{sec})$

Resolution

$$\Rightarrow \frac{60\text{sec}}{7200} \Rightarrow 8.33\text{ms}$$

Avg. Rotational latency

$$= \frac{1}{2} * 8.33\text{ms}$$

$$\Rightarrow 4.165\text{ms}$$

③ Transfer time

1 revolution — 1 track
1 revolution ; 256 sectors
1 byte
9 bytes
(128 sectors)

$$\Rightarrow \frac{8.33\text{ms} * 128}{256} \Rightarrow 0.165\text{ms.}$$

④ overhead = not bytes

$$\therefore T_{avg} = (4.0 + 4.165 + 4.165)\text{ms}$$

$$\Rightarrow 12.33\text{ms} //$$

sector # 1 / 256

sector # 20.

c). Random sectors accessing required adjustment for every sectors



? time

$$\Rightarrow \frac{8.33\text{ms}}{256} \times 1 \Rightarrow 0.0325\text{ms}$$

$$\text{Tang.} = \frac{(4.2 + 4.165 + 0.0325 + 0)\text{ms}}{\text{track}} \times 100 \text{ seconds.}$$

$$= 839.45 \text{ ms}$$

② surface rate:



$$\Rightarrow \frac{256 \times 512 \text{ B}}{8.33\text{ms}} \rightarrow \frac{256 \times 512}{8.33} \times 10^3 \text{ B/sec}$$

$$\therefore \text{Disk Rate} = \# \text{surface} \times \text{surface rate}$$

$$\Rightarrow \frac{256 \times 512}{8.33} \text{ Kbps.}$$

$$\Rightarrow 64 \times \frac{256 \times 512}{8.33} \text{ Kbps}$$

$$\Rightarrow \frac{1007.035 \text{ Mbps}}{1000 \text{ Gbps.}}$$

JO-ORGANISATION

- I/O-organisation

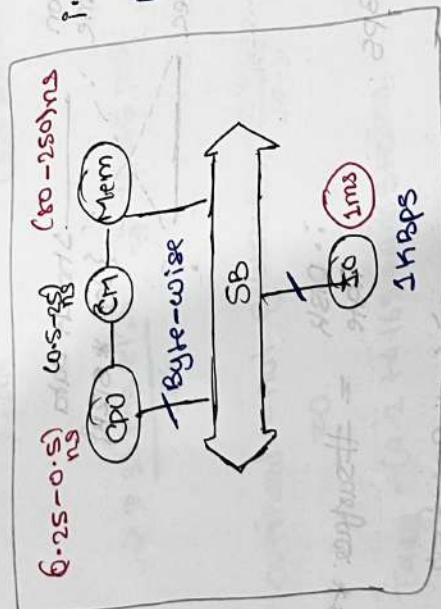
 - I/O device are electro-magnetic component & CPU is an electronic component. So there is an difference exist in term of operating mode, word formats,

Date _____

- Data transfer rate
 - to synchronize the SO speed with CPO speed
 - High speed interface chip is used, called as

- So Interface chip is module
- To understand what is responsible for IO operation.

In the system design all the IO devices are connected to the CPU via its interface chip.



1. e. SDS presentation also
SDS measurement app.

HOME TO SPEED

→ 8.35
520480 Kede

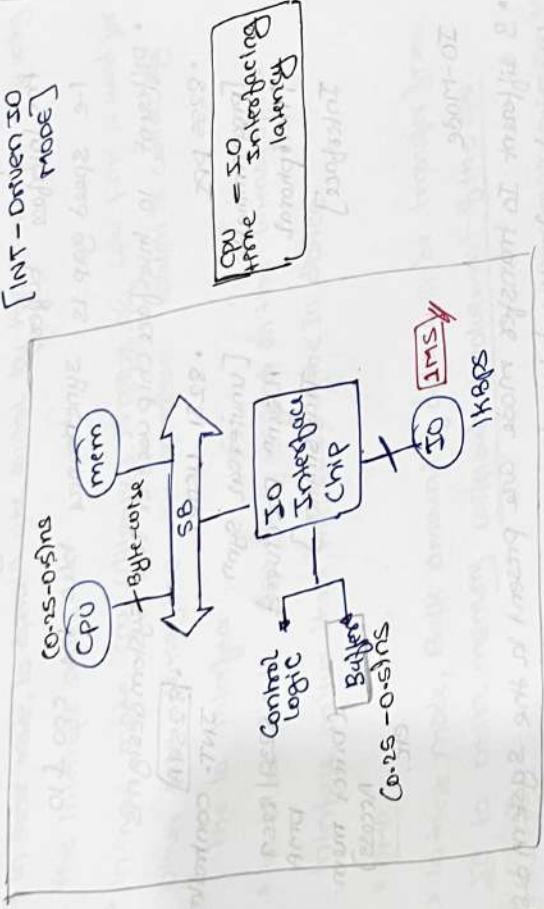
$\Delta x_B = 1 \text{ sec}$

189

$$29870 \text{ foot} \cdot \frac{1}{1152} \text{ sec} \Rightarrow 10^5 \text{ sec} \Rightarrow 1 \text{ ms}$$

i.e System design with SO interfacing chip.

[INT - Driven SO Mode]



Accessing sequence.

- CPU initially tie the SO interface chip along with a TO command later busy with other useful task
- SO interface control logic interprets the SO command & enable the SO operation
- Based on the speed of SO device consume the time & transfer the data, later transfers the data to interface buffer
- when SO data is available in the buffer then the interface sent the interrupt signal to CPU & wait for acknowledgement signal.
- After receiving the ACK signal the buffered content will be transferred to CPU.

- In these process CPU will be accessing the I/O data from the interface buffers.
- i.e. speed gap is synchronized b/w the CPU & I/O
- different I/O interface chip used in the system design is
 - 8259A INT- controller
 - 8255 PPI [Programmable Peripheral Interface]
 - 8251 USART [Universal Synchronous & Asynchronous Transmitter & Receiver]
 - 8237 DMA [Direct Memory Access]
 - etc.

I/O-mode

- 3 different I/O transfer mode are present in the system design use to transfer the data from I/O to CPU & memory

name as

① Programmed I/O ② Interrupt driven I/O

③ DMA

Programmed-I/O

- In these mode, I/O operation are programmed into CPU. So CPU is responsible for I/O operation
- In this mode, CPU enters into waiting state until the completion of I/O operation. So processor utilization is poor.
- In these mode CPU time depend on the speed of the I/O device & amount of data to be transferred. [data size]

Transfer to CPU:

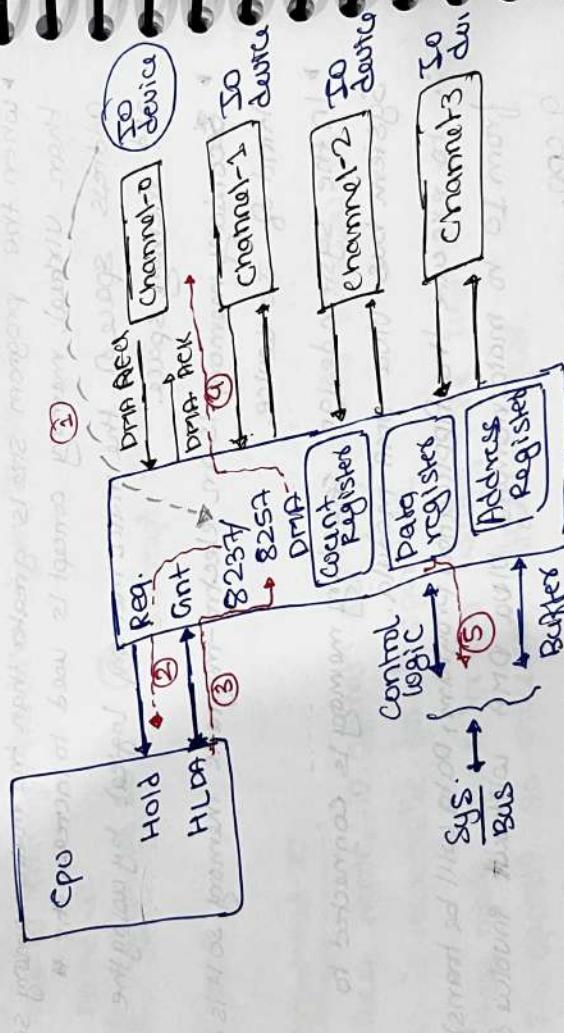
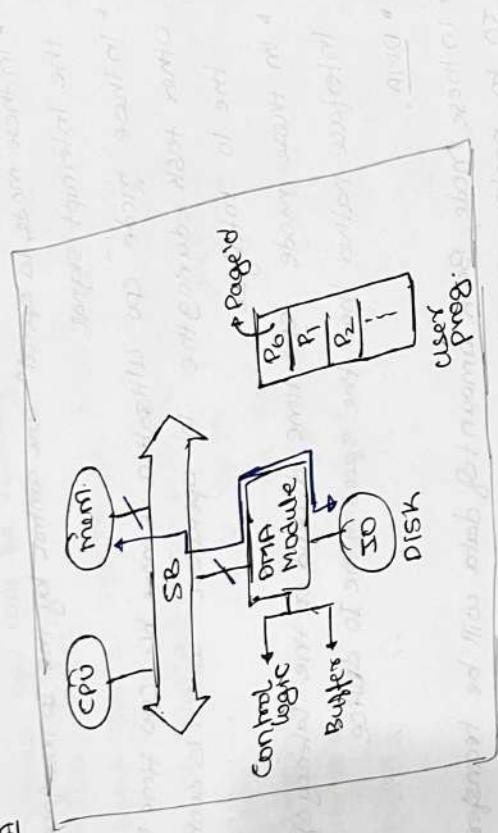
• after execution the value is stored in buffer without any loss

Interrupt-driven IO.

- In these mode, IO operation are control by the IO interface chip using the interrupt signal
- In these mode CPU utilization is more b/c CPU time is used for other task during the IO operation which is control by the IO interface
- In these mode CPU time depend on the latency of the interface rather than the speed of the IO device

DMA.

- In these mode, Bulk amount of data will be transferred from IO to main memory without involvement of the CPU.
- In these mode, program size is greater than the main memory size when the program size is used to increase the ^{RAM} than virtual memory logical but using the address space of the main memory logical so it is a 2nd memory space
- Secondary memory is an electro-magnetic memory. So it is a kind of IO device
- In the system design secondary memory is connected to system bus VISA DMA module.
- To execute the Application program, data will be transferred from IO to main memory. VISA DMA without ~~without~~ involves of a CPU.



Req: Request
Grant: Grant

Accessing sequence.

- CPU initializes the DMA along with a port address / memory address, control signal, count value, later Busy with other task.
- DMA control logic interrupt the CPU
- So operation
- Board on the speed of the IO device consume the time to prepare the data, later enable DMA REQ signal to DMA
- After receiving the above signal, DMA enable the old signal to CPU to give the control of the system bus & waits for HDMA signal
- After receiving the HDMA signal DMA enable the DMA ACK signal to IO
- After receiving the above signal, IO device start the data transmission to main memory via DMA until the count became 0. After the data transmission, bus connection will be re-established to CPU

Note:-

- In the DMA operation CPU presents 2 states
- ① Busy state
- ② Blocked state (Hold state).

- CPU is in Busy state until the IO device prepares the data
- CPU is in Block state until the IO device transfers the data into main memory.

Let x : is a preparation time.
 y : is a transfer time

$$\% \text{ time CPU} = \left(\frac{x}{x+y} \right) \times 100$$

$$= \frac{y}{x+y} \times 100$$

• DMA module is operating under 3 mode

- ① Burst mode
- ② Cycle Steady mode

③ Interleaving mode

- In Burst mode of DMA, Bulk amount of data will be transferred to main memory after receiving the system from the CPU.
- Taken
SB from CPU — Bulk Data transfer — Return SB to CPU
- In the cycle steady mode of a DMA, So device constantly preparing the data in a small unit (word, transfer to)

By requiring the System Bus multiple time

Time SB — word Data — Return SB to CPU

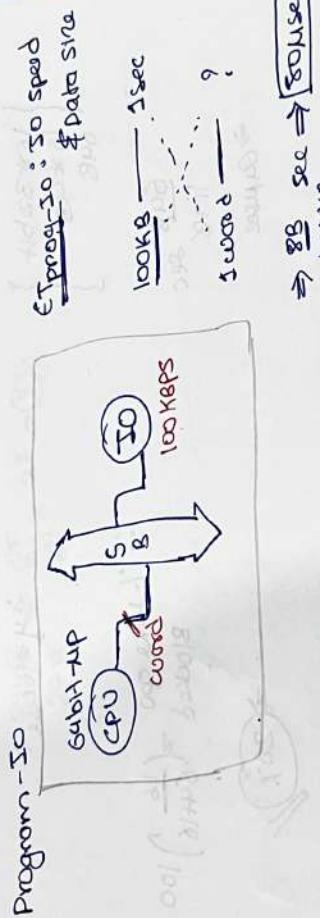
- In the interleaving mode of a DMA, Data will be transferred to main memory in form of words when the CPU is busy with local bus.

Q. 12 a) Differentiate between
i) X, 216 & 216/16 bit

Task : If IO of P0 was
 CPU : S8, Local S8, Local S8
 P0 : S8 -> S8

Note:
 Converting
 System gain
 you are
 measuring
 should be
 denormalized

- Q1) Consider workspaces to device triggered while the user program
 programs to mode. Data transmission is 8usec. Under Interrupt - ZO
 gain of 256. What is overhead under Interrupt - ZO?
 due to program is made?



$$8 \times 8 = 64 \text{ usec}$$



$$8 \times 8 = 64 \text{ usec}$$

$$8 \times 8 = 64 \text{ usec}$$

$$S = \frac{ET_{Interrupt-ZO}}{ET_{Program-ZO}} \rightarrow \frac{8}{80} = 0.125$$

268

Q) Consider 1Mbps I/O device interfaced to 32bit CPU using the DMA interface whenever 16 words data is ready then it is transferred to main memory. MIC cycle time is 1 microsec. How much % of CPU time is consumed in the DMA operation?

$$\% \text{ time} = \left(\frac{y}{x} \right) 100$$

- In one MIC cycle one can transfer 16 words
- So to transfer 16 words required cycle time.

$$x: 16 \mu\text{s} = \frac{16 \times 10^6}{10^9} \text{ sec}$$

$$= \frac{16 \times 32 \text{ bit}}{16 \times 4B} \times 64B$$

$$y: 16 \mu\text{s} = \frac{16}{10^9} \text{ sec}$$

$$\text{MIC time} = \left(\frac{16}{64 \times 16} \right) 100$$

$$\text{Blocked} = \left(\frac{16}{64 \times 16} \right) 100$$

$$\Rightarrow 20\%$$

- Q) Consider 10Mbps I/O device interfaced to 8bit CPU using DMA Interface. DMA module contains a bit count register 8 bit data register. How many DMA cycle are required to transfer 32KB datafile from SDRAM?
- DMA cycle depends on the count register
 - Count Register size = 9 bits { no initialized } \downarrow

$$\# \text{ possible} = 2^9 = 512$$

In each count, word data is transferred.

∴ In one cycle — 512 words

Data transfer

$$1 \times 512 \times 64 \text{ bits}$$

$$\Rightarrow 512 \times 8 \text{ B}$$

$$\Rightarrow 2^9 \times 2^3 \text{ B}$$

$$\Rightarrow 2^{12} \text{ B}$$

$$\Rightarrow 2^{12} \text{ B}$$

1 DMA cycle — 2^{12} B

#DMA cycle

$$\text{File (32x8) } \Rightarrow \frac{2^5}{2^2} = 2^3 = 8 //$$

Space

0000	0001	0010	0011	0100	0101	0110	0111
0000	0001	0010	0011	0100	0101	0110	0111
0000	0001	0010	0011	0100	0101	0110	0111
0000	0001	0010	0011	0100	0101	0110	0111
0000	0001	0010	0011	0100	0101	0110	0111

⑧ column-major traversal
→ row-major traversal
→ column-major traversal
→ row-major traversal
→ column-major traversal
→ row-major traversal

28

Block size

Assume:

0000	0000	0000	0000	0000	0000	0000
0001	0001	0001	0001	0001	0001	0001
0002	0002	0002	0002	0002	0002	0002
0003	0003	0003	0003	0003	0003	0003
0004	0004	0004	0004	0004	0004	0004

assume, cache

i.e. ~~00001111111111111111111111111111~~

④ Row-major scanning

Accessing:

Storage.

0000 [0]	0001 [0]	0002 [0]	0003 [0]	0004 [0]	0005 [0]	0006 [0]
0000 [1]	0001 [1]	0002 [1]	0003 [1]	0004 [1]	0005 [1]	0006 [1]
0000 [2]	0001 [2]	0002 [2]	0003 [2]	0004 [2]	0005 [2]	0006 [2]
0000 [3]	0001 [3]	0002 [3]	0003 [3]	0004 [3]	0005 [3]	0006 [3]
0000 [4]	0001 [4]	0002 [4]	0003 [4]	0004 [4]	0005 [4]	0006 [4]

Ex. (char) access

- Spatial locality within adjacent data elements, short in row-major is an order set of consecutive data elements, short in column-major
- In the implementation of an array, spatial locality is present in the memory in a sequential locality, and a non-locality in the memory in a random locality
- In the memory in a sequential locality, data access occurs from the same block in a sequence
- In the implementation of an array, spatial locality is present in the memory in a random locality, data access occurs from the same row-major in a sequence

• ACCESS:

$a[0][0]$: miss \rightarrow $a[0][0] = 100$ → CM

$a[0][2]$: hit \rightarrow $a[0][2] = 100$ → CM

$a[0][3]$: hit \rightarrow $a[0][3] = 100$ → CM

Hit follows miss

Hit = 50%

array size = 10 elements

Decoding:
1. row-major
2. column-major
3. row-major
4. column-major
array elements size = 10 (square)

Block size \rightarrow 2B

Elements (block) \rightarrow 2

Element (rows) \rightarrow 5

Block / row \rightarrow 2

{# miss / row}

(3) column-major indexing.

i.e. $a[0][0], a[2][0], a[3][0], \dots$

Assume, CM: empty

Total # miss / memory access
 \rightarrow # rows * # miss
 5 rows * 2 miss \rightarrow 10 miss
 \rightarrow 2 miss / row \rightarrow 8 miss
 \rightarrow 4 miss / row \rightarrow 2 miss / row \rightarrow 1 miss / row

- Access:
 - $a[0][0]$: miss; $a[0][0] \rightarrow a[0][0]$ → cm
 - $a[1][0]$: miss; $a[1][0] \rightarrow a[1][0]$ → cm
 - $a[2][0]$: miss; $a[2][0] \rightarrow a[2][0]$ → cm
 - $a[3][0]$: miss; $a[3][0] \rightarrow a[3][0]$ → cm
- every element inline $\boxed{\text{col}}$ $\boxed{\text{miss}}$
- B/c of random block reference, replacement possible in the col.
- Replacing $a[0][0]$ with $a[1][0]$
- Decoding intent:
 - Element size = 8
 - Block size = 28
 - # Elements / blocks = 2
 - # Elements / col = 4
 - # blocks / col = 4 {no spatial locality}
 - {# miss / col}
- Total # miss = $\frac{\text{# cols} * \text{# miss / col}}{\text{Row}}$ $\Rightarrow 4 * 4 = 16$

Storage Sequence	Accessing Sequence	Spatial Locality
Row-wise	Row-wise	yes
Row-wise	Col-wise	no
Col-wise	Row-wise	no
Col-wise	Col-wise	yes

Default Storage: Row-wise

Accessing Sequence : Program Dependent