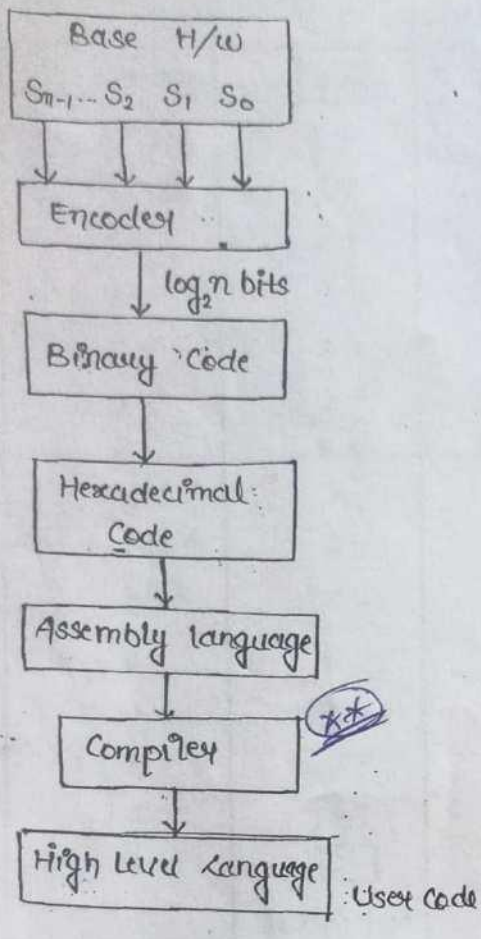
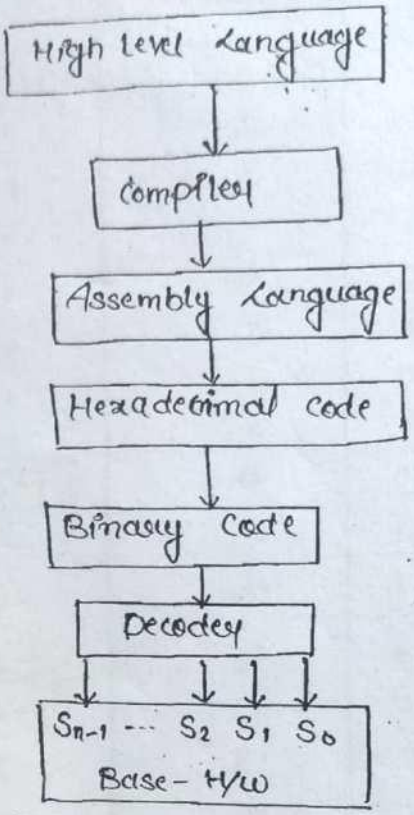
CONTENT

Designer View



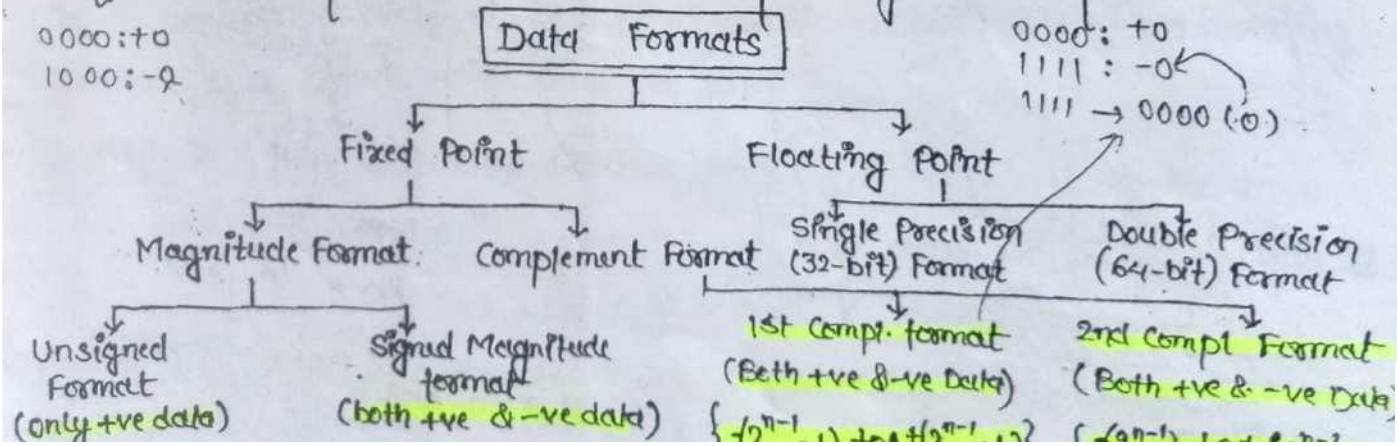
User View



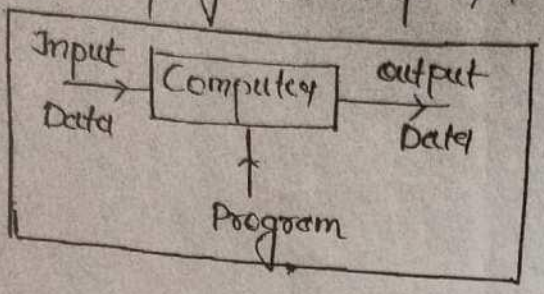
Data :→ It is a binary sequence which is associated with a value based on data format

Binary Code – Bind With – Value

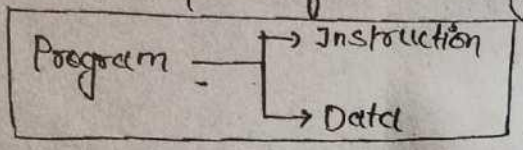
Different data formats used in the comp. design is as follows---



Computer :→ It is a computational device used to process the data under the control of a user prog. ∴ Comp s/m functionality is prog execution.



Program :→ Program is a sequence of Instr. along with data



Instruction :→ It is a binary code which is designed inside the processor to perform some task.

Binary Code	–	Bind With	–	Operation

Eg :- If CPU supports 8 diff. operations then opcode is defined as

$\log_2^8 = 3$ bit

Opcode	Operation
0 0 0	+
0 0 1	*
0 1 0	/
⋮	⋮
1 1 1	AND

Decided by designer

Control Unit ROM

Designer View

Base H/w

$S_{n-1} \cdots S_2\ S_1\ S_0$

Designer

0	0000	0	+0	+0	+0
1	0001	1	+1	+1	+1
2	0010	2	+2	+2	+2
3	0011	3	+3	+3	+3
4	0100	4	+4	+4	+4
5	0101	5	+5	+5	+5
6	0110	6	+6	+6	+6
7	0111	7	+7	+7	+7
8	1000	8	-0	-7	-8
9	1001	9	-1	-6	-7
10	1010	10	-2	-5	-6
11	1011	11	-3	-4	-5
12	1100	12	-4	-3	-4
13	1101	13	-5	-2	-3
14	1110	14	-6	-1	-2
15	1111	15	-7	-0	-1
			Not in Use	Not in Use	

1's Complement

∴ 1000 : [-7]

1000 → 0111 (7)

1001 : [-6]

1001 : 0110 (6)

2's Complement

1000 : [-8]

1000 : → 0111

+ 1

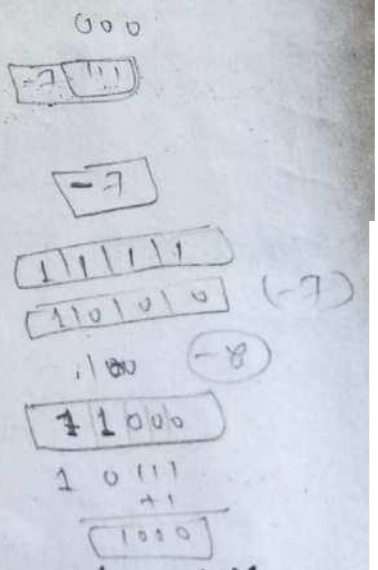
1000 = 8

1001 : [-7]

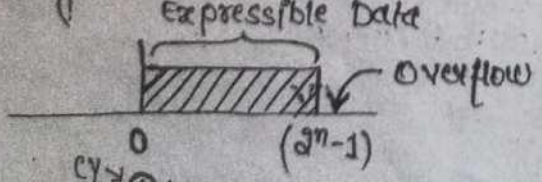
1001 → 0110

+1

0111 = 7



Unsigned Data :-



cy → 1111

15 → 1111

15 → 1111

30 1110

cy = 1

(n-bit) + (n-bit) = (n+1) bit

↓

cy flag

(n-bit) * (n-bit) = 2n bit

↓

group of 2 registers

- In the comp. design carry flag is used to indicate the range exceeding condition of a unsigned arithmetic.

Condition: Is there any extra bit out of MSB. → T = Set = 1 = CY (Carry)
→ F = Reset = 0 = NC (No carry)

Eg:→

8	——	(1)000
4	——	0100
12		1100

Justify :-

PSW
- Acc → 1100
- Flag Reg → CY (0)

Ans : 1100
(12)

Eg :-

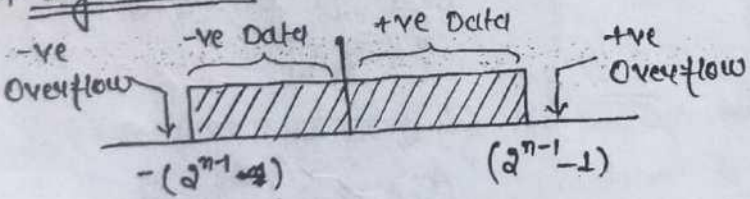
8	——	1000
9	——	1001
17		0001
CY=1		CY=1

Justify :-

PSW
- Acc → 0001
- Flag Reg → CY (1)

Ans : (CY) 10001
(17)

Signed Data :



In the comp design overflow flag is used to indicate the range exceeding condition of a signed arithmatic.

Condition :- "There is a carry into the MSB & No carry out of the MSB or Vice Versa" → T = Set = 1 = OV
→ F = Reset = 0 = NO

Operation :- "XOR" b/w the In-carry & out carry w.r.t MSB

Expression :→

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

- x : MSB of operand 1
- y : MSB of operand 2
- z : MSB of Result
- $\bar{x}\bar{y}z$: +ve OV
- $xy\bar{z}$: -ve OV

Eg-1 :→

+7	→	0111
+3	→	0011
+10	→	1010
OV = 1		

Eg-2 :→

+7	→	0111
-3	→	1101
+4	→	10100
OV = 0		OV = 0

Eg-3 :→ ~~+7 → +3 → -4 →~~

Justify :-

PSW → Acc: 1010, Flag Reg: OV (1)

Ans : 01010 (+10)

Justify :→

PSW → Acc: 0100, Flag Reg: OV (0)

Ans :→ 0100 (+4)

Eg-3 :→

-7	→	1001
+3	→	0011
-4	→	1100
OV = 0		OV = 0

Eg-4 :→

-7	→	1001
-3	→	1101
-10	→	0110
OV = 1		OV = 1

Justify :-

PSW → Acc: 1100, Flag Reg: OV = 0

Ans :→ 1100 (-4)

1100 → 0011 + 1 = 0100 = 4

Justify :→

PSW → Acc: 0110, Flag Reg: OV (1) → -ve (OV $\{xy\bar{z}\}$)

Ans :→ 10110 [-10]

10110 → 01001 + 1 = 01010 = 10

Ques:→ Consider the following signed data, processed on a addition h/w. What is the status of a overflow flag in computation.

Data : ⑧① ✓
1 0 1 0 1 1 0 0
1 1 1 0 0 1 0 0
1 0 0 1 0 ϕ 0 0
OV = 0
xyz = 0

Ques:→ Consider the following 8 bit data computation. What is the status of an overflow flag h/w during computation.

(-112) + (-63)

Soln:→ 8bit signed data Range = $-(2^{8-1})$ to $+(2^{8-1}-1)$
{Default 2's compl} - $\{-2^7$ to $2^7-1\}$
$\{-128$ to $127\}$

-112
-63
-175

-175 is not in range

∴ OV = 1
-ve OV
↓
$(xy\bar{z})$

Ques:→ Consider the following bit pattern used to processed on a addition h/w

Data: ①⑧ 1 0 1 1 0 1 1 1
1 1 0 0 1 0 0 0
0 1 1 1 1 1 1 1

a.) What is the status of CY and OV flags in computation.

b.) What is the result in decimal when data is in unsigned format.

c.) What is the result in decimal when data is in signed format.

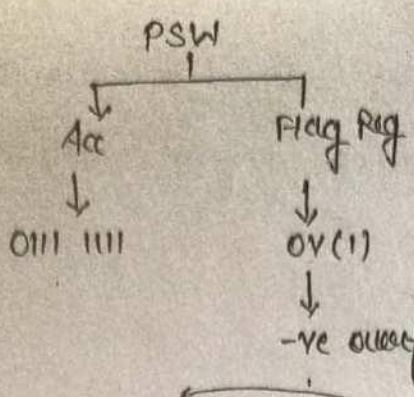
OV = 1
$xy\bar{z}$ → -ve overflow
CY = 1

b.) Unsigned Data :→

PSW
↓ Acc: 0111 1111
Flag Reg: CY(1)

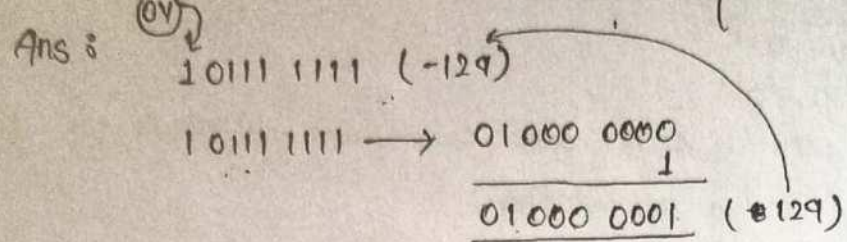
Ans: CY → 1011 1111 = 389

c) Signed Data :→



Signed me overf
dekhte hai.
unsigned me ca
flag delete hai.

Ans :



Floating Point Data Format :→

Representation of a very large and very small fraction of data consumes more m/m space.

Eg :→ +6732050000000 ; very large ($\cong \pm\infty$)
+0.00000000000375 ; very small ($\cong 0$)

To represent the large and small fraction of data within a less amou of m/m space, floating point data is used

General Form of a floating point data is $\boxed{\pm M * B^{\pm e}}$

± : Sign
M : Mantissa / Significand (Fr
B : Base / Radix
e : Exponent

In the floating pt. representation, common format is used to represe the data into a uniform look called as Normalization

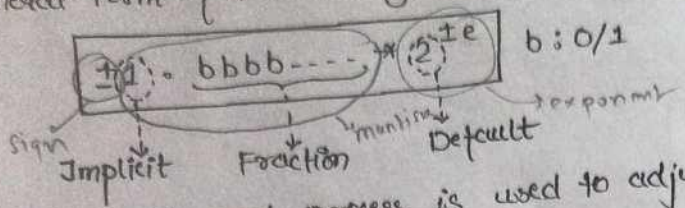
i.e. $\boxed{\pm \text{Valid Digit} \cdot \text{fraction} * B^{\pm e}}$

When Base is greater than 2 then valid digits are words so ex space is required to store the valid digit.

Base	valid Digit
B = 3	{1, 2}
B = 4	{1, 2, 3}
B = 5	{1, 2, 3, 4}

To save the m/m space base is restricted to 2. ∴ Valid digit become implicit i.e 1. Hence this digit bit need not be stored in m/m.

General Form of a Normalization is -



Mantissa alignment process is used to adjust the decimal point. In this process right alignment increment the exponent and left alignment decrement the exponent

Eg :→ Data : $+110101 . 1101 * 2^{+4}$

↓

$1 . bbb---$

↓

Align to Right upto 5 times

↓

$+1 . 10101101 * 2^{+4+5}$

$\boxed{+1 . 10101101 * 2^{+9}}$

Eg :→ Data : $+0.000000101101 * 2^{+12}$

↓

$1 . bbb---$

↓

Align to left upto 7 times

↓

$+1 . 01101 * 2^{+12-7}$

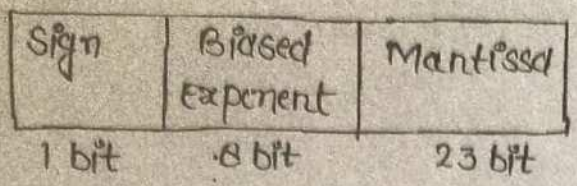
$\boxed{+1 . 01101 * 2^{+5}}$

When the data is in normal format then it is stored in the m/m by using the IEEE floating point formats.

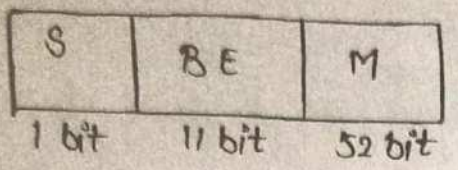
According to a IEEE standard 754, two kinds of formats are present — 1.) Single Precision (32-bit) format

2.) Double Precision (64-bit) format

32 bit Format :→



64 bit Format :→



Floating point data is stored in the m/m with 3 fields of information.

1) **Sign Field** :- This field is used to represent the sign of a data i.e.

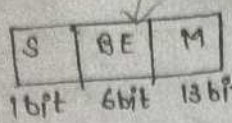
Sign(S) → 0 (+ve) } Mantissa Sign
→ 1 (-ve) }

2) Biased Exponent Field :- This Field is used to represent the exponent value. In the m/m exponent is stored in a biased exponent format because MSB bit is already reserved for Mantissa side. Hence 2's complement exp. is not possible in m/m storage.

BE = Actual Exponent (AE) + Bias

Bias is a max. possible +ve exponent depends on the format i.e.

Eg:→ Format :



Data : $+1.1011 * 2^{13}$

① Store $\boxed{BE = AE + Bias}$

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

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

$= +31$

AE : 001101

Bias : 011111

BE : 101100

② Retrieve [Read] $\boxed{AE = BE - Bias}$

BE : 101100

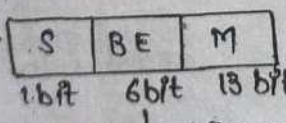
Bias : 011111

AE : 001101 [+13]

Note :→ When the format is designed with an excessed bias then bias is a centre of the exponent range i.e. $\left(\frac{2^n}{2}\right)$, n : # bits in BE Field.

In this format we need to take the complement of a MSB bit of BE to get the actual exponent rather than substraction operation.

Eg :→ Format :



Data : $+1.1011 * 2^{+13}$

① Store : BE = AE + Bias

Bias = Excess bias

$= \left(\frac{2^n}{2}\right) = \left(\frac{2^6}{2}\right)$

$= 32$

AE : 001101

Bias : 100000

BE 101101

② Retrieve : AE = BE - Bias

BE : 101101

Bias : 100000

AE : 001101 [+13]

Alternate :- AE = complement of MSB bit of BE

BE : 101101

↓ complement of MSB

↓

AE : 001101

Mantissa Field :- This field is used to represent the fraction part of a data. In the m/m, fraction is always stored in a Normalized Mantissa Format i.e. 1.bbb--- ; where b = 0/1

Implicit Fraction

In the above structure, only the fraction part is stored in the m/m Because valid digit is Implicit

Note :- To retrieve the data from m/m, adjustment is required to get the implicit digit as a fraction i.e. align the mantissa to right dir. upto 1 time to get the Implicit digit called as Denormalization (Sub-Normal)

From the m/m

Data : $\pm .bbb--- * 2^{\pm e}$

↓

De-normalization

↓

Align to Right upto 1 time to get implicit digit

↓

$\pm 0.1bbb--- * 2^{\pm e+1}$

↓

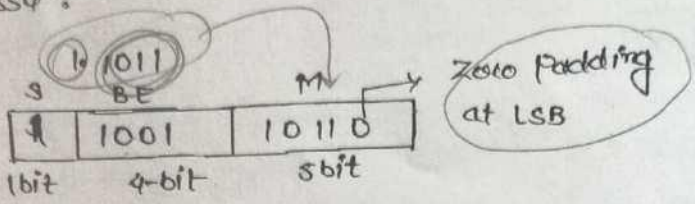
$\pm 0.1bbb--- * 2^{\pm e'}$

AE : 0010

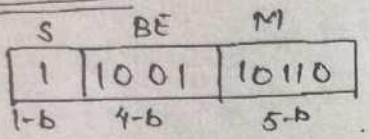
Bias : 0111

BE : 1001

③ Mantissa :



Retrieve Data :-



① Sign (S) = 1
(-ve)

② AE = BE - Bias

BE : 1001

Bias : 0111

AE : 0010 [+2]

③ Mantissa

M = 10110

From the m/m

Data : $-.10110 * 2^{+2}$

↓

Denormalization

↓

Align to Right upto 1 time to get the valid digit

↓

$-0.(1)0110 * 2^{+2+1}$

Implicit

$-0.110110 * 2^{+3}$

[-6.75]

Ques:- Consider the following hypothetical format used to represent data.

Format :

S	BE	M
1bit	7bit	12 bit

Data : $+13.25 * 2^{+14}$

What is its Hexadecimal equivalent when the data is stored in m/m

a.) Without Normalization

b) With Normalization.

Soln :→ Data : $+13.25 * 2^{+14}$

$+1011.01 * 2^{+14}$

↓

Without Normalization

↓

Store into a m/m

↓

Format Required

S	BE	M
1bit	7bit	12 bit.

① Sign :→ +ve

$= 0$

② BE :- $BE = AE + Bias$

$Bias = Normal\ Bias$

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

$= +(2^{7-1} - 1)$

$= +63.$

AE : 0001110

Bias : 0111111

BE : 1001101

③ Mantissa Without Normalization

$+1101.01$

S	BE	M
0	1001101	0100 0000 0000
1bit	7bit	12-bit
4	D	4 0 0

Data :- $+13.25 * 2^{+14}$

↓ Binary

$+1101.01 * 2^{+14}$

↓

Normalization

$(1.bbb\text{---})$

↓

Align to Right upto 3 times

↓

$+1.10101 * 2^{+14+3}$

$+1.10101 * 2^{+17}$

↓

Store the data into m/m

↓

Format Required

S	BE	M

1 bit 7 bit 12 bit

① Sign (S) = +ve
= 0

② BE = AE + Bias

AE : 0010001

Bias : 0111111

BE : 1010000

③ Mantissa with Normalization

1.10101

S	BE	M
0	1010000	10101000000

Ques :→ Consider the following m/m data which is stored in the hypothetical format :

S	BE	M

1-bit 7bit 12bit

↳ Excess Bias

Data : 0X 3FC00

↓

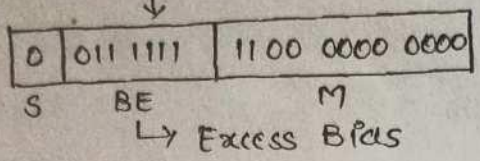
Hexadecimal

What is the actual value association with above m/m data in decimal.

Data : 0X 3FC00

Retrive :→ M/m Data : $(3FC00)_H$

↓ Binary



① Sign (S) = 0 +ve

② AE = BE − Bias
or
AE = complement of MSB bit of BE } Excess-bias

Bias = Excess Bias

$= \frac{2^n}{2} = \frac{2^7}{2} = 64$

BE : 011 1111

Bias : 1000000

AE 1111111 [-1]

1111111 → 0000000
+1
0000001 = 1

③ Mantissa 11000 ---

From the m/m

Data : $+.11 * 2^{-1}$

↓ De-normalization

↓ Align to right upto 1 time to get the valid digit

↓

$+0.111 * 2^{-1+1}$

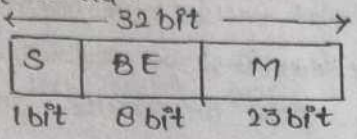
$+0.111 * 2^0$

+0.875

$(0*2^0)+(1*2^{-1})+(1*2^{-2})+(1*2^{-3})$

Range of Floating Point Data :→

Floating pt. data range always depends on the floating pt format. Lets consider single precision floating pt. format to repost the floating pt. data.



① Sign (S) → 0 (+ve)
→ 1 (-ve)

② Exponent Range :-

⊛ Depends on BE Field

BE = AE + Bias

Bias = Normal Bias

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

$= +(2^{8-1}-1)$

$= +(127)$

ⓐ Min BE = All 0's in BE

= 0000 0000

= 0

AE = BE - Bias

= 0 - (+127)

= -127

ⓑ Max BE = All 1's in BE

= 1111 1111

= 255

AE = BE - Bias

= 255 - (+127)

= +128

Exponent Range :

{-127 to +128}

③ Mantissa Range :

Depends on Normal Mantissa

ie. (1.bbbb)

ⓐ Min 'M' = All 0's in 'M'

= 0000 --- (23) 0's

= 0

Normal M = 1.M

= 1.0

= 1

ⓑ Max 'M' = All 1's in 'M'

= 1111 11--- (23) 1's

Normal 'M' = 1.M

= 1.111 --- (23) 1's

↓

Align to left upto 23 times to make fraction as an integer

↓

1111 --- (24) 1's * 2^{-23}

↓

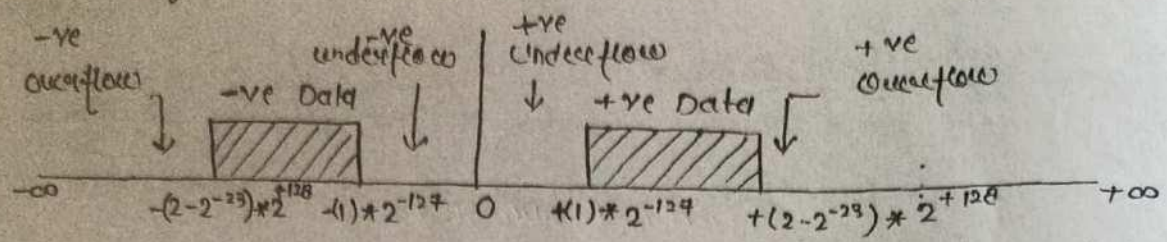
$(2^{24}-1) * 2^{-23}$

↓

$(2-2^{-23})$

∴ Mantissa Range : $\{1 \text{ to } (2-2^{-23})\}$

④ Range of a 32 bit floating point data $\{\pm M * B^{\pm e}\}$



⇒ In the floating point representation special values are defined to report the overflow and underflow conditions. They are 0, +∞ and -∞.

⇒ Different rounding techniques are used to report the result. They are –

(i) Round to zero
(ii) Round to +∞
(iii) Round to -∞
(iv) Round to nearest.

Special value '0'

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

0	0000 0000	00000 - - - (23)0's to 11111 - - - (23)1's	- - - - → 0 (2^{23} m's)

0 0000 0000 00000 - - - (23)0's

↓ S → +ve

↓ BE → 0 → AE = ~~Bias~~ BE - Bias = 0 - (+127) = -127

↓ m → 0 → Normal $m = 1.M = 1.0 = 1$

From the m/m :

Data : $+(1.0)*2^{-127}$

↓

De-Normalized Data

↓

Align to right upto 1 to get the valid digit

↓

$+0.1*2^{-127+1}$

$\boxed{+0.1*2^{-126}}$

Special Value {+∞ to −∞}

S	BE	M	Value
0	All 1's	All 0's	+∞
1	All 1's	All 0's	−∞

0	1111 1111	0000 --- (23) 0's to 1111 --- (23) 1's	2^{23} m's $(2^{23}-1)$ m's

---→ +∞

0 1111 1111 0000 ---(23) 0's

S → +ve

BE → 255 → AE = 255 − (+127) = +128

M → 0 → Normal M (1.M) → 1.0

From the m/m :→

Data = +(1.0) * 2^{+128}

↓

De normalization

↓

Align to right upto 1 time to get the valid digit

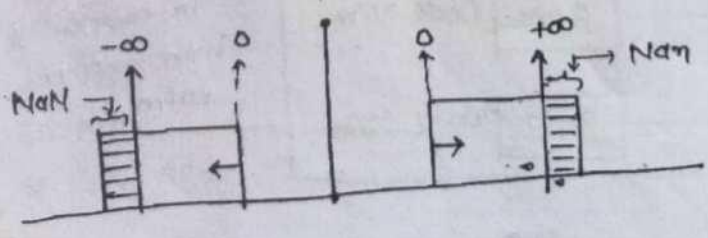
↓

+(0.1) * 2^{+128+1}

+(0.1) * 2^{+129}

(+∞)

S	BE	M	Value
0/1	All 1's	≠0	NaN



Computer Architecture :→

View - I

Based on the prog. storage, architecture is of 2 kinds -

i.) Single m/m Architecture

(Vonn Neumann) → ek hi m/m hoti hai.

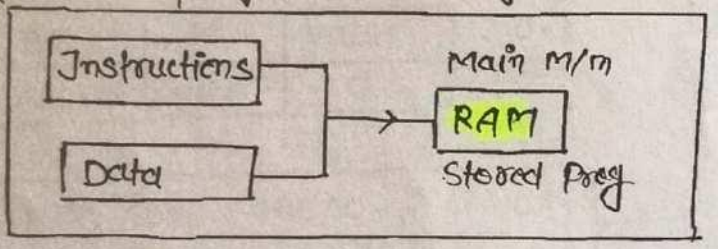
ii.) Multi m/m Architecture

(Harvard Arch.) → ek se jada m/m hoti hai

Vonn Neumann Arch. :- • In this arch. user prog. is always required in the main m/m.

• Main m/m is a volatile m/m so we can load diff. application prog into main m/m.

• In this arch. CPU always executes the main m/m part of prog so that diff. applications prog are running on a same plateform.



Instruction and data both are in same m/m.

Implemented as a uniprocessor design

Eg :- 8085 → 64 KB RAM

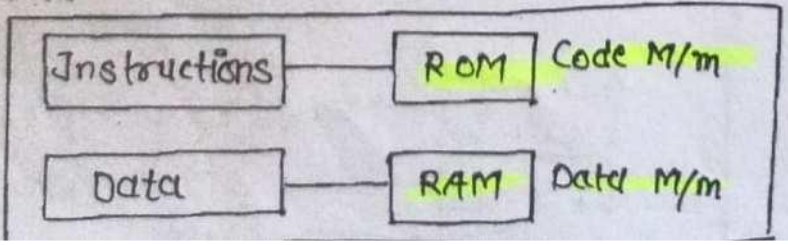
8086 → 1 MB RAM

Used in Personal Computer Design.

Harvard Arch :- In this arch., operational prog is stored in the Code M/m and data is stored in the data m/m seperately.

Code M/m is a permanent m/m i.e. ROM. & data m/m is a volatile m/m i.e RAM.

In this arch. CPU always executes the predefined prog in the code m/m on a diff. data elements.



instruction & data both are in diff. m/m.

Implemented as a Uncontrollee design

Eg :- 8051 Uncontrollee → 4 KB - ROM
→ 128 B - RAM

Used in the Intellegent system Design

Eg :- Embedded System

Note :→ Comp Organization shows the implementation of a Von - Neumann Architecture.

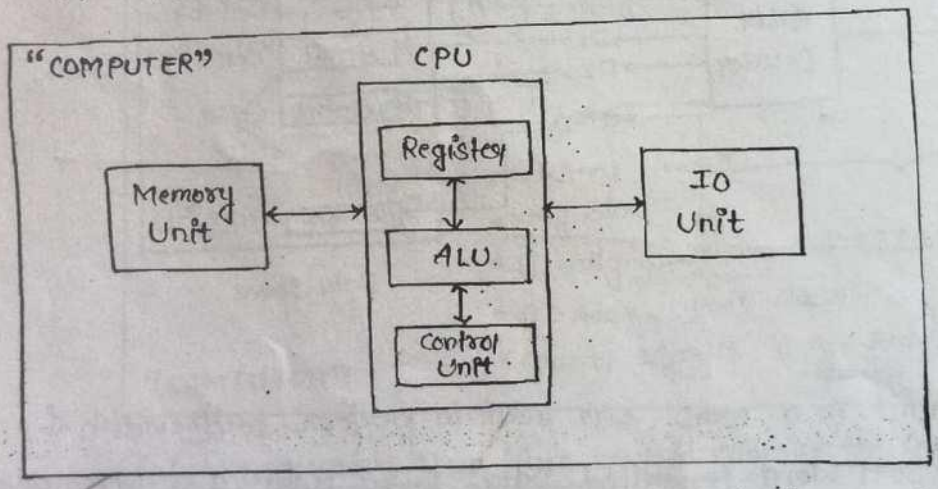
Block Diagram of a Computer :→

Comp S/m contains 3 fundamental components named as -

1.) CPU [Central Processing Unit]

2.) Memory [storage Unit]

3.) I/O [External Comm^n Unit]

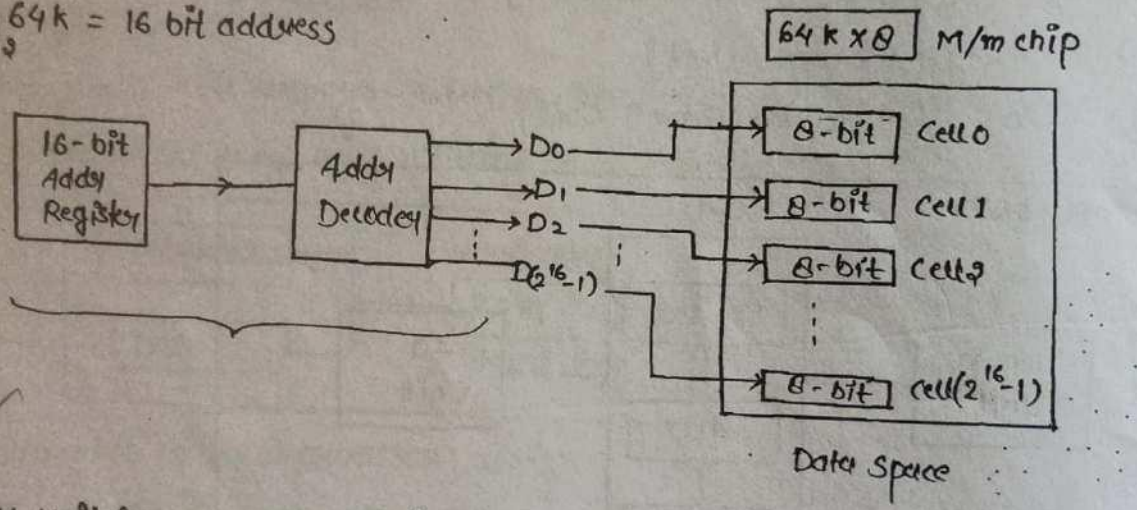


1.) Memory Unit :→ • M/m is a storage element in the Comp S/m.

- M/m chip is organized into a cells.
- Each cell is identified with a unique no. called as address.
- M/m chip recognises the control sigs to indicate the type of operation.
- default cell size in m/m chip design is 8 bit :. in the m/m chip, data will be stored in a bytewise sequence.

Eg :- 64 kB
1 MB
256 MB
4 GiB etc.

- 64 kB
  ↓
  64k x 8
  ↓
  64k x 8
  ↓ ↓
  # cells in the chip i.e. 64K cells → # Addr bits
  $\log_2 64k$ = 16 bit address
  Cell size {# bits in the cell} → 8-bit data



CPU Unit :-

- It is a processing unit in a comp. s/m used to perform arithematic & logical operations on a word formatted data. Word length of a processor is depends on the no. of data pins in the CPU.
- Based on the word length, m/m interfacing will be adjusted to access the data from the m/m unit in a word wise sequence.

Eg :- 8 bit CPU interfaced with 1 cell of m/m space
16 bit " " " 2 " " " "
32 bit " " " 3 " " " "
64 bit " " " 4 " " " "

Note :→ Data storage sequence in the m/m is byte wise.
Data accessing sequence is Word wise

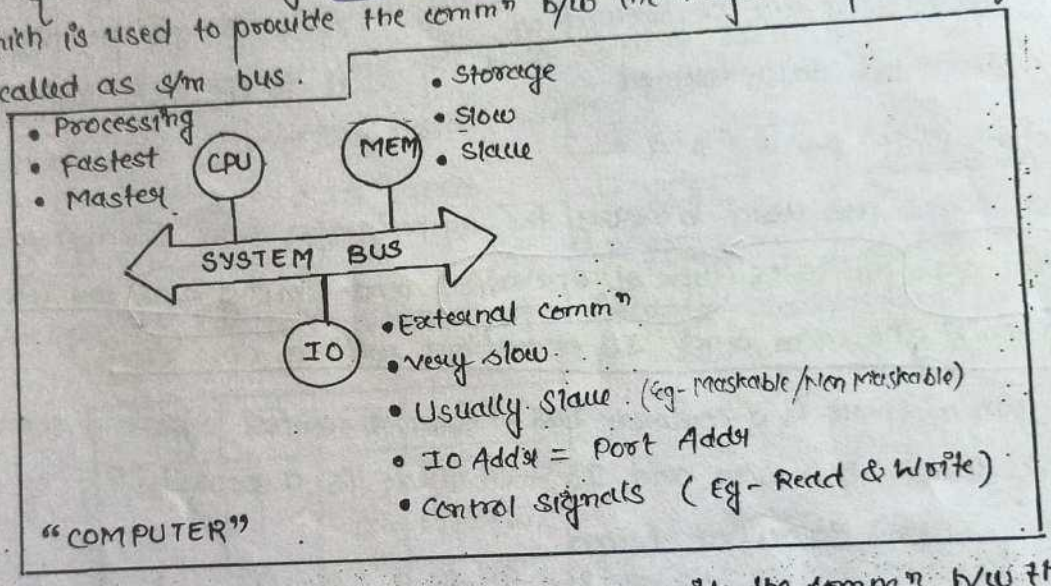
IO Unit :-

Any device which is connected to a detachable ports of a CPU is called as peripheral device. Peripheral devices are of two kinds -

1) Input Devices
2) Output Devices

System Bus Design :→

- Bus is a comm^n channel
- Characteristic of a bus is shared transmission medium.
- Limitation of a bus is only one transmission at a time. ***
- A bus which is used to provide the comm^n b/w the major components of a comp is called as s/m bus.



- S/m bus contain 3 categories of a lines used to provide the comm^n b/w the CPU, M/m and IO name as - 1) Address lines (BUS)
  2) Data lines (BUS)
  3) Control lines (BUS)

1) Address Lines :→ These lines are used to convey the addr to m/m and IO. Addr. lines are unidirectional.

Based on the width of an addr bus we can determine the capacity of a m/m system. i.e. In 8085

$A_{15} - A_8$ $AD_7 - AD_0$
↓
16 Addr lines
↓
2^{16} cells
↓
64 K cells → 64 KB

In 8086

$A_{19} - A_{16}$ $AD_{15} - AD_0$
↓
20 Addr lines
↓
2^{20} cells
↓
1 M cells
↓
1 MB

Data Lines :→

- These lines are used to convey the binary sequence b/w the CPU, m/m and IO so data lines are bidirectional. Based on the width of a data bus we can determine the word length of a CPU.
- Based on the word length we can determine the performance of a CPU i.e.

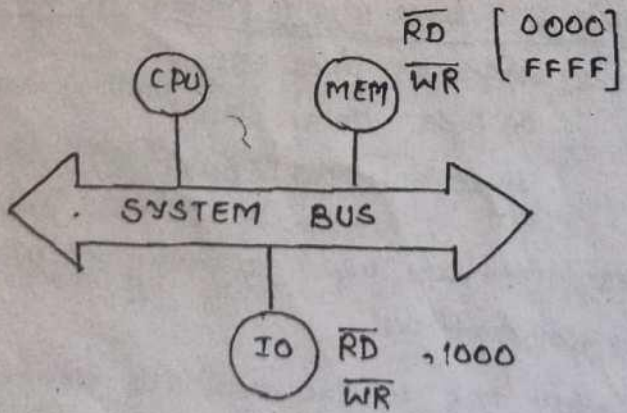
In 8085, $AD_7 - AD_0$ → 8 Data Lines → Word length = 8 bit → Operations are performed on a 8-bit data format.

Similarly in 8086, $AD_{15} - AD_0$ → 16 Data Lines → Word length = 16 bit → Operation are performed on a 16 bit data format.

Control Unit :→

- These lines are used to convey the control s/gs and timing s/g.
- Control s/gs indicates type of operation and timing s/gs are used to synchronize the m/m and IO operations with a CPU clock.

Note :- When there is a common bus, common control s/gs, & common address space is used b/w m/m and IO then there is a possibility of conflict (comm^n problem) described below ---



CPU generates the m/m request :

1000	$\overline{RD}$

1000 → AL's
$\overline{RD}$ → CL's

→ Mem controller: 1000 – valid, $\overline{RD}$ – valid } M/m open
→ IO controller: 1000 – valid, $\overline{RD}$ – valid } IO open

} Conflict

To handle the above conflicts, bus configurations are used in the comp design. They are -

(i) Iop [Input Output Processor] → Bus is not common

(ii) Isolated - IO [IO - mapped - IO] → Control s/g is not common

(iii) Mem - Mapped IO → address space is not common.

Iop :→ This configuration uses the common control s/gs and common addr space but diff. buses for both m/m & IO.

• Additional h/w is required to manage the m/m bus and IO bus so it is expensive. ∴ It is suitable in the high performanced CPU design.

Isolated - IO :→ This configuration uses the common bus and common addr space but diff. control s/gs for both m/m & IO.

Control s/g's → m/m based → $\overline{MEMRD}$; read
→ $\overline{MEMWR}$; store
IO based → $\overline{IORD}$; In
→ $\overline{IOWR}$; out

IO/$\overline{M}$ H/w Pin is used to implement the isolated - IO design.

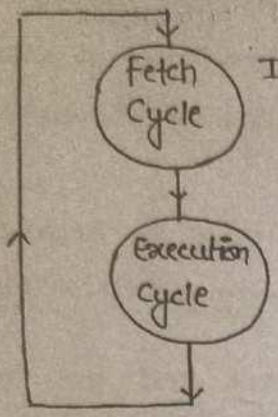
IO/$\overline{M}$	$\overline{RD}$	$\overline{WR}$	CS
0	0	1	$\overline{MEMRD}$
0	1	0	$\overline{MEMWR}$
1	0	1	$\overline{IORD}$
1	1	0	$\overline{IOWR}$

Memory Mapped IO :→ This configuration uses the common bus and common control s/g but unique addr space for both m/m & IO.

m/m addr space is shared to IO ports so limitation

Instruction Cycle :→

This concept describes the execution sequence of a user prog. It contains 2 subcycles — 1.) Fetch Cycle
2.) Execution Cycle.



IF [Instr Fetch]

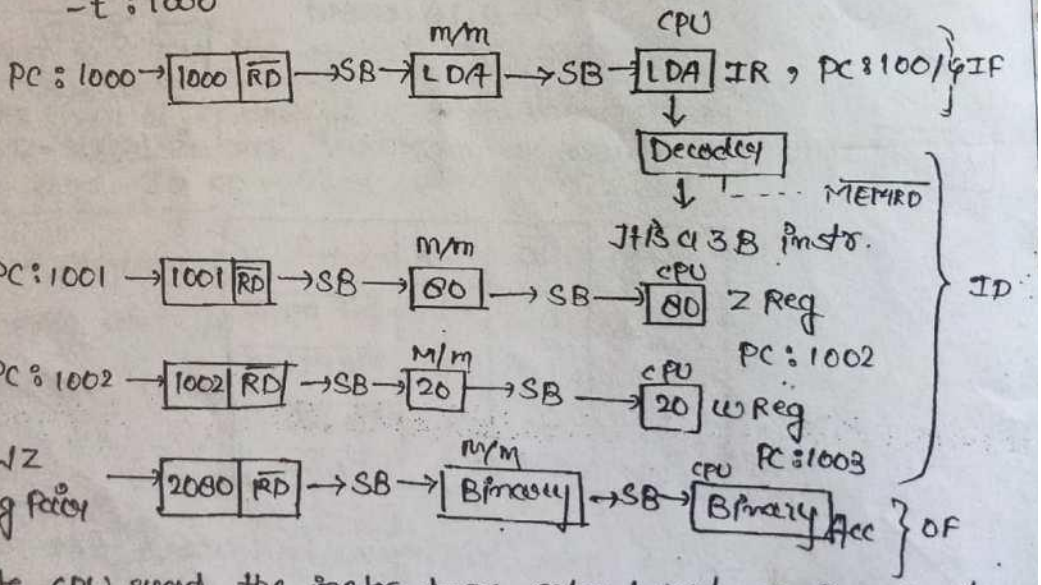
ID [Instr Decode]
OF [Operand Fetch]
PD [Process Data]
WB [Write Back]
{Operand Store}

Eg:→ Instruction : LDA [2080] ; 3B long

org 1000 →

1000	LDA
1001	80
1002	20
1003	Next

→t : 1000



IF :→ In this state CPU read the instr from m/m based on Prog Counter

ID :→ In this state CPU enables the respective h/w based on the opcode to perform the operation.

OF :→ In this state, CPU reads the data based on the addressing modes.

PD :→ In this state, i/p data is processed on a enabled h/w.

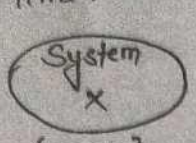
WB :→ In this state, result will be placed into a destination based on the addressing modes.

Note :→ In the Von Neumann m/c (general purpose comp) prog will be executed instr by instruction in a sequence.

Performance Analysis :→

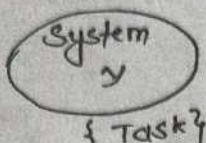
- Performance is an indirect measurement, depends on the execution time.
- Amount of the time required to complete the prog. execution is called as execution time.

Ex:-



{ Task }

$ET_x = 5ns$



{ Task }

$ET_y = 10ns$

$$\text{Performance} \propto \frac{1}{\text{Execution Time}}$$

- Prog execution time means CPU time, calculated based on the processor clock i.e. CPU Time = # Seconds / prog.

$$\#\frac{\text{Instr}}{\text{prog}} * \#\frac{\text{cycles}}{\text{Instr}} * \#\frac{\text{seconds}}{\text{cycle}}$$

$\#\frac{\text{Instr}}{\text{prog}}$ → Instr count (IC)

$\#\frac{\text{cycles}}{\text{Instr}}$ → CPI

$\#\frac{\text{seconds}}{\text{cycle}}$ → Cycle Time

execution time

$$\text{CPU Time} = IC * CPI * \text{Cycle Time}$$

$$CPI = \frac{\text{Total CPU clock cycle for the program}}{\text{Instruction count}}$$

In the prog, diff. instr. takes diff. time to complete.

$$\text{Prog ET} = \sum (IC_i * CPI_i) \text{ Cycle Time}$$

i = Type of Instr.

$$CPI = \frac{\sum_{i=1}^{n} (CPI_i \times I_i)}{\text{Instruction Count}}$$

I_i → no of instruction of a particular category in instruction

CPI_i → CPI of a particular category instruction

- Speed Up Factor is used to measure the performance gain.

i.e. $$S = \frac{\text{Performance}_x}{\text{Performance}_y}$$

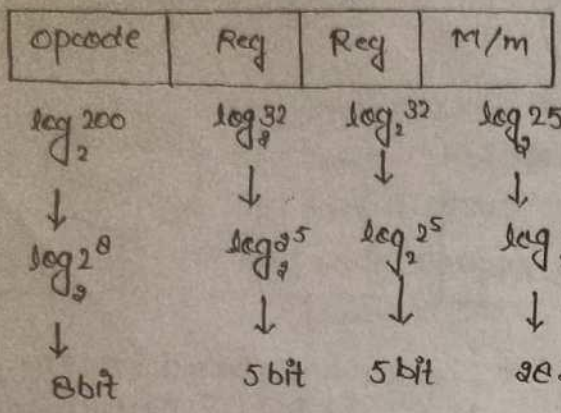
$$S = \frac{\frac{1}{ET_x}}{\frac{1}{ET_y}} = \frac{ET_y}{ET_x} = n$$

$S = n$ & where n = some const.

System X will run n times faster than System Y.

Ques- Consider a hypothetical processor which supports instr design with opcode, 2 - register operands and 1 m/m operand. CPU supports 200 instr, 32 registers and 256 MB m/m. How many bits are required to encode the instr. (instruction size).

Solution :→ Instruction Design



Instr. Size = (8 + 5 + 5 + 28) bits

= 46 bits

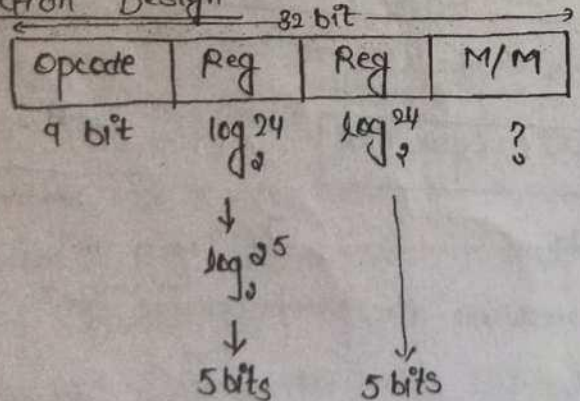
Ques :→ Consider in 16-bit hypothetical processor which supports 2 word instr. Instr contain opcode, 2 reg operands and 1 m/m operand. Processor contain 9 bit opcode and supports 24 registers. How much main m/m is possible in the comp.

Word Size = 16 bit

Instr. Size = 2 Words

= 2 × 16 = 32 bit

Instruction Design

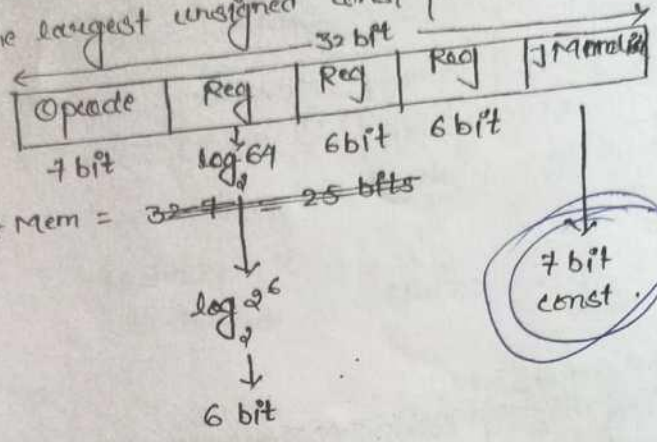


M/m addr = 32 - (19) = 13 bit

Mem Size = 2^{13} cells

= $2^3 \cdot 2^{10}$ = 8 KB cells.

Ques :→ consider 32 bit hypothetical processor 1 word instr. Instr. contain 3 reg operands, immediate operand along with a 7 bit opcode field. Processor supports 64 registers. What is the largest unsigned const possible in the instr.



① Smallest unsigned data = 0

② Largest " " = $2^7 - 1 = 127$

③ Smallest signed data = $-(2^{7-1}) = -2^6 = -64$

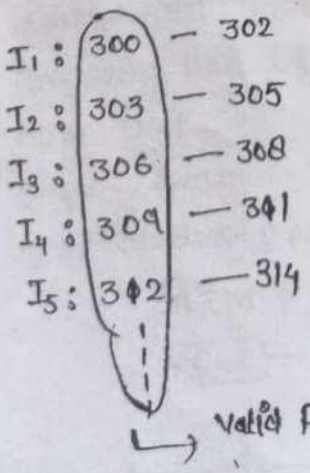
④ Largest " " = $+(2^{7-1} - 1) = +2^6 - 1 = 63$

⑤ # unsigned / signed const = $2^7 = 128 - 1 = 127$

Ques :→ A CPU has 24 bit instr. Prog is stored in the m/m with a starting address of 300 decimal onwards. Which one of the following is a valid prog. counter.

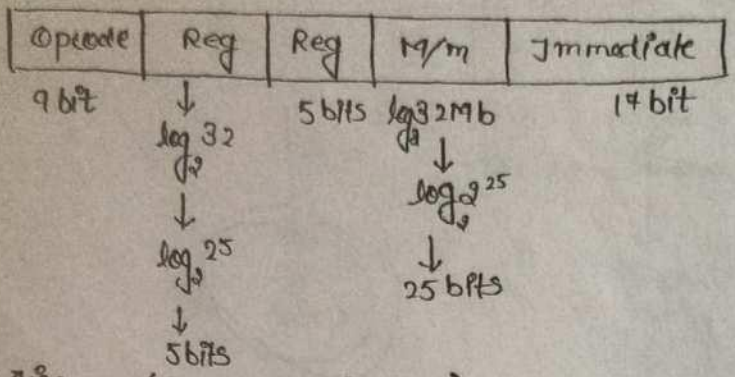
a.) 400 b.) 500 c.) 600 d.) 700

Soln :→ Instr size = 24 bit
= 3B { 3 m/m cells are required }



↳ valid PC is a multiplication factor of 3.

Ques:→ Consider a hypothetical CPU which supports instr. with 9 bit opcode, 2 reg addr fields, m/m addr field and 17 bit immediate field. Processor supports 32 reg and 32 Mb m/m. Prog contains 100 instr. how much space is required to store the prog in Bytes.



Instr size = (9+5+5+25+17)
= 61 bits = { occupies 8 cells of m/m space }

space Required for 100 instr = [illegible] × 100 × 8
= 800 B

Ques:→ Consider a hypothetical processor which is operating with a 500 MHz clock freq, consumes 9 cycles for load instr., 7 cycles for ALU instr. & 4 cycles for branch instr. Relative frequencies of these instr. are 40%, 40% and 20% resp. What is the performance of a CPU in terms of MIPS (Million Instr. per sec)

Avg Instr ET = $\frac{\text{Time Req for prog}}{\text{\# Instr. in the prog}}$

clock time $\propto \frac{1}{\text{Clk freq}}$

cycle time = $\frac{1}{500\text{ MHz}}$ sec = 2 nsec

$= \frac{[(0.4*9) + (0.4*7) + (0.2*4)]\ 2\text{nSec}}{(0.4+0.4+0.2)}$

= 14.4 ns

∴ 1 instr take —— 14.4 ns.

∴ $\frac{1}{14.4\text{ ns}}$ instr —— 1 sec

∴ $\frac{1*10^9}{14.4}$ sec = 0.0694 * 10^9 sec

= 69.4 MIPS

Ques :→ Consider 1.2 n Sec clock cycle process which consumes 8 cycles for load and store instr. 4 cycles for ALU instr and 3 cycles for branch instr. Prog contain 60 load and store instr and 40 ALU instr and 50 branch instr. what is the prog execution time.

Soln :→ 1.2 n sec

$$\text{Prog ET} = \sum (IC_i * CPI_i) \text{ cycle time}$$
$$= [(60*8) + (40*4) + (50*3)] 1.2 ns$$
$$= 948 \text{ nSec}$$

View - 2 :→

Based on the performance, comp. architecture is divided into 2 types
1) Low performance CPU Design (General purpose comp)
2) High performance CPU Design (super comp)

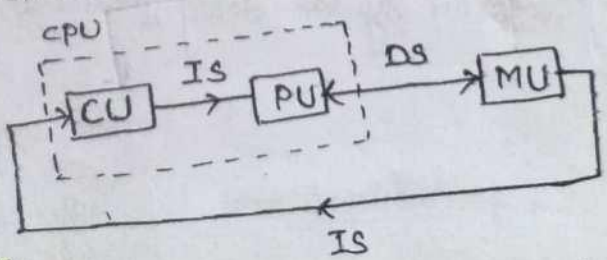
- High Performance arch. exploits the concurrency. Concurrency means two or more instr execution at a time.
- Acc to a Flynn's classification, comp. architecture is of 4 kinds named as – 1) SISD (Single Instr Stream & Single Data Stream]
  2) SIMD [" " " " Multiple " "]
  3) MIMD [Multiple " " " " " "]
  4) MISD [" " " " Single " "] X Not in Practice

Short cuts :-

CU : Control Unit
PU : Processing Unit (ALU)
MU : M/m Unit
Is : Instr. Stream
DS : Data Stream

CPU → APU

SISD :→



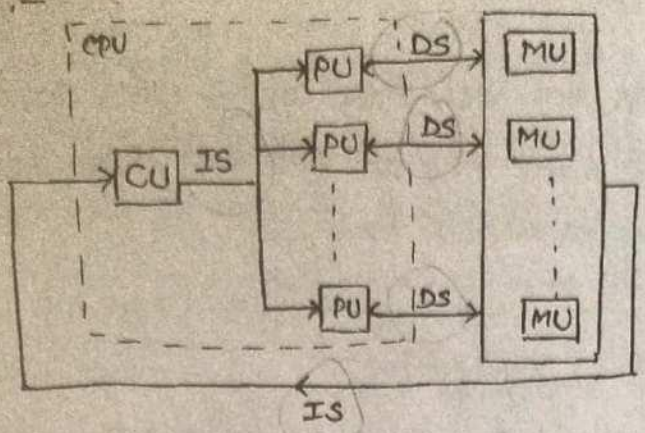
Implemented as a Uniprocessor s/m Design

Eg :→ 8085 µP
8086 µP

(No Concurrency)
↓
Adding pipelining satisfies concurrency

SIMD :-

{Concurrency Present by adding extra h/w}



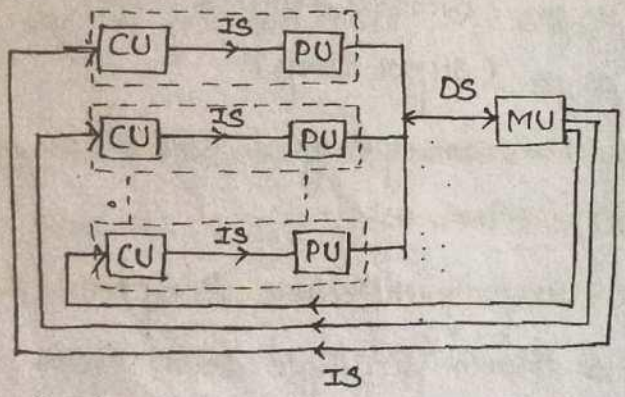
Implemented as an array processor s/m design

[Vector processor]

Eg :- Staran Processor
PEPE Processor

Ex. multiplication of matrix (MATMUL)

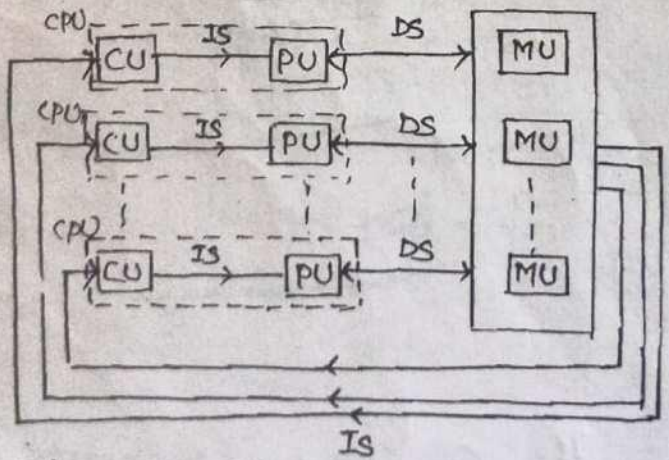
MISD :- (multiple instruction stream and single data stream)



This Architecture contains multiprocessors but 1 processor is in use at a time.

This architecture is not yet employed.

MIMD :-



Concurrency present ↓ by adding extra hardware.

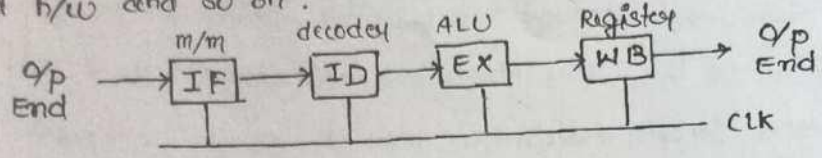
Implemented as a multiprocessor system design

Eg :- Cray Processor
Cybey Processor

Note :→ In the SISD architecture, pipelining concept is used as an enhancement technique to improve the performance.

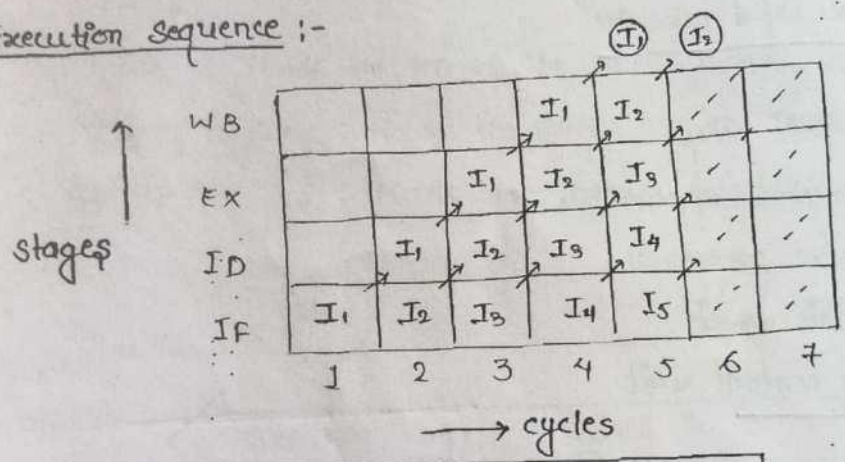
SISD with Pipeline :→

Pipelining decompose the prob statement into an independent sub-problems, assign them into an independent h/w, later connect h/w into a pipeline sequence i.e. 1st h/w o/p is connected as i/p i/p to 2nd h/w and so on.



Execution sequence :-

WB = Write Back



Cycle Per Instruction (CPI) = 1

View - 3 :

[RISC Vs CISC]

Characteristics of RISC (Reduced Instr Set Computer)

1) It supports more registers. x all operation are done within the register of the CPU.
2) It supports less addr modes.
3) It supports fixed length instr.
4) It supports successful pipeline.
5) CPI = 1
6) It is a super comp.
7) It is used in the real time applications.

8.) It is highly expensive processor.
9.) It contains smaller instr. set.
10.) It uses hard-wired control unit.
Eg:- Motorola Processor
Power PC "
ARM (Advanced Risc M/c) Processor.

• Characteristics of CISC (Complex Instr set comp):-
1.) It supports less registers.
2.) It supports more addr modes
3.) It supports variable length instr.
4.) It supports unsuccessful pipeline
5.) CPI ≠ 1
6.) It is a general purpose comp.
* 7.) It is used in the personal applications.
* 8.) It is a less expensive processor.
9.) It contains larger instr set.
* 10.) It uses micro prog control unit
Eg:- Pentium Processor

Computer Organization :→

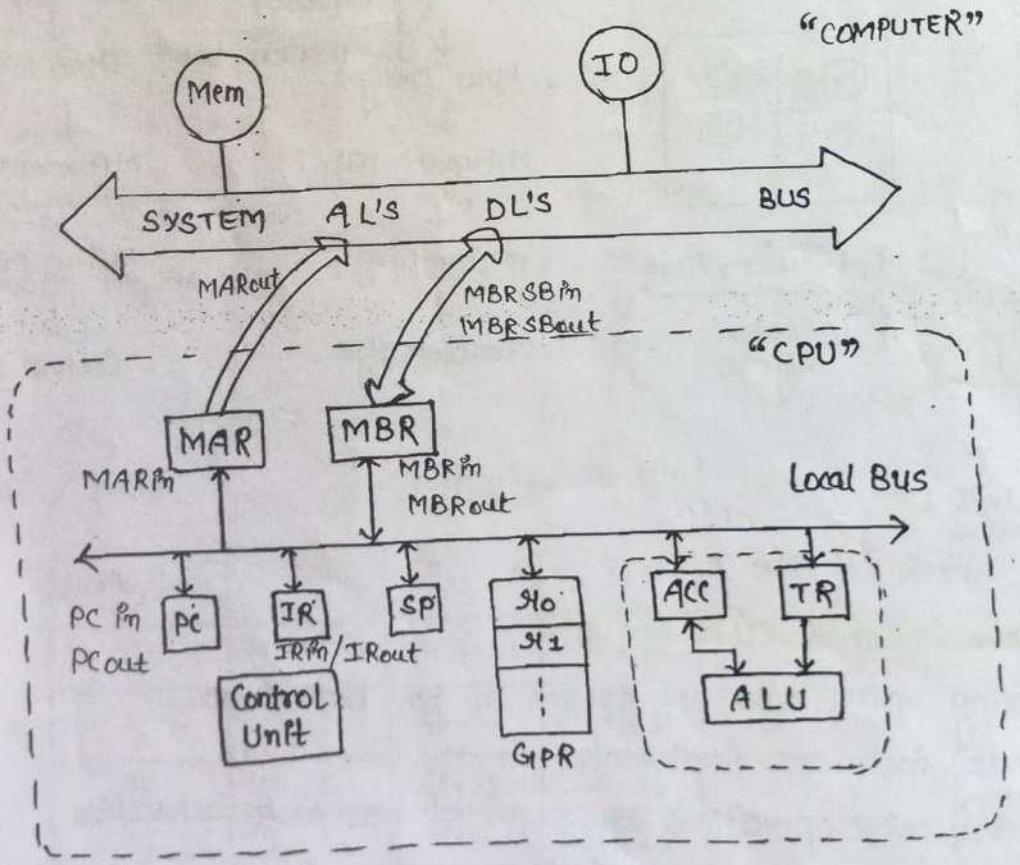
Comp. s/m contains 3 fundamental components named as -
1.) CPU
2.) Memory
3.) IO

1.) CPU Organization :-
CPU contain 3 internal components used to process the instructions named as 1.) Registers
2.) ALU
3.) Control Unit (CU)

1.) Registers :-

Registey is a internal storage element of a CPU .·. every CPU contain a minimum set of a mandatory registers described below ---

1.) PC : Holds the Instr addr
2.) IR : Holds the Opcode (Instr Reg)
3.) Acc : Holds the ALU i/p & ALU o/p
4.) TR : Holds the ALU 2nd operand (Temp Reg)
5.) MAR : Holds the m/m addr connected to a addr lines of s/m bus
6.) MBR : Holds the m/m content, connected to a Data lines of s/m bus (MDR)
7.) SP : Holds the top of the stack Addr.
8.) Flag Register : Holds the status of an Instr.
9.) GPR : Holds the Data (General purpose Reg)



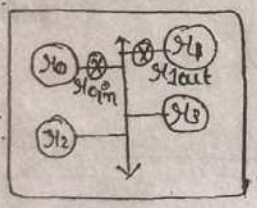
Micro-Operation :→

- Micro-operation is a primitive or elementary operation in the base h/w
- One cycle based operation is name as a micro operation (micro operation takes 1 cycle to complete).
- Register to register transfer operation is a one kind of microoperation
- When the micro-opr. contain m/m reference then it takes more than one cycle to complete.
- Micro instruction may contain one or more micro operations. Micro-instr also takes one cycle to complete because all the microoperⁿ present in the micro instr is executing in parallel.
- Control s/gs are required to execute the microoperⁿ.

* Machine Instr : Mov R0, R1

Reg Transfer Language (RTL) : $R_0 \leftarrow R_1$

H/W Design :



Micro Operation List : T_1 : R1out, R0in (Control Signals)

↓

Micro Prog

* Add R0, [2000]

↓

$R_0 \leftarrow R_0 + M[2000]$

↓

H/W Design

↓

Microoperⁿ List

↓

Micro prog

↓

Control s/gs

* MUL R0, R1

↓

$R_0 \leftarrow R_0 * R_1$

↓

H/W Design

↓

Microoperⁿ List

↓

Micro Prog

↓

Control s/g's

Control Unit :→

- CU is a brain of the CPU.
- Prerequirements of a CU design is —
  i) How many control s/gs are present in the base h/w.
  ii) How many instr are implemented in the base h/w.
  iii) How many micro operations are required for each instruction.
  iv) What are the control s/gs required for each microoperation, for each instr.

After finalising above requirements CU is implemented using following approaches —

1.) Hard Wired Approach

2.) Micro Programmed Approach.

Hard Wired Control Unit :→

1.) In this CU, control s/g is expressed in a SOP (Sum of Product) expression format.

2.) Control expression is directly realised by the independent h/w called as hard-wired CU.

3.) It is a fastest CU.

4.) It is used in the real time applications.

5.) Even a minor modification requires redesign and reconnection of a CU Hence it is not flexible.

6.) It is not suitable in the design and testing process (trials)

Hard Wired CU :-

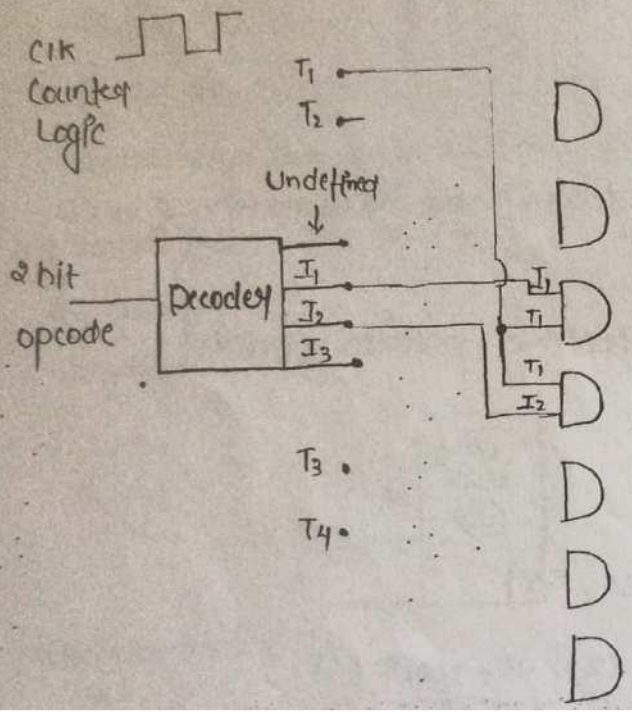
{"SOP Formatted CS"}

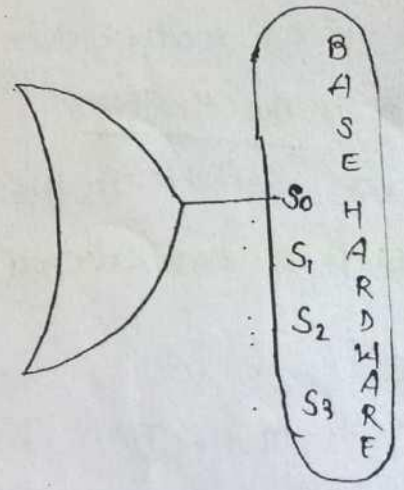
$S0 : T_1(I_1 + I_2) + T_2 I_1 + T_3(I_1 + I_3) + T_4(I_1 + I_3)$

$S1 : \underbrace{T_1(I_1 + I_2 + I_3)}_{T_1} + T_2(I_2 + I_3) + T_3 T_1 + T_4(I_2 + I_3)$

$S_2 : T_1 I_2 + T_2 + T_3 I_2 + T_4 I_2$

$S_3 : T_1(I_1 + I_3) + T_2 I_3 + T_3(I_2 + I_3) + T_4(I_1 + I_3)$





Ques:- Consider a hypothetical CU which supports following sequence of actions to say execute various instr.

I_1

T_1 : X_{in} , Y_{out} , Z_{in}

T_2 : X_{out} , Y_{in} , Z_{out}

T_3 : X_{in} , Y_{out}

I_2

T_1 : Y_{out} , X_{in} , Z_{out}

T_2 : X_{out} , X_{in} , Y_{in}

T_3 : Z_{out} , Y_{in} , X_{in}

I_3

T_1 : X_{out} , Y_{in}

T_2 : X_{in} , Z_{out} , Y_{in}

T_3 : Y_{out} , X_{in}

What is the control s/g for X_{in} s/g.

$$X_{in} = T_1 (I_1 + I_2) + T_2 (I_2 + I_3) + T_3 (I_2 + I_3) I_1)$$

Ques:- Consider the following control expression to generate the A_{in} signal in a hypothetical CU.

$$A_{in} = T_1 + T_2 (I_2 + I_3) + T_3 + T_4 (I_1 + I_4)$$

During the execution of I_4 instr, in which eqn A_{in} s/g is disabled in the h/w

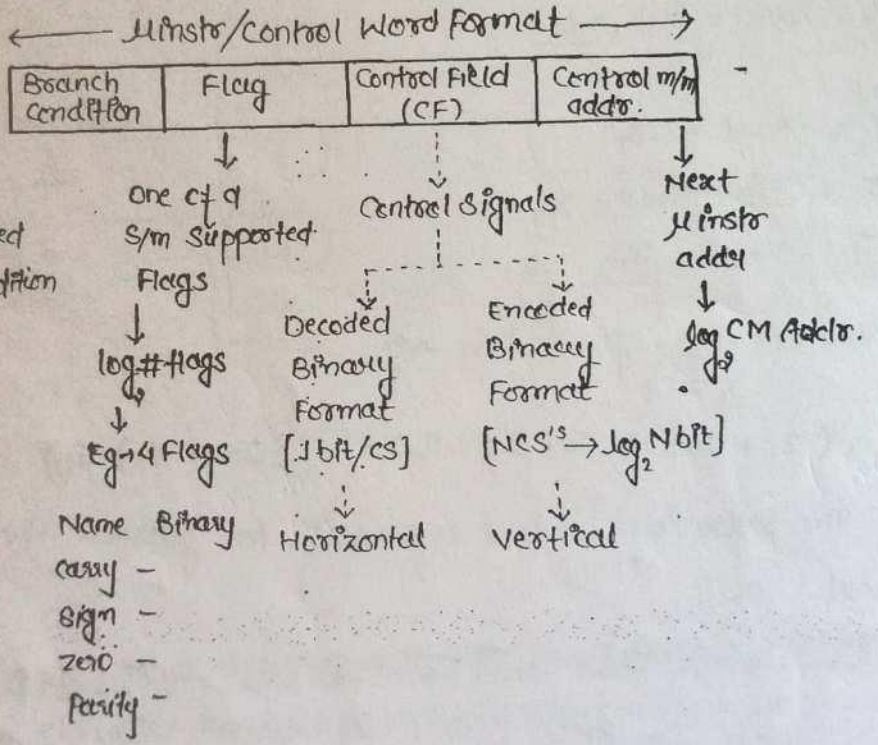
$$A_{in} = T_1 + T_2 (I_2 + I_3) + T_3 + T_4 (I_1 + I_4)$$

$\{I_1, I_2, I_3, \textcircled{I_4}\}$ $\{I_2, I_3\}$ $\{I_1, I_2, I_3, \textcircled{I_4}\}$ $\{I_1, \textcircled{I_4}\}$

Ans = T_2

Micro Program Control Unit :→

- In this design control m/m is programmed with a micro programs.
- Control m/m is a permanent m/m i.e. ROM.
- In this design microsequencer units is present to generate the μ instr addrs in a sequence.
- Control m/m contain CAR register used to hold the control m/m addrs and CDR register used to hold the control m/m content.
- In the control m/m, μ instr are stored in the following format.



Based on the style of representing the control sigs, μ instr is of 2 types -

1.) Horizontal μ instr.

2.) Vertical μ

Based on the type of μ instr. programmed in the control m/m, CU is 2 types

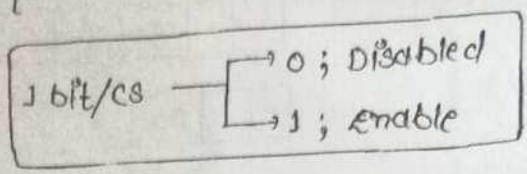
① Horizontal μ programmed CU

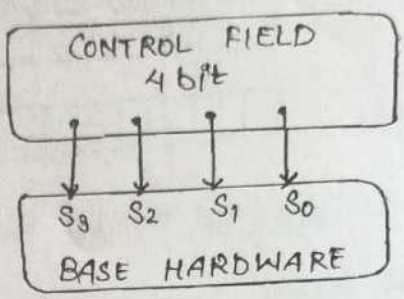
② Vertical μ μ

1.) Horizontal Microprogrammed CU :→

① # Control S/g's in H/w : {S0, S1, S2, S3}

② Decoded Binary form of a Control Signal (CS) :





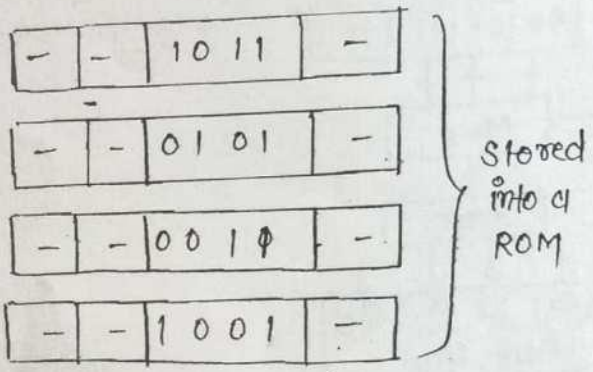
③ μInstr Design :

(I1) T1 : {S0, S1, S3}

T2 : {S0, S2}

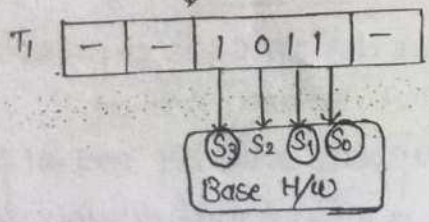
T3 : {S0, S1}

T4 : {S0, S3}



④ Operational State :

(I1) Read T1 from ROM

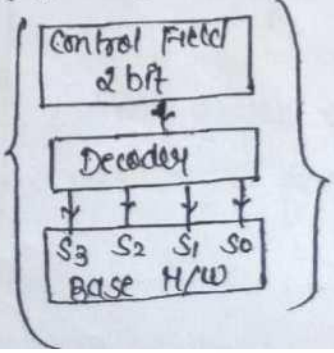


Vertical :→

① # CS's in H/w : {S0, S1, S2, S3}

② Encoded Binary Form of a CS : $\log_2 4$ = 2 bit/CS {Function code (FC)}

FC	CS
00	S0
01	S1
10	S2
11	S3



③ Microinstr Design :-

(1) T_1 {S_0, S_1, S_3} | - | - | 00 | 01 | 11 | - |

T_1 {S_0, S_2} | - | - | 00 | 10 | - |

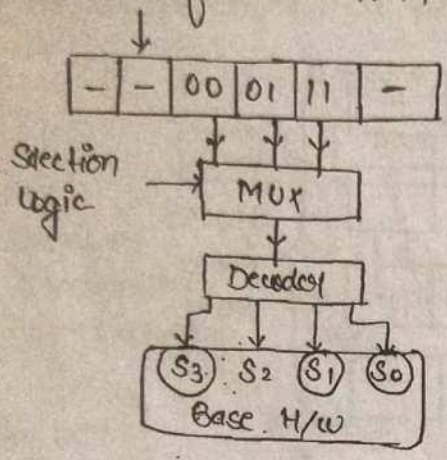
T_1 {S_0, S_1} | - | - | 00 | 01 | - |

T_1 {S_0, S_3} | - | - | 00 | 11 | - |

⇓

(4) Operational State — Stored into a ROM

(1) Read T_1 from the ROM



Horizontal Vs Vertical

Horizontal (no decoder ckt → so no delay)

- In this design CS expressed in a decoded binary format.
- It supports longer control field.
- No need of external decoders to generate control s/gs. Hence it is faster than vertical.
- It allows high degree of parallelism i.e. None or more than 1.
- It is a bit flexible compared with hard wired CU.
- It is not in practice because directly manage the decoding s/g become very complex.

Vertical (have decoder ckt → have delay)

- In this design CS is expressed in an encoded binary format.
- It supports shorter control field.
- Need of an external decoder to generate the control s/gs. Hence it is slower than horizontal.
- It allows low degree of parallelism i.e. none or one.
- It allows easy implementation of new instructions. Hence is more flexible.
- It is used in the CISC comp so default µprog control unit is vertical CU.

Note :→ Ascending order of a CU design

a.) in terms of speed is

vertical < horizontal < hard wired.

b.) In terms of flexibility is

Hard wired < horizontal < vertical.

Ques :→ Consider an hypothetical CU which supports 4k control word m/m. Hardware contains 64 control sigs and 16 flags. What is the size of a control word in bits and control m/m in bytes using -

a.) Horizontal Programming

b.) Vertical "

Soln :→ a.) Horizontal

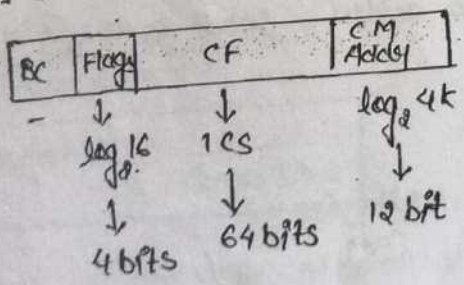
CS's in H/w = 64

Decoded form of CS = 1 bit / CS

= 64 bits

μ instr design with

Default "1 CS"



HCW size = 4 + 64 + 12 = 80 bits

HCM Size = 4 kCW

↓

4k x CW

4k x 80 bits

↓

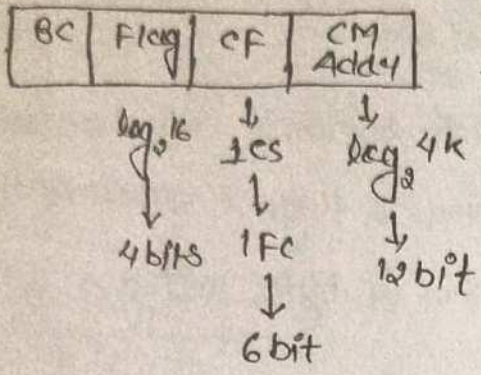
$\left[\frac{4k \times 80}{8}\right]$ Bytes

b.) Vertical

CS's in HW = 64

Encoded binary form of a CS = $\log_2 64$ = 6 bit/CS {FC}

μ Instr design with default 1 CS



VCW size = 4 + 6 + 12 = 22 bits

VCM size = 4k CW

↓

4k x CW

↓

4k x 22 bits

↓

$\left(\frac{4k \times 22}{8}\right)$ Bytes.

Ques:- A μ prog'd CU supports 400 control s/gs, μ instr is designed in such a way that 4 sequencing s/gs are active. What is the size of a control field in the μ instr.

CS's in HW = 400

Encoded binary form = $\log_2 400$ = 9 bits/CS (FC)

μ instr design with 4 CS

Group	G1	G2	G3	G4	G5
CS	1	10	20	30	40

Mutual Exclusive → Everyone supporting horz and vertical but one at a time.

Groups as horizontal

BC	Flag	G1	G2	G3	G4	G5	CM Addr
–	–	1 bit	10 bit	20-b	30-b	40-b	–

101 bit

Groups as Vertical

BC	Flag	G1	G2	G3	G4	G5	CM Addr
–	–	$\log_2 1$	$\log_2 10$	$\log_2 30$	$\log_2 30$	$\log_2 40$	
		↓	↓	↓	↓	↓	
		1 bit	4 bit	5 bit	5 bit	6 bit	

21 bits

bits saved = 101 − 21
= 80

Memory Organization :→

Performance $\propto \frac{1}{\text{Access Time}}$

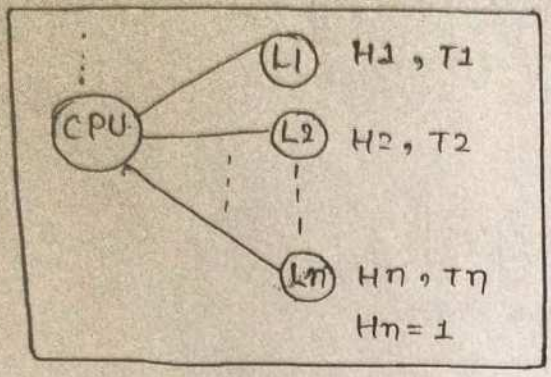
Types :- Based on the style of accessing the s/m supported memories, m/m org^n is of 2 types –

1.) Simultaneous Access. M/m org^n
2.) Hierarchial Access M/m org^n

Simultaneous Access :→

In this org^n, CPU directly connected to all the levels of memories but accessing the m/m's in a sequence ie. when there is a Mis in level 1 m/m then CPU directly access the data from level 2 m/m without copying into level 1.

When there is a MIS in level-2 m/m then CPU directly access the data from level-3 m/m without copying into level 2 and level 1 and so on.



Here H1, H2 --- Hn are the Hit ratios of respective m/m's.

$$\text{Hit Ratios} = \frac{\#\text{ Hits}}{\text{Total \# access}}$$

H : {0 to 1}

T1, T2, --- Tn are the access times of a respective m/m's.

Amount of the time required to access one word of data from the m/m is called as Avg Access Time (Tavg)

$$T_{avg} = H_1T_1 + (1-H_1)H_2T_2 + (1-H_1)(1-H_2)H_3T_3 + \text{---} + (1-H_1)(1-H_2)\text{---}(1-H_{n-1})H_nT_n$$

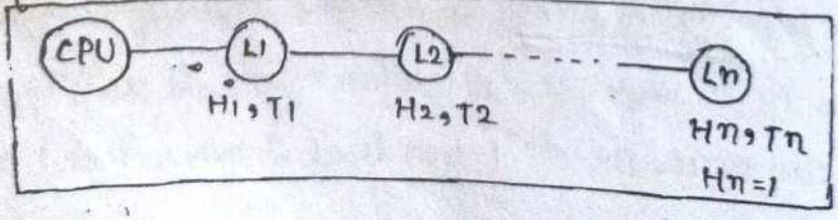
1 Word ——— Tavg

? # Words ——— 1sec

$$\eta_{mem} = \frac{1}{T_{avg}} \text{ words/sec}$$

2) Hierarchial Access :-

In this orgn CPU always access the data only from the level 1 m/m. When the operⁿ is MIS in level 1 then the respective data block will be transferred from higher levels to level 1 m/m, later CPU access the data only from the level 1 m/m.



Ex:- Prog :-

I_1 : Data (L2 m/m)
I_2 :
I_3 : Data
I_4 :
I_5 : Data
I_6 :
I_7 : Data
I_8 :
I_9 : Data

Reuse 'I1' Data

L1 M/m → 2ns
L2 M/m → 10ns

Simultaneous

I1 : ML1 , T2
I3 : ML1 , T2
I5 : ML1 , T2
I7 : ML1 , T2
I9 : ML1 , T2

Total Time
= 5T2
= 5*10
= 50ns

Hierarchical

I1 : ML1 , (T2+T1)
I3 : ML1 T1
I5 : ML1 T1
I7 : ML1 T1
I9 : ML1 T1

Locality of Reference

Total Time = (1*T2) + 5T1
= 10 + 10
= 20ns.

Note :- In the hierarchical m/m design, avg accessing time is minimised because of locality of reference.

- Locality of reference means CPU performs the operations on a reusable data space. It is of 2 types -
  1. Temporal Locality ; Means CPU refers the same word again and again in a near future.
  2. Spatial Locality : Means adjacent words in the same block is referenced by the CPU in a sequence.

- To satisfy the above locality of references, we need to transfer the data from higher level to lower level in a blockwise where block contain multiple words.

Note :- When the question contain hierarchy word or hierarchy meaning or cache m/m word then use hierarchical s/m otherwise use simultaneous s/m.

Ques :- In a 2 level m/m orgn, level-1 m/m is 8 times faster than level-2 m/m and its access time is 40ns less than the avg access time. Let level-1 m/m access time is 30nsec, what is the hit ratio.

Soln :- $L_1 \longrightarrow H_1, T_1$

$L_2 \longrightarrow H_2, T_2$

$\boxed{H_2 = 1}$

Simultaneous Orgn

$$S = \frac{T_2}{T_1}$$

$$8 = \frac{T_2}{T_1} \Rightarrow T_2 = 8T_1$$

$$T_1 = T_{avg} - 40ns$$

$$T_{avg} = T_1 + 40ns$$

Let $T_1 = 30ns$; $T_{avg} = 30ns + 40ns = 70ns$

then $T_2 = 8 * 30ns = 240\ ns$

$T_{avg} = 30 + 40ns = 70\ ns$

$T_{avg} = H_1 T_1 + (1-H_1) H_2 T_2$

$70 = (H_1 * 30) + (1 - H_1) * 1 * 240\ ns$

$\therefore \boxed{H_1 = 0.81}$

Ques :- In a 2 level hierarchical m/m orgn, level-1 m/m access time is 100 ns and its hit ratio is 80%. Level-2 m/m access time is 500 ns. What is the avg access time of a m/m s/m.

Hierarchical Orgn

$L_1 - H_1, T_1$ { $H_1 = 80\%$; $T_1 = 100\ ns$ }

$L_2 - H_2, T_2$ { $H_2 = 1$; $T_2 = 500\ ns$ }

$T_{avg} = H_1 T_1 + (1-H_1) H_2 (T_2 + T_1)$

$T_{avg} = 200\ ns$

Ques :- Consider the following m/m orgn specifications

Level	Access Time	Hit Ratio
1	100 ns	0.7
2	500 ns	0.9
3	800 ns	1

If the referred data is not in L1 then transfer it from L1 to L2. If not in L2 then transfer it from L3 to L2 and L2 to L1 what is the avg access time of m/m s/m. (nothing but Hierarchical s/m)

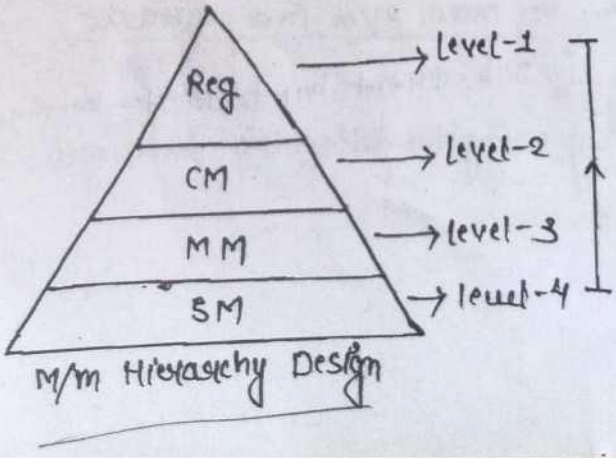
Hierarchical s/m

$$T_{avg} = H_1T_1 + (1-H_1)H_2(T_2+T_1) + (1-H_1)(1-H_2)H_3(T_3+T_2+T_1)$$
$$= 0.7(100) + (1-0.7)0.9(600) + (1-0.9)(1-0.9)1(1400)$$
$$= 274 \text{ ns}$$

Memory Hierarchy Design :→

In the comp design, m/m s/m is organised using hierarchical principles. Acc to a hier. m/m design, s/m supported m/m standards is as follows —

Levels	1	2	3	4
Name	Register	Cache M/m (CM)	Main M/m (MM)	Secondary M/m (SM)
Typical Size	< 1KB	< 16MB	< 16GiB	> 100GiB
Implementation	Customised Multiports	SRAM (flip-flops)	DRAM (capacitors)	Magnetic
Access Time	(0.25 - 0.5)	(0.5 - 25)	(80 - 250)	(5000000)
Band Width	(20000 - 1,00,000)	> (5000 - 10000)	> (1000 - 5000)	> (20 - 150)
Managed by	compiler	Hardware	OS	OS
Backed by	CM	MM	SM	Compact Disc (CD)

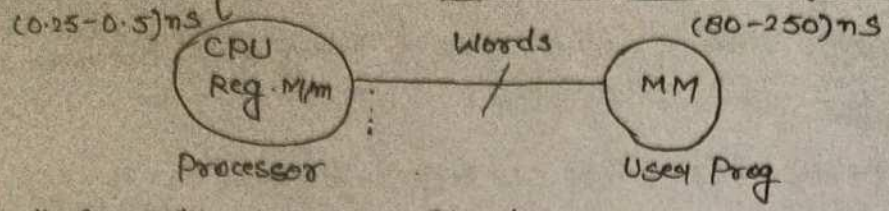


M/m Hierarchy Design

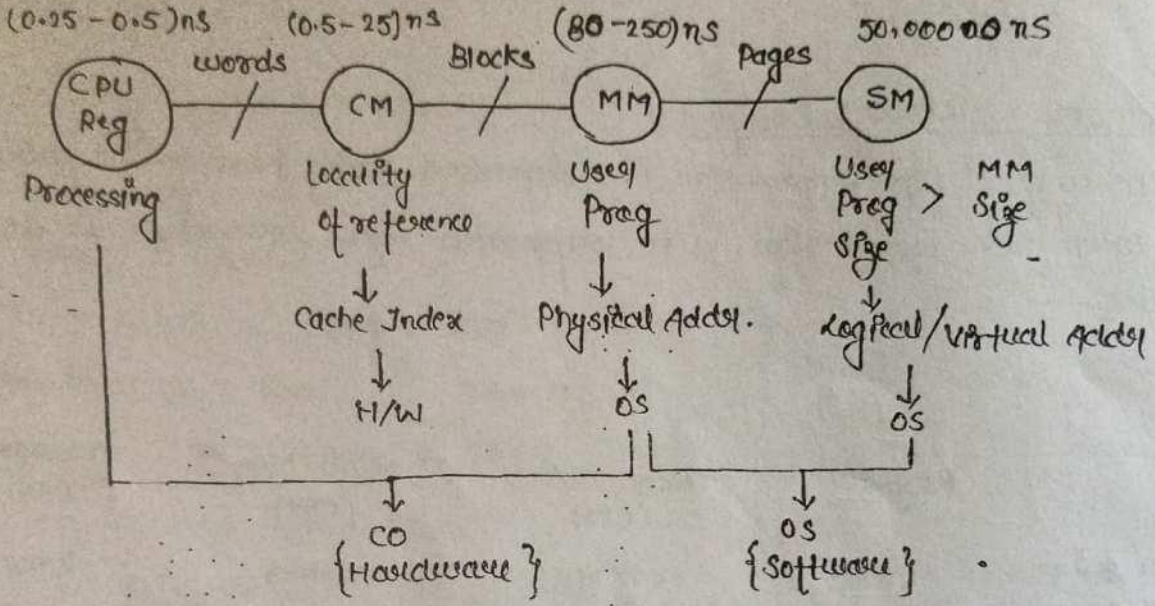
Bottom Top Approach of m/m Hierarchy states that -

① Capacity ↓
② Access Time ↓
③ Performance ↑
④ Cost / bit ↑
⑤ Expensive

Acc to m/m hierarchy design, accessing sequence of a s/m supported m/m's is as follows — * S/m Without Hierarchy structure



* S/m With Hierarch Structure



$$T_{avg} = H_c T_c + (1-H_c) H_m (T_m + T_c) + (1-H_c)(1-H_m) H_m (T_s + T_m + T_c)$$

Virtual Memory :-

① Virtual M/m concept gives the illusion (imagination) to the user that execute the huge application prog on a small amount of the main m/m space.

② Virtual m/m concept is used to increase the addr space

③ virtual m/m concept states that use the sec. m/m to store user prog, later transfer the prog from sec m/m to main m/m in a pagewise sequence to execute the user prog. (Job jarurat hogi to transfer karenge)

④ Virtual m/m concept is implemented by using the following techniques —

1.) Paging —— Fixed Partition

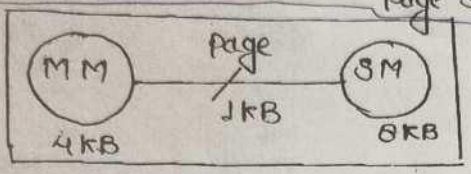
2.) Segmentation = Variable "

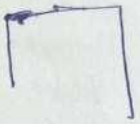
Paging :→ Acc. to a m/m hierarchy design, data will be transfered b/w sec m/m to main m/m in the form of pages so both of the m/m's are organised into a pages based on the page size.

no # Pages in sec. m/m = $\frac{\text{Sec m/m Size}}{\text{Page Size}}$

no # Page (frames) in m/m = $\frac{\text{MM Size}}{\text{Page Size}}$

{ MM → Main m/m
SM → Sec m/m

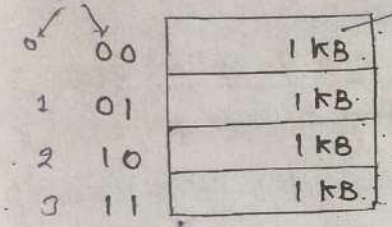




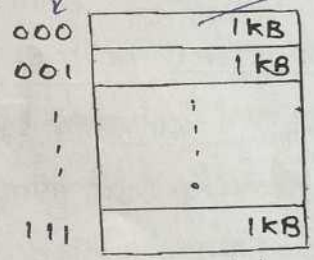
Frames = $\frac{4K}{1K}$ = 4

Pages = $\frac{8K}{1K}$ = 8

(kaunsa page) page number



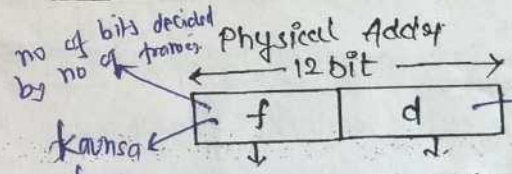
(kaunsa frame) frame no.



main memory ke me no of pages ko frame kahte hai

Address Formats :→

Main M/m size : 4 KB
↓
4K x B
↓
$\log_2 4K$
↓
$\log_2 2^{12}$
↓
12 bits
{Physical Addr}

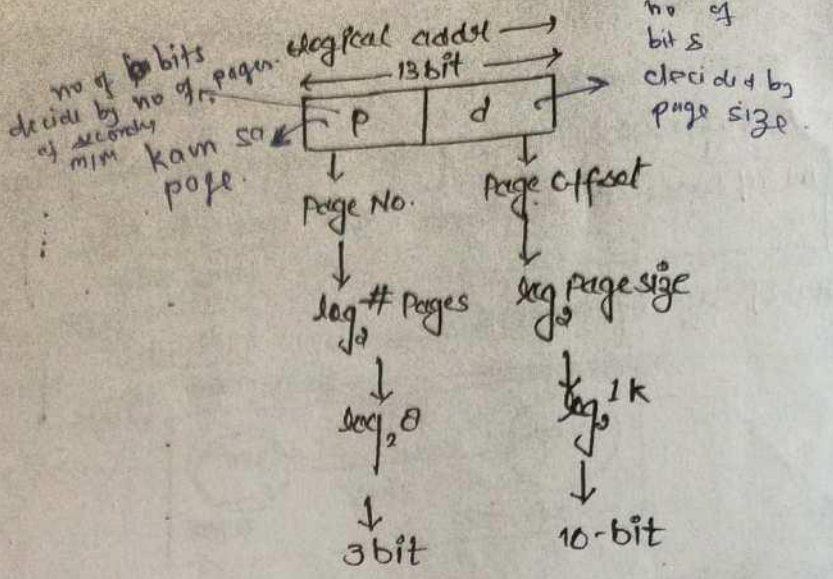


Frame No. → $\log_2$ # frames → $\log_2 4$ → 2 bit

Page Offset → $\log_2$ page size → $\log_2 1K$ → 10 bit

⇒ Logical address related to sec. memory and physical address is related to main memory.

SM size : 8KB
↓
8K x 8
↓
$\log_2 8K$
↓
$\log_2 2^{13}$
↓
13-bits Addr
{Logical Addr}

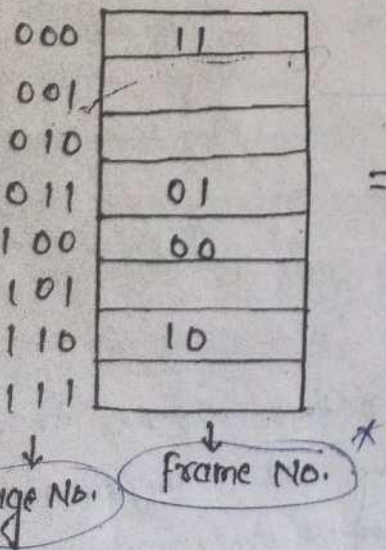


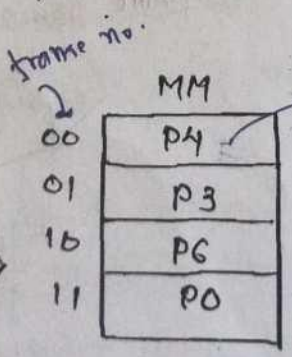
Mapping Process :→

Process of transferring data from sec m/m to main m/m is called as Mapping. Mapping process is controlled by operating S/m (OS). OS maintains page table in the main m/m to indicate the relationship b/w page no. and frame no. (mapping status)

Page table contain several entries which depends on the no. of pages in sec m/m. Each entry contain frame no.

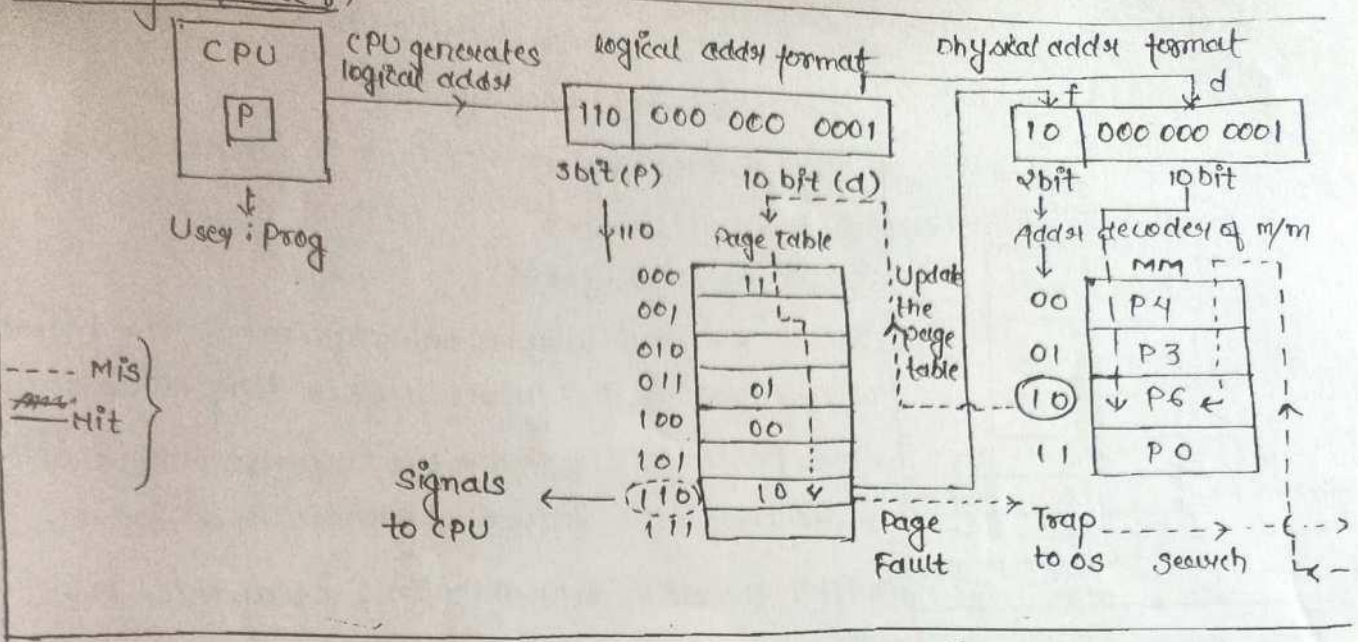
Page Table ⇒





Accessing Sequence :→

Accessing sequence :-

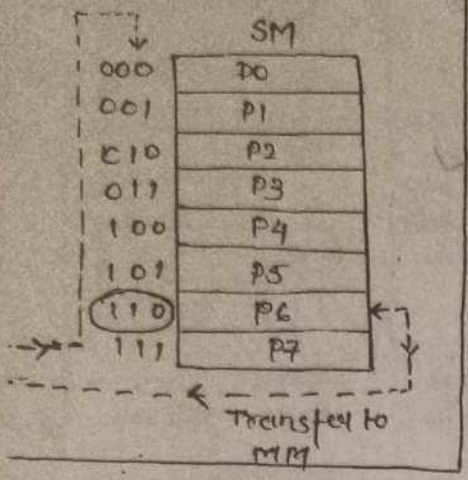


- CPU generated logical addr is interpreted as - | P | d | format.
- Based on the p-value, respective entries will be enabled in the page table.
- When the enabled entry contain frame no. then operation become page hit. so physical addr is formed by taking the frame no. from the page table.
- Physical addr is interpreted as | f | d | format
- Based on a f value, respective frame is enabled in the main m/m so CPU will be accissing data from main m/m page offset d value.

Page Fault Service :-

- When the enabled page table entry is empty then the oper^n is page fault.
- When page fault occurs CPU generate trap s/g to os to demand the page called demand paging.
- OS find out the resp. page in sec. m/m, transfer to main m/m based on the CPU demand. (existing page ko new page se replace karte hai jiski from ab jaroorat hai)
- In this process OS uses the replacement policies to replace the main m/m data when the main m/m is full. They are -

1) FIFO : (First in First Out)



Replace the page having longest time stand.

② LRU (Least Recently Used) :

Replace the page present in m/m longest time without reference.

③ Optimal Replacement :

Replace the page which is not required in the future or required in the future after a long duration.

- This policy is not in the use because CPU unable to identifies the future references in advance.
- After the data transmission, os updates the page table. Later o/p to CPU.
- Let, P is a page fault rate,
  S is a page fault service Time
  Tm = main m/m access time.

then, Effective M/m time calculated as

$$EMT = (P * S) + (1 - P)\, Tm$$

Ques:- Consider a hypothetical s/m which supports 28 bit logical addr space & 24 bit phy. addr space. M/m s/m is organised in to a 64k pages.

① No. of pages in sec m/m
② " " frames in main m/m
③ Show the addr format.
④ Page table size

Soln:- SM = 28 bit logical addr. = 2^{28} cells

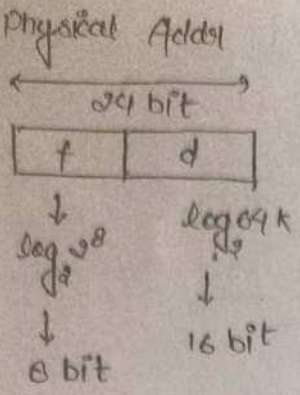
Pages = 256 MB

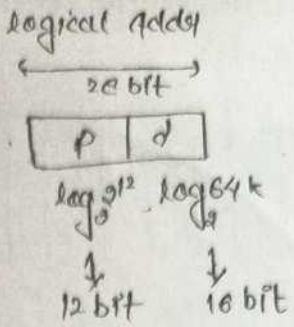
pages in SM = $\frac{256M}{64k} = \frac{2^{28}}{2^{16}} = 2^{12}$

Main m/m size = 2^{24} cell = 16 MB

frames = $\frac{MM\ size}{Page\ size} = \frac{16M}{64k} = \frac{2^{24}}{2^{16}} = 2^8$

(c)





(d) Page Table Size = # Entry in table * Entry size

Entry in table → # pages in SM

Entry size → Frame NO.

$= 2^{12} * 8$ bit

$= 2^{12} \times 2^3$ bits

$= 2^{15}$ bits

$= 2^5 \times 2^{10}$ bit

$= 32$ K bits.

Ques :- In a virtual m/m s/m design page fault rate is 85%. Page fault service time is 1000 ns. MM accessing time is 100 ns. What is the effective m/m access time.

$EMT = (p * s) + (1-p) Tm$

$= (0.85 \times 1000) + (1 - 0.85)\ 100\ ns$

$= 865$ ns.

** Ques :- Consider 4 frames main m/m initially empty with the following page references 4, 5, 7, 12, 4, 5, 13, 4, 5, 7. Identify the hit ratio and the no. of page faults in the m/m s/m using -

(1) FIFO (2) LRO (3) Optimal Replacement.

(1) FIFO :-

4	13	4 — M
5	4	5 — M
7	5	7 — M
12	7	12 — M
		4 — Hit
		5 — Hit
		13 — M
		4 — M
		5 — M, 7 — M

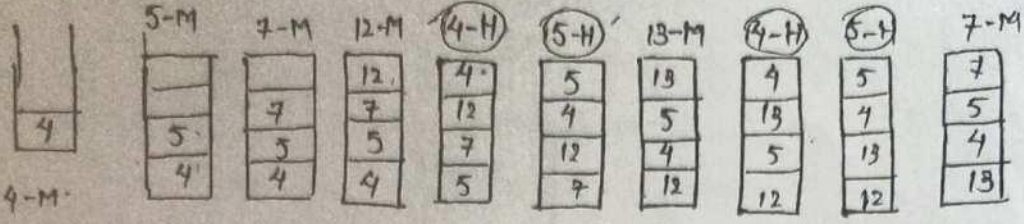
Hratio = # Hits / Total # access

$= \frac{2}{10} = 0.2$

Page faults = 8

② LRU {Always place the recent entries in the top}

4, 5, 7, 12, 4, 5, 13, 4, 5, 7

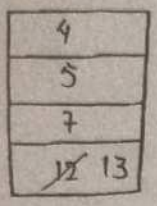


$H = \frac{4}{10}$

$H = 0.4$

page faults = 6

③ Optimal : [sequence]



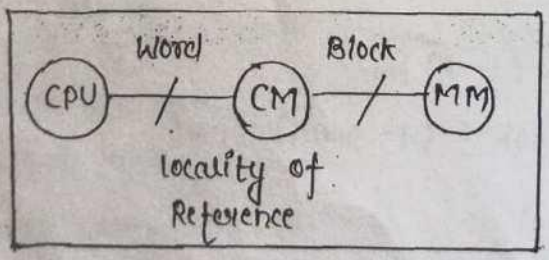
4 - m
5 - m
7 - m
12 - m
4 - H
5 - H
13 - M
4 - H
5 - H
7 - H

$H = \frac{5}{10} = 0.5$

page faults = 5

Cache Memory :→

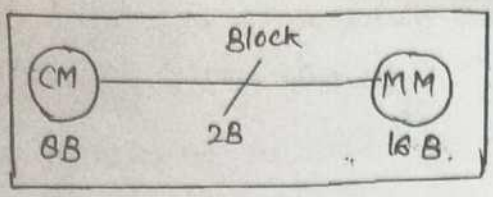
Cache m/m is an intermediate m/m b/w the CPU and main m/m, used to hold the image of a main m/m data so CPU will be accessing the main m/m data from the cache m/m within a less amount of the time. ∴ speed gap is synchronised b/w CPU and main m/m.



Acc. to a m/m hierarchy design, data will be transfered from MM to CM in a form of blocks so both of the m/m's are organised into a blocks based on the block size.

$$\#\text{ block in MM} = \frac{\text{MM Size}}{\text{Block Size}}$$

$$\#\text{ blocks (lines) in CM} = \frac{\text{CM Size}}{\text{Block size}}$$



CM Design

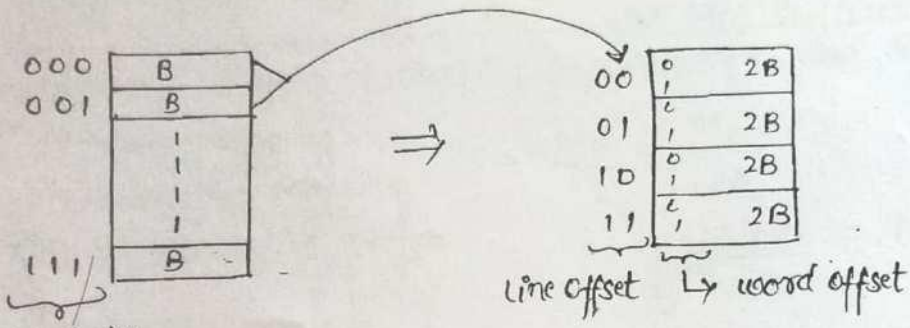
8B CM with 8Blocks

Before organization

8B → CM

After organization

lines (N) = $\frac{8}{2}$ = 4



MM Design

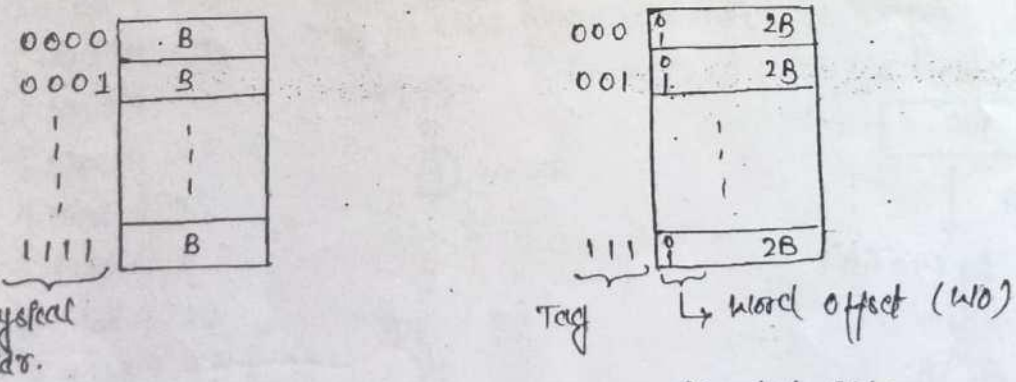
- 16B MM with 2B Block

Before Orgz^n

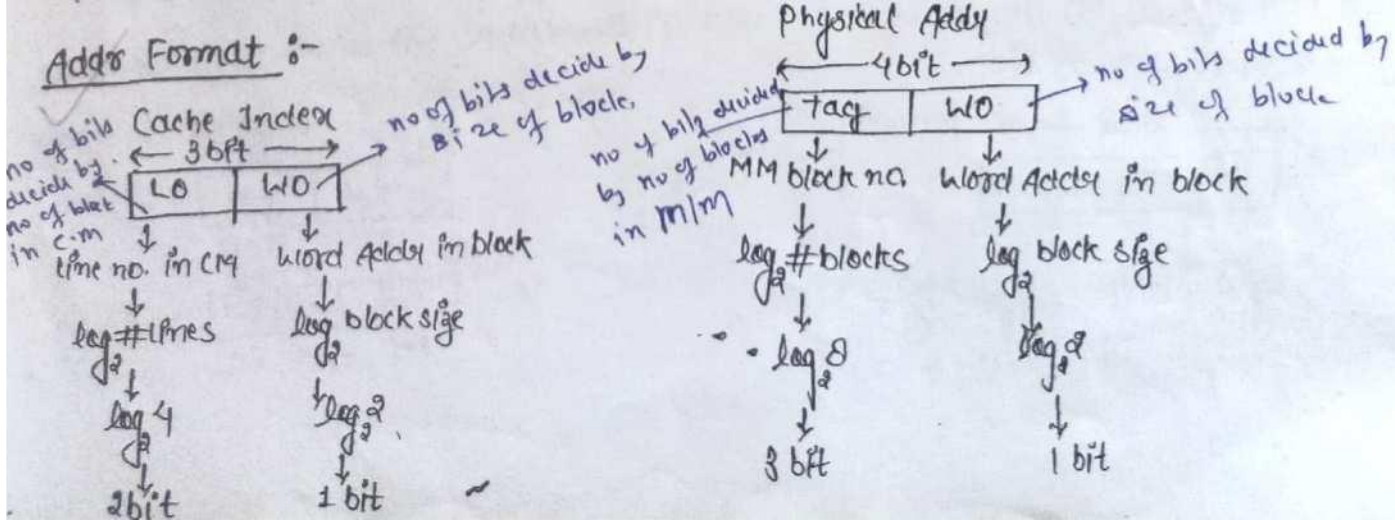
16B → MM

After Orgz^n

Blocks = $\frac{16}{2}$ = 8



Addr Format :-



Ques:→ Consider a comp s/m with a 32 kB cache organized into a 64 B blocks. CM data is a subset of 2^{28} Bytes m/m space. Calculate following

① No. of blocks in MM.
② No. of lines in CM
③ show the addr formats.

Soln:→ MM capacity = 2^{28} Bytes
CM " = 32 kB
Block Size = 64 B

a.) # Block in MM = $\frac{\text{MM Size}}{\text{Block Size}}$

$= \frac{2^{28}}{64} = \frac{2^{28}}{2^6} = 2^{22}$

b.) # lines in CM = $\frac{\text{CM Size}}{\text{Block Size}}$

$= \frac{32k}{64} = \frac{2^{15}}{2^6}$

$= 2^9$

c.) Cache Size = 32 kB
↓
32 k x B
↓
$\log_2 2^{15}$ = 15 bit addy

15 bit
← Cache Index →

LO	WO

$\log_2 2^9 = 9$ bit ↓ $\log_2 64 = 6$ bit

MM Size = 2^{28} B
Addr Size = 28 bit

← 28 bit → phy addr.

tag	WO

↓ ↓

$\log_2 2^{22} = 22$ bit $\log_2 64 = \log_2 2^6 = 6$ bit

Mapping Process :→

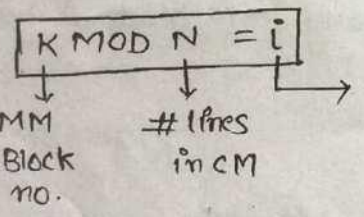
Process of a transferring the data from main m/m to CM is called as Mapping. During mapping, data block will be transfer to cache along with a complete tag so extra space is reserved in the cache design used to hold the MM tag information called as Tag Directory.

Tag Directory (Tag m/m) Size = # lines in CM * # tag bits in the line

In the cache design 3 types of mapping functions are used to map the data namely as (i) Direct Mapping
(ii) Associative Mapping
(iii) Set Associative Mapping.

1.) Direct Mapping :-

In this technique MOD Function is used to map the data.

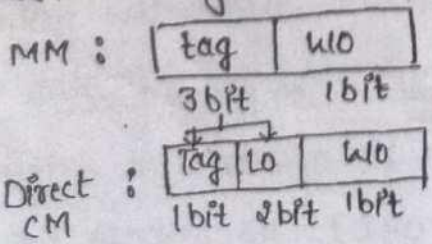


0 Mod 4 = 0
1 Mod 4 = 1
2 Mod 4 = 2
3 Mod 4 = 3
4 Mod 4 = 0
5 Mod 4 = 1
6 Mod 4 = 2
7 Mod 4 = 3

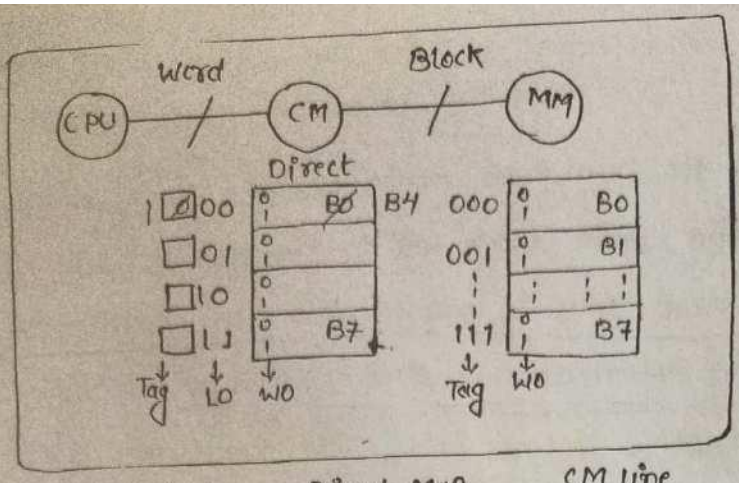
Mod function gives remainder

4)7(1
4
(3) ← rem

Above function shows the relationship b/w MM block and CM line so address binding is



Tag Directory Size = # lines in CM * # tag bit in the line
= 4 * 1bit
= 4 bits



MM Block	Direct-Map	CM Line
B0[000]	K MOD N = i 0 MOD 4 = 0	line 0
B7[111]	7 MOD 4 = 3	line 3
B4[100]	4 MOD 4 = 0	line 0

Note :- In direct cache design, explicit replacement policies are not required to replace the data because every MM block is having fixed location in CM.

Accessing Sequence :- Machine Instr :-

LDA [1001]

↓

CPU generates m/m Request

1001	$\overline{RD}$

↓

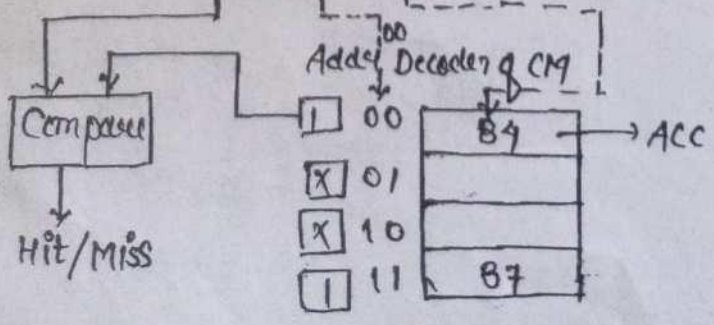
Direct CM

↓

Addr Format

tag	LO	WO
1bit	2bit	1bit

1	00	1

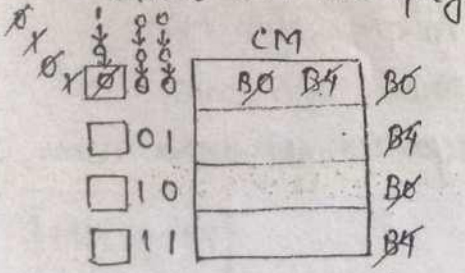


- CPU generated m/m request is initially referred in cache m/m.
- Direct cache controller interprets the request as | tag | LO | WO | format.
- Based on the line offset, the respective line is enabled in the cache m/m.
- Existing tag in the enabled line is compared with a CPU generated tag. When both matches oper^{n} become hit .so CPU will be accessing the data from CM based on word offset otherwise oper^{n} become Miss .so the respective data block is transfered from MM to cache using mapping function.

Note :- Limitation is every cache line is capable to hold only one block (tag) at a time. ∴ No.of conflict Misses will be increases.

- When the CPU frequently refer the multiple blocks which are mapped into a same cache line then the cache blocks are continuously swapping so hit ratio will be falling down. This phenomenon is called as Thrashing. described below. (Theory work)

- MM Block reference : B0, B4, B0, B4, B0 - - - -
- Direct CM is empty



B0 — M ; 0 Mod 4 = 0
B4 — M ; 4 Mod 4 = 0
B0 — M ; 0 Mod 4 = 0
B4 — M ; 4 Mod 4 = 0

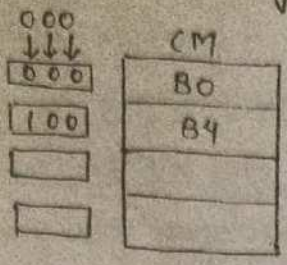
H = 0

{Thrashing}

Note :→ To handle the thrashing problem alternative cache design is required

1.) Associative Cache : This Cache is designed without addr called as content addressable m/m. In this cache sequence function is used to map

Limitation in this cache is Tag directory size is increases and search time is included in the avg access time.



BO - M ; sequence (Anytime)
B4 - M
BO - Hit
B4 - Hit
BO - Hit

2) Set Associative Cache :-

- This cache to compromise the disadv. in the direct and associative cache designs.
- In this orgz^n, lines are grouped into sets to accomodate more than one block in the set at a time.
- No. of sets (S) = $\frac{N}{P\text{-way}}$

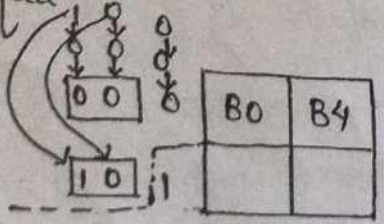
N = # lines in CM
P = # lines in the set

This cache used the Mod function to map the data i.e.

K Mod S = i

K = Main m/m block no.
S = # sets in CM
i = CM set no.

- This cache uses the FIFO or LRU policies to replace the data when the set is full.



BO - M ; 0 Mod 2 = 0
B4 - M ; 4 Mod 2 = 0
BO - H
B4 - H
BO - H

2 way set associative cache :

sets (S) = $\frac{4}{2}$ = 2

Writing Policies :→ When CPU performs write operation then only CM block is updated. The corresponding block is not updated in MM so same addr contain diff values at diff places. This kind of data inconsistency

problem in the m/m is called as Coherence.

To handle the above prob, updating techniques are used in m/m design.

These are 2 types.

① Write Through

② Write Back

- In a Write Through protocol, CPU performs the simultaneous write operation in both CM & MM so no coherence.
- In a Write Back protocol, CPU write back the CM block into MM before replacement when the update width is set so updations are copy back into MM before replacement hence data is safe.

Ques:→ Consider a 32 KB direct mapped cache organised into a 32 Byte blocks. CPU generates 32 bit physical addr to access the data. Show the addr binding.

Soln:→ Direct Map

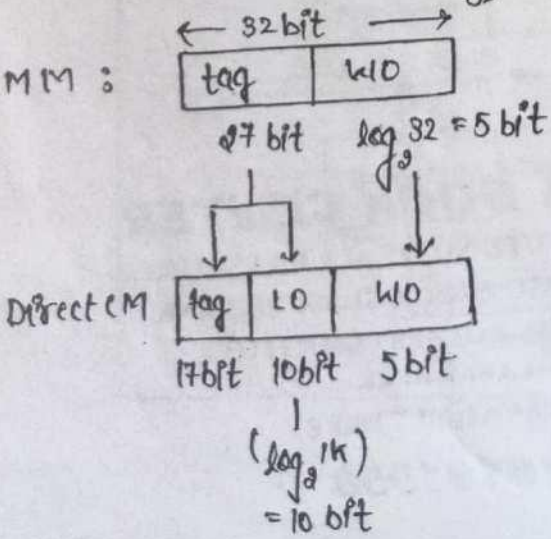
CM Size = 32 KB

Block Size = 32 Bytes

Addr size = 32 bit

{MM Addr}

$$\#\text{ lines} = \frac{32k}{32} = 1K$$



{Tag Directory Size} = # lines in CM * # tag bits in 1 line

= 1K * 17 bit

= 17 K bits

① Disk capacity = $(32 \times 2 \times 128 \times 256 \times 512 \times 8)$ B

$= 64 \times 128 \times 256 \times 512$

$= 2^6 \times 2^7 \times 2^8 \times 2^9$

$= 2^{30}$ B

$= 1$ GiB.

② a) Seek Time = 8.4 ms

Transfer Time :

b) Rotational Latency : depends on Rotational speed

9800 revolution — 1 min (60 sec)

1 revolution $= \frac{60}{9800}$ sec $= 6.12$ msec

∴ Avg Rotational Latency

$= \frac{1}{2}$ revolution

$= \frac{1}{2} * 6.12$ ms $= 3.06$ ms.

c) Transfer Time : depends on rotational speed

1 revolution time — 1 track (256 sectors)

? time — 1 file (32 sectors)

$\begin{cases} \frac{16KB}{512B} = \frac{2^{14}}{2^9} = 2^5 = 32 \end{cases}$

1 sector — 512 B

? # sector — file (16 kB)

$\Rightarrow \frac{16K}{512} \Rightarrow \frac{2^{14}}{2^9} = 32$

$\Rightarrow \frac{32 * 6.12}{256}$ ms $= 0.765$ ms

∴ $T_{avg} = (8.4 + 3.06 + 0.765 + 0)$ ms

$= 12.22$ ms.

③ Random sectors Accessing requires adjustment for every sector

1 revolution time — 1 track {256 sectors}

? time — 1 sector

$\Rightarrow \frac{6.12 \text{ ms} * 1}{256} = 0.02$ ms.

$T_{avg} = \underbrace{(8.4 + 3.06 + 0.02 + 0)}_{\text{1 sector Access time}}$ ms $* 100 = 1148$ ms

Accessing Sequence :-

- CPU initialises the IO interface along with IO command. Later busy with other useful task.
- IO interface control logic interprets the IO command and enables the oper^{n}. Based on the speed of an IO device consume the time to prepare the data later transfer to a interface buffer.
- When data is in the buffer then IO interface generate interrupt s/g to CPU and waiting for acknowledgment s/g.
- After receiving ACK s/g interface buffer content will be transfer to a CPU with high speed

- Diff IO interface chips used in the comp design is -
  - 8255 PPI (Prog Peripheral Interface)
  - 8251 USART
  - 8259A Int Controller
  - 8237/57 DMA

File System :->

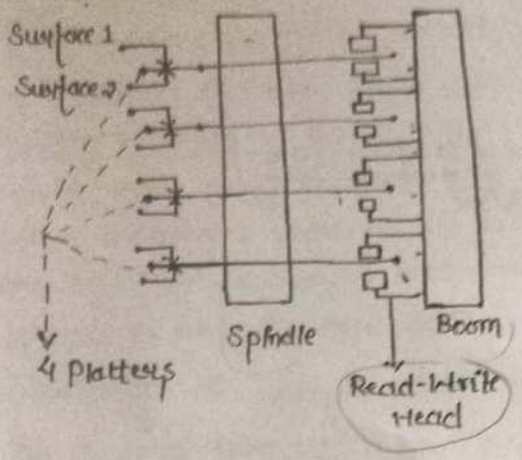
File is a collection of logically created entities. In the personal comp data base is organised in a form of file s/m. All the files are stored in the sec. storage component i.e. Hard Disc.

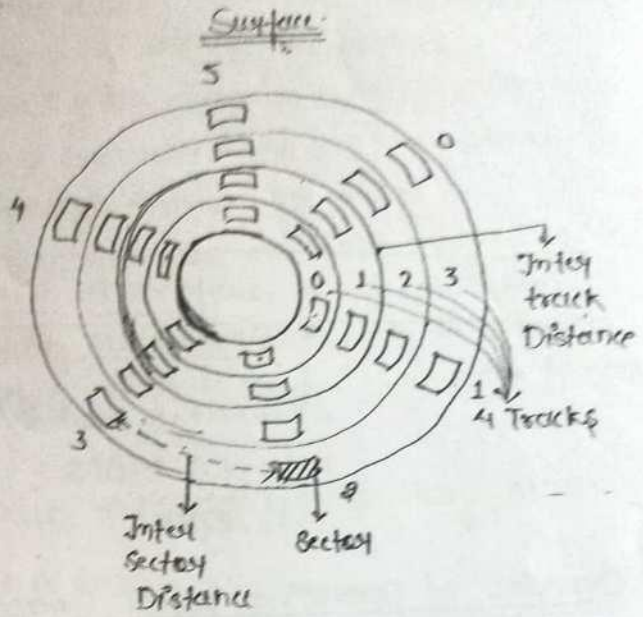
Hard Disk Structures :-

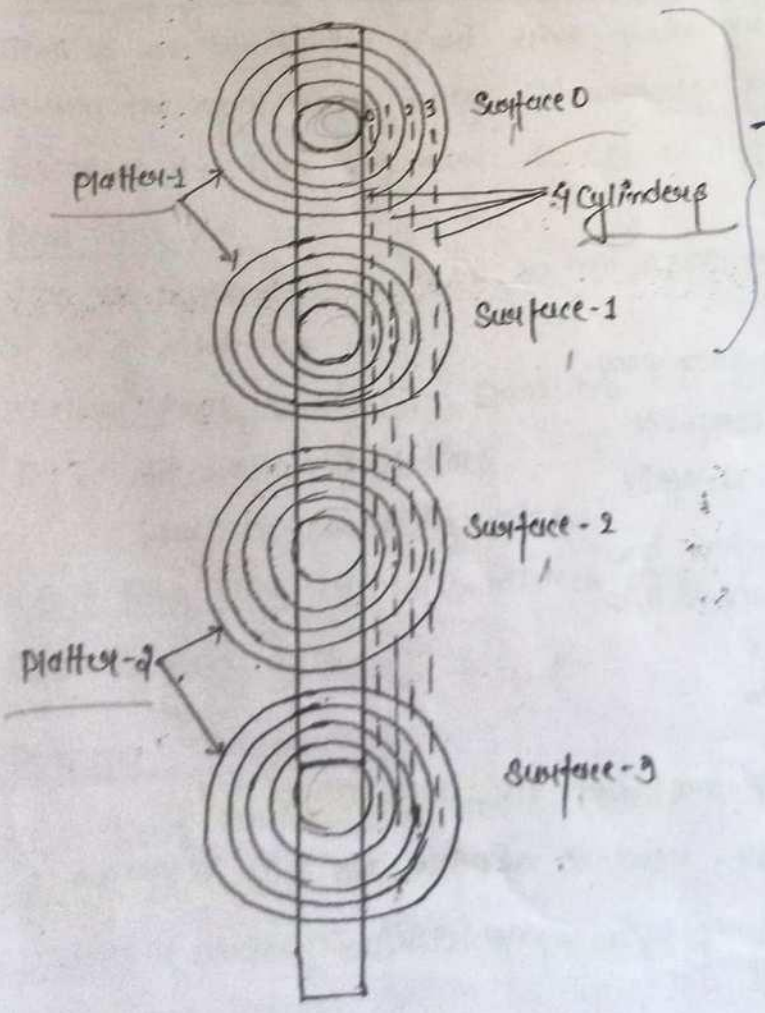
- Hard Disk is a direct access electro magnetic storage component.
- Hard disk contain a bunch of magnetic coated platters used to store data.
- Each platter contain 2 surfaces.
- Each surface contain set of concentric circles called as Tracks.

- In the Hard Disk's structure, cylinder is formed by connecting the same track no. in all the surfaces.

Hard Disk with 4 platters and 8 surfaces







→ yah ek hi disc ka hai jis ki Two Surfaces

Hard Disc with 2 Platters and 4 surfaces.

④ Disk Transfer Rate :-

Surface Rate =>

1 revolution time — 1 track data (256 * 512 B)

1 sec — ?

$$\Rightarrow \frac{256 * 512\ B}{6.12\ ms}$$

$$\Rightarrow \frac{256 * 512}{6.12} * 10^3 \text{ Bytes/sec}$$

∴ Disk Rate = # surfaces in disk * surface rate

$$= 64 * \frac{256 * 512}{6.12} * 10^3 \text{ B/sec}$$

$$= 1370607.5 \times 10^3 \text{ B/sec}$$

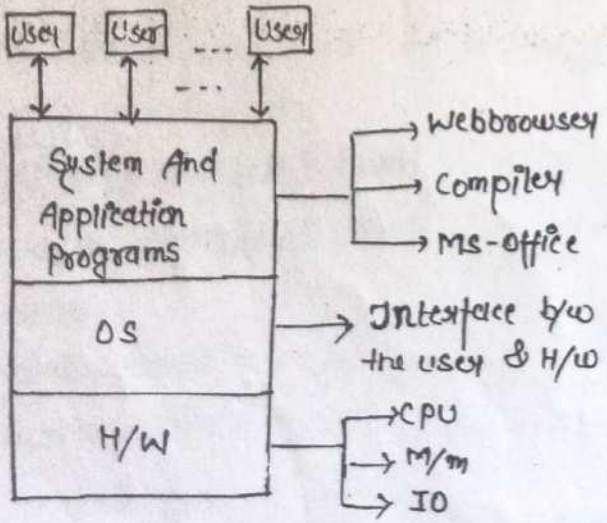
$$= 1370.6 \text{ MBPS}$$

$$= 1.37 \text{ GBPS}$$

Overview of Operating System Concepts :-

Comp. s/m contain 4 layers of an abstraction -

1) Hardware.
2) O.S.
3) S/m & Application Prog.
4) Users



Defn :- Operating s/m is an interface b/w the user and h/w. OS acts as a resource allocator, used to allocate the h/w resources to a user application prog (CPU time, m/m space and IO space)

- OS controlls and coordinates the efficient use of a H/w resources among the multiple user application prog.

Batch OS :-
- This OS allocates the main m/m progs to the CPU in sequence.
- When prog-1 requires the IO job during its execution then OS allocates the IO space to prog1 so CPU will be idle untill IO job is completed.
- Limitation in this OS is CPU idleness is more.

Multiprogramming OS :- • This OS also allocates the CPU time to a main m/m prog in a sequence.
- When the prog-1 require IO job during the execution then OS allocates the IO space to prog-1 and immediately schedule the the next prog to CPU ∴ CPU is not in idle state.
- Limitation is single user may keep the CPU into a busy state for a long time ∴ users idleness is more.

Multitasking OS (Time shared OS) :-
This OS maintains the fixed time quantum to share the resources among the application prog. It uses the efficient context switching based on the time quantum. So CPU and user idleness is eliminated.

Real Time OS :-
This OS is used in the s/m centric applications to satisfy the deadlines
It is of 2 kinds —
1) Hard Real time OS :- Deadlines are critical.
Eg :- Aircraft Simulation
Weather forecasting s/m etc.
2) Soft Real Time OS :- Deadlines are nominal.
Eg :- Banking s/m.

Process Management :-
* Defn :- Prog under execution is called as process. All the processes are stored in the main m/m.
* Attributes :- Set of attributes are assign to a process during its execution.
They are - ~~1) Process id~~

1.) Process Id (Pid)
2.) Process State (PS)
3.) Prog counter (PC)
4.) Priority
5.) General Purpose Reg (GPR)
6.) List of open Files (LOF)
7.) List of open devices (LOD)

- Process attributes list is called as process context.
- Process context is stored in the process control block (PCB)
- Every process has its own PCB.

Ex:- Process '0' contain PCB0

Pid	PS
PC	Priority
GPR	LOF
LOD	-

Process states :-

① New state :- Process Under creation

② Ready state :- Process is in Main m/m space.

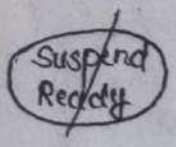
③ Running state :- Process is in CPU space.

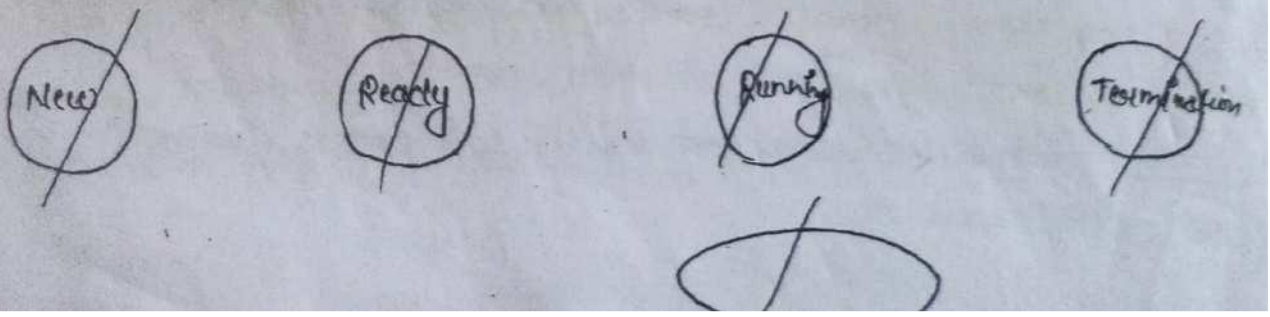
④ Wait / Block state :- Process is in IO space.

⑤ Suspend Ready state :- Suspend the process into this state later resume to ready state when MM is full.

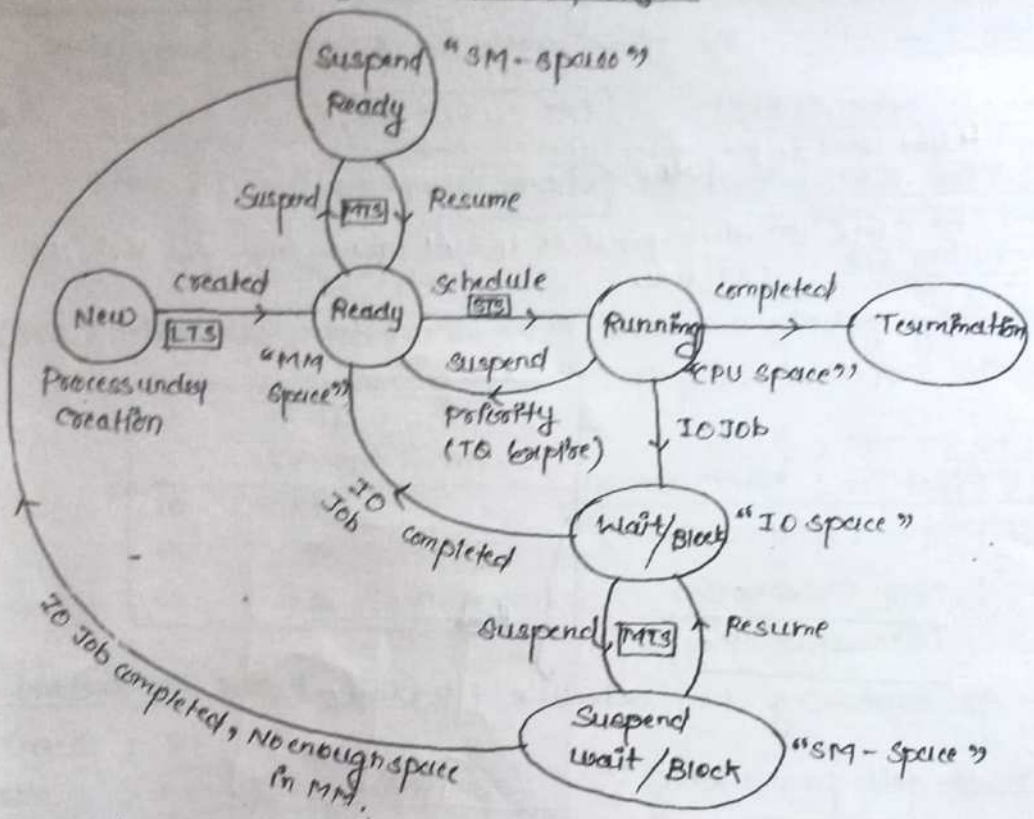
⑥ Suspend Wait (or) Suspend Block state :- Suspends the process into this state later resume to wait state when IO is BUSY.

⑦ Termination :- Process Completed.





Process Life Cycle



TQ :- Time Quantum

OS contain 3 kinds of schedulars used to manage the process life cycle,

③ Completion Time (CT) :- The time when the process is completed.

④ Turn Around Time (TAT) :- $TAT = CT - AT$
↳ Time spent by process inside que.

⑤ Waiting Time (WT) :- $WT = TAT - BT$
↳ Time in which procs is inside que and did not get cpu time.

CPU scheduling :→

Diff scheduling policies are used in the short term scheduler (STS), to minimise the TAT and WT of the processes.

They are -

1) FCFS (First Come First Serve)

Criteria : AT (Arrival Time)
Mode : Non Preemption
[One after other]

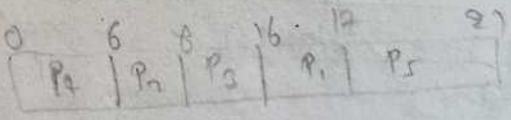
Pemb.

Note :- If AT matching then schedule the lowest Pid Process.

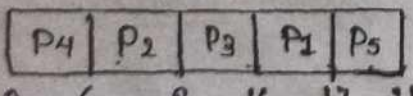
Ques: consider the foll. process table :-

Pid	AT	BT
1	3	1
2	1	2
3	2	8
4	0	6
5	3	4

What is avg waiting time using FCFS?



Soln :- Gantt chart [chart always starts from "0"]



	CT	TAT (CT - AT)	WT TAT - BT
P1	17	14	13
P2	8	7	5
P3	16	14	6
P4	6	6	0
P5	21	18	14

$\therefore$ Avg WT $= \frac{13+5+6+0+14}{5}$

$=$

$\text{Avg TAT} = \frac{\text{Total TAT}}{\text{no of procs}}$

Ques:→

Pid	BT
1	20
2	30
3	10

What is avg WT using FCFS ?

Soln:→ When AT not given then assume all are arrived at same AT = 0

∴

P1	P2	P3

0 20 50 60

CT	TAT CT-AT	WT TAT-BT
20	20	0
50	50	20
60	60	50

∴ Avg WT = $\frac{0+20+50}{3}$

$= \frac{70}{3}$ ms

2.) Shortest Job First (SJF) :-

Criteria : BT
Mode : Non-Preemption
[One after other]
} Remb.

Note : If BT matching then schedule lowest AT process.

Ques:→

Pid	AT	BT
1	2	8
2	1	4
3	4	8
4	1	12
5	3	4

What is avg WT using SJF

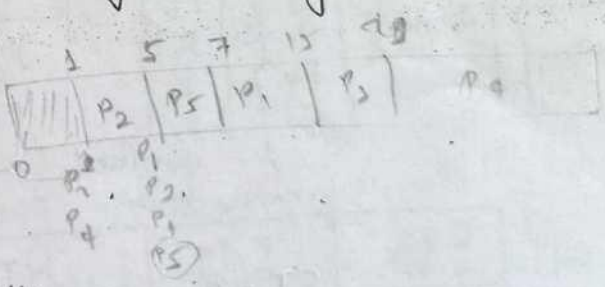
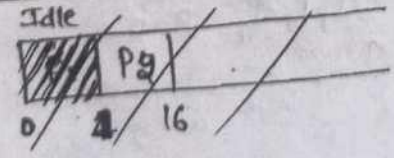
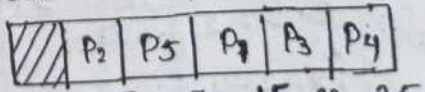


Chart [starts with 0]



Idle



0 1 5 7 15 23 35

↓ ↓
P2 All
P4 Arrived
Arrived

	CT	TAT	WT
P1:	15	13	5
P2:	5	4	0
P3:	23	21	13
P4:	35	34	22
P5:	7	4	3

3) SRTF (Shortest Remaining Time First) :-

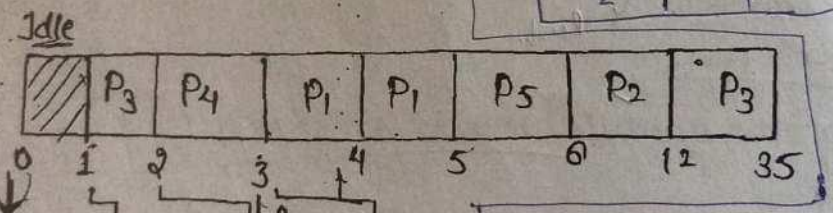
Prob { Criteria : BT
Mode : Preemption
Time Quantum : 1 → Always.

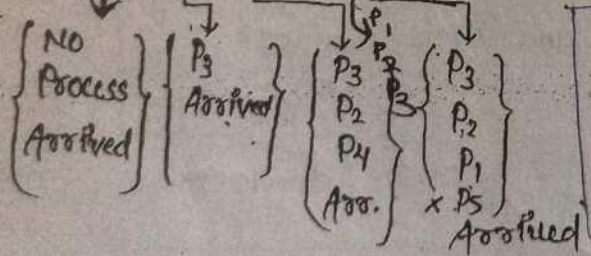
Note :- If BT matches then schedule lowest AT process.

Ques :- what is avg WT using SRTF.

Pid	AT	BT
1	3	~~2~~ 1
2	2	4
3	1	~~2~~ 23
4	2	~~1~~ Kill
5	4	3

Chart {starts with 0}





	CT	TAT = CT − AT	WT = TAT − BT
P1	5	5 − 3 = 2	0
P2	12	12 − 2 = 10	6
P3	35	35 − 1 = 34	10
P4	3	~~3~~ 4 − ~~2~~ 2 = 1	0
P5	8	8 − 4 = 4	1

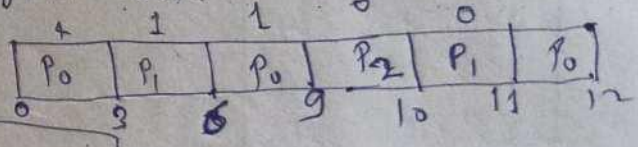
④ Round Robin:

Time quantum = 3
mode : pre-emption
~~Criteria : BT~~
Criteria → Time quantum, AT, BT

Process	AT	BT
P_0	0	7
P_1	2	4
P_2	4	1

quantum = 3 sec (every process will execute for 3 sec)

gantt chart:



Queue :-

P_0 P_1 P_0 P_2 P_1 P_0
0 2 3 4 6 9

Networking Principles :-

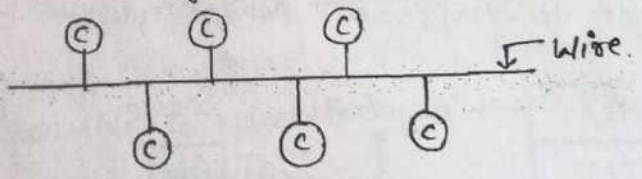
Interconnection among the n/ws used to share the information and the resources is called as comp n/w.

Types : Based on the distance, n/ws are categories into 5 types-

1) Personal Area Network (PAN) : System Wide.
   Eg :- CPU to Printer N/w.
2) Local Area Network (LAN): Campus Wide.
   Local Administrator required to manage the N/w.
   Higher Bandwidth.
3) Metropolitan Area N/w (MAN) : City Wide
4) Wide Area N/w (WAN) : Country Wide
   - Internet Service Providers required.
   - Lower Bandwidth.
5) Internet : Planet wide.

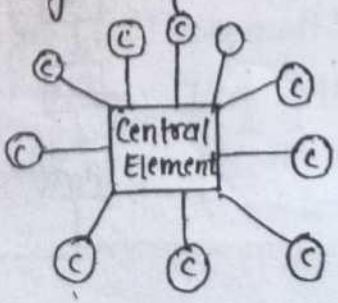
Topologies :-

1) Bus Topology :- In this structure all the components are connected to same cable (single wire).

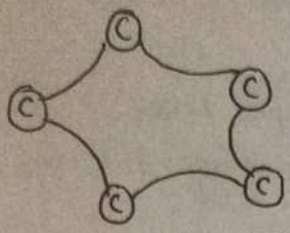


Adv : Less cabling
Disadv : If cable damaged then entire n/w is damaged.

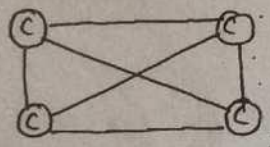
2) Star Topology :- In this structure all the components are connected to a central element. It requires more cabling. If any cable damaged then the corresponding component does not work so ~~exti~~ no effect on entire n/w.



3a) Ring Topology :- In this str. every component in the n/w is connected to adjacent components. If any cable is damaged then the comm^n is takes place in oppesite dir. It takes more permutations to find out fault.



4a) Mesh Topology :- In this str. every component is directly connected to an other components in the n/w. It is suitable to establish the small n/ws. It increases the throughput (workdone).
When the n/w contain n components then each component have n-1 edges.



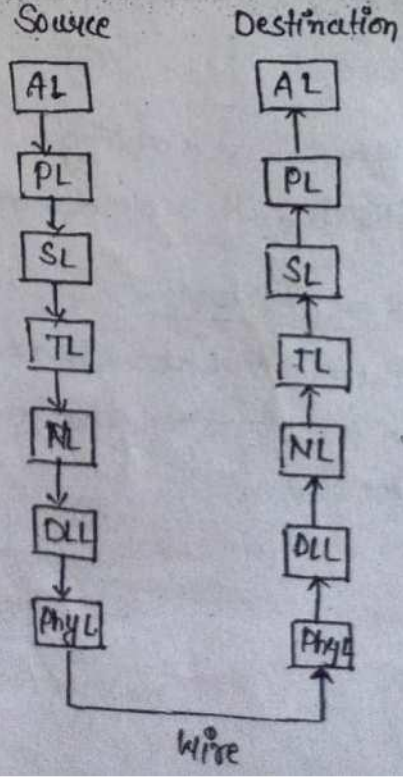
ISO-OSI Reference Model :→

It is a standard comm^n model, contain 7 layers of protocol stack used to provide s/m to s/m comm^n in the n/w.

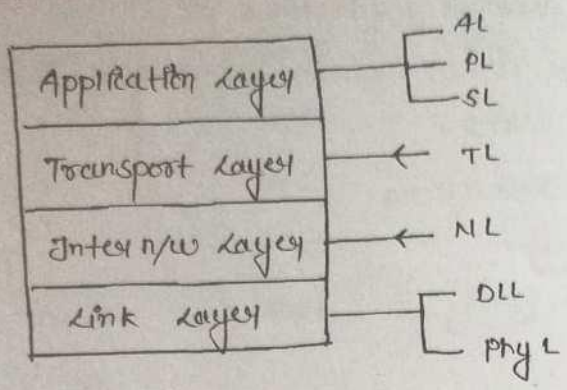
Application Layer (AL)
Presentation Layer (PL)
Session Layer (SL)
Transport Layer (TL)
Network Layer (NL)
Data Link Layer (DLL)
Physical Layer (Phy. L)

→ Layer-7

→ Layer-1



TCP/IP Model :-



TCP/IP : Transmission Control protocol / Internet Protocol

Functions :-

① AL :- This layer contain application protocols (http) used to send or receive the data to/from the n/w.

② PL :- This layer is responsible for data presentation, data conversion, data compression and data cryptography (encryption & decryption)

③ SL :- This layer is responsible to establish and manage the sessions b/w two parties in the n/w to provide ongoing comm^{n}.

④ TL :- This layer accepts the msg from SL and establish and divide the msg into small units, transferred into a n/w. This layer is responsible to establish the multiple n/w connections to increase the throughputs.

⑤ NL :- This layer organises the data unit into a packet and also provides the routing tables to transmit the data into a n/w.

⑥ DLL :- This layer is responsible to prepare error correction & error detection codes in the data. It organises the data packet into a bit frames, transmitted into a phy. layer.

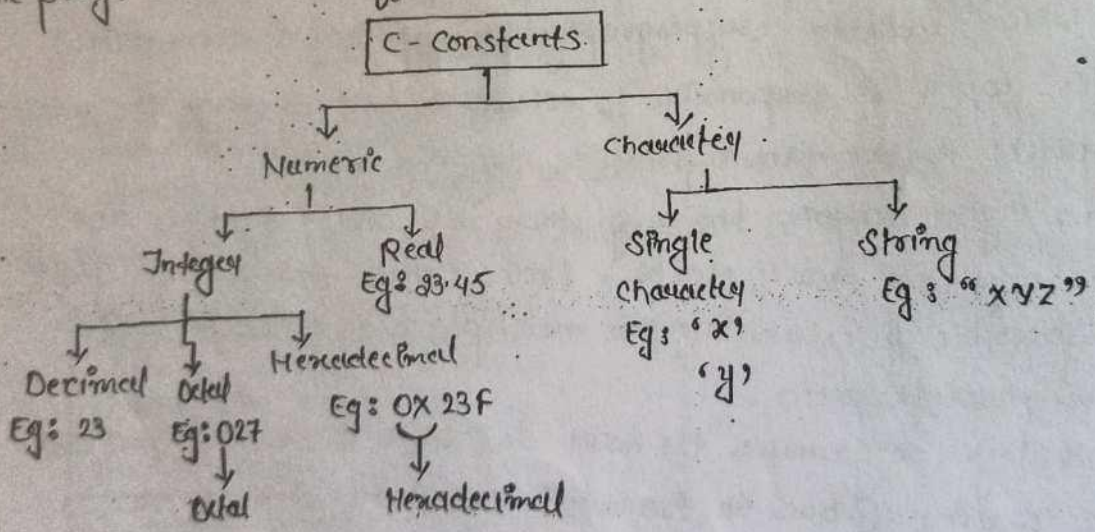
⑦ Phy L :- This layer is responsible to adjust sig levels based on the bit frames and transmitted into a physical media (coaxial cables or twisted pair cable or optical cable etc).

Program Elements :-

* Identifiers :- • Identifier is the name of a prog entity. which refers the name of variables, arrays, functions etc.
• Identifier is a user defined name which contain sequence of letters as digits with 1st character as a letter.
• Keywords are not permitted as Identifiers.
(Reserved Words)

Eg :- xyz ✓
XYZ ✓
a98 ✓
A98 ✓
98 — Invalid
for — Invalid

* Constant :- It is a fixed value which remains same throughout the prog. execution. Diff. C- consts are as follows -



* Variable :- It is the name of a m/m cell. It may accept diff values during the prog execution.

Variable is declared in the prog before using it.

Syntax :

data type variable name ;

Eg: int a, b ;
a = 20 ;
b = 30 ;

variable	Addr	cell
a	2000	20
b	3000	30

printf = pre formatted output value.

Variable	Address	Cell
a	2000	20
b	4000	30
p	6000	2000

{ Pointer Variable }

```
int a, b, *p, x;
a = 20;
b = 30;
p = &a;
x = *p;
```

O/p : x = 20

x = [[p]]
2000
20

Data Type :- It shows the range of possible values and also specifies set of operations performed on a value.

C- Data Types

- Primitive [Elementary primary]
- Non Primitive [Secondary composite]

a++ ; ++a

Post Increment

a
a+1

a = 4
b = a++

o/p: b = a ; b = 4
a = a+1 ; a = 5

Pre increment

a+1
a

a = 4
b = ++a

o/p: a = a+1 ; a = 5
b = a ; b = 5

Eg:→ a = 3
b = 4
x[++b] = a++ ;
LHS RHS

a: cell 3 b: cell 4 x[5]: cell 3

x[5] : 3

3 will be assigned to x[5]

Eg:→ n = 3 ;
a[++n] = n++ ;

n: cell 3 → 4 a[4]: cell 4

o/p : 4 is assigned to a[4] after that n = 5.

9866765629
Email : sagar262003@yahoo.co.in