sign - magnitude

drawback :- 2 representation for 0

→ 1 bit to represent sign if 2 no. are added.

ranges

sign - mag. & 1's comp } $-2^{+n-1}+1$ to $2^{n-1}-1$

convert to 2's ⇒ write its +ve sign-m form, if -ve take 2's comp. then MSB = 1

2's comp -2^{n-1} to $2^{n-1}-1$ (MSB = 1 ⇒ no is -ve

0	0	0	0	⊕ 0
0	0	0	1	+1
0	0	1	0	+2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7

-ve

1	0	0	0	-8 ⇒
1	0	0	1	-7 ⇒ -8+1
1	0	1	0	-6 -8+2
1	0	1	1	-5
1	1	0	0	-4
1	1	0	1	-3
1	1	1	0	-2
1	1	1	1	-1

Scanned by CamScanner

sign extension :

Overflow

6 + 7 ⇒

```
  0110 +
  0111
 ------
 (1)101
```

-ve not true ∴ overflow.

6 − 7 ⇒

```
  0110
  1001
 ------
  1111  ⇒ −1
```

− 6 − 7 →

```
  1010 +
  1001
 ------
(1)0011  ⇒ +ve 2 ; wrong
```

Multiplication (shift & add)

32 × 32

Multiplicant (64 bit) → shift left & add if 1 else shift left

Multiplier (32 bit) → shift right

product (64) → 32 + 32 = 64

ALU (64)

Eg: 2 × 3

Iteration	step	Mr multiplier	Mc multiplicant	P product
0		0011	00000010	00000000
1	P = P + Mc sll Mc Srl Mr.	0001	00000100	00000010
2	P = P + Mc Sll Mc	0000	00001000	00000110

Scanned by CamScanner

3rd Sll Mc 0000 00010000 00000110
Srl Mr

4th Sll Mc "
Srl Mr

Total iteration = 32.

max no. of steps = 32 × 3 = 96.

(2) To reduce size instead of multiplicant left shift we do product right shift

eg: 32×32 bit ⇒ multiplier = 32
multiplicant = 32
product = 64
ALU = 32

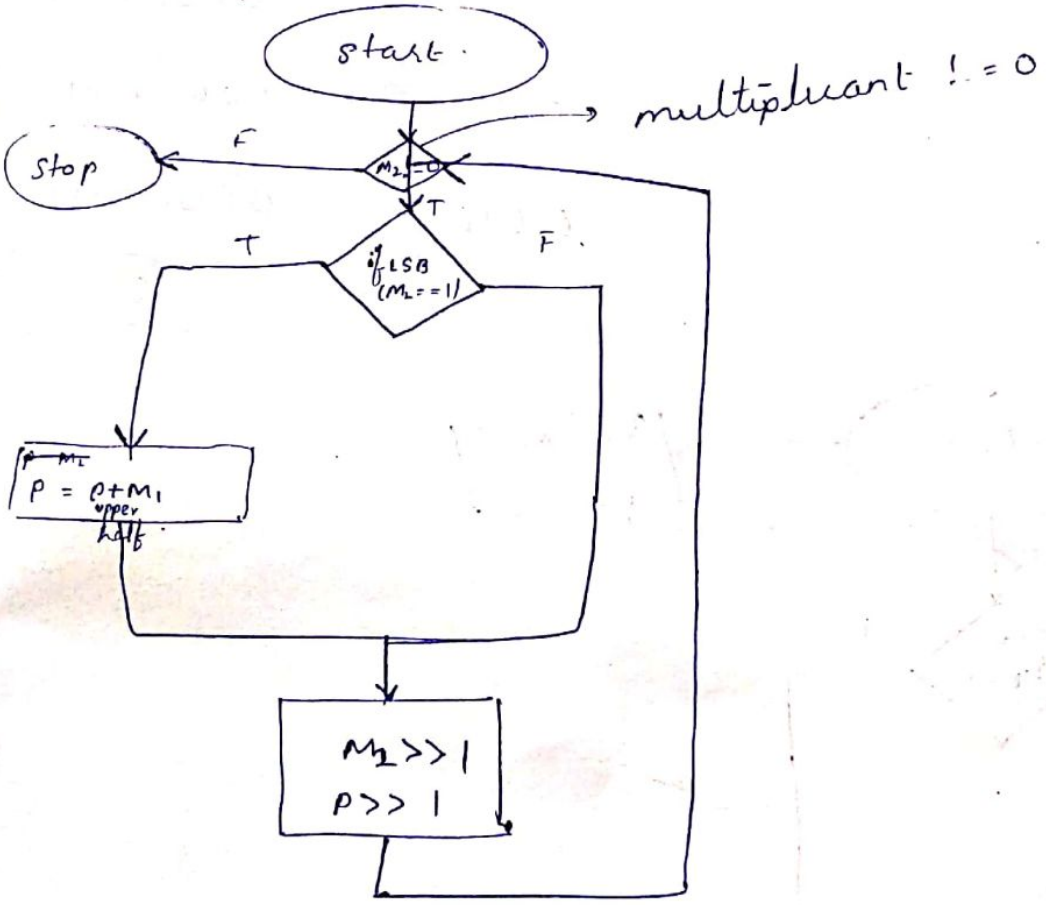
		(M2) multiplier	(M1) multiplicant	P Product
0	initial	0011	0010	0000 0011
1	P+M1	0001	0010	0010 0011
	M2>>1			⇒ 0001 0001
	P>>1			
2.	(P+M1) higher order bit	0000	0010	0011 0001
	M2>>1			0001 1000
	P>>1			

Scanned by CamScanner

3.	$M_L >> 1$	0000	0010	0000 1100
	$P >> 1$			
4)	$M_L >> 1$	0000	0010	0000 0110
	$P >> 1$			

6

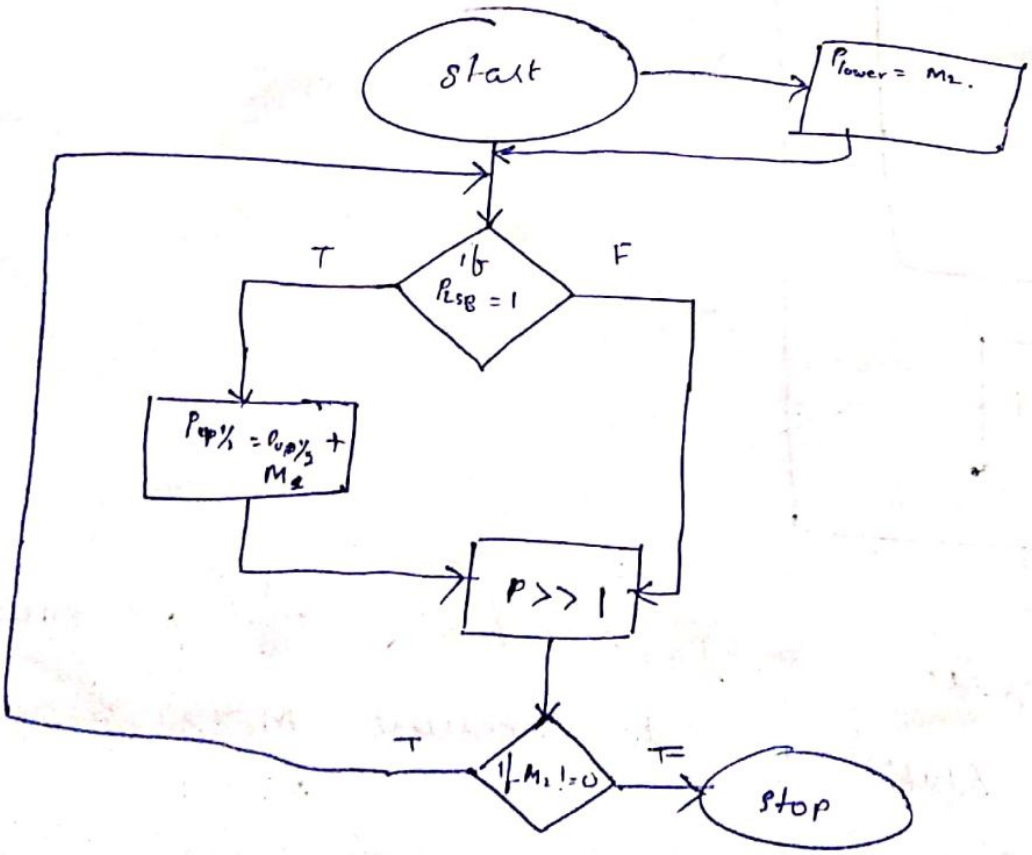
1) Draw its flow chart



(3) store multiplier in lower half of product & right shift P to combine $M_L >> 1$ & $P >> 1$

Scanned by CamScanner

iteration	step	Multiplier	multiplicand (M_1)	product
0	initial	0010	00000011	
1	$P_{LSB} = 1$ ∴			
	$P_{up} = P_{up} + M_1$	0010	0010 0011	
	P >> 1		P>>1 M_2>>1	
			0001 0001	
2	$P_{LSB} = 1$ ∴			
	$P_{up\frac{1}{2}} = P_{up\frac{1}{2}} + M_1$	0010	0011 0001 >>	
			0001 1000	
3.	P >> 1	0010	00001100	
4	P >> 1	0010	00000110	
			6	



Total no. of iteration = 32 x 2 = 64

Scanned by CamScanner

Signed bit multiplication

11.
20
14
18.75

2x - 3.

0010 x 1101 (M_1)

Iteration	Step	(M_1) multiplicand	Product
0	initial	0010	0000 1101
1	1 :	"	0010 1101 (1)
	$P_{up} = P_{up} + M_1$	"	0001 0110 (0)
	P >> 1	"	
2	P >> 1	"	0000 1011 (1)
3.	$P_{up} = P_{up} + M_1$	"	0010 1011
	P >> 1		0001 0101 (1)
4)	$P_{up} = P_{up} + M_1$	"	0011 0101
	P >> 1		0001 1010 (0) ans.
			1110 0110 2's

(-2) x 3 sign extend i shift P with 1 (IMP)

=> 0010
1110 x 00101 M_1

0.	initial	1110	0000 0010 (1)
1	$P_{up} = P_{up} + M_1$	"	1110 0011
	P >> 1	"	1111 0001 (1)
2.	$P_{up} = P_{up} + M_1$	"	1101 0001
	P >> 1	"	1110 1000

Scanned by CamScanner

(3) P >> 1 " 1111 1000 1 0100

(4) P >> 1 " 1111 0000 1010

ans = -6

1110 1010

M_1 M_2

$-2 \times -3.$

1110 1101 multiplicand M_1

0	initial	1110	0000 1101 (1) multiplier
1	$P_{up} = P_{up} + M_1$	1110	1110 1101
	P >> 1 (sign-extended)	"	1111 0110 (1)
2.	P >> 1	"	1111 1011 (0)
3	$P_{up} = P_{up} + M_1$	"	1101 1011
	P >> 1	"	1110 1101 (1)
4)	$P_{up} = P_{up} + M_1$	"	1100 1101
	P >> 1		1110 0110

-6 ans

Scanned by CamScanner

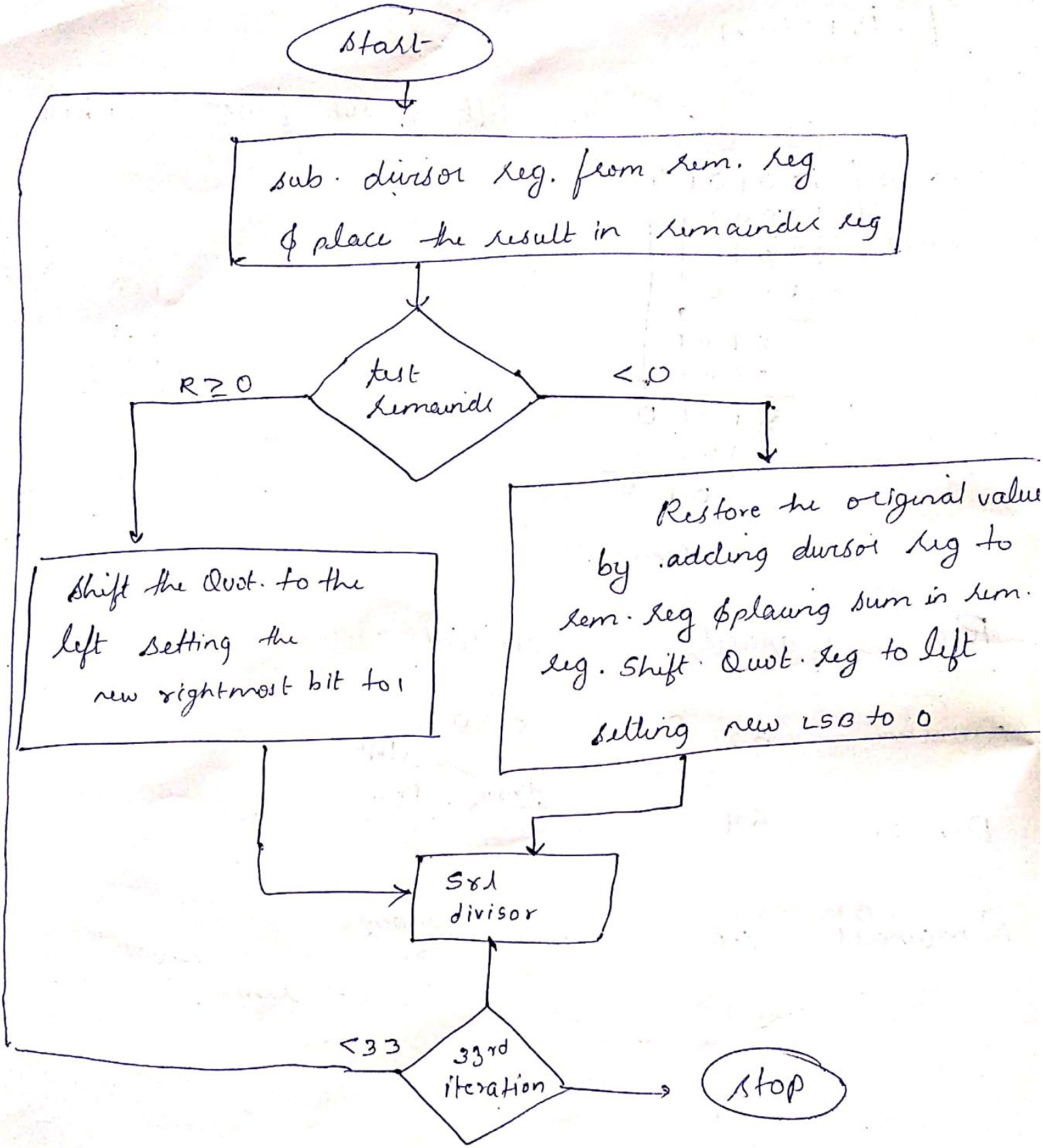
Division

shift left & sub from 1st 4 bit

```
           1001   <- quo
        ________
1000  / 1001010
divisor 1000
        ----
        0010
         0000
        ----
          0101
          0000
         ----
          1010
          1000
          ----
           010
          rem.
```

Reg.	Capacity	initial
Quotient	32	000...
Divisor	64	divisor (32) | 000 (32) — shift
Remainder	64	000-- (32) | dividend (32)

initial rem =



Scanned by CamScanner

10001 ÷ 0101

div. div.

Iteration	Step	div (0)	Rem. (R)
0	initial	0101 0000 (div, 2^00)	00010001
1	R = R - D		00010100
	R << 1	10100000	
2.	R = R - D.		00001111
	R << 1	01000011	
3	R = R - D.		00000010
	R << 1.	10000110	

7/2. If.

	Step	div	Quo.	Rem.
0	initial.	0010 0000	0000	00000111
1.	R - div		0000	11100111
	Q << 1			
	R + div.			
	DIV >> 1	0001 0000		00000111
2.	R - div		0000	11110111
	Q << 1			
	R + div			
	DIV >> 1	0000 1000		00000111
3.	R - div		0000	11111111
	Q << 1			
	R + div	00000100		00000111
	DIV >> 1			

Scanned by CamScanner

4. R - div 0000/0011

Q << 1 0001

DIV >> 1 0000/0010

(5) R - DIV 0000/0001 Rem

DIV >> 1 0000/0001

Q << 1 0011

5th iteration R - DIV ∴ stop 0000/0000

(4+1st)

[111 to 0010 [0000/0111]

Rem +ve
→ div = possible
Q << 1 set
Rem -ve
→ div = impossible
Q << 1 sets

Version 2

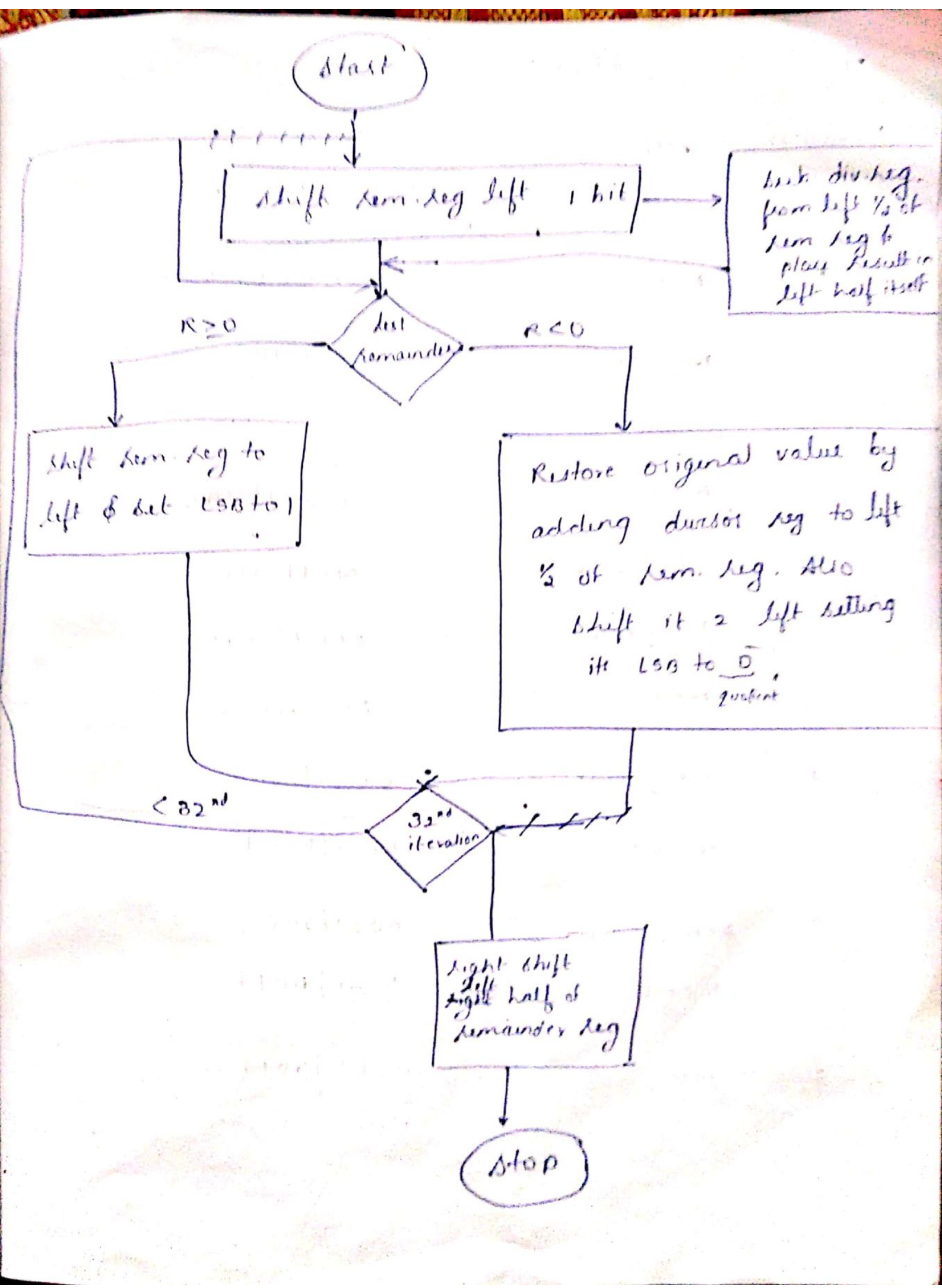
Instead of div >> 1 do rem << 1

Divisor 32 bit divisor

Rem 64 bit 0000 → dividend

ALU 32 bit

Scanned by CamScanner



Scanned by CamScanner

7/2

Iteration	Step.	div.	rem.
0	initial	0010	0000/0111 (rem 1/2)
1	R<<1		0000/1110
	rem 1/2 − div		1110/1110
	rem 1/2 + div		0000/1110 (00 10)
	R<<1		0001/1100
2.	rem 1/2 − div		0001/1100
	rem 1/2 + div		0001/1100
	R<<1		0011/1000
3.	rem 1/2 − div		0001/1000 (rem 1/2)
	R<<1		0011/0001
4.	rem 1/2 − div		0001/0001
	R<<1		0010/0011
5.	rem 1/2 >>1		0001/0011

rem quotient

Scanned by CamScanner

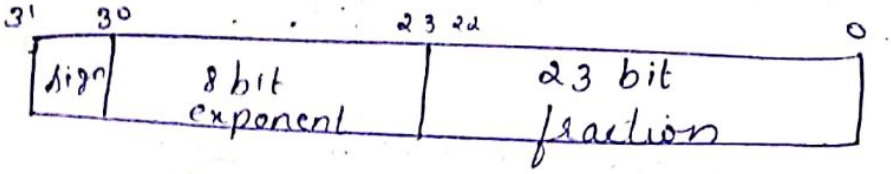
Scientific & normal form

eg normal { ① 0×10^{-9} } scientific

$\propto$ { 0.1×10^{-8} ; ⑩ 0.0×10^{-10} } $\propto$.

Representations

→ single precision



overflow : +ve exponent is too large

underflow : -ve " " "

To represent a no:

① convert to normal form

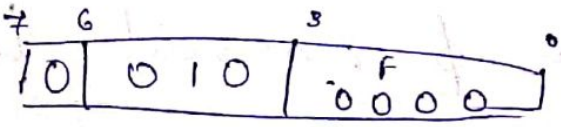
② add bias value to exponent $= 2^{n-1} - 1$

n = no. of bits

eg: ① Rep. $(0.1)_2 = 1.0 \times 2^{-1}$ in 8 bit ∴ exponent = 3 bit

② bias $= 2^{3-1} - 1 = 3$ ∴ $-1 + 3 = 2$

Scanned by CamScanner



0.0625 2^{-4}, 2 ..

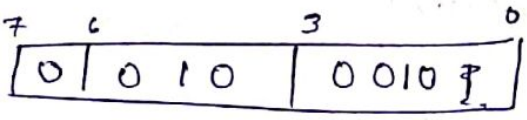
② 0.5625 $= 2^{-1} + 2^{-4}$ (8 bit)

$\frac{1}{8}$
0.0

$= 0.1 +$
$0.0001 =$
$0.1001 = 1.001 \times 2^{-1} \Longrightarrow$ normal form

bias $= 2^{3-1} - 1 = 3$

$\Rightarrow -1 + 3 = 2$



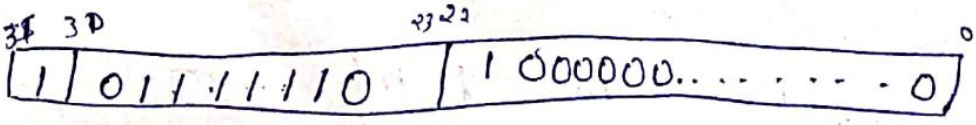
③ -0.75 single precision → 32 bit ⇒ n = 8

$= -0.1 +$
-0.01
$-0.11 = -1.1 \times 2^{-1}$

∴ n = 8

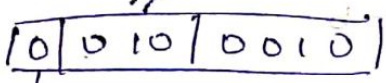
$b = 2^{n-1} - 1 = 127$

∴ ⇒ $-1 + 127 = 126$



Scanned by CamScanner

convert back



$b = 2^{3-1} - 1 = 3$ $\therefore e = 2 + -3 = -1$

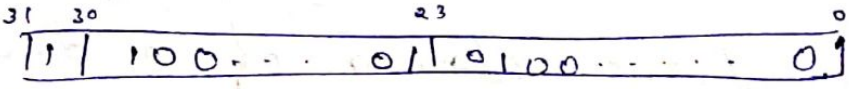
$1 \times (1.001) \times 2^{-1}$

$(1 + 0.125) \times 0.5$

$= 0.5625$

to store exp add bias

(2)



bias $= 2^{8-1} - 1 = 127$.

$\therefore exp = (27 + 1) - 127$.

$= 129 - 127 = 2$.

$\therefore -1.01 \times 2^{2} = -(1 + 0.25) \times 2^{2}$

$= -1.25 \times 2^{2}$.

$= -5$

single precision		double precision		obj
E	F	E	F	
0	0	0	0	0
0	non-zero	0	non-zero	± denorma
1-254	Any	1-2046	any	± float
255	0	2047	0	± ∞
255	Non zero	2047	non zero	Not a no.

$(-1)^{s}(1+F)^{1-b}$

Scanned by CamScanner

In denormalised form

value = $(-1)^s (a+f)^{1-b}$

Self study: floating point multiplication

⇒ Floating point addition

Assumption: We have storing of 4 decimal digits of significand & 2 digits of exponent in fraction

(1) Align the decimal point of number with smaller exponent so that it matches the larger one (After normalising & truncating)

eg: A = 9.999×10^1

B = 161.0×10^{-1} ⇒ B = $\underbrace{1.610}_{4} \times 10^1$

(2) addition of significand

$$\begin{array}{r} 9.999 + \\ 1.610 \\ \hline 11.609 \end{array} \times 10^1$$

(3) Normalisation

$\underbrace{1.1609}_{5 \text{ but } 4 \text{ allowed}} \times 10^2$

∴ round ⇒ 1.161×10^2

Scanned by CamScanner

(final result must be normalized)

0.1. 0.5 − 0.4375

$0.1 \times 2^{0} - 0.0111 \times 2^{0} \Rightarrow 1.000 \times 2^{-1} - 1.11 \times 2^{-2}$

1.0×2^{-1}

0.011×2^{0}

0.1 −
0.011
0.0010 × 2°
⇒ 1.000 × 2⁻³ = 0.0625 0.125

$1.000 \times 2^{-1} - 1.110 \times 2^{-2}$ = $1.000 \times 2^{-1} - 0.111 \times 2^{-1}$

$0.001 \times 2^{-1} = 0.0625$

$= 1.000 \times 2^{-4}$

Min value of exp in single precision = −126
+ bias = 127
1

Max value = 127 + 127 = 254 ✓

Parallelism & computer arithmetic

Associativity

Integers follow associativity but

floating point numbers not so

eg: $X = -1.5 \times 10^{38}$

$Y = 1.5 \times 10^{38}$

$Z = 1.0$

I] $X + (Y + Z) = X + 1.5 \times 10^{38}$

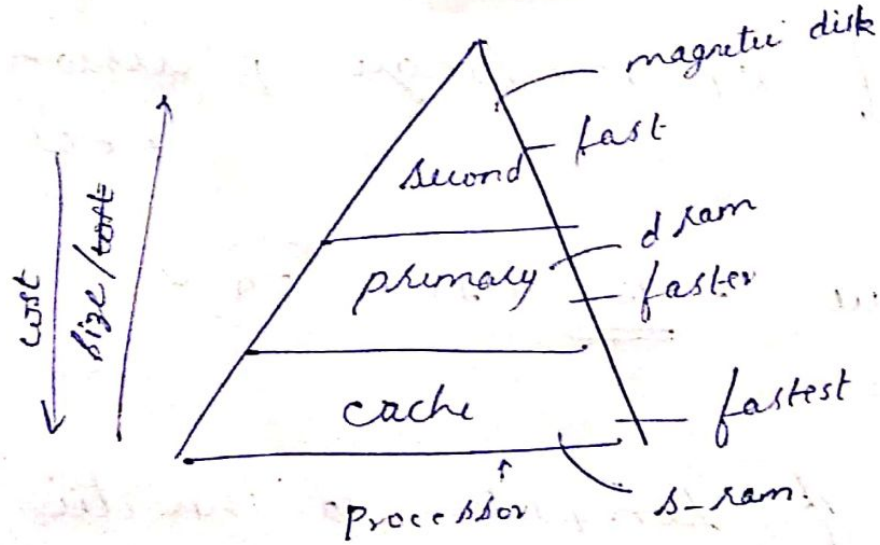
$= 0$

II] $(X + Y) + Z = 0 \times 10^{38} + Z$

$= 0 + 1$

$= 1$

Memory hierarchy



processor access data from cache

data copied from 2° → 1° → cache → proces

Scanned by CamScanner

Now, processor at once copies all the data item needed for a given piece of code rather than one at a time.

Now it is based on principle of locality

① temporal locality

⇒ If we access x, then there is higher probability of accessing x in near future (eg: loops) (∴ copy to cache)

② spacial locality

⇒ If we access x, then there is higher prob. of accessing data near to x in near future

eg: Arrays

(If we copy items / subset of data so that if we need a data we need to search only cache (smaller) than 1° or 2° (larger)

Scanned by CamScanner

Block / line : Minimum unit of info. that is either present or not present in mem.

Hit : A hit is occured if a data item is found in a given level else its a miss.

If a hit is occured it copies it to next level closer to processor

else

searches in next level above it

Hit ratio : % of memory access satisfied by that level

$$\text{i.e. } \frac{\text{no. of hits in a level}}{\text{no. of access in level}} \times 100$$

III^{ry} miss ratio = 1 − hit ratio

Hit - time : time taken to provide a data to CPU including the time to

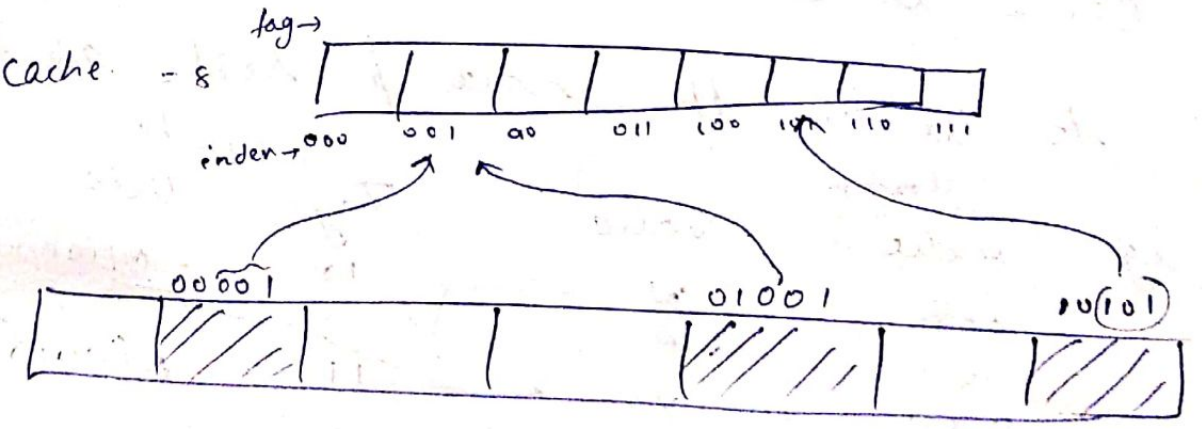
Scanned by CamScanner

check if its a miss/hit

misperiality : If its a miss time to copy lower to upper mem + time to deliver data to CPU

Now, in cache limited space is there so how do we copy all the elements needed for a program in there (duplication or overwrite may occur)

& how the processor can identify the data from cache?



i lower 3 bit as index & upper as tag here by using modulo on address %8

i if add data - 00001 we do modulo 8 to find index 001

Scanned by CamScanner

Now to know if its lower bit of 01001 or 00001 place upper bit in tag associated with that position in cache.

This is called Direct mapped cache

Now to see if its valid data or not we have a valid bit also associated with each cell.

If an address of a data is

$x_n x_{n-1} \ldots \underbrace{\ldots x_0}_{x}$ then if size of cache is 2^x, then lower x bits is used to divide the index & rest as tag.

eg:

index (lower bit)	Valid	Tag (higher bit)	Data
000	N	10	M[10000]
001	N		
010	Y	11	M[11010]
011	Y	00	M[00011]
100			
101			
110	Y	10	M[10110]
111	N		

if 10110 is needed by processor, it checks

Scanned by CamScanner

index 110 (cache = 2③)

first it check 110, since V=N & tag & index not same ∴ its a miss. ∴ copy data & store address in corresponding index

Then we need 11010
tag index → miss ∴ store

Then 10000
tag index → miss ∴ store

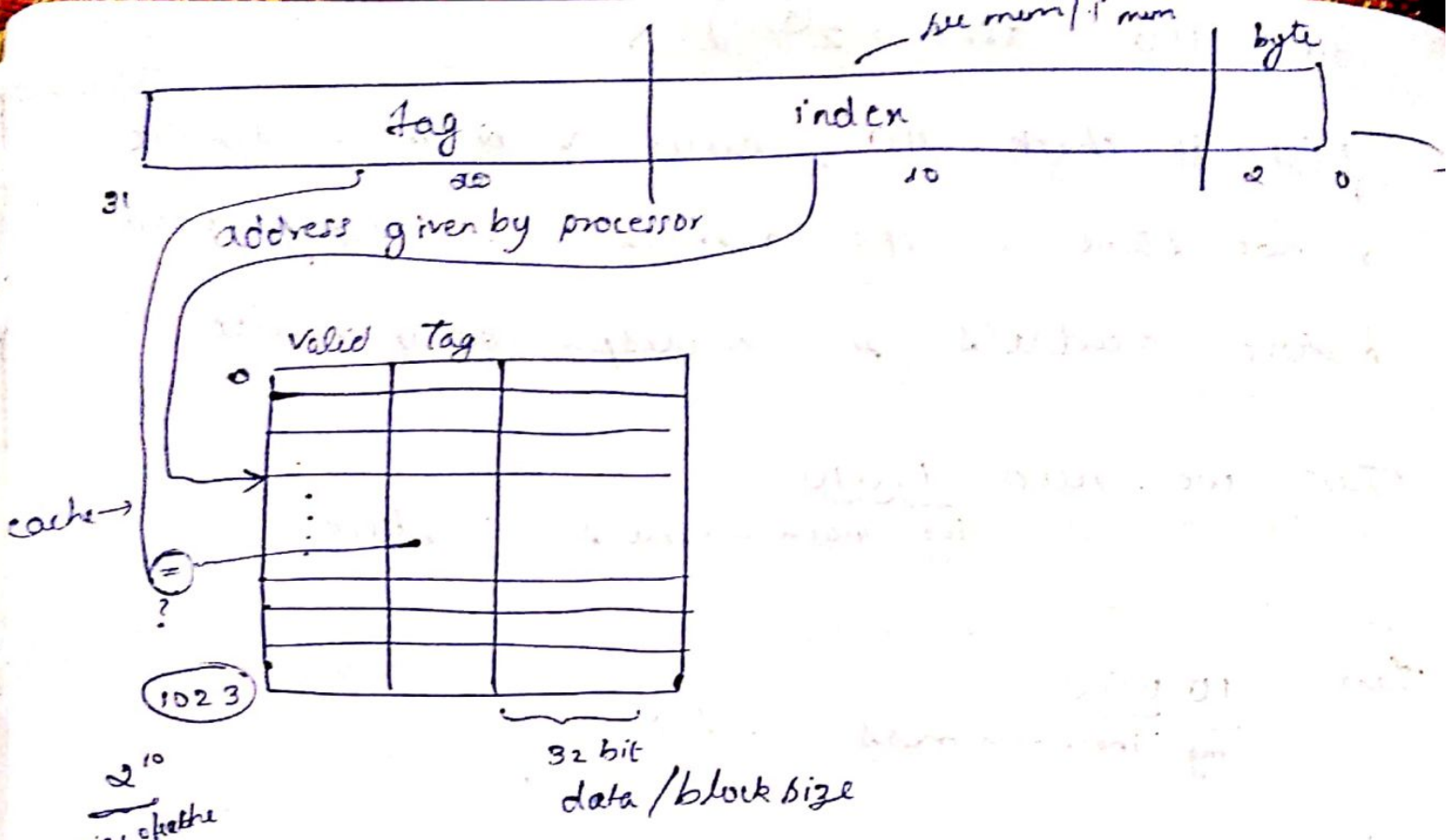
Then 00011
tag index → miss ∴ store.

Then 10010
tag index → miss [index same but tag not equal]

∴ store update tag

∴ if info is changed in sec. memory change valid bit of corresponding address as 'N' in cache.

Scanned by CamScanner



Here we need only part of data stored in cache (earlier we accessed all the bits) stored in cache. Here if we need 1st byte of data. what to do?

→ processor req all data stored in a block

Here we use 10 bits for index

Now since to access byte by byte...

we need 2 bits to know which portion of word you need to access (words has 4 bytes)

00
10
01
11

$\frac{4}{2^2}$

∴ index − 10 tag − 20 bits.
byte portion − 2

Scanned by CamScanner

If processor gives a 32 bit address, we use its index part to find its location in cache, tag part to chuck if its the requested data that is stored in that location & valid bit to chuck validity. If its a hit, then we use the byte part to know which byte portion of the word stored in that cache location is required by processor.

Q] 32 bit byte addressed directed mapped cache (→ access byte by byte) of size 2^n blocks & block size 2^m words. find size of tag field, total no. of bits in directed mapped cache

A: total · $2^n \cdot 2^m \times 32$.

tag field = $32 - n - 2 - m = 32 - [n + m + 2]$

(n: index to identify a word from 2^m words; m: to find the byte portion from each word; $n + m$: index + byte)

total = $2^n [\underbrace{\text{Block size}}_{\text{Store data}} + \text{tag size} + \text{valid}]$.

Scanned by CamScanner

$$= 2^n \left[\underbrace{2^m \times 32}_{\text{bits.}} + 32 - [n+m+2] + 1 \right]$$

Q] How many total bits r req for dir. mapped cache with 16 kB of data & 4 word block assuming 32 bit address

A: tot. block size = 16 kB

= size of 1 block × total no. of block lines

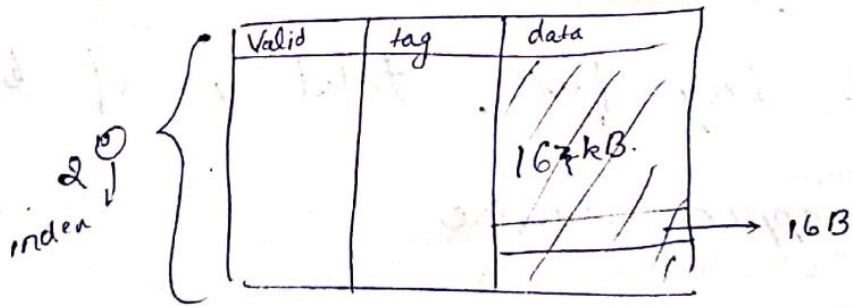
4 × 4 byte

$\Rightarrow$ total no. of block lines = $\frac{16\ kB}{16\ byte}$

1K = 1024 = 2^{10}

= ~~1kB~~

= 2^{10}



$\therefore$ tag = $32 - [10 + 2 + 2]$ (2 for word, 2 for byte)

$= 32 - [14] = 18$

$\therefore$ 1 line = $1 + 18 + 4 \times 32 = 19 + 128$

$= 147$

total = $2^{10} \times 147$ bits = 147 kBit.

Scanned by CamScanner