Computer Architecture & Organisation DA

SRUTHI.S
19MID0053

3. Solve the 100011110/10111 by using the restoring and non-restoring method using algorithms.

Dividend → 100011110

Divisor → 10111

⇒ 000010111

$-B \Rightarrow \bar{B}+1 \Rightarrow 111101000$

$+\quad\quad\quad\quad\quad\quad\quad\quad 1$

$\overline{11111\,01001}$

Restoring :-

comments	A	Q	count
Shift Left	000000000	100011110	9
A = A-B	000000001	00011110□	
	11 111 01001		
Set Q_0	11111 01010	0 00111 10 0	
Restore A.	00 00000001	000 1111 0 0	
Shift left	000000000 10	00111100□	8
A ← A-B	111 11 01001		
	11111 01011	00111100 0	
Set Q_0	00000 10111		
Restore	000000000 10	00111100 0	
Shift left	000000001 0	00111100 0	7
A ← A-B	000000100	01111000□	
	11111 01001		
Set Q_0	11111 01101	01111 000 0	
	00 00010111		
Restore A.	000000100	01111 000 0	

19MID0053

comments	A	Q	count
Shift left	0 0 0 0 0 0 1 0 0 0	1 1 1 1 0 0 0 0	6
A ← A-B	1 1 1 1 1 0 1 0 0 1		
	1 1 1 1 1 1 0 0 0 1		
Set Q_0	0 0 0 0 0 1 0 1 1 1	1 1 1 1 0 0 0 0	
Restore	0 0 0 0 0 0 1 0 0 0	1 1 1 1 0 0 0 0	
Shift left	0 0 0 0 1 0 0 0 1	1 1 1 0 0 0 0 0	5
A ← A-B	1 1 1 1 1 0 1 0 0 1		
	1 1 1 1 1 1 0 1 0		
Set Q_0	0 0 0 0 0 1 0 1 1 1	1 1 1 0 0 0 0 0	
Restore	0 0 0 0 0 1 0 0 0 1	1 1 1 0 0 0 0 0	
Shift left	0 0 0 1 0 0 0 1 1	1 1 0 0 0 0 0 0	4
A ← A-B	1 1 1 1 1 0 1 0 0 1		
Set Q_0	0 0 0 0 0 0 1 1 0 0	1 1 0 0 0 0 0 0	
Shift left	0 0 0 0 0 1 1 0 0 1	1 0 0 0 0 0 0 1	3
A ← A-B	1 1 1 1 1 0 1 0 0 1		
Set Q_0	0 0 0 0 0 0 0 0 1 0	1 0 0 0 0 0 0 1	
Shift left	0 0 0 0 0 0 0 1 0 1	0 0 0 0 0 0 1 1	2
A ← A-B	1 1 1 1 1 0 1 0 0 1		
	1 1 1 1 0 1 1 1 0		
Set Q_0	0 0 0 0 0 1 0 1 1 1	0 0 0 0 0 0 1 1	
Restore	0 0 0 0 0 0 1 0 1	0 0 0 0 0 0 1 1	
Shift left	0 0 0 0 0 0 1 0 1 0	0 0 0 0 0 1 1 0	1
A ← A-B	1 1 1 1 1 0 1 0 0 1		
	1 1 1 1 1 1 0 0 1 1		
Set Q_0	0 0 0 0 0 1 0 1 1 1	0 0 0 0 0 1 1 0	
Restore	0 0 0 0 0 0 1 0 1 0	0 0 0 0 0 1 1 0	

19M1D0053

Q = 1100 ⇒ Quotient = 12

A = 1010 ⇒ Remainder = 10

Non-restoring :-

comment	A	Q	count
Shift left	0000000000	1000 11110	9
A ← A-B	1111101001	0001 1110 □	
Set Q_0	1111101010	0001 1110 [0]	
Shift left	1111010100	0011 1100 □	8
A ← A+B	0000010111		
Set Q_0	1111101011	0011 1100 [0]	
Shift left	1111010110	0111 1000 □	7
A ← A+B	0000010111		
Set Q_0	1111101101	0111 1000 [0]	
Shift left	1111011010	1111 0000 □	6
A ← A+B	0000010111		
Set Q_0	1111110001	1111 0000 [0]	
Shift left	1111100011	1110 0000 □	5
A ← A+B	0000010111		
Set Q_0	1111111010	1110 0000 [0]	
Shift left	1111110101	1100 0000 □	4
A ← A+B	0000010111		
Set Q_0	0000001100	1100 0000 [1]	
Shift left	0000011001	1000 0001 □	3
A ← A-B	0000010111		
Set Q_0	0000000010	1000 0001 [1]	
Shift left	0000000101	0000 0011 □	2
A ← A-B	1111101001		
Set Q_0	1111110110	0000 0011 [0]	

19MID0053

comments	A	Q	count
shift left A ← A+B set Q_0	1111011100 0000010111 1111110011	0000001[1]0 0000001[1][0]	1
A ← A+B	1111110011 0000010111 0000001010	00000110 0	0
Quotient → 0000001100 → 1100 ⟹ 12 Remainder → 0000001010 → 1010 ⟹ 10			

(5) Discuss the algorithm for multiplication of -8 x -14 how the sign magnitude is handled.

BR = 11000 QR = -14 = 10010

$\overline{BR}+1$ = 01000.

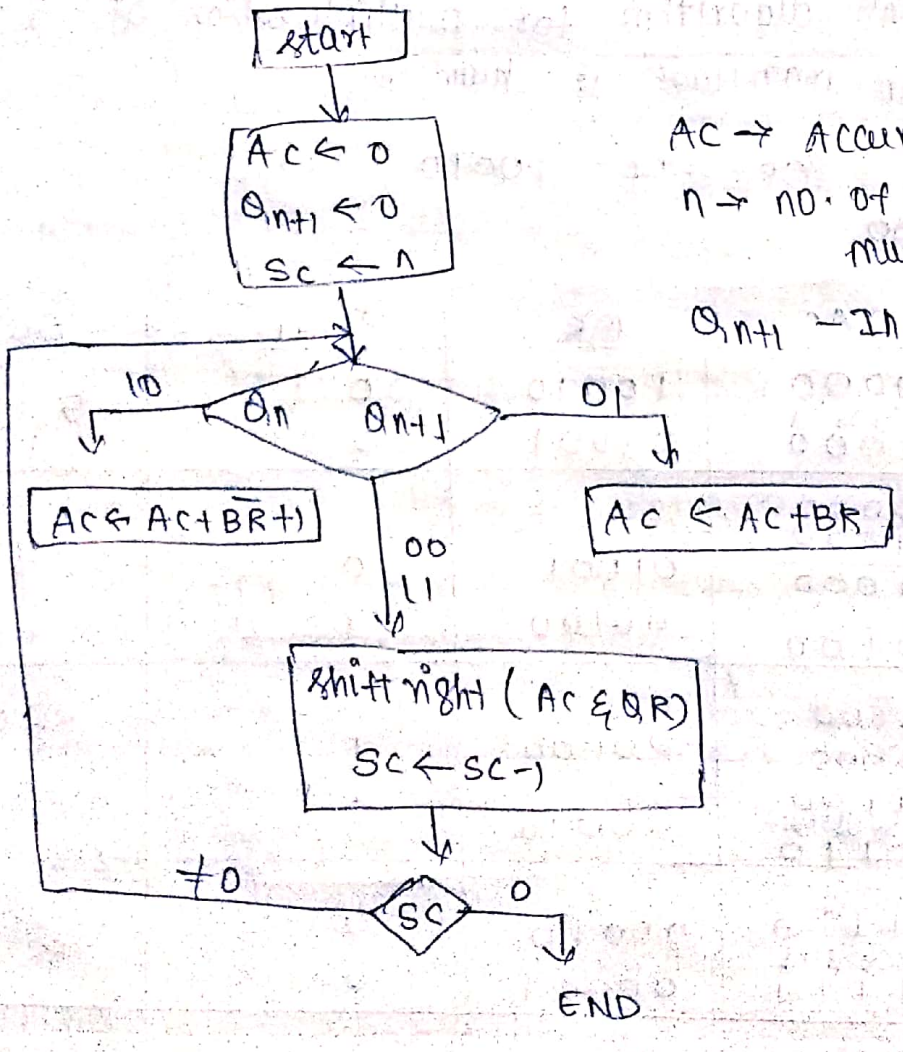
comment	AC	QR	Qn+1	SC
shift right	00000 00000	10010 01001	0 0	5
AC ← AC + $\overline{BR}+1$ shift.	00000 01000 01000 00100	01001 00100	0 1	4
AC ← AC + BR shift.	00100 11000 11100 11110	00100 00010	1 0	3
shift.	11110 11111	00010 00001	0 0	2
AC ← AC + $\overline{BR}$ + 1 shift	11111 01000 00111 00011	00001 10000	0 1	1

$\therefore 0001110000 = 112$

$-8x - 14 = 112.$

The signed magnitude representation of binary number must have either 0 (or) 1. For the positive signed binary we add '0' to the left most bit. For the given negative decimal number we represent '1' (or) we take 2's complement of it. So that we can get binary number of the negative decimal number.

Multiplicand in BR
Multiplier in QR



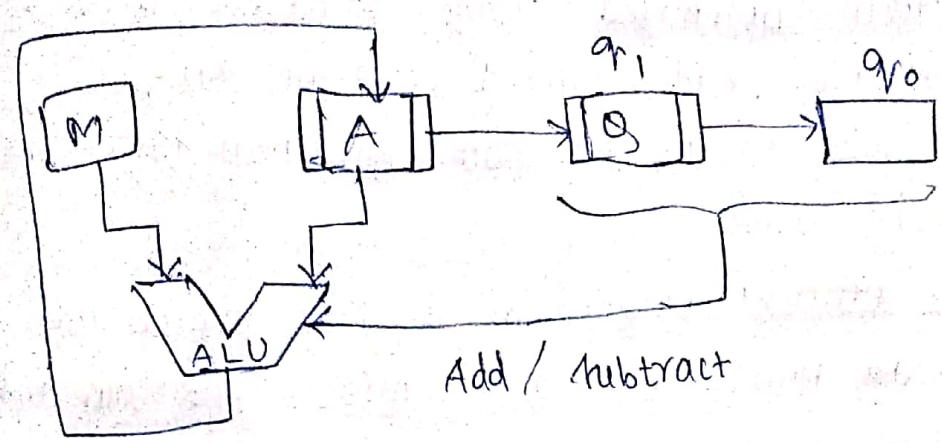
AC → Accumulator
n → no. of bits in multiplier
Q_{n+1} – Initially zero

(10) What is the modification to be done on booth multiplication algorithm when 8 numbers are given

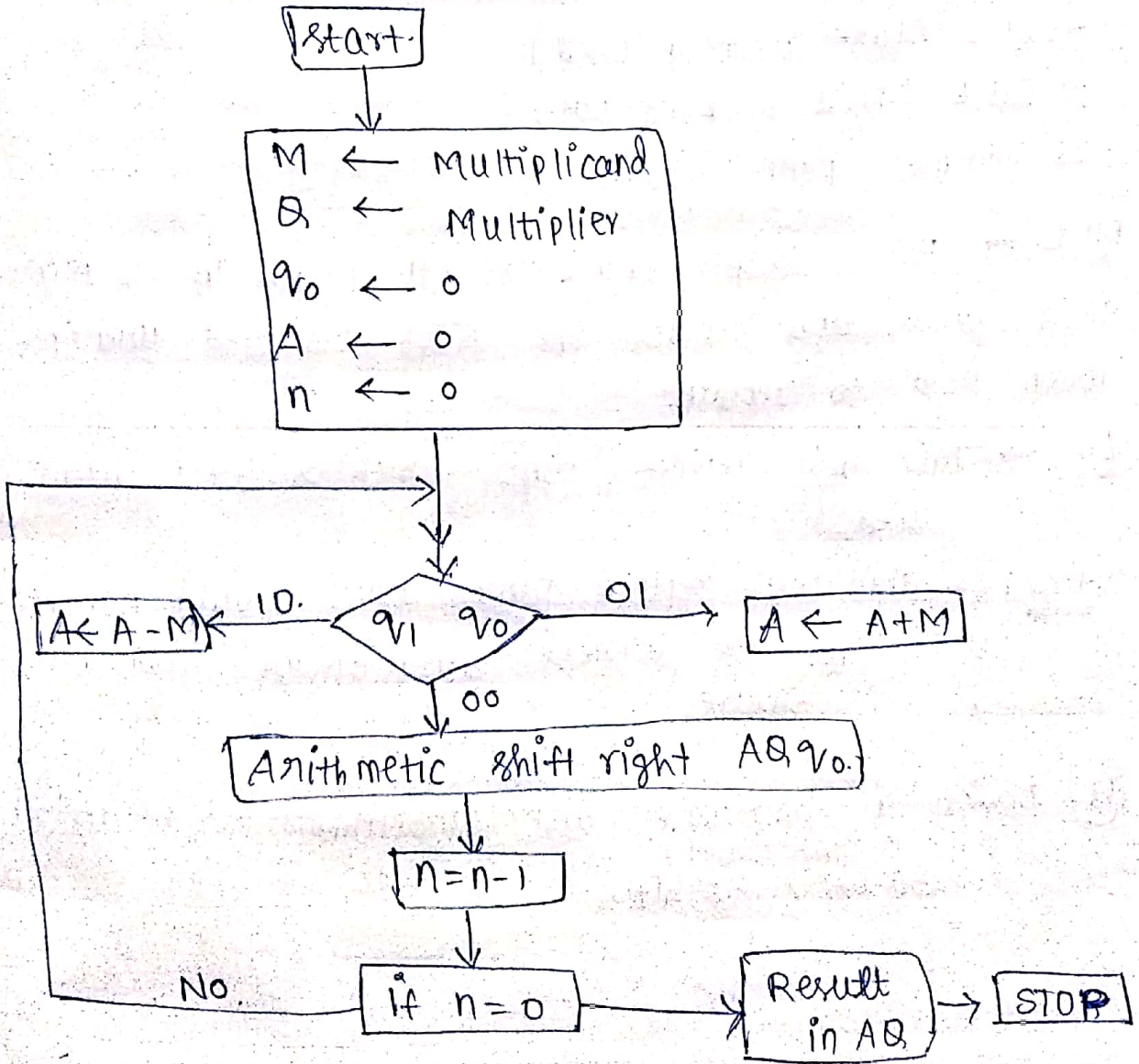
(1000 X 1001 X 0111)

19MID0053

Hardware for Booth Algorithm:



A → Accumulator | M → Multiplicand | Q → Multiplier.



(11.) Justify → "Replacement algorithm not used in cache memory" any other mechanism can be adopted ?

Replacement Algorithm :-

Replacement algorithm / policies are used in order to attain optimized usage of cache. When cache is full then this decides which piece of data, is replaced in order to make space for new data that is currently being used.

An efficient algorithm is that which can take less time and number of cache misses are low and also balancing cost.

Following are some of the algorithms.

→ LRU (Last recently used).
→ FIFO (First in First out).
→ optimal page replacement

FIFO → This keeps track of all pages in the memory in a queue and replace the first occupied data (pages) when need to replace.

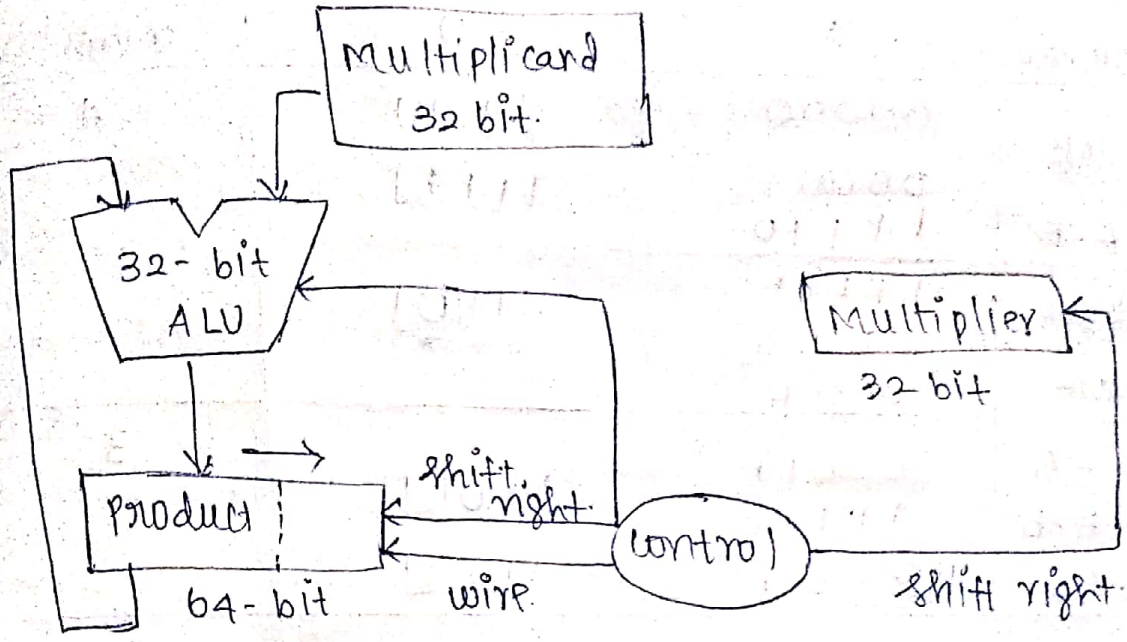
LRU → This will replace pages which is least recently used.

Optimal → This will replace pages which would not be used for the longest duration of time in future.

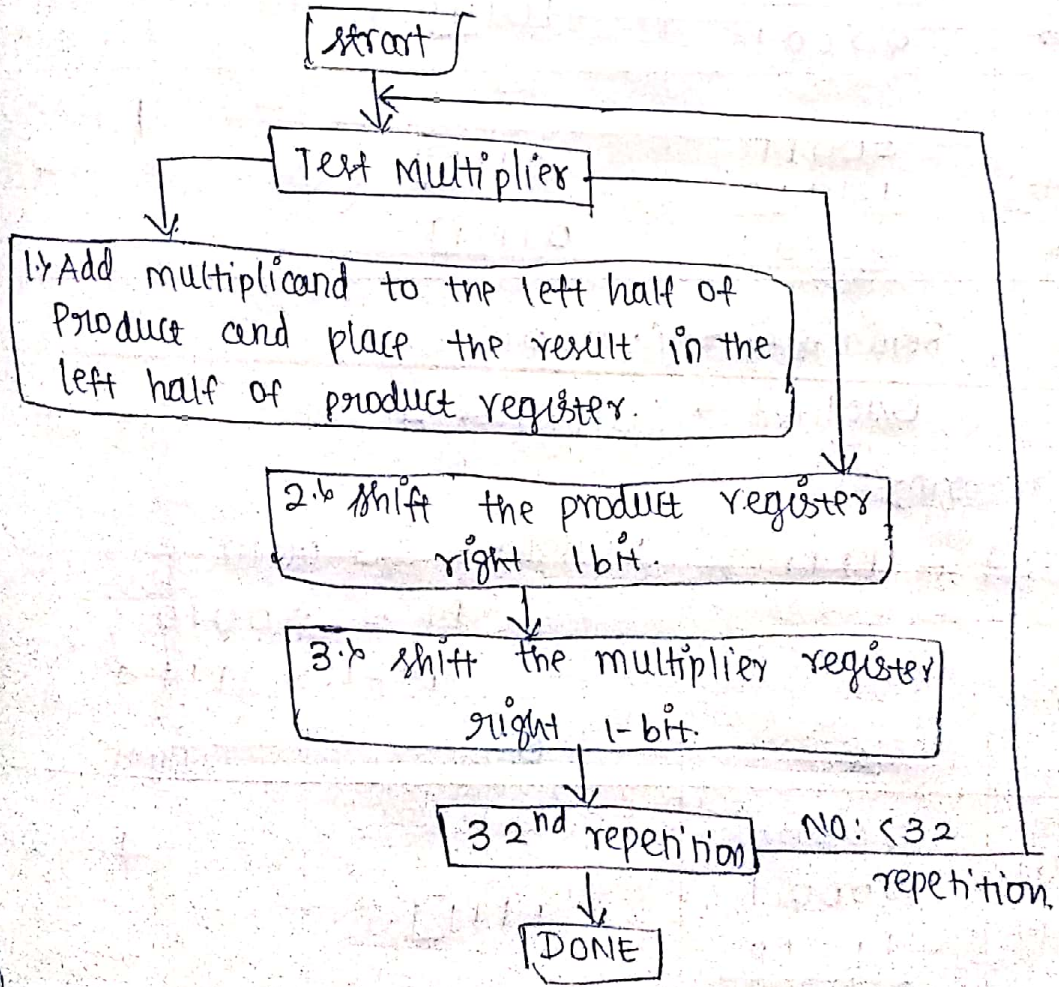
(2.) Design a hardware and algorithm for multiplying 32 bit x 32 bit.

Hardware :-

19M1D0053



Algorithm :-



(13.) Compare & contrast the application of restoring and non-restoring division method for an example 15/2.

Restoring division ⇒ Divisor → 2 = 00010 ,

Dividend ⇒ 15 = $(1111)_2$ (6)

B = 0010 ; $\bar{B}+1$ = 1110

comment	A	Q	count
shift left	00000	1111	4
A ← A-B	00001 11110	111	
set Q_0	11111	1110	
Restore	00010 0001.		
shift left	00011	110	3
A ← A-B	11110		
set Q_0	00001	1101	
shift left	00011	101	2
A ← A-B	11110		
set Q_0	00001	1011	
shift left	00011	011	1.
A ← A-B	11110		
set Q_0	00001	0111	

Remainder → (00001) ⇒ 1

Quotient → (0111) ⇒ 7.

Non-restoring :

Dividend – 1111. Divisor – 2 = 0010

M = 00010

$\bar{M}+1$ = 11110.

comment	A	Q	count
shift left	00000	1111	4
A ← A-M	00001 11110	111	
set Q_0	11111	1110	
shift left	11111	110	3
A ← A+M	00010		
set Q_0	00001	1101	

comment	A	Q	count
shift left A ← A-M set Q_0	00011 11110 00001	101□ 101□	2
shift left A ← A-M	00011 11110 00001	011□ 011□	1

Remainder → (00001) ⇒ 1

Quotient → (0111) ⇒ 7.

In restoring method, we add divisor when we get Q_0 as 0. In non-restoring we do not add divisor if we set Q_0 as either 0 (or) 1. We reduce the count & proceed the algorithm.

(14.) Differentiate the multiplication algorith & Booth multiplication algorithm with an example of 30×16.

Multiplication Algorithm :-

30×16 B = 11110 ⇒ 30 ; Q = 10000 = 16.

comment	E	A	Q	S.C.
$Q_n = 0$ shr EAQ	0 ↓ 0	00000 00000	10000 01000	5
$Q_n = 0$ shr EAQ	↓ 0	00000	00100	4
$Q_n = 0$ shr EAQ	↓ 0	00000	00010	3
shr EAQ	0	00000	00001	2
$Q_n = 1$ EA = A+B shr EAQ	0 ↓ 0	11110 11110 01111	00001 00000	1

$(01111100000)_2 = (480)_{10}$

$30 \times 16 = 480.$

Booth algorithm :-

Multiplicand × Multiplier

BR × QR

30 × 16

$30 \Rightarrow 11110 \Rightarrow 011110$

$16 \Rightarrow 10000 \Rightarrow 010000$

$\bar{BR}+1 = 100010.$

comment	Ac	QR	Qn+1	Sc
shift right	000000 ⇘⇘⇘⇘⇘ 000000	010000 001000	0 0	6
shift right	000000	000100	0	5
shift right	000000	000010	0	4
shift right	000000	000001	0	3
Ac ← Ac+BR+1	100010	000001	0	2
shift right	110001 ⇘⇘⇘⇘⇘	000000	1	
Ac ← Ac+BR	011110	000000	0	1
shift right	001111 ⇘⇘⇘⇘⇘ 000111	100000	0	

$(0001111\ 00000) = (480)_{10}$

$30 \times 16 = 480.$

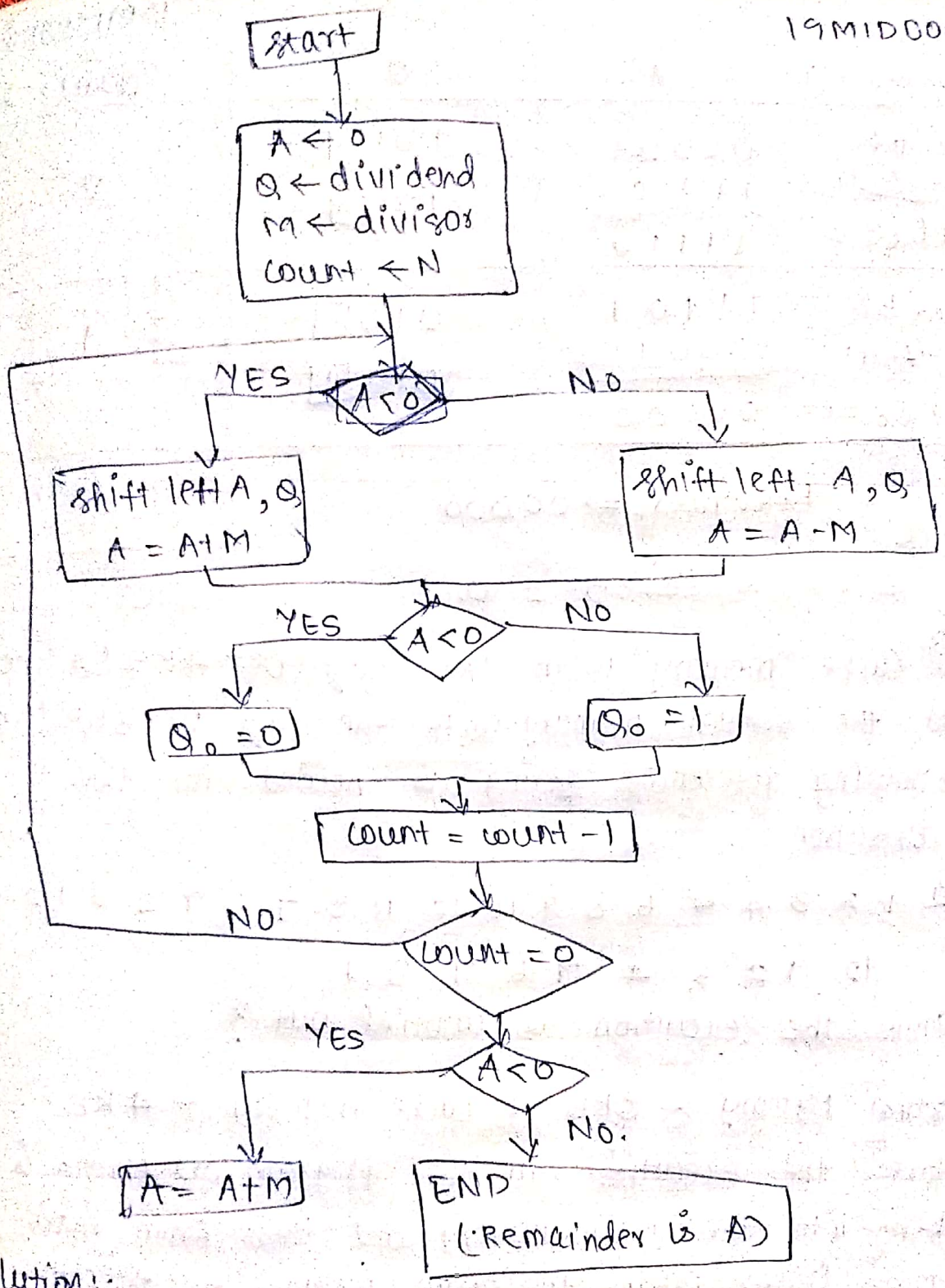
(15) Device an algorithm for performing 15 % 3

Dividend (Q) = 15 = 1111

Divisor (M) = 3 = 00011

$\bar{M}+1 = 11101$

19MID0053



Solution:-

comment	A	Q	count
shift left	0 0 0 0 0	1 1 1 1	4
A = A-M	0 0 0 0 1	1 1 1 □	
set $Q_0 = 0$	1 1 1 0 1	1 1 1 0	
	1 1 1 1 0		
shift left	1 1 1 0 1	1 1 0 □	3
A = A+M	0 0 0 1 1		
set $Q_0 = 1$	0 0 0 0 0	1 1 0 1	

Comment	A	Q	Count.
shift left A = A-M set $Q_0 = 0$	00001 11101 11110	101□ 101[0]	2
shift left A = A+M set $Q_0 = 1$	11101 00011 00000	010□ 010[1]	1

Remainder ⇒ 00000

$\therefore 15/3 = 0.$

(16.) Cache memory with the size of 4KB (4 lines) and the virtual memory with the size of 8KB (lines) following reference string is needed for the execution.

1 2 1 2 3 4 5 6 8 9 10 12 11 8 9 6 7 2 3 1 2 11

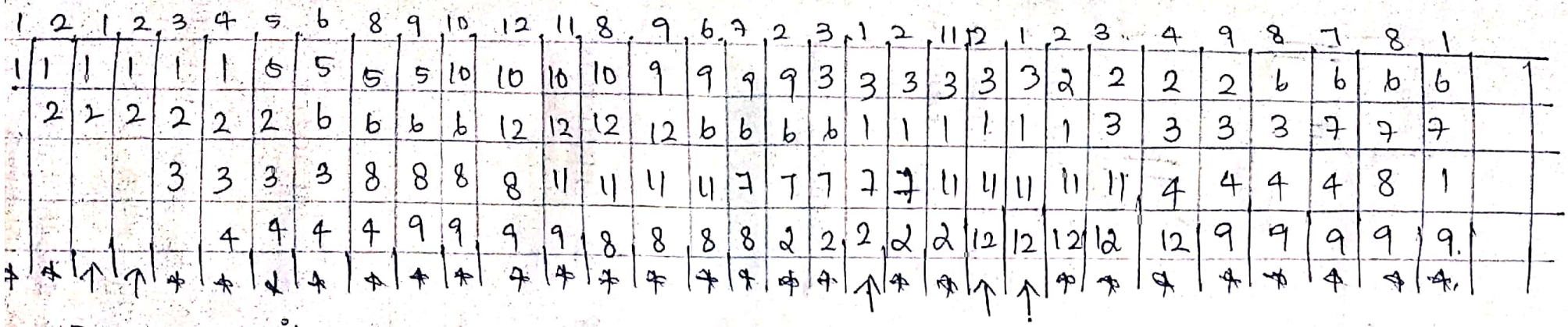
12 1 2 3 4 9 6 7 8 1.

How the execution is carried out?

Virtual Memory – 8KB ; Cache memory – 4KB.

Before the execution all the program (or) string is taken into the main memory and then from main memory it is taken to cache memory to execute.

FIFO:



FIFO :- Hit = 5 page fault = 27.

Hit ratio = $\frac{5}{32} \times 100$ = 15.6%

fault ratio = $\frac{27}{32} \times 100$ = 84.3%.

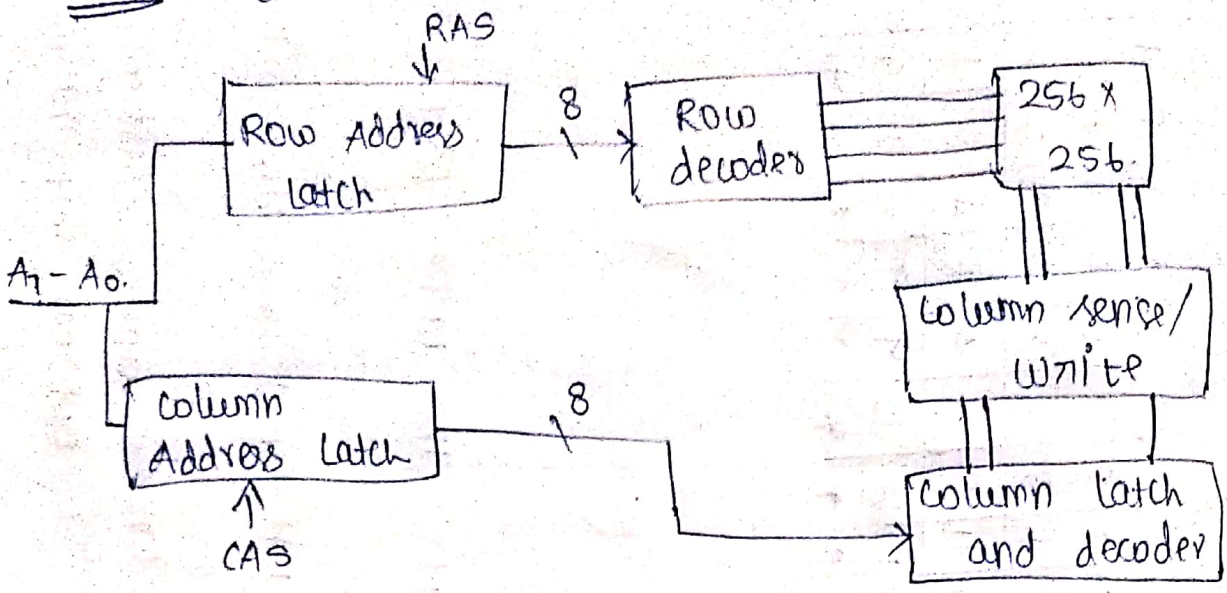
FIFO (First in First out) → Whichever page comes into the cache first it will be placed & executed & left first.

LRU (Least recently used) → Here pages which are not used recently will be removed.

Optimal → Here the pages that are not used for a longest time in future will be removed. As the name suggests it reduces and gives the optimal hit ratio.

(17.) Design hardware for storing a 6-bit number in the dynamic RAM.

6 Bit:- $2^6 = 64 \text{ K} \times 1$.



Like we have the address bit $A_7 - A_0$ that is 8 lines then we give this 8 line to Row address Latch & column address latch.

To accumulate 6-bit we need 2^6 memory that is 64 K.

Dynamic RAM (DRAM):-

→ slower than S-RAM.

→ It stores data as charge on capacitors

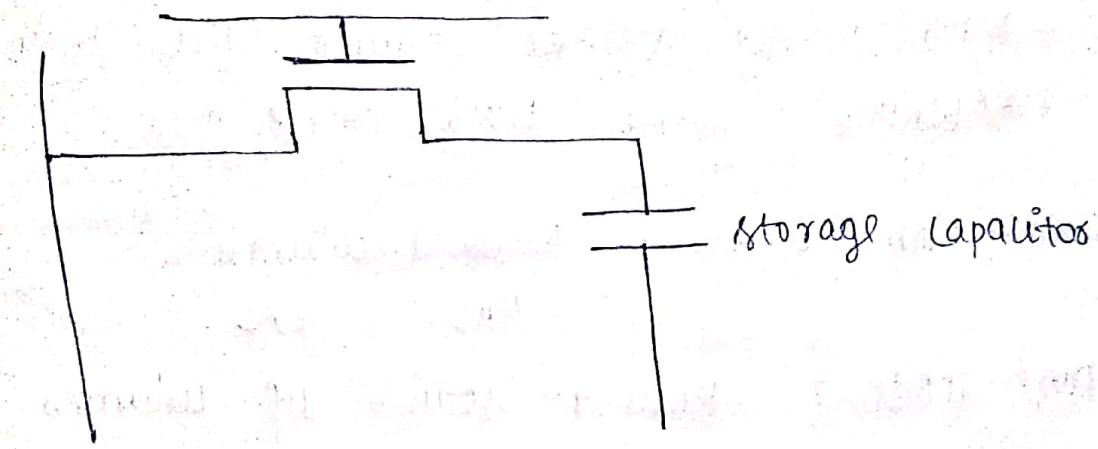
* if capacitor is charged
  data 1
  else
  data 0.

→ Needs refreshing cycle as capacitors have tendency of discharging.

→ The term dynamic refers to this tendency of the stored charge to leak away even with power continously applied.

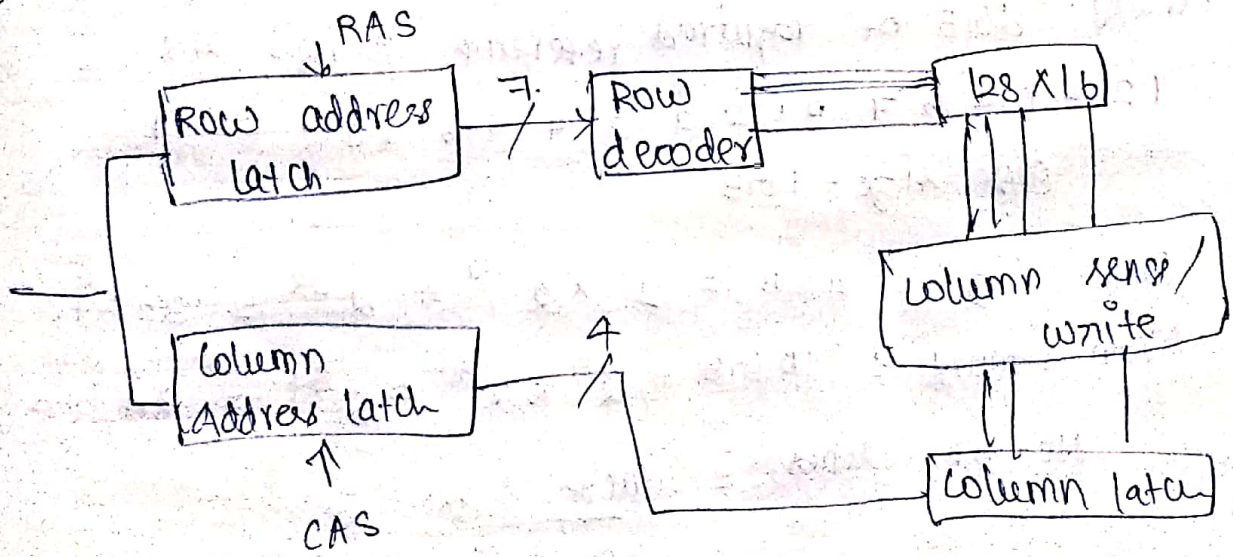
→ volatile → It consists ⇄ 1 transistor
1 capacitor.

→ When read data is lost, so restoring need to be done.



Here transistor act as switch if transistor is
Closed → allows current flow
else → No current.

(18.) Design a DRAM for 128 X 16.



Row Address :

→ Set Row address on address line & stroke RAS.
→ Entire row read & stored in column latch
→ Contents of row of memory all destroyed.

Column Address :

→ Set column address on address line & stroke CAS.

→ Access selected bit :

- READ : transfer from selected column latch to Dout.
- WRITE : set selected column latch to Din.
- REWRITE : Write back entire row.

Conventional access : Row + column
RAS CAS.

Page mode : Row + series of columns
Gives successive bit.

(1.) Designed a cache memory with the size of 4GB, main memory (16GB), start the execution of task requires 3GB of data, dynamically new requirement of 2GB of data needed, how can this situation handled in cache memory. 2GB of required reference strings are

1 2 3 0 4 5 7 0 1 2 2 3 4 5 6 7 8 7 0 0 1 1

page size = 1GB.

Cache size = 4GB = $2^2 \times 2^{30} \Rightarrow 2^{32}$ = 32 bit.

Main memory = 16GB = $2^4 \times 2^{30} \Rightarrow 2^{34}$ = 34 bits.

No. of lines = $\frac{\text{Cache size}}{\text{Page size}}$

= 4.

Optimal page replacement algorithm :-

	1	2	3	0	4	5	7	0	1	2	2	3	4	5	6	7	8	7	0	0	1	1
f_1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
f_2		2	2	2	2	2	2	2	2	2	2	3	4	5	6	6	8	8	8	8	8	8
f_3			3	3	4	5	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
f_4				0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	*	*	*	*	*	*	*	↑	↑	↑	↑	*	*	*	⊗	↑	*	↑	↑	↑	↑	↑

Optimal page replacement algorith :-

HIT = 10

page fault = 12

Hit ratio = $\frac{10}{22} \times 100$

$= 45.45$

Fault ratio = $\frac{12}{22} \times 100$

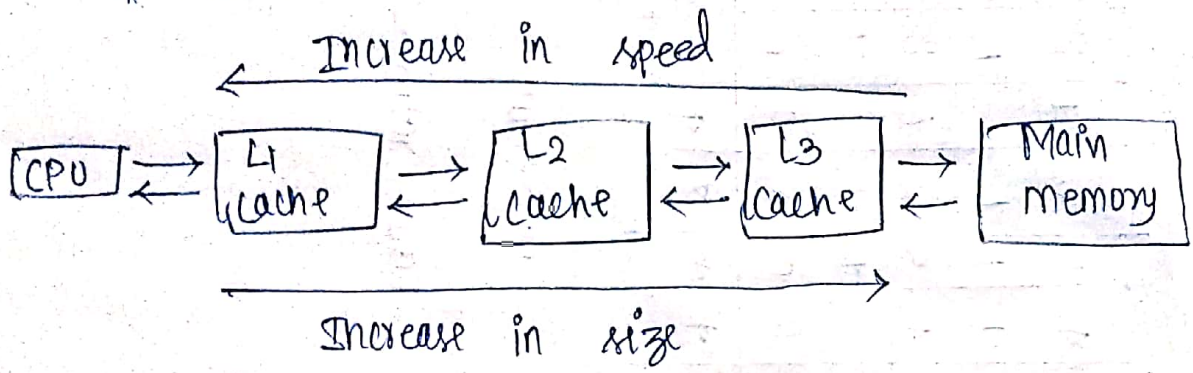
$= 54.54$

19MID0053

(4.) Assume that different levels of cache are allowed, how this will affect the performance of the system (or) improve the performance of the system.

There are three levels in the cache memory

* Level 1 (L_1) cache memory
* Level 2 (L_2) cache memory
* Level 3 (L_3) cache memory



L_1 is more near to the CPU register. If the data is present in the L_1 cache memory, its performance will be fast and it is small size compared to the remaining 2 levels cache memory.

L_2 is in the middle of L_1 and L_2. If the data is present in the L_2 cache memory then the performance will be little fast not slow, and it is bigger in size compared to the L_1 cache memory.

L_3 is in the nearest to the main memory and it is the biggest cache memory. If the given data present in the L_3 then the performance of the system is slow.

Affect the performance of system:

→ cache memory is different in different devices, due to which it quickly slow down your computer.

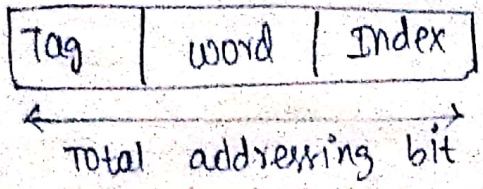
→ If the die cache takes up real estate on the die, it seems possible, the real estate could be used for other purposes.
→ Even after you clear the cache memory your system seems to be hang.
→ If a CPU locked a cache, program could run more slowly.
→ Cache memory has limited capacity. Hence it makes system to hang in certain times.
→ Cache memory holds frequently used instructions data which the processor may require next.
→ Cache is a small amount of memory.

(6.) Following requirements are given, Cache memory - 64 KB and the main memory - 128 KB, Frame - 4, discuss the advantage & disadvantage over different mapping procedure with the above given requirements.

Cache memory = 64 KB
Main memory = 128 KB
Frame = 4

Types of mapping :-
→ Direct → Associative → set associative.

Direct mapping :-



Index = $\log_2$ (No. of blocks in cache)
= $\log_2$ (16)

Index = 4 bits.

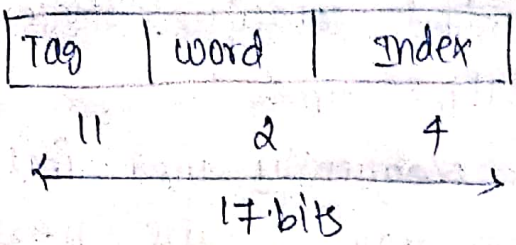
Word = $\log_2(2^2)$ = 2 bits.

Total addressing bit = $\log_2$ (main memory size)

= $\log_2(128 \times 2^{10})$

= 17 bits.

Tag bit = 17 − 6 = 11 bits



Advantages :-

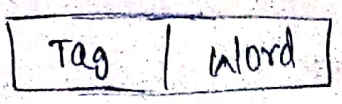
→ The i^{th} block of main memory is placed at the j^{th} block of cache memory, (i,e) j = mod (No. of blocks in cache). The blocks are directly mapped onto cache memory. This reduces time.

Disadvantage :-

→ High conflict miss - we have to replace the cache memory block even when other blocks in the cache memory are empty.

Associative mapping :-

Here the main memory address space is divided into 2 parts ⇒ tag & word.

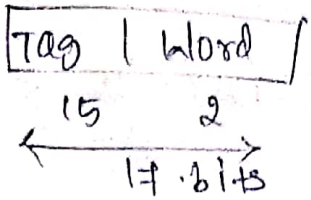


Total addressing bit = $\log_2(128 \times 10)$

= 17 bits.

Word = $\log_2(2^2)$ = 2 bits

Tag = 17 - 2 = 15 bits.



Advantage :

→ 0% of high conflict miss occurs.

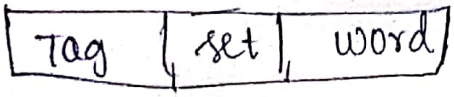
Disadvantage :

→ very high tag comparisons

Tag comparisons = No. of blocks in cache.

Set associative mapping :

Here the main memory address space is divided into 2 parts.



No. of sets = $16/2 \Rightarrow 8$

set bit = $\log_2(8) \Rightarrow 3$ bits.

Word bit = $\log_2(4) \Rightarrow 2$ bits.

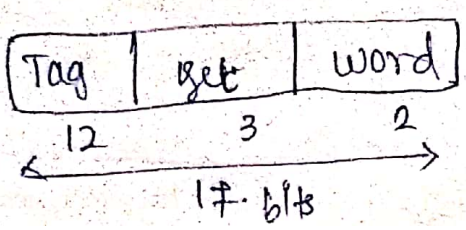
Total addressing $\Rightarrow \log_2$ (Main memory size)

$= \log_2(128 \times 2^{10})$

$= 17$ bits

Tag bit = 17 - 3 - 2

= 12 bits.



pros & cons :

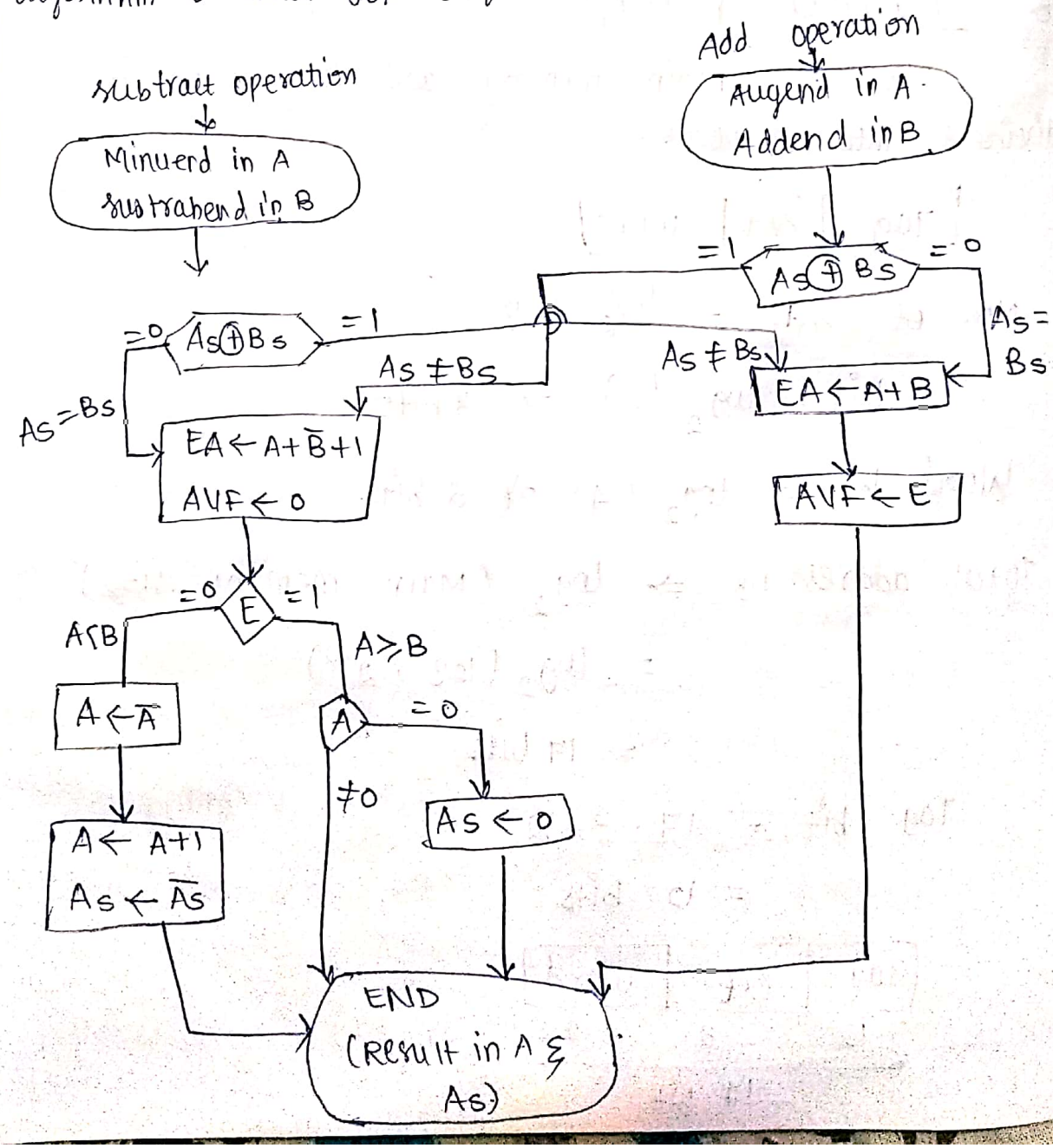
→ A new block from main memory can be placed anywhere in the set.

→ The ith block of main memory have to be placed in jth set of cache memory.

$$j = i \pmod{\text{no. of set in cache}}$$

→ This is also used to overcome high conflict miss of direct mapping.

(2.) Assume that the new design of processor involves 16 bit addition of 2 numbers, how can you design the algorithm to match cases (signed & unsigned) with an example



19M1D0053

Example :-

A = 1001001001001001 1

B = 1101110000110100 0

$A_S \oplus B_S = 1 \oplus 1 = 0$

$\Rightarrow (A+B)_S = 0$.

EA ← A+B

sign bit

```
 1 | 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 1
 1 | 1 0 1 1 1 0 0 0 0 1 1 0 1 0 0 0
---+---------------------------------
 0 | 1 1 0 1 1 1 0 0 1 1 1 1 1 0 1 1
```

AVF ← E

AVF = 0

⇒ A+B = 0 1101110011111011

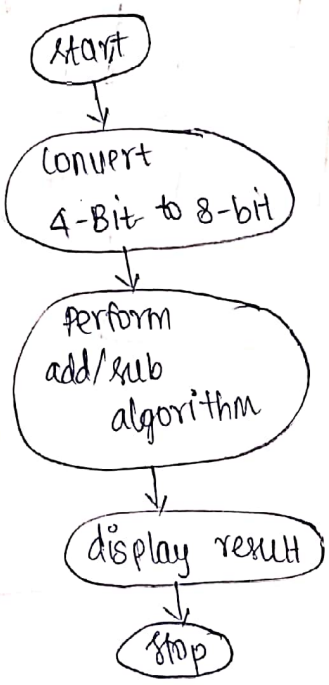
(15 Bits + 1 sign bit)

(7.) Register A holds 8 (11100111) bit number, register B holds the 4 (1111) bit number, how can you develop the algorithm in your design?

19MID0053

We need to have same number of Bits to add (or) subtract.

∴ We need convert to 4 Bit number to 8 bit number



Initialize result R=0

1st number : 11100111 → 231

2nd number : 1111 → 15

converting to 8-Bit number

00001111. 246

add 11100111

00001111

11110110

Result :

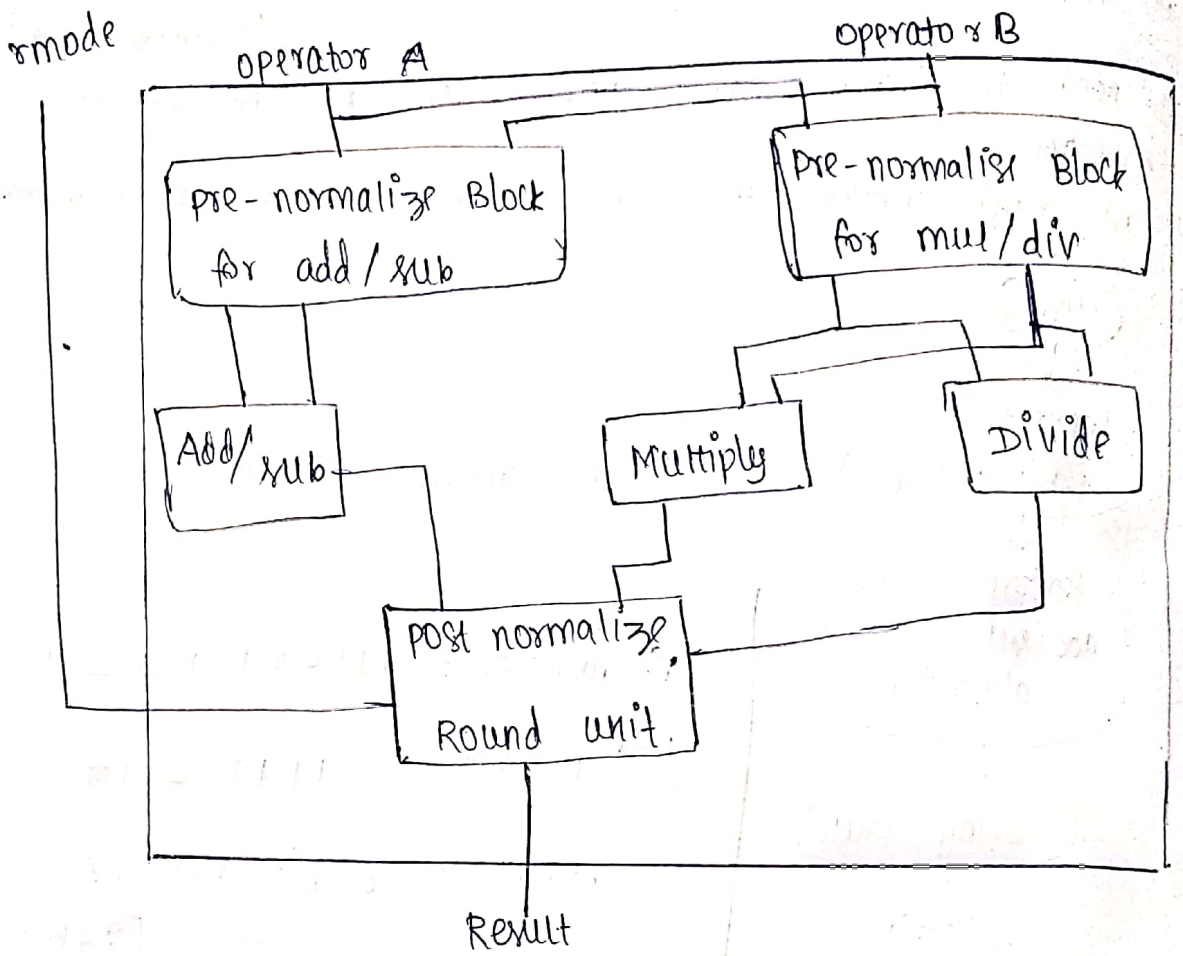
1st no. + converted second number

→

Now store A as augend & B as addend → for addition

And A as minuend & B as subtrahend → for subtraction.

(8.) Design hardware (single unit) to perform [9M1D00S3
addition, subtraction, multiplication, division.



Here we can perform addition, subtraction, multiplicatio and division in same hardware.

Pre-normalize Block helps us to understand whether the operation is add/sub (or) mul/div.

(9.) Main memory with size of 16kB & virtual memory is 64kB and execution of single program need 1GB of reference string. How can you resolve this issue?

Main memory size → 16 kB

16×2^{10} B ⇒ 2^{14} Bytes

19MID0053

Virtual memory size → 64 KB

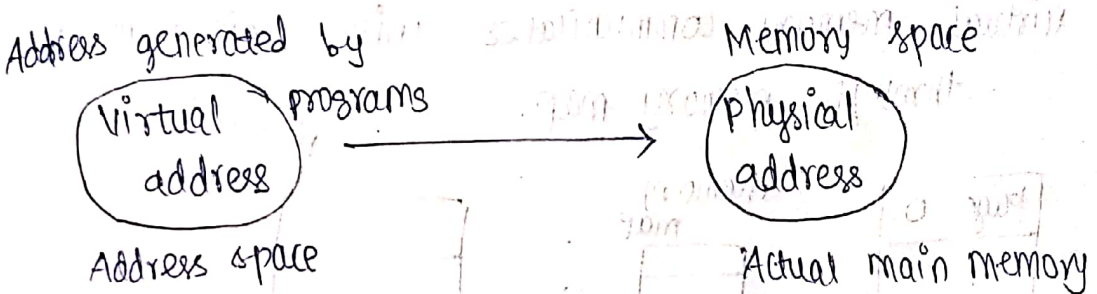
$= 2^6 \times 2^{10}$ B ⇒ 2^{16} Bytes.

Our need : 1GB : 2^{30} Bytes.

→ This problem can be resolved by concept by virtual memory.

→ Virtual memory gives a illusion that system has a very large memory, even though system actually has relatively small main memory.

→ As our main memory size is 2^{14} B, we can store 2^{14} B from 2^{30} B need, rest of the memory can be used by concept of virtual memory.



→ The CPU requests for "pages" which is part of a process. The required page is brought into main memory, thus satisfying the need.

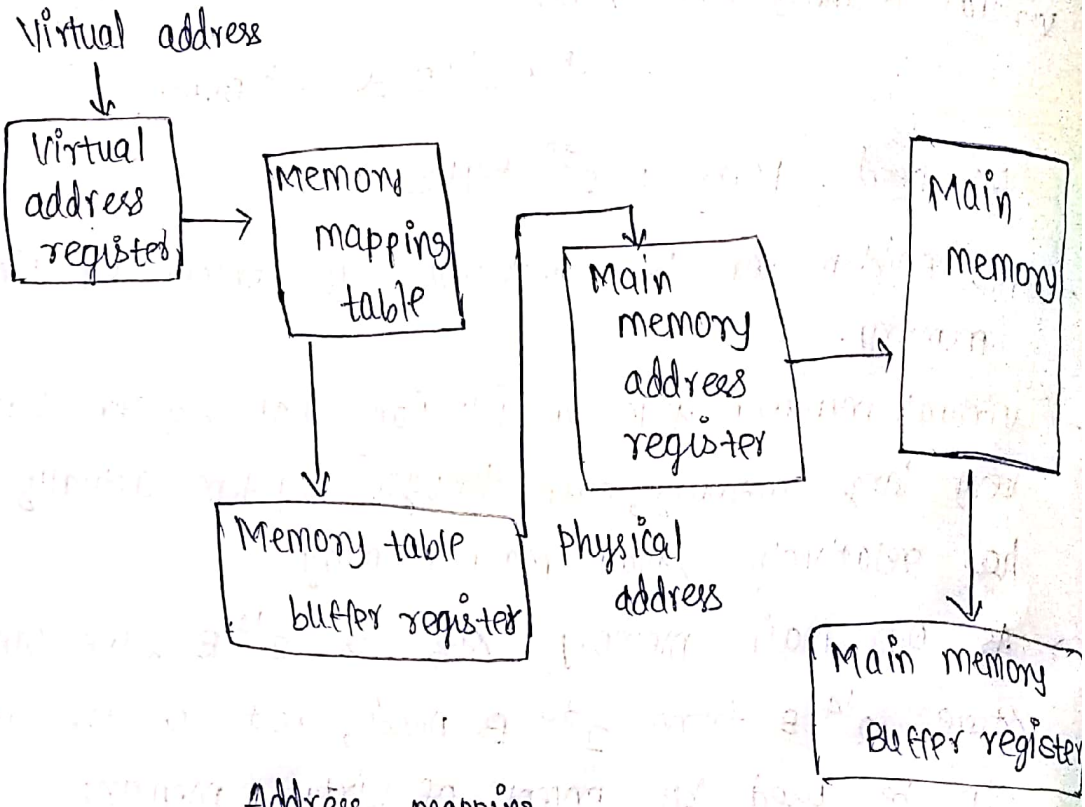
→ We use 3 kind of page replacement algorithms

⇒ FIFO ⇒ LRU ⇒ Optimal.

→ If the required page is already present in main memory, then it is called hit.

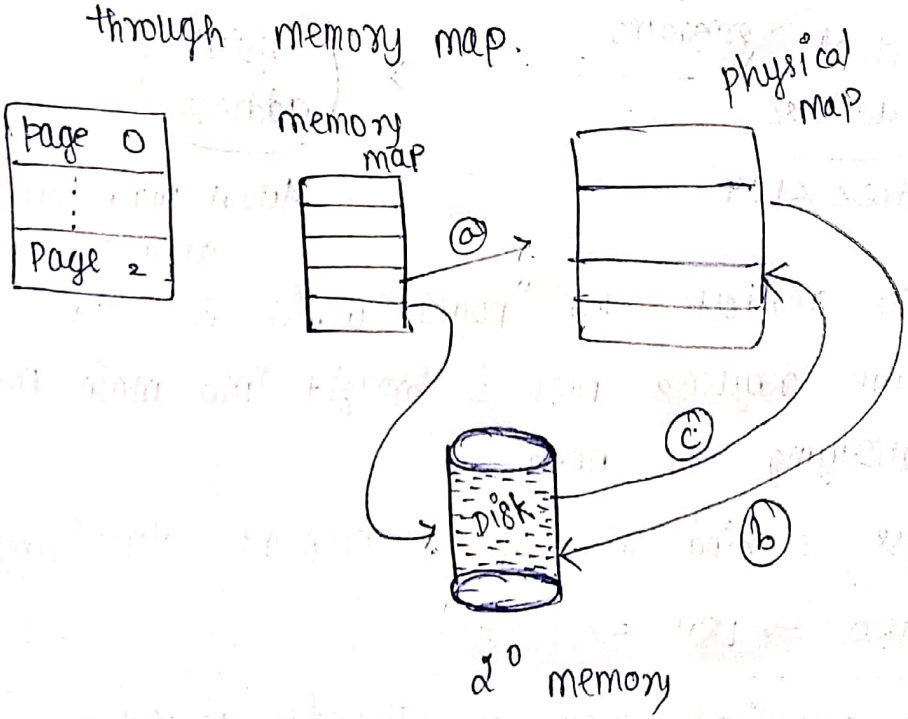
→ If the page is absent in main memory, then it is called miss (or) page fault.

19MID0053



Address mapping

→ Virtual memory communicates with main memory through memory map.



2° memory

→ Memory map table will have information whether data is available in physical memory (or) not.

Advantages of Virtual memory :- 19MID0053

→ less I/O needed.

→ less memory needed

→ faster response.

* Virtual memory can be implemented via

→ demand paging

→ demand segmentation.