CSI1004

COMPUTER ORGANIZATION AND ARCHITECTURE

Name : S.THARUN

Reg No : 19MID0031

1) Cache size = 4 GB = $2^2 \times 2^{30} = 2^{32}$ ⇒ 32 bits

Main Memory = 16 GB = $2^4 \times 2^{30} = 2^{34}$ ⇒ 34 bits

No. of lines = $\frac{\text{cache size}}{\text{page size}} = 4$

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

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

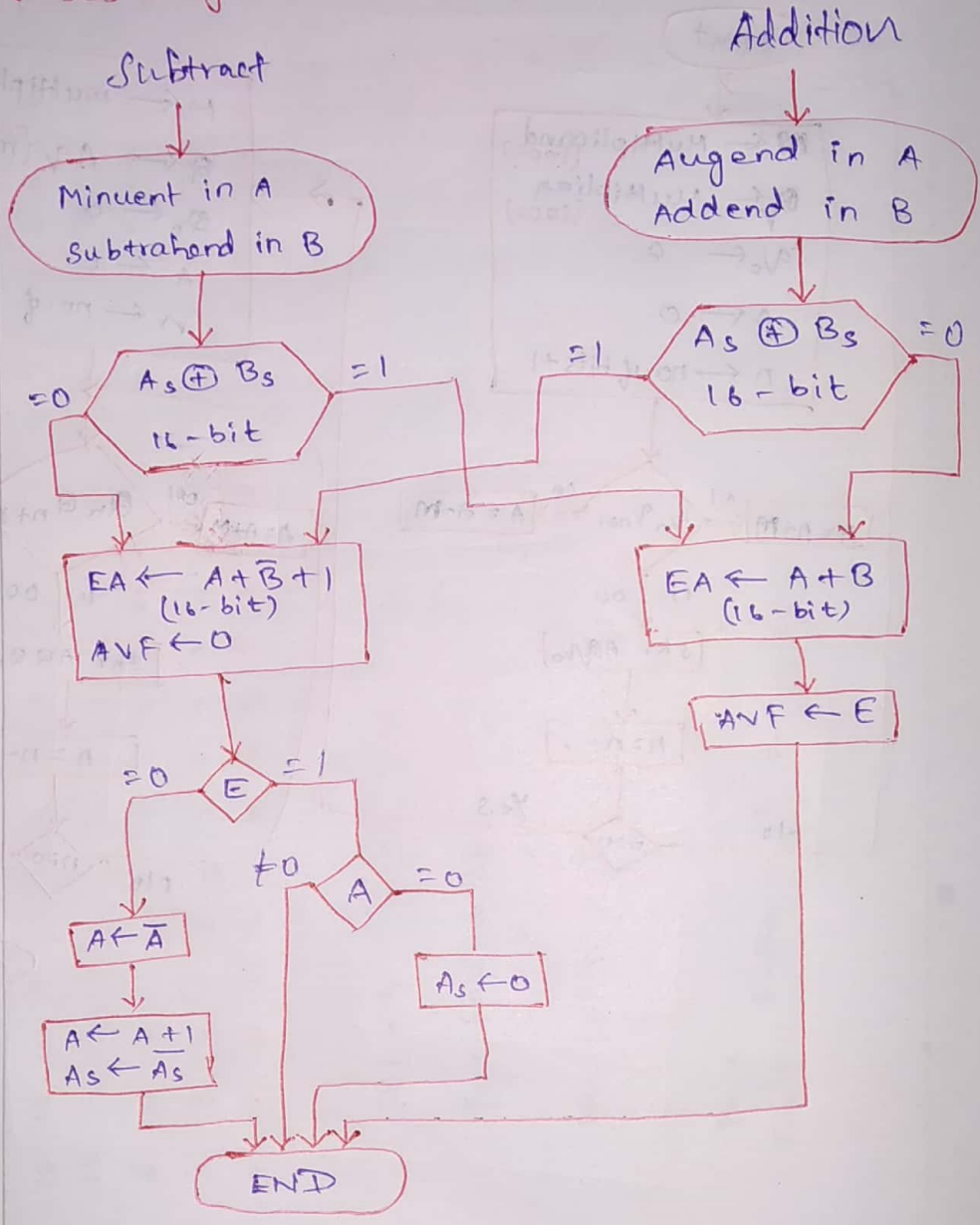
Hit = $\frac{10}{22}$ Page fault = $\frac{12}{22}$

Hit ratio = $\frac{10}{22} \times 100$ = 45.45

Miss Ratio = $\frac{12}{22} \times 100$ = 54.55

Algorithm used: optimal

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



Eg:

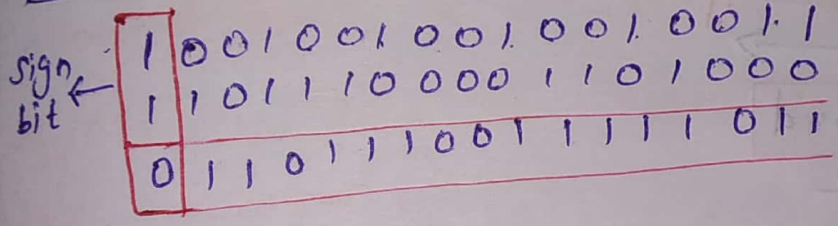
A = 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 1

B = 1 1 0 1 1 1 0 0 0 0 1 1 0 1 0 0 0

$A_s \oplus B_s = 1 \oplus 1 = 0$

$\Rightarrow (A+B)_s = 0$

EA ← A + B



AVF ← E

AVF = 0

⇒ A+B = 0110111001111011

(16 bits + 1 sign bit)

3) Q = 10001110

M = 10111 = 000010111

-M = 111110100

Restoring

	A	Q	n
	000000000	10001110	9
Shift left	000000001	00011100	
A = A-M	111110100		
	111110101	00011100	
A = A+M (Restore)	000010111		
	000000001	00011100	8
Shift left	000000010	00111000	
A = A-M	111110100		
	111110110	00111000	
Restore A	000010111		
	000000010	00111000	7
Shift left	000000100	01110000	
A = A-M	111110100		
	111110100	01110000	
Restore A	000010111		
	000000100	01110000	6

17M1D0031

Shift left	0 0 0 0 0 0 1 0 0 0	1 1 1 1 0 0 0 0 □	
A=A-M	1 1 1 1 1 0 1 0 0 1		
	1 1 1 1 1 1 1 0 0 0 1	1 1 1 1 0 0 0 0 0	
Restore A	0 0 0 0 0 0 1 0 0 0	1 1 1 1 0 0 0 0 0	5
Shift left	0 0 0 0 0 1 0 0 0 1	1 1 1 0 0 0 0 0 □	
A=A-M	1 1 1 1 1 0 1 0 0 1		
	1 1 1 1 1 1 1 1 0 1 0	1 1 1 0 0 0 0 0 0	
Restore A	0 0 0 0 0 1 0 0 0 1	1 1 1 0 0 0 0 0 0	4
Shift left	0 0 0 0 1 0 0 0 1 1	1 1 0 0 0 0 0 0 □	
A=A-M	1 1 1 1 1 0 1 0 0 1		
	0 0 0 0 0 1 1 0 0	1 1 0 0 0 0 0 1	3
Shift left	0 0 0 0 0 1 1 0 0 1	1 0 0 0 0 0 1 1 □	
A=A-M	1 1 1 1 1 0 1 0 0 1		
	0 0 0 0 0 0 0 1 0	1 0 0 0 0 0 1 1	2
Shift left	0 0 0 0 0 0 0 1 0 1	0 0 0 0 0 0 1 1 □	
A=A-M	1 1 1 1 1 0 1 0 0 1		
	1 1 1 1 1 0 1 1 1 0	0 0 0 0 0 0 1 1 0	
Restore A	0 0 0 0 0 0 0 1 0 1	0 0 0 0 0 1 1 0	1
Shift left	0 0 0 0 0 1 0 1 0	0 0 0 0 1 1 0 □	
A=A-M	1 1 1 1 1 0 1 0 0 1		
	1 1 1 1 1 1 0 0 1 1	0 0 0 0 1 1 0 0	
Restore A	0 0 0 0 0 1 0 1 0	0 0 0 0 1 1 0 0	0

Quotient = 00001100 = 1100

Remainder = 0000001010 = 1010

19M1D0031

Non Restoring

	A	Q	n
	0000000000	1000111110	9
Shift left	0000000001	0001111100	
A = A - M	1111101001		
	1111101010	0001111100	8
Shift left	1111010100	0011111000	
A = A + M	0000010111		
	1111101011	0011111000	7
Shift left	1111010110	0111110000	
A = A + M	0000010111		
	1111101101	0111110000	6
Shift left	1111011010	1111100000	
A = A + M	0000010111		
	1111110001	1111100000	5
Shift left	1111100011	1111000000	
A = A + M	0000010111		
	1111111010	1111000000	4
Shift left	1111110101	1110000000	
A = A + M	0000010111		
	0000001100	1110000001	3
Shift left	0000011001	1100000010	
A = A - M	1111101001		
	0000000010	1000000011	2

19MID0031

ShiftLeft
A=A-M

```
0 0 0 0 0 0 0 0 1 0 1        0 0 0 0 0 0 1 1
1 1 1 1 1 0 1 0 0 1
-----------------
1 1 1 1 1 0 1 1 1 0        0 0 0 0 0 0 1 1 0        1
```

ShiftLeft
A=A+M

```
1 1 1 1 0 1 1 1 0 0        0 0 0 0 0 1 1 0
0 0 0 0 0 1 0 1 1 1
-----------------
1 1 1 1 1 1 0 0 1 1        0 0 0 0 0 1 1 0 0        0
```

A=A+M

```
1 1 1 1 1 1 0 0 1 1
0 0 0 0 0 1 0 1 1 1
-----------------
0 0 0 0 0 0 1 0 1 0        0 0 0 0 0 1 1 0 0
```

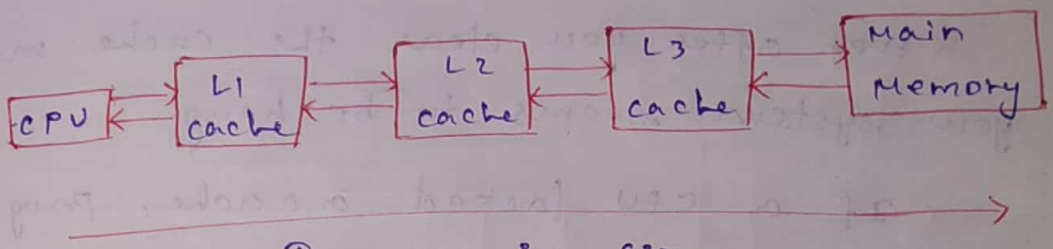
Quotient = 000001100 = 1100

Remainder = 00000 01010 = 1010

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 3 levels in cache memory

Level 1 (L1) cache memory
Level 2 (L2) cache memory
Level 3 (L3) cache memory



L1 is more near to the CPU register. If the data is present in the L1 cache memory, its performance will be fast and it is small size compound to the remaining 2 levels of cache memory.

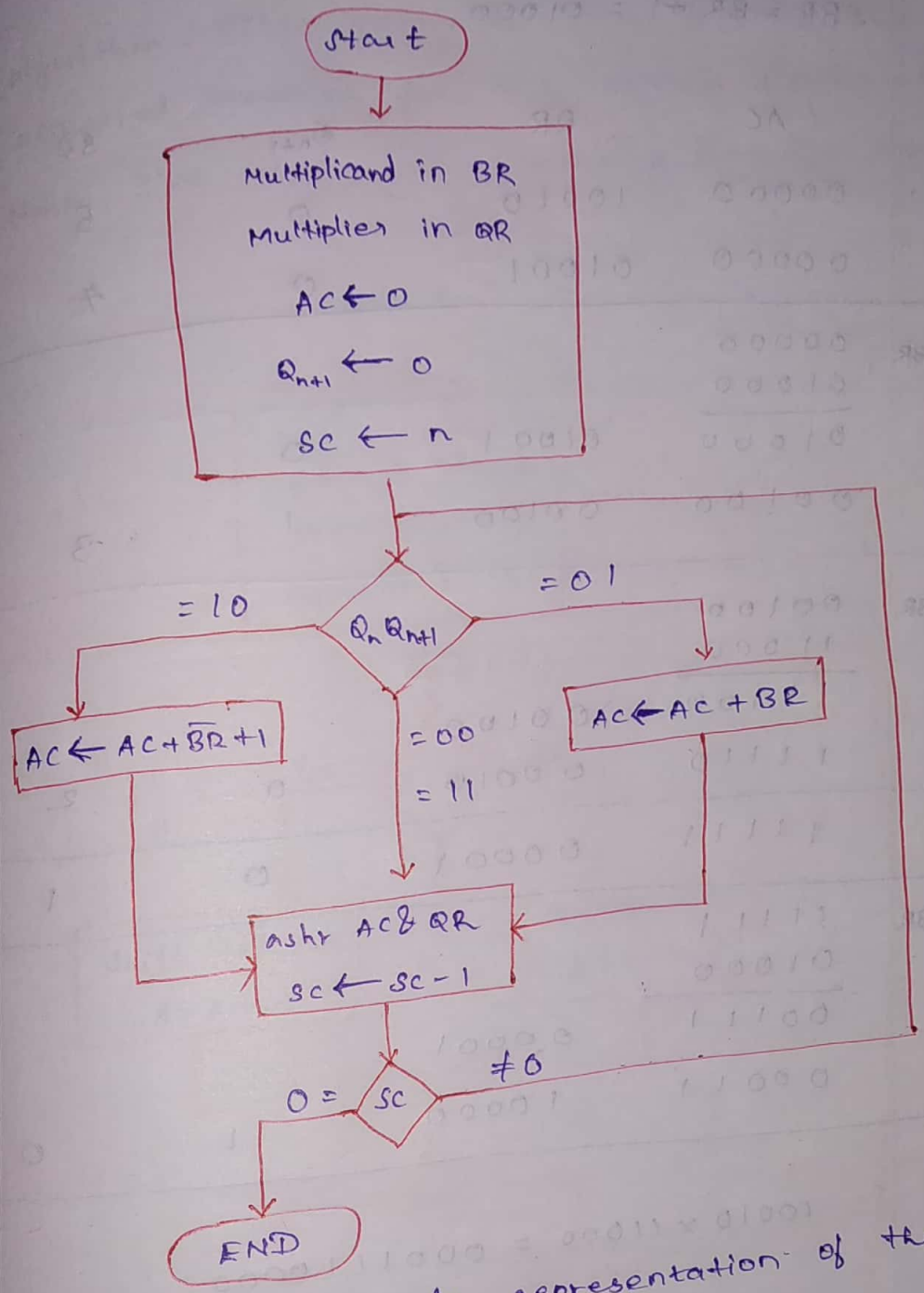
L2 is in the middle of L1 and L3. If no data is present in the L2 cache memory, then the performance will be little fast not slow and it is bigger than L1 cache memory.

L3 is in the nearest to the main memory and it is the biggest cache memory. If the given data is present in L3 cache memory, 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 lacked a cache, programs 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) Discuss the algorithm for multiplication of -8 x -14, how the sign magnitude is handled?



The signed magnitude representation of the binary number must have either 0 or 1. For the positive binary number, the MSB is 0. For the negative binary number, the MSB is 1. Other wise, we take 2's complement for negative numbers with sign bit, so that we can get signed binary number for negative answers.

QR = Multiplier = -14 = 10010

BR = Multiplicand = -8 = 11000

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

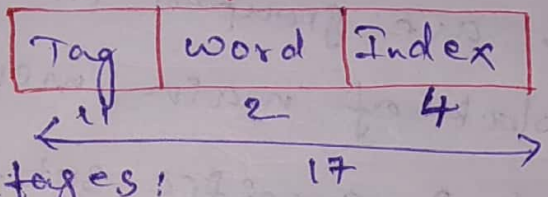
	AC	QR	Q_{n+1}	SC
A = 0	00000	10010	0	5
Ashr	00000	01001	0	4
AC = AC-BR	00000 01000 01000	01001	0	
Ashr	00100	00100	1	3
AC = AC+BR	00100 11000 11100	00100		
Ashr	11110	00010	0	2
Ashr	11111	00001	0	1
AC = AC-BR	11111 01000 00111	00001		
Ashr	00011	10000	1	0

10010 x 11000 = 0001110000

-8 x -14 = 112

6) Following requirements are given, cache memory - 64 KB and Main Memory - 128 KB. Frame - 4. Discuss the advantage and Disadvantage over different mapping procedures with the above given requirements

Direct Mapping:



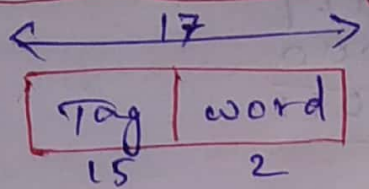
Advantages:

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 will have an situation to replace the cache memory block even when other blocks in the cache memory are empty.

Associative Mapping:



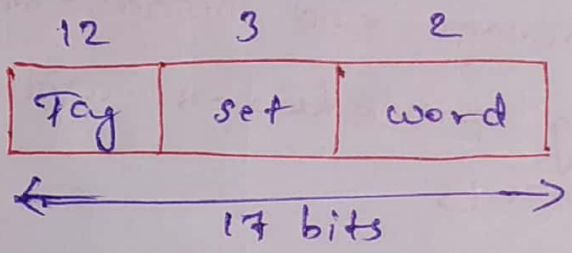
Advantage:

0% of high conflict miss occurs

Disadvantage:

Tag comparison = No. of blocks in cache.
i.e) very high tag comparisons

Set Associative Mapping

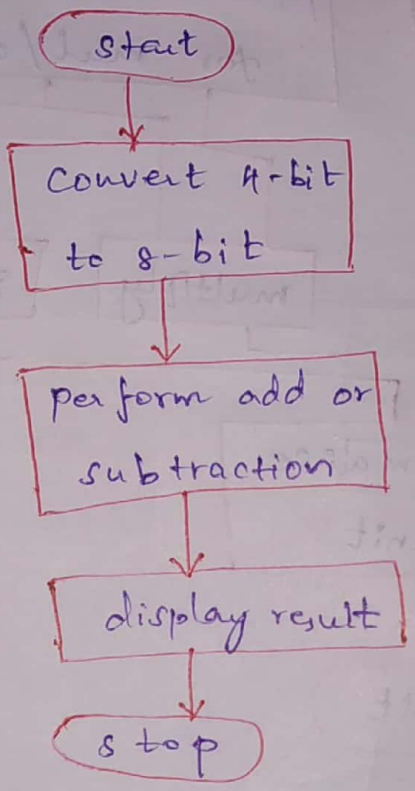


Advantage

cache blocks are grouped into sets and mapping allow block of main memory reside into any block of a specific set. Hence contention problem of direct mapping is eased.

7) Register A holds 8 bit number (1110 0111), register B holds 4 bit number (1111), how can you develop algorithm in your design?

we need to have same number of bits to add or subtract. Therefore, we need to convert 4-bit number to 8-bit number.



initialize result R=0

1st number : 1110 0111 (231)

2nd number : 1111 (15)

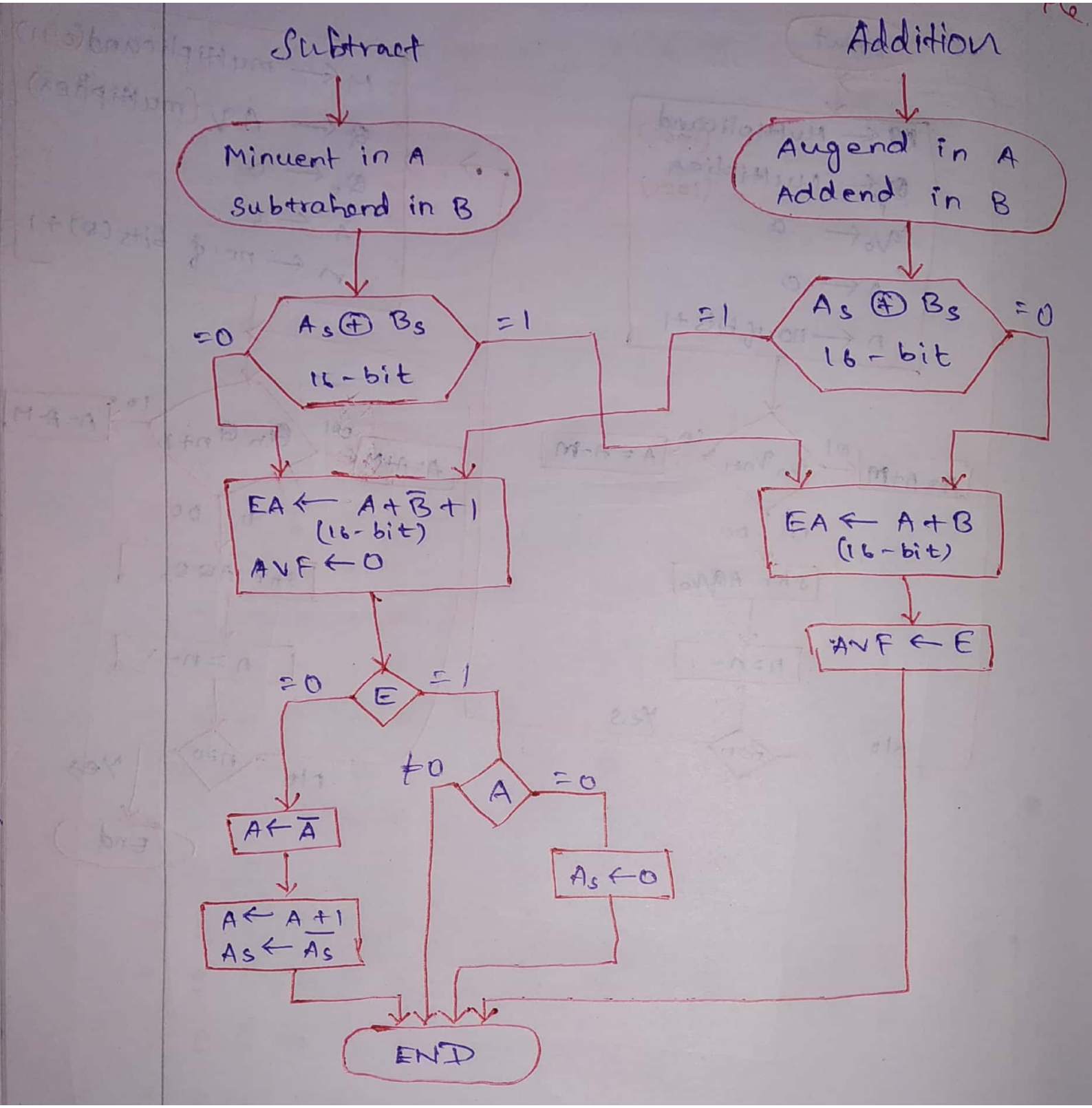
converting to 8-bit number

2nd number : 0000 1111

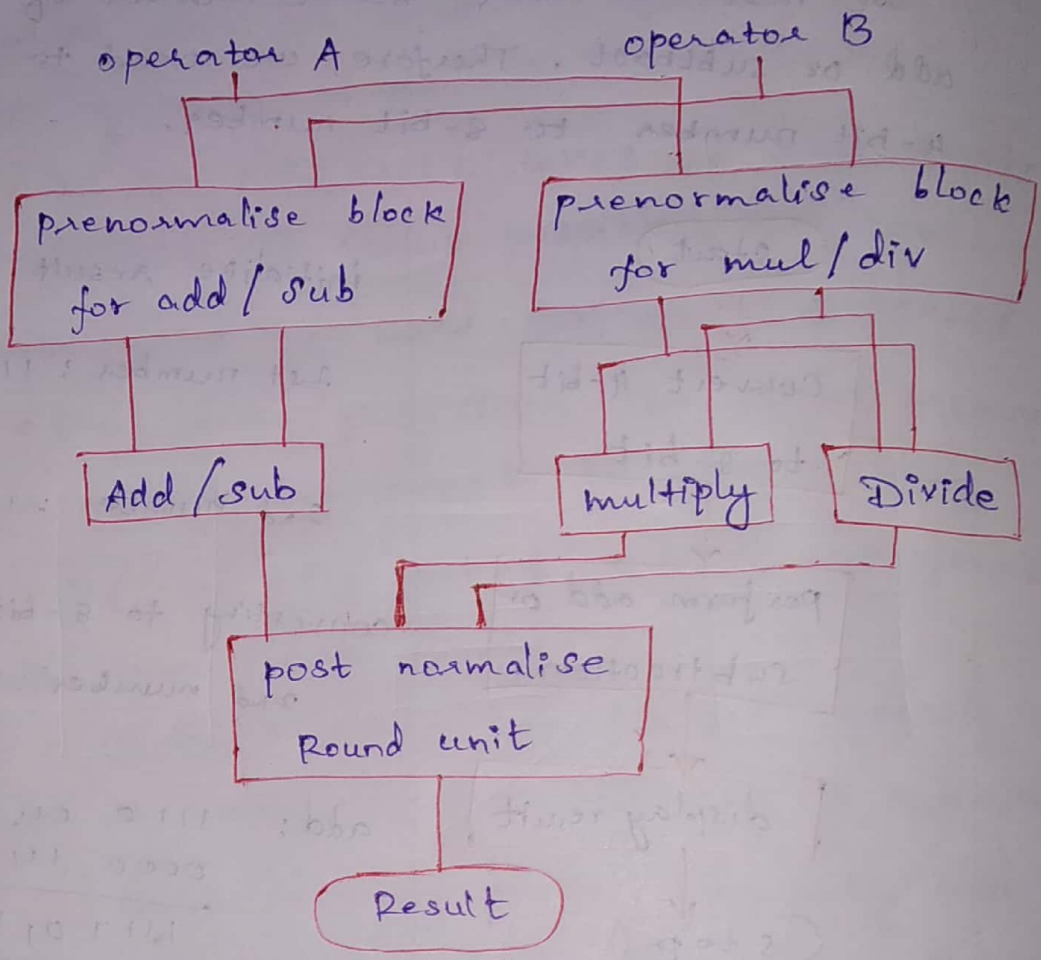
add: 1110 0111
0000 1111
1111 0110 (246)

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 the addition, subtraction, multiplication, division.



Here we can perform addition, subtraction, multiplication, division as well is same hardware.

Pre-normalise blocks helps to understand whether the operation is add / sub or multiplication / Division.

9) Main memory with the size of 16 KB and virtual memory 64 KB and execution of single program needs 1 GB of reference string. How can you resolve this issue?

Main memory = 16 KB

$= 2^4 \times 2^{10} = 2^{14}$ bytes = 14 bits

Virtual memory = 64 KB

$= 2^6 \times 2^{10} = 2^{16}$ bytes = 16 bits.

Our need = 1 GB = 2^{30} bytes = 30 bits

⇒ This problem can be resolved using the concept of virtual memory.

⇒ virtual memory gives an illusion that system has a very large memory even though computer actually has relatively small memory.

⇒ As our main memory size is 2^{14} Bytes, we can store 2^{14} B from 2^{30} B need, rest of the memory uses virtual memory concept

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

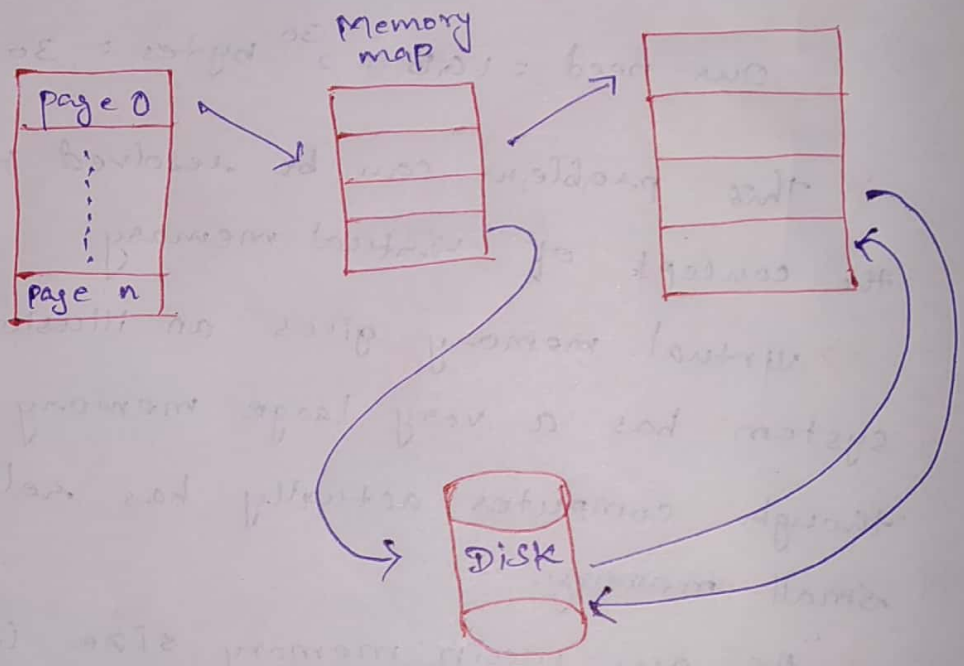
⇒ we use 3 kind of page replacement algorithms

* FIFO
* LRU
* optimal

⇒ If the required page is already present in Main memory, then it is called a hit.

⇒ If the page is absent in Main memory, then it is called miss or page fault.

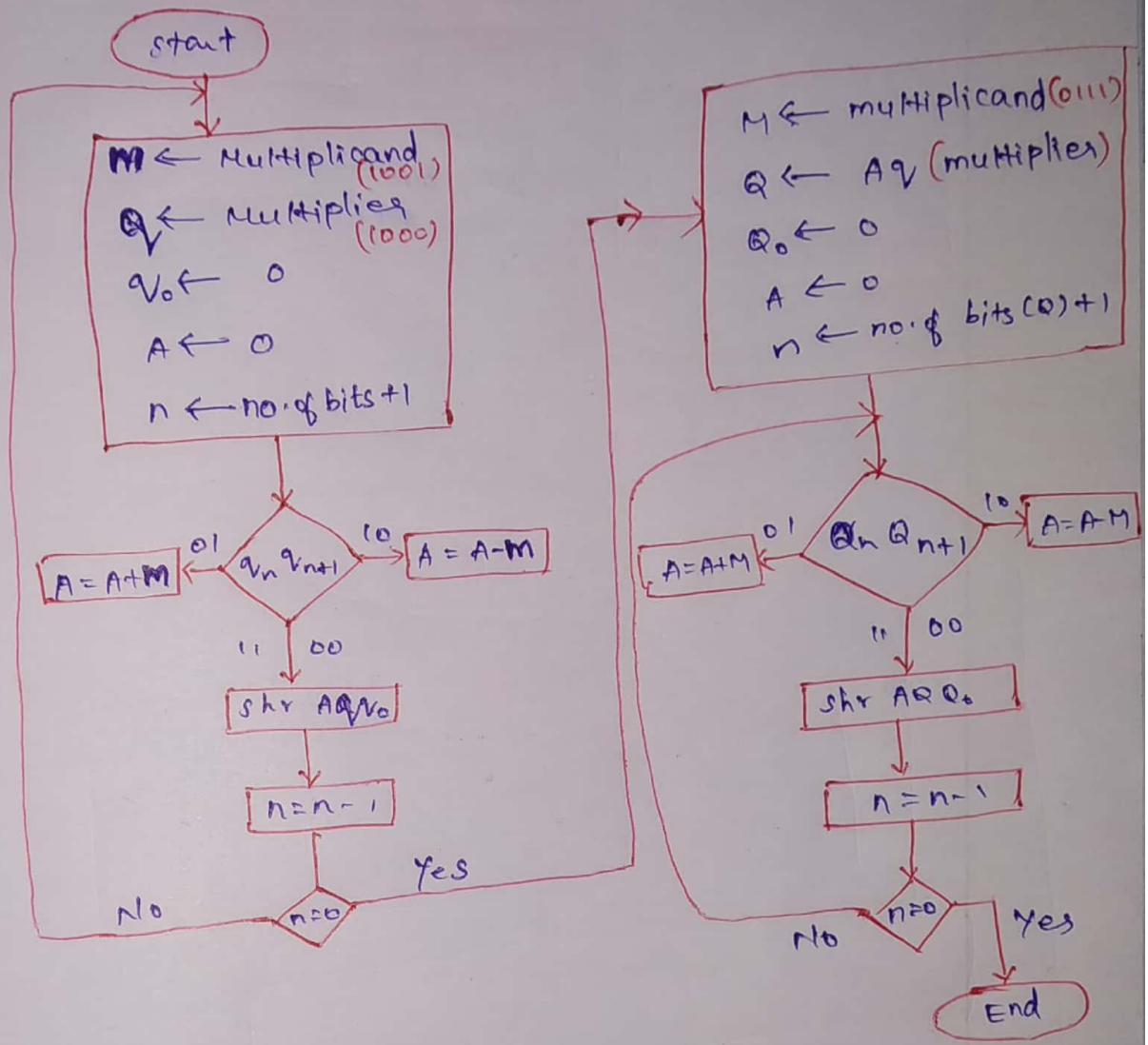
⇒ virtual memory communicates with main memory through memory map.



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

10) What is the modification to be done on booth multiplication algorithm when 3 numbers given multiplication: (1000 x1001 x0111)

Algorithm :



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

The replacement algorithm which is not used in cache memory is "optimal Algorithm". Because it is futuristic approach of page replacement which is just theoretical. The future pages approaching could not be determined and hence it is almost impossible to implement.

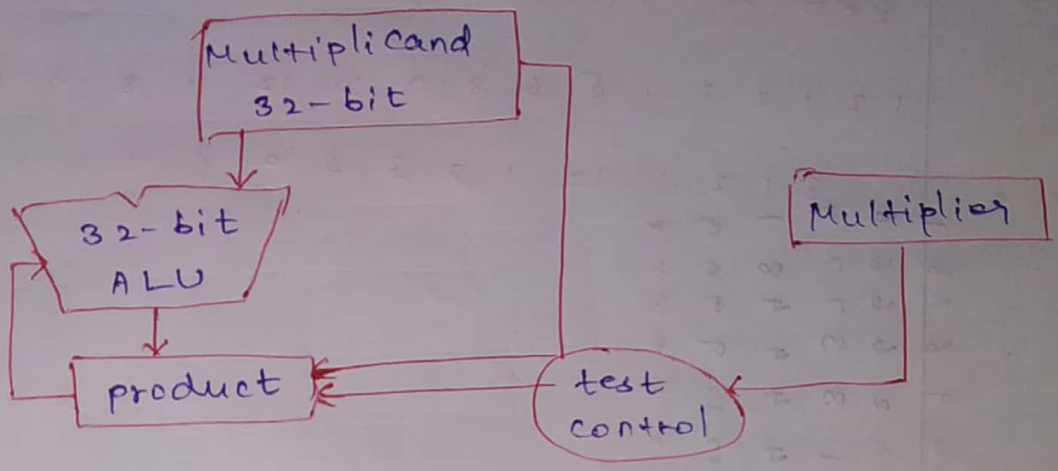
Other mechanisms

FIFO: This page replacement algorithm replaces the firstly entered page among the presently existing page.

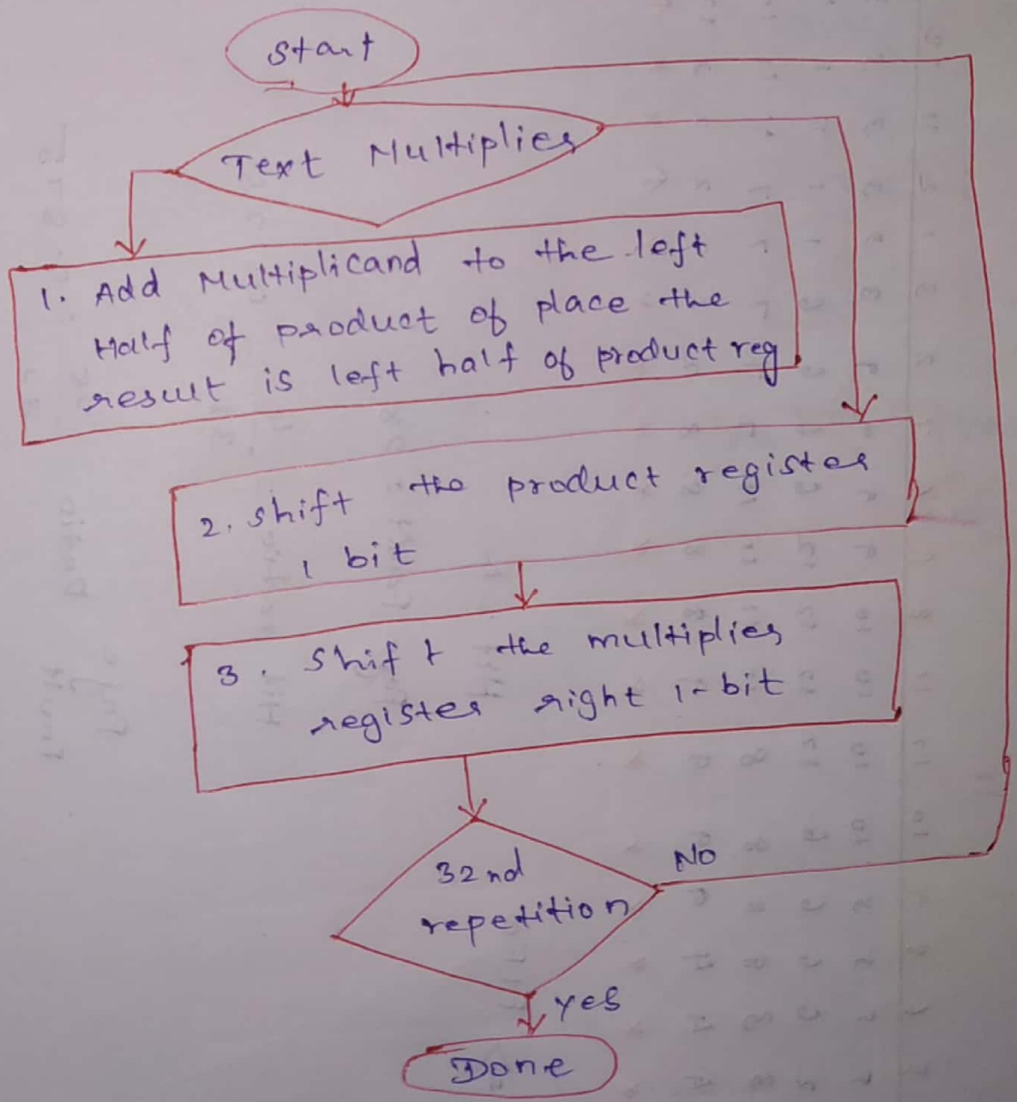
LRU: This page replacement algorithm replaces the least recently used page if it is not in cache memory.

12) Design a Hardware and Algorithm for multiplying 32-bit x 32-bit

Hardware



Algorithm :



13) Compare and contrast the application for restoring and non-restoring division method for 15/2.

Q = Dividend = 15 = 1111

B = Divisor = 2 = 00010

$-B = 2's(B) = 2's(00010) = (11101 + 1) = 11110$

RESTORING

	A	Q	n
	00000	1111	4
A=0			
Shift left	00001	111□	
	11110		
A=A-B	11111	1110	
	00010		
A=A+B restore A	00001	1110	3
Shift left	00011	110□	
	11110		
A=A-B	00001	1101	2
Shift left	00011	101□	
	11110		
A=A-B	00001	1011	1
Shift left	00111	011□	
	11110		
A=A-B	00001	0111	0

Quotient = $(0111)_2$ = 7

Remainder = $(00001)_2$ = 1

NON-RESTORING:

	A	Q	n
			4
A = 0	00000	1111	
Shift left	00001	111□	
A = A − M	11110		
$Q_0 = 0$	11111	1110	3
Shift left	11111	110□	
A = A + M	00010		
$Q_0 = 1$	00001	1101	2
Shift left	00011	101□	
A = A − M	11110		
$Q_0 = 1$	00001	1011	1
Shift left	00011	011□	
A = A − M	11110		
	00001	0111	0

Quotient = $(0111)_2 = 7$

Remainder = $(00001)_2 = 1$

In Restoring Method, we add divisor when we set Q_0 as 0.

In Non Restoring method, we do not add divisor if we set Q_0 as either 0 or 1 and reduce the count and proceed the algorithm.

14) Differentiate the multiplication algorithm and booth's multiplication algorithm with an example of 30x16.

BOOTH'S ALGORITHM

Multiplicand = BR = 30 = 11110 = 011110

Multiplier = QR = 16 = 10000 = 010000

$\overline{BR}+1$ = (100001 + 1) = 100010 = −BR

	AC	QR	Q_{n+1}	SC
				6
A=0	000000	010000	0	
Ashr	000000	001000	0	5
Ashr	000000	000100	0	4
Ashr	000000	000010	0	3
Ashr	000000	000001	0	2
AC=AC-BR	100010			
	100010	000001	0	
Ashr	110001	000000	1	1
AC=AC+BR	011110		1	
	001111	000000		
Ashr	000111	100000	0	0

$(000111100000)_2 = (480)_{10}$

30x16 = 480

B = 11110 = 30
Q = 10000 = 16

MULTIPLICATION ALGORITHM :

	E	A	Q	SC
	0	00000	10000	5
Ashr EAQ	0	00000	01000	4
Ashr EAQ	0	00000	00100	3
Ashr EAQ	0	00000	00010	2
Ashr EAQ	0	00000	00001	1
EA = A+B	0	11110	00001	1
Ashr EAQ	0	01111	00000	0

$(01111000000)_2 = (480)_{10}$

$30 \times 16 = 480$

Difference :

* In Multiplication algorithm the sign bit is calculated separately in beginning.

In Booth's algorithm, the sign bit is automatically calculated in the algorithm

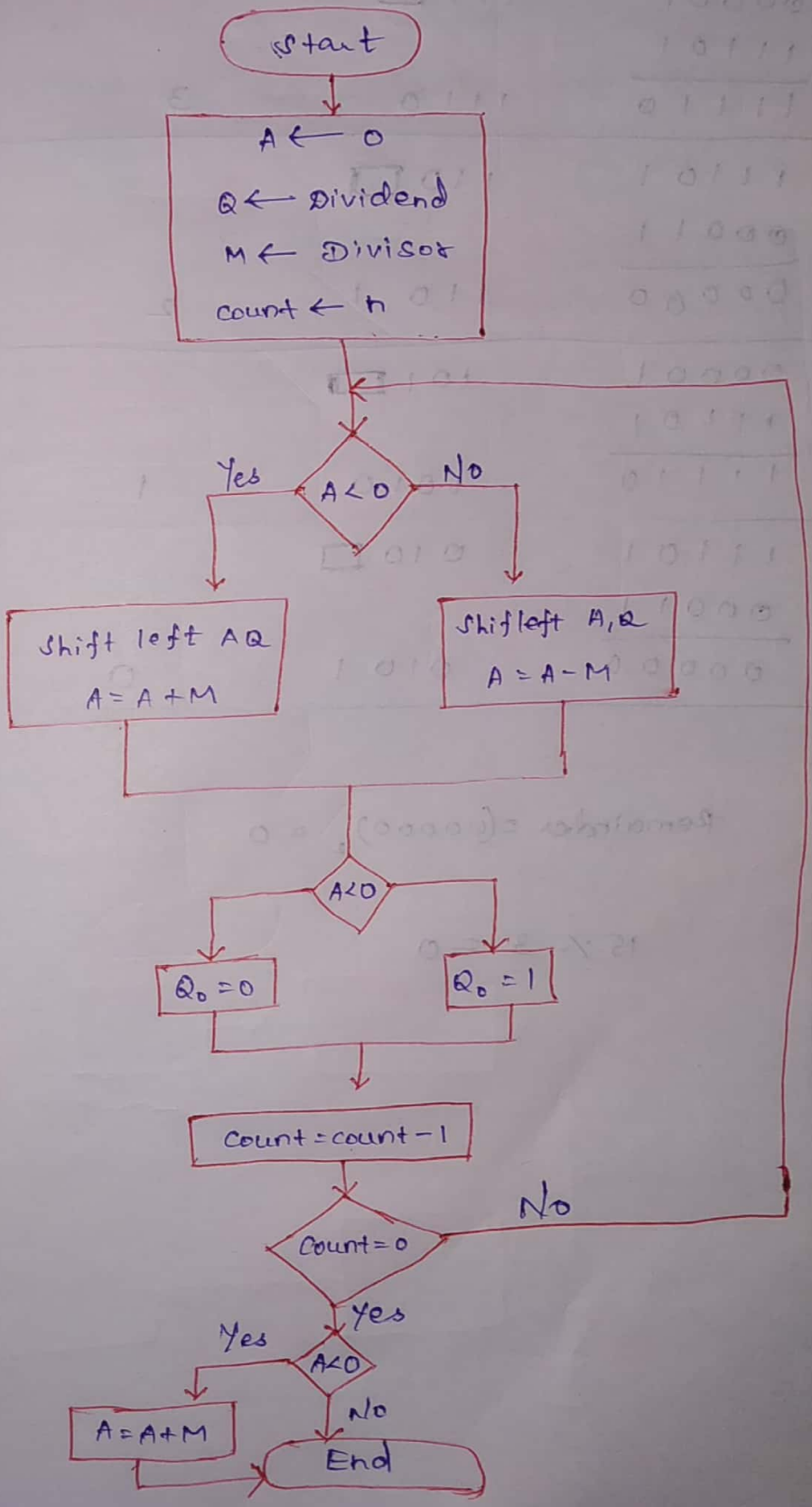
* In Multiplication algorithm, the Q_n bit signifies the next step to be adopted.

In Booth's algorithm, the $Q_n Q_{n+1}$ bits signifies the next step to be adopted.

15) Device an algorithm for performing 15 % 3

Modulo operates uses any of the division algorithm. The reminder obtained is the required answer for the modulo operator.

Here I've used Non-Restoring Division Algorithm.



Dividend = Q = 15 = 1111

Divisor = M = 3 = 00011

$-M = \bar{M} + 1 = 11101$

	A	Q	count
	00000	1111	4
Shift left	00001	111□	
A = A - M	11101		
$Q_0 = 0$	11110	1110	3
Shift left	11101	110□	
A = A + M	00011		
$Q_0 = 1$	00000	1101	2
shift left	00001	101□	
A = A - M	11101		
$Q_0 = 0$	11110	1010	1
shift left	11101	010□	
A = A + M	00011		
	00000	0101	0

Remainder = $(00000)_2 = 0$

15 % 3 = 0

16) cache memory with size of 4 KB (4 lines) and the virtual memory with the size of 8 KB (Lines). How the execution is carried out?

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

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

Algorithm used => FIFO

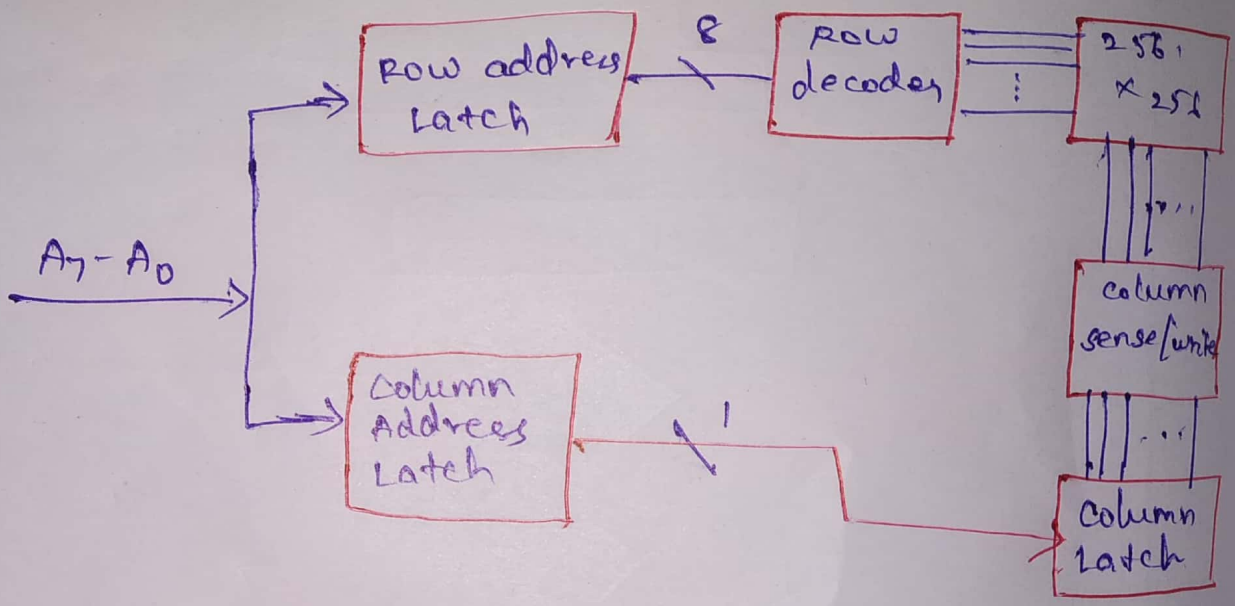
Hit = 4

Page Fault = 28

Hit ratio = $\frac{4}{32}$ = 0.125

Page Fault Ratio = $\frac{28}{32}$ = 0.875

17. Design Hardware for storing a 6-bit no in the dynamic RAM 6 bit $\Rightarrow 2^6 = 64K \times 1$



18) Design a DRAM for 128 x 16