Assignment III

PAGE NO.

1) Perform following operations on 5 bit signed no.s using 2's complement representation system. Also indicate whether overflow has occured.

a) (-9) + (-7)

(-9) → 01001 → 1's complement 10110

+1 2's complement

10111

(-7) → 00111 → 1's complement 11000

+1 2's complement

11001

Add these two

10111

11001

11000

ignore MSB carry bit

-16 i.e. 10000

b) (+7) - (-8)

(+7) → 00111 →

(-8) → 01000 → 1's complement 10111 2's complement

+1

11000

Add these two

11001	00111
11000	11000
110001	11111
	15

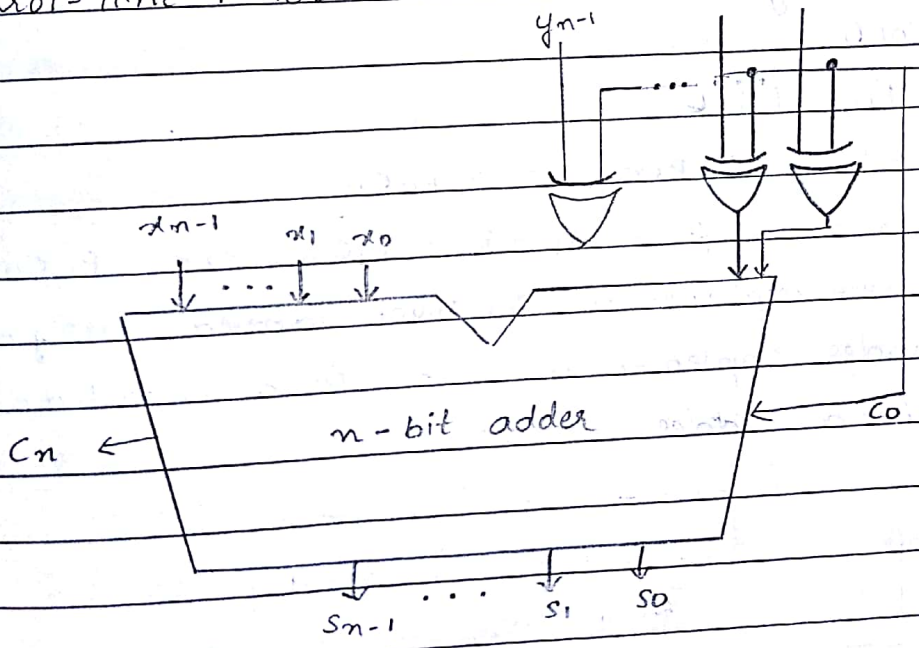
But MSB is 1 which is -ve

Hence overflow occurs.

KSIT

2) Design a logic circuit to perform addition/subtraction of n bit no. x and y

- n- bit adder can be used to add 2's complement no.s x and y
- Overflow can only occur when signs of 2 operands are same
- In order to perform the subtraction operation x-y on 2's complement no.s x and y, we form 2's complement of y and add it to x
- Addition or subtraction operation is done based on value applied to Add/Sub input control line
- Control-line = 0 for addition, applying the y vector unchanged to one of the adder inputs
  Control-line = 1 for subtraction, the y vector is 2's complemented.



3) Draw 4 bit carry lookahead adder and explain. (or)
Explain fast adder
The logic expression for si(sum) and ci+1 (carry-out) of stage i are

$s_i = x_i + y_i + c_i$

$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$

$c_{i+1} = x_i y_i + (x_i + y_i) c_i$

$c_{i+1} = G_i + P_i c_i$ $\quad G_i = x_i y_i \quad P_i = x_i + y_i$

- The expressions of G_i and P_i are called generate and propagate functions
- If $G_i = 1$ then $C_{i+1} = 1$ independent of input carry C_i. This occurs when both x_i and y_i are 1. Propagate function means that an input carry will produce an output-carry when either $x_i = 1$ or $y_i = 1$
- All G_i and P_i functions can be performed independently and in parallel in one logic delay.
- Expanding c_i terms of $i-1$ and substituting

$$C_{i+1} = G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} \cdots + P_1 G_0 + P_i P_{i-1} \cdots P_0 C_0$$

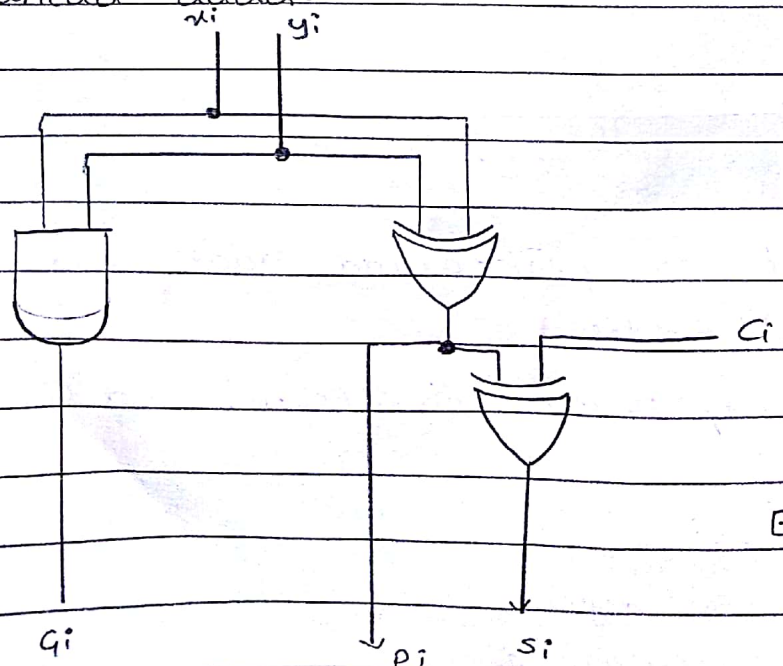
- Delay through the adder is 3 gate delays for carry-bits and 4 gate delays for sum bits.
- $C_1 = G_0 + P_0 C_0$

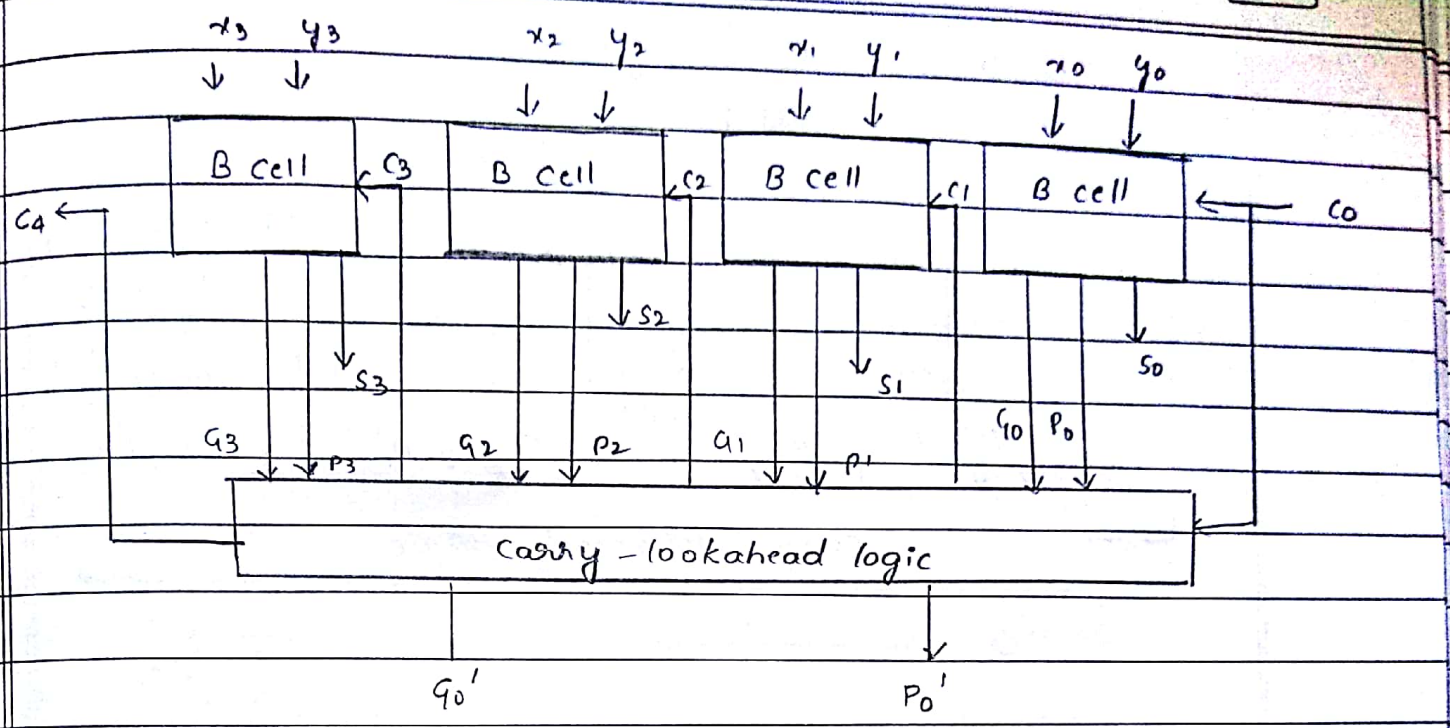
$$C_2 = G_1 + P_1 G_0 + P_1 P_0 C_0$$

$$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$

$$C_4 = G_3 + G_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$$

- The carries are implemented in block labeled carry-lookahead logic. An adder implemented in this form is called a carry-lookahead adder.





4) Explain with figure the design and working of a 16 bit carry-look-ahead adder built using 4 bit adder

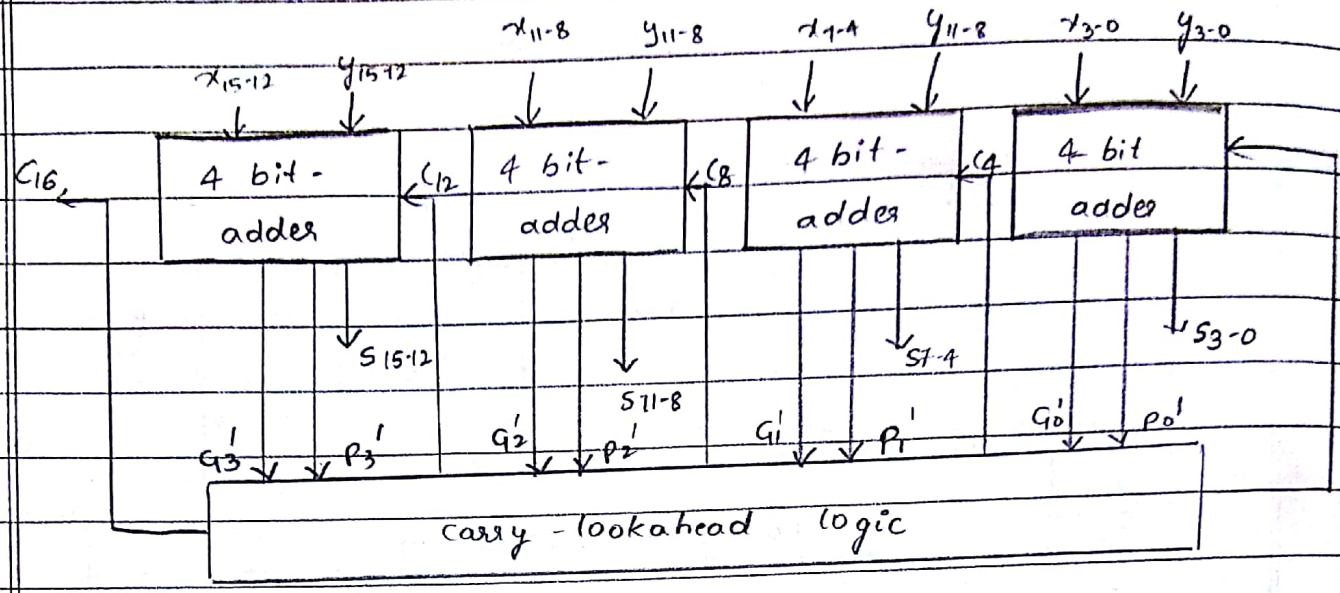
- 16 bit adder can be built from 4-bit adder blocks
- These blocks provide new output functions defined as G_k and P_k where $k=0$ for first 4-bit block $k=1$ for second 4 bit block and so on
- In first block $P_0 = P_3 P_2 P_1 P_0$

$$G_0 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0$$

- The first level G_i and P_i functions determine whether bit stage i generates or propagates a carry, and the second level G_k and P_k functions determine whether block-k generates or propagates a carry.
- Carry C_{16} is performed by one of the carry-lookahead circuits as

$$C_{16} = G_3 + P_3 P_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$$

- All carries are available 5 gate delays after X, Y and C_0 are applied as inputs.

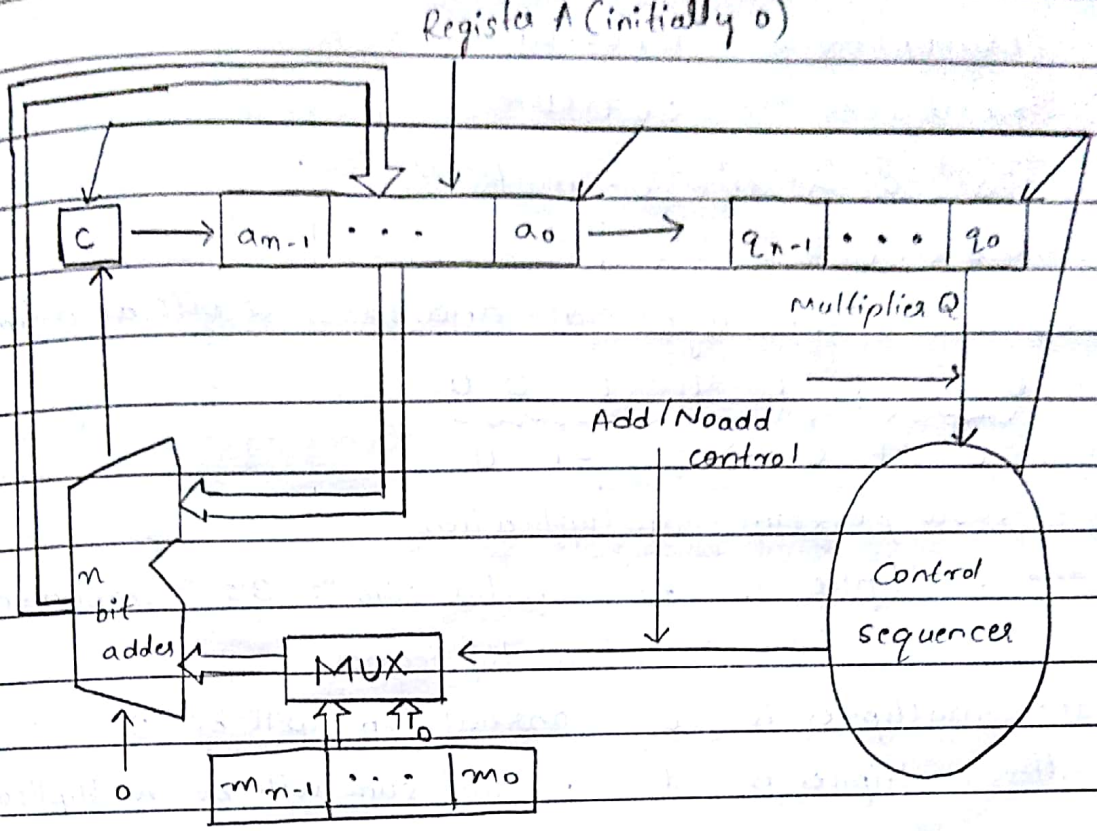


5) Explain Sequential circuit for binary multiplication.

- Registers A and Q combined hold partial product PPi while the multiplier bit q_i generates the signal Add / No add
- The carry-out from the adder is stored in flip-flop C

Procedure for multiplication.

1) Multiplier is loaded into register Q
Multiplicand is loaded into register M
C and A are cleared to 0
2) if $q_0 = 1$ add M to A and store sum in A
Then C, A and Q are shifted right one bit position
if $q_0 = 0$ no addition performed and
C, A and Q are shifted right one bit-position.
3) After n-cycles, the high order half of the product is held in register A and low-order half is held in register Q.



6) Explain Booth Algorithm for binary multiplication. State its advantages and disadvantages.

→ This algorithm generates a 2n-bit product

→ treats both positive and negative 2's complement n-bit operands uniformly.

→ This algorithm suggests that we can reduce the no. of operations required for multiplication by representing multiplier as a difference betn 2 no.s

→ This algorithm achieves some efficiency in no. of addition required when multiplier has a few large blocks of 1s

Multiplier		Version of multiplicand selected
Bit i	Bit i-1	by bit i
0	0	0 × M
0	1	+1 × M
1	0	-1 × M
1	1	0 × M

Eg: Multiplicand 0101101

Multiplier 0011110

Step 1: Put 0 at end of multiplier

00111100

Step 2: Start from right side and take 2 bits at a time

0 0 1 1 1 1 0 0

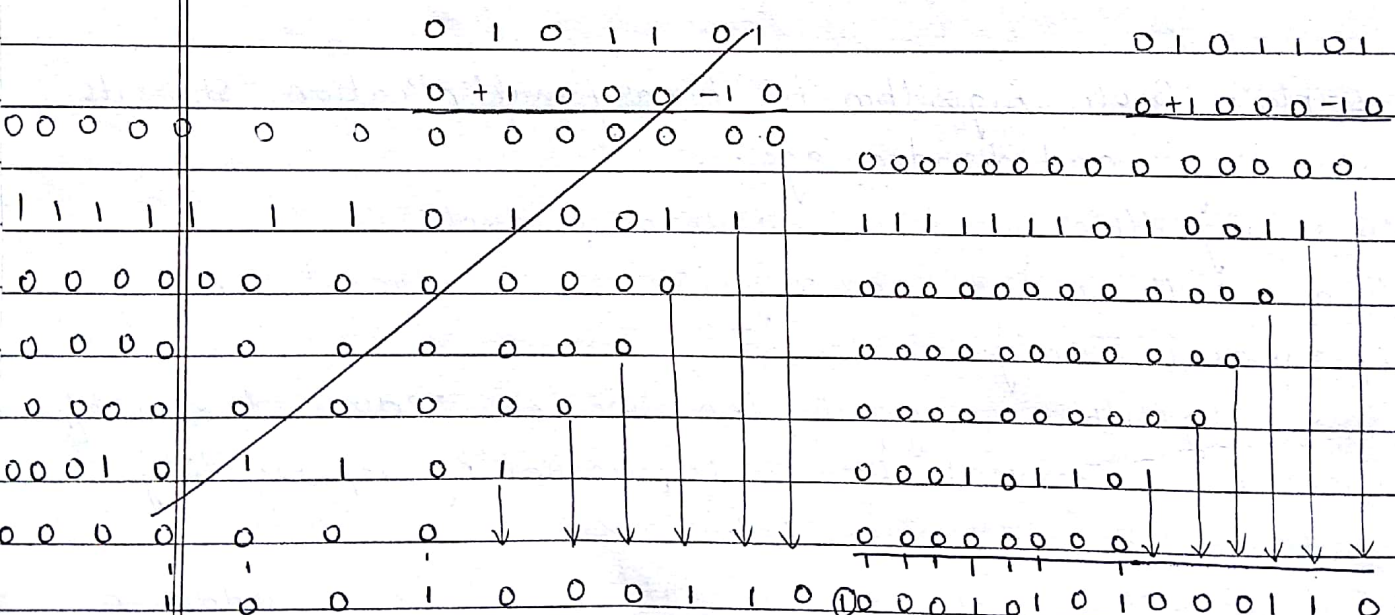
0 +1 0 0 0 −1 0

Step 3: Now perform multiplication

if the multiplier is −1 partial sum is 2's complement of multiplicand

if the multiplier is 0 partial sum will be 0

if the multiplier is 1 partial sum will be multiplicand



MSB 1 it is negative and take 2's complement.

7) Explain Fast Multiplication Method Bit pair recoding of multipliers.

It is derived from Booth algorithm

reduces no. of summands by a factor of 2

1. Group the Booth-recoded multiplier bits in pairs for bit pair recoding derived from Booth recoding.

- pair (+1, -1) is equivalent to (0,+1), (-1,+1) equivalent to (0,-1)
- (+1,0) is equivalent to (0,+2), (-1,0) equivalent to (0,-2)

(0,0) equivalent to 0

Sign extension

[1] 1 1 0 1 0 [0] ← Implied 0 to right of LSB

0 0 -1 +1 1 0

0 -1 -2

Multiplier bit-pair		Multiplier bit on the right	Multiplicand selected at position
i+1	i	i-1	
0	0	0	0×M
0	0	1	+1×M
0	1	0	+1×M
0	1	1	+2×M
1	0	0	+2×M
1	0	1	-1×M
1	1	0	-1×M
1	1	1	0×M

Eg: 0 1 1 0 1 (+13)

1 1 0 1 0 (-6)

0 1 1 0 1

0 -1 +1 -1 0

0 0 0 0 0 0 0 0 0 0

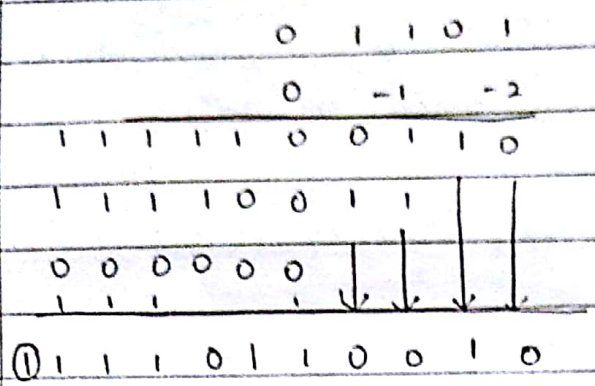
1 1 1 1 1 0 0 1 1

0 0 0 0 1 1 0 1

1 1 1 0 0 1 1

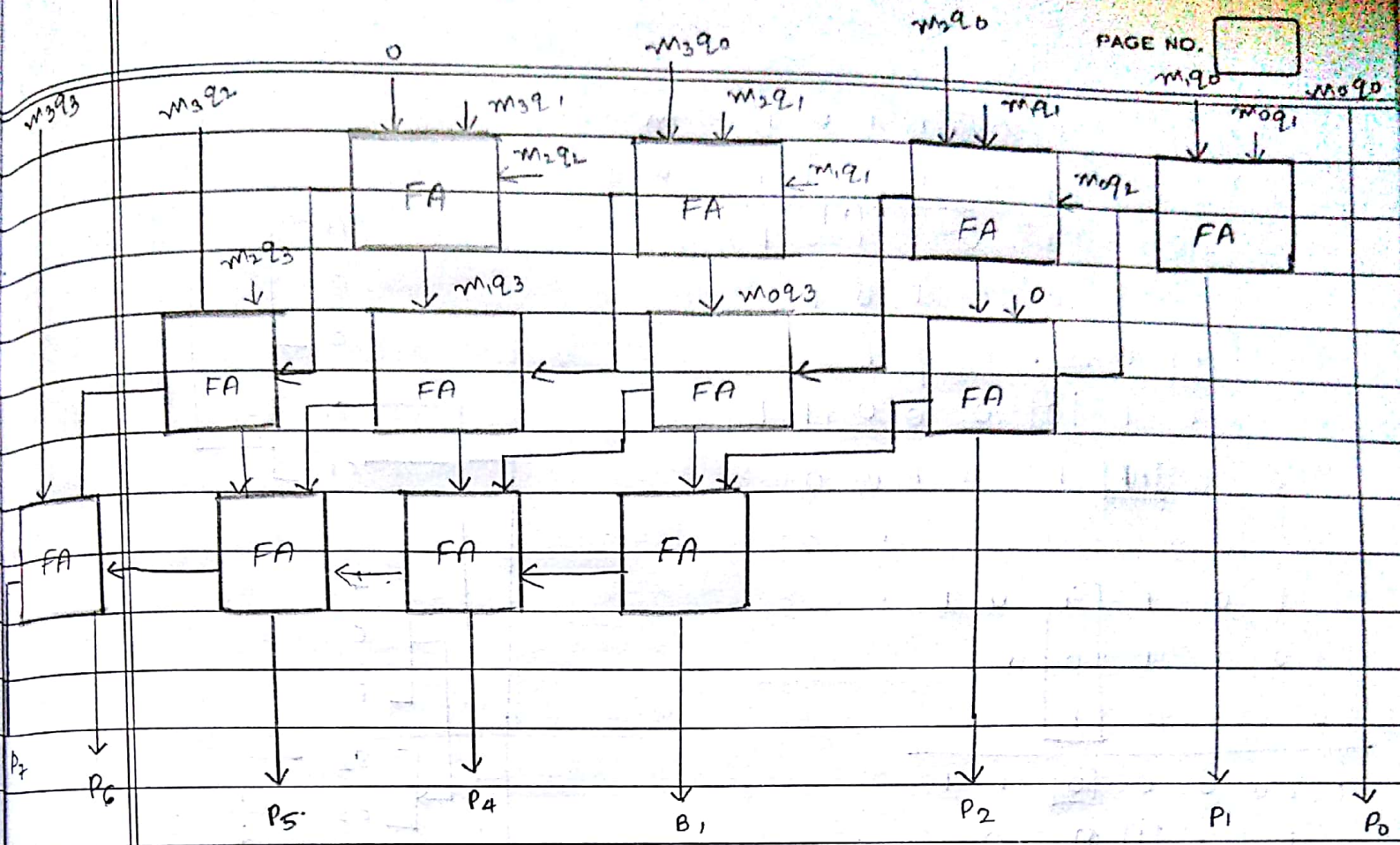
0 0 0 0 0 0

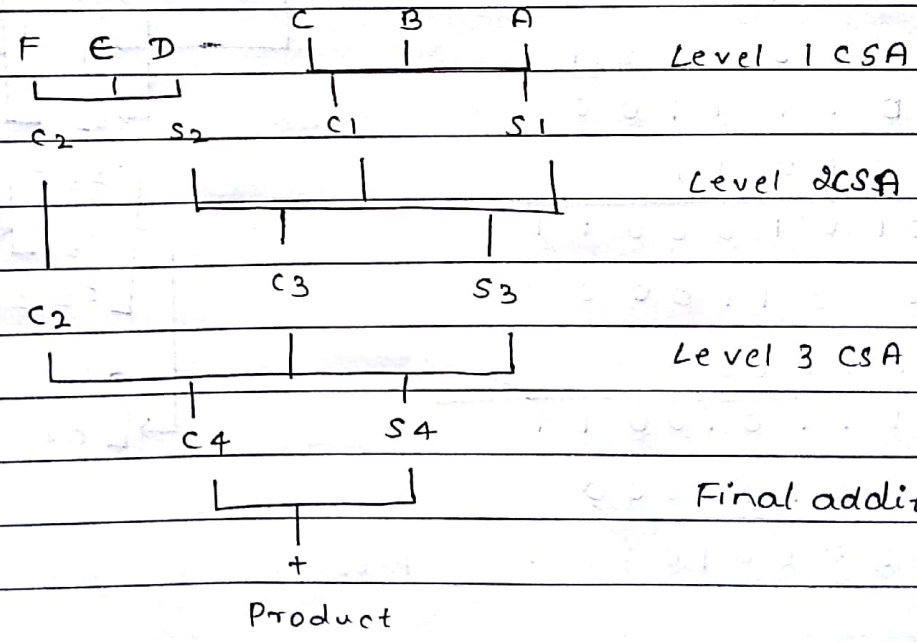
(1) 1 1 0 1 1 0 0 1 0 (-78)



8) Explain the concept of carry save addition for the multiplication operation $M \times Q = P$ for 4 bit operands with diagram and examples.

- Consider array for 4 * 4 multiplication. Needs 29 gate delays.
- Instead of letting the carries ripple along the rows they can be "saved" and introduced into the next row, at correct weighted positions.
- Consides the add'n of many summands.
- Group summands in three's and perform carry-save addition on each of these groups in parallel to generate a set of S and C vectors in one full-addes delay.
- Group all of the S and C vectors into three's and perform carry-save addition on them, generating a further set of S and C vectors in one more full-adder delay.
- Continue with this process until there are only two vectors remaining. They can be added in a Ripple carry Adder or Carry lookahead to produce the desired product
- By comparision the total gate delay in performing multiplication by using $n \times n$ array is $6(n-1)-1 = 17$

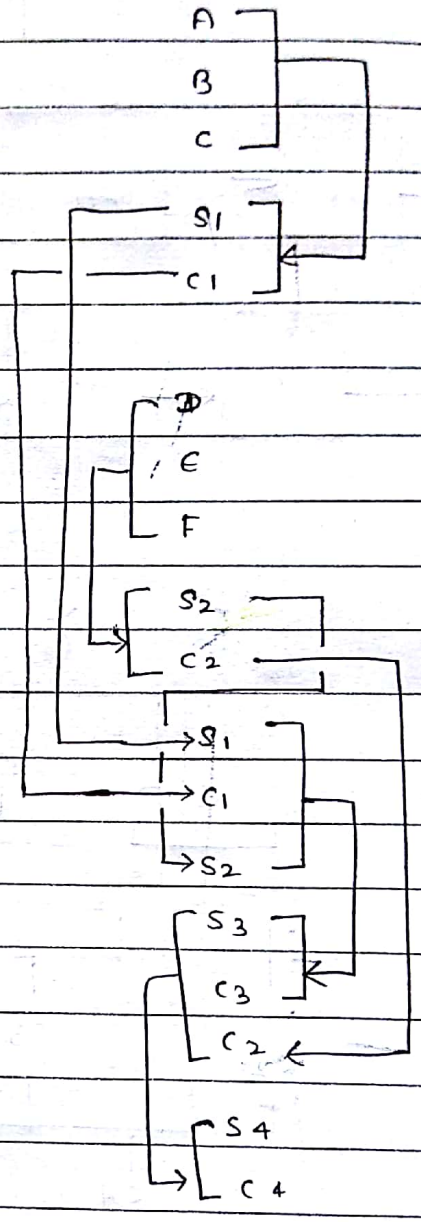




```
      1 0 1 1 0 1      M
    x 1 1 1 1 1 1      Q
      ---------
      1 0 1 1 0 1
    1 0 1 1 0 1
  1 0 1 1 0 1
  ---------------
  1 1 0 0 0 0 1 1
0 0 1 1 1 1 0 0
```

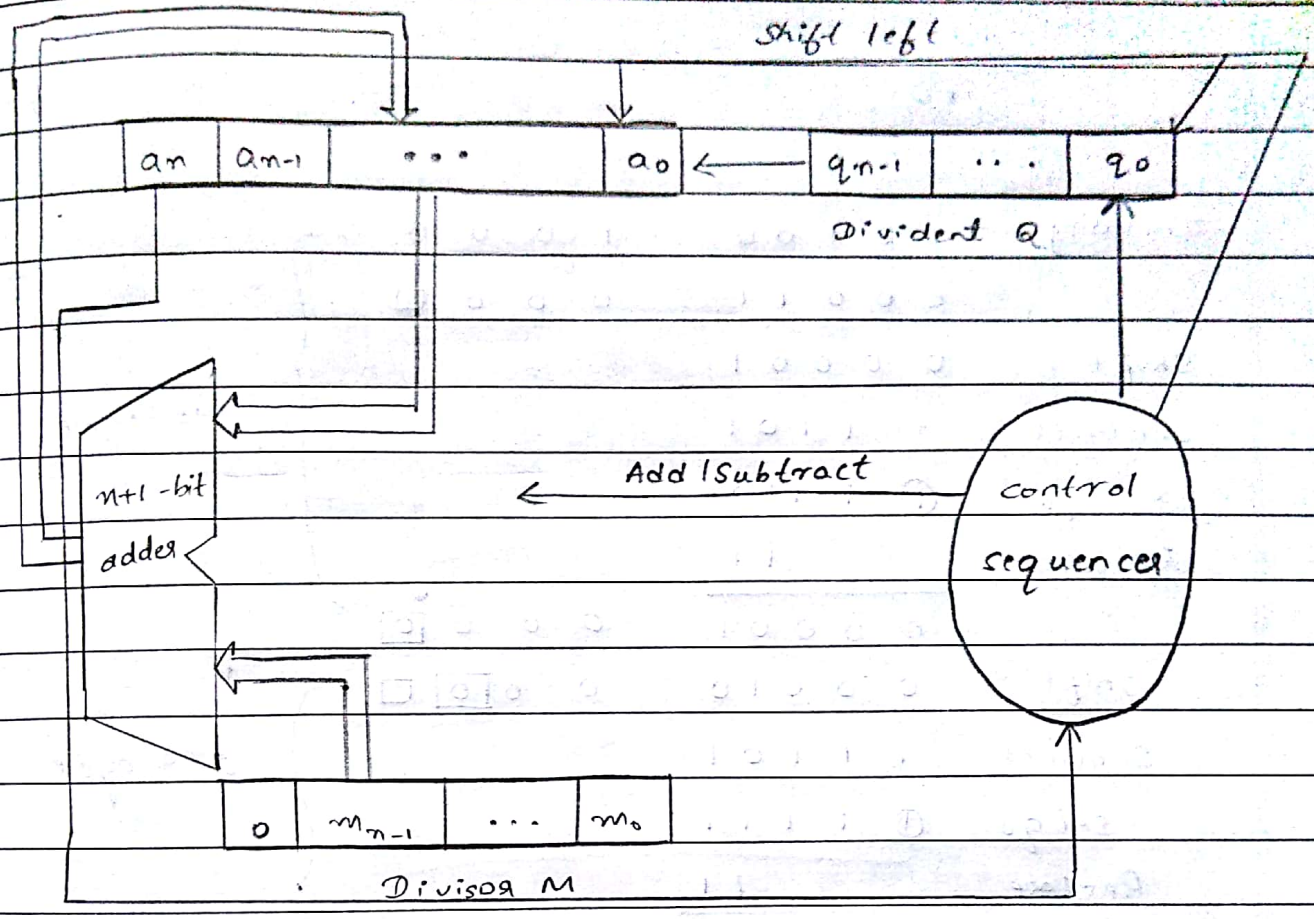
```
    1 0 1 1 0 1
  1 0 1 1 0 1
1 0 1 1 0 1
-----------
1 1 0 0 0 0 1 1
0 0 1 1 1 1 0 0
```

```
      1 1 0 0 0 0 1 1
    0 0 1 1 1 1 0 0
  1 1 0 0 0 0 1 1
  ---------------
  1 1 0 1 0 1 0 0 0 0 1 1
0 0 0 0 1 0 1 1 0 0 0
0 0 1 1 1 1 0 0
-----------------
0 1 0 1 1 1 0 1 0 0 1 1
+ 0 1 0 1 0 1 0 0 0 0 0
-----------------------
1 0 1 1 0 0 0 1 0 0 1 1      Product
```



9) Explain with figure circuit arrangement for binary division

- An n-bit positive divisor is loaded into register M
- An n-bit positive dividend is loaded into register Q at the start of the operation Register A is set to 0
- After division operation, the n-bit quotient is in register Q and the remainder is in register A



10) Give an algorithm and explain with an example both restoring and non restoring methods for integer division.

Restoring division:

Do the following n times

1) Shift A and Q left one binary position
2) Subtract M from A, and place the answer back in A
3) If the sign of A is 1, set q0 to 0 and add M back to A
   If the sign of A is 0, set q0 to 1 and no restoring

```
      10
11 ) 1000
     11
     --
     10
```

Initially	0 0 0 0 0	1 0 0 0
	0 0 0 1 1	0 0 0 □
Shift	0 0 0 0 1	
Subtract	1 1 1 0 1	1st cycle
Set q_0	① 1 1 1 0	
Restore	1 1	
	0 0 0 0 1	0 0 0 [0]
shift	0 0 0 1 0	0 0 [0] □
Subtract	1 1 1 0 1	2nd cycle
set q_0	① 1 1 1 1	
Restore	1 1	
	0 0 0 1 0	0 0 [0] [0]
Shift	0 0 1 0 0	0 [0] [0] □
Subtract	1 1 1 0 1	3rd cycle
set q_0	⓪ 0 0 0 1	
shift	0 0 0 1 0	0 [0] [0] [1]
Subtract	1 1 1 0 1	[0] [0] [1] □
Set q_0	① 1 1 1 1	4th cycle
Restore	1 1	
	0 0 0 1 0	[0] [0] [1] [0]
	Remainder	quotient

Non restoring algorithm:

1) If the sign of A is 0, shift A and Q left one bit position and subtract M from A otherwise shift A and Q left and add M to A
2) Now, if the sign of A is 0, set q_0 to 1 otherwise, set q_0 to 0
3) If sign of A is 1, add M to A

Initially	0 0 0 0 0	1 0 0 0
	0 0 0 1 1	
Shift	0 0 0 0 1	0 0 0 □
Subtract	1 1 1 0 1	
Set q_0	① 1 1 0 0	0 0 0 0
Shift	1 1 1 0 0	0 0 0 □
Add	0 0 0 1 1	
set q_0	① 1 1 1 1	0 0 0 0
Shift	1 1 1 1 0	0 0 0 □
Add	0 0 0 1 1	
set q_0	⓪ 0 0 0 1	0 0 0 1
Shift	0 0 0 1 0	0 0 1 □
Subtract	1 1 1 0 1	
set q_0	① 1 1 1 1	0 0 1 0

Quotient

Add 1 1 1 1 1
0 0 0 1 1
0 0 0 0 0 } Restore remainder

Remainder

11) Explain Excess notation, Normalization and special conditions for exponents.

Excess notation

- Rather than representing an negative exponent in 2's complement form, it turns out to be more beneficial to represent the exponent in excess notation.
- If 7 bits are allocated to exponent exponents can be represented as -64 to +63 i.e :: $-64 <= e <= 63$

Exponent can also be represented using the following coding called as excess-64

$$E' = E_{true} + 6$$

In general excess-p coding

$$E' = E_{true} + p$$

True exponent of -64 is represented as 0

0 is represented as 64

63 is represented as 127

Normalization:

Consider

$0.0011101000(10110) \times 2^8 = 1.1010001010(10) \times 2^5$

Exponent of x was decreased by 1 for every left shift of x

A no. which is brought into a form so that all of the mantissa digits are optimally used is called a normalized no

In base-2 representation this implies that MSB of the mantissa is always 1. We can do away without storing MSB

IEEE notation assumes that all nos are normalized so that the MSB of the mantissa is a '1' and does not store this bit

The values of nos represented in IEEE single precision notation are $(+,-)\,1.M \times 2^{(E-127)}$

The hidden 1 forms integer part of the mantissa.

special conditions

1) When $e' = 0$ then Mantissa $M = 0$ and value exact 0 is represented
2) When $E' = 255$, $M = 0$ value ∞ is represented ∞ is number dividing by 0
3) When $E' = 0$ and $M \neq 0$ denormal nos are represented
4) $E' = 255$, $M \neq 0$ the value is called Not a Number (NaN)
5) $E' < -126$ Underflow
6) $E' > 127$ Overflow

In such cases exceptions such as overflow and underflow divide by zero, inexact and invalid are raised.

12) Explain IEEE standard for floating pt nos.

2 representations

1) Single precision representation occupies single 32-bit word
scale factor has a range of 2^{-126} to 2^{+127}
32 bit word is divided into 3 fields : sign (1 bit)
exponent (8 bits)
mantissa (23 bits)

Signed exponent = E
Last 23 bits represent mantissa. Since binary normalization is used MSB of mantissa is equal to 1
24 bit mantissa provides a precision equivalent to about 7 decimal-digits.

2) Double precision representation occupies single 64-bit word
E' is in range $1 < E' < 2046$
53 bit mantissa provides a precision equivalent to 16 decimal digits

In IEEE notation exponent is in excess -127 (excess -1023) notation

Actual exponents represented as

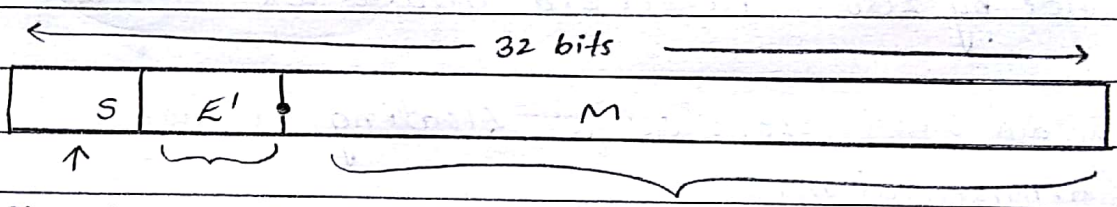
$-126 <= E <= 127$ and $-1022 <= E <= 1023$

not

$-127 <= E <= 128$ and $-1023 <= E <= 1024$

This is because the IEEE uses exponents -127 and 128 i.e actual values 0 and 255 represent special conditions

- exact 0
- infinity (∞)

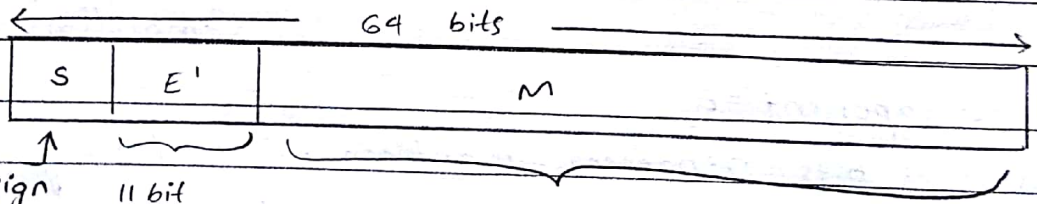


Sign of no
0 signifies +
1 signifies -

8 bit signed exponent in excess - 127 representation

23 - bit mantissa fraction

Value is represented $= \pm 1.M \times 2^{E'-127}$



sign

11 bit excess - 1023 exponent

52 bit mantissa fraction

Value represented $= \pm 1.M \times 2^{E'-1023}$

13) With a neat diagram explain floating point addition/subtraction.

Step 1:

- Compare components to determine how far to shift the

mantissa of the no. with smaller exponent

- The shift-count value, n, is determined by 8 bit subtractor circuit.

Magnitude of difference $E'A - E'B$, or n, is sent to SHIFTER unit

- In step 1, the sign is sent to the swap network
  - If sign = 0 $E'A > E'B$ and mantissas M_A & M_B are sent straight through SWAP network
  - If sign = 1 then $E'A < E'B$ and mantissas are swapped before they are sent to SHIFTER

Step 2:

2:1 MUX is used. The exponent of result E is tentatively determined as EA if $E_A > E_B$ or E_B if $E_A < E_B$

Step 3:

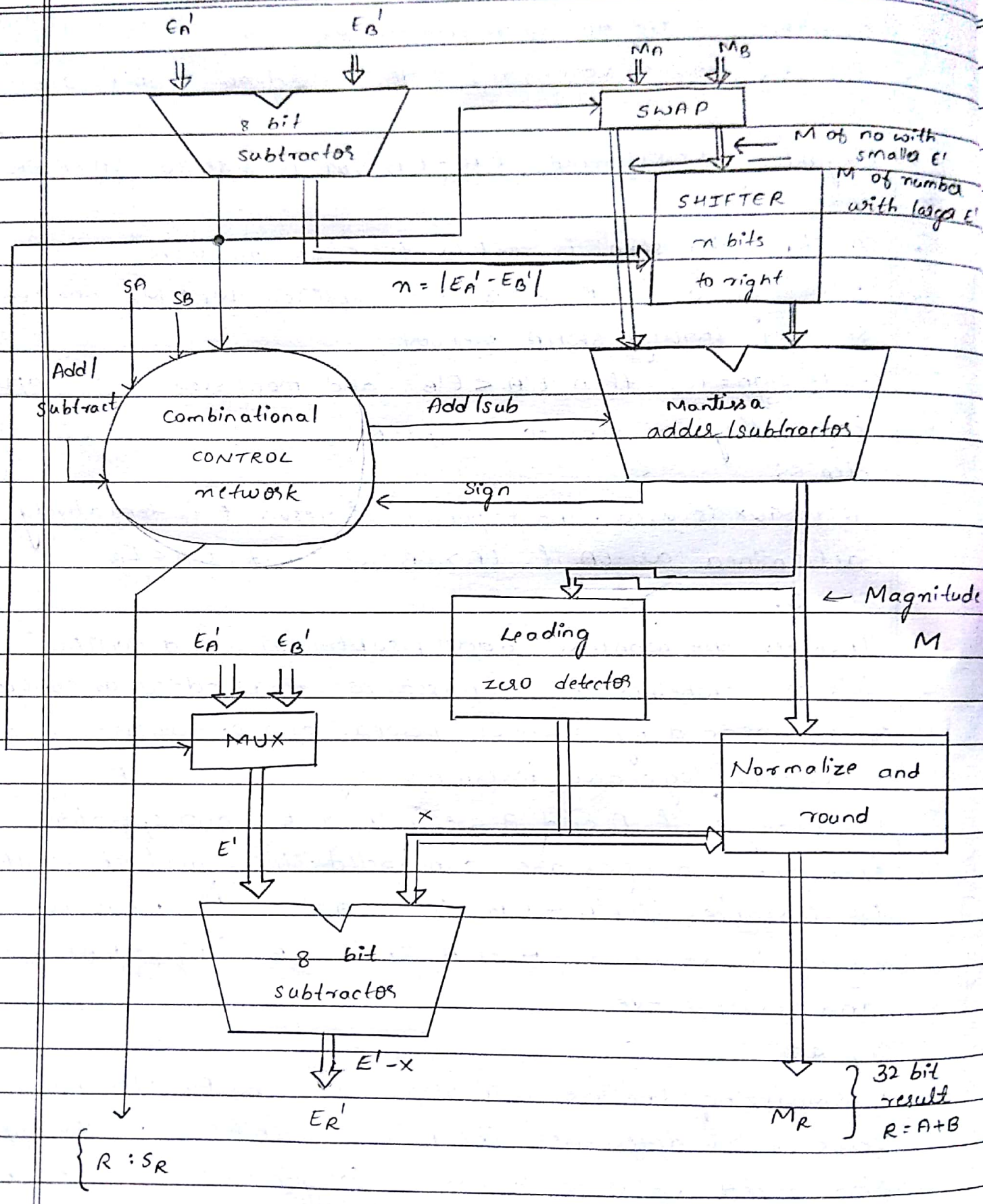
Involves the mantissa adder/subtractor and CONTROL logic to determine whether the mantissas are to be added or subtracted. This is decided by signs of operands and operation

- CONTROL logic also determines sign of result, SR

For example, if A and B are both +ve and operation is A-B then the mantissas are subtracted. The sign of the result, SR. For instance if $E'A \rightarrow E'B$ the M_A - (shifted M_B) is +ve and result is +ve. But $E'A > E'B$ then M_B - (shifted M_A) is +ve and result is -ve.

Step 4:

Normalizing the result in step 3, mantissa M. The no. of leading zeros in M determines the no. of bit shifts, X, to be applied to M. Normalized value is truncated to generate the 24 bit mantissa M_R, of the result. The value X is also subtracted from the tentative result exponent E to generate true result exponent, E_R



Problems:

1) Perform multiplication of -13 and +09 using Booth's algorithm

$$\begin{array}{r} -13 \\ \times\ 09 \end{array}$$

$-13 \rightarrow 01101 \xrightarrow{1's} 10010$

$+1$ 2's

10011

10011 (-13) Multiplicand

01001 (+9) Multiplier

$+1\ -1\ 0\ +1\ -1$

$0\ 1\ 0\ 0\ 1\ 0$

$1\quad 1\ 0\ 0\ 1\ 1$

$+1\ -1\ 0\ +1\ -1$

$0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 1$

$1\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ 1$

$0\ 0\ 0\ 0\ 0\ 0\ 0\ 0$

$0\ 0\ 0\ 1\ 1\ 0\ 1$

$1\ 1\ 0\ 0\ 1\ 1$

$①\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 1$

↓ 2's complement

$-\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0$

$+\ 1$

$0\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 1$

64 32 16 8 4 2 1

$\Rightarrow (-117)$

-1 2's complement of multiplicand

01100

$+1$

01101

2) Perform signed multiplication of nos (-12) and (-11) using Booth's Algorithm.

(-12) → 01100 1's complement → 10011

+1 2's complement

10100

(-11) → 01011 1's complement → 10100

+1 2's complement

10101

10100 (-12) Multiplicand

10101 (-11) Multiplier

1 0 1 0 1 0 −1 2's complement of multiplicand

−1 +1 −1 +1 −1 10100

01011

+1

01100

1 0 1 0 0

−1 +1 −1 +1 −1

0 0 0 0 0 0 1 1 0 0

1 1 1 1 1 0 1 0 0

0 0 0 0 1 1 0 0

1 1 1 0 1 0 0

0 0 1 1 0 0

10 10 10 10 1

1 0 0 0 1 0 0 0 1 0 0

Ignore 128 64 32 16 8 4 2 1

= (+132)

4) Perform restoring division given numbers 1000 / 11
Show all cycles
This problem is solved in Question no. 10

5) Perform 14 * -8 using Booth's algorithm.

14 → 01110

-8 → 01000 → 1's complement 10111 2's complement

$$\begin{array}{r} 10111 \\ +\quad 1 \\ \hline 11000 \end{array}$$

01110 (14)
11000 (-8)

1 10000
0 -1000

$$\begin{array}{r} 01110 \\ 0\ -1\ 0\ 0\ 0 \\ \hline 00000\ 00000 \\ 00000\ 000\ \ \\ 00000\ 00000 \\ 11101\ 0\ \ \\ 0000\ 00\ \ \\ \hline (1)\ 11001\ 00000 \end{array}$$

-1 takes 2's complement of multiplicand

01110
10001
+ 1
10010

$$\begin{array}{r} 00111\ 0000 \\ +\quad 1 \\ \hline 00111\ 0000 \end{array}$$

= (-112)

KSIT

6) Applying Booth's algorithm to multiply signed nos +13 and -6

+13 → 0 1 1 0 1

(-6) → 1 1 0 1 0

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

-ve

```
0 0 0 1 0 0 1 1 0 1
              + 1
0 0 0 1 0 0 1 1 1 0
```

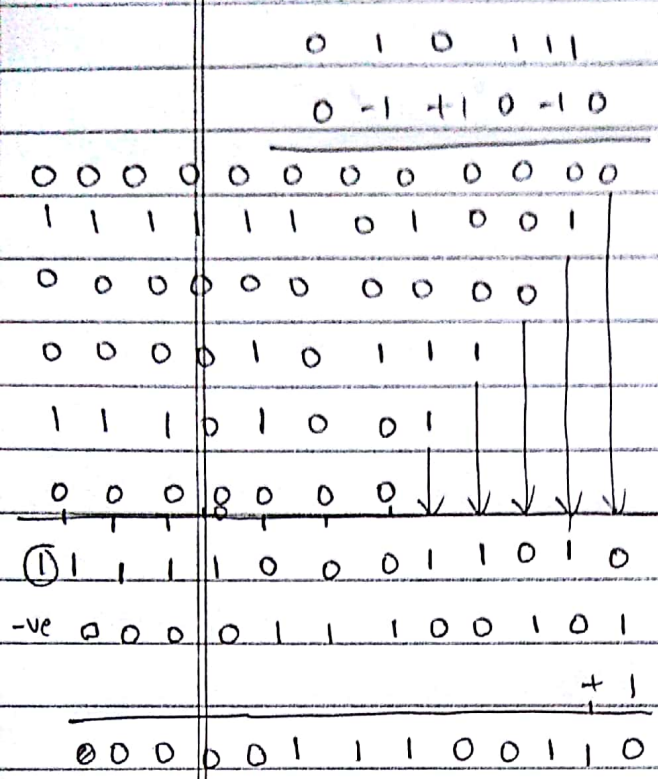
(-78)

7) Multiply the following using Booth's algorithm of signed 2's complement nos

Multiplicand = $(010111)_2$

Multiplier = $(110110)_2$

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



2's complement of multiplicand

1 0 1 0 0 0
+1
1 0 1 0 0 1

8) Perform division operation on the following unsigned no. using restoring method
Dividend = $(101010)_2$ and Divisor = $(00100)_2$