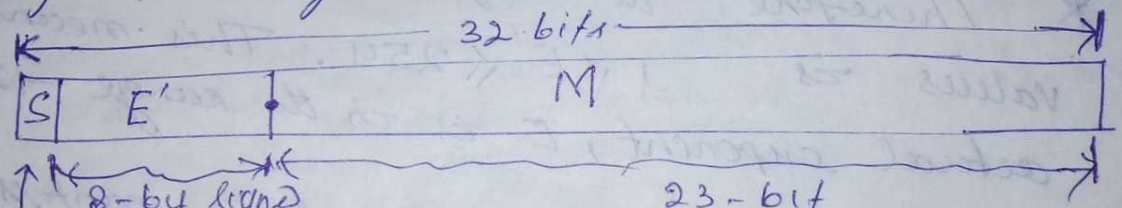
School of Computer Engineering
KIIT University, Bhubaneswar
Student Name.....
Roll No..... .....Branch/Sec.....
Sub.. .....Dt.....

IEEE Standard for Floating-Point Numbers

The standard for representing floating-point numbers in 32 bits has been developed and specified in detail by the Institute of Electrical and Electronics Engineers (IEEE). The 32-bit representation is given below.

* The sign of the number is given in the first bit, followed by a representation for the exponent (to the base 2) of the scale factor.
* Instead of the signed exponent E, the value actually stored in the exponent field is an unsigned integer $E' = E + 127$.



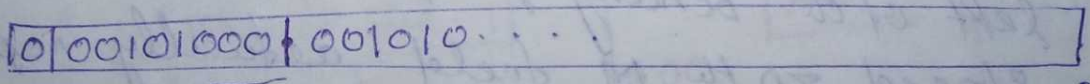
Sign of number: 0 signifies +, 1 " -

8-bit signed exponent in excess-127 representation

23-bit mantissa fraction

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

(Single Precision)



Sign

40

Value represented $= 1.001010\ldots0 \times 2^{-87}$

Actual $E = E' - 127 = 40 - 127 = -87$

(Example of a Single-Precision number.)

* This is called the excess-127 format.
Thus E' is in the range $0 \leq E' \leq 255$.

* The end values of this range, 0 & 255 are used to represent special values. When $E' = 0$ and the mantissa fraction M is zero, the value exact 0 is represented. When $E' = 255$ & $M = 0$ the value ∞ is represented. ~~& when 0 is the result~~ The sign bit is still part of these representations, so there are $+0$ & -0 ~~representations~~

* Therefore, the range of E' for normal values is $1 \leq E' \leq 254$. This means that actual exponent, E is in the range $-126 \leq E \leq 1$

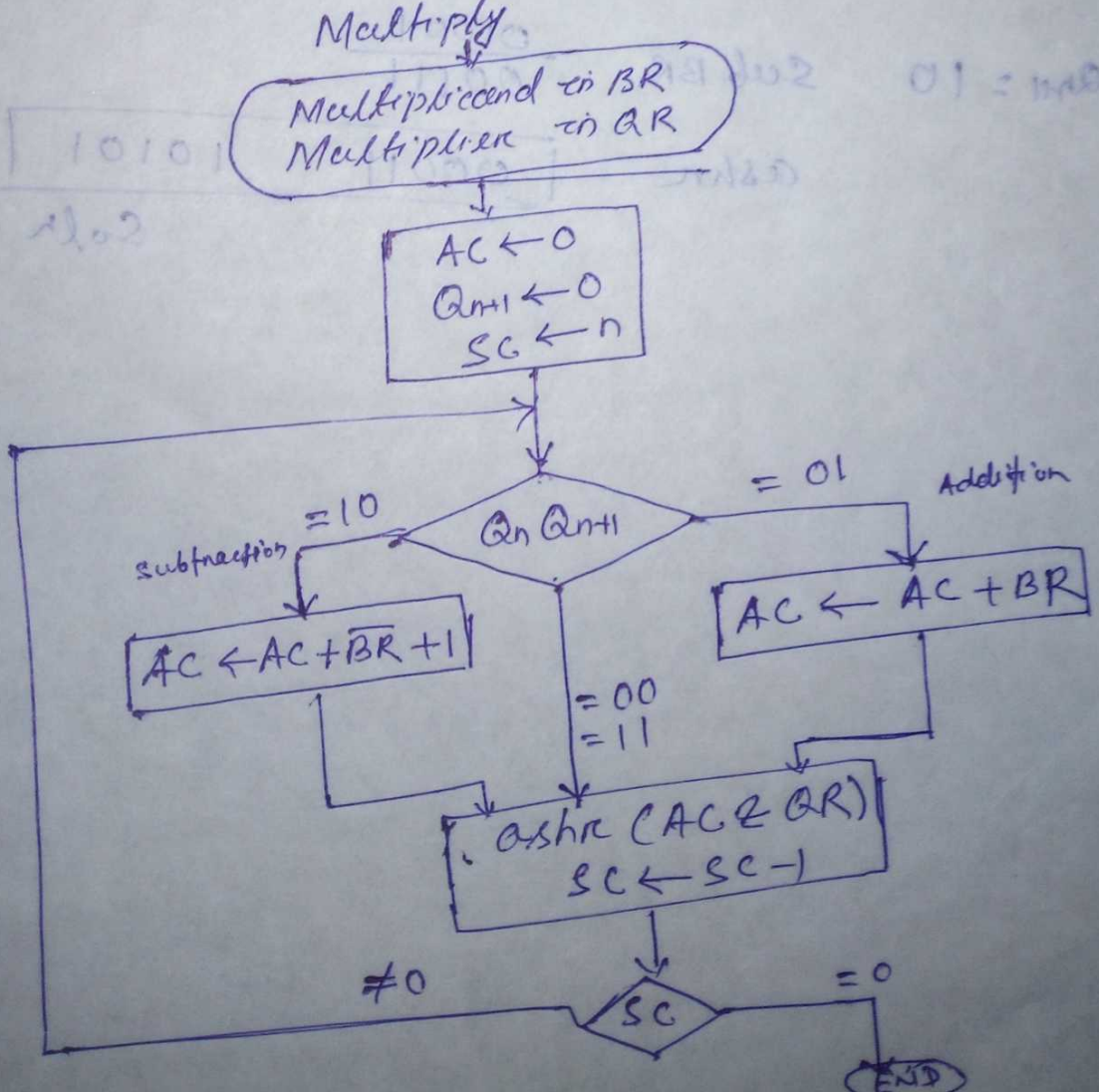
* The last 23 bits represent the mantissa. Since binary normalization is used, the most significant bit of the mantissa is always equal to 1. This bit is not explicitly represented, it is assumed to be the immediate left of the binary point. Hence, the 23 bits stored in the M field actually represent the fractional part of the mantissas. i.e, the bits to right of the binary point.

Booth Multiplication Algm of Signed-2's complement numbers.

* Aimed to improve the speed of multiplication of fixed point signed numbers.
* It doesn't require any correction to the final result irrespective of sign bit of the operands.

Description

* Multiplier and multiplicand are placed in QR and BR registers respectively.
* Qn designates the least significant bit of the multiplier in register QR. ~~An extra Qn~~
* An extra flip-flop Q_{n+1} is appended to QR to facilitate a double bit inspection of the multiplier.
* Ac and the appended bit Q_{n+1} are initially cleared to 0 and the sequence counter Sc is set to a number n equal to the number of bits in the multiplier.
* The flow-chart is given below.



Example $(-9) \times (-13) = +117$

$BR \leftarrow 10111$

$\overline{BR}+1 \leftarrow 01001$ (sub BR)

$QR \leftarrow (-13) = 10011$

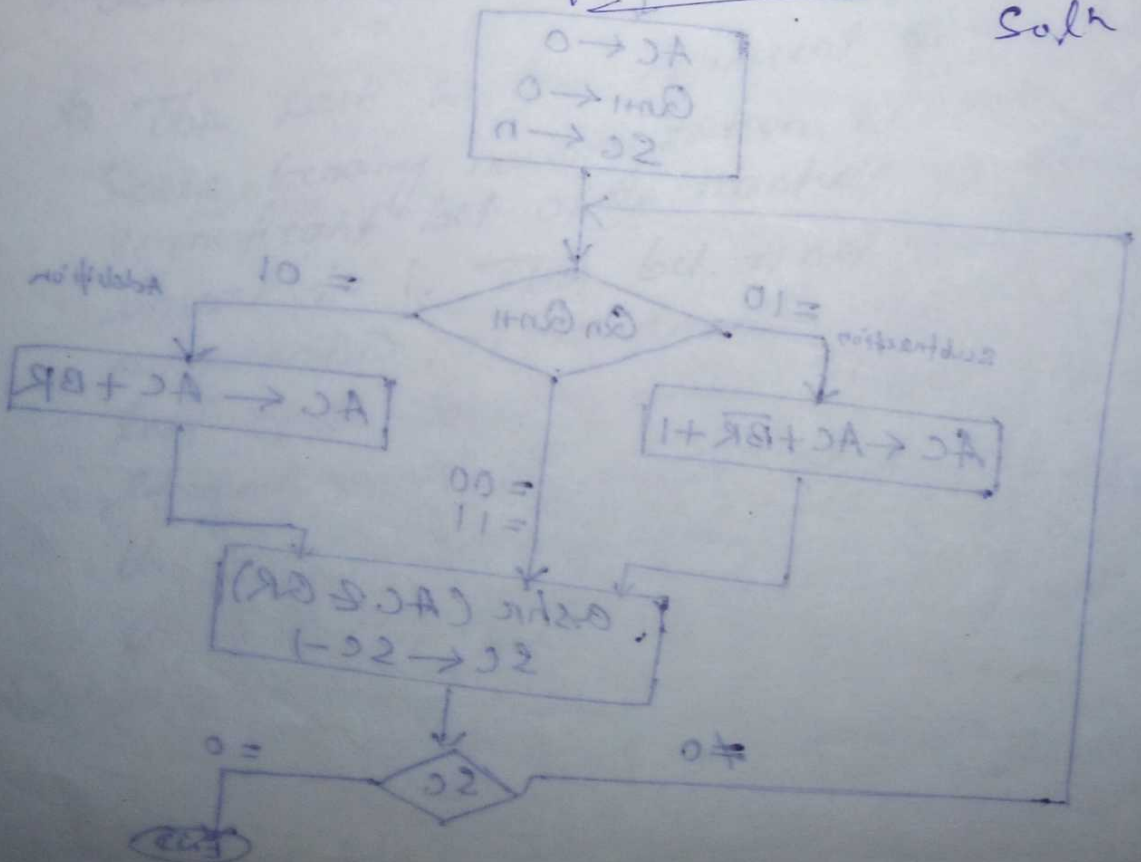
$9 = 01001$

$-9 = 10110 + 1$

$BR \leftarrow 10111$

operation	AC	QR	Q_n	Q_{n+1}	sc (decimal)
initial	00000	10011		0	5
$Q_nQ_{n+1}=10$ sub BR	01001				
	01001				
ashr	00100	11001	1		4
$Q_nQ_{n+1}=11$ ashr	00010	01100	1		3
$Q_nQ_{n+1}=01$ Add BR	10111				
	11001				
ashr	11100	10110	0		2
$Q_nQ_{n+1}=00$ ashr	11110	01011	0		1
$Q_nQ_{n+1}=10$ sub BR	01001				
	00111				
ashr	00011	10101	1		0

Soln



School of Computer Engineering
KIIT University, Bhubaneswar
Student Name.....
Roll No.....Branch/Sec.....
Sub.....Dt.....

INTEGER DIVISION

* An n-bit positive divisor is loaded into register BR and an n-bit positive dividend is loaded into register QR at start of the operation
* After the division is complete, the n-bit quotient is in register QR and the remainder is in register AC.
* The required subtraction are facilitated by using 2's complement arithmetic.
* The extra bit position at the left end of both AC and BR accomodates the sign bit during subtractions.

Restoring Division Algm (Do the following n-times)

1. Left shift AC and QR one binary position
2. AC ← AC − BR
3. if the sign of A is 1, set q_0 to 0 and
   AC ← AC + BR (restore AC)
   else
   set q_0 to 1

Nonrestoring Division Algm (Do the following n-times)

1. If the sign of AC is 0 then
   Shift AC and QR left one bit position and
   AC ← AC − BR
   else
   Shift AC and QR left and
   AC ← AC + BR
2. Now, if the sign of AC is 0 then
   Set q_0 ← 1
   else
   Set q_0 ← 0
3. if sign of A is 1 then AC ← AC + BR

Restoring Example :

$1000 \div 11$

$BR \leftarrow 00011$
$\overline{BR}+1 \leftarrow 11101$
$QR \leftarrow 1000$
$AC \leftarrow 00000$

Sign bit AC	Operation	AC	QR	cycle
	initially	00000	1000	
	[Shift left	00001	000□	1
	sub	11101		
		11110		
	Add (Restore)	11		
		00001	000□	
	Set q0	00010	000□	
	[Shift left	11101		2
	sub	11111		
	Add (Restore)	11		
		00010	00□□	
	Set q0	00100	00□□	
	Shift	11101		3
	sub	00001		
	Set q0	00010	00□1	
	Shift	11101		4
	sub	11111		
	Add	11	0010	
		00010		
	Set q0			

Remainder — Quotient

Sign bit
if 0 ⟷ Shift → Subtract
if 1 ⟷ Shift → Add
check sign bit of A 0 set $q_0 \to 1$ else $q_0 \to 0$

A non-restoring example

Sign bit of A	operation	AC	QR	cycle
0	initially	00000	1000	
	Shift left	00001	000□	
0		11101		1
		11110	0000	
1	Set q_0			
	Shift left	11100	00□□	
1	Add	00011	00□□	2
		11111		
	Set q_0			
	Shift	11110	0□□□	3
1	Add	00011	0□□□	
		00001		
0	Set q_0			
	Shift	00010	□□□□	4
0	Sub	11101	□□□□	
		00011		
1	Set q_0			

1	~~Add~~ Add	11111	
		00011	
		00010	0010
		Remainder	Quotient