Chapter 8
Computer Arithmetic

Arithmetic instructions manipulate data to produce solution for computational problems. The four basic arithmetic operations are addition, subtraction, multiplication and division. From these 4, it is possible to formulate other specific problems by means of numerical analysis methods.

There are three ways of representing negative fixed-point binary numbers: signed magnitude, signed 1's complement and signed 2's complement. Signed 2's complemented form is used most but occasionally we deal with signed magnitude representation.

8.1 Addition and Subtraction of Signed-Magnitude Data

When two signed numbers A and B are added and subtracted, we find 8 different conditions to consider as described in the following table:

Operation	**ADD Magnitudes**	**SUBTRACT Magnitudes**		
		A > B	**A < B**	**A = B**
(+A) + (+B)	+ (A + B)			
(+A) + (-B)		+ (A – B)	- (B – A)	+ (A – B)
(-A) + (+B)		- (A – B)	+ (B – A)	+ (A – B)
(-A) + (-B)	- (A + B)			
(+A) - (+B)		+ (A – B)	- (B – A)	+ (A – B)
(+A) - (-B)	+ (A + B)			
(-A) - (+B)	- (A + B)			
(-A) - (-B)		- (A – B)	+ (B – A)	+ (A – B)

Table: addition and subtraction of signed-magnitude numbers

Addition (subtraction) algorithm:

- When the signs of A and B are identical (different), add magnitudes and attach the sign of A to the result.
- When the signs of A and B are different (identical), compare the magnitudes and subtract the smaller from larger.

Hardware implementation

To implement the two arithmetic operations with hardware, we have to store numbers into two registers A and B. Let A_s and B_s be two flip-flops that hold the corresponding signs. The results are transferred to A and A_s. A and A_s together form an accumulator.

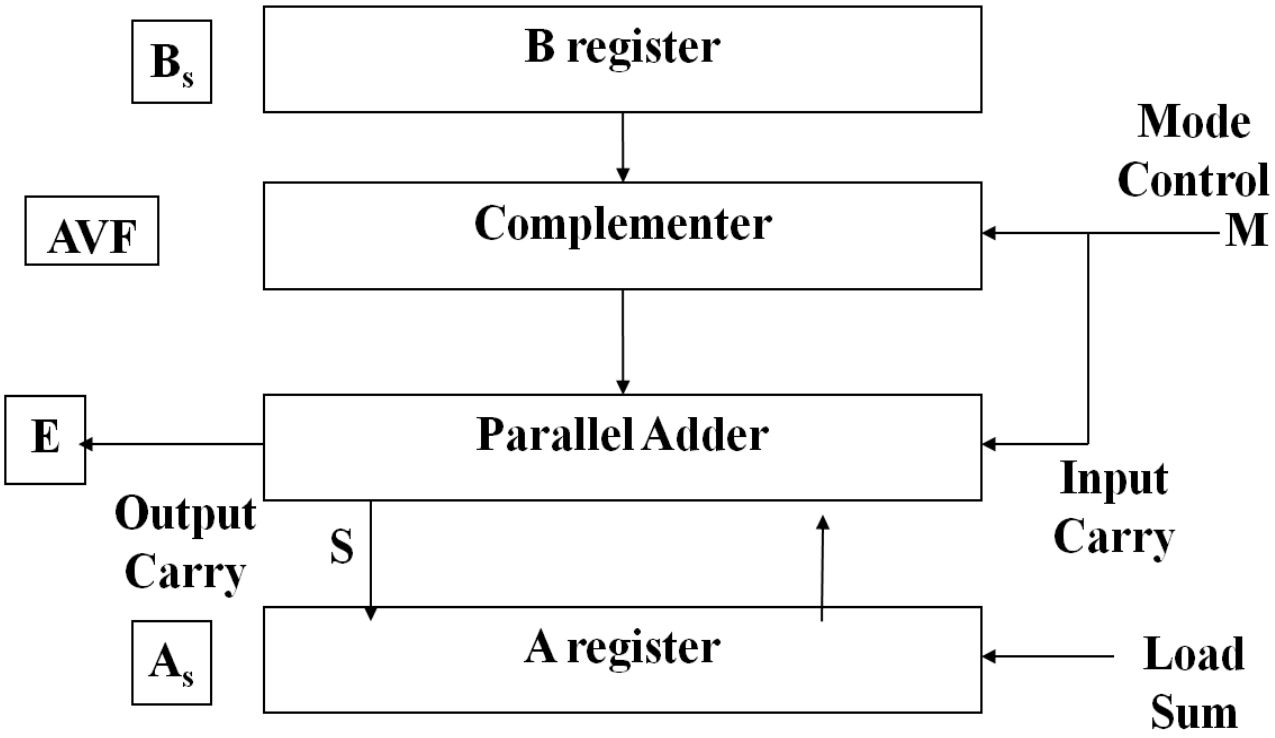


Fig: Hardware for signed-magnitude data addition and subtraction

We need:

- Two registers A and B and sign flip-flops A_s and B_s
- A magnitude comparator: to check if A>B, A<B, or A=B
- A parallel adder: to perform A+B
- Two parallel subtractors: for A-B and B-A
- The sign relationships are determined from an X-OR gate with A_s and B_s as inputs.

Block Diagram Description

- Hardware above consists of registers A and B and sign flip-flops A_s and B_s.
- The complementor provides an output of B or B' depending on mode input M.
- When M=0, the output of B is transferred to the adder, the input carry is 0 and thus output of adder is A+B. Output carry is transferred to flip-flop E, where it can be checked to determine overflow. When E=1, it is overflow, otherwise, not. The E=1 is transferred to the AVF (add-overflow-flip-flop) which holds the overflow bit when A ns B are added.
- When M=1, 1's complement of B is applied to the adder, input carry is 1 and output is S=A+B'+1 (i.e. A-B). Output carry is transferred to flip-flop E, where it can be checked to determine the relative magnitude of two numbers.

Hardware Algorithm

The flowchart for the hardware algorithm is given below:

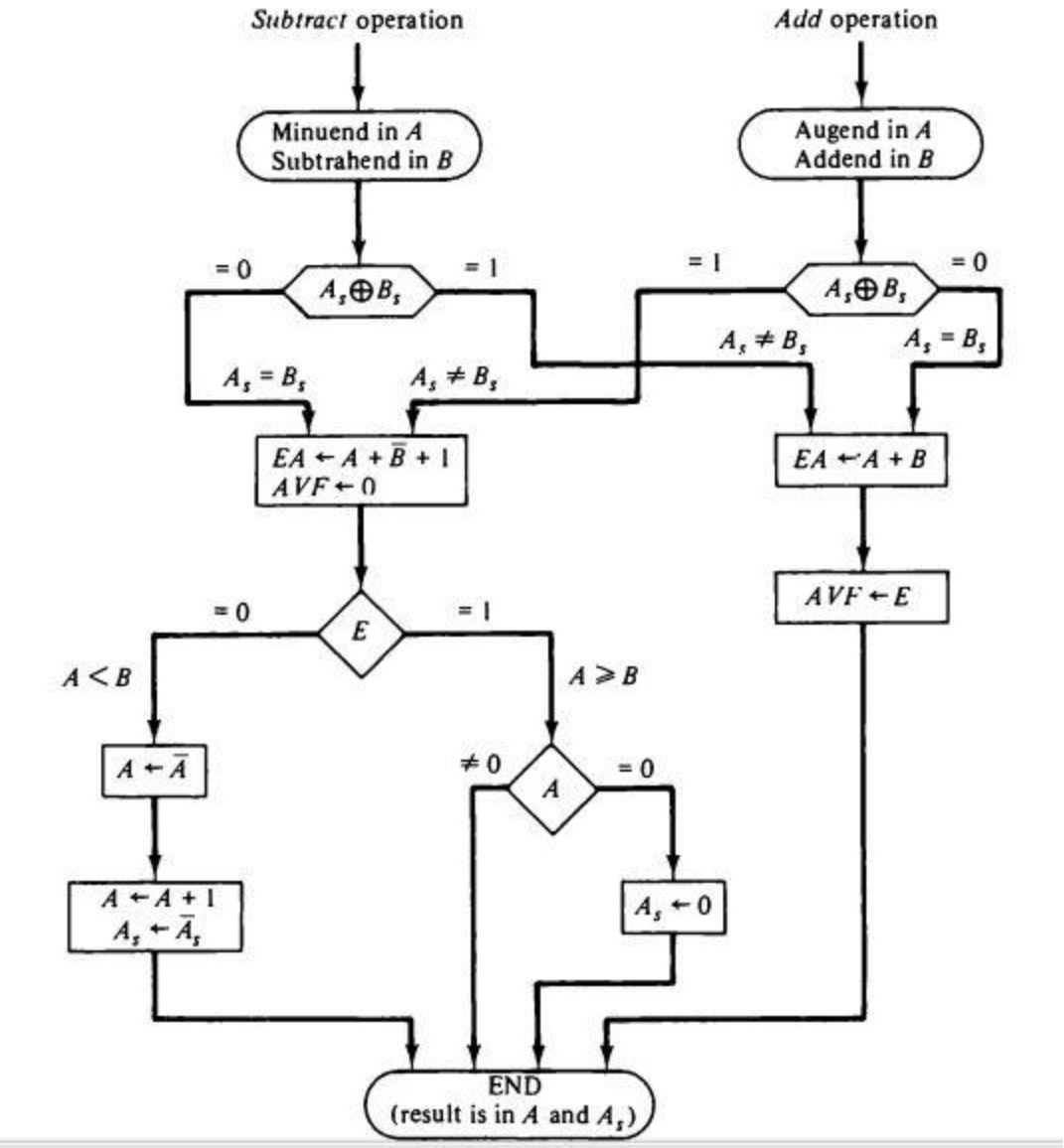


Fig: Flowchart for add and subtract operations with signed-magnitude data

→ As and Bs are compared by an X-OR gate. If output=0, signs are identical, if 1, signs are different.

→ For *add* operation, **identical** signs dictate addition of magnitudes. For *subtraction*, **different** signs dictate magnitudes be added.

- Magnitudes are with a micro operation E A ← A+B (E A is register that combines E and A). If E=1, overflow occurs and is transferred to AVF.

→ Two magnitudes are subtracted if signs are different for add operation and identical for subtract operation.

- Magnitudes are subtracted with a micro operation E A ← A+B'+1.
- No overflow occurs if the numbers are subtracted, so AVF is cleared to 0.
- If E=1, it indicates A≥B and the result in A is correct. If the number in A is 0, the sign A_s must be made positive (A_s=0) to avoid negative zero.
- If E=0, it indicates that A<B for which it is necessary to take 2's complement of value in A. the micro operation is A ←A'+1. The sign of A_s must be sign of Bs (i.e. $A_s \leftarrow A_s'$).

8.2 Addition and Subtraction of Signed 2's Complement Data

In signed 2's complement representation, the leftmost bit represents sign (0 for positive and 1 for negative). If sign bit is 1, entire number is represented in 2's complement form.

Addition: Sign bit is treated as other bits of the number. Carry out of the sign bit is discarded.

Subtraction: It consists of first taking 2's complement of subtrahend and then adding it to minuend.
When two numbers of n-digit each are added, the sum occupies n+1 bits. Overflow occurs which is detected by applying last two carries out of the addition to XOR gate. The overflow occurs only if output of the gate is 1.

Hardware Implementation

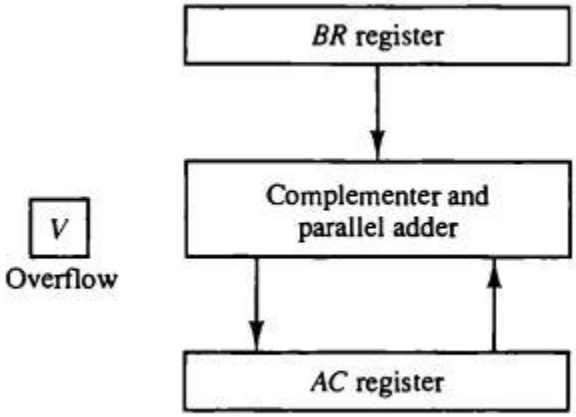


Fig: Hardware for signed-2's complement data addition and subtraction

→ Register configuration is same as signed-magnitude representation except sign bits are not separated. The leftmost bits in AC and BR represent sign bits.
→ Significant Difference: Sign bits are added together with the other bits in complementer and parallel adder. The overflow flip-flop V is set to 1 if there is an overflow. Output carry in this case is discarded.

Hardware Algorithm

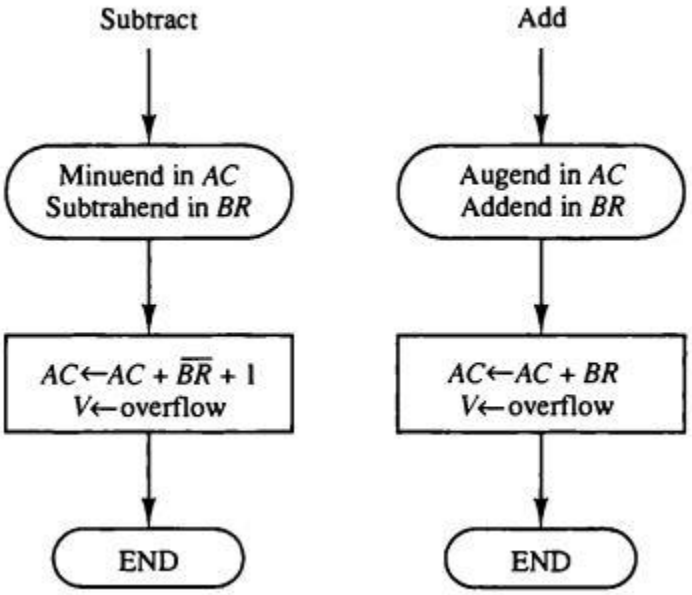


Fig: Algorithm for signed-2's complement data addition and subtraction

Comparing this with its signed-magnitude counterpart, it is much easier to add and subtract numbers. For this reason, most computers adopt this representation over the more familiar signed-magnitude.
Example: 33+(-35)
AC = 33 = 00100001
BR = -35 = 2's complement of 35 = 11011101
AC+BR = 11111110 = -2

8.3 Multiplication of Signed-Magnitude Data

For this representation, multiplication is done by a process of successive shift and adds operations. As an example:

```
 23      10111     Multiplicand
 19    × 10011     Multiplier
       ------
       10111
      10111
     00000    +
    00000
   10111
  ---------
437 110110101     Product
```

Process consists of looking successive bits of the multiplier, least significant bits first.

- If the multiplier bit is 1, the multiplicand bit is copied down; otherwise zeroes are copied down.
- Numbers copies down in successive lines are shifted one position left. Finally, numbers are added to form a product.

The sign of the product is determined from the signs of the multiplicand and multiplier.

- If they are alike, the sign of the product is positive.
- If they are unlike, the sign of the product is negative.

Hardware Implementation

It needs same hardware as that of addition and subtraction of signed-magnitude. In addition, it needs two more registers Q and SC.

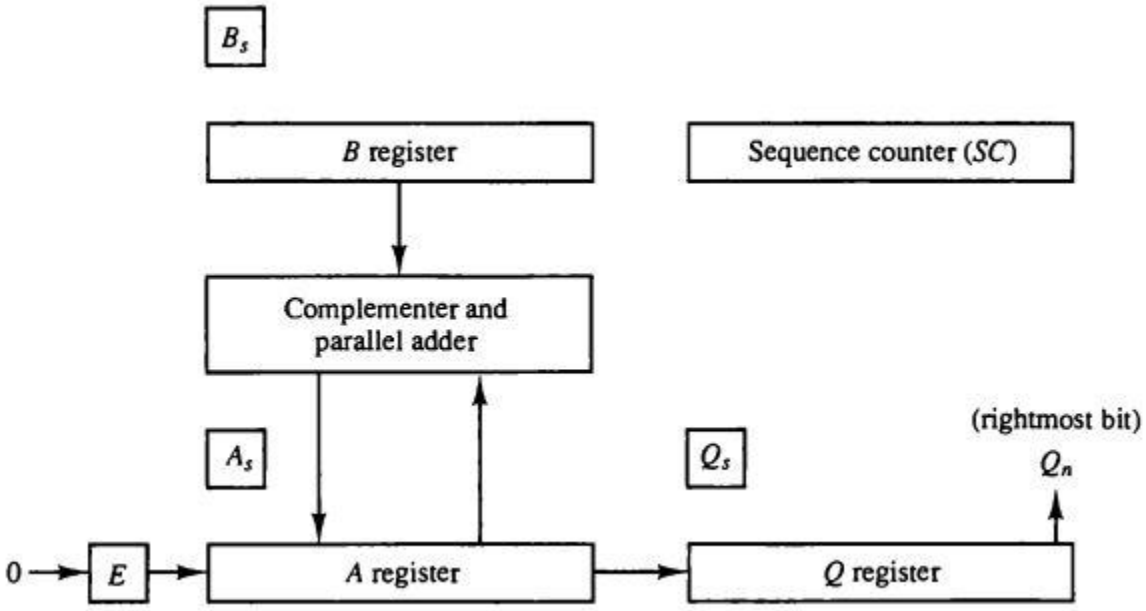
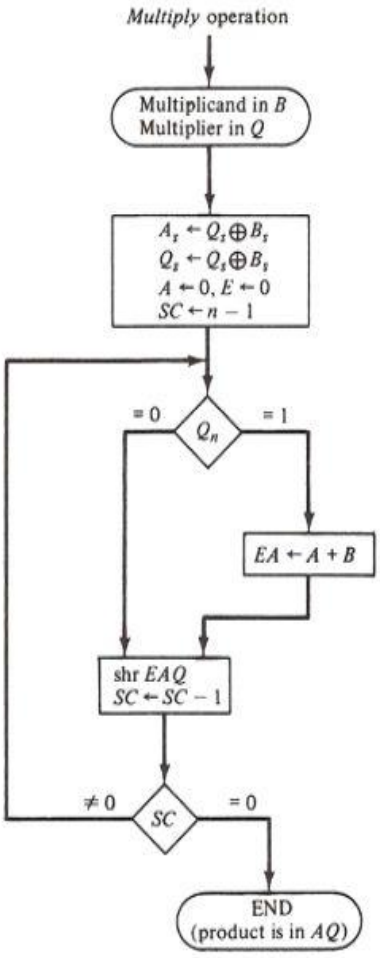


Fig: Hardware for signed-magnitude multiply operation

→ B←multiplicand, B_s←sign
→ Q←multiplier, Q_s←sign
→ Successively accumulate partial products and shift it right
→ SC←no. of bits in multiplier (magnitude only)
→ SC is decremented after forming each partial product. When SC is 0, process halts and final product is formed.
→ Sum of A and B forms a partial product.

Hardware Algorithm



Example: B = 10111 (**Multiplicand**)
Q = 10011 (**Multiplier**)

Operation	E	A	Q	SC
Initial conf.	0	00000	10011	101
Iteration 1 (Q_n = 1)		00000		
EA← A+B		+ 10111		
PP1-->	0	10111		
shr EAQ, SC←SC-1	0	01011	11001	100
Iteration 2 (Q_n = 1)		01011		
EA← A+B		+ 10111		
PP2-->	1	00010	11001	
shr EAQ, SC←SC-1	0	10001	01100	011
Iteration 3 (Q_n = 0)				
shr EAQ, SC←SC-1	0	01000	10110	010
Iteration 4 (Q_n = 0)				
shr EAQ, SC←SC-1	0	00100	01011	001
Iteration 5 (Q_n = 1)		00100		
EA← A+B		+ 10111		
PP3-->	0	11011	01011	
shr EAQ, SC←SC-1	0	01101	10101	000
Final Product in AQ	0110110101			

Fig: Algorithm for signed-magnitude multiply operation

8.4 Multiplication of Signed 2's Complement Data (Booth Multiplication Algorithm)

Booth algorithm is used to multiply binary numbers in signed-2's complement form.

Rules:

i) The partial product doesn't change when the multiplier bit is identical to the previous multiplier bit. (Q_nQ_{n+1} = 00 or 11)

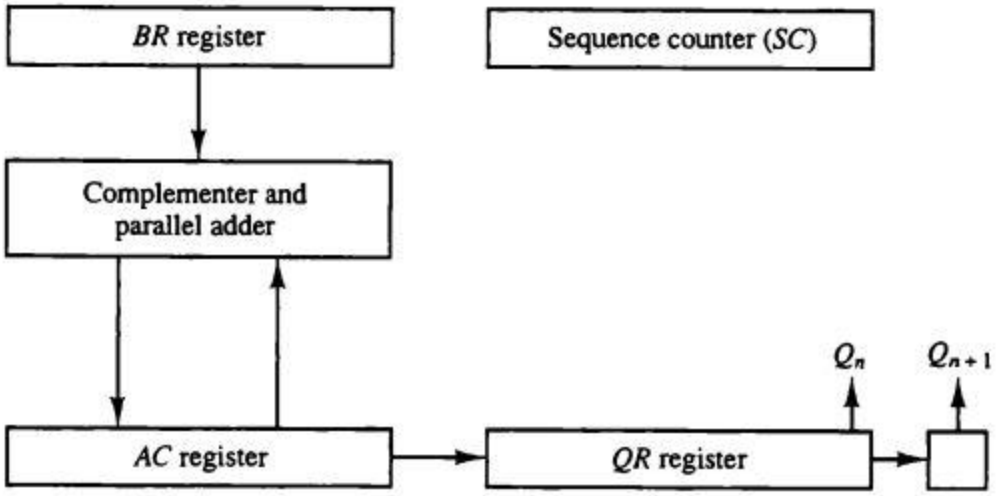
ii) The multiplicand is added to the partial product if LSB is 0 in the string of 0's in the multiplier. (Q_nQ_{n+1} = 01)

iii) The multiplicand is subtracted from the partial product if LSB is 1 in the string of 1's in the multiplier. (Q_nQ_{n+1} = 10)

iv) After each addition/subtraction, the partial product is shifted right using arithmetic shift.

This algorithm can be used for both the positive and negative multiplier in 2's complement form.

Hardware for Booth Algorithm



- Here, sign bits are not separated.
- Registers A, B and Q are renamed to AC, BR and QR respectively.
- Extra flip-flop Q_{n+1} appended to QR is needed to store almost lost right shifted bit to the multiplier (which along with current Q_n gives information about bit sequencing of multiplier).
- Pair Q_nQ_{n+1} inspect double bits of the multiplier.

Hardware Booth Algorithm

E.g. (-9)*(-13) = +117
10111*10011

Numerical Example: **Booth algorithm**

BR = 10111 (**Multiplicand**)
QR = 10011 (**Multiplier**)

Q_nQ_{n+1}	BR = 10111 $\overline{BR}$ + 1 = 01001	AC	QR	Q_{n+1}	SC
	Initial	00000	10011	0	101
1 0	Subtract BR	01001			
		01001			
	ashr	00100	11001	1	100
1 1	ashr	00010	01100	1	011
0 1	Add BR	10111			
		11001			
	ashr	11100	10110	0	010
0 0	ashr	11110	01011	0	001
1 0	Subtract BR	01001			
		00111			
	ashr	00011	10101	1	000

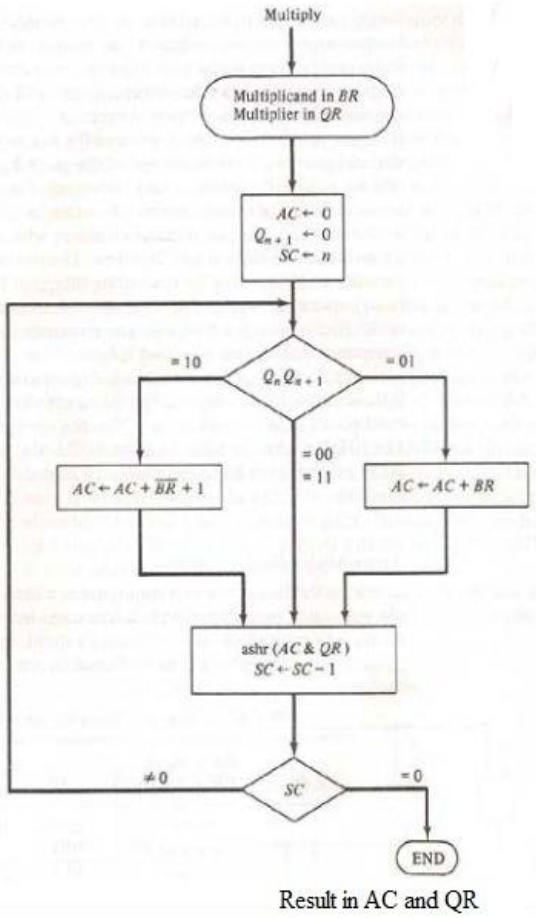


Fig: Flowchart for Booth Algorithm for signed-2's complement data multiplication