



## Unit-7 Computer Arithmetic:

In this chapter, hardware implementation are not more imp in this chapter. Algorithms and questions / Numericals are important.

### Q Addition and Subtraction with Signed Magnitude Data:

When the signed numbers are added or subtracted, then there are eight different conditions to consider, depending on the sign of the numbers and the operation performed. These conditions are listed in the first column of table below. The other columns in the table show the actual operation to be performed with the magnitude of the numbers. The last column should be positive. (i.e., when two equal numbers are subtracted the result should be +0 not -0).

Operation	Add Magnitudes	Subtract Magnitudes		
		When $A > B$	When $A < B$	When $A = B$
$(+A) + (+B)$	$+ (A + B)$	$+ (A - B)$	$- (B - A)$	$+ (A - B)$
$(+A) + (-B)$		$- (A - B)$	$+ (B - A)$	$+ (A - B)$
$(-A) + (+B)$		$- (A - B)$	$+ (B - A)$	$+ (A - B)$
$(-A) + (-B)$	$- (A + B)$			
$(+A) - (+B)$		$+ (A - B)$	$- (B - A)$	$+ (A - B)$
$(+A) - (-B)$	$+ (A + B)$			
$(-A) - (+B)$	$- (A + B)$			
$(-A) - (-B)$		$- (A - B)$	$+ (B - A)$	$+ (A - B)$

table: Addition and Subtraction of Signed Magnitude Numbers.

### Hardware Implementation:

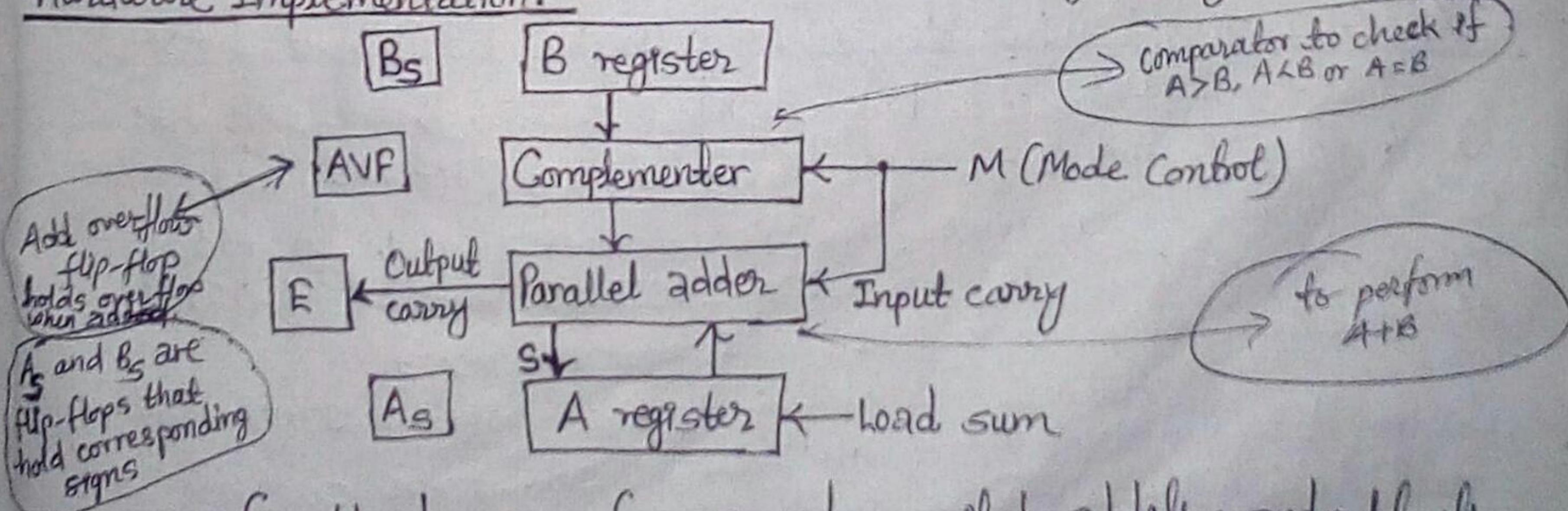


fig. Hardware for signed-magnitude addition and subtraction.

## Q. Hardware Algorithm / Flowchart:-

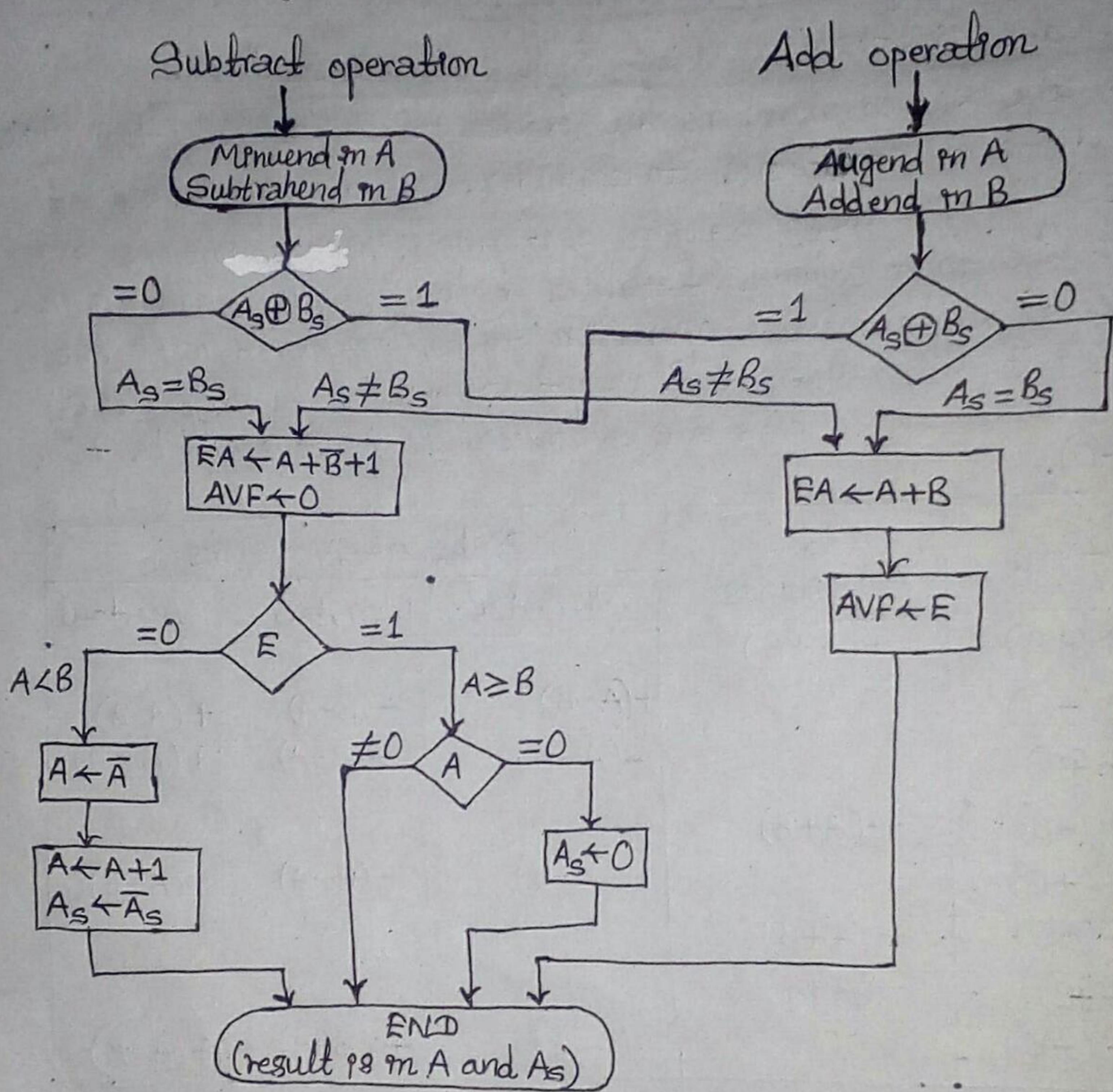


fig. flowchart for add and subtract operations.

## Q. Addition and Subtraction with Signed 2's Complement Data:-

The leftmost bit of a binary number represents the sign bit: 0 for positive and 1 for negative. If the sign bit is 1, the entire number is represented in 2's complement form. Thus, +33 is represented as 00100001 and -33 as 11011111. Note that 11011111 is the 2's complement of 00100001, and vice versa.

A carry-out of the sign-bit position is discarded during the addition of two numbers. The subtraction consists of first taking the 2's complement of the subtrahend and then adding it to the minuend.

## Hardware Implementation:

- Register configuration is same as in signed magnitude representation except sign bits are not separated. The leftmost bits in AC and BR represent sign bits.
- Sign bits are added and subtracted 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.

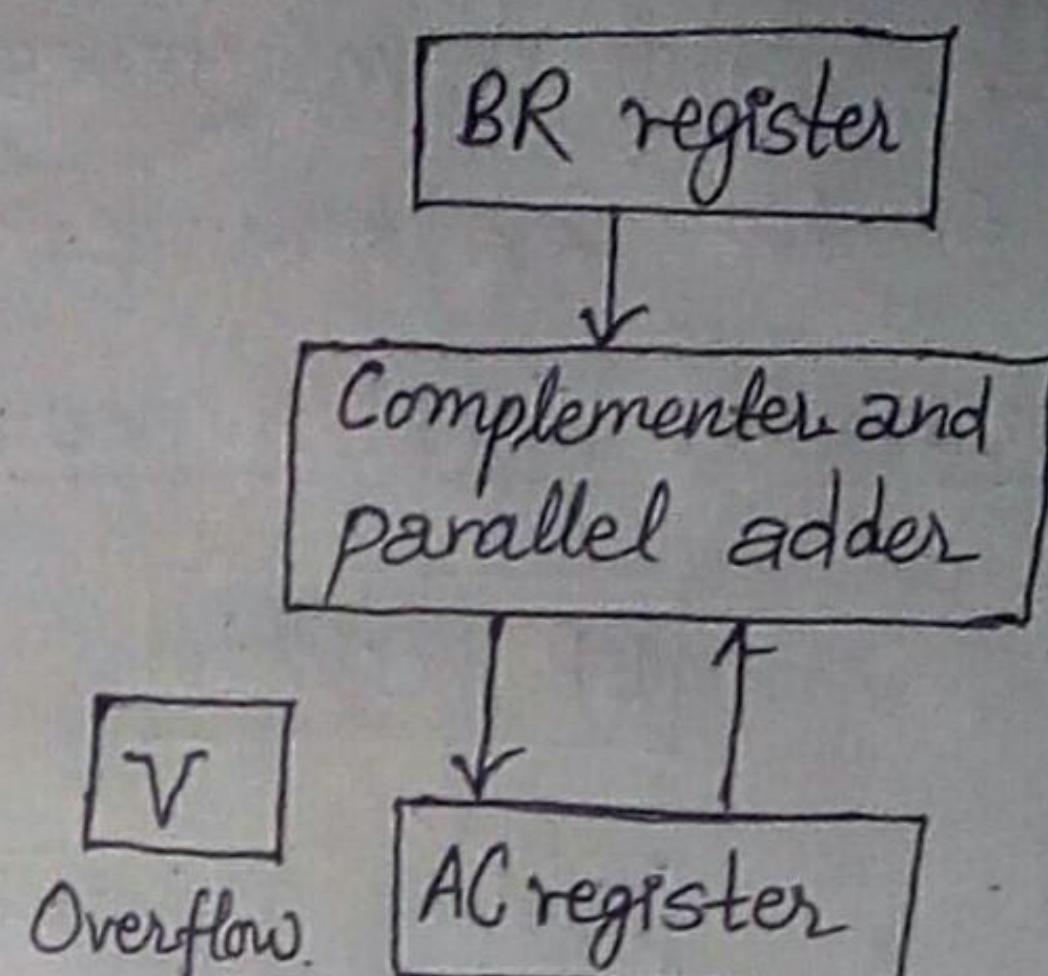


fig. Hardware for signed -2's complement addition and subtraction.

## Algorithm :

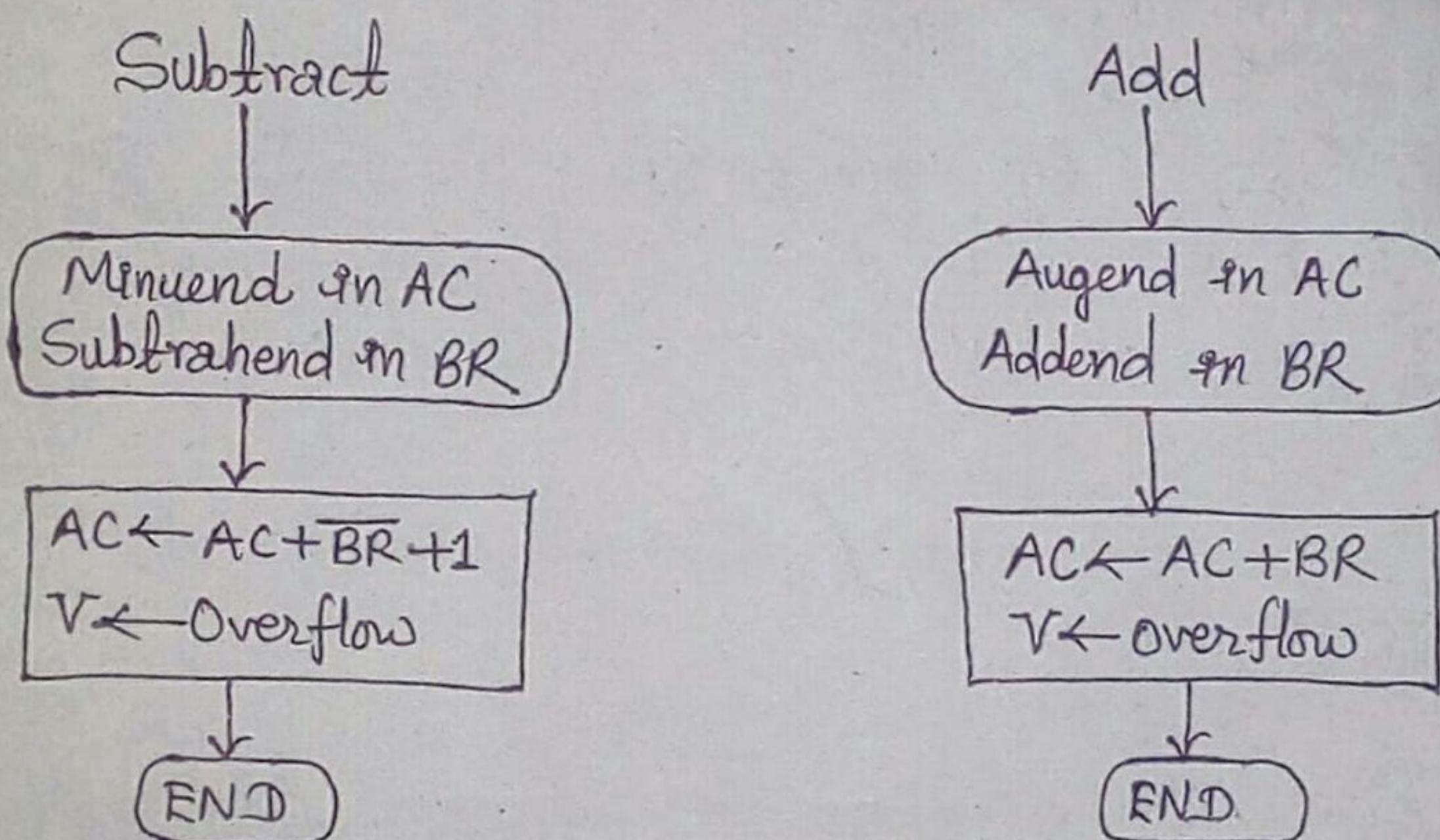


fig. Algorithm for adding and subtracting numbers in signed -2's complement representation.

## Booth Multiplication Algorithm: [v.Imp]

Booth algorithm gives a procedure for multiplying binary integers in signed 2's complement notation.

### Hardware Implementation:-

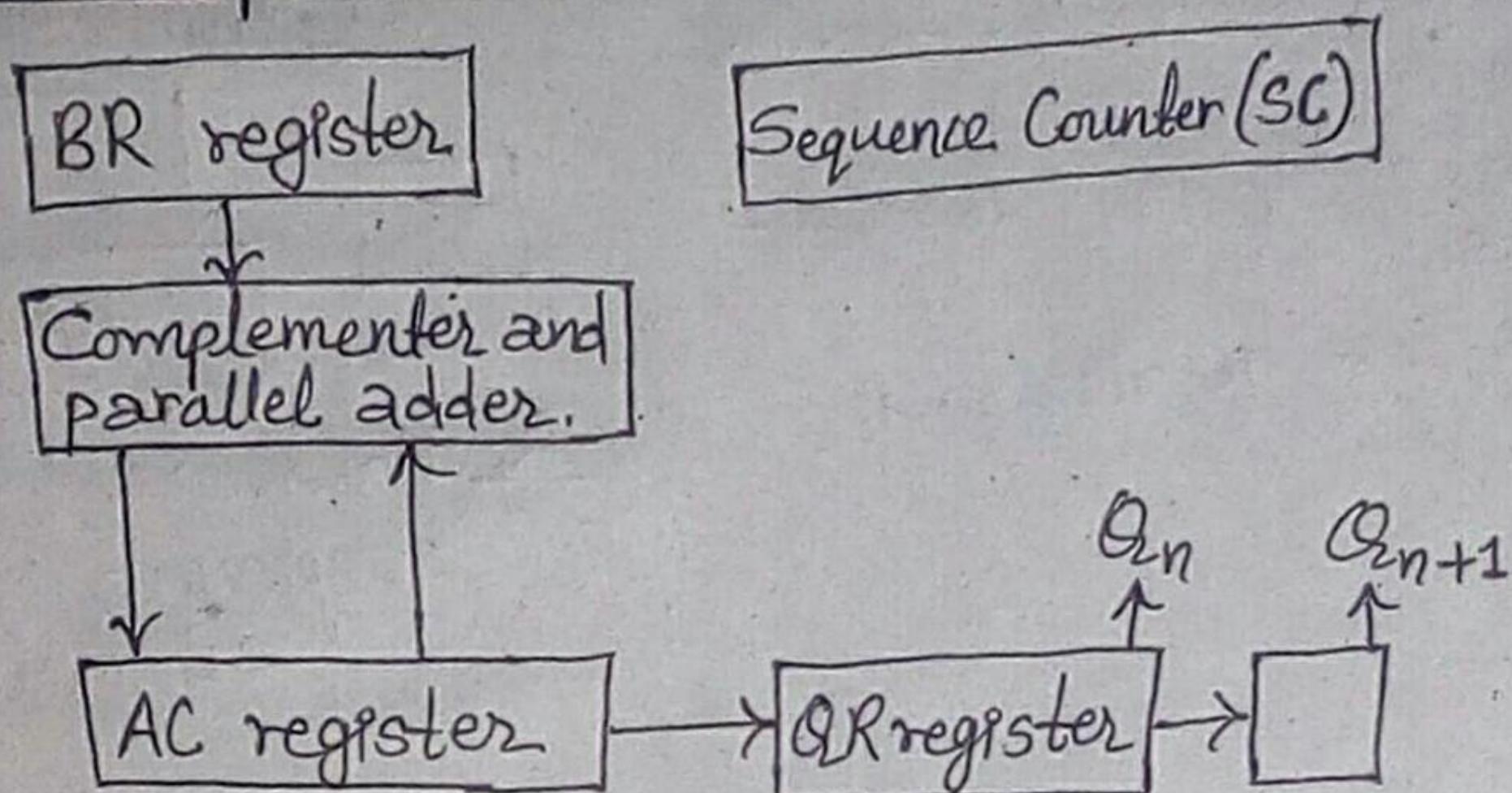


fig. Hardware for Booth Algorithm:

- Here, sign bits are not separated.
- Registers A, B and Q are renamed to AC, BR and QR.
- Extra flip-flop  $O_{n+1}$  is appended to QR which stores almost lost right shifted bit of the multiplier.
- Pair  $O_n O_{n+1}$  inspect double bits of the multiplier.

## Algorithm / Flowchart

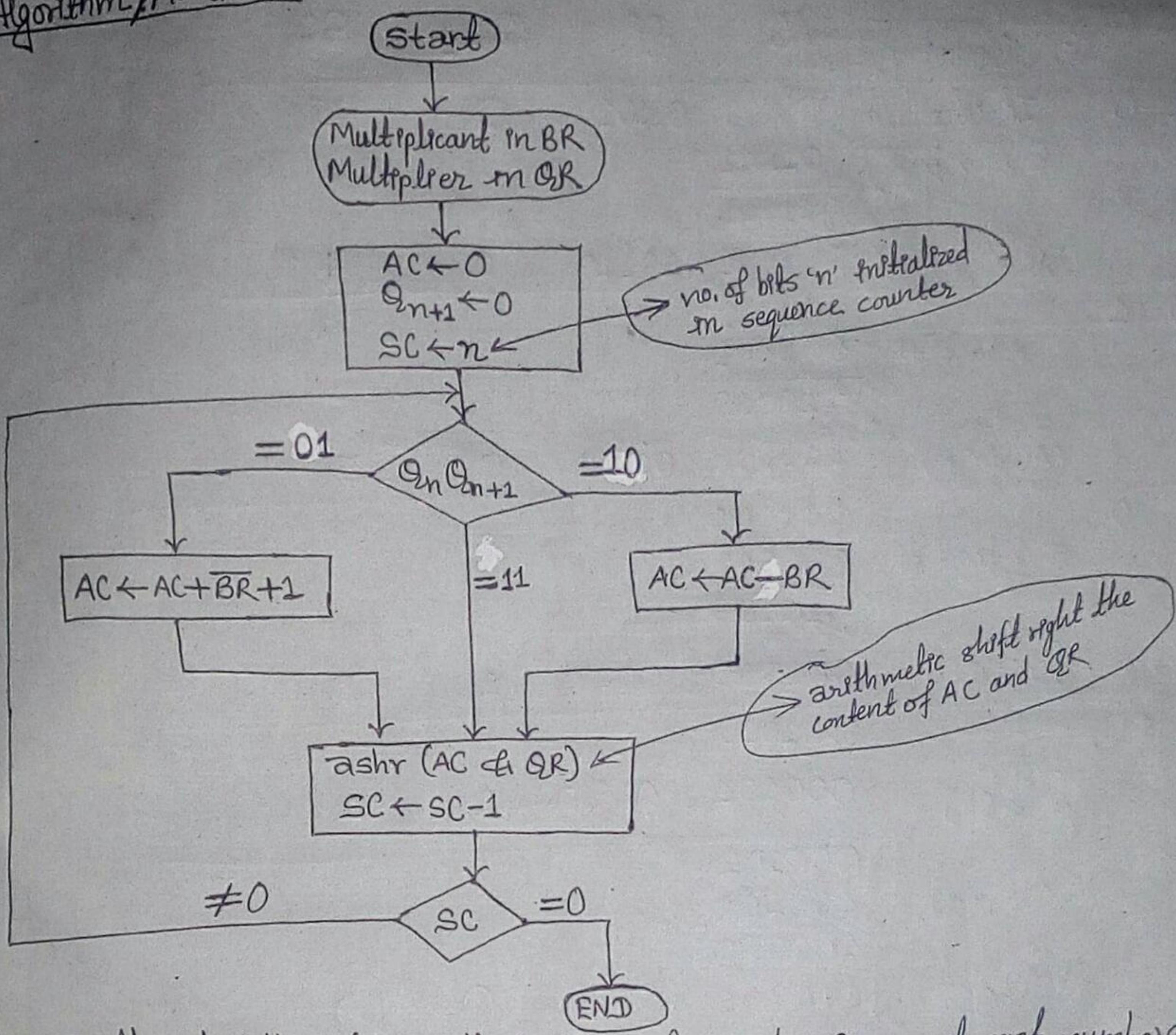


fig. Booth algorithm for multiplication of signed -2's complement numbers.

Example / Numerical Question:- (V. Imp)

Numerical संख्या संग्रह  
संग्रहीय flowchart परि छेत्री र  
compare गर्ने

⊗ Multiply  $(-27) * (+13)$  using Booth multiplication algorithm.

Solution:

Multiplicand (BR)  $+27 = 011011$

Now,

$$\overline{BR} = 100100$$

$$\overline{BR} + 1 = 100101$$

binary of 27  
additional 1-bit

everytime we  
add 0 as one  
additional bit

Multiplexer (QR) =  $+13 = 01101$

binary of 13  
additional one bit

for understanding  
only skip this

Note: Before solving question remember that:

$Q_n$  = LSB of multiplier on QR register.

Conditions: i) If  $Q_n Q_{n+1} = 10$ , then subtract  $BR$  and do ashtr.

ii) If  $Q_n Q_{n+1} = 01$ , then add  $\overline{BR} + 1$  and do ashtr.

iii) If  $Q_n Q_{n+1} = 11$ , simply do ashtr.

Algorithm

And sequence  
counter (SC) is  
decremented by 1 in  
each step.

$Q_n$	$Q_{n+1}$	Comment (Operation)	AC	QR	$Q_{n+1}$	SC
1 0	0	Initialization	000000	01101	0	5
		Subtract BR	$\frac{011011}{011011}$	001101	1	4
		ashr (AC & QR)	$\frac{001101}{10110}$	110010	0	3
0 1		Add $\overline{BR+1}$	$\frac{001101}{100101}$	111001		
		ashr (AC & QR)	$\frac{111001}{010100}$	01011		
1 0		Subtract BR	$\frac{011011}{010100}$			
		ashr (AC & QR)	001010	00101	1	2
1 1			001010			
		ashr (AC & QR)	000101	00010	1	1
0 1			000101			
		Add $\overline{BR+1}$	$\frac{100101}{101010}$			
		ashr (AC & QR)	110101	00001	0	0

Hence, Result = AC & QR

i.e. 110101 00001

Since sign bit is 1, so, the result will be negative.

The 2's complement of 11010100001 is 00101011110

$$\begin{array}{r} 00101011110 \\ +1 \\ \hline 00101011111 \end{array}$$

$$\begin{aligned}
 \therefore \text{Result} &= -(2^8 + 2^6 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0) \\
 &= -(256 + 64 + 32 + 8 + 4 + 2 + 1) \\
 &= -357 \quad \underline{\text{Ans}}
 \end{aligned}$$

Left side जाए  
1 अंकों place  
दूर गणना  
पर 2 मा ले जाए

## \* Division Algorithm: [V. Imp]

This algorithm provides a quotient and remainder, when we divide two numbers. While implementing division in digital systems, we adopt slightly different approach. Instead of shifting divisor right, the dividend is shifted left. Hardware implementation is similar to multiplication (but not booth) as below:

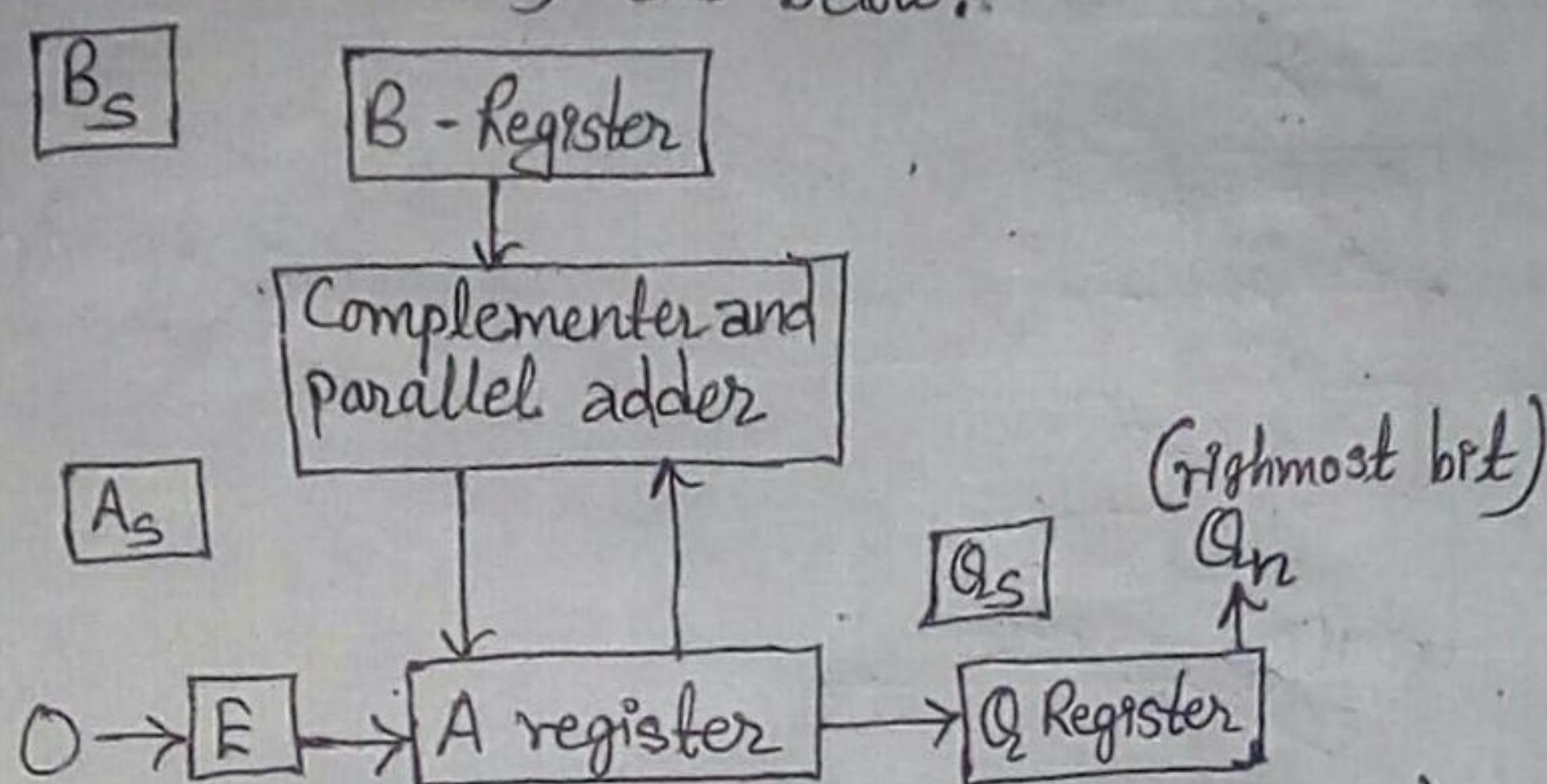
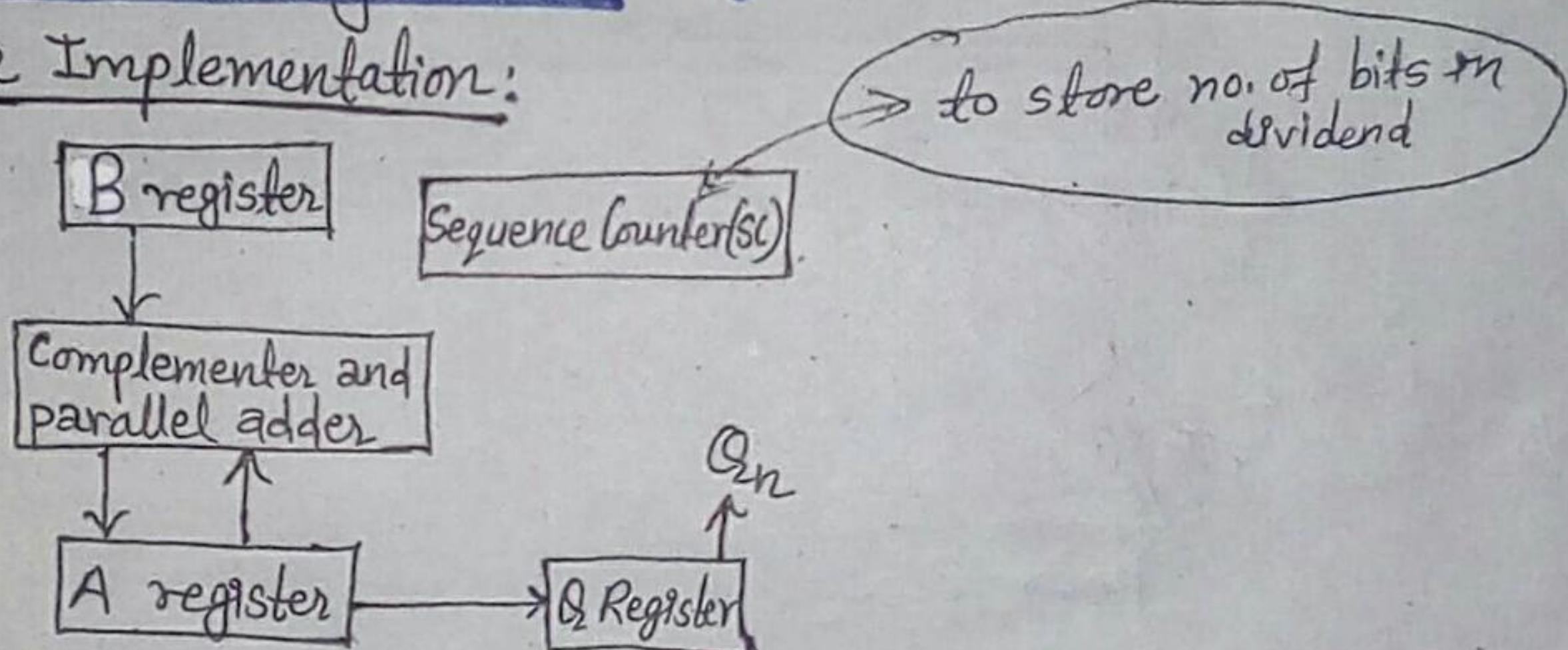


fig. Hardware for division (as well as multiplication) operation.

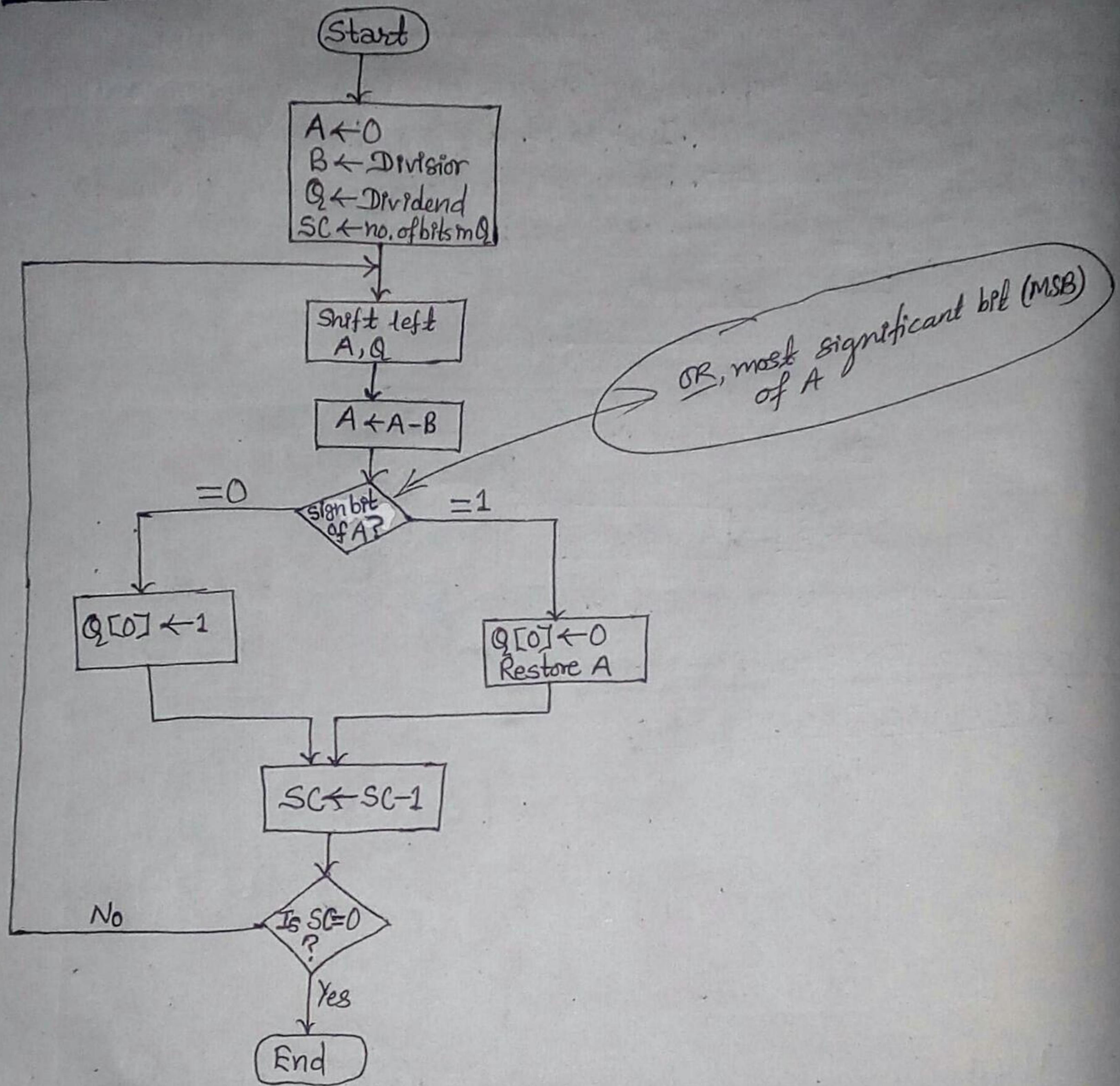
## 1) Restoring Division Algorithm:-

### Hardware Implementation:



Hardware implementation of division algorithm consist of B register contains divisor, Q register contains dividend and A register is initially kept zero, and this is the register whose value is restored during iteration due to which this is named as restoring. It consist of a sequence counter(SC) and a complementer and parallel adder also to check if  $A < B$  or  $A > B$  or  $A = B$  and to perform addition operation between A register and B register.

## Flowchart:



Quotient is in  $Q$   
& Remainder is in  $A$

Example: Divide 11 by 3 using restoring division algorithm OR  
division of unsigned integer method, by restoring.

Solution: dividend =  $Q = 11 = 1011$  (In binary) of  $n$ -bit

& divisor =  $B = 3 = 00011$  (In binary) of  $n+1$ -bit

$B$	Action/operation	$A$	$Q$	SC
$= 00011$				
$00011$	Initialization	$00000$	$1011$	
	Shift left A, Q	$00001$	$011?$	
	$A \leftarrow A - B$	$11110$	$011?$	
	$Q[0] \leftarrow 0$	$00001$	$0110$	4
	Restore A	$00010$	$110?$	
	Shift left A, Q	$11111$	$110?$	
	$A \leftarrow A - B$	$00010$	$1100$	
	$Q[0] \leftarrow 0$	$00101$	$100?$	
	Restore A	$00010$	$100?$	
	Shift left A, Q	$00010$	$1001$	3
	$A \leftarrow A - B$	$00010$	$1001$	
	$Q[0] \leftarrow 1$	$00010$	$0011$	
	Shift left A, Q	$00101$	$001?$	
	$A \leftarrow A - B$	$00010$	$001?$	
	$Q[0] \leftarrow 1$	$00010$		2
		$00101$	$001?$	
		$00010$	$001?$	
		$00010$		1
		$00101$	$001?$	
		$00010$	$0011$	0

Hence, Quotient =  $Q = 0011 = 3$

& Remainder =  $A = 00010 = 2$ .

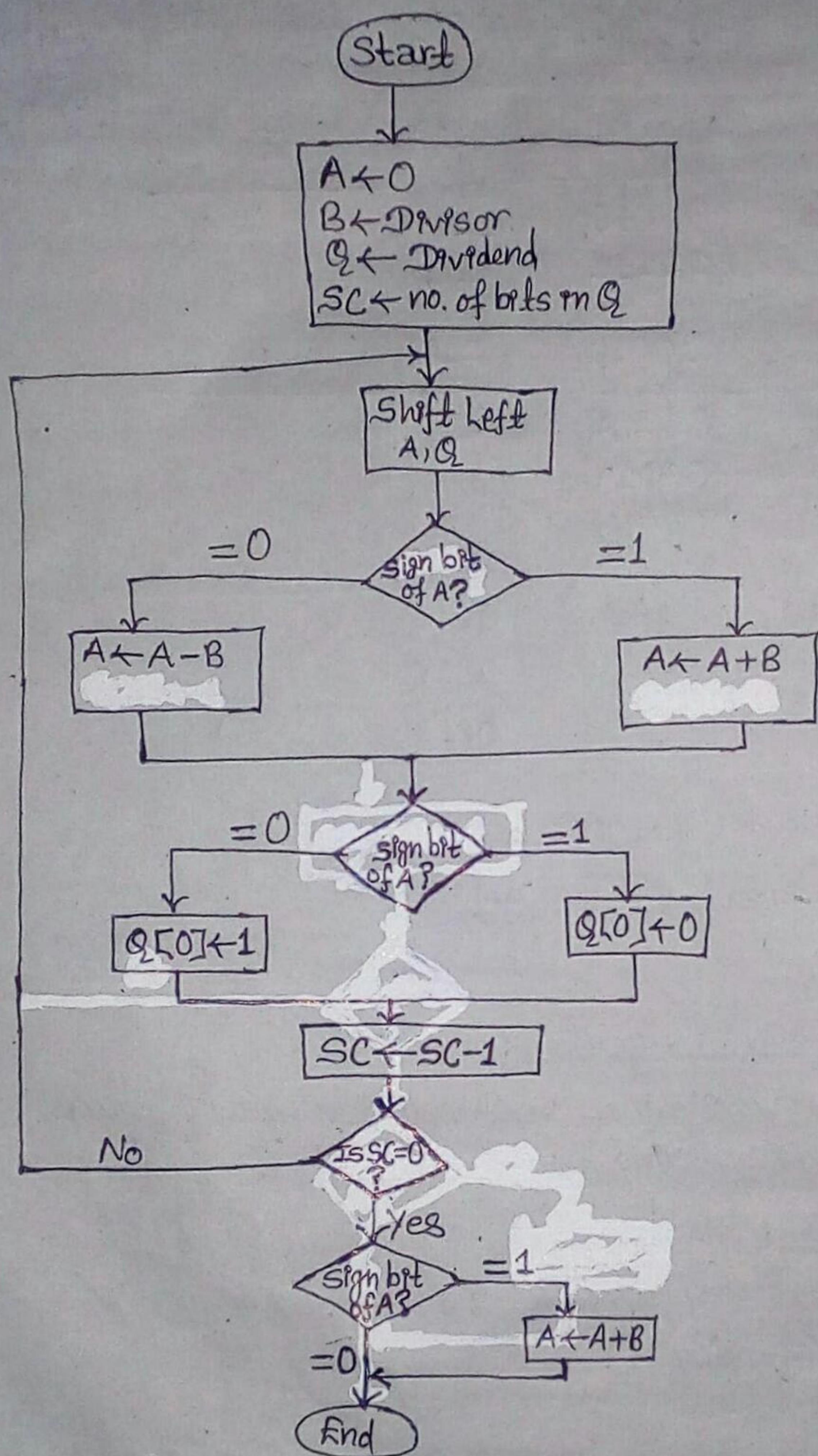
for unsigned integer

## 2) Non-Restoring division algorithm:

It is the improved version of restoring division algorithm. It provides quotient and remainder when we divide two numbers.

Hardware Implementation: Hardware implementation is exactly same to that of restoring division algorithm. So no need to draw and discuss it again.

## Flowchart:-



Quotient is in Q & Remainder is in A.

Example: Divide 11 by 3 by using non-restoring division algorithm.

Solution:

$$\text{dividend} = Q = 11 = 1011$$

$$\& \text{ divisor} = B = 3 = 00011$$

B = 00011	Action/ operation	A	Q	SC
00011	Initialization	00000	1011	4
	Shift left A, Q	00001	011?	
	$A \leftarrow A - B$	11110	011?	
	$Q[0] \leftarrow 0$	11110	0110	3
	Shift left A, Q	11100	110?	
	$A \leftarrow A + B$	11111	110?	
	$Q[0] \leftarrow 0$	11111	1100	2
	Shift left A, Q	11111	100?	
	$A \leftarrow A + B$	00010	100?	
	$Q[0] \leftarrow 1$	00010	1001	1
	Shift left A, Q	00101	001?	
	$A \leftarrow A - B$	00010	001?	
	$Q[0] \leftarrow 1$	00010	0011	0

Hence,

$$\text{Quotient} = Q = 0011 = 3$$

$$\& \text{ Remainder} = A = 00010 = 2.$$

add JTF  
 excess bit AND  
 MSB OR excess  
 bit JTF remove  
 JTF

Since here the sign bit is zero  
 so, we are directly terminating.  
 But instead if it was 1 then  
 we perform  $A \leftarrow A + B$  then  
 we terminate

Imp

Q. What is overflow? Explain overflow detection process with signed and unsigned number addition with suitable example.

Ans: When two numbers of n-digits are added and the sum occupies  $n+1$  digits, we say that an overflow has occurred. A result that contains  $n+1$  bits can't be accommodated in a register with standard length of n-bits. For this reason many computers detect the occurrence of an overflow setting corresponding flip-flop.

An overflow may occur if two numbers added are both positive or both negative. For e.g., Two signed binary no. +70 & +80 are stored in two 8-bit registers.

Carries: 0 1

$$\begin{array}{r} +70 \\ +80 \\ \hline +150 \end{array} \quad \begin{array}{r} 0\ 1000110 \\ 0\ 1010000 \\ \hline 1\ 0010110 \end{array}$$

Carries: 1 0

$$\begin{array}{r} -70 \\ -80 \\ \hline -150 \end{array} \quad \begin{array}{r} 1\ 0111010 \\ 1\ 0110000 \\ \hline 0\ 1101010 \end{array}$$

Since the sum of 150 exceeds the capacity of the register. (Since 8-bit register can store values ranging from +127 to -128), hence the overflow.

Overflow Detection → An overflow condition can be detected by observing two carry into the sign bit position and carry out of the sign bit position.

Example: Above 8-bit register, if we take the carry out of the sign bit position as a sign bit of the result 9-bit answer so obtained will be correct. Since answer can not be accommodated within 8-bits we say that an overflow occurred.

If these two carries are equal  $\Rightarrow$  no overflow

If two carries are applied to an exclusive-OR gate, an overflow will be detected when output of the gate is equal to 1.