Computer Arithmetic

Arithmetic Processor

- Arithmetic instruction in digital computers manipulate data to produce results necessary for the solution of the computational problems.
- An arithmetic processor is the part of a processor unit that execute arithmetic instruction
- An arithmetic instruction may specify binary or decimal data, and it may be represented in, Fixed point (integer or fraction) OR floating point form.
- The designer must be thoroughly familiar with sequence of steps in order to carry out the operation and achieve a correct result.

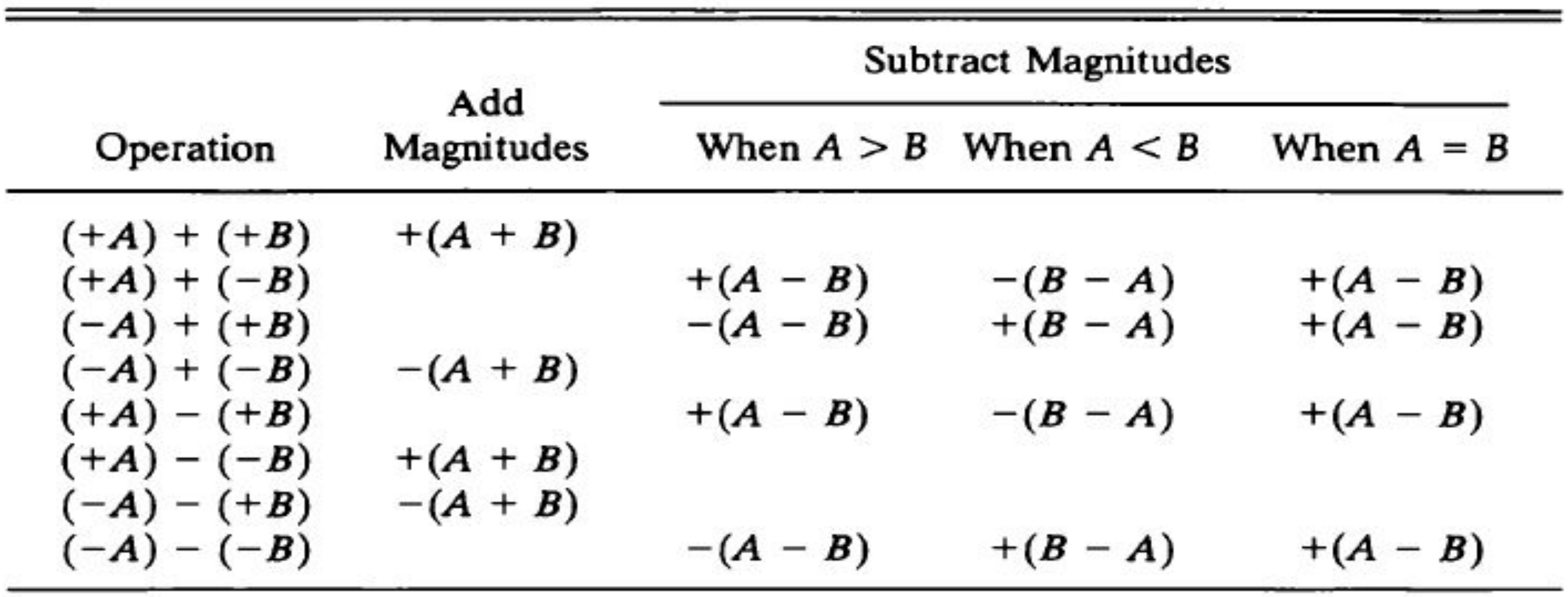
- Algorithm:
  - The solution to any problem that is stated by a finite number of well defined procedural step is called **algorithm.**
- Flowchart:
  - The convenient method for presenting algorithm is a flowchart.
  - The computational steps are specified in rectangular boxes
  - Decision steps indicated inside diamond-shaped boxes from which two or more alternate path emerge

Addition and Subtraction

- Data types considered for the arithmetic operations are,
  - Fixed-point binary data in signed magnitude representation
  - Fixed-point binary data in signed-2’s compliment representation
  - Floating point binary data
  - Binary –coded decimal (BCD) data
- Negative fixed point binary number can be represented in three ways,
  - Signed magnitude (most computers use for floating point operations)
  - Signed 1’s compliment
  - Signed 2’s compliment(most computer use for integers)

Addition and Subtraction with Signed-Magnitude Data

- Eight different conditions to consider for addition and subtraction

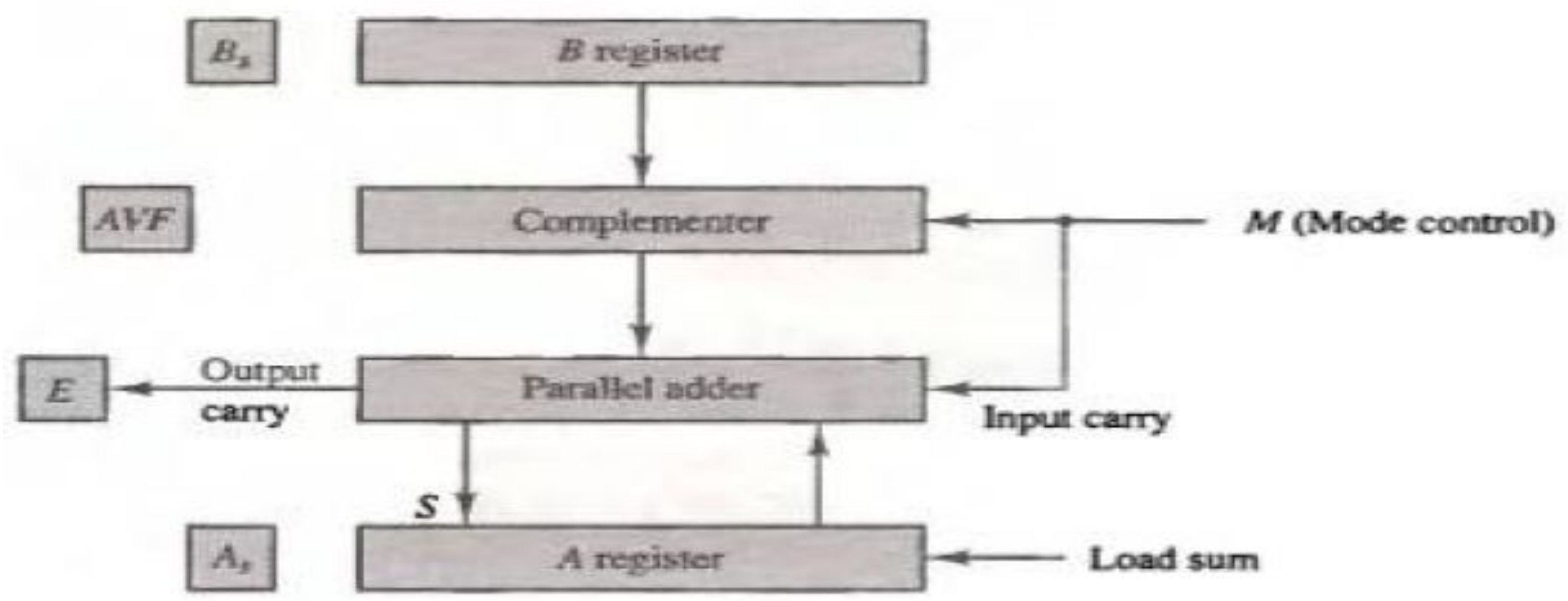


Operation	Add Magnitudes	Subtract Magnitudes		
		When $A > B$	When $A < B$	When $A = B$
$(+A) + (+B)$	$+(A + B)$			
$(+A) + (-B)$		$+(A - B)$	$-(B - A)$	$+(A - B)$
$(-A) + (+B)$		$-(A - B)$	$+(B - A)$	$+(A - B)$
$(-A) + (-B)$	$-(A + B)$			
$(+A) - (+B)$		$+(A - B)$	$-(B - A)$	$+(A - B)$
$(+A) - (-B)$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

- Addition(Subtraction) algorithm:
  - When the signs of A and B are identical(different), add the two magnitudes and attach the sign of A to the result.
  - When the sign of A and B are different(identical), compare the magnitudes and subtract the smaller number from the larger.
  - Choose the sign of the result to be same as A if A > B or the compliment of the sign of A if A < B.
  - For equal magnitude subtract B from A and make the sign of the result

Hardware Implementation

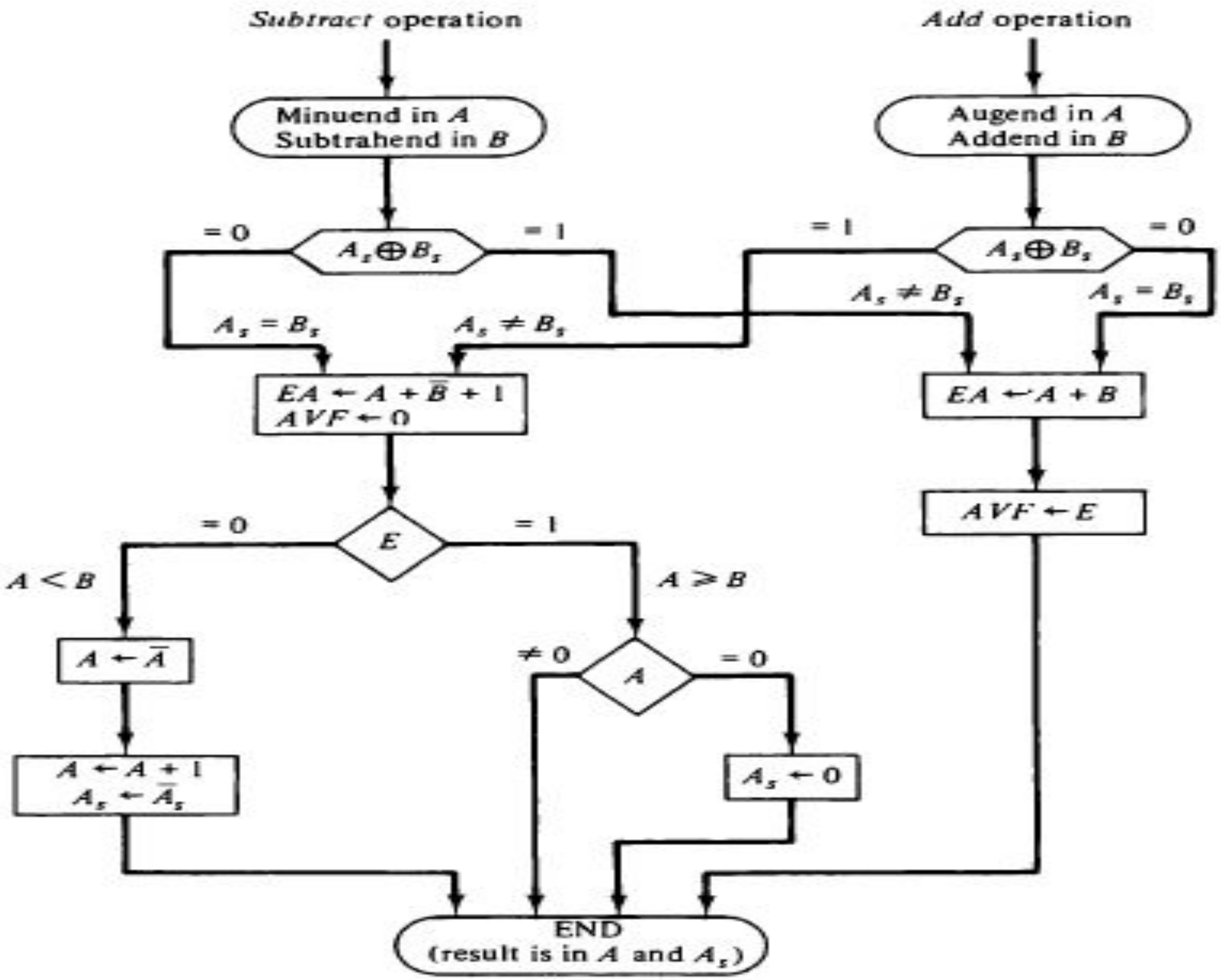
- A and B be two registers that hold the magnitudes of No
- As and Bs be two flip-flops that hold the corresponding signs
- The Result is transferred into A and As.



Cont..

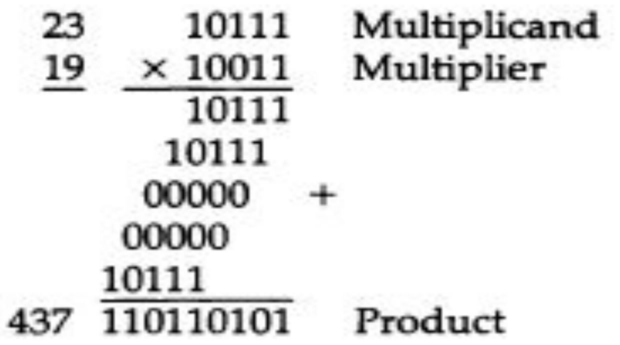
- Parallel adder is needed to perform the micro operation A + B.
- Parallel subtractor are needed to perform A – B or B – A.
  - Can be accomplished by means of compliment and add
- Comparator circuit is needed to establish if A > B, A = B or A < B
  - Comparison can be determine from the end carry after the subtraction
- The sign relationship can be determine from an exclusive-OR gate with As and Bs as inputs
- Output carry are transferred to E flip-flop
  - Where it can be checked to determine the relative magnitude of the Nos.
- Add overflow flip-flop (AVF) holds the overflow bit when A and B are added.

Hardware Algorithm



Multiplication Algorithm

- Multiplication of two fixed point binary numbers in signed magnitude representation is done by the process of successive shift and add operations.
- Numerical Example:



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

- The process consist nultiplier, least significant bit first.
- If the multiplier bit i ıerwise zeros are copied down
- Copied number in successive line shifted one position to the left from the previous number. Finally, the numbers are added and their sum forms the product.

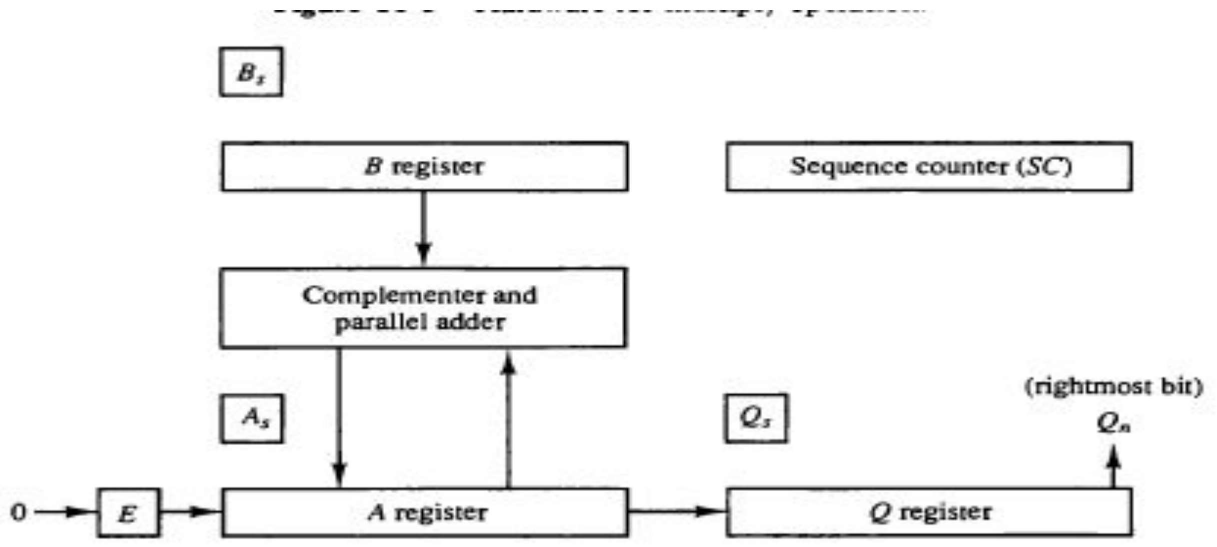
Hardware Implementation

- Multiplication process is change slightly.
- First, instead of providing registers to store and add simultaneously as many binary numbers as there are bits in multiplier,
  - It is convenient to provide an adder for the summation of two number
  - And successively accumulate the partial products in a register
- Second, instead of shifting the multiplicand to the left, the partial product is shifted right
  - Which result in leaving the partial product and the multiplicand in the required relative position
- Third, when the corresponding bit of the multiplier is 0, there is no need to add all zeros to the partial product since it will not alter its value

Cont..

- The hardware for multiplication consist of the equipment shown in figure.
- The multiplier stored in Q register and its sign in Qs.
- The sequence counter SC is initially set to a number equal to the number of bits in the multiplier.
  - The counter is decremented by 1 after forming each partial product.

Fig: (Hardware for multiply operation)



Cont..

- Initially the multiplicand is in register B and the multiplier in Q.
- The sum of A and B forms a partial product which is transferred to the EA register.
  - Both the partial product and multiplier are shifted to the right.(shr EAQ)
  - Least significant bit of A is shifted into the most significant position of Q,
  - The bit from E is shifted into the most significant position of A and 0 to E.
- After the shift, one bit of partial product is shifted into Q, pushing the multiplier bit one position to the right.
- In this manner, the rightmost flip-flop in register Q, designated by Qn, will hold the bit of the multiplier, which must be inspected next.

Hardware Algorithm

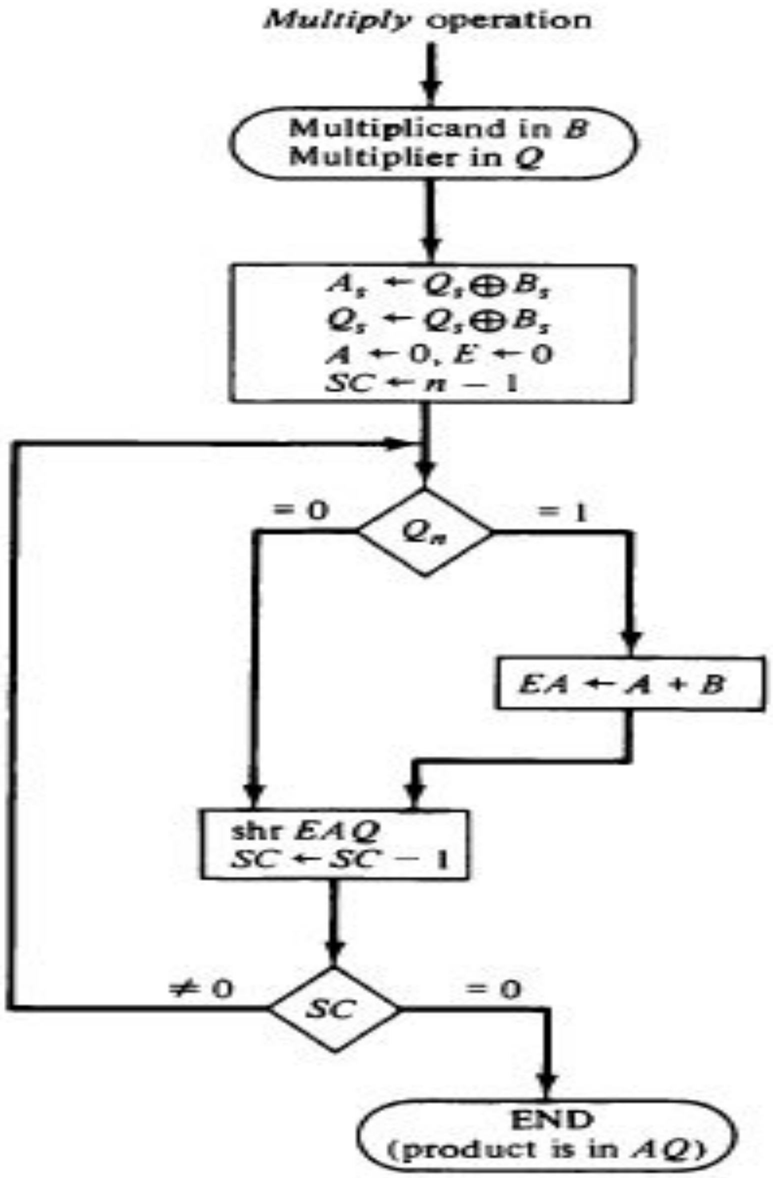
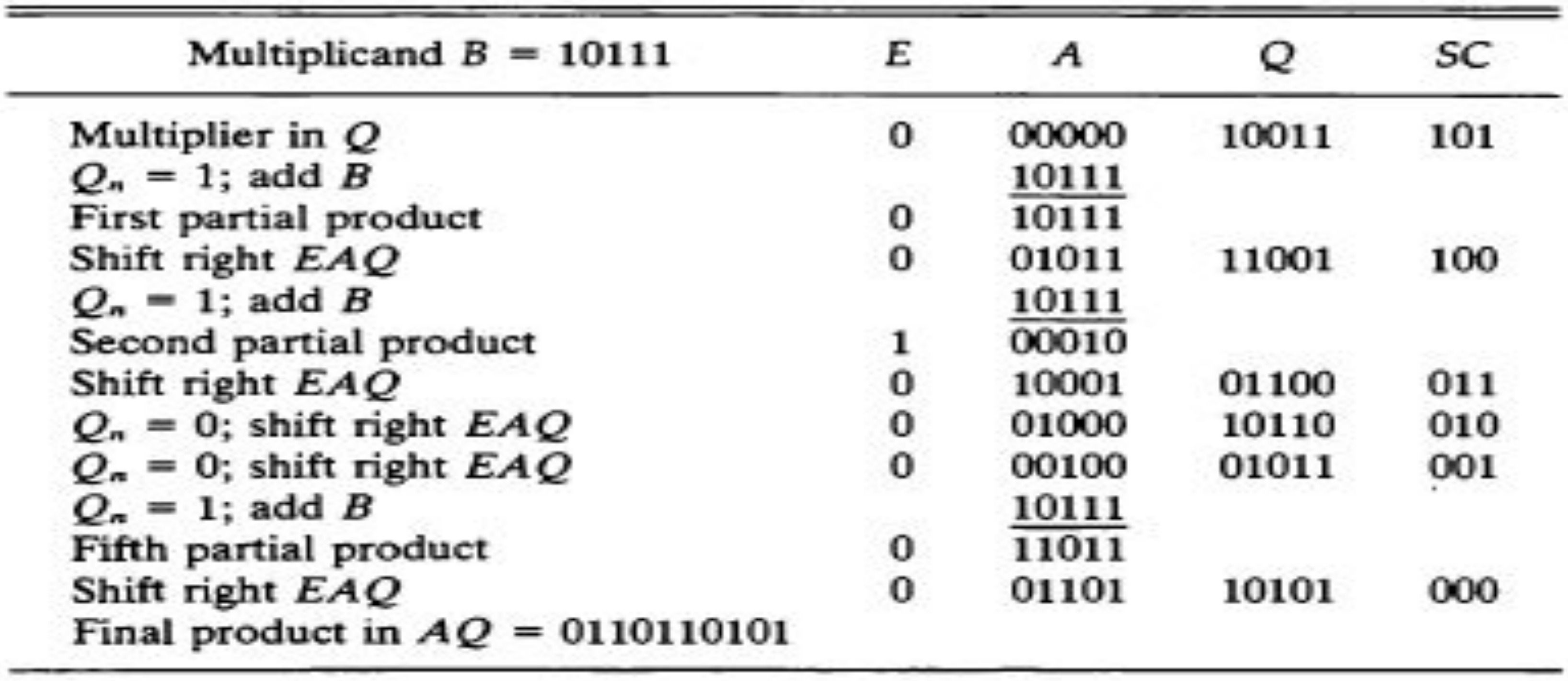
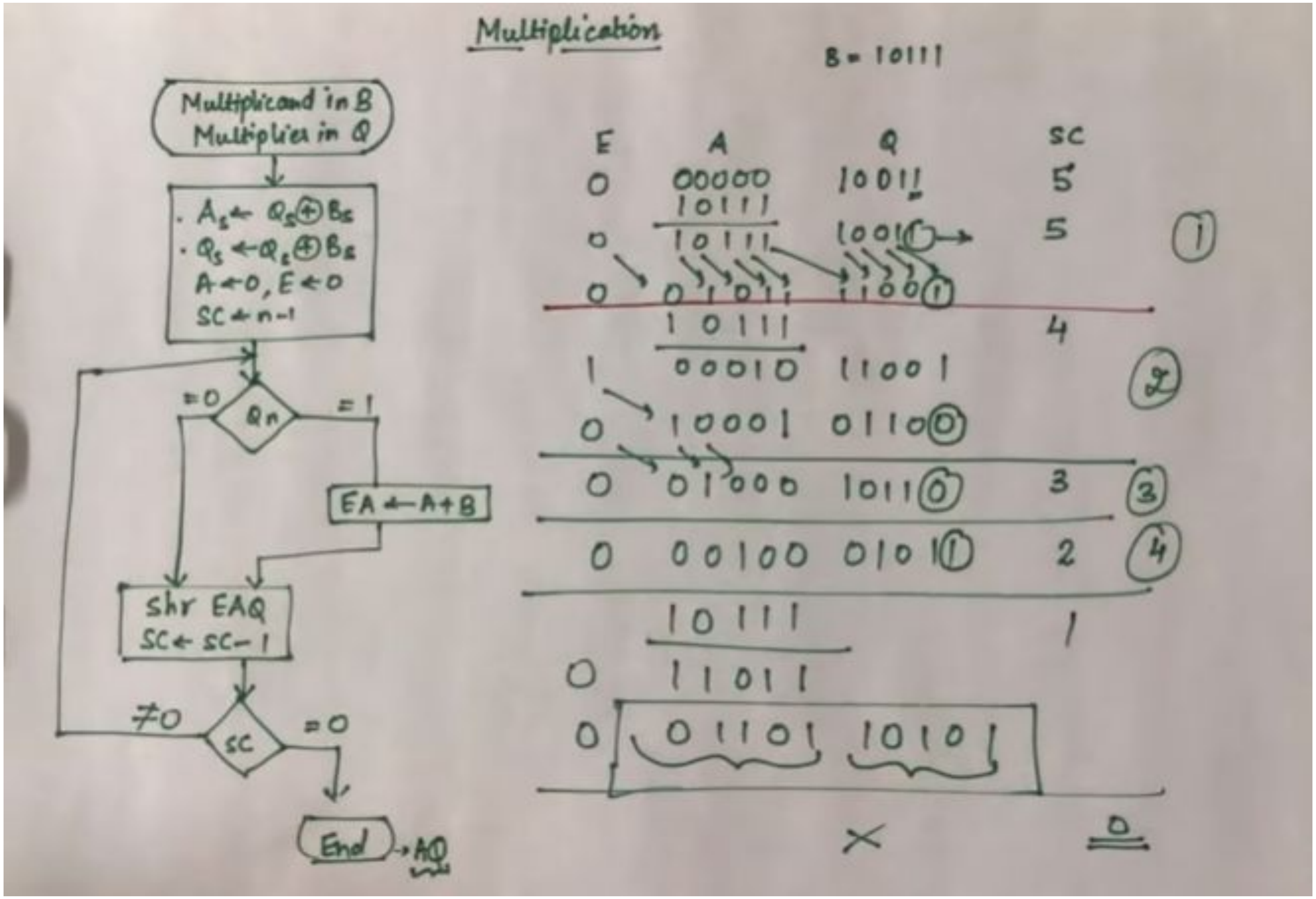


Fig: Numerical Example for binary multiplier



Multiplicand B = 10111	E	A	Q	SC
Multiplier in Q	0	00000	10011	101
$Q_n = 1$; add B		$\underline{10111}$		
First partial product	0	10111		
Shift right EAQ	0	01011	11001	100
$Q_n = 1$; add B		$\underline{10111}$		
Second partial product	1	00010		
Shift right EAQ	0	10001	01100	011
$Q_n = 0$; shift right EAQ	0	01000	10110	010
$Q_n = 0$; shift right EAQ	0	00100	01011	001
$Q_n = 1$; add B		$\underline{10111}$		
Fifth partial product	0	11011		
Shift right EAQ	0	01101	10101	000
Final product in AQ = 0110110101				

Example of multiplication algorithm



B = 10111

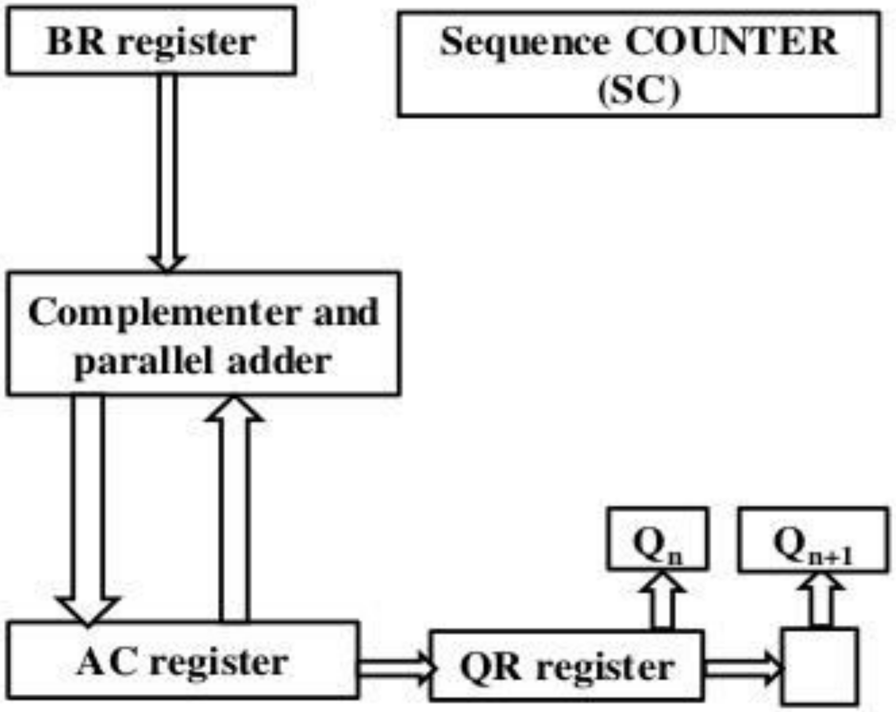
E	A	Q	SC	
0	00000	10011	5	
	10111			
0	10111	10011	5	1
0	01011	11001		
	10111		4	
1	00010	11001		2
0	10001	01100		
0	01000	10110	3	3
0	00100	01011	2	4
	10111		1	
0	11011			
0	01101	10101		

Booth’s algorithm

- Booth algorithm gives a procedure for multiplying binary integers in signed- 2’s complement representation.
- It operates on the fact that strings of 0’s in the multiplier require no addition but just shifting, and a string of 1’s in the multiplier from bit weight 2k to weight 2m can be treated as 2k+1 – 2m.
- For example, the binary number 001110 (+14) has a string 1’s from 23 to 21 (k=3, m=1). The number can be represented as 2k+1 – 2m. = 24 – 21 = 16 – 2 = 14.
- Therefore, the multiplication M X 14, where M is the multiplicand and 14 the multiplier, can be done as M X 24 – M X 21 .
- Thus the product can be obtained by shifting the binary multiplicand M four times to the left and subtracting M shifted left once.
- As in all multiplication schemes, booth algorithm requires examination of the multiplier bits and shifting of partial product.

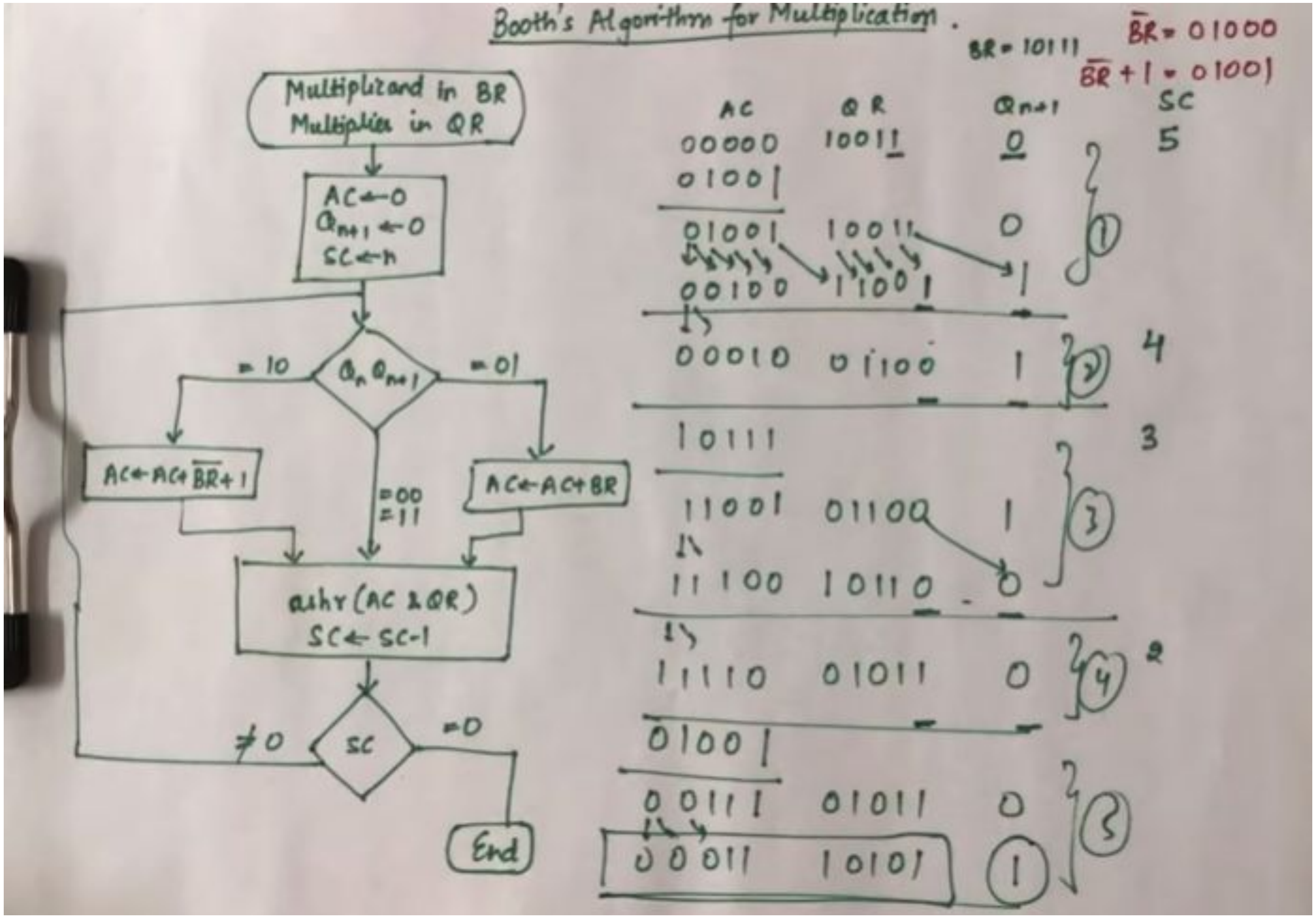
Hardware for Booth Algorithm

- Sign bits are not separated from the rest of the registers
- rename registers A,B, and Q as AC,BR and QR respectively
- Q_n designates the least significant bit of the multiplier in register QR
- Flip-flop Qn+1 is appended to QR to facilitate a double bit inspection of the multiplier



- The algorithm works for positive or negative multipliers in 2’s complement
- representation.
- This is because a negative multiplier ends with a string of 1’s and the last operation will be a subtraction of the appropriate weight.
- The two bits of the multiplier in Qn and Qn+1 are inspected.
- If the two bits are equal to 10, it means that the first 1 in a string of 1 's has been encountered. This requires a subtraction of the multiplicand from the partial product in AC.
- If the two bits are equal to 01, it means that the first 0 in a string of 0's has been encountered. This requires the addition of the multiplicand to the partial product in AC.
- When the two bits are equal, the partial product does not change.

Example of Booth multiplication algorithm



Booth's Algorithm for Multiplication.

$BR = 10111$ $\quad \overline{BR} = 01000$ $\quad \overline{BR} + 1 = 01001$

```
  AC       QR      Qn+1     SC
00000    10011      0        5
01001
-----
01001    10011      0
00100    11001      1       (1)
-----
00010    01100      1       (2)   4
-----
10111                              3
-----
11001    01100      1
11100    10110      0       (3)
-----
11110    01011      0       (4)   2
-----
01001
-----
00111    01011      0
00011    10101      1       (5)
```