Computer Arithmetic
Or
Arithmetic Logical Unit

Introduction

- Data is manipulated by using the arithmetic instructions in digital computers.
- Data is manipulated to produce results necessary to give solution for the computation problems.
- The Addition, subtraction, multiplication and division are the four basic arithmetic operations. If we want, then we can derive other operations by using these four operations.
- To execute arithmetic operations there is a separate section called arithmetic processing unit in central processing unit. The arithmetic instructions are performed generally on binary or decimal data.
- In order to solve the computational problems, arithmetic instructions are used in digital computers that manipulate data. These instructions perform arithmetic calculations.
- The four basic arithmetic operations addition, subtraction, multiplication and division, are used to derive arithmetic operations and solve scientific problems by means of numerical analysis methods.
- Negative numbers may be in a signed magnitude or signed complement representation.
- There are three ways of representing negative fixed point - binary numbers signed magnitude, signed 1's complement or signed 2's complement.

Addition and Subtraction with Signed –Magnitude Data

- We designate the magnitude of the two numbers by A and B.
- The signed numbers are added or subtracted, we find that there are eight different conditions to consider, depending on the sign of the numbers and the operation performed.
- When the signs of A and B are identical (different), add the two magnitude and attach the sign of A to the result.
- When the sign of A and B are different (identical), compare the magnitudes.

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)

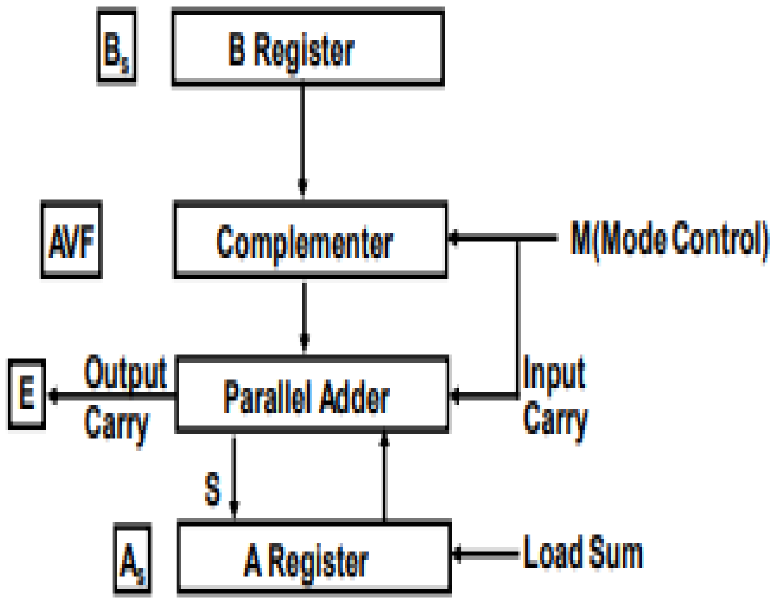
Addition: A + B ;
A: Augend; B: Addend

Subtraction: A - B:
A: Minuend; B: Subtrahend

- The last column is required to prevent negative zero. In other words, when two equal numbers are subtracted, the result should be +0 not -0.

Hardware Implementation for signed magnitude Addition and Subtraction

- To implement the two arithmetic operations with hardware, it is first required that the two numbers be stored in registers.
- Let A and B be two registers that hold the magnitude of the numbers, and A_s and B_s be two flip flops that hold the corresponding signs.
- The results of the operation may be transferred to a third register however, a saving achieved if the result is transferred into A and A_s.
- A and A_s together form an Accumulator register.
- Consider now the hardware implementation of the algorithms above.
- First, a parallel adder is needed to perform the micro-operation A+B.
- Second, comparator circuit is needed to establish if A>B, A=B, or A<B.
- Third, two parallel subtractor circuits are required to perform the micro-operation A-B and B-A.
- The sign relationship can be determined from an XOR gate with As and Bs as inputs.



- The output carry is transferred to flip-flop E.
- The complementor consists of exclusive-OR gates and the parallel adder consists of full adder circuit.

Flowchart for Hardware Implementation for Signed Addition and Subtraction

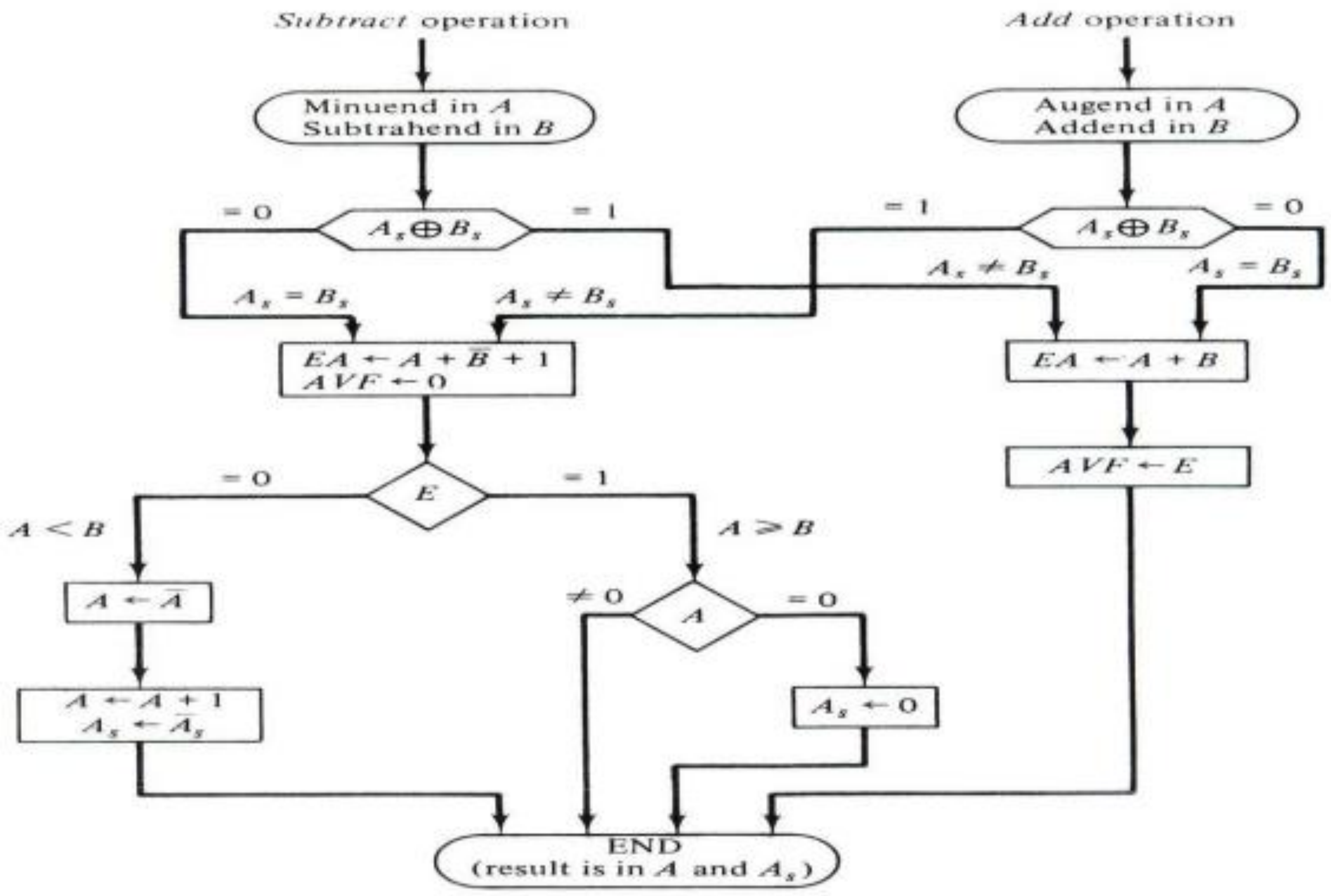


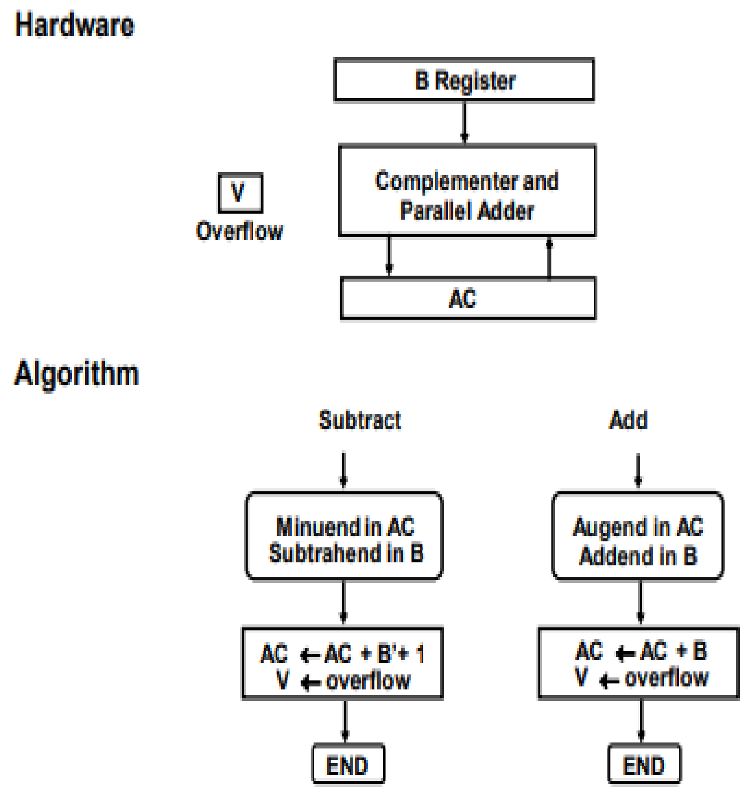
Figure 10-2 Flowchart for add and subtract operations.

Hardware Algorithm

- The flowchart is shown in Figure. The two signs A, and B, are compared by an exclusive-OR gate.
    - If the output of the gate is 0 the signs are identical;
    - If it is 1, the signs are different.
- For an add operation, identical signs dictate that the magnitudes be added. For a subtract operation, different signs dictate that the magnitudes be added.
- The magnitudes are added with a microoperation EA= A + B, where EA is a register that combines E and A. The carry in E after the addition constitutes an overflow if it is equal to 1. The value of E is transferred into the add-overflow flip-flop AVF.
- The two magnitudes are subtracted if the signs are different for an add operation or identical for a subtract operation. The magnitudes are subtracted by adding A to the 2's complemented B. No overflow can occur if the numbers are subtracted so AVF is cleared to 0.
- 1 in E indicates that A >= B and the number in A is the correct result. If this numbs is zero, the sign A must be made positive to avoid a negative zero.
- 0 in E indicates that A < B. For this case it is necessary to take the 2's complement of the value in A. The operation can be done with one microoperation A=A' +1.
- However, we assume that the A register has circuits for microoperations complement and increment, so the 2's complement is obtained from these two microoperations.
- In other paths of the flowchart, the sign of the result is the same as the sign of A. so no change in A is required. However, when A < B, the sign of the result is the complement of the original sign of A. It is then necessary to complement A, to obtain the correct sign.
- The result is found in register A and its sign in As. The value in AVF provides an overflow indication. The final value in E is immaterial.

Hardware Algorithm for Signed 2's Compliment

- Figure shows a block diagram of the hardware for implementing the addition and subtraction operations using the 2's compliment.
- It consists of registers A and B and sign flip-flops As and Bs.
- Subtraction is done by adding A to the 2's complement of B.
- The output carry is transferred to flip-flop E , where it can be checked to determine the relative magnitudes of two numbers.
- The add-overflow flip-flop AVF holds the overflow bit when A and B are added.
- The A register provides other microoperations that may be needed when we specify the sequence of steps in the algorithm.



Signed 2's compliment for addition and subtraction

Examples: (a) (+5)-(+6)

(b) (+5)-(+5)

Examples: (c) (-5)-(+6)

(d) (+6)-(-5)

Decimal Multiplication

```
(carry) 1_
        37
     x  12
     -----
        74
       370
     -----
       444
```

What are the rules?

- ❖ Successively multiply the multiplicand by each digit of the multiplier starting at the right shifting the result left by an extra left position each time keeping the first bit (LSB) as 0.
- ❖ Sum all partial results

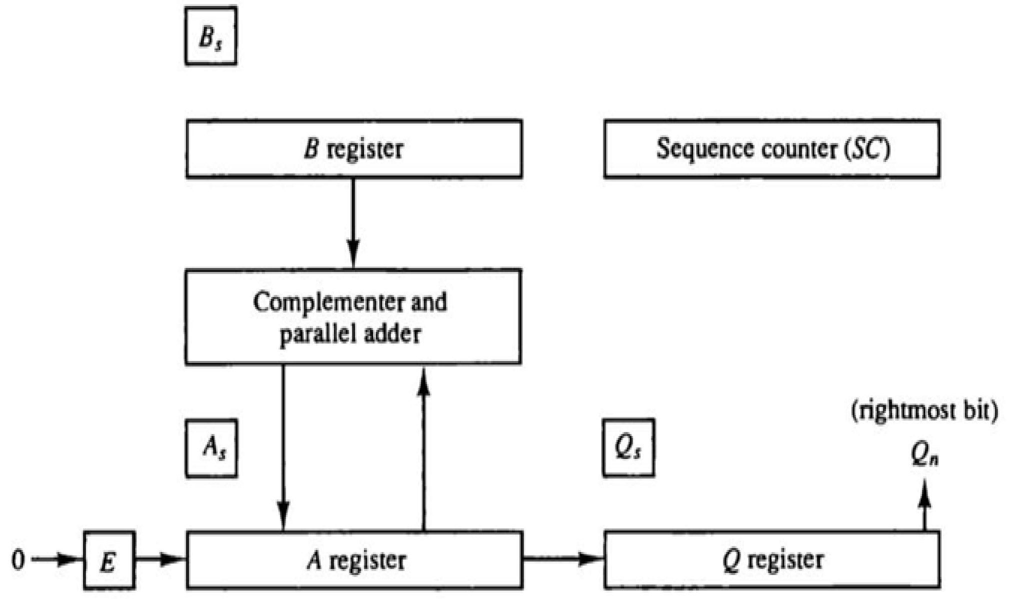
Binary Multiplication

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

- Multiplication of two fixed point binary numbers in signed magnitude representation is done with paper and pencil of successive shift and add operation.
- If the multiplier bit is a 1, the multiplicand is copied down; otherwise, zeroes are copied down.

- When multiplication is implemented in a digital computer, it is convenient to change the process slightly.
- First instead of providing register to store and add simultaneously as many binary numbers as there are bits in the multiplier, as it is convenient to provide an adder for the summation of only two binary numbers and successively accumulate the partial products in a register.
- Second instead of shifting the multiplicand to the left, the partial product is shifted to the right.

Hardware Implementation for Multiplication

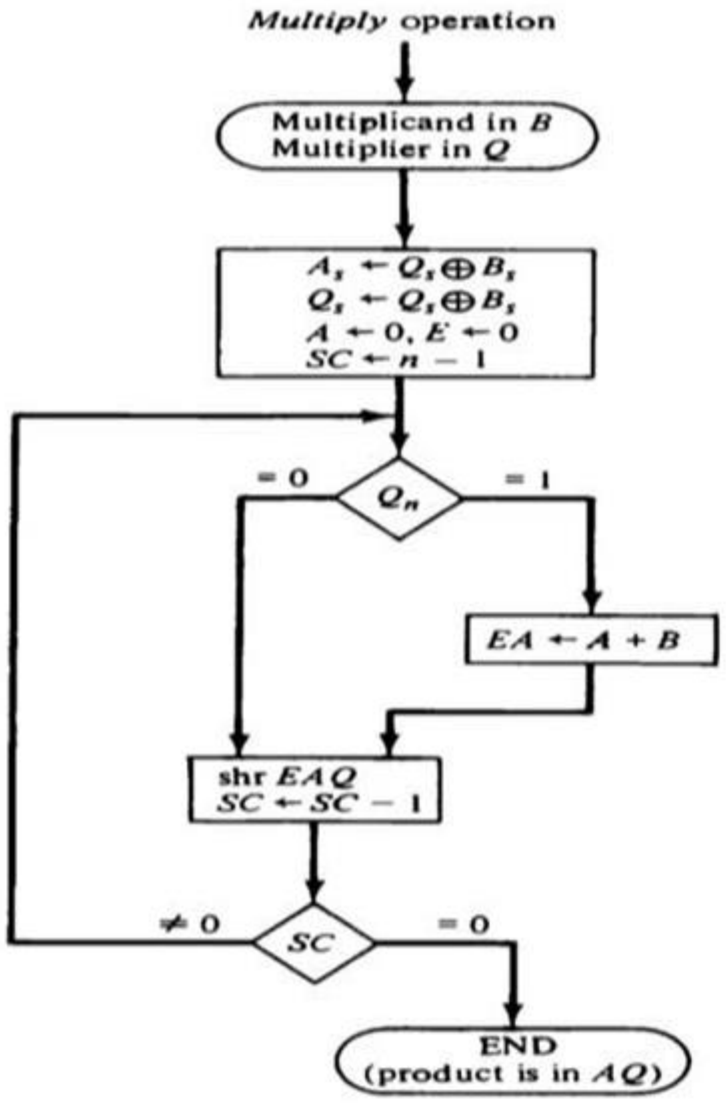


- The hardware for multiplication consists of the equipment shown in following figure plus two more registers.
- These registers are together with registers A and B.
- The multiplier stored in the Q register and its sign in Qs.
- The sequence counter SC is initially set to a number equal to the number of bits in the multiplier. The counter is decremented by 1 after forming each partial product.
- The sum of A and B forms a partial product which is transferred to the EA register .
- The shift will be denoted by the statement SHR EAQ to designate the right shift depicted.
- The least significant bit of A is shifted into the most significant position of Q.

1.Registers:
Two Registers B and Q are used to store multiplicand and multiplier, respectively. Register A is used to store partial product during multiplication. Sequence Counter register (SC) is used to store number of bits in the multiplier.
2.Flip Flop:
To store sign bit of registers we require three flip flops (A sign, B sign and Q sign). Flip flop E is used to store carry bit generated during partial product addition.
3.Complement and Parallel adder:
This hardware unit is used in calculating partial product i.e., perform addition required.

Flowchart For Binary Multiplication

Flowchart for multiply operation.

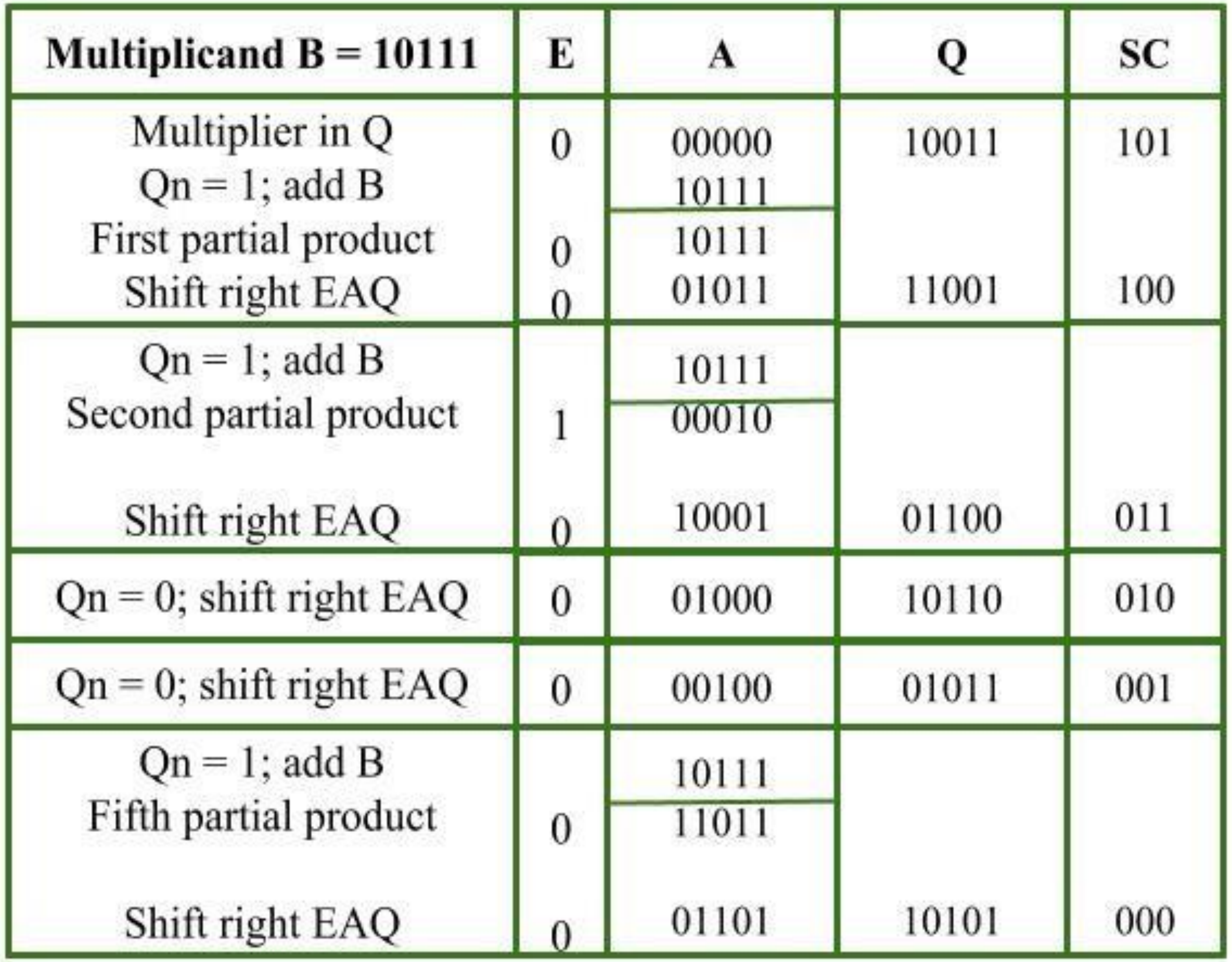


1) In the beginning, the multiplicand is in B and the multiplier in Q. Their corresponding signs are in Bs and Qs, respectively.
2) We compare the signs of both A and Q and set to corresponding sign of the product since a double-length product will be stored in registers A and Q.
3) Registers A and E are cleared, and the sequence counter SC is set to the number of bits of the multiplier.
4) Since an operand must be stored with its sign, one bit of the word will be occupied by the sign and the magnitude will consist of n-1 bits.
5) Now, the low order bit of the multiplier in Qn is tested. If it is 1, the multiplicand (B) is added to present partial product (A), 0 otherwise.
6) Register EAQ is then shifted once to the right to form the new partial product.
7) The sequence counter is decremented by 1 and its new value checked. If it is not equal to zero, the process is repeated, and a new partial product is formed. When SC = 0 we stops the process.

Example:

Multiplicand = 10111

Multiplier = 10011



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

Final product in AQ
0110110101

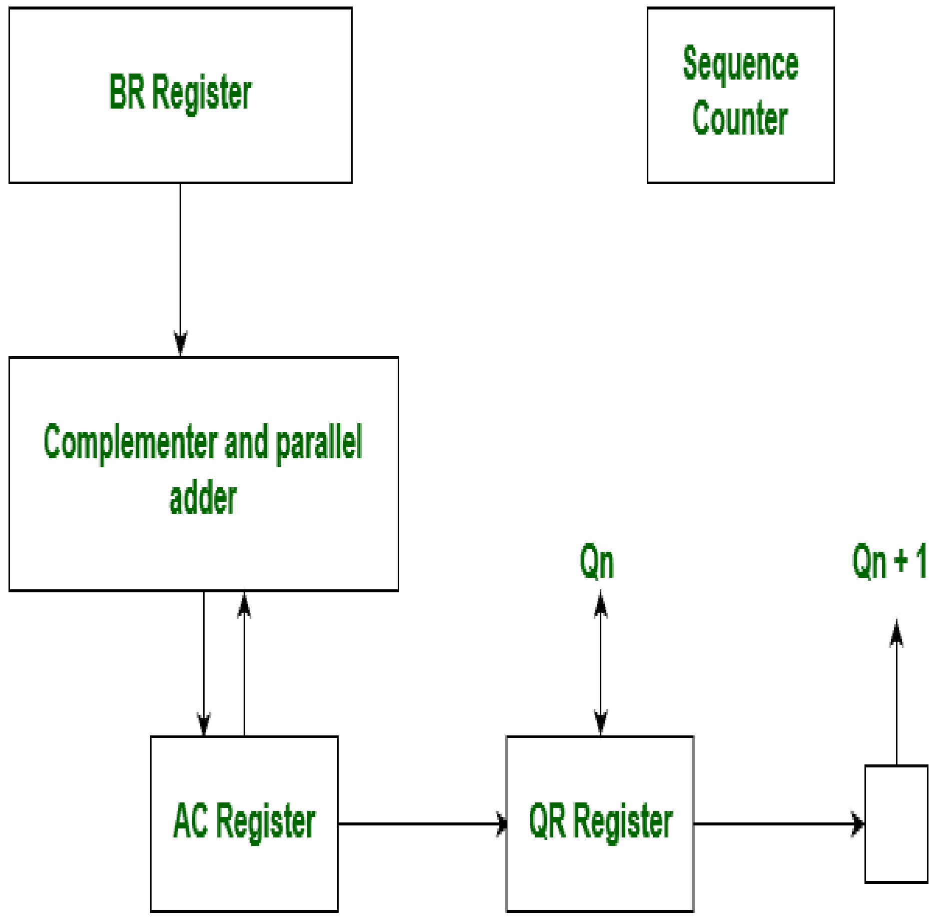
Booth’s Algorithm

1) Booth's multiplication algorithm is a multiplication algorithm that multiplies two signed binary numbers in two's complement notation.
2) It is also used to speed up the performance of the multiplication process.
3) It works on the string bits 0's in the multiplier that requires no additional bit only shift the right-most string bits and a string of 1's in a multiplier bit weight 2k to weight 2m that can be considered as 2^(k+1)-2^m
4) As in all multiplication schemes, booth algorithm requires examination **of the multiplier bits** and shifting of the partial product. Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial product, or left unchanged according to following rules:
    - The multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of 1’s in the multiplier
    - The multiplicand is added to the partial product upon encountering the first 0 (provided that there was a previous ‘1’) in a string of 0’s in the multiplier.
    - The partial product does not change when the multiplier bit is identical to the previous multiplier bit.

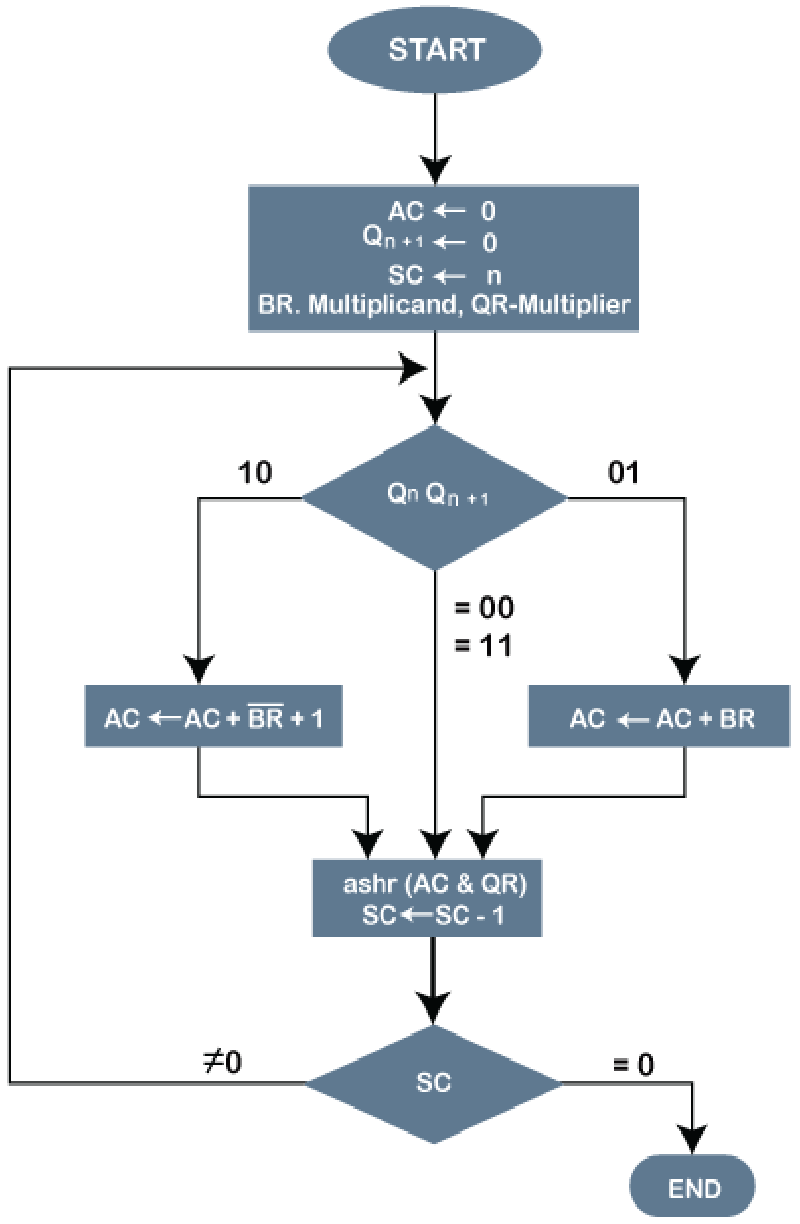
Hardware Implementation of Booths Algorithm

- The hardware implementation of the booth algorithm requires the register configuration shown in the figure below.

1) Initially, **AC** and **Q_{n+1}** bits are set to 0.
2) The **SC** is a sequence counter that represents the total bits set **n,** which is equal to the number of bits in the multiplier.
3) There are **BR** that represent the **multiplicand bits,** and QR represents the **multiplier bits**.
4) After that, we encountered two bits of the multiplier as Q_n and Q_{n+1}, where Qn represents the last bit of QR, and Q_{n+1} represents the incremented bit of Qn by 1.
5) Suppose two bits of the multiplier is equal to 10; it means that we have to subtract the multiplier from the partial product in the accumulator AC and then perform the arithmetic shift operation (ashr).
6) If the two of the multipliers equal to 01, it means we need to perform the addition of the multiplicand to the partial product in accumulator AC and then perform the arithmetic shift operation (**ashr**), including **Q_{n+1}**.
7) The arithmetic shift operation is used in Booth's algorithm to shift AC and QR bits to the right by one and remains the sign bit in AC unchanged. And the sequence counter is continuously decremented till the computational loop is repeated, equal to the number of bits (n).



Flowchart of Booth's Algorithm

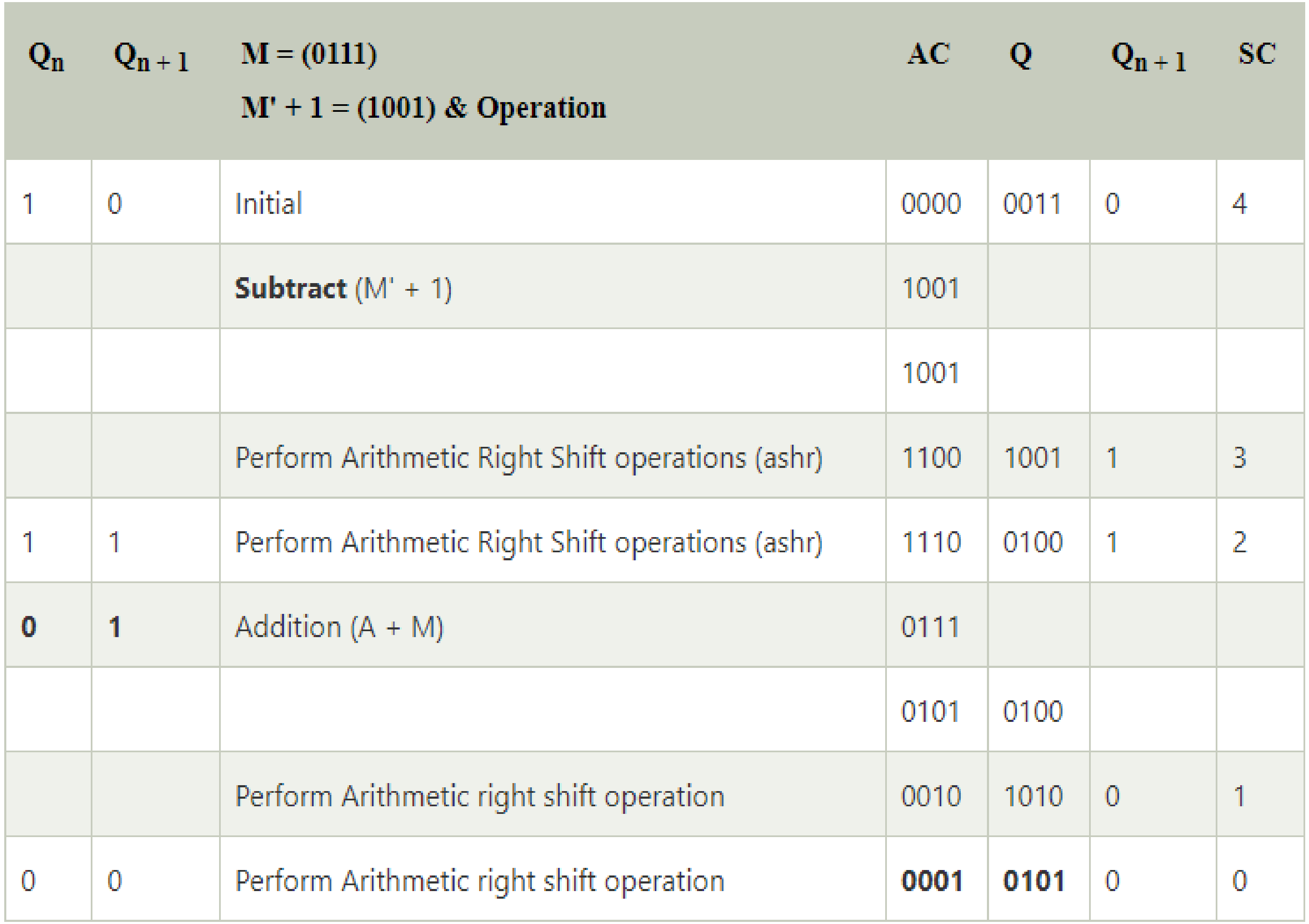


1) Set the Multiplicand and Multiplier binary bits as M and Q, respectively.
2) Initially, we set the AC and Q_{n+1} registers value to 0.
3) SC represents the number of Multiplier bits (Q), and it is a sequence counter that is continuously decremented till equal to the number of bits (n) or reached to 0.
4) A Qn represents the last bit of the Q, and the Q_{n+1} shows the incremented bit of Qn by 1.
5) On each cycle of the booth algorithm, Q_n and Q_{n+1} bits will be checked on the following parameters as follows:
   i. When two bits Q_n and Q_{n+1} are 00 or 11, we simply perform the arithmetic shift right operation (ashr) to the partial product AC. And the bits of Qn and Q_{n+1} is incremented by 1 bit.
   ii. If the bits of Q_n and Q_{n+1} is shows to 01, the multiplicand bits (M) will be added to the AC (Accumulator register). After that, we perform the right shift operation to the AC and QR bits by 1.
   iii. If the bits of Q_n and Q_{n+1} is shows to 10, the multiplicand bits (M) will be subtracted from the AC (Accumulator register). After that, we perform the right shift operation to the AC and QR bits by 1.
6) The operation continuously works till we reached n - 1 bit in the booth algorithm.
7) Results of the Multiplication binary bits will be stored in the AC and QR registers.

Example 1: Multiply the two numbers 7 and 3 by using the Booth's multiplication algorithm.

- Here we have two numbers, 7 and 3. First of all, we need to convert 7 and 3 into binary numbers like 7 = (0111) and 3 = (0011). Now set 7 (in binary 0111) as multiplicand (M) and 3 (in binary 0011) as a multiplier (Q). And SC (Sequence Count) represents the number of bits, and here we have 4 bits, so set the SC = 4. Also, it shows the number of iteration cycles of the booth's algorithms and then cycles run SC = SC - 1 time.

- The numerical example of the Booth's Multiplication Algorithm is 7 x 3 = 21 and the binary representation of 21 is 10101. Here, we get the resultant in binary 00010101. Now we convert it into decimal, as $(000010101)_{10}$ = 2*4 + 2*3 + 2*2 + 2*1 + 2*0 => 21.

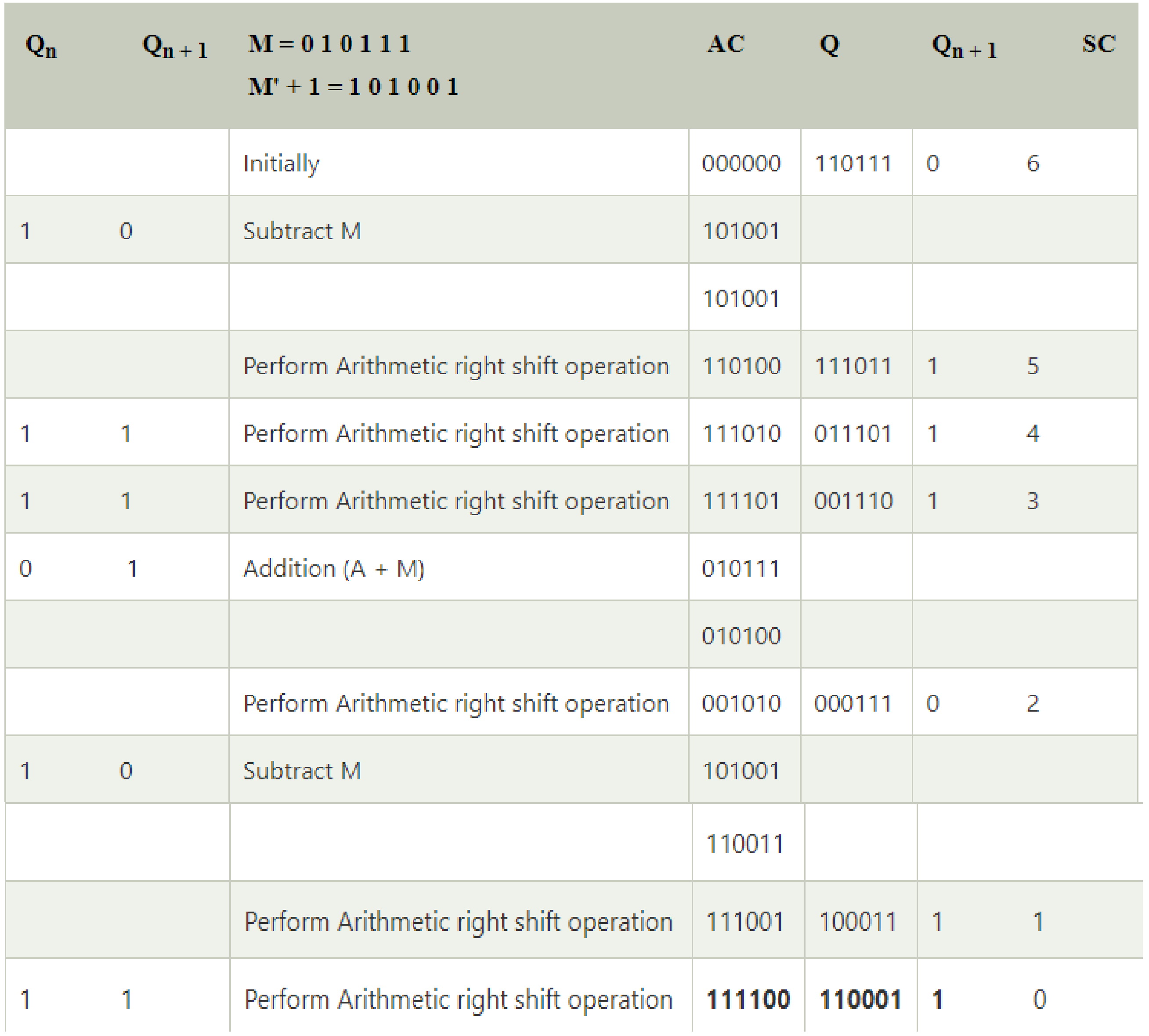


Q_n	Q_{n+1}	M = (0111) M' + 1 = (1001) & Operation	AC	Q	Q_{n+1}	SC
1	0	Initial	0000	0011	0	4
		Subtract (M' + 1)	1001			
			1001			
		Perform Arithmetic Right Shift operations (ashr)	1100	1001	1	3
1	1	Perform Arithmetic Right Shift operations (ashr)	1110	0100	1	2
0	**1**	Addition (A + M)	0111			
			0101	0100		
		Perform Arithmetic right shift operation	0010	1010	0	1
0	0	Perform Arithmetic right shift operation	**0001**	**0101**	0	0

Example 2: Multiply the two numbers 23 and -9 by using the Booth's multiplication algorithm.

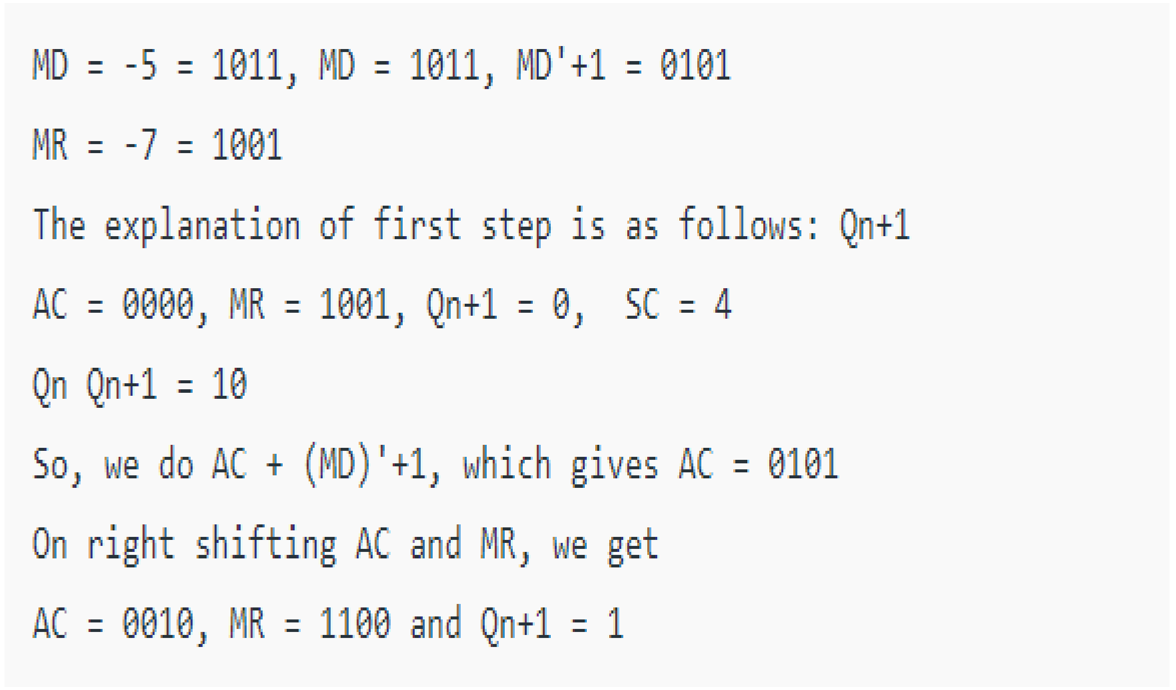
Here, M = 23 = (010111) and Q = -9 = (110111)

Q_{n+1} = 1, it means the output is negative.
Hence, 23 * -9 = 2's complement of 111100110001 => **(00001100111)**

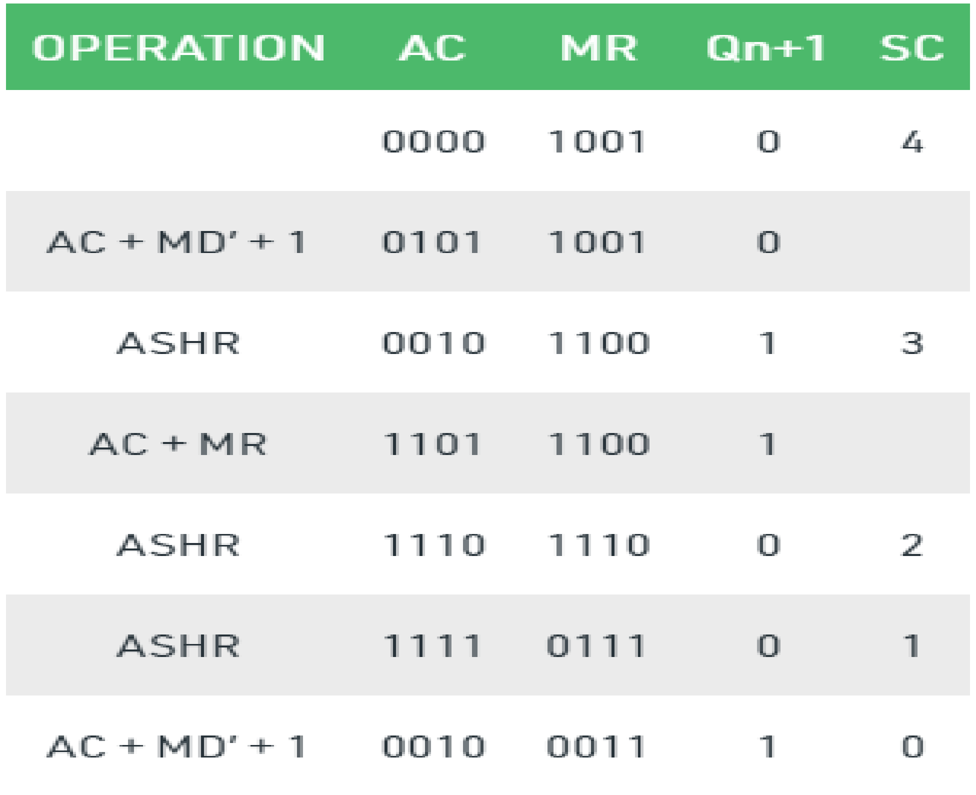


Q_n	Q_{n+1}	M = 0 1 0 1 1 1 M' + 1 = 1 0 1 0 0 1	AC	Q	Q_{n+1}	SC
		Initially	000000	110111	0	6
1	0	Subtract M	101001			
			101001			
		Perform Arithmetic right shift operation	110100	111011	1	5
1	1	Perform Arithmetic right shift operation	111010	011101	1	4
1	1	Perform Arithmetic right shift operation	111101	001110	1	3
0	1	Addition (A + M)	010111			
			010100			
		Perform Arithmetic right shift operation	001010	000111	0	2
1	0	Subtract M	101001			
			110011			
		Perform Arithmetic right shift operation	111001	100011	1	1
1	1	Perform Arithmetic right shift operation	**111100**	**110001**	**1**	0

Example – A numerical example of booth's algorithm is shown below for n = 4. It shows the step-by-step multiplication of -5 and -7.

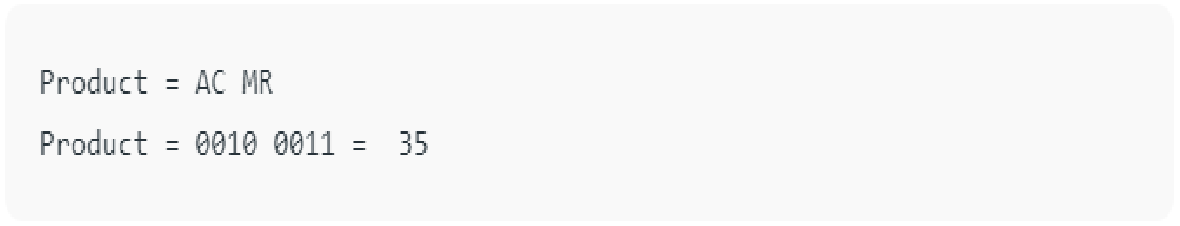


```
MD = -5 = 1011, MD = 1011, MD'+1 = 0101
MR = -7 = 1001
The explanation of first step is as follows: Qn+1
AC = 0000, MR = 1001, Qn+1 = 0,  SC = 4
Qn Qn+1 = 10
So, we do AC + (MD)'+1, which gives AC = 0101
On right shifting AC and MR, we get
AC = 0010, MR = 1100 and Qn+1 = 1
```



OPERATION	AC	MR	Qn+1	SC
	0000	1001	0	4
AC + MD' + 1	0101	1001	0	
ASHR	0010	1100	1	3
AC + MR	1101	1100	1	
ASHR	1110	1110	0	2
ASHR	1111	0111	0	1
AC + MD' + 1	0010	0011	1	0

Product is calculated as follows:



```
Product = AC MR
Product = 0010 0011 =  35
```