Booth’s Algorithm

By Probir Mondal

Introduction

Booth's Algorithm is an efficient method for performing binary multiplication. The algorithm is named after Andrew Donald Booth, who developed it in 1950.

Binary multiplication involves multiplying two binary numbers, represented as sequences of bits, to produce a result. The basic idea behind Booth's Algorithm is to encode partial products and accumulate them to reduce the number of operations required. The algorithm uses a clever encoding scheme to represent partial products as 2-bit values that can be added or subtracted to generate the final result.

Advantages

Efficient use of hardware: Booth's Algorithm requires fewer operations than other binary multiplication algorithms, making it an efficient use of hardware resources.

High speed: The algorithm is designed to reduce the number of operations required to perform binary multiplication, resulting in high-speed performance.

Low power consumption: Booth's Algorithm requires fewer operations than other binary multiplication algorithms, reducing the power consumption of digital systems.

Simplicity: The algorithm is simple to implement and does not require complex hardware structures, making it a popular choice for digital systems.

Versatility: Booth's Algorithm can be used to multiply binary numbers of any size, making it a versatile tool in the design of digital systems.

Widely used: The algorithm is widely used in computer systems and embedded systems, making it a well-established and well-understood tool in the field of digital design.

Example

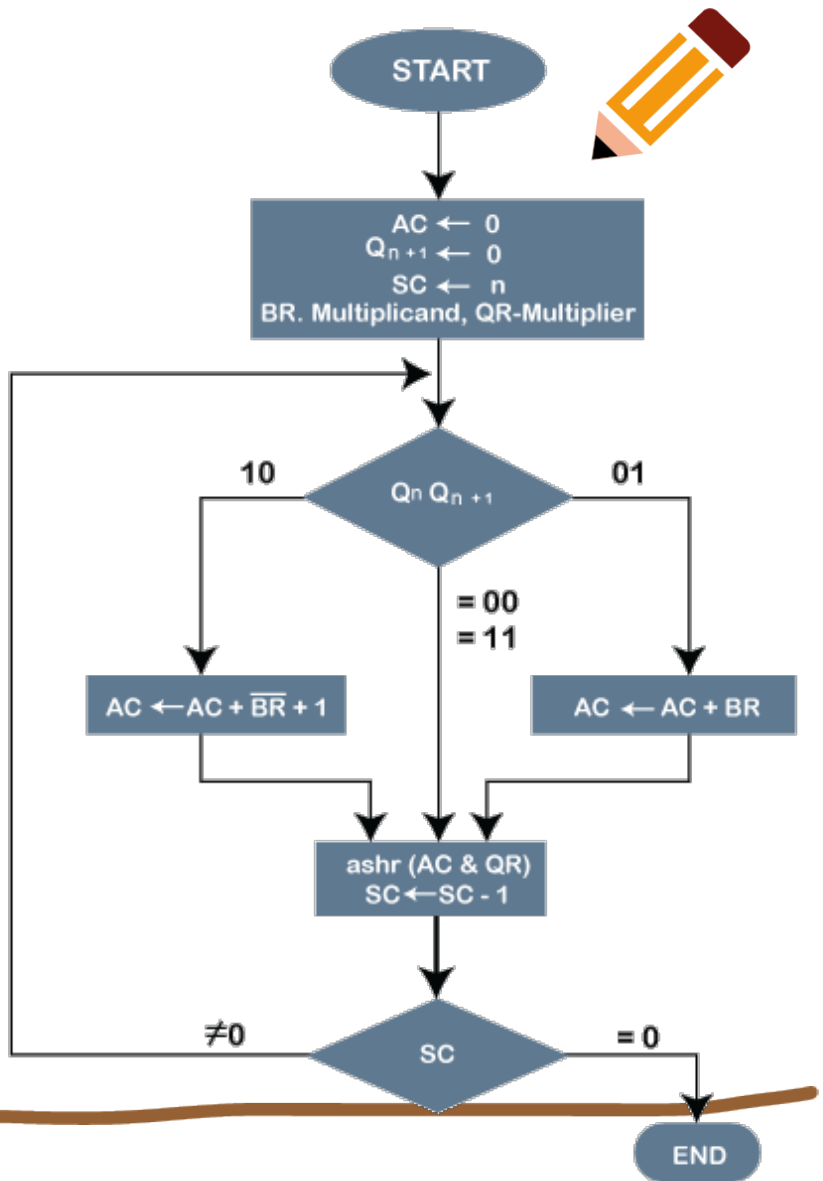
Multiply the two numbers 7 and 3 by using the Booth's multiplication algorithm.

Ans. 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. There are **BR** that represent the multiplicand bits, and **QR** represents the multiplier bits.

Results of the Multiplication binary bits will be stored in the AC and QR registers.

If Qn + 1 = 1, it means the output is negative.

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



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.

Example 2

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

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

Note:

Qn + 1 = 1, it means the output is negative.

Hence, 23 * -9 = 2's complement of 111100110001 => (00001100111)

Disadvantages

Complexity: While the algorithm is relatively simple, it still requires a certain level of complexity compared to other binary multiplication algorithms.

Implementation difficulty: The algorithm requires specific hardware to be implemented effectively, making it more difficult to implement in some cases.

Limited precision: The algorithm is designed for use with binary numbers, which have limited precision compared to floating-point numbers. This can result in inaccuracies in some cases.

Increased hardware requirements: The algorithm requires more hardware resources than other binary multiplication algorithms, making it less suitable for some embedded systems with limited resources.

Limited applicability: While the algorithm is widely used in computer systems and embedded systems, it may not be suitable for all applications and may not provide the desired level of performance in all cases.

Incompatibility with certain architectures: The algorithm may not be compatible with certain processor architectures, making it less suitable for some systems.