Booth's Multiplication

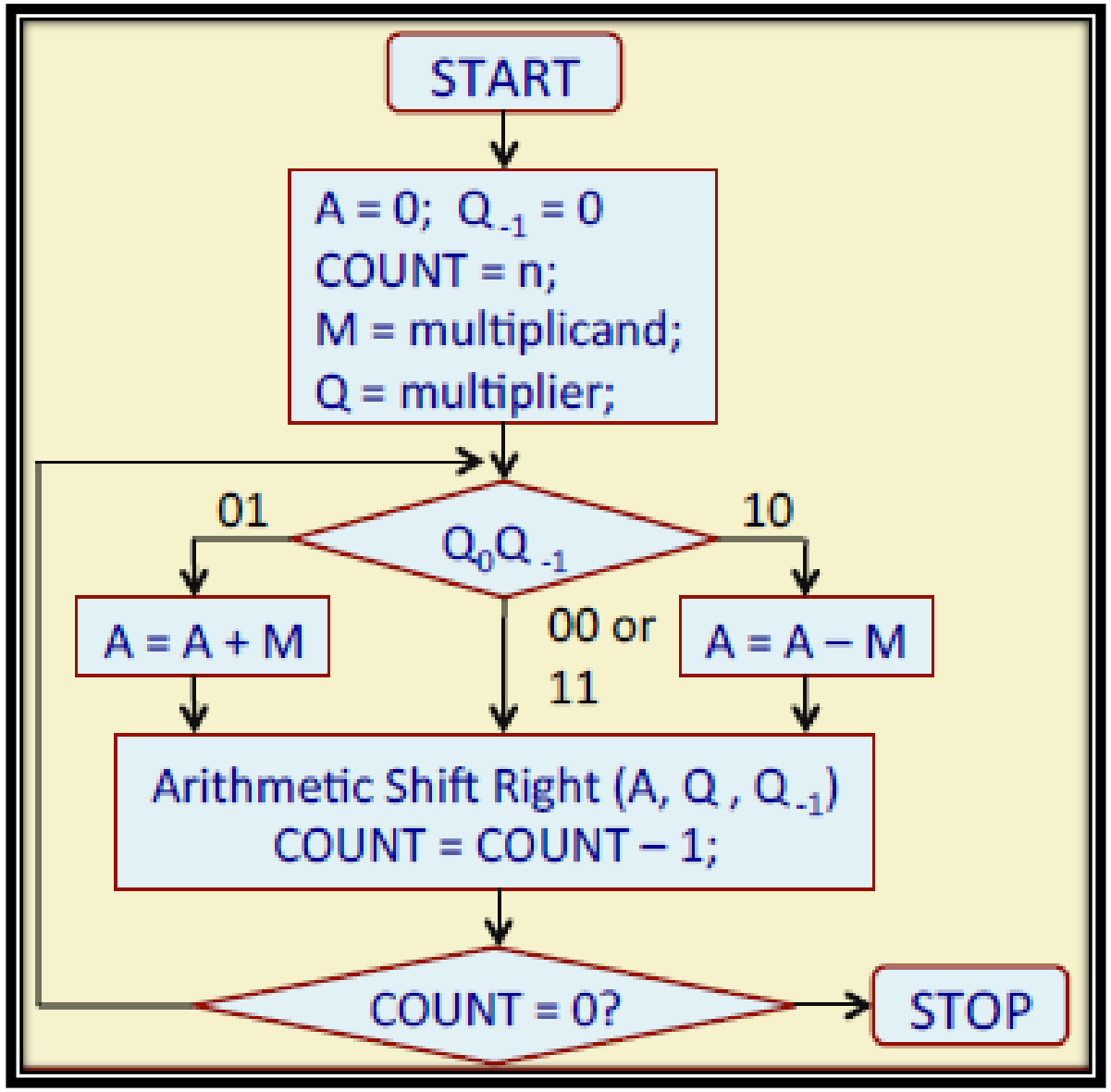
- In the conventional shift-and-add multiplication, for n-bit multiplication, we iterate n times.

- Essentially we need n addition and n shift operation

- **Booth's algorithm** is an improvement whereby we can avoid the addition whenever consecutive 0's or 1's are detected in the multiplier. Makes the **process faster**.

Basic Idea Behind Booth's Algorithm

- We inspect two bits of the multiplier (Qi, Qi-1) at a time.
  - ✓ If the bits are same (00 or 11), we only shift the partial product.
  - ✓ If the bits are 01, we do an addition and then shift.
  - ✓ If the bits are 10, we do a subtraction and then shift.
- Q -1 is assumed to be equal to 0.
- Significantly reduces the number of additions / subtractions.



M: n-bit multiplicand

Q: n-bit multiplier

A: n-bit temporary register

Q_{-1}: 1-bit flip-flop

Example 1: (-10) x (13)

Assume 5-bit numbers.

M: $(1\,0\,1\,1\,0)_2$
-M: $(0\,1\,0\,1\,0)_2$
Q: $(0\,1\,1\,0\,1)_2$

Product = -130
= $(1\,1\,0\,1\,1\,1\,1\,1\,1\,0)_2$

A	Q	Q_{-1}		
0 0 0 0 0	0 1 1 0	1	Initialization	
0 1 0 1 0	0 1 1 0	1	A = A - M	Step 1
0 0 1 0 1	0 0 1 1	0	Shift	
1 1 0 1 1	0 0 1 1	0	A = A + M	Step 2
1 1 1 0 1	1 0 0 1	1	Shift	
0 0 1 1 1	1 0 0 1	1	A = A - M	Step 3
0 0 0 1 1	1 1 0 0	1	Shift	
0 0 0 0 1	1 1 1 1	0	Shift	Step 4
1 0 1 1 1	1 1 1 0	0	A = A + M	Step 5
1 1 0 1 1	1 1 1 1	0	Shift	

Example 2:

(-31) x (28)

Assume 6-bit numbers.

M: $(1\,0\,0\,0\,0\,1)_2$

-M: $(0\,1\,1\,1\,1\,1)_2$

Q: $(0\,1\,1\,1\,0\,0)_2$

Product = -868

= $(1\,1\,0\,0\,1\,0\;\;0\,1\,1\,1\,0\,0)_2$

	A	Q	Q_{-1}		
	0 0 0 0 0 0	0 1 1 1 0	0	Initialization	
Step 1	0 0 0 0 0 0	0 0 1 1 1	0	Shift	
Step 2	0 0 0 0 0 0	0 0 0 1 1	1	Shift	
Step 3	0 1 1 1 1 1	0 0 0 1 1	1	A = A - M	
	0 0 1 1 1 1	1 0 0 0 1	1	Shift	
Step 4	0 0 0 1 1 1	1 1 0 0 0	1	Shift	
Step 5	0 0 0 0 1 1	1 1 1 0 0	0	Shift	
Step 6	1 0 0 1 0 0	1 1 1 0 0 0	1	A = A + M	
	1 1 0 0 1 0	0 1 1 1 0 0	0	Shift	