Booth’s Multiplication

COA Lab

30.08.2023

Signed Multiplication

- Basic shift-and-add multiplication can be extended to handle signed numbers.
- Required to sign-extend all the partial products before they are added.
  - For 2’s complement representation, sign extension can be done by replicating the sign bit any number of times.

0101 = 0000 0101 = 0000 0000 0000 0101 = 0000 0000 0000 0000 0000 0000 0000 0101

1011 = 1111 1011 = 1111 1111 1111 1011 = 1111 1111 1111 1111 1111 1111 1111 1011

An Example: 6-bit 2's complement multiplication

Note: For n-bit multiplication, since we are generating a 2n-bit product, overflow can never occur.

```
             1 1 0 1 0 1     (-11)
           x 0 1 1 0 1 0     (+26)
-------------------------
0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 0 1 0 1
0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 0 1 0 1
1 1 1 1 0 1 0 1
0 0 0 0 0 0 0
-------------------------
1 1 1 0 1 1 1 0 0 0 1 0    (-286)
```

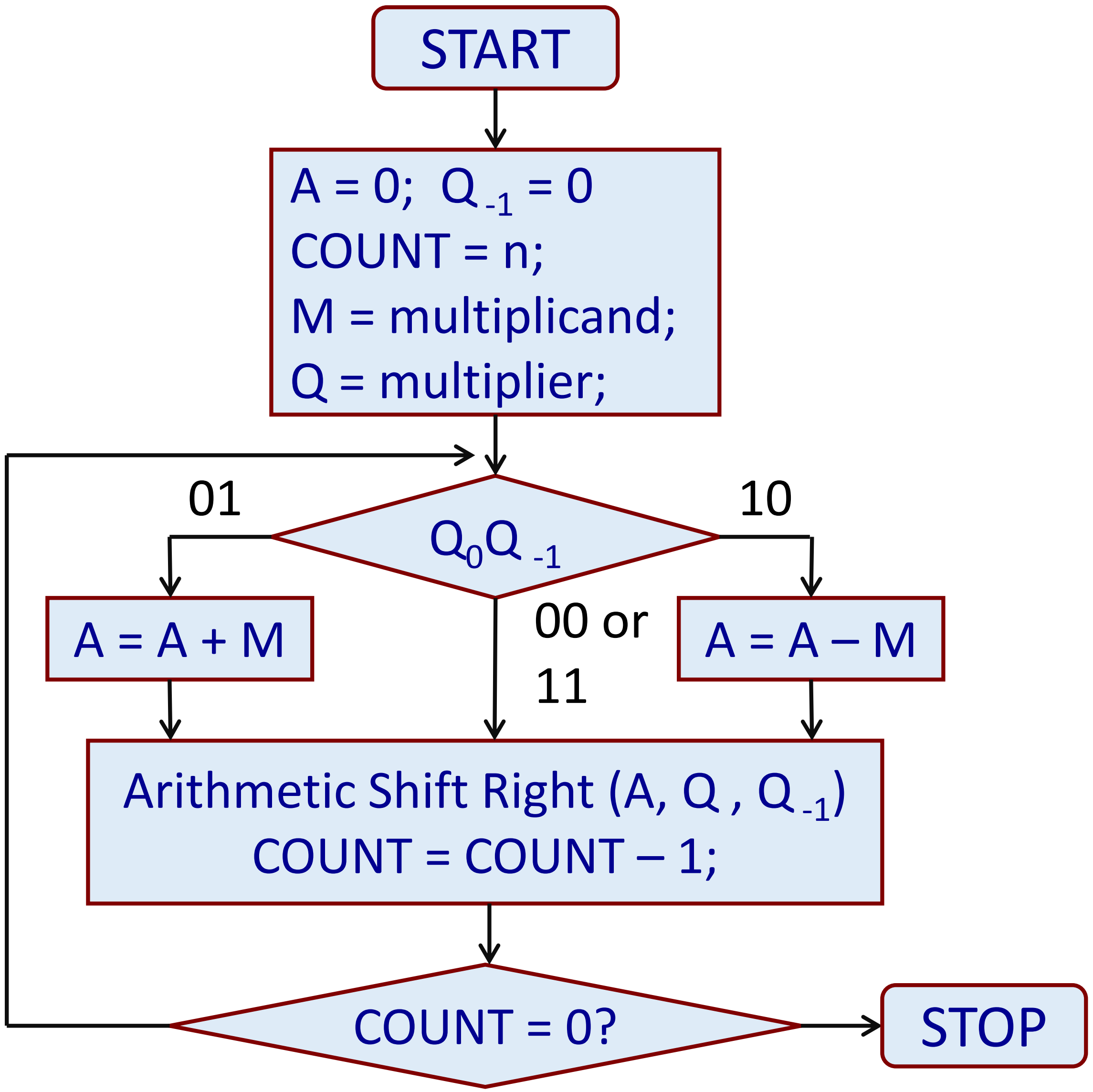
Booth’s Algorithm for Signed Multiplication

- In the conventional shift-and-add multiplication as discussed, for n-bit multiplication, we iterate n times.
  - Add either 0 or the multiplicand to the 2n-bit partial product (depending on the next bit of the multiplier).
  - Shift the 2n-bit partial product to the right.
- Essentially we need *n additions and n shift operations*.
- Booth’s algorithm is an improvement whereby we can avoid the additions 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 (Q_i, Q_{i-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 right shift.
  - If the bits are 10, we do a subtraction and then right shift.
- Significantly reduces the number of additions / subtractions.
- Inspecting bit pairs as mentioned can also be expressed in terms of *Booth's Encoding*.
  - Use the symbols +1, -1 and 0 to indicate changes w.r.t. Q_i and Q_{i-1}.
  - 01 → +1, 10 → -1, 00 or 11 → 0.
  - For encoding the least significant bit Q_0, we assume $Q_{-1} = 0$.

- Examples of Booth encoding:
  a) 0 1 1 1 0 0 0 0 :: +1 0 0 -1 0 0 0 0
  b) 0 1 1 1 0 1 1 0 :: +1 0 0 -1 +1 0 -1 0
  c) 0 0 0 0 0 1 1 1 :: 0 0 0 0 +1 0 0 -1
  d) 0 1 0 1 0 1 0 1 :: +1 -1 +1 -1 +1 -1 +1 -1
- The last example illustrates the worst case for Booth’s multiplication (alternating 0’s and 1’s in multiplier).
  - In the illustrations, we shall show the two multiplier bits explicitly instead of showing the encoded digits.



M: n-bit multiplicand

Q: n-bit multiplier

A: n-bit temporary register

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

Skips over consecutive 0's and 1's of the multiplier Q.

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	0	Initialization	
0 1 0 1 0	0 1 1 0 1	0	A = A - M	Step 1
0 0 1 0 1	0 0 1 1 0	1	Shift	
1 1 0 1 1	0 0 1 1 0	1	A = A + M	Step 2
1 1 1 0 1	1 0 0 1 1	0	Shift	
0 0 1 1 1	1 0 0 1 1	0	A = A - M	Step 3
0 0 0 1 1	1 1 0 0 1	1	Shift	
0 0 0 0 1	1 1 1 1 0	1	Shift	Step 4
1 0 1 1 1	1 1 1 0 0	1	A = A + M	Step 5
1 1 0 1 1	1 1 1 1 0	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	0	Initialization	
0 0 0 0 0 0	0 0 1 1 1 0	0	Shift	Step 1
0 0 0 0 0 0	0 0 0 1 1 1	0	Shift	Step 2
0 1 1 1 1 1	0 0 0 1 1 1	0	A = A – M	Step 3
0 0 1 1 1 1	1 0 0 0 1 1	1	Shift	
0 0 0 1 1 1	1 1 0 0 0 1	1	Shift	Step 4
0 0 0 0 1 1	1 1 1 0 0 0	1	Shift	Step 5
1 0 0 1 0 0	1 1 1 0 0 0	1	A = A + M	Step 6
1 1 0 0 1 0	0 1 1 1 0 0	0	Shift	

Data Path for Booth's Algorithm