Lecture 11-12: March 5, 2021

Computer Architecture and Organization-I

Biplab K Sikdar

Multiplication of n-bit numbers in sign magnitude shift/addition technique

Y = $y_7y_6\cdots y_1y_0$=01100101 and X = $x_7x_6\cdots x_1x_0$ = 11011101.

To get product P = $p_6p_5\cdots p_1p_0$, 7 magnitude bits of Y and X are to be multiplied.

Here, P is the sum of partial products P$_0$, P$_1$, $\cdots$, P$_6$.

A partial product can be all 0s or Y$\times 2^i$ (that is, Y with i left shift).

	1100101	$y_6y_5y_4y_3y_2y_1y_0$
	1011101	$x_6x_5x_4x_3x_2x_1x_0$
0000000	1100101	P_0
000000**0**	0000000	P_1
00000**11**	0010100	P_2
0000**110**	0101000	P_3
000**1100**	1010000	P_4
00**00000**	0000000	P_5
0**110010**	1000000	P_6

In m/c, we can implement it by right shift of the register that stores product P.

Consider accumulator A and register Q stores the product.

A stores most significant part of P, and Q stores least significant part of P.

A and Q are connected and form a single shift register.

Algorithm 0.1 In sign magnitude, we need to consider multiplier[n-2:0], multiplicand[n-2:0] and A[n-2:0].

Step 1: A[n-2:0] $\leftarrow$ 0; C $\leftarrow$ 0; M[n-2:0] $\leftarrow$ multiplicand[n-2:0]; Q[n-2:0] $\leftarrow$ multiplier[n-2:0]

Step 2: if $Q_0 \neq 0$, then A $\leftarrow$ A+M

Step 3: right shift A, Q

Step 4: if C $\neq$ n-2 then increment C = C+1 and go to Step 2.

Step 5: output A, Q and exit.

Example 0.2 *Here n=5. The magnitude is of 4-bit.*

Action	*CY*	*A*	*Q*	*M*	*C*
	0	0000	$110\underline{1}$	1010	0
$+$		1010			
	0	1010			
RS	0	0101	$011\underline{0}$		1
RS	0	0010	$101\underline{1}$		2
$+$		1010			
	0	1100			
RS	0	0110	$010\underline{1}$		3
$+$		1010			
	1	0000			
RS	1	1000	0010		

$10 \times 13 = 130$

0.9.2 Fixed point multiplication in 2's complement

Multiplication in 2's complement can be done by modifying Algorithm 0.1.

Here, sign bits of multiplier and multiplicand are treated as magnitude.

Consider M = 0110 = 6 and Q = 1101 = -3 = 13 -16 in 2's complement.

If Algorithm 0.1 is followed, product P_{sign} = 0100 1110 = 78. But it should be -18.

The final product P needs correction as

$P = P_{sign} - 2^4 \times M = 78 - 16 \times 6 = -18$.

Let multiplicand Y = $y_{n-1} \cdots y_1 y_0$ and multiplier X = $x_{n-1} \cdots x_1 x_0$ in 2's complement.

- Case I: $\underline{y_{n-1} = x_{n-1} = 0}$: Algorithm 0.1 can produce the correct result.
- Case II: $\underline{y_{n-1} = 1 \text{ and } x_{n-1} = 0}$: (Y = 1101 = -3 and X = 0110 = +6).

  As per Algorithm 0.1 product P_{sign} is 78 (incorrect).

  Reason is - Algorithm 0.1 assumes partial products (P_0 = 0000 0000, P_1 = 0001 1010, P_2 = 0011 0100, and P_3 = 0000 0000) $P_i = 2^i\ Y\ x_i\ \forall_{i=0}^{n-1}$ as positive.

  As multiplicand is negative, all partial products must be negative.

  Correction needed in Step 3 of Algorithm 0.1: enter 1 in MSB if partial product is not 0.
- Case III: $\underline{y_{n-1} = 0 \text{ and } x_{n-1} = 1}$: Algorithm 0.1 results in incorrect product $P_{sign} = (2^n + X)Y$.

  The correction needed is

  $$P = P_{sign} - 2^n Y.$$
- Case IV: $\underline{y_{n-1} = x_{n-1} = 1}$: Algorithm 0.1 has to be corrected as in Case II/III.

```
Algorithm 0.2 Step 1: A[$n$-1:0] $\leftarrow$ 0; CY $\leftarrow$ 0; Count $\leftarrow$ 0;
M[$n$-1:0] $\leftarrow$ multiplicand[$n$-1:0]; Q[$n$-1:0] $\leftarrow$ multiplier[$n$-1:0]
Step 2: if $Q_0 \neq 0$, then A $\leftarrow$ A+M
Step 3: right shift A, Q
        $AQ_i \leftarrow AQ_{i+1} \forall_{i=0}^{n-2}$
        $A_{n-1} \leftarrow A_{n-1} \vee CY$
Step 4: if Count $\neq n$-2 then increment Count = Count+1; go to Step 2.
Step 5: if $Q_0$ = 0, then right shift; go to Step 8
Step 6: A $\leftarrow$ A+M; right shift
Step 7: A $\leftarrow$ A-M
Step 8: Output A,Q
```

Action	CY	A	Q	M	Count
	0	0000	110$\underline{1}$	1010	0
$+$		1010			
	0	1010			
RS	0	1101	011$\underline{0}$		1
RS	0	1110	101$\underline{1}$		2
$+$		1010			
	1	1000			
RS	1	1100	010$\underline{1}$		
$+$		1010			
	1	0110			
RS	1	1011	001$\underline{0}$		
$-$		1010			
		0001	0010	Product	

This requires at least p additions/subtractions (p: number of 1s in multiplier). Further, correction (subtraction) is needed if multiplier is negative.

0.9.3 Booth's algorithm for multiplication in 2's complement

Andrew Donald Booth (British) has proposed Booth's algorithm.

Let consider computation of Product P = M$\times$Q, where Q is the multiplier and

$$\text{Q} = \overset{i=4}{0}\ \ \overset{3\,2\,1\,0}{1110} = 14.$$

Therefore, P = 14$\times$M = (+16 - 2)$\times$M = +$2^4\times$M - $2^1\times$ M.

Similarly, for the multiplier

$$\text{Q} = \overset{i=11}{0}\ \ \overset{10\ 9}{0\,1}\ \ \overset{8\ 7\ 6}{1\,1\,0}\ \ \overset{5\ 4\ 3}{0\,1\,1}\ \ \overset{2\ 1\ 0}{1\,1\,0},$$

P = +$2^{10}\times$M - $2^7\times$M + $2^5\times$M - $2^1\times$M.

Total number of additions/subtractions is 4 only. It signifies

(i) Number of additions/subtractions may be less than the number of 1s in multiplier. It depends on number of flips (0 to 1 or 1 to 0) in multiplier.

(ii) If i^{th} and $(i-1)^{th}$ bit-pair of multiplier is 10, then subtraction is needed.

(iii) If i^{th} and $(i-1)^{th}$ bit-pair of multiplier is 01, then addition is required.

<u>**Features**</u> In Booth's algorithm of 2's complement multiplication,

i) No correction is needed as required for Algorithm 0.2,

ii) Negative and positive numbers are treated uniformly,

iii) Computation of product is faster than Algorithm 0.2.

While computing product of Booth's algorithm scans multiplier X = $x_{n-1}x_{n-2}\cdots x_2x_1x_0$ considering adjacent bits x_ix_{i-1}.

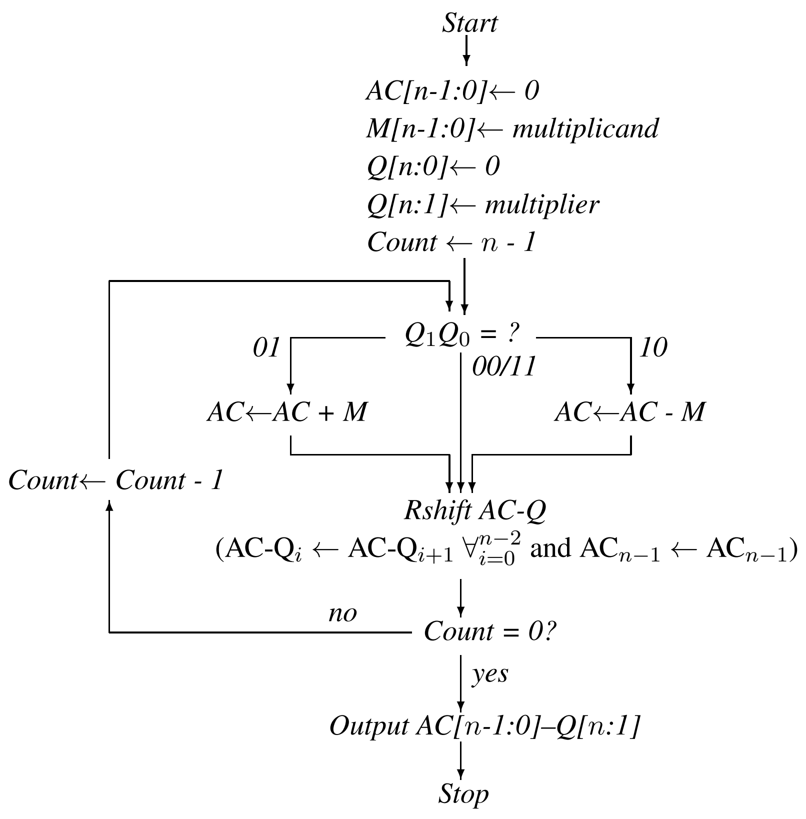
- If x_ix_{i-1} $\begin{array}{ll} = 01, & multiplicand\,Y\ is\ added\,to\,partial\ product\ PP_{i-1}. \\ = 10, & Y\ is\ subtracted\ from\ PP_{i-1}. \end{array}$

  Folled by a left shift to get PP$_i$. Here, PP$_i$ defines product of Y and $x_ix_{i-1}\cdots x_2x_1x_0$.

- If x_i = x_{i-1}, then only left shift of PP$_{i-1}$ is carried out to get PP$_i$.

LSB of multiplier X is the *i*=0th bit.

A bit x[0]=0 is appended with LSB of X to facilitate bit pair for i=0.



Multiplicand M = 1010 and multiplier Q is 1101.

Action	AC	Q	M	Count
	0000	110$\underline{1\mathbf{0}}$	1010	3
−	1010			
	0110			
RS	0011	011$\underline{01}$		2
+	1010			
	1101			
RS	1110	101$\underline{10}$		1
−	1010			
	0100			
RS	0010	010$\underline{11}$		0
RS	$\underbrace{0001}$	$\underbrace{0010}$ 1		

The product is AC[3:0]–Q[4:1] = 00010010.

Hardware realization of Booth's multiplier is in Figure 30.

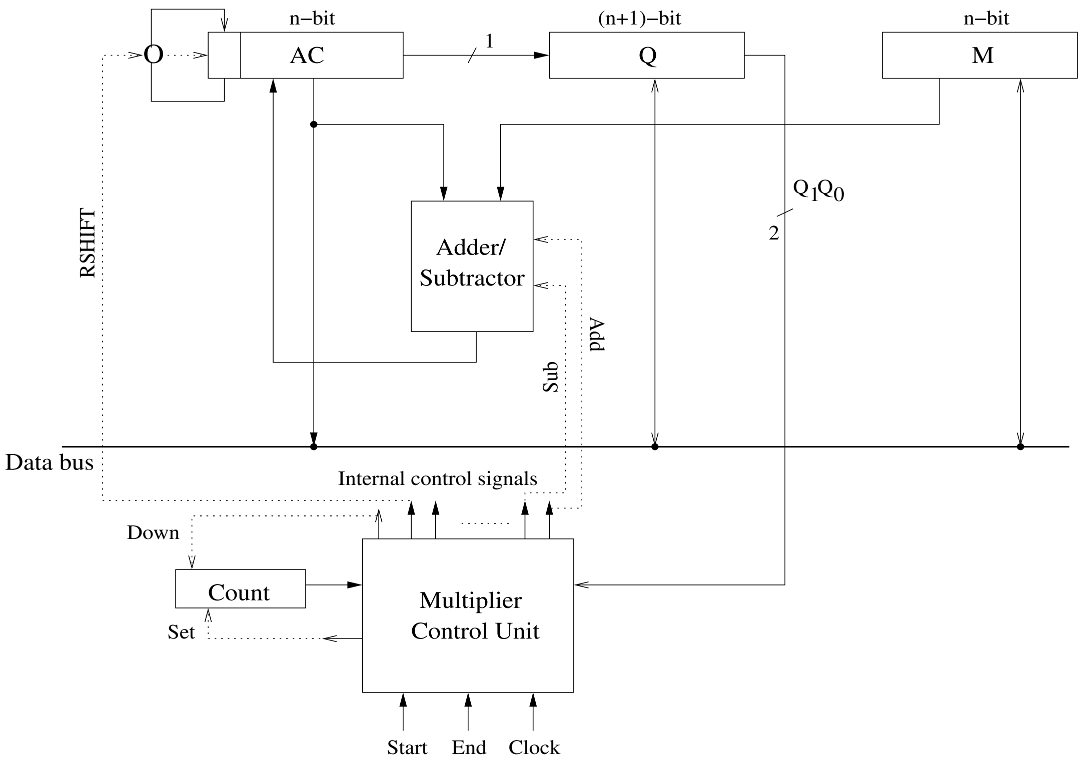


Figure 30: Booth's multiplication algorithm hardware realization

Multiplier control unit compares two least significant bits of register Q and generates appropriate control signals (Add, Sub, RSHIFT, Down Counter, etc).

Limitations: Booth's agorithm enhances speed of multiplication for runs of 1s in multiplier.

For an n-bit multiplier with $\frac{n}{2}$ isolated 1s, Booth's algorithm becomes more costly.

For such a case, Booth's algorithm needs n additions/subtractions.

Let multiplier X = 01010101. Number of additions/subtractions needed 8.

X - Multiplier = 01010101

Q - Multiplier with augmented 0 at least significant position = 010101010

Number of 01 pairs in Q (additions in Booth's algorithm) = 4

Number of 10 pairs (subtractions in Booth's algorithm) = 4

0.9.4 Bit-pair multiplication scheme

If $(i+1)^{th}$, i^{th}, and $(i-1)^{th}$ bits in multiplier are 101, then as per Booth's algorithm,

$$\begin{aligned} PP_{i+1} &= PP_{i-1} + 2^iY - 2^{i+1}Y \\ &= PP_{i-1} - 2^iY. \end{aligned}$$

So, for bit pair 101, a subtraction can replace an addition and a subtraction.

Similarly, for 010

$$\begin{aligned} PP_{i+1} &= PP_{i-1} - 2^iY + 2^{i+1}Y \\ &= PP_{i-1} + 2^iY. \end{aligned}$$

Hence two arithmatic operations can be replaced by a single operation.

Actions to be taken for all such 8 bit pairs are described in following table.

$i+1$	i	$i-1$	*Action*
0	0	0	$PP_{i+1} = PP_{i-1}$
0	0	1	$PP_{i+1} = PP_{i-1} + 2^iY$
0	1	0	$PP_{i+1} = PP_{i-1} + 2^iY$
0	1	1	$PP_{i+1} = PP_{i-1} + 2^{i+1}Y$
1	0	0	$PP_{i+1} = PP_{i-1} - 2^{i+1}Y$
1	0	1	$PP_{i+1} = PP_{i-1} - 2^iY$
1	1	0	$PP_{i+1} = PP_{i-1} - 2^iY$
1	1	1	$PP_{i+1} = PP_{i-1}$