SASTRA	**COMPUTER ORGANIZATION AND ARCHITECTURE** **Course Code: CAP403R01 Semester : I / MCA** **Dr. K. Geetha/SAP/CSE/SOC/SASTRA**

1.5(a) Arithmetic for Computers: Multiplication

Multiplication
unsigned integers Figure 10.7 illustrates the multiplication of unsigned
binary integers, as might be carried out using paper and pencil

Manual multiplication

1. Multiplication involves the generation of partial products, one for each digit in the multiplier. These partial products are then summed to produce the final product.
2. The partial products are easily defined. When the multiplier bit is 0, the partial product is 0. When the multiplier is 1, the partial product is the multiplicand.
3. The total product is produced by summing the partial products. For this operation, each successive partial product is shifted one position to the left relative to the preceding partial product.
4. The multiplication of two n-bit binary integers results in a product of up to 2n bits in length (e.g., 11 * 11 = 1001).

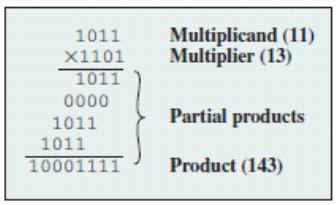


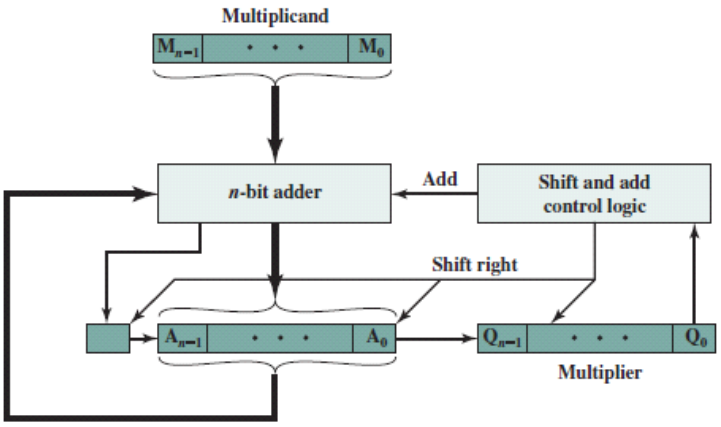
Figure 10.7 Multiplication of Unsigned Binary Integers

Figure 10.8a shows a possible implementation employing these measures.

The multiplier and multiplicand are loaded into two registers (Q and M).
A third register, the A register, is also needed and is initially set to 0.
There is also a 1-bit C register, initialized to 0, which holds a potential carry bit resulting from addition.

The operation of the multiplier is as follows:
Control logic reads the bits of the multiplier one at a time.
If Q0 is 1, then the multiplicand is added to the A register and the result is stored in the A register, with the C bit used for overflow.
Then all of the bits of the C, A, and Q registers are shifted to the right one bit, so that the C bit goes into An-1, A0 goes into Qn-1, and Q0 is lost.
If Q0 is 0, then no addition is performed, just the shift.

This process is repeated for each bit of the original multiplier.
The resulting 2n-bit product is contained in the A and Q registers.
A flowchart of the operation is shown in Figure 10.9, and an example is given in Figure 10.8b.
Note that on the second cycle, when the multiplier bit is 0, there is no add operation



(a) Block diagram

C	A	Q	M		
0	0000	1101	1011	Initial values	
0	1011	1101	1011	Add	First cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	Second cycle
0	1101	1111	1011	Add	Third cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	Add	Fourth cycle
0	1000	1111	1011	Shift	

(b) Example from Figure 10.7 (product in A, Q)

Figure 10.8 Hardware Implementation of Unsigned Binary Multiplication

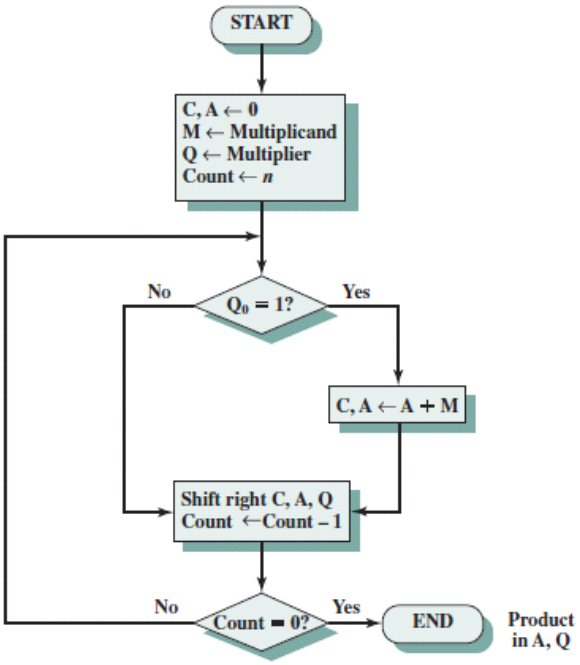


Figure 10.9 Flowchart for Unsigned Binary Multiplication

Signed multiplication

1 +7 * +3
2 -7 * +3
3 +7 * -3
4 -7 * -3

Booth algorithm

Multiplier		Version of multiplicand selected by bit i
Bit i	Bit $i-1$	
0	0	$0 \times M$
0	1	$+1 \times M$
1	0	$-1 \times M$
1	1	$0 \times M$

Figure 9.12 Booth multiplier recoding table.

Logical Shift

- A ***Left Logical Shift*** of one position moves each bit to the left by one. The vacant least significant bit (LSB) is filled with zero and the most significant bit (MSB) is discarded.
- A ***Right Logical Shift*** of one position moves each bit to the right by one. The least significant bit is discarded and the vacant MSB is filled with zero.

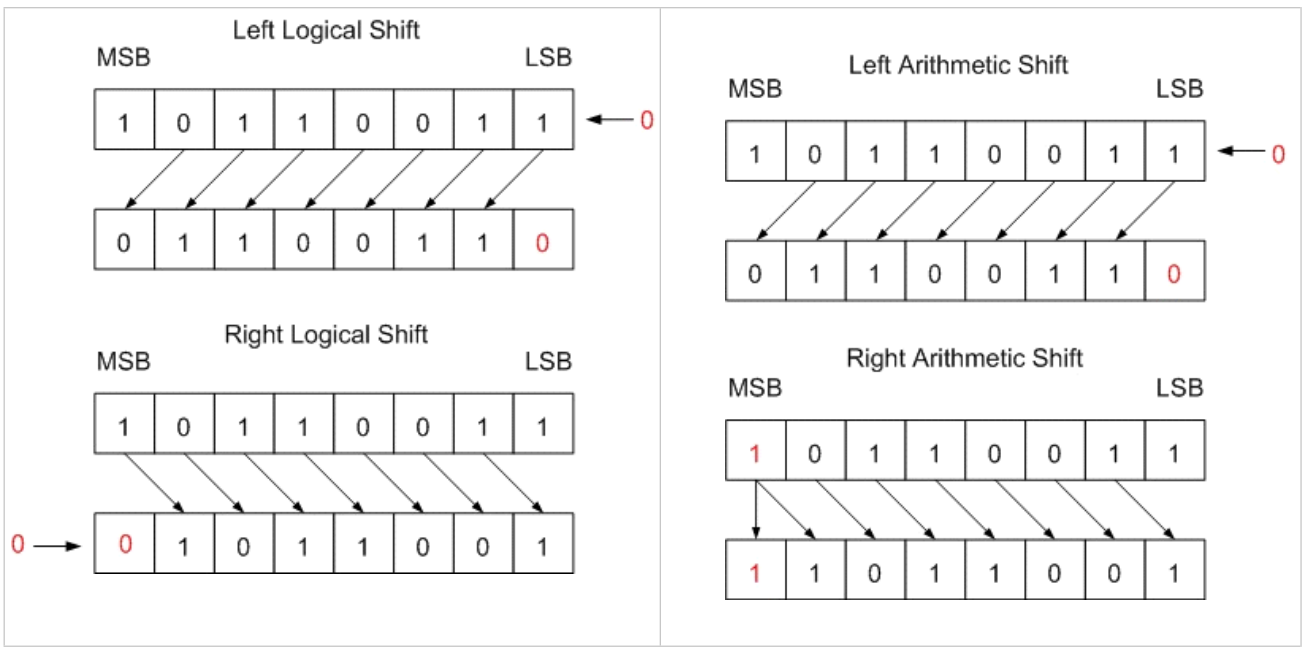


Fig. 1 Logical Shift by one bit

Arithmetic Shift

- A ***Left Arithmetic Shift*** of one position moves each bit to the left by one. The vacant least significant bit (LSB) is filled with zero and the most significant bit (MSB) is discarded. It is identical to Left Logical Shift.
- A ***Right Arithmetic Shift*** of one position moves each bit to the right by one. The least significant bit is discarded and the vacant MSB is filled with the value of the previous (now shifted one position to the right) MSB.

Fig. 1 Left and Right Arithmetic Shift by One Bit

Arithmetic Shift operations can be used for dividing or multiplying an integer variable.

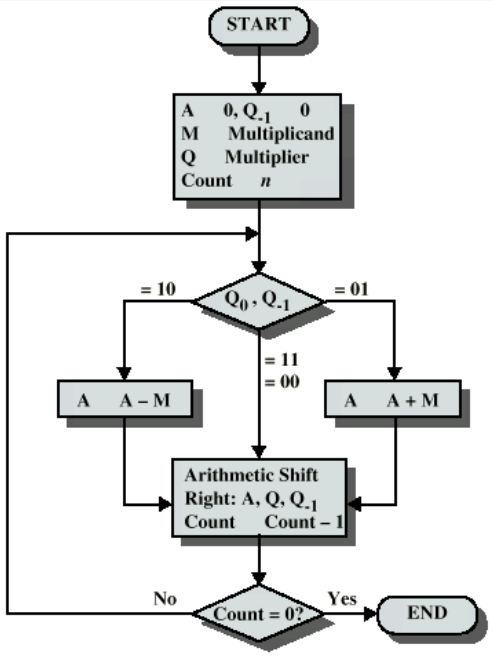


Figure 8.12 Booth's Algorithm for Twos Complement Multiplication

A	Q	Q_{-1}	M		
0000	0011	0	0111	Initial values	
1001	0011	0	0111	A ← A − M	First cycle
1100	1001	1	0111	Shift	First cycle
1110	0100	1	0111	Shift	Second cycle
0101	0100	1	0111	A ← A + M	Third cycle
0010	1010	0	0111	Shift	Third cycle
0001	0101	0	0111	Shift	Fourth cycle

Figure 9.13 Example of Booth's Algorithm (7 × 3)

```
                                  0  1  0  1  1  0  1
                                  0  0+1 +1 +1 +1 +1  0
                                  0  0  0  0  0  0  0
                               0  1  0  1  1  0  1
                            0  1  0  1  1  0  1
                         0  1  0  1  1  0  1
                      0  1  0  1  1  0  1
                   0  0  0  0  0  0  0
                0  0  0  0  0  0  0
-------------------------------------------------------
0  0  0  1  0  1  0  1  0  0  0  1  1  0
```

```
                                  0  1  0  1  1  0  1
                                  0 +1  0  0  0 –1  0
                                  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0
1  1  1  1  1  1  1  0  1  0  0  1  1   <-- 2's complement of the multiplicand
0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0
0  0  0  1  0  1  1  0  1
0  0  0  0  0  0  0  0
-------------------------------------------------------
0  0  0  1  0  1  0  1  0  0  0  1  1  0
```

```
   0  1  1  0  1   (+13)
×  1  1  0  1  0   (–6)
-----------------
```

⟹

```
                  0  1  1  0  1
                  0 –1 +1 –1  0
                  0  0  0  0  0  0  0  0  0  0
               1  1  1  1  1  0  0  1  1
            0  0  0  0  1  1  0  1
         1  1  1  0  0  1  1
      0  0  0  0  0  0
-------------------------------------------------------
1  1  1  0  1  1  0  0  1  0   (–78)
```

Figure 9.11 Booth multiplication with a negative multiplier.

Figure 9.13 shows the sequence of events in Booth's algorithm for the multiplication of 7 by 3. More compactly, the same operation is depicted in Figure 9.14a. The rest of Figure 9.14 gives other examples of the algorithm. As can be seen, it works with any combination of positive and negative numbers. Note also the efficiency of the algorithm. Blocks of 1s or 0s are skipped over, with an average of only one addition or subtraction per block.

0111		0111	
× 0011	(0)	× 1101	(0)
11111001	1–0	11111001	1–0
0000000	1–1	0000111	0–1
000111	0–1	111001	1–0
00010101	(21)	11101011	(–21)

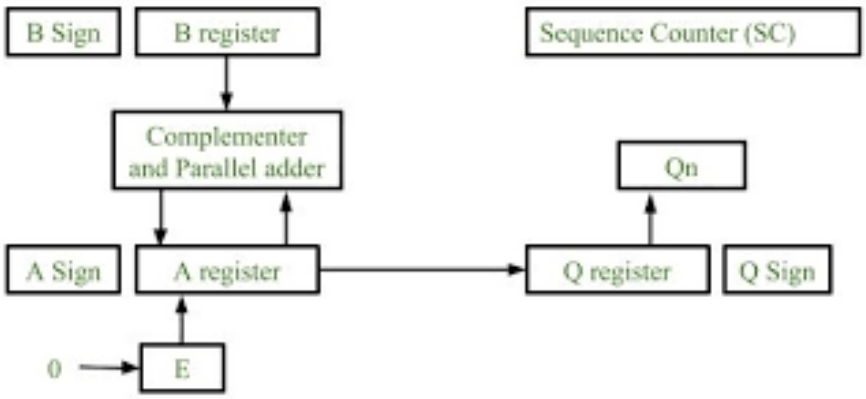
(a) (7) × (3) = (21) (b) (7) × (–3) = (–21)

1001		1001	
× 0011	(0)	× 1101	(0)
00000111	1–0	00000111	1–0
0000000	1–1	1111001	0–1
111001	0–1	000111	1–0
11101011	(–21)	00010101	(21)

(c) (–7) × (3) = (–21) (d) (–7) × (–3) = (21)

Figure 9.14 Examples Using Booth's Algorithm

Hardware For Muliplication


B Sign
B register
Sequence Counter (SC)
Complementer
and Parallel adder
Qn
A Sign
A register
Q register
Q Sign
0
E