Multiplier

- A multiplier multiplies two binary numbers.
- It follows the same rules as in decimal numbers.
- Most of the multiplication techniques involve a set of partial products, shifting them to left, and then sum up them together using adders. A repeated shift and add process.
- In partial products, there is no need to remember any multiplication table of numbers. Because there are 0 and 1 only, and result will be 0 and 1. It is easier.

```
   11
 x 10
 ----
   00
  11
 ----
  110
```

- This process can be done by using the add and shift features of the ALU.
- For long numbers, this process is time consuming.
- Hardware multiplier can be used also which is faster.
- In modern computers, multiplication of signed numbers is done usually using two’s compliment representation.

Multiplier (contd.)

- Multiplication of positive numbers: is a combination of a full adder and an AND gate in each portion of partial product. Figure 6.6
- Multiplication of signed numbers: two's complement representation.
  - Case 1: multiplicand is negative and multiplier is positive. For example: multiplicand -13 (11101) and multiplier +11 (01011). Generates a double length product. Here, the sign bit of the multiplicand has to extend to the left as far as the product will extend.

```
          1 0 0 1 1 (two's complement of -13)
        x 0 1 0 1 1
  ---------------------
1 1 1 1 1 1 0 0 1 1
1 1 1 1 1 0 0 1 1
0 0 0 0 0 0 0 0
1 1 1 0 0 1 1
0 0 0 0 0 0
  ---------------------
1 1 0 1 1 1 0 0 0 1
```

  - Case 2: If the multiplier is negative, then two's complement of the both multiplicand and multiplier is made. Then proceed as case 1.

Booth’s Algorithm

- It is an algorithm that multiplies two signed binary numbers efficiently using two’s compliment representation.
- It requires less addition/subtraction operations.
- It speeds up the multiplication process.
- It treats both positive and negative numbers uniformly.
- This algorithm generates a 2n-bit products.

See Zaky (Chapter 6: Arithmetic)

Booth's Algorithm(contd.)

How it works:

- Booth's algorithm checks the bits of the multiplier and shifts the partial product.
- Before shifting, the multiplicand may be added to or subtracted from the partial product depending on the following rules:
  - ✓ The multiplicand is subtracted from the partial product upon encountering the first least significant 1 (provided that there was a previous 0) in a string of 1's in the multiplier.
  - ✓ The multiplicand is added to the partial product upon encountering the first 0 (provided that there was a previous 1) in a string of 0's in the multiplier.
  - ✓ The partial product does not change when the multiplier bit is identical to the previous multiplier bit (0 0 or 1 1).

In other words,

- ✓ If Q_0 and Q_n are 10, then subtract the multiplier from the partial product.
- ✓ If Q_0 and Q_n are 01, then add the multiplier to the partial product.
- ✓ If Q_0 and Q_n are the same, then no change required.

Booth's Algorithm Example

- MD (multiplicand)=-5=1011

  2's complement of MD=0101

  MR(multiplier)=-7=1001

  So, the product would be of 8 bit

Sequence Counter	AC Register	MR	1 bit Register Qn	Operation
4	0000	1001	0	Initialization
3	0101 0010	1001 1100	0 1	AC=AC-MD then right shift
2	1101 1110	1100 1110	1 0	AC=AC+MD then right shift
1	1110 1111	1110 0111	0 0	No change then right shift
0	0100 0010	0111 0011	0 1	AC=AC-MD then right shift

Product=AC MR=00100011=35

Divider

- Binary division follows the same ways as decimal division does.
- Restoring division: In restoring division,

  an n bit positive divisor is loaded into register M.

  an n bit positive dividend is loaded into register Q (after completing the division operation, register Q contains the quotient).

  Register A is set to 0 (after completing the division operation, register A contains the remainder).

  Algorithm: repeat the following steps n times.

  Step 1: Initialization: M $\leftarrow$ divisor, Q $\leftarrow$ dividend, A $\leftarrow$ 0 and n $\leftarrow$ number of bits in dividend

  Step 2: Left shift of A and Q by one bit position.

  Step 3: Subtract M from A and place the result in A.

  Step 4: if the sign bit of A is 1, set q_0 to 0 and add M back to A (that is restore A). Otherwise set q_0 to 1.

See Zaky (Chapter 6: Arithmetic)

Divider (contd.)

- Example:

  Dividend=8=1000

  Divisor=3=11

n	M	A	Q	Operation
4	00011	00000	1000	Initialization
	00011	00001	000_	Left shift of A and Q
	00011	11110	000_	A=A-M
		00001	0000	Sign bit of A is 1, set q_0 to 0, add M back to A
3				