

## 14. Multiplication of Binary Integers

Like addition, there are only four "rules" for multiplication of binary integers.

$$\begin{array}{r} 0 \\ \times 0 \\ \hline 0 \end{array} \quad \begin{array}{r} 1 \\ \times 0 \\ \hline 0 \end{array} \quad \begin{array}{r} 0 \\ \times 1 \\ \hline 0 \end{array} \quad \begin{array}{r} 1 \\ \times 1 \\ \hline 1 \end{array}$$

What makes multiplication difficult is the need to add the partial products obtained by multiplying the multiplicand (the "upper" integer) by each bit of the multiplier (the "lower" integer).

```
1011 <- multiplicand (13d)
 101  <- multiplier (5d)
-----
 1011 <- partial products
 0000 <-
1011  <-
-----
110111 <- final product (65d)
```

However, close observation of the example above reveals that binary multiplication can be performed by scanning the bits of the multiplier *right to left*, testing each bit, adding or not adding the multiplicand to the product depending on whether the bit is 1 or 0, then *left shifting* the multiplicand after each "test". Hence the following "test and shift" algorithm for binary multiplication. In the "pseudo code" presented below, the testing of each bit of the multiplier is done by *right shifting* the multiplier and testing the right most bit.

```
Product <- 0
While Multiplier is not 0 do
{
  If right-most bit of multiplier = 1 then
    Product <- Product + Multiplicand
  Left Shift Multiplicand
  Right Shift Multiplier
}
```

Left shifting a binary number shifts each bit one position to the left while adding a zero bit to the right end and, given a fixed length representation, discarding the left-most bit. For example, left shifting the 8-bit representation 00001011 results in 00010110. It is the same as multiplication by 2.

Right shifting is almost the opposite with the right-most bit being discarded and a 0 attached to the left end. For example, right shifting 00000101 results in 00000010. It is the same as integer division by 2 (e.g. 5 DIV 2 is 2).

**Exercise:** Trace the execution of the above "test and shift" pseudo-code using the examples given below. Check your work by converting each binary integer to decimal then doing the multiplication in decimal.

$$\begin{array}{r} \text{a.} \quad 1011 \\ \quad \times 101 \\ \hline \end{array}$$

$$\begin{array}{r} \text{b.} \quad 10011 \\ \quad \times 110100 \\ \hline \end{array}$$

$$\begin{array}{r} \text{c.} \quad 1111 \\ \quad \times 1101 \\ \hline \end{array}$$

The algorithm given above only works for unsigned binary integers. To multiply signed binary integers, determine the sign of the product (negative if and only if both factors have opposite signs), convert all negative integers to positive (unsigned) integers, multiply and if the product was determined to be negative, "complement and add one" to make the product negative. With n-bit two's complement representation, the above algorithm also works if the multiplier is positive; the multiplicand can be positive or negative.