Multiplication of Binary Integers

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

```
      0           1           0           1
    x 0         x 0         x 1         x 1
  -----       -----       -----       -----
      0           0           0           1
```

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 (11d)
    101  <- multiplier (5d)
 ------
   1011  <- partial products
  0000   <-
 1011    <-
-------
 110111  <- final product (55d)
```

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 c-code program segment presented below, the testing of each bit of the multiplier is done by *right shifting* the multiplier and testing the right most bit.

```
p = 0;                        // initialize product p to 0
while (n != 0)                // while multiplier n is not 0
   {
   if ((n & 0x01) != 0)       // test lsb of multiplier
        p = p + m;            //    if 1 then add multiplicand m
   m = m << 1;                // left shift multiplicand
   n = n >> 1;                // 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 / 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.

```
a.     1101         b.     10011        c.   1111
     x  101            x 110100            x 1101
     ------            --------            ------
```

Multiplication is implemented on a computer using a slightly different but equivalent approach. Since the product can be twice as long as the multiplier or the multiplicand, a pair of registers is needed to hold it. Initially the right-most (low order) member of the pair holds the multiplier; the left-most (high order) member is zeroed out. The multiplicand is stored in a third register. The carry flag is also used to catch any overflow in what follows

```
          +----------+
          |  m'cand  |                test least significant bit
          +----------+
               ↓ add
     +--+ +----------+ +----------+
     |cf|→| 00000000 |→|multiplier|    <- double register pair
     +--+ +----------+ +----------+
            hi-order     lo-order
```

Multiplication work like this

1. The right-most (least significant) bit of the multiplier in the right hand register is tested; if it is 1 then the multiplicand is added into the left (hi-order) half of the register pair.

2. The double register pair is right shifted which moves the least significant of the hi-order half into the most significant bit of the lo-order half and moves the next bit of the multiplier into the right-most position where is can be tested.

The right shift of the register pair is equivalent to left shifting the multiplicand and right shifting the multiplier as given in the multiplication algorithm above.

3. Steps 1 and 2 are repeated `n` times where `n` is the width of the registers. At the end the hi-order half of the product is in the left register and the lo-order half of the product is in the right register

Note: On a single accumulator machine with an MQ register, the Accumulator is left half and the MQ register is the right half of the double register pair.

Exercise: Using four bit registers compute the product of 5 × 3 using the register set up. Loop 4 times

```
                      +----+
      0011            |0011|                test least significant bit
     ×0101            +----+
     -----              ↓ add
                  +-+ +----+ +----+
                  |0|→|0000| |0101|
                  +-+ +----+ +----+
```

Do the same with 10 × 13 (i.e. 1010 ×1101) which generates an 8 bit result

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.