Division of Binary Integers

Division is more difficult than multiplication. However division of *binary* integers is less difficult than you might first expect since with the standard “pencil and paper algorithm”, it’s fairly easy to determine if the divisor “goes into” the dividend. Assuming you are fairly "fluent" in binary subtraction for unsigned integers (for instance subtract 101 from 1000 without changing 101 to its negative two's complement representation), binary division can be done like decimal division.

Example : `Calculate 10001 (17d) ÷ 101 (5d)`

```
                          11  <- quotient
                       -----
       divisor ->  101 / 10001  <- dividend
                      - 101
                       ----
                        0111
                       - 101
                         ---
                          10  <- remainder
```

This standard "pencil and paper" technique is the basis for the "shift, test and restore" algorithm (discussed below) that is fairly easy to implement on computers.

To begin, the algorithm only works with positive "fixed length" integer representations where each integer is represented using the same number of bits. Leading zeroes, if necessary, are added to an integer's representation so that all integers have the same number of bits. For example, with 8-bit bytes 13 decimal is represented by 00001101 not 1101. Since all computers use fixed length representations whose length is based on a multiple of the number of bits in each addressable memory cell (PDP-8 is 12 bits, Intel 80x86 is a multiple of 8 bits, ARC is 32-bits etc), this is not a problem.

The "shift, test, and restore" algorithm begins by placing the dividend right justified into a double width register. For example, if our dividend of 13 was represented using an 8-bit byte, 00001101 would be placed right justified in a 2 byte register like so

```
    R        Q
00000000 00001101
```

We'll call the left-most byte R and the right-most byte Q (so the dividend is initially placed in Q) for reasons that will become apparent. Let 5 be the divisor. Its 8-bit representation, 00000101, is placed in a single byte register.

The "shift, test and restore" algorithm iterates a fixed number of times.

```
For i = 1 to 8 do  { we're using 8-bit representations }
    {
    Double Left Shift the RQ pair
    Subtract the Divisor from R
    If R is positive then
       Set the low order (right most) bit in Q to 1
    Else
       Restore R by Adding back the Divisor
    }
```

Here is a c-code example (assuming 32 bit integers),

```
    for (i = 0; i < 32; i++)              // do 32 times
        {                                 // double left shift r-q pair
        if ((q & 0x80000000) != 0)        // check if carry from q to r
            r = (r << 1) + 1;             // left shift r w/ carry in
        else
            r = r << 1;                   // left shift r w/o carry
        q = q << 1;                 // left shift q
        if (r - b >= 0)             // test if b can be subtracted from q
            {
            q = q | 0x01;           // set lsb of q
            r = r - b;
            }
        }
```

Example

```
   R        Q          Divisor     Negative Divisor
00000000 00001101      00000101        11111011

00000000 00011010      Left Shift RQ pair (1st iteration)

11111011 00011010      add in Negative Divisor (-5 or 11111011)

00000000 00011010      since R is negative restore by
                       adding back the Divisor
```

this same scenario repeats 5 more times until we obtain

```
00000011 01000000      end of 6th iteration

00000110 10000000      Left Shift RQ Pair (7th iteration)

00000001 10000000      add in Negative Divisor

00000001 10000001      since positive, set low order bit in Q
                       to 1

00000011 00000010      Left Shift RQ Pair (last iteration)

11111110 00000010      add in Negative Divisor

00000011 00000010      since R is negative restore by
                       adding back the Divisor
```

At the end, the "Q half" has the Quotient which is 2 and the "R half" has the Remainder which is 3.
Note 13 ÷ 5 = 2 r 3.

Exercises: Using 8-bit representations, use the "shift, test and restore" algorithm to perform the following divisions. Check your work by converting values to decimal and performing the decimal calculations.

```
a.    1101101 ÷ 1101    b.    1100000 ÷ 100    c.    11111 ÷ 11000
```

As in multiplication, the "shift, test and restore" algorithm only works for unsigned integers. It can be modified in the same way multiplication was to work with signed values.