Binary Multiplication and Division

Multiplication

Binary multiplication works just like decimal multiplication. Given two binary numbers, M with m bits and N with n bits, set the most significant bit of M m+n rods to the left of your chosen units rod. Then begin to add in the partial products just to the right of M with each product of two single bits occupying two rods, just as the product of two base 10 digits occupies two rods

Example:
M x N = 110 x 110
m+n = 3+3 = 6

Set N at the left side and set M beginning 6 rods left of the units rod

```
                    U                    U marks the units rod
110 000 001 100 000 000 000
----        ----
 N           M (110)
```

multiply N by the 0 in M and add partial products, beginning just to the right of M but setting aside two rods for each partial product, exactly as we do in base 10

```
              +  0 0
                +  00
                  + 00
-----------------------------------
110 000 001 100 000 000 000
----        ----
 N           M (110)
```

clear the least significant bit in M (it is already 0)

```
110 000 001 100 000 000 000
----        ---
 N           M (11)
```

multiply N by the middle bit of M and add partial products

```
              + 01
                + 0 1
                  + 00
-----------------------------------
110 000 001 101 100 000 000
----        ---
 N           M (11)
```

clear the middle bit of M

```
110 000 001 001 100 000 000
----        --
 N          M (1)
```

multiply N by the most significant bit of M and add partial products

```
            + 01
              + 01
                + 0 0
-----------------------------------
110 000 001 100 100 000 000
----        --
 N          M (1)
```

clear the most significant bit of M

```
                        U
110 000 000 100 100 000 000
----
 N
```

the product is 100 100 which is 36 in base 10.

Division

Binary division works the same as decimal division, except that there is no multiplication involved. Instead, we either subtract 0 or subtract the divisor from the dividend depending on whether the quotient bit was 0 or 1. In binary we can compute A-B by A+B/ where A and B are binary numbers and B/ is the 2's complement of B. To form the 2's complement of a binary number, you can invert all the bits and then add a 1 to the least significant 1 bit, or an easier way is to start at the right side with the least significant "1" bit - keep that one unchanged, then invert all the bits to its left. So, for example, the 2's complement of 11 is 01. If we need more than two bits, we can say that 11 is really 011 and the 2's complement is 101, or we could say that 11 is really 0011 and the 2's complement is 1101, etc., for however many bits we need.

So, to find 10111/11, first find the place to set the dividend. Just as with decimal arithmetic, we take the number of digits (bits in this case) in the dividend minus the bits in the divisor minus 2, or 5-2-2 = 1, so we enter the dividend one rod to the left of the units rod.

```
            U
011 000 010 111 000 000 000 000
----        -------
DVSR        DVDND
```

Because the divisor will not go into the first two bits of the dividend (10), we place the first quotient bit just to the left of the dividend, as usual in decimal arithmetic, and divide 11 into the first three bits of the dividend (101). Then we add the 3-bit 2's complement of the divisor (101) to the first three bits of the dividend.

```
          q  (quotient bit)
011 000 110 111 000 000 000 000
          + 10 1
----------------------------------------
011 000 120 211 000 000 000 000
          q
```

Then clean up the carries, but do not carry into the quotient bits (these carry bits which would overflow into the quotient bits are just dropped into the "bit bucket") :-)

```
          q
011 000 101 011 000 000 000 000
```

Now again divide 11 into the next three bits and add the 2's complement

```
          qq
011 000 111 011 000 000 000 000
          + 1 01
----------------------------------------
011 000 112 021 000 000 000 000
          qq
```

clean up carries

```
          qq
011 000 110 101 000 000 000 000
```

Again divide 11 into 101

```
          qqq
011 000 111 101 000 000 000 000
            + 101
----------------------------------------
011 000 111 202 000 000 000 000
          qqq
```

clean up the carries

```
          qqq
011 000 111 010 000 000 000 000
```

Divide 11 into 100

```
          qqq q
011 000 111 110 000 000 000 000
              + 10 1
----------------------------------------
011 000 111 120 100 000 000 000
          qqq q
```

clean up the carries

```
            qqq q
011 000 111 100 100 000 000 000
```

now 11 will not divide into the next three bits (010), so the quotient bit is 0

```
            qqq qq
011 000 111 100 100 000 000 000
                  + 0 00
-----------------------------------------
011 000 111 100 100 000 000 000
            qqq qq
```

11 goes into the next three bits (100)

```
            qqq qqq
011 000 111 101 100 000 000 000
                    + 101
-------------------------------
011 000 111 101 201 000 000 000
            qqq qqq
```

clean up carries

```
            qqq qqq
011 000 111 101 001 000 000 000
```

now 11 will not divide into the next three bits (010), so the quotient bit is 0

```
            qqq qqq q
011 000 111 101 001 000 000 000
                      + 00 0
-----------------------------------------
011 000 111 101 001 000 000 000
            qqq qqq q
```

11 goes into the next three bits (100)

```
            qqq qqq qq
011 000 111 101 011 000 000 000
                      +  1 01
-------------------------------
011 000 111 101 012 010 000 000
            qqq qqq qq
```

clean up carries

```
            qqq qqq qq
011 000 111 101 010 010 000 000
              U
```

et cetera

Continuing, we find that the quotient is 111.101 010 101 010 101 010 .....
or in octal, 7.5252525.....
which is, in decimal 7.6666666.....

Steve Treadwell
Sept., 2011