Approximation of Fractions

floating point values in a computer system are *approximations* of fractional values. Because we are dealing with powers of two, we can only precisely represent numbers that can be written as x * 2^y. Other values, like 1/5 for example, must be approximated. We represent numbers with fractional components in binary in the form:

B_m, B_m-1,...B_1,B_0.B_-1,B_-2...B_-n-1,B_-n

Where the (.) is called the *binary point*. Bits to the left of the binary point are weighted with positive powers of two while bits to right are weighted with negative pwoers of two.

<u>Example:</u> 101.11

```
101.11
        = (1 * 2^2) + (0 * 2^1) + (1 * 2^0) + (1 * 2 ^ -1) + (1 * 2^-2)
        =(4) + (0) + (1) + (1/2) + (1/4)
            = 5 3/4
```

Consider the following two for loops:

```
int i;
for(i=0;i<MAXLOOPS;i++);

float j;
for(j=0;<MAXLOOPS;j++);
```

Using the clock() function (#include), we can compute the clock ticks needed to process each for loop. Running with MAXLOOPS defined as 100000, will produce results similar to ~320 ticks for the integer loop and 540 ticks for the floating point loop! Even on something as trivial as incrementing the counter variable in a for-loop, using a float for the counter decreases the speed by over 60%! WHY?!?

IEEE 754 representation

IEEE 754 is a standard for implementing floating point computation. It defines two basic formats, a single precision floating point (float) and a double precision floating point (double). IEEE 754 represented a number in the form of:

> V = (-1)^s x M x 2^(E - bias) where:

- *s* is the sign bit that designates if the number is positive or negative.
- *M* is called the **significand** and is a fractional binary number.
-
- *E* is the **exponent** and weighs the value by a power of 2.
- the **bias** is an offset that is introduced the help make comparisons of floating points faster. The bias is computed using the word length of the exponent by: 2^(k-1) - 1. (127 for single precision, 1023 for double precision).

IEEE 754 is an example of using a fixed-bit width encoding. where we have 1 bit for the sign, followed by k bits to designate the exponent and n bits to encode the fractional component. For single precision, k=8 and n =23. For double precision, k=11 and n=52.

<u>Example: Convert the following single precision IEEE 724 into a decimal value</u>

```
0xC0D00000

1) convert Hex to Bin
        1100 0000 1101 0000 0000 0000 0000 0000

2) Now we group:

      1-bit sign     [1]
      8-bits exponent     [1000 0001]
      23-bits significand   [101 0000 0000 0000 0000 0000]

3) evaluate the exponent:
      E
       (1 * 2^7 + 1 * 2 ^0 )
      = 129


4) compute the significant:

     1+ (1*2^-1) + (1*2^-3)
   = 1+ (1/2) + (1/8)
   = 1+5/8
   = 1.625

 5) put it together:

        V = (-1)^s * (M) * 2^(E-bias)

           =(-1)^(1) * (1.625) * 2 ^(129-127)
           = (-1) * (1.625) * (4)
           = -6.5
```

<u>Example: </u>Convert the following decimal value to a single precision IEEE 724:

```
.085

Step 1: Set the sign bit.

   in this case it is positive, so 0.  (-1^0) = 1

Step 2: Write .085 in a base-2 notation: n * 2^y, where 1>=n<=2

       0.085 / 2^-1 = .17
       0.085/ 2^-2 =  .34
       0.085 / 2^-3 = .68
       ---> 0.085/2^-4 = 1.36 <----
         1.36 * 2^-4

Step 3, compute the exponent:
    e = E - bias
    e + bias = E
    (-4) + (127) = 123
```

Step 4: approximate the fractional part in binary form:

.36 =
(0 * 2^-1) + (1*2^-2) + (0 * 2^-3) + (1 * 2^-4) + (1 * 2^5) + (1 * 2^6)
(0) + (.25) + (0) + (.0625) + (.03125) + (.015625) â€¦

Step 5: build binary string:
sign bit: 0
exponent: 0111 1011
fraction: 010 1110 0001 0100 0111 1011

0011 1101 1010 1110 0001 0100 0111 1011

To Hex:
0x3DAE147B