CS 61C Fall 2020

Floating Point

Discussion 3: September 14, 2020

1 Pre-Check

This section is designed as a conceptual check for you to determine if you conceptually understand and have any misconceptions about this topic. Please answer true/false to the following questions, and include an explanation:

1.1 True or False. The goals of floating point re to have a large range of values, a low amount of precision, and real arithmetic results

False. Although floating point DOES
- Provide support for a wide range of values. (Both very small and very large)
- Help programmers deal with errors in real arithmetic because floating point can represent + ∞, -∞, NaN (Not a number)

Floating point actually has HIGH precision. Recall that precision is a count of the number of bits in a computer word used to represent a value. Floating point helps you keep as much precision as possible because we have so much freedom to interpret our bits as whatever negative powers of 2 are useful for specifying the number.

1.2 True or False. The distance between floating point numbers increase as the absolute value of the numbers increase.

True. The uneven spacing is due to the exponent representation of floating point numbers. There are a fixed number of bits in the significand. In IEEE 32 bit storage there are 23 bits for the significand, which means the LSB is 2^{-22} times the MSB. If the exponent is zero (after allowing for the offset) the difference between two neighboring floats will be 2^{-22}. If the exponent is 8, the difference between two neighboring floats will be 2^{-14} because the mantissa is multiplied by 2^8. Limited precision makes binary floating-point numbers discontinuous; there are gaps between them.

1.3 True or False. Floating Point addition is associative.

False. Because of rounding errors, you can find Big and Small numbers such that:
(Small + Big) + Big != Small + (Big + Big)
FP approximates results because it only has 23 bits for Significand

2 Floating Point

The IEEE 754 standard defines a binary representation for floating point values using three fields.

- The *sign* determines the sign of the number (0 for positive, 1 for negative).
- The *exponent* is in **biased notation**. For instance, the bias is -127 which comes from -$(2^{8-1}-1)$ for single-precision floating point numbers.
- The *significand* or *mantissa* is akin to unsigned integers, but used to store a fraction instead of an integer.

The below table shows the bit breakdown for the single precision (32-bit) representation. The leftmost bit is the MSB and the rightmost bit is the LSB.

1	8	23
Sign	Exponent	Mantissa/Significand/Fraction

For normalized floats:
Value $= (-1)^{Sign} * 2^{Exp+Bias} * 1.\textbf{significand}_2$
For denormalized floats:
Value $= (-1)^{Sign} * 2^{Exp+Bias+1} * 0.\textbf{significand}_2$

Exponent	**Significand**	**Meaning**
0	Anything	Denorm
1-254	Anything	Normal
255	0	Infinity
255	Nonzero	NaN

Note that in the above table, our exponent has values from 0 to 255. When translating between binary and decimal floating point values, we must remember that there is a bias for the exponent.

2.1 How many zeroes can be represented using a float?

2; one "positive" zero and one "negative" zero.

2.2 What is the largest finite positive value that can be stored using a single precision float?

`0x7F7FFFFF` $= (1 + (1 - 2^{-23})) * 2^{127}$

The mantissa for the largest value will be 23 1's. This corresponds to a value of

$$.11\ldots1 = 2^{-1} + 2^{-2} + \cdots + 2^{-23} = 2^{-23}(2^{22} + 2^{21} + \cdots + 1)$$

Here, we apply the formula that $\sum_{i=0}^{n-1} 2^i = 2^n - 1$, so we have that the mantissa is

$$2^{-23}(2^{22} + 2^{21} + \cdots + 1) = 2^{-23}(2^{23} - 1) = 1 - 2^{-23}$$

We have $(1 + (1 - 2^{-23})) * 2^{127}$. Since this is a normalized number, it has a 1 to the left of the decimal point.

2.3 What is the smallest positive value that can be stored using a single precision float?

`0x00000001` $= 2^{-23} * 2^{-126}$

2.4 What is the smallest positive normalized value that can be stored using a single precision float?

`0x00800000` $= 2^{-126}$

2.5 Cover the following single-precision floating point numbers from binary to decimal or from decimal to binary. You may leave your answer as an expression.

- 0x00000000

  0

- 8.25

  0x41040000

- 0x00000F00

  $(2^{-12} + 2^{-13} + 2^{-14} + 2^{-15}) * 2^{-126}$

- 39.5625

  0x421E4000

- 0xFF94BEEF

  NaN

- -∞

  0xFF800000

3 More Floating Point Representation

Not every number can be represented perfectly using floating point. For example, $\frac{1}{3}$ can only be approximated and thus must be rounded in any attempt to represent it. For this question, we will only look at positive numbers.

3.1 What is the next smallest number larger than 2 that can be represented completely?

For this question, you increment the number by the smallest amount possible. This is the same as incrementing the significand by 1 at the rightmost location.
$(1 + 2^{-23}) * 2 = 2 + 2^{-22}$

3.2 What is the next smallest number larger than 4 that can be represented completely?

For this question, you increment the number by the smallest amount possible. This is the same as incrementing the significand by 1 at the rightmost location.
$(1 + 2^{-23}) * 4 = 4 + 2^{-21}$

3.3 Define stepsize to be the distance between some value x and the smallest value larger than x that can be completely represented. What is the step size for 2? 4?

This would be the amount added in part 1. This gives 2^{-22} and 2^{-21}.

3.4 Now let's see if we can generalize the stepsize for normalized numbers (we can do so for denorms as well, but we won't in this question). If we are given a normalized number that is not the largest representable normalized number with exponent value x and with significand value y, what is the stepsize at that value? Hint: There are 23 significand bits.

Here we need to generalize the solution we got in 1 and 2. However, this is the same approach just increment the signifcand by the 1.
$curr\_number = 2^{x-127} + 2^{x-127} * y$
$next\_number = 2^{x-127} + 2^{x-127} * y + 2^{x-127} * 2^{-23}$
$stepsize = next\_number - curr\_number = 2^{x-150}$

3.5 Now let's apply this technique. What is the largest odd number that we can represent? Part 4 should be very useful in finding this answer.

To find the largest odd number we can represent, we want to find when odd numbers will stop appearing. This will be with step size of 2.

As a result, plugging into Part 4: $2 = 2^{x-150} \rightarrow x = 151$
This means the number before $2^{151-127}$ was a distance of 1 (it is the first value whose stepsize is 2) and no number after will be odd. Thus, the odd number is simply subtracting the previous step size of 1.This gives,
$2^{24} - 1$

$(1 + 0.111\ldots) \times 2^{152-127}$
$= (1 + (1 - 2^{-23})) \times 2^{25} = 2^{24} - 1$