15213 - Lecture 4 - POGIL Activity (Floating-point)

Introduction

In this activity you will learn about floating point operations.

Before you begin, take a minute to assign roles to each member in your group. Try to switch up the roles as much as possible: please don't pick a role for yourself that you have already done more than once. Below is a summary of the four roles; write the name of the person taking that role.

If your group only has three members, combine the roles of Facilitator and Process Analyst.

- **Facilitator**: Reads question aloud; keeps track of time and makes sure everyone contributes appropriately.
- **Quality Control**: Records all answers & questions, and provides team reflection to team & instructor.
- **Spokesperson**: Talks to the instructor and other teams. Compiles and runs programs when applicable.
- **Process Analyst**: Considers how the team could work and learn more effectively.

Fill in the following table showing which group member is performing each role:

Role	**Person**
Facilitator	
Quality Control	
Spokesperson	
Process Analyst	

Model 1: What is floating point?

1. Write 15213 in scientific notation. Describe the different parts of the notation.
2. How many digits were required to write the previous number? There is more than one valid answer depending on how you write scientific notation, but be consistent in this section.
3. If we add 3000 to the number, what is the result? How many digits?
4. If we multiply the sum by 1000, what is the result? How many digits?
5. What is the smallest number greater than zero that can be written, using the same number of digits as 15213 was in scientific notation?
6. What is the largest number that we can write using the same number of digits in scientific notation?
7. If we wrote out each of these numbers without scientific notation, would the decimal point be in the same place?

Model 2: Binary Scientific Notation

1. In the previous model, how many digits were to the left of the floating decimal point?
2. Write out the following binary numbers in scientific (binary) notation. In this notation, we use powers of 2 rather than powers of 10.

Value	Binary	Answer
23	10111	
5 3/4	101.11	
2 7/8	10.111	
1 7/16	1.0111	

3. What value(s) is to the left of the binary point in each number?

Model 3: IEEE Representation

In this model, we are going to learn the IEEE floating point representation. Subsequent models will cover the special cases to this representation.

Value $= (-1)^s * 1.f * 2^E$

1. What does s represent in the equation? If the value for s is 1, what happens to the number?
2. The fractional bits, f, have an implicit 1 in the front. This pattern is termed, "normalized". Returning to the table in model 2, what are the values for f for each number?
3. The IEEE format provides 23 fractional bits for single-precision floats and 52 for double-precision floats. With E equal to 23, what is the difference in decimal between a value for f of 0x000000 and 0x000001 (note these are all 23 bits of f)?
4. With the 8 exponent bits of a single-precision float, what is the smallest, positive number with 0x01 as the exponent (E)? Is this value greater or less than one?
5. IEEE subtracts a bias from the exponent bits to adjust the range of representable values. For the single-precision floats, the bias is 127. With $E = exp - bias$, what exponent bits (exp) is required for the number in the previous question?
6. 15213_{10} is 11101101101101_2. What is this number as a single-precision float (i.e., what is the fractional component and exponent bits exp)?
7. Double-precision floats (i.e., doubles) use 11 bits for the exponent and have a bias of 1023. If the exponent bits range from 0x001 to 0x7FE (all bits but the last set), what is the range of E for doubles?

In the IEEE representation, the bits are placed: `seee*ffff*`, with the appropriate number of exponent bits and fractional bits for that type (float, double, etc).

Model 4: Extreme Exponents

1. Suppose we have 5 bits, x.xxxx. If this bit pattern is a normalized value, then what is the smallest number that we can store?
2. For what you know of the representation (i.e. normalized) so far, can you represent 0 with the IEEE format?

One of the goals of the IEEE representation was to make the bit pattern of 0 be the value of 0. Thererfore, in the IEEE representation, an exponent bit value of 0 is made a special case. If the exponent value is 0, then the biased exponent is the same as having an exponent value of 1. Furthermore, the fractional part is "denormalized", such that it no longer has a leading 1.

3. How many representations for zero are there with denormalized floats?
4. If the 5-bit pattern from before does not need to be normalized, then what is the smallest positive number that you can represent with these bits?

The maximum exponent pattern is also a special case. If the exponent bits are all 1s, then the fractional bits take on different meanings. With fractional bits of all 0s, the value is either $+inf$ or $-inf$. If the fractional bits are non-zero, then the value is NAN, which is short for not-a-number.

5. Which is larger, 2^{127} or $+inf$? Does this ordering hold when these numbers are floats (i.e., if just the bit patterns are compared)?
6. Consider the largest denormalized positive number with the smallest normalized positive number?
   How do these numbers' bit patterns compare and order?

Model 5: Addition and Multiplication

To perform addition with floating point numbers, the hardware first works to line up the binary points. And then it can add the significands together. Finally, the result must be renormalized and a new exponent computed.

1. Compute the sum of the following binary numbers: $1.010 * 2^2 + 1.110 * 2^3$. Apply each step in turn.
2. How many fractional bits were required for the result of the previous question?
3. There are several possible rounding schemes for floating point values. There are two components of rounding. First, is what to do in general? Should the float be rounded up, down, to zero, or to nearest? With the following, round the following integer and floating point expressions to a whole number.

Value	Number
4/3	
4/3.0	
5/3	
5/3.0	

4. The second component is what to do about ties with round to nearest. So should 9/2.0 be 4 or 5? The default IEEE scheme is round to nearest even. Apply it to the following values for a system that only has three bits for the final values.

Value	Binary	Rounded	Final
1 3/32			
1 5/32			
1 7/8			
1 5/8			

5. What is the rounded result from question 1, if you have only 4 bits (not including exponent)?

With multiplication, the hardware can both add exponents and multiply significands in parallel. The resulting product must be renormalized, which may then update the sum of the exponents.

6. Compute the following product: $64 * 32$.
7. Now compute $2^6 * 2^5$. Which product was easier to compute?

Model 6: Simple Floating-point

For the following model, we are going to use a smaller representation. 1 bit for the sign. 3 bits for the exponent, with a bias of 3. And 4 fractional bits.

1. Excluding 0 and inf, what is the largest absolute value that we can represent with these bits? Smallest?
2. Term the largest value L and smallest S. What is the result of $L + S - L$? What is this behavior called, and why does it happen?
3. Given the following value 0x5C, what is this 8-bit float's decimal value? What is the binary representation of this value?
4. Compute the sum of the following floats: `0x5C + 0x43`.
5. Compute the product of the following floats: `0x5C * 0x43`.

Model 7: Review

1. (advanced) Will the following code ever terminate? If so, what value will sum contain?

```
float sum = 0.0f;
float inc = 1.0f;
float tsum;

do {
    tsum = sum;
    sum += inc;
} while (tsum != sum);
```

2. If the floats and constants are changed to ints, how (if at all) do your answers change?

Casting between *int*, *float* and *double* can change their bit representations. On current systems, ints and floats use 32 bits, while double-precision floats use 64 bits.

3. Look back to the previous questions, did the two versions behave the same? What does this imply about converting from int to float?
4. Is there the same behavior when converting from int to double? Why or why not?