|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 🔑 **Essential** 🔑 | | | |  | 🏆 **Enhanced** 🏆 | | | |
|  |  |  |  |  |  |  |  |  |
| \_\_\_\_\_ | \_\_\_\_\_ | \_\_\_\_\_ | \_\_\_\_\_ |  | \_\_\_\_\_ | \_\_\_\_\_ | \_\_\_\_\_ | \_\_\_\_\_ |

**Score: \_\_\_\_\_**

**08 – Fractional Numbers**

**Activities**

COMP256 – Computing Abstractions

Dickinson College

Spring 2022

Prof. Grant Braught

**Name:**

**Introduction:**

Today’s class introduced two ways that binary can be used to represent fractional or non-whole numbers. We saw that we can use a fixed-point representation or a floating point representation. With those representations, we saw that we are not able to represent every real number exactly and thus we must consider the issues of precision and rounding. In today’s activities you will gain more experience working with fixed and floating point values and see a little of how they are optimized in practice. We’ll wrap up by looking at floating point in the real-world.

For all questions involving number conversions you will need to show sufficient work using steps similar to those used in class to demonstrate your understanding. Answers without sufficient work will receive a ✔︎- score and will require resubmission.

**Fixed Point Binary:**

🔑 1. Give the decimal value of the following fixed-point binary number.

001101.0111002XP

🔑 2. Give a fixed point binary representation of the following decimal number using 12 bits with 6 bits to the right of the binary point.

15.57812510

**Precision and Rounding:**

🔑 3. Using an 8-bit fixed point representation with 5 bits to the right of the binary point and the following decimal number:

6.3310

a. Give the binary and decimal value of the nearest under-estimate for the above decimal value.

b. Give the binary and decimal value of the nearest over-estimate for the above decimal value.

c. By default, most programming languages will round numbers that cannot be represented exactly to the nearest representable value – this is called *round to nearest*. This nearest value will be either the underestimate or the overestimate, whichever is closest to the desired value\*. What base 10 value do we get if we use round to nearest when representing 6.3310 in 8-bit fixed point with 5 bits to the right of the binary point?

\* For technical reasons, what is actually done is a bit more complicated. It is not required reading, but if you are interested in more details about the precise rounding mode used you might find the article *How to Round Binary Numbers* by Max Koretskyi.

* <https://indepth.dev/posts/1017/how-to-round-binary-numbers>

**Floating Point Values:**

For the following questions use the 14-bit floating point model from the class slides (I.e. the one with 1 sign bit, 5-bit exponent in two’s complement and 8-bit significand).

🔑 4. Give the base 10 value of the following binary number. Show sufficient work to demonstrate how you arrived at your answer.

1 11001 1011 01002FP-14

🔑 5. Give the binary representation of the following base 10 value. If the significand will not fit in 8 bits round it to its nearest underestimate.

78.562510

🔑 6. Like our representations for whole numbers, the number of bits and how they are used in a floating point representation determines the range of values that can be represented. Operations that result in values outside of this range create *arithmetic overflow*. This question explores the range of values for the 14-bit floating point system from class.

a. Give the binary and base 10 values of the largest positive value that can be represented using this floating point model. Hint: Figure out the binary value and then convert it to base 10.

b. Using what you know about our 14-bit floating point model and your answer to part a, give the binary and base 10 values of the largest negative number that can be represented using this floating point model. Hint: You should not have to convert the binary to find the base 10 value here.

🏆 7. Because a floating point system cannot represent every number this also means that there are numbers that are too small (i.e. too close to zero) to be represented. When an operation results in one of these values it is called *arithmetic underflow*. This question explores the smallest values that can be represented in the 14-bit floating point system from class.

a. Give the binary and base 10 value of the smallest value greater than zero that can be represented using this floating point model. Be sure to remember that the significand must be in normalized form.

b. Give the binary and base 10 value of the smallest value less than zero that can be represented using this floating point model. Hint: Like in #6b, this answer should be easily derivable from your answer to part a.

**Back to Homework #1:**

🏆 8. Recall back in homework #1 when we saw that the expression (int)(4.35\*100) evaluated to 434 rather than 435 as would be expected. This question demonstrates a similar behavior using a simpler example that makes it easier to work out by hand.

a. If everything were perfect, what value would you expect the following expression to evaluate to?

(int)(6.23 \* 2)

b. Give the fixed point representation of 6.23 using a total of 6 bits, with 3 bits to the right of the binary point. You will not be able to represent this value exactly. Use the nearest overestimate, which is what round to nearest would do in this case.

c. Recall that when we move the binary point to the right, each place that we move it multiplies the value by 2. Perform the multiplication by 4 in binary, by rewriting your answer from part b with the binary point moved two bits to the right.

d. Recall, that a type cast to int discards any fractional part of a number. Type cast the binary value from part c to int by rewriting it without the fractional part that appears to the right of the binary point.

e. What is the base 10 value of your answer in part c if it is interpreted as unsigned binary?

f. Was your answer to part e what was expected? Explain why it was or was not.

So, as said this isn’t the exact same example we saw running in Java in homework #1. However, the issue we see is caused by the same necessity of rounding of values that can’t be represented exactly given the number of bits available. If there are more bits available, the issues become smaller and rarer, but they don’t ever go away.

**Floating Point in the Real-World:**

🔑 9. Modern processors all use a floating point representation similar to the one above to represent fractional numbers (e.g. float or double values in Java/C/C++). Complete the table below with information about the float and double data types in Java. **Give base 10 values in the table using scientific notation**. Use your favorite search engine to find these answers.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  | **Data Type** | **Number of Bits** | **Largest Positive** | **Smallest Positive** |  |
|  | float |  |  |  |  |
|  | double |  |  |  |  |
|  |  |  |  |  |  |

🔑 10. A weak understanding of fixed and floating point values can be fatal when doing system design. On February 25th 1991, 28 people were killed in a missile attack due to a failure of the Patriot Missile Defense System.

Read the articles below to gain a general understanding of the problem.

* Michael Barr’s article, *Lethal Software Defects: Patriot Missile Failure* gives a good high-level overview of the event and hints at the underlying cause.
  + <https://barrgroup.com/software-expert-witness/articles/case-study-lethal-software-defects-patriot-missile-failure>
* Douglas Arnold, on his page *The Patriot Missile Failure* gives a short explanation of the underlying problem and its relationship to fixed and floating point representations.
  + <http://www-users.math.umn.edu/~arnold/disasters/patriot.html>
* You may also pick up some additional helpful insight from Dinesh Manocha’s page on the *Patriot Missile Software Problem* where he shows some sample calculations.
  + <http://www.cs.unc.edu/~smp/COMP205/LECTURES/ERROR/lec23/node4.html>

Give a few sentences of your own words that briefly describe what happened and how it is related to the content of today’s class.

**Hey… What about Zero?**

🏆 11. It turns out that the number 0 presents an interesting special case for floating point representations. This question explores this special case.

a. Recall that as we have defined the floating point system thus far, the significand is always in normalized form. This actually implies that the value 0.0 cannot be represented. Give a few sentences that explain why 0.0 cannot be represented when the significand is in normalized form.

b. To address this issue, most floating point systems use the convention that when the exponent takes on its largest negative value, the significand will be interpreted as *denormalized* (i.e. it not in normalized form). In denormalized form the bit to the left of the binary point is a 0instead of a 1. Assume that our 14-bit floating point system supports denormalized forms as just described. Give the binary representation of 0.0 in this system.

c. If being able to represent 0.0 isn’t important enough on its own, supporting denormalized form in a floating point sustem has an additional advantage as well.

i. What is the base 10 value of the following denormalized floating point number in our 14-bit system? You can express your value as a power of 2 if you like.

0 10000 0000 00012FP-14

ii. In question #7a above you found the smallest normalized value that can be represented in our system. Is the value you found in part I larger or smaller than what you found in part #7a? Based on that, what is the additional advantage that we receive by supporting denormalized forms in a floating point system?

Optional: To help me improve and scope these activities for future semesters please consider providing the following feedback.

a. Approximately how much time did you spend on this activity outside of class time?

b. Please comment on any particular challenges you faced in completing this activity.