CIS 351 -- Intro to Floating-point Numbers

1. Assume you are working with a fixed-point binary system with 3 places before and after the “decimal point”, such as

   ___ ___ ___ . ___ ___ ___

   Convert the following decimal numbers into this system. You can simply truncate any leftover value if the number does not convert exactly.

   a. 7

   b. 4.5

   c. 0.00125

   d. 5.375

   e. 7.25125

2. Try to convert the decimal number 0.8 to binary. Stop when you see a pattern (or get bored).

3. What is the smallest nonzero number that can be represented in this system? What is the largest number?

4. In the system from page 1, how many numbers can be represented? (If you're not sure, think about how many bits we have and how many things we can represent in general.)

5. What if you had 12 bits instead of 6, and they were still split evenly on each side of the point? What would be the largest and smallest non-zero numbers you could represent? How many numbers could you represent?

6. Given a 4-bit fixed-point system, with numbers of the form

__ __ . __ __

draw a number line showing the points you can represent.

0 ---------- 4

7. Now for something completely different. Imagine you were trying to measure the width of this classroom, and your measurement was off from the correct value by 1 foot. Next, imagine you were to measure the diameter of the earth, and this measurement was also incorrect by 1 foot. Would you say one of these measurements was better? Why or why not?

8. You should have found that two numbers from (1) were not representable in the system. What was the *error* in representing those numbers (i.e. how far off was the binary value from the original number)? Would you say that one answer was better than the other?

9. Assume you are working with the 4-bit system from (6). Representing any real number in the range [0, 4] in this system has a maximum *absolute error* (maximum difference between approximate and true number). What is it? Think about your number line, and how far any number on the line can be from the nearest representable point.

10. Can you generalize the above maximum-error result to fixed-point systems of other sizes? For example, what would be the maximum error for representing any number in [0, 8] in a 6-bit system?

11. The key to moving from fixed-point to floating-point systems is the use of scientific notation. Write the following numbers in scientific notation in binary.

a. 1101.10

b. 101

c. 1100110

d. 10.11

12. Convert the following decimal numbers to scientific notation in binary.

a. 5.875

b. 10.25

c. 8.5

d. 65

Next, we need to move to using a fixed number of bits, as we did with fixed-point numbers. To keep track of numbers in scientific notation, we need to keep track of the *mantissa* (the part with the ones and zeros) and the *exponent*. We will represent the exponent as a *two's complement* integer value to allow for negative exponents. (The way exponents are actually stored in your computer is similar, but slightly different).

If we have, for example, 6 bits to work with, we have a choice to make. We can allocate more or fewer of the bits for the mantissa, and the rest go to the exponent. A few different possible systems are below.

1. __ __ __ __ __ * 2 —

1. __ __ __ __ * 2 — —

1. __ __ __ * 2 — — —

13. Assume we are working with the third system above (3 bits for the mantissa, 3 for the exponent). What exponent values can we represent? (Not the binary values like 110 – the integer values they represent)

14. Convert the following numbers to this system. If the system does not have enough precision to represent the number exactly, truncate any extra bits.
    a. 2

    b. 9

    c. 0.75

    d. 8.125

    e. 16

15. What is the largest number representable in the above system? What is the smallest nonzero number representable in the system? How many numbers can you represent?

16. Consider the three different formats from the previous page with differing numbers of bits allocated to the mantissa and the exponent. For the first and third examples, with 1 bit and 3 bits allocated to the exponent, respectively, give an example of a number that can be represented in one system but not the other.

17. For the three different formats, can they represent a different amount of numbers? Why or why not?

18. Given a 4-bit floating-point system with numbers of the form

$$1.\_\_ * 2^{--}$$

draw a number line showing the points you can represent.

0 ---------- 4

19. What difference do you notice about the number line from (18) compared to the previous number line you drew?

20. In this 4-bit system, what is the difference between 1 and the next (larger) number? What is the difference between 2 and the next number? What about 0.5 and the next number?

21. Which would make the gap between successive numbers smaller – more bits for the mantissa, or more bits for the exponent?

22. Given a 4-bit floating-point system with numbers of the form

$$1.\_\_ * 2^{---}$$

draw a number line showing the points you can represent.

0 ---------- ?

23. Think back to the system from (18). What is the largest *absolute* error for representing a number in this system? It may help to think about your answer from (20).

24. For question (20), you should have found that the difference between successive numbers in that system is always

$$0.01 * 2^X$$

for some X. Think about why that is the case. Using this same principle, what is the difference between successive values in the system from (22)?

25. In floating-point systems, the gaps between numbers grow as the numbers get bigger, as you saw in (18), (20), and (22). This can lead to larger *absolute errors* than we saw in fixed-point numbers. However, because floating-point systems have small gaps between small numbers and large gaps between large numbers, they have a bounded *relative error*.

Consider your answers from (20). What do you get if you divide [the gap between 2 and the next number] by 2? What about dividing [the gap between 0.5 and the next number] by 0.5?

26. Hopefully you found that [gap between numbers]/[size of number] was the same for these examples. The number that results is called *machine epsilon*. (Note: There are various definitions of machine epsilon, but they all result in a similar value). This number is important because it bounds the *relative error* for representing a number in floating point.

Next, you are given a few numbers that are not exactly representable in the 4-bit system from (22). Compute the relative errors of converting these numbers to this 4-bit system. Use the following procedure to compute the relative error:

a. Convert the number to 4-bit floating-point and ignore any remainder.
b. Convert the 4-bit floating-point number back to decimal (it will not be exactly equal to the original).
c. Subtract the answer to (b) from the original number
d. Divide the answer to (c) by the original number

a. 1.75

b. 3.5

c. 15.5

d. 0.625

27. Given the errors that you found, what would you guess the value of machine epsilon to be for the system from (22)? Considering this, along with your answer to (24), how do you think machine epsilon can be determined for a given floating-point system?

28. Assume you are working with double-precision IEEE floating-point numbers, so machine epsilon is 1e-16. What are the approximate relative errors of converting each of the following numbers to a machine number? What are the approximate absolute errors? (You do not actually have to convert the numbers – since it is a 64-bit system, I would suggest you not try to do so.)

    a. 1.78932837578923847562398437 9

    b. 100.79882347293847928 36

    c. 5,435,689.23498732

    d. 123,456,789,123,456,789

29. Working with a fixed-point system, we would probably never get an absolute error as big as we found in (8d). However, a fixed-point system would still generally not be useful for representing that number. Why not?