Homework

CSC 28 – Discrete Structures for Computer Science

This work is not turned in but should be done before any quiz on it. To maximize your quiz performance: attend lectures, do readings, start homework early, ask for help when you need it, review problems and solutions before the quiz. If anything in this assignment does not make sense, please ask for help.

Read) *Floating-point representation* on the course webpage. Familiarize yourself with other module resources on the course webpage and read them as needed.

1) For each of the following integers, give the binary representation as an 8-bit signed integer, 8-bit unsigned integer, and 8-bit excess-127. If it is not possible, briefly explain why not.

$$-128, -127, -100, -1, 0, 1, 100, 127, 128$$

2) What real number does the 32-bit binary string 1100 0001 1010 1000 0000 0000 0000 0000 represent according to IEEE-754 single-precision floating-point representation?

3) Write $2\frac{3}{8}$ in 32-bit IEEE-754 single-precision representation. Briefly explain each step as you go.

We've seen that single-precision floating point representation splits-up 32-bits as: 1 bit for the sign, 8 bits for the exponent and 23 bits for the significand. Other divisions of the bits are possible: double-precision uses 1 bit for the sign, 11 bits for the exponent and 52 bits for the significand. When the exponent uses n bits, it is expressed in excess-$(2^{n-1}-1)$ notation (ie, when the exponent is 8 bits then we use excess-(2^7-1) notation and when the exponent is 11 bits then we use excess-$(2^{10}-1)$ notation). Recall that if n bits are being interpreted as excess-$(2^{n-1}-1)$, then the n bits are first interpreted as unsigned binary and then $2^{n-1}-1$ is subtracted.

4) What is the bit-pattern and *exact* values of the largest and smallest positive *double*-precision numbers? Ignore the fact the IEEE floating point has defined some special bit patterns for things like zero and infinity.

5) Let's pretend there exists a "mini-precision" floating point representation that is defined just like single- and double-precision except that it uses only 5 bits: 1 bit for the sign, 2 bits for the exponent and 2 bits for the significand. (i) In what excess would the exponent be represented? (ii) List as fractions (reducing as much as possible) all numbers that could be represented using mini-precision floating point. Ignore the fact the IEEE floating point has defined some special bit patterns for things like zero and infinity. (Because there are 5 bits, there are $2^5 = 32$ bit patterns, each representing a different number. That sounds like a lot of fractions to write, but in fact they are easy to write down once you notice the patterns.)

Solutions

Please notify me of any errors you find. Not all solutions are given. If you need help, ask.

1) *For each of the following integers, give the binary representation as an 8-bit unsigned integer, 8-bit signed integer, and 8-bit excess-127. If it is not possible, briefly explain why not.*

$$-128, -127, -100, -1, 0, 1, 100, 127, 128$$

The range of values that an 8-bit unsigned integer can represent is 0 through 255. So, all the negative values are not representable. For each of the others, we simply add 127 to the value and write the result as an unsigned 8-bit binary number. $0 = 00000000$, $1 = 00000001$, $100 = 01100100$, $127 = 01111111$, $128 = 10000000$.

The range of values that an 8-bit signed integer can represent is -128 through 127. So, 128 is not representable. For each of the remaining non-negative values, the 8-bit signed representation is the same as the 8-bit unsigned one: $0 = 00000000$, $1 = 00000001$, $100 = 01100100$, $127 = 01111111$. For each of the negative numbers, the first bit must be 1 and contributes -128. The question then becomes what unsigned 7-bit value do we need to add to -128 to make the desired number. For example, $-128 + x = -100$ means $x = 128 - 100 = 28$ which is 0011100. Thus -100=10011100. 128-128=0=0000000, so -128=10000000. 128-127=1=0000001, so -127=10000001. 128-1=127=1111111, so -1=11111111.

The range of values that 8-bit excess-127 can represent is 0-127 through 255-127, or -127 though 128. So, -128 is not representable. For each of the others, we simply add 127 to the value and write the result as an unsigned 8-bit binary number. $-127 + 127 = 0 = 00000000$, $-100 + 127 = 27 = 00011011$, $-1 + 127 = 126 = 01111110$, $0 + 127 = 127 = 01111111$, $1 + 127 = 128 = 10000000$, $100 + 127 = 227 = 11100011$, $127 + 127 = 254 = 11111110$, $128 + 127 = 255 = 11111111$.

2) *What real number does the 32-bit binary string 1100 0001 1010 1000 0000 0000 0000 0000 represent according to IEEE-754 single-precision floating-point representation?*

Let's refer to the 32 bits as $b_1 \ldots b_{32}$. First, we split the bits into parts:

$$1 \quad 10000011 \quad 01010000000000000000000 \, .$$

Next, we extract s, e, f. Because $b_1 = 1$, we know $s = 1$ and the final result will be negative. Because 10000011 is $128 + 2 + 1 = 131$, we know $e = 131 - 127 = 4$. Finally, we know $f = 1/4 + 1/16 = 5/16$ because only b_{11} and b_{13} are 1. Plugging these values into the equation gives us

$$\begin{aligned} (-1)^s \times (1+f) \times 2^e &= (-1)^1 \times (1 + \frac{5}{16}) \times 2^4 \\ &= -\frac{21}{16} \times 16 \\ &= -21 \, . \end{aligned}$$

3) *Write $2\frac{3}{8}$ in 32-bit IEEE-754 single-precision representation. Briefly explain each step as you go.*

First, because the number is positive, we know $s = 0$. Next, we write the magnitude of our number in the form $|x| \times 2^0$.

$$2\frac{3}{8} \times 2^0$$

Since $2\frac{3}{8}$ is 2 or more, we must halve it until it's less than 2, increasing the exponent by one each time.

$$\begin{aligned} 2\frac{3}{8} \times 2^0 &= \frac{19}{8} \times 2^0 \\ &= \frac{19}{16} \times 2^1 \end{aligned}$$

This can be rewritten $(1 + \frac{3}{16}) \times 2^1$, which tells me $e = 1$ and $f = 3/16$. Because e needs to be expressed in excess-127 notation, the correct bit pattern is the binary representation of $e + 127$. The value f needs to be expressed as a fixed-point fraction (ie, sum of powers of 1/2), so we need a 1 in the 1/8 and 1/16 position. The final bit pattern is: 0 10000000 00110000000000000000000.

4) *What is the bit-pattern and exact values of the largest and smallest positive double-precision numbers? Ignore the fact the IEEE floating point has defined some special bit patterns for things like zero and infinity.*

The magnitude is maximized when e and f are both maximized. So, 0 followed by 63 ones is going to be maximal. In this case, e is 11 ones as binary $(2^{11} - 1)$ excess 1023, so $e = 1024$. The value f is almost 1, but not quite. The last bit in f has place value $1/2^{52}$, so $f = 1 - 1/2^{52}$. Multiplying $(1 + f) \times 2^e$ gives us $2^{1025} - 2^{972}$. Similarly, the magnitude is minimized when e and f are minimized using all zeros. In this case $e = -1023$ and $f = 0$. So, $(1 + f) \times 2^e = 1/2^{1023}$.

5) *List as fractions (reducing as much as possible) all numbers that could be represented using 5-bit floating point. Ignore the fact the IEEE floating point has defined some special bit patterns for things like zero and infinity.*

The two bits for the exponent are in excess-1 notation, so the values e can be are 2, 1, 0 and -1. With two bits, f can only take on the values $0, \frac{1}{4}, \frac{1}{2}$ and $\frac{3}{4}$. This means the 32 values that are achievable are exactly the 32 ways you can pick one of $\{-1, +1\}$ and one of $\{2^2, 2^1, 2^0, 2^{-1}\}$ and one of $\{1, 1\frac{1}{4}, 1\frac{1}{2}, 1\frac{3}{4}\}$ and multiply then together. This results in the possible values $\pm\frac{1}{2}$, $\pm\frac{5}{8}$, $\pm\frac{3}{4}$, $\pm\frac{7}{8}$, ±1, $\pm1\frac{1}{4}$, $\pm1\frac{1}{2}$, $\pm1\frac{3}{4}$, ±2, $\pm2\frac{1}{2}$, ±3, $\pm3\frac{1}{2}$, ±4, ±5, ±6 and ±7.