AMSC 660 Assignment 1

Jiaqi Leng

September 5, 2019

1 Problem 1: Floating point representation

(a) The term used in [1] to call the mantissa is **significand**.
(b) The IEEE standard requires that a floating-point representation be normalized because such a representation is unique. But this restriction makes it impossible to represent zero.

The IEEE solution to represent zero is $1.0 \times \beta^{e_{\min}-1}$. This representation is compatible with the representation of denormalized numbers.
(c) The IEEE solution to represent infinity (i.e., ∞) is $1.0 \times \beta^{e_{\max}+1}$, that to represent NaN is $1.f \times \beta^{e_{\max}+1}$, where f is a fraction that is non-zero. For example, $1.1 \times \beta^{e_{\max}+1}$ can be a representation of NaN.
(d) The exponent can be positive or negative, so we need to care about the sign of exponent when representing the exponent. To avoid representing the sign explicitly, IEEE standard adopts the *biased representation*: a number (127 for single precision, 1023 for double precision) is added to the exact/unbiased exponent so that the sum is always non-negative. The added number is called *bias*, and the unsigned sum is called *biased exponent*. To retrieve the unbiased exponent, one just subtracts the bias from the biased exponent.

2 Problem 2: Invalid operations

(a) $\infty/0 = \infty$, $0/\infty = 0$, and ∞/∞ is not defined (NaN).

We can justify our answer by sequence approximation. Suppose there are two positive sequences, $a_n \to \infty$ and $b_n \to 0$. So $\infty/0$ can be realized as

$$\infty/0 = \lim_{n\to\infty} \frac{a_n}{b_n}.$$

For any large $L > 0$, there are two integer M_1, M_2 such that

$$a_n > \sqrt{L} \text{ for all } n \geq M_1,$$

$$0 < b_n < \sqrt{L}^{-1} \text{ for all } n \geq M_2,$$

hence for all $n \geq \max\{M_1, M_2\}$, $a_n/b_n > L$. It follows that $a_n/b_n \to \infty$ as $n \to \infty$.

A similar argument shows $b_n/a_n \to 0$ as $n \to 0$, implying that $0/\infty = 0$.

If we have another sequence $c_n \to \infty$ as $n\infty$, a_n/c_n can have various behaviors. For example, if $a_n = c_n = n$, then $a_n/c_n \equiv 1$ for all $n \geq 1$; but if $a_n = n^2, c_n = n$, then $a_n/c_n = n \to \infty$ as $n \to \infty$. As a result, ∞/∞ is not defined.
(b) For a finite non-negative value a, $a/\infty = 0$. We use the same idea as in part (a). Let $a_n \to \infty$ as $n \to \infty$, then given any small $\epsilon > 0$, we will be able to find an integer $M > 0$ such that for all $n \geq M$, $a_n > \frac{1}{\alpha\epsilon}$, so

$$\frac{\alpha}{a_n} < \epsilon \text{ for all } n \geq M.$$

Then $\alpha/a_n \to 0$ as $n \to \infty$, for all $\{a_n\}$ diverging to positive infinity. It turns out that $\alpha/\infty = 0$.
(c) If $R_1 = 0$ and $R_2 \neq 0$, then the formula

$$\frac{R_1 R_2}{R_1 + R_2} = \frac{0}{R_2} = 0.$$

Under IEEE standard, $1/0 = \infty$, $\infty + \text{finite} = \infty$, and $\text{finite}/\infty = 0$. So for $R_1 = 0$ and $R_2 \neq 0$,

$$\frac{1}{1/R_1 + 1/R_2} = \frac{1}{\infty + 1/R_2} = \frac{1}{\infty} = 0.$$

Thus, if only one of R_1, R_2 is 0 and the other is non-zero, the formula $\frac{1}{1/R_1+1/R_2}$ still returns the correct answer under IEEE standard.

When $R_1 = R_2 = 0$, the formula $\frac{R_1 R_2}{R_1+R_2} = \frac{0}{0} = NaN$, it does not have a meaningful result. But the formula

$$\frac{1}{1/R_1 + 1/R_2} = \frac{1}{1/0 + 1/0} = \frac{1}{\infty + \infty} = \frac{1}{\infty} = 0,$$

which is not the correct answer.

3 Problem 3

(a) Proof: Recall the formula $\sin\alpha - \sin\beta = 2\cos\frac{\alpha+\beta}{2}\sin\frac{\alpha-\beta}{2}$, it is then clear that

$$\begin{aligned} f_{j,k} - f_{j+1,k} &= \sin(x_0 + (j-k)\pi/3) - \sin(x_0 + (j+1-k)\pi/3) \\ &= 2\cos(x_0 + (j-k+1/2)\pi/3)\sin(-\pi/6) \\ = 2(-1/2)\cos(x_0 + (j-k-1)\pi/3 + \pi) &= \sin(x_0 + [j-(k+1)]\pi/3) = f_{j,k+1}. \end{aligned}$$

(b) By the triangle inequality and $|\hat{f}_{j,k} - f_{j,k}| \leq \epsilon$ for all j, we have

$$\begin{aligned} |\hat{f}_{j,k+1} - f_{j,k+1}| &= |(\hat{f}_{j,k} - \hat{f}_{j+1,k}) - (f_{j,k} - f_{j+1,k})| \\ &= |(\hat{f}_{j,k} - f_{j,k}) - (\hat{f}_{j+1,k} - f_{j+1,k})| \\ &\leq |\hat{f}_{j,k} - f_{j,k}| + |\hat{f}_{j+1,k} - f_{j+1,k}| \leq \epsilon + \epsilon = 2\epsilon. \end{aligned}$$

(c) See the code submitted to ELMS.

4 Problem 4

(a) Arithmetic operations are done exactly, then the result is correctly rounded. Since $x * y = y * x$ exactly, the rounding result must be the same, as they are rounding numbers of the same result under the same standard. Similarly, $a + b = b + a$ is always true under IEEE standard. We conclude that the output of $(x * y) + (z - w)$ is precisely the same as that of $(z - w) + (y * x)$.
(b) The two arithmetic expressions may not result in the same floating point number. We cook up an example: (for simplicity, we set p = 4, i.e., the mantissa only has 3 bits, and base = 2)
consider

$$x = +1.01 * 2^0 = (1.01)_2, y = +1.00 * 2^{-3} = (0.001)_2, z = +1.10 * 2^{-2} = (0.011)_2,$$

thus, $x + y = (1.0101)_2 \approx +1.10 * 2^0$ (reject $+1.01 * 2^0$, as the last bit is 1). It turn out that

$$(x + y) + z = (1.10)_2 + (0.011)_2 = (1.111)_2 \approx +1.00 * 2^1.$$

And

$$x + (y + z) = x + (0.10)_2 = (1.11)_2 = +1.11 * 2^0.$$

Clearly, the arithmetic results are not the same.
(c) The two arithmetic expressions yield the same floating point number. This is because `OneHalf` $= 1.00000..0 * 2^{-1}$, and multiplication with `OneHalf` only results in -1 in the exponent of 2, i.e.,

$$(d_0.d_1...d_{p-1} * 2^e) * \texttt{OneHalf} = d_0.d_1...d_{p-1} * 2^{e-1}.$$

This says, multiplication with `OneHalf` does not change the relative order of the mantissa, so the rounding of the mantissa of $x * \texttt{OneHalf} + y * \texttt{OneHalf}$ will be exactly the same as the that of rounding of $x + y$. It turns out that

$$x * \texttt{OneHalf} + y * \texttt{OneHalf} = (x + y) * \texttt{OneHalf}.$$

(d) The two arithmetic expressions may not result in the same floating point number. We consider a simplified case, in which p = 4 (the mantissa only has 3 bits), and base = 2. In this setting,

$$1/3 = 0.3333 = 2^{-2} + 2^{-4} + 2^{-6} + ... \approx +1.01 * 2^{-2} = \texttt{OneThird}.$$

Now, consider $x = +1.01 * 2^2 = (101)_2, y = +1.00 * 2^0 = (1.00)_2$.

$$x * \texttt{OneThird} = (1.01)_2 * (1.01)_2 = (1.1001)_2 \approx +1.10 * 2^0,$$

$$y * \texttt{OneThird} = (1.00)_2 * (0.0101)_2 = (0.0101)_2 = +1.01 * 2^{-2},$$

$$x * \texttt{OneThird} + x * \texttt{OneThird} = (1.10)_2 * (0.0101)_2 = (1.1101)_2 \approx +1.11 * 2^0.$$

On the other hand,

$$(x + y) * \texttt{OneThird} = (110)_2 * (0.0101)_2 = (1.111)_2 \approx +1.00 * 2^2.$$

Clearly, the roundoff error appears and make the two results different.

References

[1] David Goldberg. *What Every Computer Scientist Should Know about Floating-Point Arithmetic.*