# Lecture 7

Wednesday, January 22, 2020

A computer can represent a number only with finitely many bits. Suppose that it can represent a number using only 8 bits:

$$a_1\ a_2\ a_3\ a_4\ a_5\ a_6\ a_7\ a_8$$

What number should this bit sequence represent? In other word, how should we interpret this bit sequence? Note that there are only 256 such bit sequences. Thus, there are only 256 numbers that can be represented by these sequences (without invoking approximation). One can wish for the following properties:

- Whole numbers (positive and negative) can be represented.
- Fractional numbers can be represented.
- Small numbers (close to 0) and large numbers can be represented.

---

See Lecture 6 for some examples of possible ways to interpret a bit sequence. The format $x = 6 \cdot \bar{x} \cdot 2^e$ is called a floating-point format The reason for this name is that one can shift the position of the dot (binary point) in $\bar{x}$ by increasing or decreasing the value of $e$.

Floating-point format has a much larger dynamic range than fixed-point format (see Lecture 6 for example)

$$\text{dynamic range} = \frac{\text{largest positive number}}{\text{smallest positive number}}$$

Fixed-point format can represent numbers that are evenly space on the real line.

Floating-point format, on the other hand, can represent very small numbers (i.e. close to 0). The numbers are dense near 0 and get more and more sporadic away from 0.

However, this drawback of floating-point numbers is not too bad in the following sense: for small numbers, the efficiency of approximation is usually measure by difference. For example, to see how good the approximation $a \approx b$ is, we look at the error $\Delta = |a-b|$. For large numbers, we usually look at the quotient $a/b$. For example, the approximation $1 \approx 2$ can be perceived as bad, but the approximation $101 \approx 102$ can be regarded good.

We will examine this issue more carefully when we discuss absolute error and relative error.

* Add two floating-point numbers:

Let $x = \bar{0} \cdot \bar{x} \cdot 2^e$ and $y = \bar{0} \cdot \bar{y} \cdot 2^f$, where $e, f$ belong to a certain range, and $\bar{x}$ and $\bar{y}$ have a certain number of digits after the binary point. We want to add $x$ and $y$ the way that makes sure the sum remains in the same format. This is not a regular sum $x+y$ as we would normally perform. Instead, we follow the procedure:

- Rewrite the number with smaller exponent in a way such that the exponent will match the exponent of the other number.
- Add two significands.
- Shift the binary point in the significand (by adjusting the exponent) so that the significand is in the corrected format
- Round the significand to agree with the format.

Ex:

Use the floating-point format in the worksheet to add

$$x = (1.101)_2 \times 2^2$$

and $y = (1.001)_2 \times 2^3$

We rewrite $x = (0.1101)_2 \times 2^3$ to match with the exponent of $y$.

Add significands:

$$\begin{array}{r} 0.1101 \\ +\ 1.001 \\ \hline 1.1111 \end{array}$$

Then $x+y = (1.1111)_2 \times 2^3$

Next, we round the significand to three digits after the dot.

$$(1.1111)_2 \approx (1.111)_2 + (0.001)_2 = (10.000)_2.$$

Then

$$x+y \approx (10.000)_2 \times 2^3$$

We shift the dot to the left.

$$x+y \approx (1.0000)_2 \times 2^3$$

Then round again:

$$x+y \approx (1.000)_2 \times 2^3$$

* Multiply two floating-point numbers.

To multiply $x = 0.\bar{x}.2^e$ by $y = z.\bar{y}.2^f$, we follow the procedure:

- Add two exponents: $e+f$
- Multiply the significands: $\bar{x}.\bar{y}$
- Shift the binary point in $\bar{x}.\bar{y}$ by adjusting the exponent.
- Round the significand

Ex:

$$x = (1.101)_2 \times 2^{-3}$$

$$y = (1.111)_2 \times 2^2$$

Add the exponents: $-3+2 = -1$

Multiply $(1.101)_2$ by $(1.111)_2$:

$$(1.101)_2 \times (1.111)_2 = (11.000011)_2$$

Then

$$xy = (11.000011)_2 \times 2^{-1}$$

Shift the dot:

$$x_y = (1.1000011)_2 \times 2^0$$

Round the significand to three digits after the dot:

$$x_y \approx (1.100)_2 \times 2^0$$