MA 3257 LECTURE 2

How are Numbers Stored in a Computer?

Machines are finite, while each real number has infinite length, i.e. $\frac{1}{3}$, $\sqrt{2}$ and π. Therefore, inaccuracies are guaranteed to arise. How then does a computer store a number so that the inaccuracies are somewhat mitigated?

Binary Machine Numbers. A 64-bit (binary digit) representation, used for a real number x, is called a floating-point representation of x. We will explore the "floating" feature of this representation in the following example. The floating-point representation of x is commonly written as,

$$fl\,(x) = (-1)^s\, 2^{c-1023}\,(1+f)\,,$$

where s is the **sign** of the number; c, the decimal representation of an 11-bit exponent, each bit 0 or 1, is called the **characteristic**; f, the decimal representation of a 52-bit array (each bit 0 or 1), is called the **mantissa**.

Since c is represented by 11 bits, that is,

$$c_1 c_2 \ldots c_{11}, \quad c_i = 0 \,\text{or}\, 1,$$

it represents decimal numbers 0 to $\sum_{i=0}^{10} 2^i = \sum_{i=1}^{11} 2^{i-1} = \frac{1\left(1-2^{11}\right)}{1-2} = 2^{11} - 1 = 2047$ (smallest being $c_i = 0$ for all $i = 1, 2, \ldots, 11$ and largest being $c_i = 1$ for all $i = 1, 2, \ldots, 11$). However, if we ignore the mantissa and simply use $fl\,(x) = (-1)^s\, 2^{c-1023}$, we can only represent powers of 2 (power ranging from -1023 to 1024). This is still too coarse of a representation, meaning that the gaps between numbers are too large. The **mantissa** therefore is necessary to represent numbers with even smaller magnitude so as to close the gap between numbers. Consider the 52-bit array,

$$f_1 f_2 \cdots f_{51} f_{52}, \quad f_i = 0 \,\text{or}\, 1.$$

The decimal representation of this array then is

$$f = \sum_{j=1}^{52} f_j \left(\frac{1}{2}\right)^j$$

where f_j is the j^{th} entry of the 52-bit (0 or 1).

Example. Consider the machine number

$$x = \underbrace{0}_{s,\,\text{sign}} \underbrace{10000000011}_{c,\,\text{characteristic}} \underbrace{1011100100010\overbrace{0\cdots0}^{40\ \text{zeros}}}_{f,\,\text{mantissa}}$$

Here, $s = 0$ (positive real number),

$$c = 2^{10} + 2^1 + 2^0 = 1027.$$

Thus, the exponential part is

$$2^{c-1023} = 2^4.$$

Now,

$$f = \left(\frac{1}{2} + \left(\frac{1}{2}\right)^3 + \left(\frac{1}{2}\right)^4 + \left(\frac{1}{2}\right)^5 + \left(\frac{1}{2}\right)^8 + \left(\frac{1}{2}\right)^{12}\right).$$

Altogether,

$$(-1)^s\, 2^{c-1023}\,(1+f) = 27.56640625$$

exactly.

Date: Jan 12th, 2023.

The number immediately smaller than x is

$$x_- = \underbrace{0}_{s,\text{ sign}}\ \underbrace{10000000011}_{c,\text{ characteristic}}\ \underbrace{1011100100001\overbrace{0\cdots1}^{40\text{ ones}}}_{f,\text{ mantissa}},$$

by going down from the $1\overbrace{0\cdots0}^{40\text{ zeros}}$ to $0\overbrace{1\cdots1}^{40\text{ ones}}$. The number immediately bigger than x is

$$x_+ = \underbrace{0}_{s,\text{ sign}}\ \underbrace{10000000011}_{c,\text{ characteristic}}\ \underbrace{101110010001\overbrace{0\cdots01}^{39\text{ zeros}}}_{f,\text{ mantissa}}.$$

This means, x represents half the numbers between x_- and x plus half the numbers between x and x_+, namely,

$$\frac{x+x_-}{2} \le x \le \frac{x+x_+}{2}$$

where

$$\frac{x+x_-}{2} = 27.566406249999998223643160599749535322189331054687 5,$$
$$\frac{x+x_+}{2} = 27.566406250000001776356839400250464677810668945312 5.$$

This means, the machine number x alone covers **all** real numbers bounded between the above two rationals. In fact, this gap is called the **machine epsilon** for 64-bit binary representation, and is approximately $2^{-52} \approx 2.22 \times 10^{-16}$ – the reason that it is 2^{-52} is by advancing the mantissa one unit to get the next largest/smallest number.

Decimal Machine Numbers. While storing the numbers in binary is efficient, it is hard to analyze them in their binary representations. We revert back to the more familiar decimal machine number represetation, in the following floating-point form

$$\pm 0.d_1d_2\ldots d_k \times 10^n, \quad 1 \le d_1 \le 9 \text{ and } 0 \le d_i \le 9, \quad i = 2, 3, \ldots, k.$$

We call numbers of this form k-digit **decimal machine numbers**.

Any positive real number within the numerical range of the machine can be normalized to the form

$$y = 0.d_1d_2\ldots d_kd_{k+1}d_{k+2}\ldots \times 10^n$$

though a termination on the index is certainly needed. There are two commons ways.

Chopping. Simply chop off the digits $d_{k+1}d_{k+2}$ and produce the floating-point form

$$fl\,(y) = 0.d_1d_2\ldots d_k \times 10^n.$$

Rounding. We do it in our daily life all the time. The procedure is to add $5 \times 10^{n-(k+1)}$ and then chops the result to obtain a number of the form

$$fl\,(y) = 0.\delta_1\delta_2\ldots\delta_k \times 10^n.$$

For rounding, when $d_{k+1} \ge 5$, we add 1 to the previous digit d_k to obtain $fl\,(y)$; that is, we **round up**. On the other hand, if $d_{k+1} < 5$, we chop off all but the first k digits; that is, we **round down**. When we round down, $\delta_i = d_i$ for each $i = 1, 2\ldots, k$.

Example. Determine the five-digit (a) chopping and (b) rounding values of the irrational number π.

Solution. The infinite decimal expansion of the form $\pi = 3.1415926535\ldots$. In normalized form, we have

$$\pi = 0.31415926535\ldots \times 10^1.$$

(1) Five-digit chopping is

$$fl\,(\pi) = 0.31415 \times 10^1 = 3.1415.$$

(2) Five-digit rounding involves the sixth digit here. Note the sixth digit is 5, so we round up. Using the exact procedure laid out in the above subsection, we add $5 \times 10^{n-(k+1)}$ with $n = 1$ and $k = 5$ (since we want five digits).

$$\begin{aligned}\pi + 5 \times 10^{1-(5+1)} &= 0.314159... \times 10^1 + 5 \times 10^{-6} \times 10^1 \\ &= (0.314159... + 0.000005) \times 10^1 \\ &= 0.314164... \times 10^1\end{aligned}$$

and thus with five-digit chopping now, we have

$$fl(\pi) = 0.31416 \times 10^1 = 3.1416.$$

The error that results from replacing a number with its floating-point form is called **round-off error** regardless of wehther the rounding or the chopping method is used.