Monday 01/09/12

IEEE Standard for Floating-Point Arithmetic = Standard 754

for binary number representation

(854 for general basis β)

Integers : e.g. int in C or integer in Fortran

usually 32 bits

$2^{31} - 1 = (01\ldots\ldots 111)_2$

$\vdots$

$2 = (00\ldots\ldots 010)_2$

$1 = (00\ldots\ldots 001)_2$

$0_{10} = (00\ldots\ldots 000)_2$

$-1 = (11\ldots\ldots 111)_2$

$-2 = (11\ldots\ldots 110)_2$

$\vdots$

$-2^{31} = (10\ldots\ldots 000)_2$

Fixed-point numbers:

$$\tilde{x} = \pm (d_{-3} d_{-2} d_{-1} d_0 . d_1 d_2 d_3 d_4)_{10}$$

→ Idea: Pick a 'symmetric' representation so that reciprocals $1/\tilde{x}$ are still in the number system!

Idea of floating-point numbers: Let point move right/left and multiply by 10^E to maintain the value

Also use concept of normalized number to determine how far to move left/right, namely such that exactly 1 digit before the point is non-zero, that is, a normalized number is

$$\tilde{x} = \pm (\underbrace{d_0 . d_1 d_2 \dots d_{p-1}}_{p \text{ digits}})_{10} 10^E, \; d_i \in \{0, 1, \dots, 9\}, \; d_0 \neq 0$$

Already notice: there is a trade-off between accuracy of the number system and its range

accuracy = number of digits p

range $\Leftrightarrow E_{min} \leq E \leq E_{max}$

Since the storage is finite in a computer/calculator making p larger forces range of E smaller and vice versa.

The above is BCD = binary coded decimal

scientific calculator $\pm (d_0 . d_1 \dots d_{11})_{10} \, 10^E$, $-99 \leq E \leq 99$

with only 10 digits in display area.

Notice: We are concerned with internal representation, not display!

How is BCD stored? Every decimal digit d_i in 4 binary bits.

Problem of inefficiency: 4 bits could store 16 choices ⇒ Waste of 4

History: Obvious fix = use hexadecimal base !

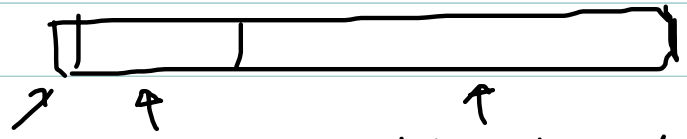
$\tilde{x} = \pm (d_0 . d_1 \ldots d_{p-1})_{16} \; 16^E$, $d_i \in \{0, \ldots, 9, a, b, \ldots, f\}$, $d_0 \neq 0$

→ IBM mainframes in 1960 to 80s

IEEE-Standard – voluntary standard using binary base for maximum flexibility of all choices and some additional efficiencies.

We focus on double-precision, since we should use it by default and Matlab uses it by default.

Starting point: 64 bits for double !



Sign bit | stores E | bits $b_1, b_2, \ldots b_{p-1}$

11 bits | 52 bits

normalized binary number

$$\tilde{x} = (-1)^S (b_0 . b_1 b_2 \ldots b_{p-1})_2 \; 2^E$$

$b_i \in \{0, 1\}$, $b_0 \neq 0$

$\Rightarrow b_0 = 1$ for any normalized binary number

$\Rightarrow$ "implied 1" or "hidden bit" meaning that b_0 is never stored !

$\Rightarrow p = 53$

fractional part $f = (b_1 b_2 \ldots b_{p-1})$

See p. 9 of handout Fig. 2

and Table 1 for "double" → confirms p!

Terminology:

$$(1.\overbrace{b_1 b_2 \ldots b_{p-1}}^{\text{fractional part } f})_2 = (1.f)_2$$

mantissa $m = (1.f)_2$

Want to store "special numbers"

and "errors" as part of the number:

errors: $\frac{0}{0}$, $\frac{\infty}{\infty}$

infinities are best example of reasonable numbers that extend the system,

could come from $\frac{1}{0}$, $n!$ for $n \geq 172$

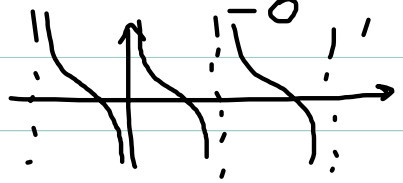
$\log(0) = -\infty$

$\Rightarrow$ Want $+\infty$ and $-\infty$!

Caution: There are bared on +0 and our thinking about functions like log!

How about $\frac{-1}{0}$ or $\frac{1}{-0}$ log(−0)

cot(0) = ∞



but could equally be −∞ mathematical

⇒ Be careful not to rely on these without testing.

Realize what the standard actually says:

Results for −0 must be same as of +0!

Also, when we use software, we do interact with the hardware directly and this can make a difference.

⇒ Upshot for IEEE:

Want special values inf, −inf, NaN

NaN = not-a-number = error

Also need x = 0! → Want this to be all 0's!

characteristic e that indicates a
corner distinction:

if $e = 0$, then $\tilde{x} = (-1)^s (0)_2$

if $e = (\underbrace{11\ldots11}_{\text{11 bits}})_2$, then

// $2^{11} - 1 = 2047$

$$\tilde{x} = \begin{cases} (-1)^s \inf & \text{if } f = 0 \\ \text{NaN} & \text{otherwise} \end{cases}$$

if $0 < e < 2047$, then $\tilde{x}$ is normalized

$$\tilde{x} = (-1)^s (1.f)_2 \, 2^E = (-1)^s (1.b_1 b_2 \ldots b_{p-1}) 2^E$$

e and E are connected by a shift = bias

$$E = e - \text{bias}$$

What are E_{min}, E_{max}, bias?

Have $1 \le e \le 2046 \Rightarrow$ bias ≈ 1023

to make range of E symmetric about 0

Recall idea: Want $\frac{1}{\tilde{x}}$ to be in the number system

Table 1 $\Rightarrow E_{min} = -1022$, $E_{max} = 1023$

To understand this choice, realize that we have the case of $e=0$ and $f \neq 0$ not used efficiently

if $e=0$ then

$\tilde{x} = (-1)^s (0)_2$ if $f=0$

$\tilde{x} = (-1)^s (0.f)_2 2^{E_{min}}$ de-normalized

The largest de-normalized number is the "next" smaller from the smallest normalized one $\tilde{x} = (1.0\ldots0)_2 2^{E_{min}}$

Understand the E_{min}, E_{max} choice:

$$\frac{1}{2^{E_{min}}} = \frac{1}{2^{-1022}} = 2^{1022} \leq 2^{1023} = 2^{E_{max}}$$

$$\frac{1}{2^{E_{max}}} = \frac{1}{2^{1023}} = 2^{-1023} = 0.5 \cdot 2^{-1022} = (0.1)_2 2^{E_{min}}$$

Some thinking gives that bias $= 1023$

Since we want $e = \frac{2046}{2}$ to agree with $E=0$.

Notice: these nubs are not equally spaced but rather all errors are relative

Ex.: | s | 3 bits | 4 bits |