Prepared by Julian García, based on material by
Arun Konagurthu

Workshop 2
Floating point number systems

FIT 3139
Computational Modelling and Simulation

$$(321.40625)_{10} \longrightarrow (\ \texttt{??????}\ )_2$$

$$(321.40625)_{10} \longrightarrow (\ \texttt{??????}\ )_2$$

321 **// 2** = 160 ⟶ 1 $2^0 = 1$

160 **// 2** = 80 ⟶ 0

80 **// 2** = 40 ⟶ 0

40 **// 2** = 20 ⟶ 0

20 **// 2** = 10 ⟶ 0

10 **// 2** = 5 ⟶ 0

5 **// 2** = 2 ⟶ 1 $2^6 = 64$

2 **// 2** = 1 ⟶ 0

1 **// 2** = 0 ⟶ 1 $2^8 = 256$

$$(321)_{10} = (101000001)_2$$

0.40625 x 2 = **0**.8125

0.8125 x 2 = **1**.625

0.625 x 2 = **1**.25

0.25 x 2 = **0**.5

0.5 x 2 = **1**.0

$$(0.40625)_{10} = (0.01101)_2$$

$$101000001.01101$$

$$1.0100000101101 \times 2^8$$

In your own study time…

$$(89.158)_{10} \longrightarrow (\ \texttt{??????}\ )_2$$

Expand only
up to 6 terms after
radix point

Computer Arithmetic

Everything in a computer is finite, and yet real numbers are uncountably infinite… **approximation is inevitable**

In a digital computer, real numbers are represented by a ***floating point*** number system

Scientific Notation: Express big (or small) numbers, compactly.

$$2347 = +2.347 \times 10^{3}$$

$$-0.0007396 = \underbrace{-}_{\text{sign}}\ \underbrace{7.396}_{\substack{\text{mantissa}\\(\text{significant})}} \times \underbrace{10}_{\text{base}}\ \underbrace{^{-}4}_{\text{exponent}}$$

$$2347 = +\ 2.347 \times 10^{3}$$

sign — mantissa (significant) — base — exponent

$$2.347$$

$$\left(\frac{2}{10^0} + \frac{3}{10^1} + \frac{4}{10^2} + \frac{7}{10^3}\right) \times 10^3$$

base or radix = 10

precision = how many terms in the sum

exponent = number in a range ***[L, U]***

(Note that numerators in the mantissa are always between 0 and base-1)

Floating-point numbers

Any floating point number *x* has the form:

$$x = \pm \left(\frac{d_0}{b^0} + \frac{d_1}{b^1} + \frac{d_2}{b^2} + \cdots + \frac{d_{p-1}}{b^{p-1}} \right) b^E$$

where any d_i is an integer such that:

$$0 \leq d_i \leq b - 1, \qquad i = 0, \cdots, p - 1$$

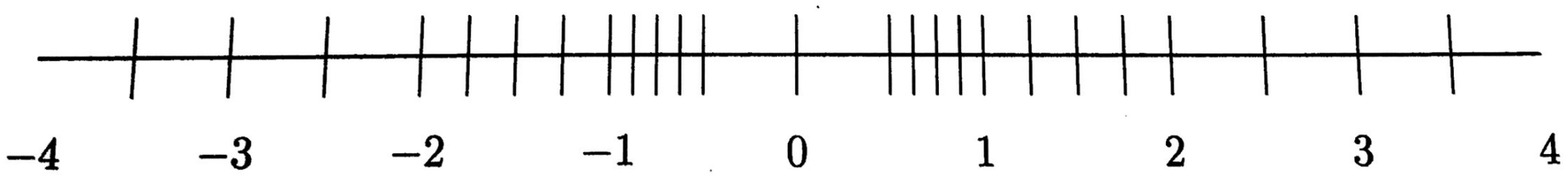
In a **normalised system** d_0 cannot be *0*.

With b=2, d_0 does not need to be stored.

Because of this, and other advantages **most systems are normalised.**

A floating point number system is characterised by four integers:

- The base, **b**
- The precision, **p**
- The exponent range, **[L, U]**



- The base, **b** = 2
- The precision, **p** = 3
- The exponent range E in **[-1, 1]**

Heath. Chapter 1.3

IEEE-SP

1 bit	8 bits					23 bits							
Sign (S)	Exponent (E)					Mantissa or Fraction (F)							
31	30	29	···	24	23	22	21	20	19	···	2	1	0

- The base, **b = 2**
- The precision, **p = 24**
- **L =-126 , U = 127**

$$\epsilon_{mach} = \frac{1}{2} b^{1-p} \approx 10^{-7}$$

7 digits of precision

IEEE-DP

1 bit	11 bits					52 bits							
Sign (S)	Exponent (E)					Mantissa or Fraction (F)							
63	62	61	···	53	52	51	50	49	48	···	2	1	0

- The base, **b = 2**
- The precision, **p = 53**
- **L =-1022 , U = 1023**

$$\epsilon_{mach} = \frac{1}{2} b^{1-p} \approx 10^{-16}$$

16 digits of precision

special values

+0: S=0, E= 0, F=0

-0: S=1, E= 0, F=0

+∞: S=0, E= all 1, F=0

-∞: S=1, E= all 1, F=0

NaN: S=0 or 1, E= all 1, F= not 0

See a nice step by step example:

https://www.youtube.com/watch?v=H79PNQ4Z9HE

Rounding

Not all real numbers are exactly representable in a given floating-point system. They must be approximated by (or *rounded to*) some *nearby* floating point number.

Rounding by chopping: The base-b expansion of *x* is truncated after the (p-1)st digit.

Rounding to nearest: The real number is rounded to the nearest floating point number.

Rounding to nearest is the default rounding rule in IEEE standards. It is more cumbersome and less efficient to implement, but is more accurate.

Machine epsilon

(a.k.a unit roundoff, machine precision)

Provides a sense for the "**granularity**" of the floating point number system.

The smallest value ϵ_{mach} such that …

$$1 + \epsilon_{mach} > 1$$

The machine epsilon gives **an upper bound on the relative error due to rounding** in floating point arithmetic.

$$\left| \frac{fl(x) - x}{x} \right| \leq \epsilon_{mach}$$

What Every Computer Scientist Should Know About Floating-Point Arithmetic

DAVID GOLDBERG

Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, California 94304

Floating-point arithmetic is considered an esoteric subject by many people. This is rather surprising, because floating-point is ubiquitous in computer systems: Almost every language has a floating-point datatype; computers from PCs to supercomputers have floating-point accelerators; most compilers will be called upon to compile floating-point algorithms from time to time; and virtually every operating system must respond to floating-point exceptions such as overflow This paper presents a tutorial on the aspects of floating-point that have a direct impact on designers of computer systems. It begins with background on floating-point representation and rounding error, continues with a discussion of the IEEE floating-point standard, and concludes with examples of how computer system builders can better support floating point.

Categories and Subject Descriptors: (Primary) C.0 [**Computer Systems Organization**]: General—*instruction set design*; D.3.4 [**Programming Languages**]: Processors—*compilers*, *optimization*; G.1.0 [**Numerical Analysis**]: General—*computer arithmetic*, *error analysis*, *numerical algorithms* (Secondary) D.2.1 [Software Engineering]: Requirements/Specifications—*languages*; D.3.1 [**Programming Languages**]: Formal Definitions and Theory—*semantics* D.4.1 [**Operating Systems**]: Process Management—*synchronization*

General Terms: Algorithms, Design, Languages

Additional Key Words and Phrases: denormalized number, exception, floating-point, floating-point standard, gradual underflow, guard digit, NaN, overflow, relative error, rounding error, rounding mode, ulp, underflow

Please read….