

Mathematics 3670: Computer Systems Floating Point Representation

Dr. William Slough

Mathematics and Computer Science Department
Eastern Illinois University

Fall 2009

Mathematical number systems

- \mathbb{Z} — the set of whole numbers: negative, zero, positive
- \mathbb{Q} — the set of all numbers of the form $\frac{p}{q}$, $p, q, \in \mathbb{Z}, q \neq 0$
- \mathbb{R} — the set of all points on an infinitely long line
 - There is no largest real number
 - There is no smallest real number
 - There are an infinite number of real numbers
 - There are an infinite number of real numbers on $[0, 1]$
 - There is no smallest positive number
 - Enjoys many familiar properties:
 associativity, commutativity, ...

How do these systems compare with the data types available on computer systems?

Integer types

- Unsigned
- Signed
- Both are *subsets* of \mathbb{Z}
- Typically come in several *widths*, e.g., 16-bit vs. 32-bit
- Arithmetic is *exact*
- *Arithmetic overflow* is possible (and often “silent”)

Floating point types

- Used extensively in scientific computing
- Source of subtle problems and catastrophes (e.g., Ariane 5)
- **Common misconception:** floating point = \mathbb{R}
- **Reality:** set of floating point values is a finite subset of \mathbb{Q}
- Floating point arithmetic is fraught with traps:
 - Possible *overflow*
 - Possible *underflow*
 - Properties of arithmetic in \mathbb{R} may not apply, e.g.

$$(a + b) + c \stackrel{?}{=} a + (b + c)$$

- In the “old days,” computer makers used different floating point systems, leading to low portability
- Today, IEEE standards are used for increased portability

Floating point representation: at a glance

Item	Single Precision	Double Precision
bits in sign	1	1
bits in exponent	8	11
bits in fraction	23	52
bits, total	32	64
exponent	excess 127	excess 1023
exponent range	-126 to 127	-1022 to 1023
smallest normalized value	2^{-126}	2^{-1022}
largest normalized value	$\approx 2^{128}$	$\approx 2^{1024}$
decimal range	$\approx 10^{-38}$ to 10^{38}	$\approx 10^{-308}$ to 10^{308}
smallest denormalized value	$\approx 10^{-45}$	$\approx 10^{-324}$

Source: Tanenbaum, *Structured Computer Organization*, 4th ed., Prentice Hall

IEEE standard: single-precision floating point

Types:

- **Normalized**
- ± 0
- Denormalized
- \pm infinity
- NaN, or “not a number”

Normalized values

$$-3.156 \times 10^{17}$$

sign $s = 1$
 significand $M = 3.156$
 exponent $E = 17$

$$(-1)^s \times M \times 10^E$$

How can we adjust this scheme for binary values?

Subdivision of a floating point value



- The width of each field is specified by the IEEE standard
- For single precision, there are $1 + 8 + 23 = 32$ bits
- For double precision, there are $1 + 11 + 52 = 64$ bits

How is each field specified?

The sign field

- There is the easy part: use a **sign bit** in the usual way
- $s = 0$ means **non-negative**
- $s = 1$ means **negative**

The significand requires normalization and a trick

Consider the binary value $(101011.10)_2$. How do we write this?

$$\begin{aligned}
 &1.0101110 \times 2^5 \\
 &10.101110 \times 2^4 \\
 &101.01110 \times 2^3 \\
 &1010.1110 \times 2^2 \\
 &\dots \text{among many other choices} \dots
 \end{aligned}$$

Like scientific notation, we place the radix point so there is exactly one digit to its left, adjusting the exponent as needed:

$$\text{significand} = M = (1.f_1f_2f_3 \dots f_{23})_2$$

There is always a leading 1 bit provided the value is not zero.

Sneaky trick: Don't store the leading 1 — the "hidden" bit!

Bias system used for exponents

- We want the exponent field to represent negative, zero, and positive exponents
- For an 8-bit exponent, we use the **excess-127** scheme
- Every exponent value has a built-in **bias** of 127

bit pattern	unsigned value	excess-127
1111 1111	—	not used for normalized values
1111 1110	254	$254 - 127 = 127$
1111 1101	253	$253 - 127 = 126$
...
1000 0000	128	$128 - 127 = 1$
0111 1111	127	$127 - 127 = 0$
...
0000 0010	2	$2 - 127 = -125$
0000 0001	1	$1 - 127 = -126$
0000 0000	—	not used for normalized values

Floating point representation: examples

Example 1: How can we represent $-6\frac{5}{8}$ as a 32-bit **float**?

Example 2: What **float** value is represented by **3D800000** ?

Floating point types

Normalized

±	$0 < E < \text{Max}$	any bit pattern
---	----------------------	-----------------

Denormalized

±	0	any nonzero bit pattern
---	---	-------------------------

Zero

±	0	0
---	---	---

Infinity

±	111...1	0
---	---------	---

NaN (not a number)

±	111...1	any nonzero bit pattern
---	---------	-------------------------