

# Data representation: Fixed Point and Floating Point

Yipeng Huang

Rutgers University

October 13, 2026

# Data representation: Fixed Point and Floating Point

## Announcements

Canvas timed quiz 4 and programming assignment 2

## Integers and basic arithmetic

Representing negative and signed integers

## Fractions and fixed point representation

`monteCarloPi.c` Using floating point and random numbers to estimate  $\pi$

## Floats: Overview

## Floats: Normalized numbers

Normalized: exp field

Normalized: frac field

Normalized: example

# Data representation: Fixed Point and Floating Point

## Announcements

Canvas timed quiz 4 and programming assignment 2

## Integers and basic arithmetic

Representing negative and signed integers

## Fractions and fixed point representation

`monteCarloPi.c` Using floating point and random numbers to estimate  $\pi$

## Floats: Overview

## Floats: Normalized numbers

Normalized: exp field

Normalized: frac field

Normalized: example

# Canvas timed quiz 4 and programming assignment 2

## Quiz 4

1. Spanning Friday 10/2 - Friday 10/16.
2. 60 minutes.
3. Three tries.
4. Memory bugs and bits, bytes, and integers.

## Programming assignment 2

1. Due Tuesday 10/13.
2. Data structures mastery.

# Data representation: Fixed Point and Floating Point

## Announcements

Canvas timed quiz 4 and programming assignment 2

## Integers and basic arithmetic

Representing negative and signed integers

## Fractions and fixed point representation

`monteCarloPi.c` Using floating point and random numbers to estimate PI

## Floats: Overview

## Floats: Normalized numbers

Normalized: exp field

Normalized: frac field

Normalized: example

# Data representation: Fixed Point and Floating Point

## Announcements

Canvas timed quiz 4 and programming assignment 2

## Integers and basic arithmetic

Representing negative and signed integers

## Fractions and fixed point representation

`monteCarloPi.c` Using floating point and random numbers to estimate  $\pi$

## Floats: Overview

## Floats: Normalized numbers

Normalized: exp field

Normalized: frac field

Normalized: example

# Representing negative and signed integers

## Ways to represent negative numbers

1. Sign magnitude

redundant zero, addition was not sound

2. 1s' complement

1111...11 redundant zero, addition sound

3. 2's complement

# Representing Negative Integers: Sign-Magnitude

- ▶ **Concept:** The Most Significant Bit (MSB) acts as a sign flag (0 = positive, 1 = negative). The remaining bits encode the absolute magnitude.
- ▶ **Range (8-bit):**  $-127$  to  $+127$ .
- ▶ **Fatal Flaws:**
  - ▶ **Dual Zeros:** Creates two representations for zero (0000\_0000 for  $+0$ , 1000\_0000 for  $-0$ ).
  - ▶ **Addition Fails:** Standard binary addition circuits produce incorrect results (e.g., adding  $+1$  and  $-1$  yields  $-2$ ).

# Representing Negative Integers: Ones' Complement

- ▶ **Concept:** Represent a negative number by flipping all the bits of its positive counterpart.
- ▶ **Addition Fix:** Binary addition mostly works, but requires a **carry-around rule** (if a carry overflows the MSB, it must be added back to the Least Significant Bit).
- ▶ **The Math:** The leading bit effectively carries a weight of  $-127$  (for 8-bit), meaning the system complements against the value represented by all ones.
- ▶ **Lingering Issue:** Still suffers from the dual zero problem (0000\_0000 and 1111\_1111).

# Representing Negative Integers: Two's Complement

power-of-two    1000...000

- ▶ **The Standard:** The universally adopted encoding for modern hardware architectures.
- ▶ **The Rule:** Flip all bits and add 1.
- ▶ **The Math:** The leading bit functions as a large negative weight (exactly  $-128$  for an 8-bit integer), while all other bits contribute standard positive powers of two.
- ▶ **Advantages:**
  - ▶ Eliminates the dual representation of zero.
  - ▶ Binary addition works natively without any special carry-around rules.

# signed char

most neg number

$$1000-0000 = -128$$

Zero

$$0000-0000$$

most pos number

$$0111-1111 = +127$$

neg one

$$1111-1111$$



pos one

$$0000-0001$$

# Representing negative and signed integers

## 2's complement

| signed char | weight in decimal |
|-------------|-------------------|
| 00000001    | 1                 |
| 00000010    | 2                 |
| 00000100    | 4                 |
| 00001000    | 8                 |
| 00010000    | 16                |
| 00100000    | 32                |
| 01000000    | 64                |
| 10000000    | -128              |

Table: Weight of each bit in a signed char type

- ▶ what is the most positive value you can represent? 127
- ▶ what is the most negative value you can represent? -128
- ▶ how to represent -1? 11111111
- ▶ how to represent -2? 11111110

# Representing negative and signed integers

## 2's complement

| signed char | weight in decimal |
|-------------|-------------------|
| 00000001    | 1                 |
| 00000010    | 2                 |
| 00000100    | 4                 |
| 00001000    | 8                 |
| 00010000    | 16                |
| 00100000    | 32                |
| 01000000    | 64                |
| 10000000    | -128              |

Table: Weight of each bit in a signed char type

- ▶ MSB: 1 for negative
- ▶ To make a number negative: flip all bits and add 1.
- ▶ Addition in 2's complement is sound

# Importance of paying attention to limits of encoding

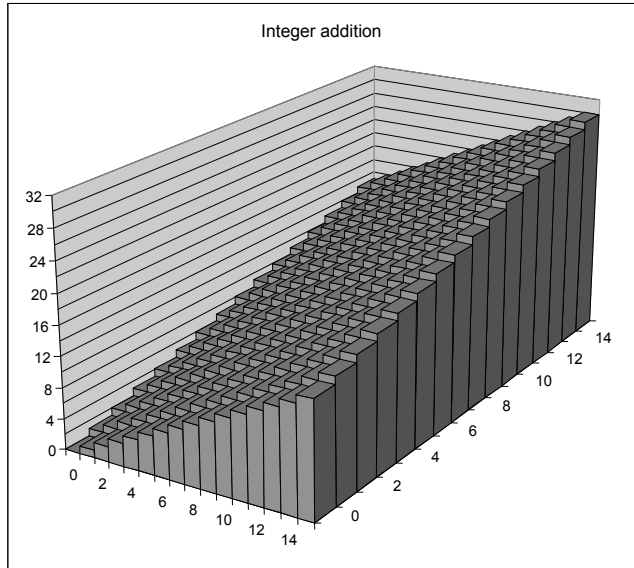


Figure: Image credit: CS:APP

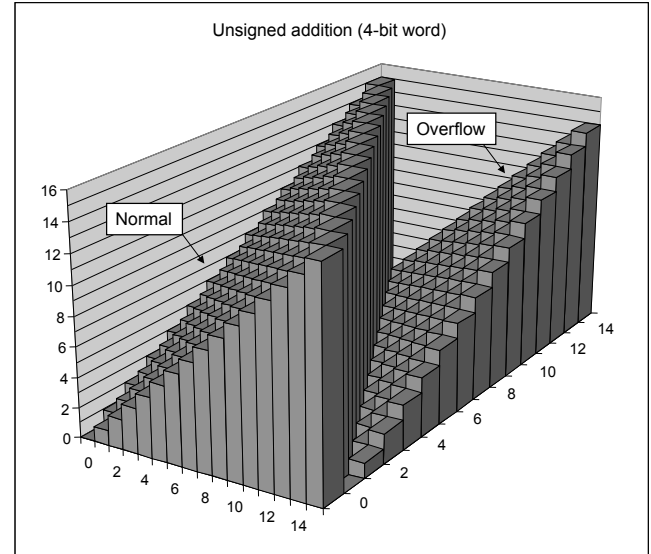


Figure: Image credit: CS:APP

# Importance of paying attention to limits of encoding

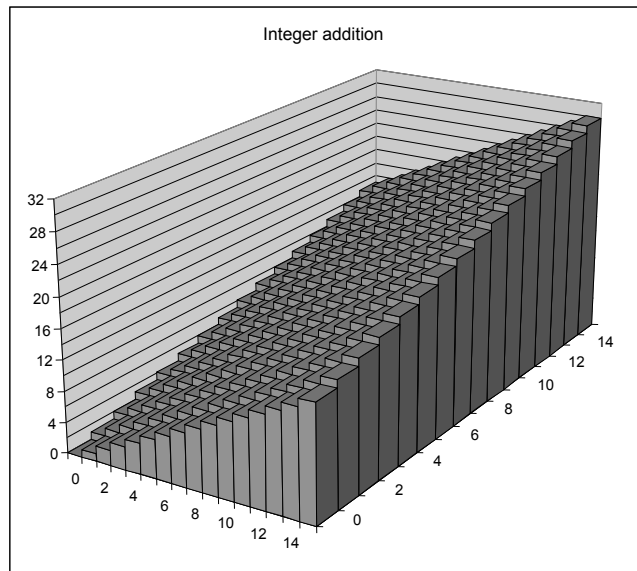


Figure: Image credit: CS:APP

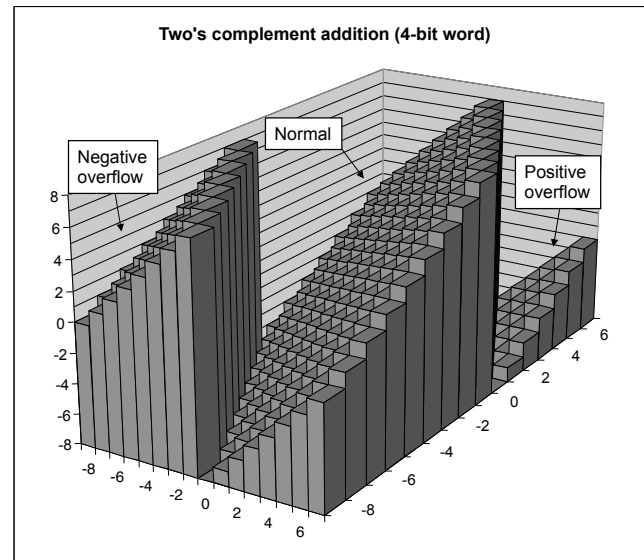
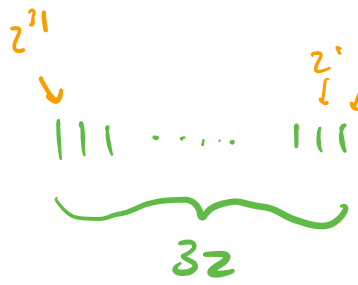


Figure: Image credit: CS:APP

<https://www.theatlantic.com/technology/archive/2014/12/how-gangnam-style-broke-youtube/383389/>

32 bit unsigned int.

$$2^{10} = 1024 \approx 1000 = 10^3$$


$$\begin{aligned} &= 2^{32} - 1 = 2^{3(10)+2} - 1 \\ &\approx (2^{10})^3 \cdot 2^2 \\ &\approx (10^3)^3 \cdot 2^2 \\ &= 2^2 \cdot 10^9 \\ &= 4 \cdot 10^9 \end{aligned}$$

32-bit signed int.

$$\approx 2 \cdot 10^9$$

# Data representation: Fixed Point and Floating Point

## Announcements

Canvas timed quiz 4 and programming assignment 2

## Integers and basic arithmetic

Representing negative and signed integers

## Fractions and fixed point representation

(division?)

`monteCarloPi.c` Using floating point and random numbers to estimate PI

## Floats: Overview

## Floats: Normalized numbers

Normalized: exp field

Normalized: frac field

Normalized: example

.93



$$9 \cdot \frac{1}{10}$$

$$3 \cdot \frac{1}{10^2}$$

# Unsigned fixed-point binary for fractions

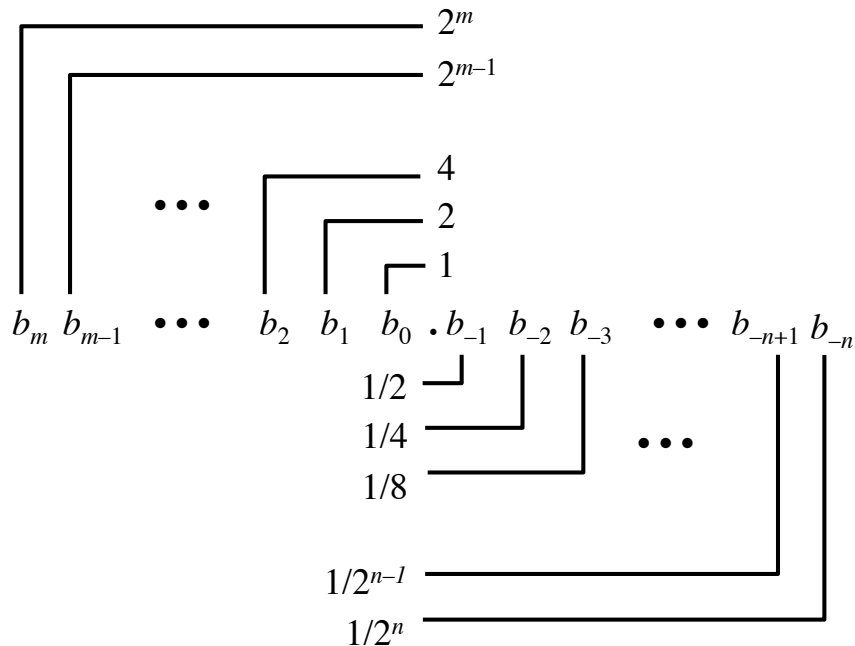


Figure: Fractional binary. Image credit CS:APP

# Unsigned fixed-point binary for fractions

| unsigned fixed-point char example | weight in decimal |
|-----------------------------------|-------------------|
| 1000.0000                         | 8                 |
| 0100.0000                         | 4                 |
| 0010.0000                         | 2                 |
| 0001.0000                         | 1                 |
| 0000.1000                         | 0.5               |
| 0000.0100                         | 0.25              |
| 0000.0010                         | 0.125             |
| 0000.0001                         | 0.0625            |

**Table:** Weight of each bit in an example fixed-point binary number

- ▶  $.625 = .5 + .125 = 0000.1010_2$
- ▶  $1001.1000_2 = 9 + .5 = 9.5$

# Signed fixed-point binary for fractions

| signed fixed-point char example | weight in decimal |
|---------------------------------|-------------------|
| 1000.0000                       | -8                |
| 0100.0000                       | 4                 |
| 0010.0000                       | 2                 |
| 0001.0000                       | 1                 |
| 0000.1000                       | 0.5               |
| 0000.0100                       | 0.25              |
| 0000.0010                       | 0.125             |
| 0000.0001                       | 0.0625            |

Table: Weight of each bit in an example fixed-point binary number

- ▶  $-.625 = -8 + 4 + 2 + 1 + 0 + .25 + .125 = 1111.0110_2$
- ▶  $1001.1000_2 = -8 + 1 + .5 = -6.5$

# Limitations of fixed-point

- ▶ Can only represent numbers of the form  $x/2^k$
- ▶ Cannot represent numbers with very large magnitude (great range) or very small magnitude (great precision)

# Bit shifting

## $\ll$ $N$ Left shift by $N$ bits

- ▶ multiplies by  $2^N$
- ▶  $2 \ll 3 = 0000\_0010_2 \ll 3 = 0001\_0000_2 = 16 = 2 * 2^3$
- ▶  $-2 \ll 3 = 1111\_1110_2 \ll 3 = 1111\_0000_2 = -16 = -2 * 2^3$

## $\gg$ $N$ Right shift by $N$ bits

~~▶ divides by  $2^N$~~

- ▶  $16 \gg 3 = 0001\_0000_2 \gg 3 = 0000\_0010_2 = 2 = 16/2^3$
- ▶  $-16 \gg 3 = 1111\_0000_2 \gg 3 = 1111\_1110_2 = -2 = -16/2^3$

# Data representation: Fixed Point and Floating Point

## Announcements

Canvas timed quiz 4 and programming assignment 2

## Integers and basic arithmetic

Representing negative and signed integers

## Fractions and fixed point representation

`monteCarloPi.c` Using floating point and random numbers to estimate PI

## Floats: Overview

## Floats: Normalized numbers

Normalized: exp field

Normalized: frac field

Normalized: example

# monteCarloPi.c Using floating point and random numbers to estimate PI

# Data representation: Fixed Point and Floating Point

## Announcements

Canvas timed quiz 4 and programming assignment 2

## Integers and basic arithmetic

Representing negative and signed integers

## Fractions and fixed point representation

`monteCarloPi.c` Using floating point and random numbers to estimate  $\pi$

## Floats: Overview

## Floats: Normalized numbers

Normalized: exp field

Normalized: frac field

Normalized: example

# Floating point numbers

$$\sim 60.2 \times 10^{22}$$

## Avogadro's number

$$+6.02214 \times 10^{23} \text{ mol}^{-1}$$

$$6.02 \times 10^{23}$$

## Scientific notation

- ▶ sign
- ▶ mantissa or significand
- ▶ exponent

$$0.602 \times 10^{24}$$

# Floating point numbers

## Before 1985

1. Many floating point systems.
2. Specialized machines such as Cray supercomputers.
3. Some machines with specialized floating point have had to be kept alive to support legacy software.

## After 1985

1. IEEE Standard 754.
2. A floating point standard designed for good numerical properties.
3. Found in almost every computer today, except for tiniest microcontrollers.

## Recent

1. Need for both lower precision and higher range floating point numbers.
2. Machine learning / neural networks. Low-precision tensor network processors.

# Floats and doubles

Single precision



Double precision

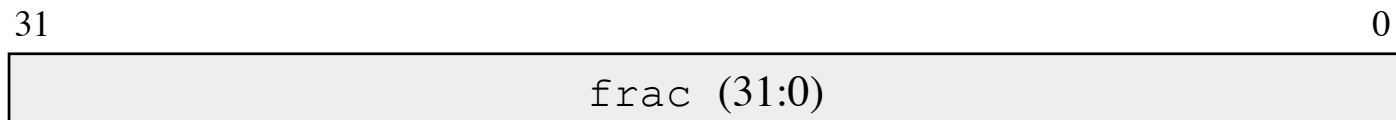


Figure: The two standard formats for floating point data types. Image credit CS:APP

# Floats and doubles

| property                    | half* | float | double |
|-----------------------------|-------|-------|--------|
| total bits                  | 16    | 32    | 64     |
| s bit                       | 1     | 1     | 1      |
| exp bits                    | 5     | 8     | 11     |
| frac bits                   | 10    | 23    | 52     |
| C printf() format specifier | None  | "%f"  | "%lf"  |

Table: Properties of floats and doubles

# The IEEE 754 number line

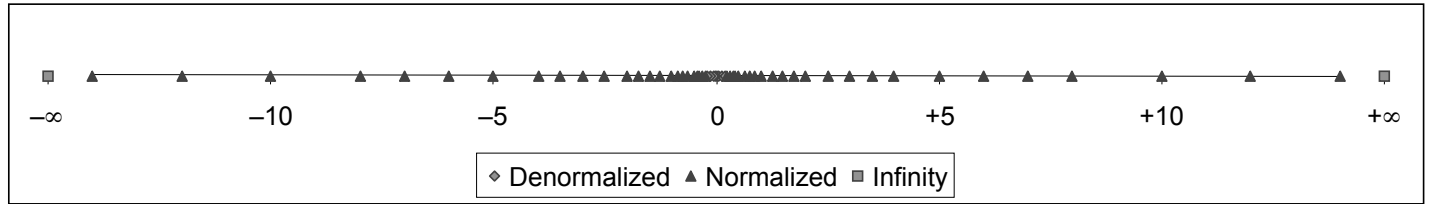


Figure: Full picture of number line for floating point values. Image credit CS:APP

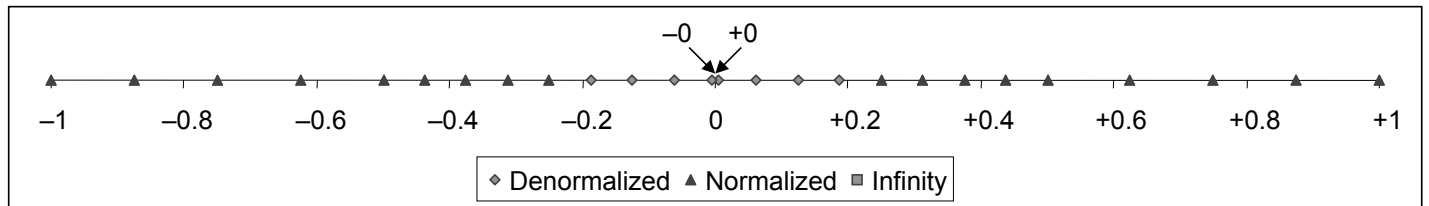


Figure: Zoomed in number line for floating point values. Image credit CS:APP

## Different cases for floating point numbers

Value of the floating point number =  $(-1)^s \times M \times 2^E$

- ▶  $E$  is encoded the exp field
- ▶  $M$  is encoded the frac field

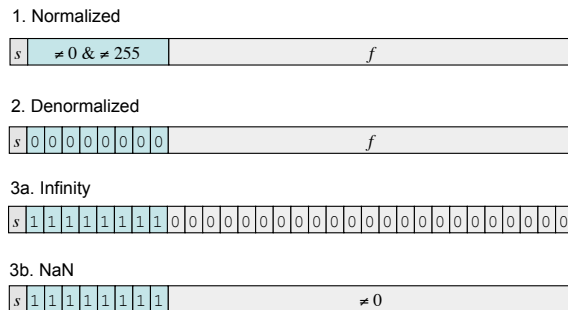


Figure: Different cases within a floating point format. Image credit CS:APP

## Normalized and denormalized numbers

Two different cases we need to consider for the encoding of  $E, M$

# Data representation: Fixed Point and Floating Point

## Announcements

Canvas timed quiz 4 and programming assignment 2

## Integers and basic arithmetic

Representing negative and signed integers

## Fractions and fixed point representation

`monteCarloPi.c` Using floating point and random numbers to estimate  $\pi$

## Floats: Overview

## Floats: Normalized numbers

Normalized: exp field

Normalized: frac field

Normalized: example

# Data representation: Fixed Point and Floating Point

## Announcements

Canvas timed quiz 4 and programming assignment 2

## Integers and basic arithmetic

Representing negative and signed integers

## Fractions and fixed point representation

`monteCarloPi.c` Using floating point and random numbers to estimate  $\pi$

## Floats: Overview

### Floats: Normalized numbers

Normalized: exp field

Normalized: frac field

Normalized: example

# Normalized: exp field

For normalized numbers,

$$0 < \text{exp} < 2^k - 1$$

- exp is a  $k$ -bit unsigned integer

## Bias

- need a bias to represent negative exponents
- $\text{bias} = 2^{k-1} - 1$
- bias is the  $k$ -bit unsigned integer: 011..111

For normalized numbers,

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

In other words,  $\text{exp} = E + \text{bias}$

|                                 | property | float | double |
|---------------------------------|----------|-------|--------|
|                                 | $k$      | 8     | 11     |
|                                 | bias     | 127   | 1023   |
| smallest E (greatest precision) |          | -126  | -1022  |
| largest E (greatest range)      |          | 127   | 1023   |

Table: Summary of normalized exp field

# Data representation: Fixed Point and Floating Point

## Announcements

Canvas timed quiz 4 and programming assignment 2

## Integers and basic arithmetic

Representing negative and signed integers

## Fractions and fixed point representation

`monteCarloPi.c` Using floating point and random numbers to estimate  $\pi$

## Floats: Overview

### Floats: Normalized numbers

Normalized: exp field

Normalized: frac field

Normalized: example

## Normalized: frac field

$$M = 1.\text{frac}$$

# Data representation: Fixed Point and Floating Point

## Announcements

Canvas timed quiz 4 and programming assignment 2

## Integers and basic arithmetic

Representing negative and signed integers

## Fractions and fixed point representation

`monteCarloPi.c` Using floating point and random numbers to estimate  $\pi$

## Floats: Overview

### Floats: Normalized numbers

Normalized: exp field

Normalized: frac field

Normalized: example

## Normalized: example

- ▶ 12.375 to single-precision floating point
- ▶ sign is positive so  $s=0$
- ▶ binary is  $1100.011_2$
- ▶ in other words it is  $1.100011_2 \times 2^3$
- ▶  $\text{exp} = E + \text{bias} = 3 + 127 = 130 = 1000\_0010_2$
- ▶  $M = 1.100011_2 = 1.\text{frac}$
- ▶  $\text{frac} = 100011$