Data representation: Integers, Fixed Point, Floating Point

Yipeng Huang

Rutgers University

October 9, 2025

Table of contents

Representing negative and signed integers

Ways to represent negative numbers

	Comparison	Addition
1. Sign magnitude	redundant zero	unconventional
2. 1s' complement	redundant zero	sound, remember to add back carry
3. 2's complement	unique zero	sound

Representing negative and signed integers

most neg number

0b1111_1111

= -127_{10}

Zero

0b0000_0000

0b1000_0000

largest pos number

0b0111_1111

= $+127_{10}$

Sign magnitude

Flip leading bit.

neg one

0b1000_0001

= -1_{10}

One

0b0000_0001

= 1_{10}

$$-I_0 + I_0 = 0$$

$$0b1000\_0001$$

$$+)\ 0b0000\_0001$$

$$0b1000\_0010 = -2$$

Representing negative and signed integers

1s' complement

- Flip all bits
- Addition in 1s' complement is sound
- In this encoding there are 2 encodings for 0
- -0: 0b1111
- +0: 0b0000

most negative number

0b1000_0000

-127_{10}

Zero

0b0000_0000

0b1111_1111

most positive number

0b0111_1111

$= +127_{10}$

neg one

0b1111_1110

-1_{10}

one

0b0000_0001

$+1_{10}$

$-1_{10} + 2_{10}$

0b1111_1110

+) 0b0000_0010

0000_0000₂

$= +1_{10}$

Representing negative and signed integers

2's complement

power-of-two 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

most neg number

$0b1000\_0000$

$= -128_{10}$

zero

$0b0000\_0000$

$= 0_{10}$

most positive number

$0b0111\_1111$

$= +127_{10}$

negone

$0b1111\_1111$

$= -1_{10}$

one

$0b0000\_0001$

$= 1_{10}$

$+123_{10} - 52_{10}$

$1 \times 64 + 1 \times 32 + 1 \times 16 + 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1$

$0b0111\_1011$

-52_{10}

$= 1\times32 + 1\times16 + 0\times8 + 1\times4 + 0\times2 + 0\times1$

$= 0b\,0011\_0100$

↓ flip all bits

$0b\,1100\_1011$

↓ add 1

$0b\,1100\_1100$

-52_{10}

$= -128 + 1\times64 + 0\times32 + 0\times16 + 1\times8 + 1\times4 + 0\times2 + 0\times1$

$= 0b\,1100\_1100$

$123_{10} - 52$

$$\begin{array}{r} {\scriptstyle 1\ 1\ 1\ 1\ 1} \\ 0b\,0111\_1011 \\ +)\ 0b\,1100\_1100 \\ \hline 0100\ 0111 \end{array}$$

64 32 16 8 4 2 1

$1\times64 + 4 + 2 + 1$

$= 64 + 7 = 71$

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

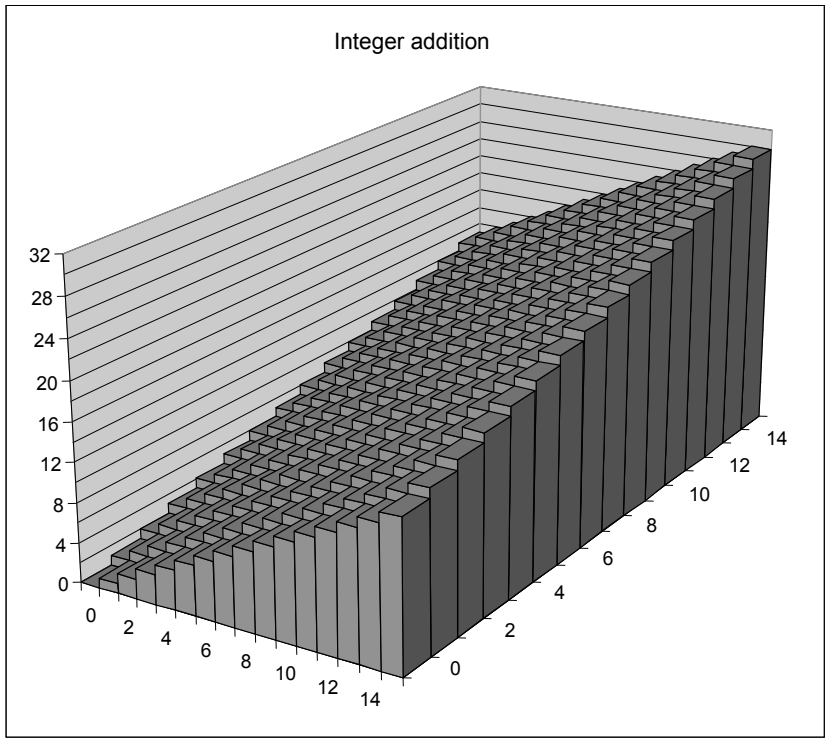

Figure: Image credit: CS:APP

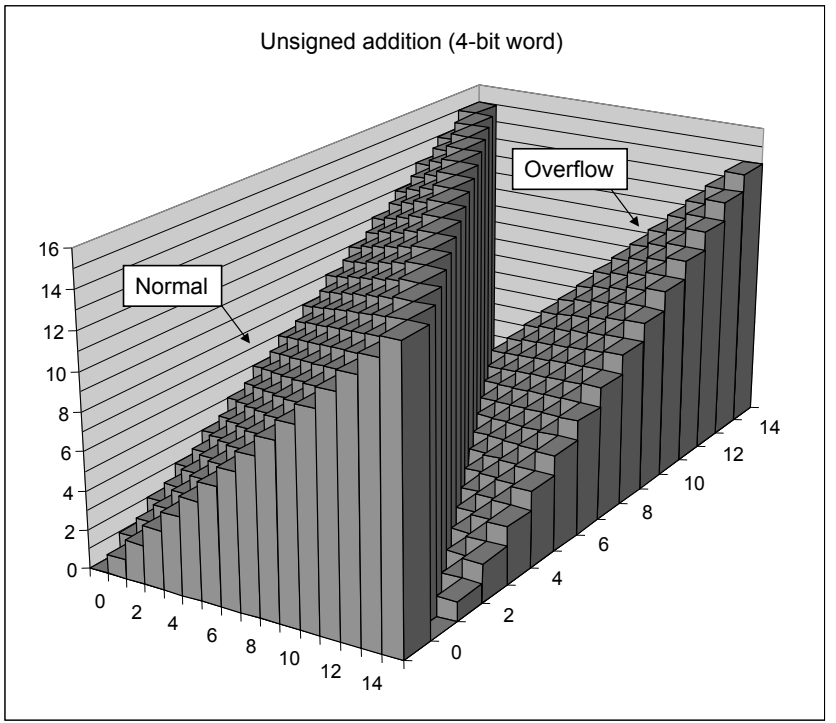

Figure: Image credit: CS:APP

Importance of paying attention to limits of encoding

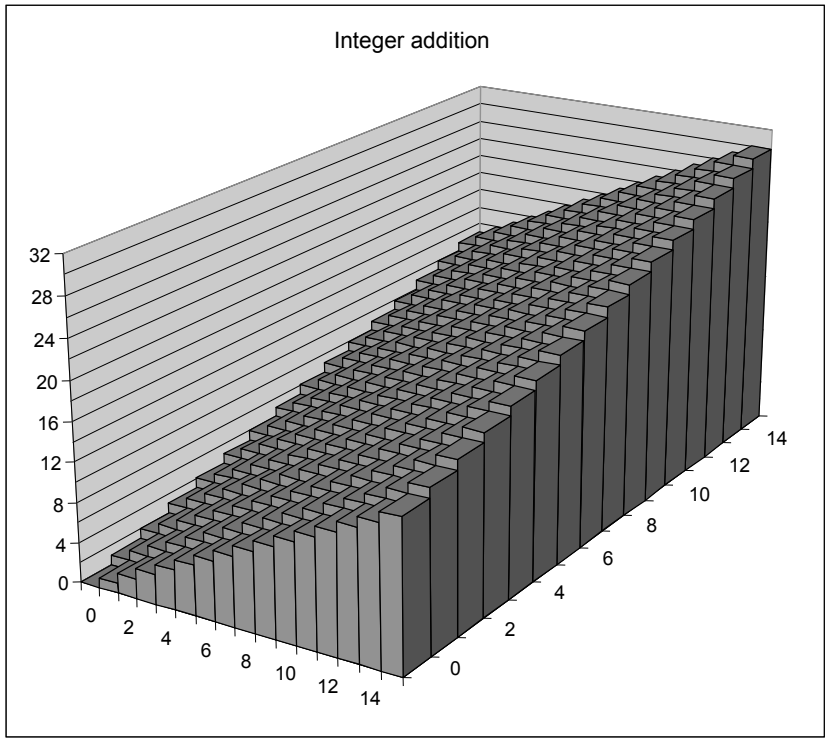

Figure: Image credit: CS:APP

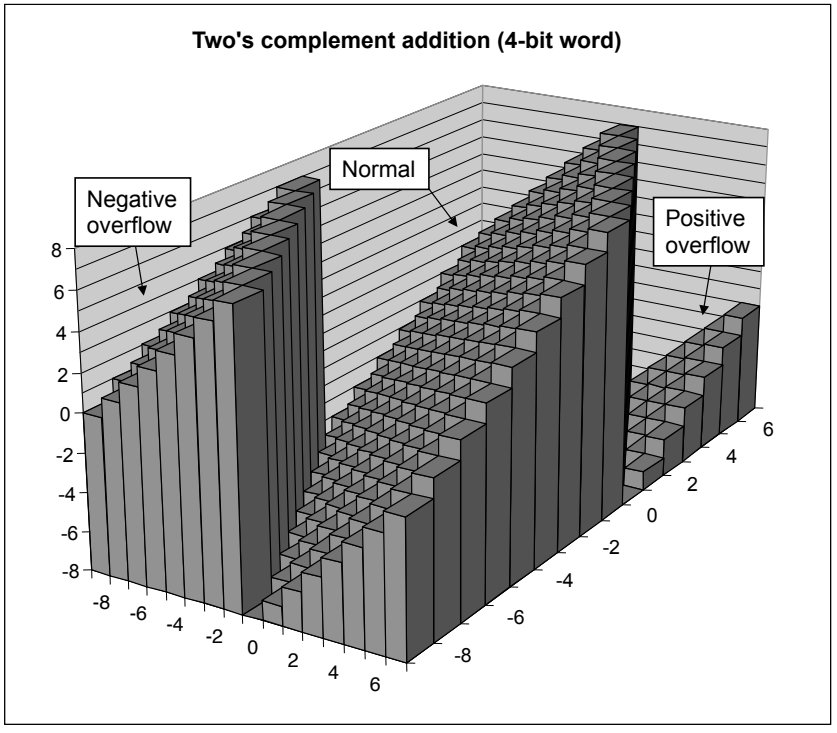

Figure: Image credit: CS:APP

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

signed int_ (32-bit)

most pos number

0b0111_...._1111

31 ones

$2^{16} = 2^{(10+6)} = 2^{10} \cdot 2^6 = 1024 \cdot 2^6$

$\approx 64 \times 1000$

~~$2^{32-1} = 2^{(3\times10+2)-1} = 2^{3\times10} \, 2^2 - 1 \approx (1000)^3 \cdot 2^2 - 1$~~

$2^{31} - 1 = 2^{(3\times10+1)} - 1 = (2^{10})^3 2^1 - 1$ ~~$\approx 4B - 1$~~

$\approx$ 2 Billion

GB : 10^9 B

GiB : $(2^{10})^3$ B

Gig: Byte

Table of contents

Unsigned fixed-point binary for fractions

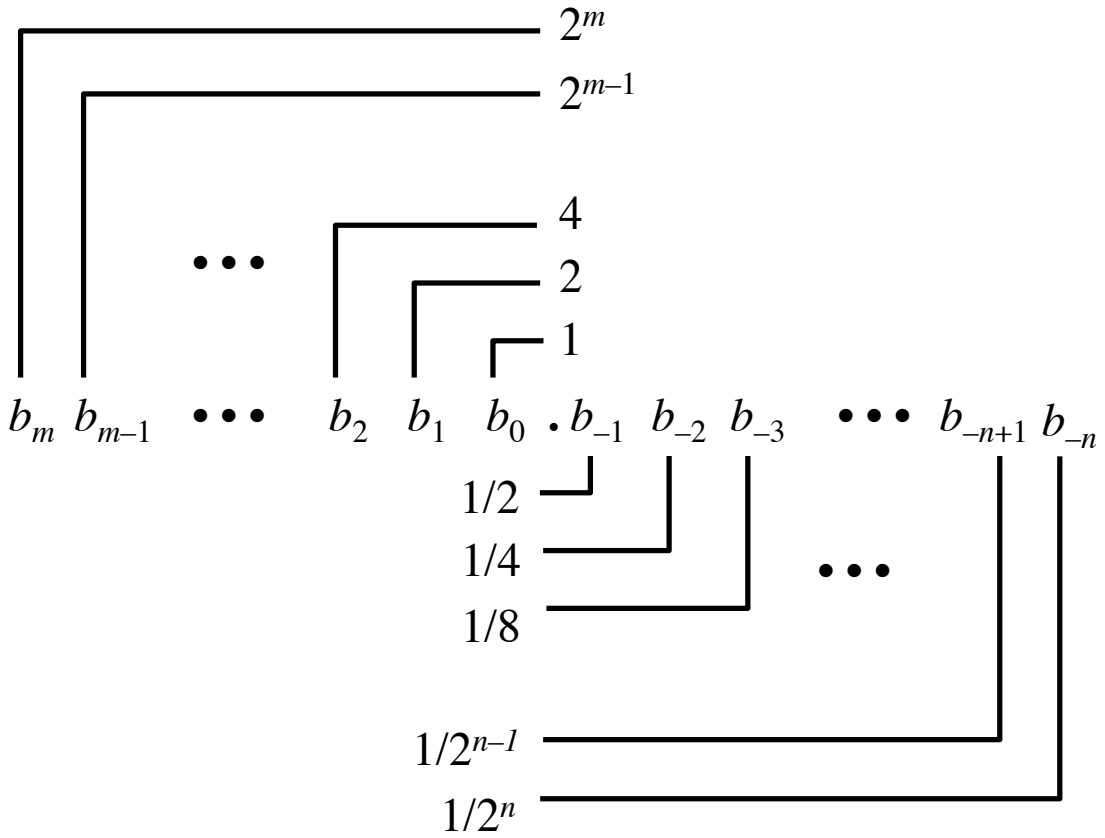

Figure: Fractional binary. Image credit CS:APP

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

$<<$ N Left shift by N bits

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

$>>$ N Right shift by N bits

- divides by 2^N
- $16 >> 3 = 0001\_0000_2 >> 3 = 0000\_0010_2 = 2 = 16/2^3$
- $-16 >> 3 = 1111\_0000_2 >> 3 = 1111\_1110_2 = -2 = -16/2^3$

123_{10}

$0b0111\_1011$

signed char number = 123;

printf("%d", number >> 0); // 123

printf("%d", number >> 1); // 61 : 123/2

0b0111_1011;

0b0011_1101; : 32+16+8+4+1

32 16 8 4 2 1

= 61

printf("%d", number >> 6; // 1

(number >> 6) & 0b1

0b0111_1011;

0b0000_0001

& 0b0000_0001

1

Table of contents

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

Table of contents

Floating point numbers

Avogadro's number

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

Scientific notation

- sign
- mantissa or significand
- exponent

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

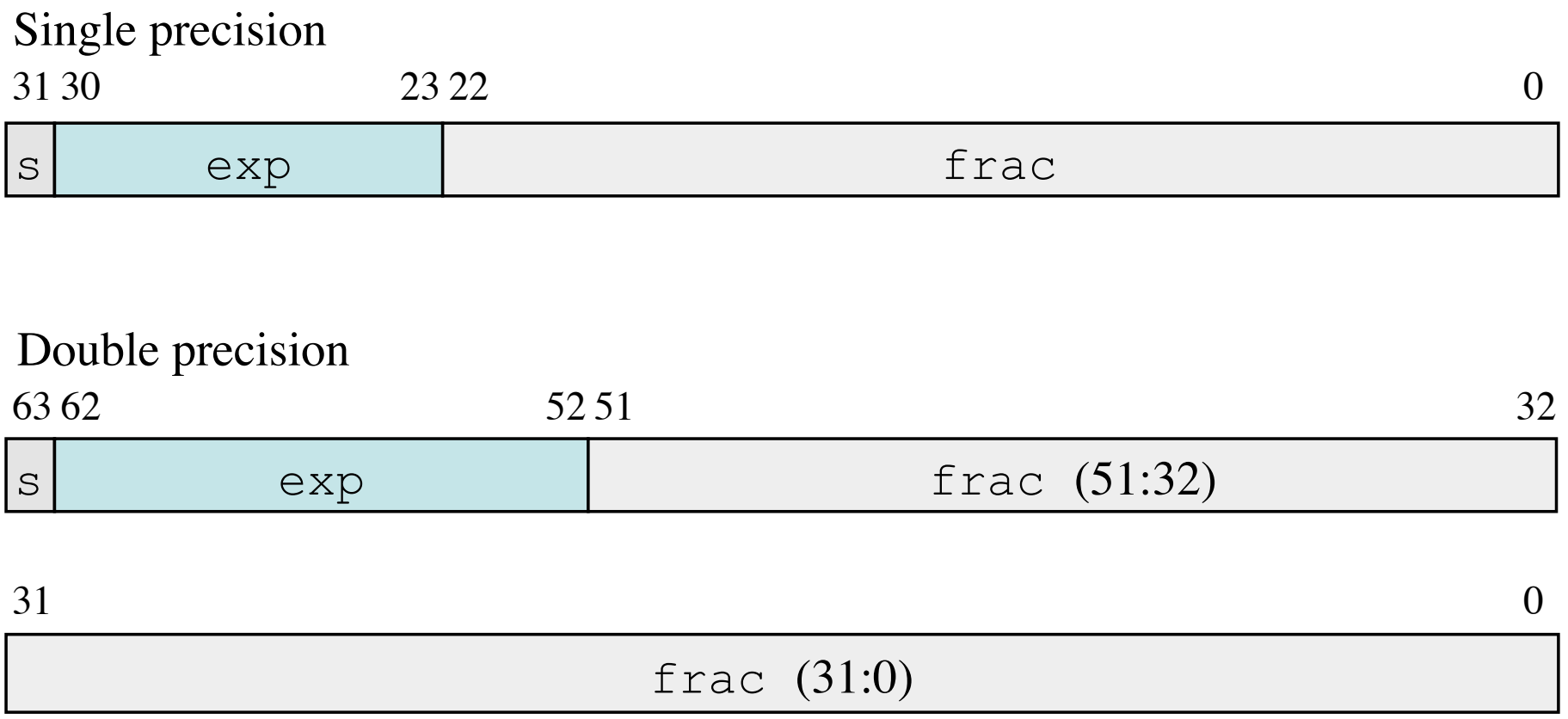

Figure: The two standard formats for floating point data types. Image credit CS:APP

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

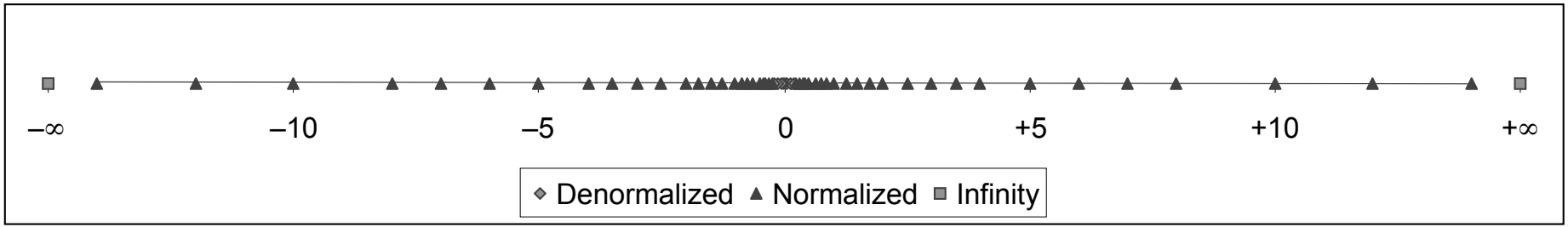

Figure: Full picture of number line for floating point values. Image credit CS:APP

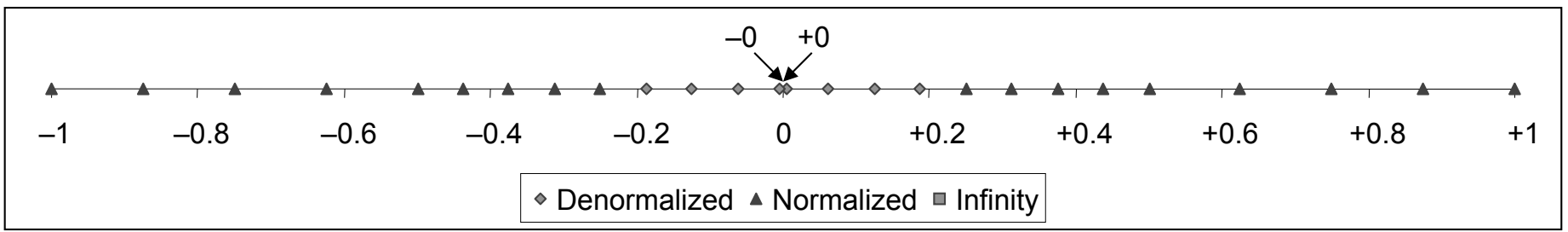

Figure: Zoomed in number line for floating point values. Image credit CS:APP

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

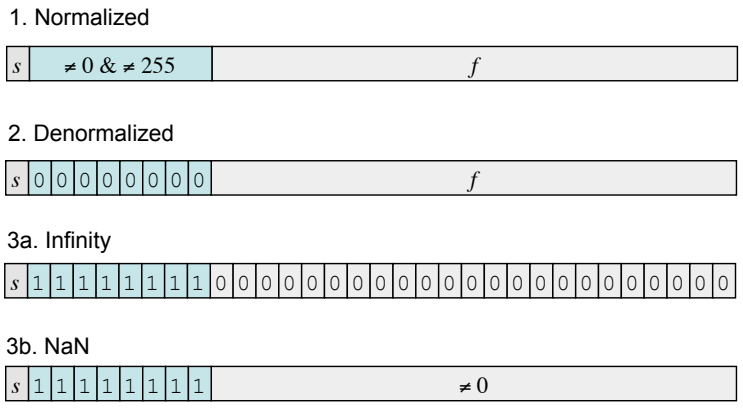

Figure: Different cases within a floating point format. Image credit CS:APP

Normalized and denormalized numbers

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

Table of contents

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
- bias = $2^{k-1} - 1$
- bias is the k-bit unsigned integer: 011..111

For normalized numbers,
E = exp-bias

In other words, exp = E+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

Normalized: frac field

M = 1.frac

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 $1.100011_2 \times 2^3$
- exp $= E +$ bias $= 3 + 127 = 130 = 1000\_0010_2$
- M = 1.100011_2 = 1.frac
- frac = 100011