Representing and Manipulating Information: Fixed point and floating point

Yipeng Huang

Rutgers University

February 24, 2022

Table of contents

Announcements
- Quizzes and programming assignments
- Reading and class session plan

Integers and basic arithmetic
- Representing negative and signed integers

Fractions and fixed point representation

Floats: Overview

Floats: Normalized numbers
- Normalized: exp field
- Normalized: frac field
- Normalized: example

Quizzes and programming assignments

Short quiz 4

- Weekly short quiz resumes next week. Same time frame, Tuesday to Thursday.

Programming assignment 2

- Has been out, extended now due tomorrow February 25, 11:59 pm.

Programming assignment 3

- Will release after lecture, due Thursday before spring break.

Reading and class session plan

Reading: CS:APP Chapter 2

- Chapter 2: Representing and manipulating information
- Read Chapter 2.4: floating point.

Recitation update

- Starting next week, March 2, Abhilash's recitation will move to Wednesdays 3:50 to 4:45 pm. CoRE 301.

Don't confuse the bitstring vs. the interpreted value

The bitstring

11111111, 377, 255, FF

Interpretation of the value

To interpret the value of a bitstring, you need to know:

1. the radix, number base: 2, 8, 10, 16.
2. the representation of signed values: two's complement.
3. size of the data type: char, short, int, long
4. decimal point

Table of contents

Announcements
Quizzes and programming assignments
Reading and class session plan

Integers and basic arithmetic
Representing negative and signed integers

Fractions and fixed point representation

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
2. 1s' complement
3. 2's complement

Representing negative and signed integers

Sign magnitude

Flip leading bit.

Representing negative and signed integers

1s′ complement

- Flip all bits
- Addition in 1s′ complement is sound: *if overflow add 1*.
- In this encoding there are 2 encodings for 0
- -0: 0b1111
- +0: 0b0000

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

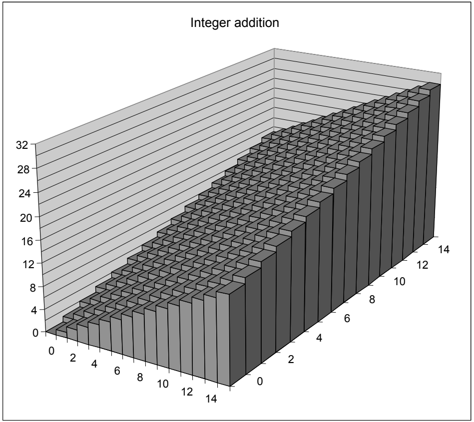

Figure: Image credit: CS:APP

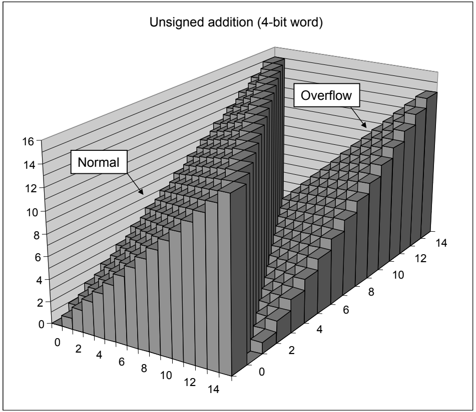

Figure: Image credit: CS:APP

Importance of paying attention to limits of encoding

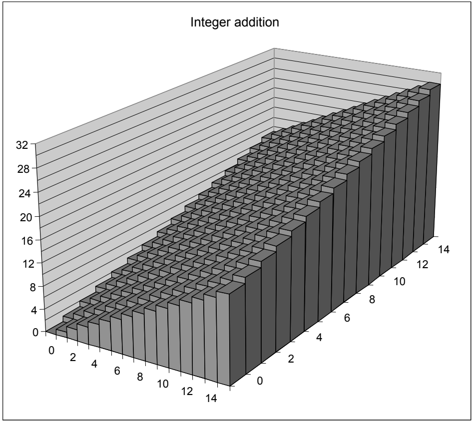

Figure: Image credit: CS:APP

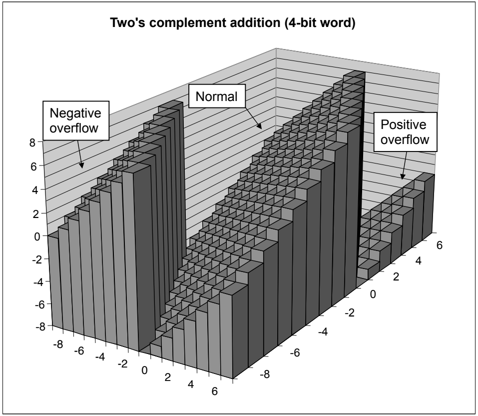

Figure: Image credit: CS:APP

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

`toBin.c`: Printing the binary representation

- Shifting and masking
- Try modifying to print octal.

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

Table of contents

Announcements
- Quizzes and programming assignments
- Reading and class session plan

Integers and basic arithmetic
- Representing negative and signed integers

Fractions and fixed point representation

Floats: Overview

Floats: Normalized numbers
- Normalized: exp field
- Normalized: frac field
- Normalized: example

Unsigned fixed-point binary for fractions

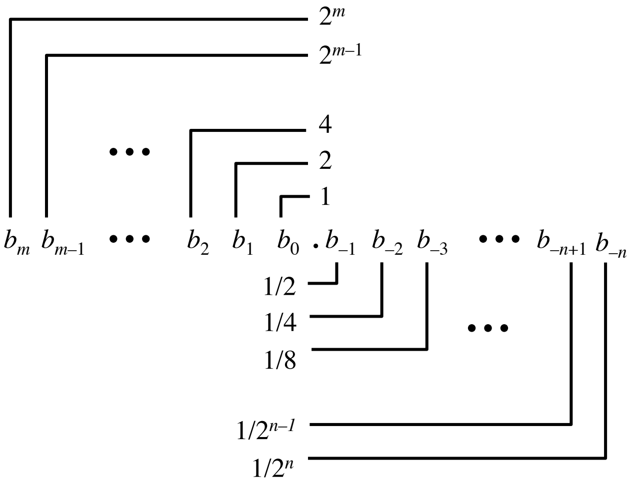

Figure: Fractional binary. Image credit CS:APP

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)

Table of contents

Announcements
- Quizzes and programming assignments
- Reading and class session plan

Integers and basic arithmetic
- Representing negative and signed integers

Fractions and fixed point representation

Floats: Overview

Floats: Normalized numbers
- Normalized: exp field
- Normalized: frac field
- Normalized: example

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

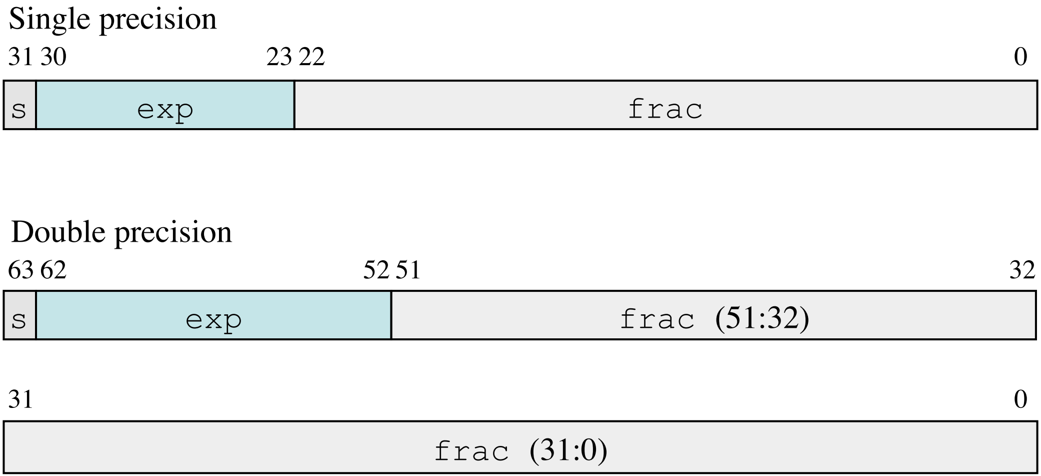

Figure: The two standard formats for floating point data types. Image credit CS:APP

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

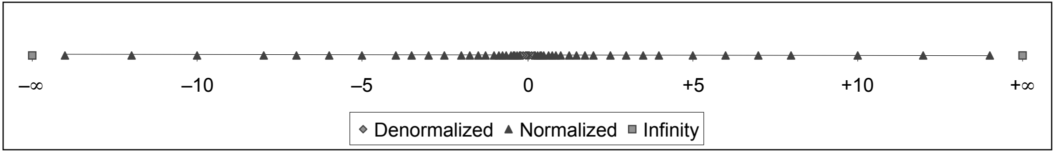

Figure: Full picture of number line for floating point values. Image credit CS:APP

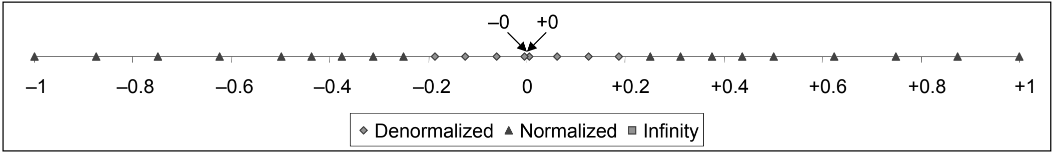

Figure: Zoomed in number line for floating point values. Image credit CS:APP

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

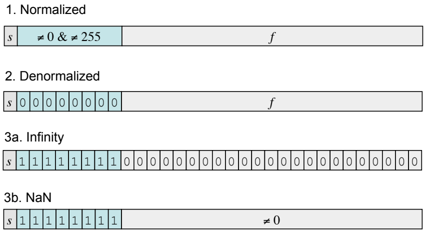

Figure: Different cases within a floating point format. Image credit CS:APP

Normalized and denormalized numbers

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

Table of contents

Announcements
- Quizzes and programming assignments
- Reading and class session plan

Integers and basic arithmetic
- Representing negative and signed integers

Fractions and fixed point representation

Floats: Overview

Floats: Normalized numbers
- Normalized: exp field
- Normalized: frac field
- Normalized: example

Normalized: exp field

For normalized numbers,
$0 < \exp < 2^k - 1$

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