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

<table>
<thead>
<tr>
<th>signed char</th>
<th>weight in decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000001</td>
<td>1</td>
</tr>
<tr>
<td>00000010</td>
<td>2</td>
</tr>
<tr>
<td>00000100</td>
<td>4</td>
</tr>
<tr>
<td>00001000</td>
<td>8</td>
</tr>
<tr>
<td>00010000</td>
<td>16</td>
</tr>
<tr>
<td>00100000</td>
<td>32</td>
</tr>
<tr>
<td>01000000</td>
<td>64</td>
</tr>
<tr>
<td>10000000</td>
<td>-128</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>signed char</th>
<th>weight in decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000001</td>
<td>1</td>
</tr>
<tr>
<td>00000010</td>
<td>2</td>
</tr>
<tr>
<td>00000100</td>
<td>4</td>
</tr>
<tr>
<td>00001000</td>
<td>8</td>
</tr>
<tr>
<td>00010000</td>
<td>16</td>
</tr>
<tr>
<td>00100000</td>
<td>32</td>
</tr>
<tr>
<td>01000000</td>
<td>64</td>
</tr>
<tr>
<td>10000000</td>
<td>-128</td>
</tr>
</tbody>
</table>

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

toBin.c: Printing the binary representation

- Shifting and masking
- Try modifying to print octal.
Bit shifting

$<< \; N$ Left shift by $N$ bits

- multiplies by $2^N$
- $2 << 3 = \text{0000}_2 << 3 = \text{0001}_2 = 16 = 2 \times 2^3$
- $-2 << 3 = \text{1111}_2 << 3 = \text{1111}_2 = -16 = -2 \times 2^3$

$>> \; N$ Right shift by $N$ bits

- divides by $2^N$
- $16 >> 3 = \text{0001}_2 >> 3 = \text{0000}_2 = 2 = 16/2^3$
- $-16 >> 3 = \text{1111}_2 >> 3 = \text{1111}_2 = -2 = -16/2^3$
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

<table>
<thead>
<tr>
<th>unsigned fixed-point char example</th>
<th>weight in decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000.0000</td>
<td>8</td>
</tr>
<tr>
<td>0100.0000</td>
<td>4</td>
</tr>
<tr>
<td>0010.0000</td>
<td>2</td>
</tr>
<tr>
<td>0001.0000</td>
<td>1</td>
</tr>
<tr>
<td>0000.1000</td>
<td>0.5</td>
</tr>
<tr>
<td>0000.0100</td>
<td>0.25</td>
</tr>
<tr>
<td>0000.0010</td>
<td>0.125</td>
</tr>
<tr>
<td>0000.0001</td>
<td>0.0625</td>
</tr>
</tbody>
</table>

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

- \(0.625 = 0.5 + 0.125 = 0000.1010_2\)
- \(1001.1000_2 = 9 + 0.5 = 9.5\)
Signed fixed-point binary for fractions

<table>
<thead>
<tr>
<th>signed fixed-point char example</th>
<th>weight in decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000.0000</td>
<td>-8</td>
</tr>
<tr>
<td>0100.0000</td>
<td>4</td>
</tr>
<tr>
<td>0010.0000</td>
<td>2</td>
</tr>
<tr>
<td>0001.0000</td>
<td>1</td>
</tr>
<tr>
<td>0000.1000</td>
<td>0.5</td>
</tr>
<tr>
<td>0000.0100</td>
<td>0.25</td>
</tr>
<tr>
<td>0000.0010</td>
<td>0.125</td>
</tr>
<tr>
<td>0000.0001</td>
<td>0.0625</td>
</tr>
</tbody>
</table>

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

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
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.
Floats and doubles

Single precision

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>23 22</td>
</tr>
<tr>
<td>s</td>
<td>exp</td>
<td>frac</td>
</tr>
</tbody>
</table>

Double precision

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>62</td>
<td>52 51</td>
</tr>
<tr>
<td>s</td>
<td>exp</td>
<td>frac (51:32)</td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>frac (31:0)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure:** The two standard formats for floating point data types. Image credit CS:APP
## Floats and doubles

<table>
<thead>
<tr>
<th>property</th>
<th>half*</th>
<th>float</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>total bits</td>
<td>16</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>s bit</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>exp bits</td>
<td>5</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>frac bits</td>
<td>10</td>
<td>23</td>
<td>52</td>
</tr>
<tr>
<td>C printf() format specifier</td>
<td>None</td>
<td>&quot;%f&quot;</td>
<td>&quot;%lf&quot;</td>
</tr>
</tbody>
</table>

**Table:** Properties of floats and doubles
The IEEE 754 number line

![Full picture of number line for floating point values. Image credit CS:APP](image)

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

![Zoomed in number line for floating point values. Image credit CS:APP](image)

**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

1. Normalized

$s \neq 0 \& s \neq 255$

2. Denormalized

$s = 0$

3a. Infinity

$s = 1$, $E = 0$

3b. NaN

$s = 1$, $E = 1$

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, $exp = E + bias$

<table>
<thead>
<tr>
<th>property</th>
<th>float</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>bias</td>
<td>127</td>
<td>1023</td>
</tr>
<tr>
<td>smallest E</td>
<td>-126</td>
<td>-1022</td>
</tr>
<tr>
<td>largest E</td>
<td>127</td>
<td>1023</td>
</tr>
</tbody>
</table>

Table: Summary of normalized exp field
Normalized: frac field

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