Assembly: Introduction.

Yipeng Huang

Rutgers University

February 23, 2021
Table of contents

Announcements

Floats: Review
   Normalized: exp field
   Normalized: frac field
   Normalized/denormalized
   Special cases

Floats: Mastery
   Normalized number bitstring to real number it represents
   Floating point multiplication
   Properties of floating point

Instruction set architectures
   why are instruction set architectures important
   8-bit vs. 16-bit vs. 32-bit vs. 64-bit
   CISC vs. RISC
Looking ahead

Class plan

1. Today, Tuesday, 2/23: Bits to floats. Introduction to the software-hardware interface.

2. Reading assignment for next four weeks: CS:APP Chapter 3.

3. Thursday, 2/25: Programming Assignment 3 on bits, bytes, integers, floats out.

Table of contents

Announcements

Floats: Review
   Normalized: exp field
   Normalized: frac field
   Normalized/denormalized
   Special cases

Floats: Mastery
   Normalized number bitstring to real number it represents
   Floating point multiplication
   Properties of floating point

Instruction set architectures
   why are instruction set architectures important
   8-bit vs. 16-bit. vs. 32-bit vs. 64-bit
   CISC vs. RISC
Floating point numbers

Avogadro’s number

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

Scientific notation

- sign
- mantissa or significand
- exponent
Floats and doubles

Single precision
31 30 23 22 0

\[ \text{s \ exp \ frac} \]

Double precision
63 62 52 51 32

\[ \text{s \ exp \ frac \ (51:32)} \]

31 0

\[ \text{frac \ (31:0)} \]

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

<table>
<thead>
<tr>
<th>property</th>
<th>float</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>total bits</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>s bit</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>exp bits</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>frac bits</td>
<td>23</td>
<td>52</td>
</tr>
<tr>
<td>C printf() format specifier</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

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

+1.0/INF = +0; -1.0/INF = -0

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

2. Denormalized

3a. Infinity

3b. NaN

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
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, $\text{exp} = E + \text{bias}$

<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 $E$ (greatest precision)</td>
<td>-126</td>
<td>-1022</td>
</tr>
<tr>
<td>largest $E$ (greatest range)</td>
<td>127</td>
<td>1023</td>
</tr>
</tbody>
</table>

Table: Summary of normalized exp field
Normalized: \( \frac{\text{field}}{\text{M} = 1.\frac{\text{}}{\text{}}} \)
## Floats: Summary

<table>
<thead>
<tr>
<th></th>
<th>normalized</th>
<th>denormalized</th>
</tr>
</thead>
<tbody>
<tr>
<td>value of number</td>
<td>$(-1)^s \times M \times 2^E$</td>
<td>$(-1)^s \times M \times 2^E$</td>
</tr>
<tr>
<td>E</td>
<td>$E = \text{exp-bias}$</td>
<td>$E = -\text{bias} + 1$</td>
</tr>
<tr>
<td>bias</td>
<td>$2^{k-1} - 1$</td>
<td>$2^{k-1} - 1$</td>
</tr>
<tr>
<td>exp</td>
<td>$0 &lt; \text{exp} &lt; (2^k - 1)$</td>
<td>$\text{exp} = 0$</td>
</tr>
<tr>
<td>M</td>
<td>$M = 1.\text{frac}$</td>
<td>$M = 0.\text{frac}$</td>
</tr>
<tr>
<td></td>
<td>M has implied leading 1</td>
<td>M has leading 0</td>
</tr>
<tr>
<td>greater range</td>
<td>greater precision</td>
<td>greater precision</td>
</tr>
<tr>
<td>large magnitude numbers</td>
<td>denser near origin</td>
<td>small magnitude numbers</td>
</tr>
<tr>
<td>denser near origin</td>
<td>evenly spaced</td>
<td></td>
</tr>
</tbody>
</table>

**Table:** Summary of normalized and denormalized numbers
# Floats: Special cases

<table>
<thead>
<tr>
<th>number class</th>
<th>when it arises</th>
<th>exp field</th>
<th>frac field</th>
</tr>
</thead>
<tbody>
<tr>
<td>+0 / -0</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>+infinity / -infinity</td>
<td>overflow or division by 0</td>
<td>$2^k - 1$</td>
<td>0</td>
</tr>
<tr>
<td>NaN not-a-number</td>
<td>illegal ops. such as $\sqrt{-1}$, inf-inf, inf*0</td>
<td>$2^k - 1$</td>
<td>non-0</td>
</tr>
</tbody>
</table>

Table: Summary of special cases
Table of contents

Announcements

Floats: Review
   Normalized: exp field
   Normalized: frac field
   Normalized/denormalized
   Special cases

Floats: Mastery
   Normalized number bitstring to real number it represents
   Floating point multiplication
   Properties of floating point

Instruction set architectures
   why are instruction set architectures important
   8-bit vs. 16-bit. vs. 32-bit vs. 64-bit
   CISC vs. RISC
Normalized number bitstring to real number it represents

- Tiny FP: 1-bit s, 4-bit exp, 3-bit frac
- What does this encode: $1_{1001\_101}$
- $s=1$, so positive? negative? Negative number
- $exp = 1001_2 = 8+1 = 9$
- $bias = 2^{k-1} - 1 = 2^{4-1} - 1 = 7$
- $E = exp-bias = 9-7 = 2$
- $frac = 101_2$
- $M = 1.frac = 1.101_2$
How to multiply scientific notation?

Avagadro number squared?

\((+6.02\times6.02)(10^{23})\times(10^{23}) = (+6.02\times6.02)\times10^{46}\)
\[= 36.0\times10^{46}\]
\[= 3.60\times10^{47}\]

Recall: \(\log(x \times y) = \log(x) + \log(y)\)
Floating point multiplication

**FP Multiplication**

- \((-1)^{s_1} M_1 \times 2^{E_1} \times (-1)^{s_2} M_2 \times 2^{E_2}\)
- **Exact Result:** \((-1)^s M \times 2^E\)
  - Sign \(s\): \(s_1 \oplus s_2\)  \(= s_1 \text{ XOR } s_2 = s_1+s_2 \text{ (mod2)}\)
  - Significand \(M\): \(M_1 \times M_2\)
  - Exponent \(E\): \(E_1 + E_2\)

- **Fixing**
  - If \(M \geq 2\), shift \(M\) right, increment \(E\)
  - If \(E\) out of range, overflow
  - Round \(M\) to fit \texttt{frac} precision

- **Implementation**
  - Biggest chore is multiplying significands
Mathematical Properties of FP Add

- Compare to those of Abelian Group
  - Closed under addition? \( \text{Yes} \)
    - But may generate infinity or NaN
  - Commutative? \( \text{Yes} \)
  - Associative? \( \text{No} \)
    - Overflow and inexactness of rounding
      - \((3.14+1e10)-1e10 = 0\), \(3.14+(1e10-1e10) = 3.14\)
  - 0 is additive identity? \( \text{Yes} \)
  - Every element has additive inverse? \( \text{Yes} \)
    - Yes, except for infinities & NaNs \( \text{Almost} \)

- Monotonicity
  - \( a \geq b \Rightarrow a+c \geq b+c? \) \( \text{Almost} \)
    - Except for infinities & NaNs
Mathematical Properties of FP Mult

- Compare to Commutative Ring
  - Closed under multiplication? \[ \text{Yes} \]
  - But may generate infinity or NaN
  - Multiplication Commutative? \[ \text{Yes} \]
  - Multiplication is Associative? \[ \text{No} \]
  - Possibility of overflow, inexactness of rounding
    - Ex: \((1e20\times1e20)\times1e-20=\text{inf},\ 1e20\times(1e20\times1e-20)=1e20\)
  - 1 is multiplicative identity? \[ \text{Yes} \]
  - Multiplication distributes over addition? \[ \text{No} \]
    - Possibility of overflow, inexactness of rounding
      - \(1e20\times(1e20-1e20)=0.0,\ 1e20\times1e20 - 1e20\times1e20 = \text{NaN}\)

- Monotonicity
  - \(a \geq b \& c \geq 0 \Rightarrow a \times c \geq b \times c?\) \[ \text{Almost} \]
  - Except for infinities & NaNs
Table of contents

Announcements

Floats: Review
  Normalized: exp field
  Normalized: frac field
  Normalized/denormalized
  Special cases

Floats: Mastery
  Normalized number bitstring to real number it represents
  Floating point multiplication
  Properties of floating point

Instruction set architectures
  why are instruction set architectures important
  8-bit vs. 16-bit. vs. 32-bit vs. 64-bit
  CISC vs. RISC
Computer organization
Layer cake

- Society
- Human beings
- Applications
- Algorithms
- High-level programming languages  Python, Java
- Interpreters
- Low-level programming languages
- Compilers
- Architectures
- Microarchitectures
- Sequential/combinational logic
- Transistors
- Semiconductors
- Materials science
why are instruction set architectures important

Interface between computer science and electrical and computer engineering

- Software is varied, changes
- Hardware is standardized, static

Computer architect Fred Brooks and the IBM 360

- IBM was selling computers with different capacities,
- Compile once, and can run software on all IBM machines.
- Backward compatibility.
- An influential idea.
CISC vs. RISC

Complex instruction set computer

- Intel and AMD
- Have an extensive and complex set of instructions
- For example: x86’s extensions: x87, IA-32, x86-64, MMX, 3DNow!, SSE, SSE2, SSE3, SSSE3, SSE4, SSE4.2, SSE5, AES-NI, CLMUL, RDRAND, SHA, MPX, SGX, XOP, F16C, ADX, BMI, FMA, AVX, AVX2, AVX512, VT-x, VT-d, AMD-V, AMD-Vi, TSX, ASF
- Can license Intel’s compilers to extract performance
- Secret: inside the processor, they break it down to more elementary instructions
CISC vs. RISC

Reduced instruction set computer

- MIPS, ARM, RISC-V (can find Patterson and Hennessy Computer Organization and Design textbook in each of these versions), an PowerPC
- Have a relatively simple set of instructions
- For example: ARM’s extensions: SVE;SVE2;TME; All mandatory: Thumb-2, Neon, VFPv4-D16, VFPv4 Obsolete: Jazelle
- ARM: smartphones, Apple ARM M1 Mac

\[ [1, 2, 4, 5, \ldots, 9] \times 2 = [2, 4, \ldots] \]
Into the future: Post-ISA world

Post-ISA world

- Increasingly, the CPU is not the only character
- It orchestrates among many pieces of hardware
- Smartphone die shot
- GPU, TPU, FPGA, ASIC

Figure: Apple A13 (2019 Apple iPhone 11 CPU). Image credit AnandTech