CS 61C:  
Great Ideas in Computer Architecture  
*Performance and Floating Point Arithmetic*

**Instructors:**  
John Wawrzynek & Vladimir Stojanovic  
http://inst.eecs.berkeley.edu/~cs61c/
New-School Machine Structures (It’s a bit more complicated!)

- **Parallel Requests**
  Assigned to computer
  e.g., Search “Katz”

- **Parallel Threads**
  Assigned to core
  e.g., Lookup, Ads

- **Parallel Instructions**
  >1 instruction @ one time
  e.g., 5 pipelined instructions

- **Parallel Data**
  >1 data item @ one time
  e.g., Add of 4 pairs of words

- **Hardware descriptions**
  All gates @ one time

- **Programming Languages**

---

Software

- **Harness**
  Parallelism & Achieve High Performance

Hardware

- Warehouse Scale Computer
- How do we know?

Workflow:

- Smart Phone
- Hardware
  - Core
  - Memory
  - Input/Output
  - Instruction Unit(s)
  - Functional Unit(s)
  - Cache Memory
  - Logic Gates

How do we know?
What is Performance?

• *Latency* (or *response time* or *execution time*)
  – Time to complete one task

• *Bandwidth* (or *throughput*)
  – Tasks completed per unit time
Cloud Performance: Why Application Latency Matters

<table>
<thead>
<tr>
<th>Server Delay (ms)</th>
<th>Increased time to next click (ms)</th>
<th>Queries/user</th>
<th>Any clicks/user</th>
<th>User satisfaction</th>
<th>Revenue/User</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>200</td>
<td>500</td>
<td>--</td>
<td>-0.3%</td>
<td>-0.4%</td>
<td>--</td>
</tr>
<tr>
<td>500</td>
<td>1200</td>
<td>--</td>
<td>-1.0%</td>
<td>-0.9%</td>
<td>-1.2%</td>
</tr>
<tr>
<td>1000</td>
<td>1900</td>
<td>-0.7%</td>
<td>-1.9%</td>
<td>-1.6%</td>
<td>-2.8%</td>
</tr>
<tr>
<td>2000</td>
<td>3100</td>
<td>-1.8%</td>
<td>-4.4%</td>
<td>-3.8%</td>
<td>-4.3%</td>
</tr>
</tbody>
</table>

Figure 6.10 Negative impact of delays at Bing search server on user behavior [Brutlag and Schurman 2009].

- Key figure of merit: application responsiveness
  - Longer the delay, the fewer the user clicks, the less the user happiness, and the lower the revenue per user
Defining CPU Performance

- What does it mean to say X is faster than Y?
- Ferrari vs. School Bus?
  - 2013 Ferrari 599 GTB
    - 2 passengers, quarter mile in 10 secs
  - 2013 Type D school bus
    - 50 passengers, quarter mile in 20 secs

- Response Time (Latency): e.g., time to travel ¼ mile
- Throughput (Bandwidth): e.g., passenger-mi in 1 hour
Defining Relative CPU Performance

• Performance$_X$ = 1/Program Execution Time$_X$
• Performance$_X$ > Performance$_Y$ =>
  1/Execution Time$_X$ > 1/Execution Time$_Y$ =>
  Execution Time$_Y$ > Execution Time$_X$
• Computer X is N times faster than Computer Y
  Performance$_X$ / Performance$_Y$ = N or
  Execution Time$_Y$ / Execution Time$_X$ = N

• Bus to Ferrari performance:
  – Program: Transfer 1000 passengers for 1 mile
  – Bus: 3,200 sec, Ferrari: 40,000 sec
Measuring CPU Performance

• Computers use a clock to determine when events takes place within hardware

• *Clock cycles*: discrete time intervals
  – aka clocks, cycles, clock periods, clock ticks

• *Clock rate* or *clock frequency*: clock cycles per second (inverse of clock cycle time)

• 3 GigaHertz clock rate
  => clock cycle time = $1/(3 \times 10^9)$ seconds
  clock cycle time = 333 picoseconds (ps)
CPU Performance Factors

• To distinguish between processor time and I/O, *CPU time* is time spent in processor

• CPU Time/Program
  = Clock Cycles/Program
  x Clock Cycle Time

• Or

  CPU Time/Program
  = Clock Cycles/Program ÷ Clock Rate
Iron Law of Performance
by Emer and Clark

• A program executes instructions

• CPU Time/Program
  = Clock Cycles/Program $\times$ Clock Cycle Time
  = Instructions/Program
    $\times$ Average Clock Cycles/Instruction
    $\times$ Clock Cycle Time

• 1$^{st}$ term called *Instruction Count*

• 2$^{nd}$ term abbreviated *CPI* for average *Clock Cycles Per Instruction*

• 3rd term is 1 / Clock rate
Restating Performance Equation

- Time = \frac{\text{Seconds}}{\text{Program}} \times \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}}
What Affects Each Component?
A) Instruction Count, B) CPI, C) Clock Rate

<table>
<thead>
<tr>
<th>Component</th>
<th>Affects What? (click in letter of component not affected)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td></td>
</tr>
<tr>
<td>Programming Language</td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td></td>
</tr>
<tr>
<td>Instruction Set Architecture</td>
<td></td>
</tr>
</tbody>
</table>
## What Affects Each Component? Instruction Count, CPI, Clock Rate

<table>
<thead>
<tr>
<th>Component</th>
<th>Affects What?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>Instruction Count, CPI</td>
</tr>
<tr>
<td>Programming Language</td>
<td>Instruction Count, CPI</td>
</tr>
<tr>
<td>Compiler</td>
<td>Instruction Count, CPI</td>
</tr>
<tr>
<td>Instruction Set Architecture</td>
<td>Instruction Count, Clock Rate, CPI</td>
</tr>
</tbody>
</table>
Clickers

<table>
<thead>
<tr>
<th>Computer</th>
<th>Clock frequency</th>
<th>Clock cycles per instruction</th>
<th>#instructions per program</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1GHz</td>
<td>2</td>
<td>1000</td>
</tr>
<tr>
<td>B</td>
<td>2GHz</td>
<td>5</td>
<td>800</td>
</tr>
<tr>
<td>C</td>
<td>500MHz</td>
<td>1.25</td>
<td>400</td>
</tr>
<tr>
<td>D</td>
<td>5GHz</td>
<td>10</td>
<td>2000</td>
</tr>
</tbody>
</table>

• Which computer has the highest performance for a given program?
Workload and Benchmark

- **Workload**: Set of programs run on a computer
  - Actual collection of applications run or made from real programs to approximate such a mix
  - Specifies programs, inputs, and relative frequencies

- **Benchmark**: Program selected for use in comparing computer performance
  - Benchmarks form a workload
  - Usually standardized so that many use them
SPEC
(System Performance Evaluation Cooperative)

• Computer Vendor cooperative for benchmarks, started in 1989
• SPECCPU2006
  – 12 Integer Programs
  – 17 Floating-Point Programs
• Often turn into number where bigger is faster
• SPECratio: reference execution time on old reference computer divide by execution time on new computer to get an effective speed-up
<table>
<thead>
<tr>
<th>Description</th>
<th>Instruction Count (B)</th>
<th>CPI</th>
<th>Clock cycle time (ps)</th>
<th>Execution Time (s)</th>
<th>Reference Time (s)</th>
<th>SPEC-ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpreted string processing</td>
<td>2,118</td>
<td>0.75</td>
<td>400</td>
<td>637</td>
<td>9,770</td>
<td>15.3</td>
</tr>
<tr>
<td>Block-sorting compression</td>
<td>2,389</td>
<td>0.85</td>
<td>400</td>
<td>817</td>
<td>9,650</td>
<td>11.8</td>
</tr>
<tr>
<td>GNU C compiler</td>
<td>1,050</td>
<td>1.72</td>
<td>400</td>
<td>724</td>
<td>8,050</td>
<td>11.1</td>
</tr>
<tr>
<td>Combinatorial optimization</td>
<td>336</td>
<td>10.0</td>
<td>400</td>
<td>1,345</td>
<td>9,120</td>
<td>6.8</td>
</tr>
<tr>
<td>Go game</td>
<td>1,658</td>
<td>1.09</td>
<td>400</td>
<td>721</td>
<td>10,490</td>
<td>14.6</td>
</tr>
<tr>
<td>Search gene sequence</td>
<td>2,783</td>
<td>0.80</td>
<td>400</td>
<td>890</td>
<td>9,330</td>
<td>10.5</td>
</tr>
<tr>
<td>Chess game</td>
<td>2,176</td>
<td>0.96</td>
<td>400</td>
<td>837</td>
<td>12,100</td>
<td>14.5</td>
</tr>
<tr>
<td>Quantum computer simulation</td>
<td>1,623</td>
<td>1.61</td>
<td>400</td>
<td>1,047</td>
<td>20,720</td>
<td>19.8</td>
</tr>
<tr>
<td>Video compression</td>
<td>3,102</td>
<td>0.80</td>
<td>400</td>
<td>993</td>
<td>22,130</td>
<td>22.3</td>
</tr>
<tr>
<td>Discrete event simulation library</td>
<td>587</td>
<td>2.94</td>
<td>400</td>
<td>690</td>
<td>6,250</td>
<td>9.1</td>
</tr>
<tr>
<td>Games/path finding</td>
<td>1,082</td>
<td>1.79</td>
<td>400</td>
<td>773</td>
<td>7,020</td>
<td>9.1</td>
</tr>
<tr>
<td>XML parsing</td>
<td>1,058</td>
<td>2.70</td>
<td>400</td>
<td>1,143</td>
<td>6,900</td>
<td>6.0</td>
</tr>
</tbody>
</table>
## Summarizing Performance ...

<table>
<thead>
<tr>
<th>System</th>
<th>Rate (Task 1)</th>
<th>Rate (Task 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>B</td>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>

**Clickers: Which system is faster?**

- **A: System A**
- **B: System B**
- **C: Same performance**
- **D: Unanswerable question!**
... Depends Who’s Selling

<table>
<thead>
<tr>
<th>System</th>
<th>Rate (Task 1)</th>
<th>Rate (Task 2)</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>20</td>
<td>15</td>
</tr>
<tr>
<td>B</td>
<td>20</td>
<td>10</td>
<td>15</td>
</tr>
</tbody>
</table>

Average throughput

<table>
<thead>
<tr>
<th>System</th>
<th>Rate (Task 1)</th>
<th>Rate (Task 2)</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.50</td>
<td>2.00</td>
<td>1.25</td>
</tr>
<tr>
<td>B</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Throughput relative to B

<table>
<thead>
<tr>
<th>System</th>
<th>Rate (Task 1)</th>
<th>Rate (Task 2)</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>B</td>
<td>2.00</td>
<td>0.50</td>
<td>1.25</td>
</tr>
</tbody>
</table>

Throughput relative to A
Summarizing SPEC Performance

• Varies from 6x to 22x faster than reference computer

• Geometric mean of ratios: N-th root of product of N ratios
  – Geometric Mean gives same relative answer no matter what computer is used as reference

• Geometric Mean for Barcelona is 11.7
Administrivia

• HW3 released, due 11/01@23:59:59

• Project 3-2 released, due 11/01@23:59:59

• Midterm 2 on 11/10 in class
  – For midterm 2 re-scheduling (not dsp) contact Fred and William
CodeBears Announcement

• “CodeBears is a rapidly growing organization on campus aimed at providing members with the opportunity to compete with each other and against teams in other universities in algorithmic software competitions that require strong problem solving abilities. Winners of our intra-college programming competitions will be able to meet with industry leaders, network with founders of various organizations, and represent UCB at inter-college competitions. We hope to create a new hacking culture at Cal! Learn more about us at: https://callink.berkeley.edu/organization/codebears

• We already have over 100 registered members and had a tremendous turnout rate at our first GM last Friday! Visit our Facebook page at: https://www.facebook.com/CodeBears to access last Friday’s slides and learn how to join CodeBears!
Review of Numbers

• Computers are made to deal with numbers

• What can we represent in N bits?
  • $2^N$ things, and no more! They could be…
  • Unsigned integers:
    
    $0$ to $2^N - 1$

    (for $N=32$, $2^{32} - 1 = 4,294,967,295$)
  • Signed Integers (Two’s Complement)
    
    $-2^{(N-1)}$ to $2^{(N-1)} - 1$

    (for $N=32$, $2^{31} = 2,147,483,648$)
What about other numbers?

1. Very large numbers? (seconds/millennium)
   ⇒ 31,556,926,000_{10} (3.1556926_{10} x 10^{10})

2. Very small numbers? (Bohr radius)
   ⇒ 0.0000000000529177_{10}m (5.29177_{10} x 10^{-11})

3. Numbers with both integer & fractional parts?
   ⇒ 1.5

First consider #3.

...our solution will also help with #1 and #2.
Representation of Fractions

“Binary Point” like decimal point signifies boundary between integer and fractional parts:

Example 6-bit representation: $xx \cdot yyyyy$

$10.1010_{two} = 1 \times 2^1 + 1 \times 2^{-1} + 1 \times 2^{-3} = 2.625_{ten}$

If we assume “fixed binary point”, range of 6-bit representations with this format:

0 to 3.9375 (almost 4)
## Fractional Powers of 2

<table>
<thead>
<tr>
<th>(i)</th>
<th>(2^{-i})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
</tr>
<tr>
<td>3</td>
<td>0.125</td>
</tr>
<tr>
<td>4</td>
<td>0.0625</td>
</tr>
<tr>
<td>5</td>
<td>0.03125</td>
</tr>
<tr>
<td>6</td>
<td>0.015625</td>
</tr>
<tr>
<td>7</td>
<td>0.0078125</td>
</tr>
<tr>
<td>8</td>
<td>0.00390625</td>
</tr>
<tr>
<td>9</td>
<td>0.001953125</td>
</tr>
<tr>
<td>10</td>
<td>0.0009765625</td>
</tr>
<tr>
<td>11</td>
<td>0.00048828125</td>
</tr>
<tr>
<td>12</td>
<td>0.000244140625</td>
</tr>
<tr>
<td>13</td>
<td>0.0001220703125</td>
</tr>
<tr>
<td>14</td>
<td>0.00006103515625</td>
</tr>
<tr>
<td>15</td>
<td>0.000030517578125</td>
</tr>
</tbody>
</table>
What about addition and multiplication?

Addition is straightforward:

\[
\begin{align*}
01.100 & \quad 1.5\text{_{ten}} \\
+ 00.100 & \quad 0.5\text{_{ten}} \\
\hline
10.000 & \quad 2.0\text{_{ten}}
\end{align*}
\]

Multiplication a bit more complex:

\[
\begin{align*}
000000 & \\
01100 & \\
000000 & \\
0000110000 & \quad \text{1.5}_{ten}
\end{align*}
\]

Where's the answer, 0.11? (need to remember where point is)
Representation of Fractions

So far, in our examples we used a “fixed” binary point. What we really want is to “float” the binary point. Why?

Floating binary point most effective use of our limited bits (and thus more accuracy in our number representation):

example: put $0.1640625_{\text{ten}}$ into binary. Represent with 5-bits choosing where to put the binary point.

... 00000.001010100000...

Store these bits and keep track of the binary point 2 places to the left of the MSB

Any other solution would lose accuracy!

With floating-point rep., each numeral carries an exponent field recording the whereabouts of its binary point.

The binary point can be outside the stored bits, so very large and small numbers can be represented.
Scientific Notation (in Decimal)

mantissa \(6.02_{\text{ten}}\) x \(10^{23}\) exponent

decimal point

radix (base)

• Normalized form: no leadings 0s (exactly one digit to left of decimal point)

• Alternatives to representing \(1/1,000,000,000\)
  
  • Normalized: \(1.0 \times 10^{-9}\)
  
  • Not normalized: \(0.1 \times 10^{-8}, 10.0 \times 10^{-10}\)
Scientific Notation (in Binary)

mantissa \[1.01_{\text{two}} \times 2^{-1}\] exponent

“binary point” radix (base)

- Computer arithmetic that supports it called floating point, because it represents numbers where the binary point is not fixed, as it is for integers
  - Declare such variable in C as float
    - double for double precision.
Floating-Point Representation (1/2)

- Normal format: $+1.xxx...x_{two} \times 2^{yyy...y_{two}}$

- Multiple of Word Size (32 bits)

<table>
<thead>
<tr>
<th>S</th>
<th>Exponent</th>
<th>Significand</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bit</td>
<td>8 bits</td>
<td>23 bits</td>
</tr>
</tbody>
</table>

- **S** represents **Sign**
- **Exponent** represents **y**’s
- **Significand** represents **x**’s

- Represent numbers as small as $2.0_{ten} \times 10^{-38}$ to as large as $2.0_{ten} \times 10^{38}$
Floating-Point Representation (2/2)

• What if result too large?

\( (> 2.0 \times 10^{38} , < -2.0 \times 10^{38} ) \)

  - **Overflow!** \( \Rightarrow \) Exponent larger than represented in 8-bit Exponent field

• What if result too small?

\( (>0 & < 2.0 \times 10^{-38} , <0 & > -2.0 \times 10^{-38} ) \)

  - **Underflow!** \( \Rightarrow \) Negative exponent larger than represented in 8-bit Exponent field

• What would help reduce chances of overflow and/or underflow?
IEEE 754 Floating-Point Standard (1/3)

Single Precision (Double Precision similar):

\[ \begin{array}{cccccccccccccccccccccccccccccccccc}
\text{31} & \text{30} & \text{23} & \text{22} & \text{0} \\
S & \text{Exponent} & \text{Significand} \\
\end{array} \]

1 bit 8 bits 23 bits

- **Sign bit:** 1 means negative
  0 means positive

- **Significand in sign-magnitude format** (not 2’s complement)
  - To pack more bits, leading 1 implicit for normalized numbers
  - 1 + 23 bits single, 1 + 52 bits double
  - always true: 0 < Significand < 1
    (for normalized numbers)

- **Note:** 0 has no leading 1, so reserve exponent value 0 just for number 0
IEEE 754 Floating Point Standard (2/3)

- IEEE 754 uses “biased exponent” representation
  - Designers wanted FP numbers to be used even if no FP hardware; e.g., sort records with FP numbers using integer compares
  - Wanted bigger (integer) exponent field to represent bigger numbers
  - 2’s complement poses a problem (because negative numbers look bigger)
    - Use just magnitude and offset by half the range
IEEE 754 Floating Point Standard (3/3)

• Called **Biased Notation**, where bias is number subtracted to get final number
  
  - IEEE 754 uses bias of 127 for single prec.
  - Subtract 127 from Exponent field to get actual value for exponent

• **Summary (single precision):**

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>23</th>
<th>22</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Exponent</td>
<td>Significand</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  1 bit 8 bits 23 bits

  
  • \((-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}\)

  • Double precision identical, except with exponent bias of 1023 (half, quad similar)
"Father" of the Floating point standard

IEEE Standard 754 for Binary Floating-Point Arithmetic.

1989 ACM Turing Award Winner!

www.cs.berkeley.edu/~wkahan/ieee754status/754story.html
Clickers

• Guess this Floating Point number:
  1 1000 0000 1000 0000 0000 0000 0000 0000

A: -1x \(2^{128}\)
B: +1x \(2^{-128}\)
C: -1x 2
D: +1.5x 2
E: -1.5x 2
Representation for $\pm \infty$

- In FP, divide by 0 should produce $\pm \infty$, not overflow.

- Why?
  - OK to do further computations with $\infty$
    E.g., $X/0 > Y$ may be a valid comparison

- IEEE 754 represents $\pm \infty$
  - Most positive exponent reserved for $\infty$
  - Significands all zeroes
Representation for 0

- Represent 0?
  - exponent all zeroes
  - significand all zeroes
- What about sign? Both cases valid

+0: 0 00000000 000000000000000000000000

-0: 1 00000000 000000000000000000000000
Special Numbers

• What have we defined so far? (Single Precision)

<table>
<thead>
<tr>
<th>Exponent</th>
<th>Significand</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>nonzero</td>
<td>???</td>
</tr>
<tr>
<td>1-254</td>
<td>anything</td>
<td>+/- fl. pt. #</td>
</tr>
<tr>
<td>255</td>
<td>0</td>
<td>+/-  ∞</td>
</tr>
<tr>
<td>255</td>
<td>nonzero</td>
<td>???</td>
</tr>
</tbody>
</table>

• Professor Kahan had clever ideas:
  • Wanted to use Exp=0,255 & Sig!=0
Representation for Not a Number

• What do I get if I calculate \( \sqrt{-4.0} \) or \( 0/0 \)?
  • If \( \infty \) not an error, these shouldn’t be either
  • Called **Not a Number** (NaN)
  • Exponent = 255, Significand nonzero

• Why is this useful?
  • Hope NaNs help with debugging?
  • They contaminate: \( \text{op}(\text{NaN}, X) = \text{NaN} \)
  • Can use the significand to identify which!
Representation for Denorms (1/2)

- **Problem:** There’s a gap among representable FP numbers around 0

  - **Smallest representable pos num:**
    
    \[ a = 1.0\ldots_2 * 2^{-126} = 2^{-126} \]

  - **Second smallest representable pos num:**
    
    \[ b = 1.000\ldots_2 * 2^{-126} = (1 + 0.00\ldots_2) * 2^{-126} = (1 + 2^{-23}) * 2^{-126} = 2^{-126} + 2^{-149} \]

  \[ a - 0 = 2^{-126} \]

  \[ b - a = 2^{-149} \]

  ![Diagram showing a gap between 0 and a and b]

  Normalization and implicit 1 is to blame!
• Solution:
  • We still haven’t used Exponent = 0, Significand nonzero
  • DEnormalized number: no (implied) leading 1, implicit exponent = -126.
  • Smallest representable pos num:
    \[ a = 2^{-149} \]
  • Second smallest representable pos num:
    \[ b = 2^{-148} \]

-∞ \[ \rightarrow \] 0 \[ \rightarrow \] +∞
### Special Numbers Summary

- **Reserve exponents, significands:**

<table>
<thead>
<tr>
<th>Exponent</th>
<th>Significand</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>nonzero</td>
<td><strong>Denorm</strong></td>
</tr>
<tr>
<td>1-254</td>
<td>anything</td>
<td>+/- fl. pt. #</td>
</tr>
<tr>
<td>255</td>
<td>0</td>
<td>+/- ∞</td>
</tr>
<tr>
<td>255</td>
<td>nonzero</td>
<td><strong>NaN</strong></td>
</tr>
</tbody>
</table>
Conclusion

Floating Point lets us:

- Represent numbers containing both integer and fractional parts; makes efficient use of available bits.
- Store approximate values for very large and very small numbers.

IEEE 754 Floating-Point Standard is most widely accepted attempt to standardize interpretation of such numbers (Every desktop or server computer sold since ~1997 follows these conventions)

Summary (single precision):

<table>
<thead>
<tr>
<th>S</th>
<th>Exponent</th>
<th>Significand</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bit</td>
<td>8 bits</td>
<td>23 bits</td>
</tr>
</tbody>
</table>

\((-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent} - 127)}\)

Double precision identical, except with exponent bias of 1023 (half, quad similar)
And In Conclusion, ...

- Time (seconds/program) is measure of performance

\[
\text{Time (seconds/program)} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}}
\]

- Floating-point representations hold approximations of real numbers in a finite number of bits