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

Instructors:
Nick Weaver & John Wawrzynek
http://inst.eecs.berkeley.edu/~cs61c/sp18

Outline

- Floating-Point Representation
- Defining and Measuring Performance
- And in Conclusion …

Outline

- Floating-Point Standard
- Defining Performance
- And in Conclusion …

Quick Number Review

- Computers 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, $2^N - 1$ = 4,294,967,295)
    - Signed Integers (Two's Complement)

      $-2^{(N-1)}$ to $2^{(N-1)} - 1$

      (for N=32, $2^{(N-1)}$ = 2,147,483,648)

Other Numbers

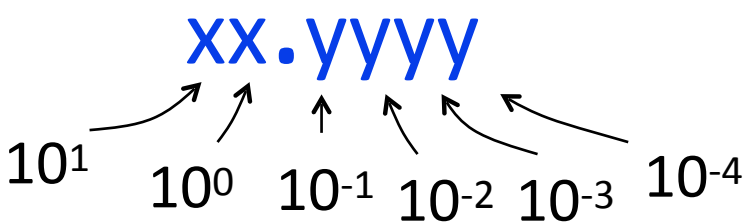
1. Numbers with both integer & fractional parts?
   ex: 1.5
2. Very large numbers? (seconds/millennium)
   ex: $31{,}556{,}926{,}000_{10}$ ($3.1556926_{10} \times 10^{10}$)
3. Very small numbers? (Bohr radius)
   ex: 0.000000000529177_{10}m ($5.29177_{10} \times 10^{-11}$)

- *First consider #1*
- *…our solution will also help with #2 and #3*

Representation of Fractions

- Look at decimal (base 10) first:
- Decimal "point" signifies boundary between integer and fractional parts:

Example 6-digit representation:



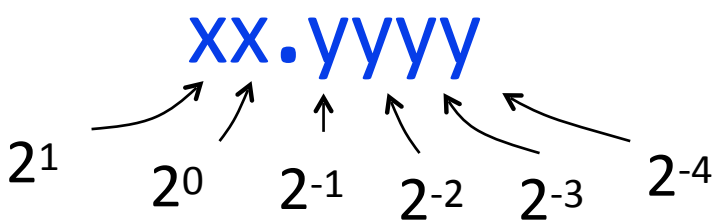
$$25.2406_{ten} = 2x10^{1} + 5x10^{0} + 2x10^{-1} + 4x10^{-2} + 6x10^{-4}$$

If we assume "fixed decimal point", range of 6-digit representations with this format: 0 to 99.9999. Not much range, but lots of "precision"

Binary Representation of Fractions

- "Binary Point" like decimal point signifies boundary between integer and fractional parts:

Example 6-bit representation:



$10.1010_{two} = 1x2^1 + 1x2^{-1} + 1x2^{-3} = 2.625_{ten}$

If we assume "fixed binary point", range of 6-bit representations with this format: 0 to 3.9375 (almost 4)

Fractional Powers of 2

i	2^{-i} (base 2)	(base 10)	(fraction)
0	1.0	1.0	1
1	0.01	0.5	1/2
2	0.001	0.25	1/4
3	0.0001	0.125	1/8
4	0.00001	0.0625	1/16
5	0.000001	0.03125	1/32
6	0.0000001	0.015625	1/64
7	0.00000001	0.0078125	1/128
8	0.000000001	0.00390625	1/256
9	0.0000000001	0.001953125	1/512
10	0.00000000001	0.0009765625	1/1024
11	0.000000000001	0.00048828125	1/2048
12	0.0000000000001	0.000244140625	1/4096

Representation of Fractions with Fixed Point

What about addition and multiplication?

Addition is straightforward:

```
  01.100   1.5ten
+ 00.100   0.5ten
----------------
  10.000   2.0ten
```

Multiplication a bit more complex:

```
   01.100   1.5ten
   00.100   0.5ten
-----------------
   00 000
  000 00
 0110 0
 00000
 00000
-----------------
0000110000
```

Where's the answer, **0.11**? (e.g., 0.5 + 0.25; Need to remember where point is!)

Representation of Fractions

Our examples used a “fixed” binary point.

What we really want is to “float” the binary point to make most effective use of limited bits

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

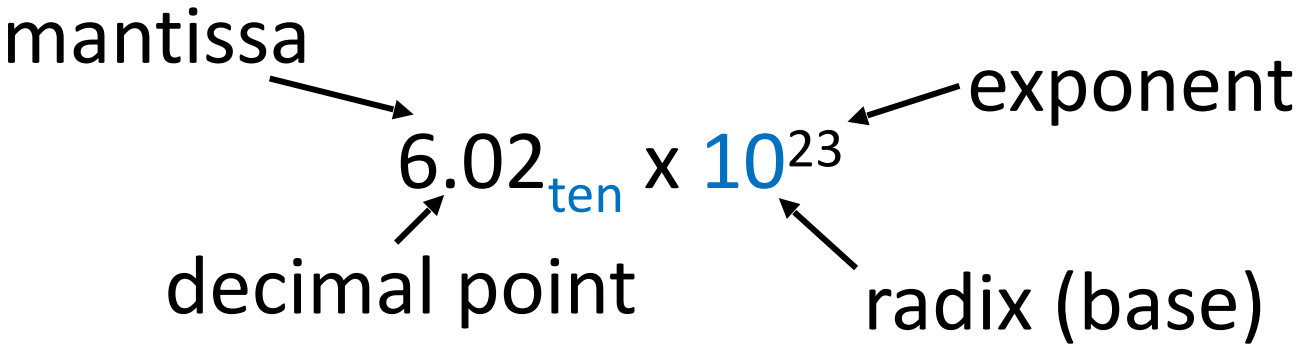
… 000000.001010100000…

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

Any other solution would lose accuracy!

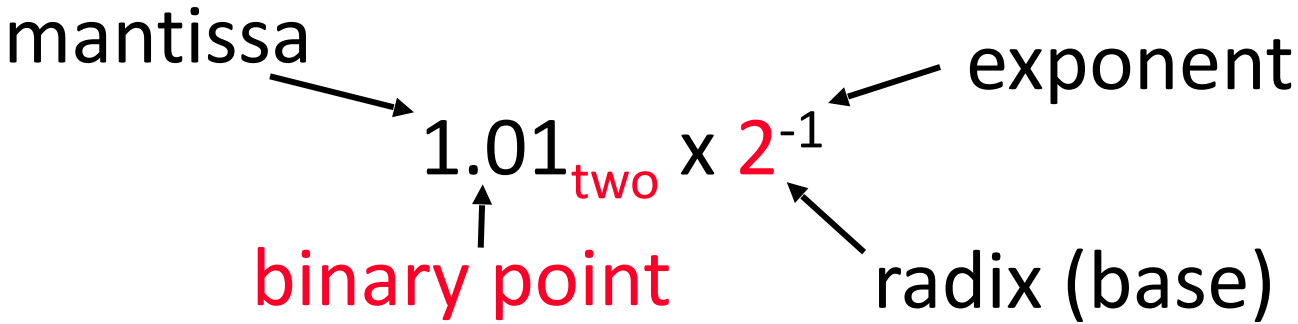
- With floating-point representation, each numeral carries an *exponent* field recording the whereabouts of its binary point
- Binary point can be outside the stored bits, so very large and small numbers can be represented

Scientific Notation (in Decimal)



- Normalized form: no leadings 0s
  (exactly one digit to left of decimal point)
- Alternatives to representing 1/1,000,000,000
  - Normalized: 1.0×10^{-9}
  - Not normalized: 0.1×10^{-8}, 10.0×10^{-10}

Scientific Notation (in Binary)



- Computer arithmetic that supports it is 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

UCB's "Father" of IEEE Floating point

IEEE Standard 754 for Binary Floating-Point Arithmetic.

Prof. Kahan

1989
ACM Turing
Award Winner!

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

Goals for IEEE 754 Floating-Point Standard

- Standard arithmetic for reals for all computers
  - Important because computer representation of real numbers is approximate. Want same results on all computers.
- Keep as much precision as possible
- Help programmer with errors in real arithmetic

  +∞, -∞, Not-A-Number (NaN), exponent overflow, exponent underflow
- Keep encoding that is somewhat compatible with two's complement
  - E.g., 0 in Fl. Pt. is 0 in two's complement
  - Make it possible to sort without needing to do floating-point comparison

Floating-Point Representation

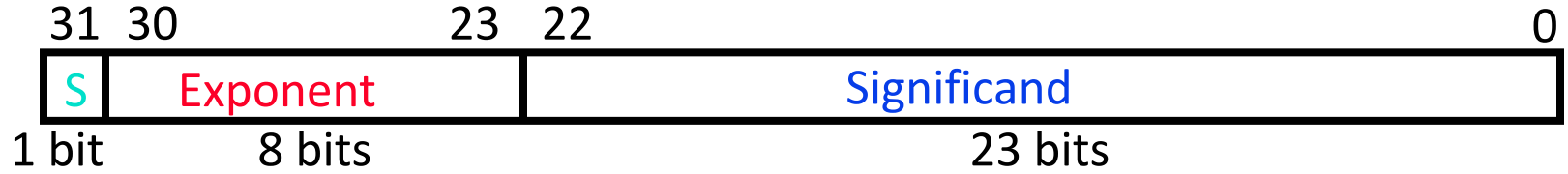
- For “single precision”, a 32-bit word.
- IEEE 754 single precision Floating-Point Standard:
  - 1 bit for *sign (s)* of floating point number
  - 8 bits for *exponent (E)*
  - 23 bits for *fraction (F)*
    (get 1 extra bit of precision because leading 1 is implicit)

$$(-1)^{s} \times (1 + F) \times 2^{E}$$

- Can represent approximately numbers in the range of 2.0×10^{-38} to 2.0×10^{38}

Floating-Point Representation

- Normal format: $(-1)^S * 1.xxx…x * 2^{(yyy…y - 127)}$



  - S represents Sign
    - *1 for negative, 0 for positive*
  - x's represent Fractional part called Significand
    - *implicit leading 1, signed-magnitude (not 2's complement)*
  - y's represent Exponent
    - *in special form called* biased notation *(stored value - 127)*

IEEE 754 Floating-Point Standard

- 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

Bias Notation (exponent = stored value - 127)

How it is interpreted

How it is encoded

∞, NaN

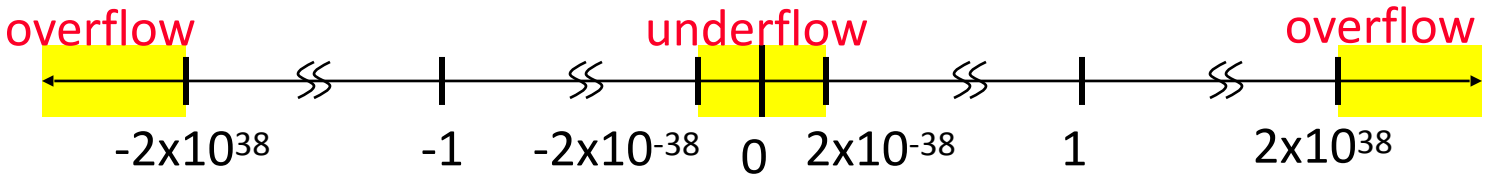
Getting closer to zero

Zero

Decimal Exponent	signed 2's complement	Biased Notation	Decimal Value of Biased Notation
For infinities		11111111	255
127	01111111	11111110	254
...	...	...	...
2	00000010	10000001	129
1	00000001	10000000	128
0	00000000	01111111	127
-1	11111111	01111110	126
-2	11111110	01111101	125
...	...	...	...
-126	10000010	00000001	1
For Denorms	10000001	00000000	0

Floating-Point Representation

- What if result too large?
  (> 2.0×10^{38} , < -2.0×10^{38})
  - Overflow! ⇒ Exponent larger than represented in 8-bit Exponent field
- What if result too small?
  (>0 & < 2.0×10^{-38} , <0 & > -2.0×10^{-38})
  - Underflow! ⇒ Negative exponent larger than represented in 8-bit Exponent field



- What would help reduce chances of overflow and/or underflow?

Floating-Point Representation

- What about bigger or smaller numbers?
- IEEE 754 Floating-Point *Double Precision Standard* (64 bits)
  - 1 bit for *sign (s)* of floating-point number
  - 11 bits for *exponent (E)*
  - 52 bits for *fraction (F)*
    (get 1 extra bit of precision if leading 1 is implicit)

$(-1)^s \times (1 + F) \times 2^E$

- Can represent from 2.0×10^{-308} to 2.0×10^{308}
- More importantly, 53 bits of precision!
- Recall, 32-bit format called *Single Precision*

Peer Instruction

- What's the value of this Floating Point number:

1 1000 0000 1000 0000 0000 0000 0000 000

- A: -1×2^{128}
- B: $+1 \times 2^{-128}$
- C: -1×2^{1}
- D: -1.5×2^{1}

Break-time entertainment:
Which is Less? (i.e., closer to -∞)

- **0** vs. 1×10^{-127} ?
- 1×10^{-126} vs. **1×10^{-127}** ?
- **-1×10^{-127}** vs. 0 ?
- **-1×10^{-126}** vs. -1×10^{-127} ?

More Floating Point: Preview

- What about 0?
  - Bit pattern all 0s means 0 (so no implicit leading 1 in this case)
- What if divide 1 by 0?
  - Can get infinity symbols +∞, -∞
  - Sign bit 0 or 1, largest exponent (all 1s), 0 in fraction
- What if do something stupid? (∞ – ∞, 0 ÷ 0)
  - Can get special symbols NaN for “Not-a-Number”
  - Sign bit 0 or 1, largest exponent (all 1s), not zero in fraction
- What if result is too big?
  - Get *overflow* in exponent, alert programmer!
- What if result is too small?
  - Get *underflow* in exponent, alert programmer!

Representation for 0

- Represent 0?
  - Exponent all zeroes
  - Significand all zeroes
  - What about sign? Both cases valid!

```
+0: 0 00000000 00000000000000000000000
-0: 1 00000000 00000000000000000000000
```

Representation for ± ∞

- In FP, divide by 0 should produce ± ∞, not overflow
- Why?
  - OK to do further computations with ∞
    E.g., X/0 > Y may be a valid comparison
- IEEE 754 represents ± ∞
  - Most positive exponent reserved for ∞
  - Significand all zeroes

Special Numbers

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

Exponent	Significand	Object
0	0	0
0	nonzero	???
1-254	anything	+/- fl. pt. #
255	0	+/- ∞
255	nonzero	???

Representation for Not-a-Number

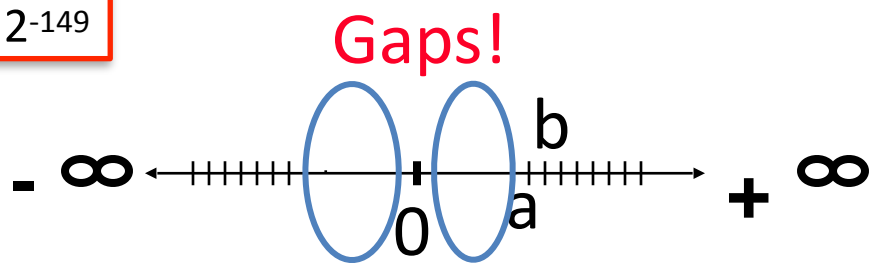
- What do I get if I calculate `sqrt(-4.0)` or `0/0`?
  - If ∞ not an error, these shouldn't be either
  - Called Not a Number (NaN)
  - Exponent = 255, Significand nonzero
- Why is this useful?
  - Hope NaNs help with debugging?
  - They contaminate: op(NaN, X) = NaN
  - Can use the significand to identify which! (e.g., quiet NaNs and signaling NaNs)

Representation for Denorms (1/2)

- Problem: There's a gap among representable FP numbers around 0
  - Smallest representable positive number:
    $a = 1.0\ldots_2 * 2^{-126} = 2^{-126}$
  - Second smallest representable positive number:
    $b = 1.000\ldots\ldots1_2 * 2^{-126}$
    $= (1 + 0.00\ldots1_2) * 2^{-126}$
    $= (1 + 2^{-23}) * 2^{-126}$
    $= 2^{-126} + 2^{-149}$

$a - 0 = 2^{-126}$
$b - a = 2^{-149}$

Normalization and implicit 1 are to blame!

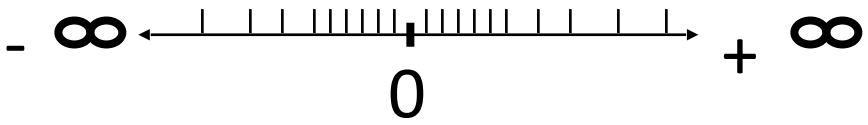


Representation for Denorms (2/2)

- Solution:
  - We still haven't used Exponent = 0, Significand nonzero
  - DEnormalized number: no (implied) leading 1, **implicit exponent = -126**
  - Smallest representable positive number:

    $a = 2^{-149}$ (i.e., $2^{-126} \times 2^{-23}$)
  - Second-smallest representable positive number:

    $b = 2^{-148}$ (i.e., $2^{-126} \times 2^{-22}$)



Special Numbers Summary

- Reserve exponents, significands:

Exponent	Significand	Object
0	0	0
0	nonzero	Denorm
1-254	anything	+/- fl. pt. number
255	0	+/- ∞
255	Nonzero	NaN

Saving Bits

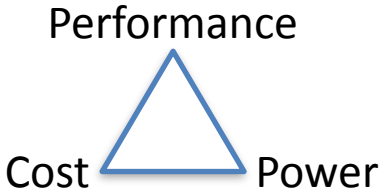
- Many applications in machine learning, graphics, signal processing can make do with lower precision
- IEEE “half-precision” or “FP16” uses 16 bits of storage
  - 1 sign bit
  - 5 exponent bits (exponent bias of 15)
  - 10 significand bits
- Microsoft “BrainWave” FPGA neural net computer uses proprietary 8-bit and 9-bit floating-point formats

Outline

- Floating Point Standard
- Defining Performance
- And in Conclusion …

What's important to measure about computer systems?

- As a ***consumer*** and user of computer systems, what's important to you?
  - *cost, performance, power consumption*
  - reliability, software infrastructure (OS, etc.)
- As a ***designer*** of computer systems, what's important to you?
  - *cost, performance, power consumption*
  - reliability, software infrastructure, time to market
- Achieving all of low-cost, high-performance, and low-power consumption is not possible:
  - Ex: Very high-performance computers are high-cost and high-power (supercomputers), Low-cost computers are low-performance (Arduino, RP), Low-power computers are low-performance (few watts versus hundreds of watts).



These characteristics "tradeoff" with one another. Improving on one usually comes at the expense of the others.

Review: 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

Server Delay (ms)	Increased time to next click (ms)	Queries/ user	Any clicks/ user	User satisfac- tion	Revenue/ User
50	--	--	--	--	--
200	500	--	-0.3%	-0.4%	--
500	1200	--	-1.0%	-0.9%	-1.2%
1000	1900	0.7%	1.9%	1.6%	2.8%
2000	3100	-1.8%	-4.4%	-3.8%	-4.3%

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

Review of CPU Latency Performance Equation

- $$\text{Time} = \frac{\text{Seconds}}{\text{Program}}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}}$$

Total number of instructions *executed* to run program of interest.

Average number of cycles per instruction

System clock period (1/F)

What Affects Each Component? Instruction Count, CPI, Clock Rate

Hardware or software?	Affects What?
Algorithm	Instruction Count, CPI
Programming Language	Instruction Count, CPI
Compiler	Instruction Count, CPI
Instruction Set Architecture	Instruction Count, CPI, Clock Rate
Micro-architecture	CPI, Clock Rate

Why is the CPI not determined only by the instruction set architecture and the micro-architecture?

Peer Instruction

Computer	Clock frequency	Clock cycles per instruction	#instructions per program
A	1.0 GHz	2	1000
B	2.0 GHz	5	800
C	0.5 GHz	1.25	400
D	5.0 GHz	10	2000

- Which computer has the highest performance for a given program?

Administrivia

- Midterm next Tuesday
  - Review Sunday 1-4 100 Genetics & Plant Biology
- Guerrilla Session tonight 7-9 Barrows 20 - *Caches*

Measuring Processor Performance with Standard Benchmarks

- The best way to measure computer system performance is to measure using the set of programs of interest to you, or a set that is representative of your typical “workload”.
- Performing accurate measurements is tricky and your future needs might be unknown. Most users relay on *standard benchmarks* for comparisons.
- Computer designers (and manufacturers) rely on *standard benchmarks* for understanding the effect of design decisions on performance.
- Popular Benchmark Suites:
  - SPEC - Standard Performance Evaluation Corporation (https://www.spec.org/benchmarks.html)
  - EEMBC - Industry-Standard Benchmarks for Embedded Systems (https://www.eembc.org)
  - Many more used in industry and academia …

SPEC
(Standard Performance Evaluation Corporation)

A non-profit organization founded in 1988 to establish standardized performance benchmarks that are objective, meaningful, clearly defined, and readily available.

- SPEC members include hardware and software vendors, universities, and researchers.
- SPEC was founded on the realization that "An ounce of honest data is worth a pound of marketing hype".

- SPEC CPU benchmarks focus on compute intensive performance:
  - Processor - The CPU chip(s).
  - Memory - The memory hierarchy, including caches and main memory.
  - Compilers - C, C++, and Fortran compilers, including optimizers.
- SPEC CPU2006
  - 12 Integer Programs
  - 17 Floating-Point Programs
- Often turn into number where bigger is faster
- *SPECratio*: reference execution time on old reference computer divided by execution time on new computer, reported as speed-up

SPECINT2006 on AMD Barcelona

Description	Instruction Count (B)	CPI	Clock cycle time (ps)	Execution Time (s)	Reference Time (s)	SPEC-ratio
Interpreted string processing	2,118	0.75	400	637	9,770	15.3
Block-sorting compression	2,389	0.85	400	817	9,650	11.8
GNU C compiler	1,050	1.72	400	724	8,050	11.1
Combinatorial optimization	336	10.0	400	1,345	9,120	6.8
Go game	1,658	1.09	400	721	10,490	14.6
Search gene sequence	2,783	0.80	400	890	9,330	10.5
Chess game	2,176	0.96	400	837	12,100	14.5
Quantum computer simulation	1,623	1.61	400	1,047	20,720	19.8
Video compression	3,102	0.80	400	993	22,130	22.3
Discrete event simulation library	587	2.94	400	690	6,250	9.1
Games/path finding	1,082	1.79	400	773	7,020	9.1
XML parsing	1,058	2.70	400	1,143	6,900	6.0

Summarizing SPEC Performance

- Varies from 6x to 22x faster than reference computer
- *Geometric mean* of ratios:
  N-th root of product
  of N ratios

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

  - Geometric Mean, When comparing any two measured systems, their relative performance to each other remains the same no matter what computer is used as reference
- Geometric Mean for Barcelona is 11.7

SPEC 2017 - Execution-time and Throughput

Short Tag	Suite	Contents	Metrics	How many copies? What do Higher Scores Mean?
intspeed	SPECspeed 2017 Integer	10 integer benchmarks	SPECspeed2017_int_base SPECspeed2017_int_peak	SPECspeed suites always run one copy of each benchmark. Higher scores indicate that less time is needed.
fpspeed	SPECspeed 2017 Floating Point	10 floating point benchmarks	SPECspeed2017_fp_base SPECspeed2017_fp_peak	
intrate	SPECrate 2017 Integer	10 integer benchmarks	SPECrate2017_int_base SPECrate2017_int_peak	SPECrate suites run multiple concurrent copies of each benchmark. The tester selects how many. Higher scores indicate more *throughput* (work per unit of time).
fprate	SPECrate 2017 Floating Point	13 floating point benchmarks	SPECrate2017_fp_base SPECrate2017_fp_peak	

- The reference machine is a historical Sun Microsystems server, the Sun Fire V490 with 2100 MHz UltraSPARC-IV+ chips. The UltraSPARC-IV+ was introduced in 2006, and is newer than the chip used in the CPU2000 and CPU2006 reference machines (the 300 MHz 1997 UltraSPARC II).

SPEC CPU2017 Integer Suite

SPECrate 2017 Integer	SPECspeed 2017 Integer	Language [1]	KLOC [2]	Application Area
500.perlbench_r	600.perlbench_s	C	362	Perl interpreter
502.gcc_r	602.gcc_s	C	1,304	GNU C compiler
505.mcf_r	605.mcf_s	C	3	Route planning
520.omnetpp_r	620.omnetpp_s	C++	134	Discrete Event simulation - computer network
523.xalancbmk_r	623.xalancbmk_s	C++	520	XML to HTML conversion via XSLT
525.x264_r	625.x264_s	C	96	Video compression
531.deepsjeng_r	631.deepsjeng_s	C++	10	Artificial Intelligence: alpha-beta tree search (Chess)
541.leela_r	641.leela_s	C++	21	Artificial Intelligence: Monte Carlo tree search (Go)
548.exchange2_r	648.exchange2_s	Fortran	1	Artificial Intelligence: recursive solution generator (Sudoku)
557.xz_r	657.xz_s	C	33	General data compression

SPEC CPU2017 Floating Point Suite

SPECrate 2017 Floating Point	SPECspeed 2017 Floating Point	Language [1]	KLOC [2]	Application Area
503.bwaves_r	603.bwaves_s	Fortran	1	Explosion modeling
507.cactuBSSN_r	607.cactuBSSN_s	C++, C, Fortran	257	Physics: relativity
508.namd_r		C++	8	Molecular dynamics
510.parest_r		C++	427	Biomedical imaging: optical tomography with finite elements
511.povray_r		C++, C	170	Ray tracing
519.lbm_r	619.lbm_s	C	1	Fluid dynamics
521.wrf_r	621.wrf_s	Fortran, C	991	Weather forecasting
526.blender_r		C++, C	1,577	3D rendering and animation
527.cam4_r	627.cam4_s	Fortran, C	407	Atmosphere modeling
	628.pop2_s	Fortran, C	338	Wide-scale ocean modeling (climate level)
538.imagick_r	638.imagick_s	C	259	Image manipulation
544.nab_r	644.nab_s	C	24	Molecular dynamics
549.fotonik3d_r	649.fotonik3d_s	Fortran	14	Computational Electromagnetics
554.roms_r	654.roms_s	Fortran	210	Regional ocean modeling

Outline

- Floating Point Standard
- Defining Performance
- And in Conclusion …

And In Conclusion, …

- Time (seconds/program) is measure of performance

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}}$$

- Floating-point representations hold approximations of real numbers in a finite number of bits
  - 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)
  - Single Precision: