CS 110
Computer Architecture
Performance and Floating Point Arithmetic

Instructor:
Sören Schwertfeger

http://shtech.org/courses/ca/

School of Information Science and Technology SIST

ShanghaiTech University

Slides based on UC Berkley's CS61C

CPI/Miss Rates/DRAM Access SpecInt2006

Name	CPI	Data Only L1 D cache misses/1000 instr	Data Only L2 D cache misses/1000 instr	Instructions and Data DRAM accesses/1000 instr
perl	0.75	3.5	1.1	1.3
bzip2	0.85	11.0	5.8	2.5
gcc	1.72	24.3	13.4	14.8
mcf	10.00	106.8	88.0	88.5
go	1.09	4.5	1.4	1.7
hmmer	0.80	4.4	2.5	0.6
sjeng	0.96	1.9	0.6	0.8
libquantum	1.61	33.0	33.1	47.7
h264avc	0.80	8.8	1.6	0.2
omnetpp	2.94	30.9	27.7	29.8
astar	1.79	16.3	9.2	8.2
xalancbmk	2.70	38.0	15.8	11.4
Median	1.35	13.6	7.5	5.4

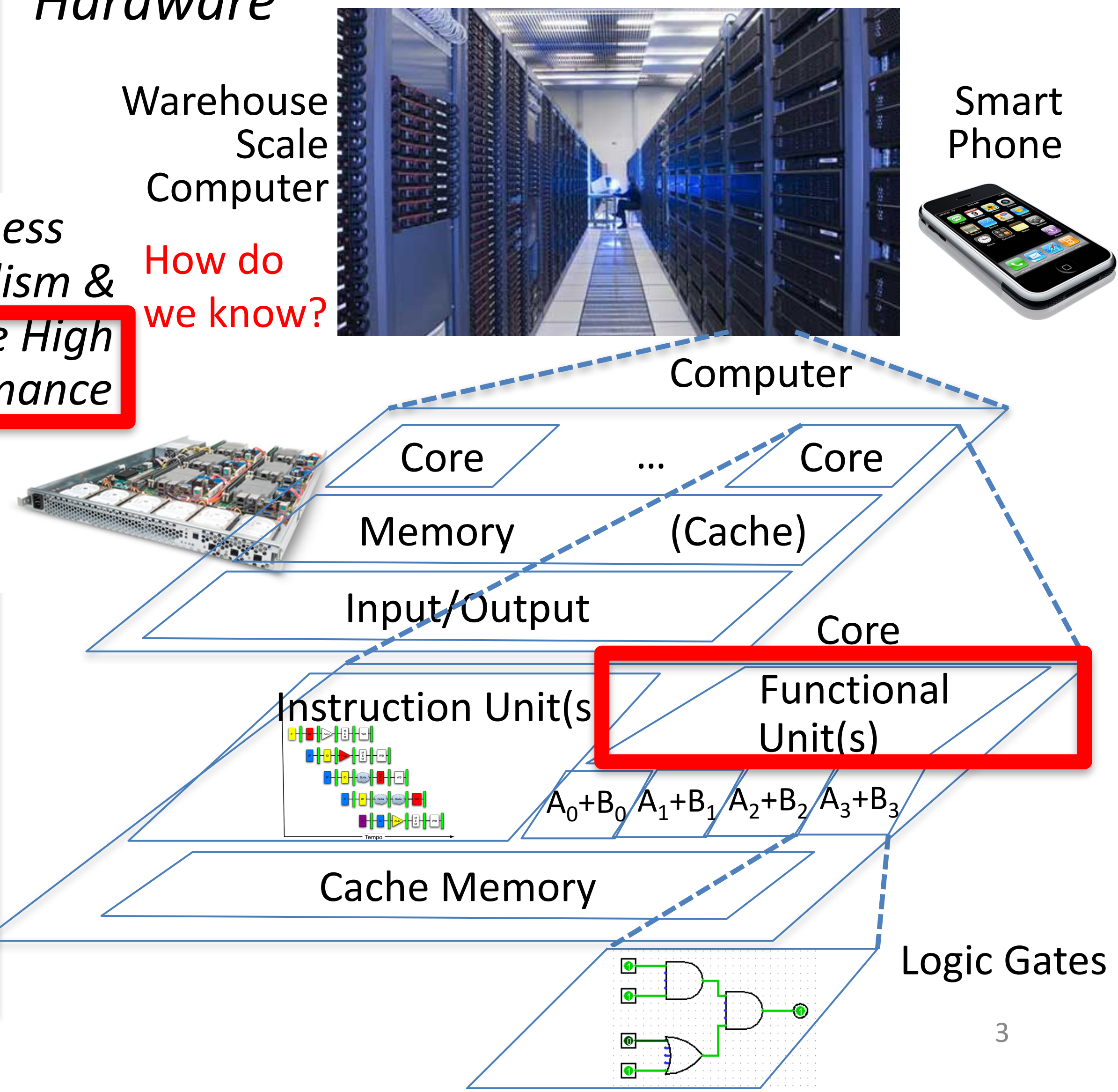
New-School Machine Structures (It’s a bit more complicated!)

Software

- 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

Harness Parallelism & Achieve High Performance

Hardware



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

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

- $\text{Performance}_X = 1/\text{Program Execution Time}_X$
- $\text{Performance}_X > \text{Performance}_Y \Rightarrow$
  $1/\text{Execution Time}_X > 1/\text{Execution Time}_y \Rightarrow$
  $\text{Execution Time}_Y > \text{Execution Time}_X$
- Computer X is N times faster than Computer Y
  $\text{Performance}_X / \text{Performance}_Y = N$ or
  $\text{Execution Time}_Y / \text{Execution Time}_X = N$

- 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 = $1/(3\text{x}10^9)$ 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

- A program executes instructions
- `CPU Time/Program`

```
  = Clock Cycles/Program x Clock Cycle Time
  = Instructions/Program
    x Average Clock Cycles/Instruction
    x Clock Cycle Time
```

- 1st term called *Instruction Count*
- 2nd term abbreviated *CPI* for average
  ***C**lock Cycles **P**er **I**nstruction*
- 3rd term is 1 / Clock rate

Restating Performance Equation

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

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

What Affects Each Component? A)Instruction Count, B)CPI, C)Clock Rate

	Affects What?
Algorithm	
Programming Language	
Compiler	
Instruction Set Architecture	

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

	Affects What?
Algorithm	Instruction Count, CPI
Programming Language	Instruction Count, CPI
Compiler	Instruction Count, CPI
Instruction Set Architecture	Instruction Count, Clock Rate, CPI

Question

Computer	Clock frequency	Clock cycles per instruction	#instructions per program	
A	1GHz	2	1000	
B	2GHz	5	800	
C	500MHz	1.25	400	
D	5GHz	10	2000	

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

Question

Computer	Clock frequency	Clock cycles per instruction	#instructions per program	Calculation
A	1GHz	2	1000	1ns * 2 * 1000 = 2μs
B	2GHz	5	800	0.5ns 5 *800 = 2μs
C	500MHz	1.25	400	2ns 1.25 * 400 = 1μs
D	5GHz	10	2000	0.2ns * 10 * 2000 = 4μs

- 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

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 Performance …

System	Rate (Task 1)	Rate (Task 2)
A	10	20
B	20	10

Clickers: Which system is faster?

A: System A

B: System B

C: Same performance

D: Unanswerable question!

… Depends Who's Selling

System	Rate (Task 1)	Rate (Task 2)	Average
A	10	20	15
B	20	10	15

Average throughput

System	Rate (Task 1)	Rate (Task 2)	Average
A	0.50	2.00	1.25
B	1.00	1.00	1.00

Throughput relative to B

System	Rate (Task 1)	Rate (Task 2)	Average
A	1.00	1.00	1.00
B	2.00	0.50	1.25

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

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

  - Geometric Mean gives same relative answer no matter what computer is used as reference
- Geometric Mean for Barcelona is 11.7

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

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

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

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 <u>both</u> 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.yyyy

2^1 2^0 2^{-1} 2^{-2} 2^{-3} 2^{-4}

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

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}	
0	1.0	1
1	0.5	1/2
2	0.25	1/4
3	0.125	1/8
4	0.0625	1/16
5	0.03125	1/32
6	0.015625	
7	0.0078125	
8	0.00390625	
9	0.001953125	
10	0.0009765625	
11	0.00048828125	
12	0.000244140625	
13	0.0001220703125	
14	0.00006103515625	
15	0.000030517578125	

Representation of Fractions with Fixed Pt.

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`? (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_{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 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 **exponent**

$$6.02_{ten} \times 10^{23}$$

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: 1.0×10^{-9}
  - Not normalized: $0.1 \times 10^{-8}, 10.0 \times 10^{-10}$

Scientific Notation (in Binary)

mantissa **exponent**

$$1.01_{two} \times 2^{-1}$$

"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: $+1.xxx...x_{two} * 2^{yyy...y_{two}}$**
- **Multiple of Word Size (32 bits)**

31	30 … 23	22 … 0
S	**Exponent**	**Significand**
1 bit	8 bits	23 bits

- **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?

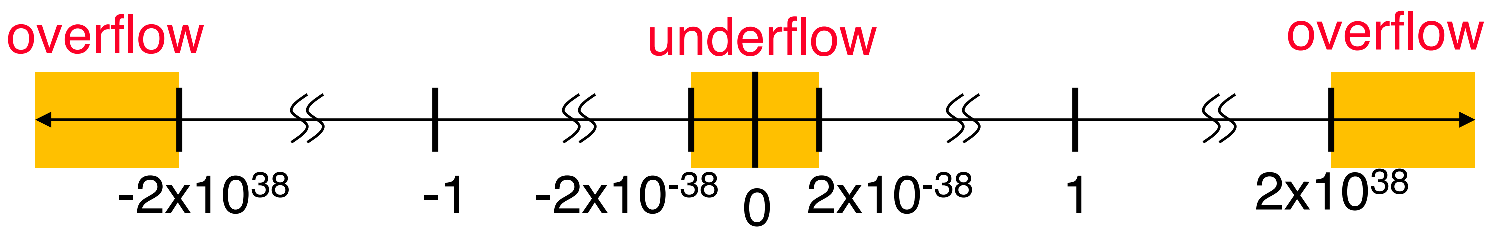
  (> 2.0×10^{38} , < -2.0×10^{38})

  – **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?

IEEE 754 Floating-Point Standard (1/3)

Single Precision (Double Precision similar):

31	30 … 23	22 … 0
S	**Exponent**	**Significand**
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: $0 < \text{Significand} < 1$ (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):**

31	30 … 23	22 … 0
S	**Exponent**	**Significand**
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)

Question

- Guess this Floating Point number:

1 1000 0000 1000 0000 0000 0000 0000 000

A: $-1\text{x } 2^{128}$

B: $+1\text{x } 2^{-128}$

C: $-1\text{x } 2^{1}$

D: $+1.5\text{x } 2^{-1}$

E: $-1.5\text{x } 2^{1}$

Representation for $\pm \infty$

- In FP, divide by 0 should produce $\pm \infty$, not overflow.
- Why?
  - OK to do further computations with ∞
    E.g., X/0 > Y may be a valid comparison

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

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
```

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	???

- **Clever idea:**
  - Use exp=0,255 & Sig!=0

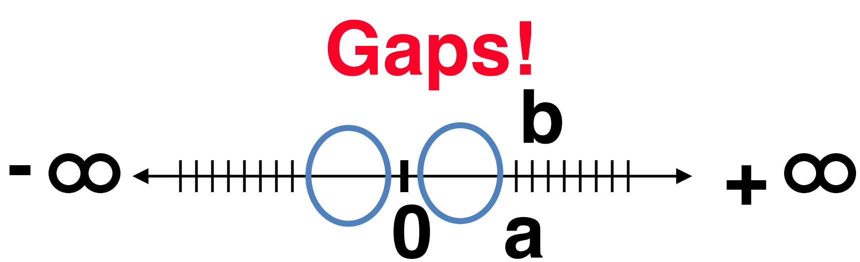
Representation for Not a Number

- What do I get if I calculate
  `sqrt(-4.0)` or `0/0`?
  - If ∞ 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!

Representation for Denorms (1/2)

- Problem: There’s a gap among representable FP numbers around 0
  - Smallest representable pos num:
    - a = 1.0… 2 * 2-126 = 2-126
  - Second smallest representable pos num:
    - b = 1.000……1 2 * 2-126
      = (1 + 0.00…12) * 2-126
      = (1 + 2-23) * 2-126
      = 2-126 + 2-149
  - a - 0 = 2-126
  - b - a = 2-149

Normalization and implicit 1 is to blame!

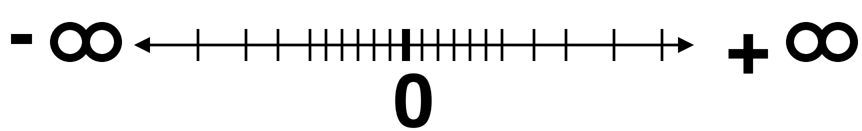


Representation for Denorms (2/2)

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

    $a = 2^{-149}$
  - **Second smallest representable pos num:**

    $b = 2^{-148}$



Special Numbers Summary

- **Reserve exponents, significands:**

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

Conclusion

Exponent tells Significand how much (2^i) to count by (…, 1/4, 1/2, 1, 2, …)

- 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 #s.

Can store NaN, ± ∞

- 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):**

31	30 … 23	22 … 0
S	Exponent	Significand
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)

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