CS 211: Intro to Computer Architecture

4.1: Floating Point and C Data Representations

Minesh Patel

Spring 2025 – Tuesday 11 February

Announcements

- **Assigned**
  - **WA1 last Saturday** (Gradescope, due Sunday 23:59)
  - **PA2 tonight** (if all goes well…)
- **Planned vs. Actual Course Schedule**
  - We have an ambitious plan, but….
  - Will shift topics backward as needed

Recap: Fraction Base Conversions

$(11)_{10} \leftarrow (11.1875)_{10} \rightarrow (0.1875)_{10}$

Divide By 2

$\frac{11}{2} = 5$, $11 \% 2 = 1$

$(11)_{10} = (5)_{10} \times 2^1 + (1)_{10} \times 2^0$

Least Significant Bit (LSB)

$\frac{5}{2} = 2$, $5 \% 2 = 1$

$(5)_{10} \times 2^1 = (2)_{10} \times 2^2 + (1)_{10} \times 2^1$

$\frac{2}{2} = 1$, $2 \% 2 = 0$

$(2)_{10} \times 2^2 = (1)_{10} \times 2^3 + (0)_{10} \times 2^2$

$\frac{1}{2} = 0$, $1 \% 2 = 1$

$(1)_{10} \times 2^3 = (0)_{10} \times 2^4 + (1)_{10} \times 2^3$

Most Significant Bit (MSB)

Multiply By 2

$0.1875 \times 2 = 0.375$

Most Significant Bit (MSB)

$(0.1875)_{10} = (0 + 0.375)_{10} \times 2^{-1}$

$0.375 \times 2 = 0.75$

$(0.375)_{10} \times 2^{-1} = (0 + 0.75)_{10} \times 2^{-2}$

$0.75 \times 2 = 1.5$

$(0.75)_{10} \times 2^{-2} = (1 + 0.5)_{10} \times 2^{-3}$

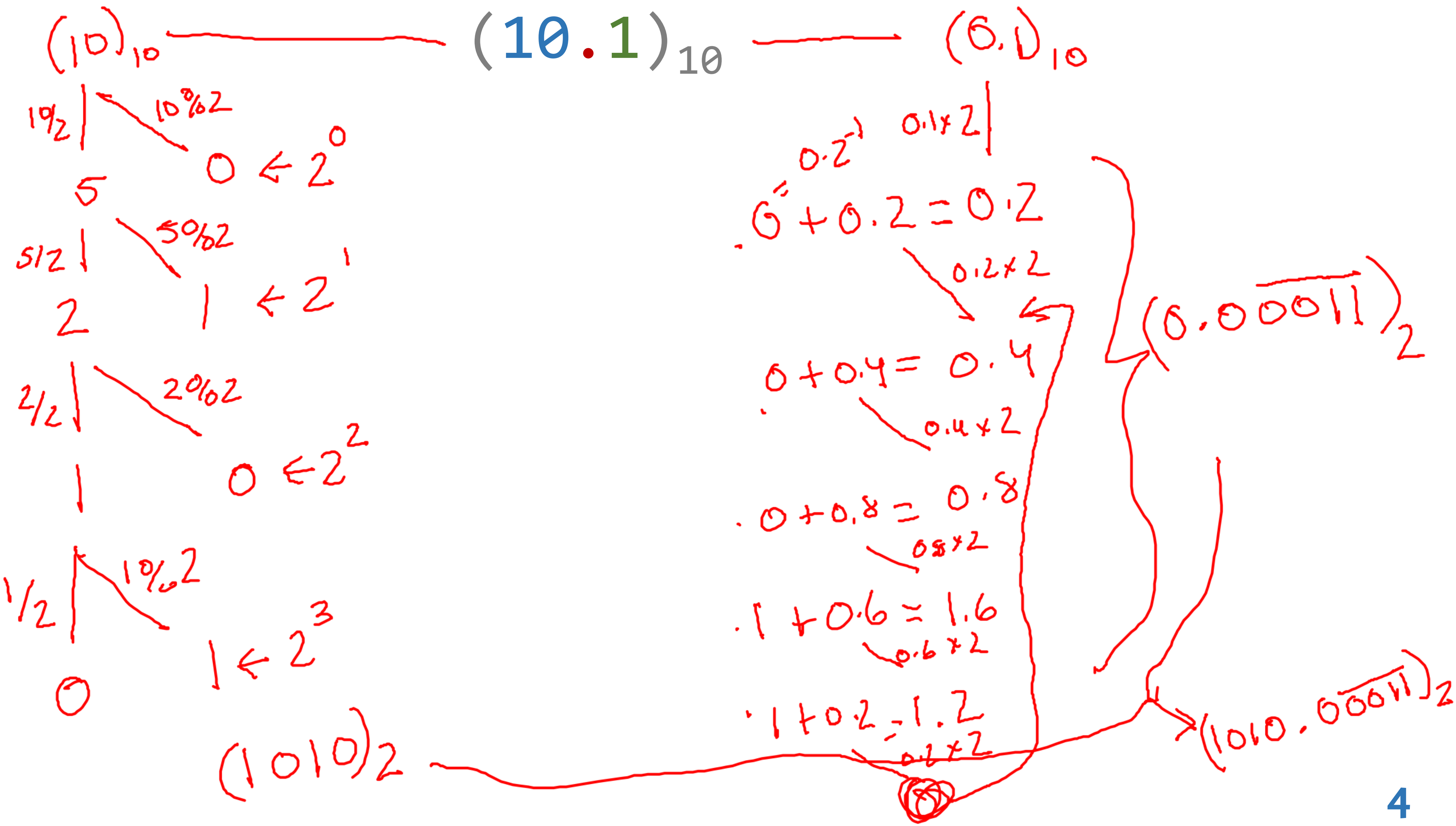
$0.5 \times 2 = 1.0$

$(0.5)_{10} \times 2^{-3} = (1 + 0)_{10} \times 2^{-4}$

Least Significant Bit (LSB)

Recap: Fraction Base Conversions (again)

$(10.1)_{10}$

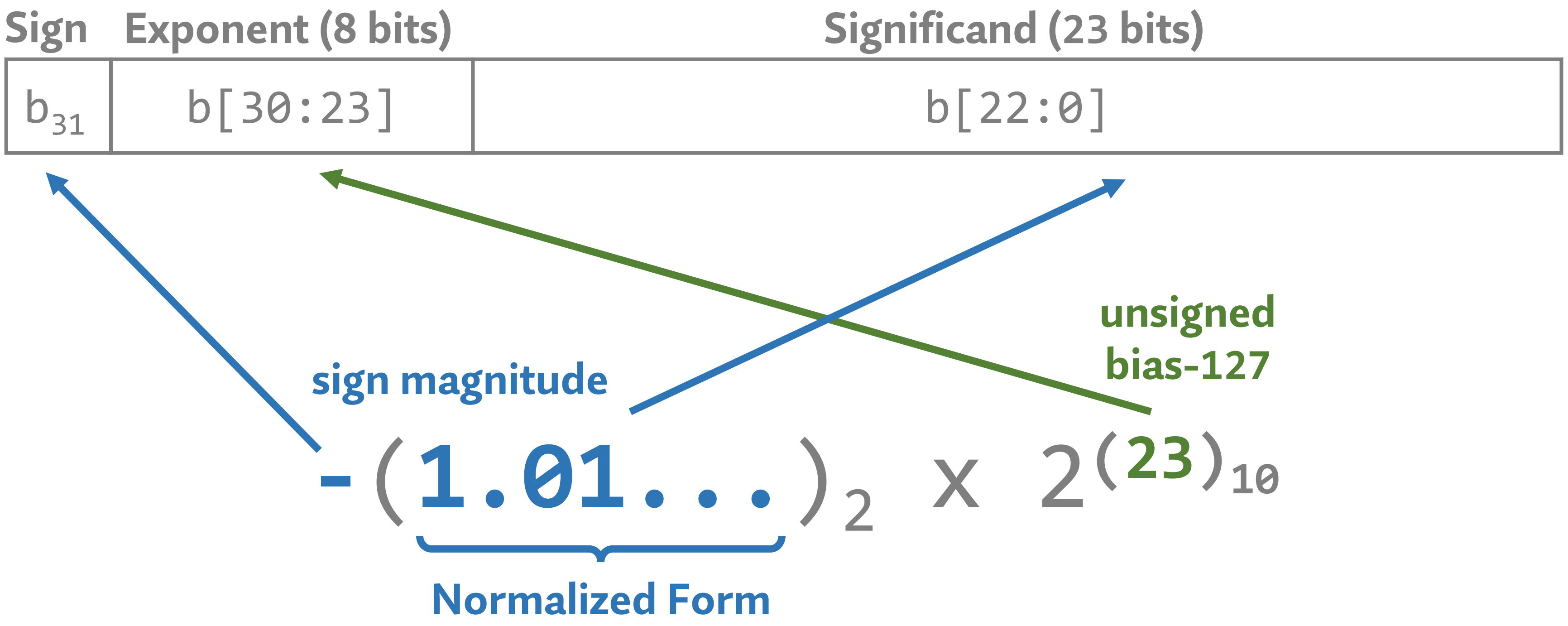


Agenda

- **Floating Point**
  - Special Values
- Operations on Representations
  - C vs Java
  - “Hello World” Analysis
  - Compiling C Code

Recap: Floating-Point Representations

IEEE 754 “Single Precision” Floating Point



Floating Point Example

1000 1100 0011

Representation (Big Endian): `0x8c30_0000`

bias - 127

Sign	Exponent (8 bits)	Significand (23 bits)
1	00011000	011 →

neg.

$(11000)_2 \rightarrow (16+8)_{10}$

$(24)_{10}$

$2^{24-127} = 2^{-103}$

$(1.011)_2$

$(1)_2 = (1)_{10}$

$(0.011)_2 = (2^{-2} + 2^{-3})_{10}$

$= (0.25 + 0.125)_{10}$

$= (0.375)_{10}$

$-(1.375)_{10} \times 2^{(-103)_{10}}$

$-(1.011)_2 \times 2^{(-103)_{10}} = (10.11)_2 \times 2^{-104}$

Floating Point Example

Representation (Big Endian): `0x8c30_0000`

Sign	Exponent (8 bits)	Significand (23 bits)
1	0001_1000	011_0000_0000_0000_0000_0000

Unbiased exponent = $(11000)_2 - (127)_{10} = (-103)_{10}$

Significand = $(.011)_2$

Sign = -

$$-(1.011)_2 \times 2^{-103} = -(1.375)_{10} \times 2^{-103}$$

More Examples

$$(0.1)_{10} = (1.\overline{1001})_2 \times 2^{-4}$$

biased exponent: -4 + 127

0.1 to base 2

NATURAL LANGUAGE | MATH INPUT

Input interpretation

convert 0.1 to base 2

Result

$0.00011001100110011\ldots_2$

0.1 to float

NATURAL LANGUAGE | MATH INPUT

Input interpretation

convert 0.1_{10} to IEEE single–precision number

Result

cdcccc3d

Binary representation

sign digit	0
exponent	01111011
significand	10011001100110011001101

$$(0.2)_{10} = (1.\overline{1001})_2 \times 2^{-3}$$

biased exponent: -3 + 127

0.2 to base 2

NATURAL LANGUAGE | MATH INPUT

Input interpretation

convert 0.2 to base 2

Result

$0.0011001100110011\ldots_2$

0.2 to float

NATURAL LANGUAGE | MATH INPUT

Input interpretation

convert 0.2_{10} to IEEE single–precision number

Result

cdcc4c3e

Binary representation

sign digit	0
exponent	01111100
significand	10011001100110011001101

Floating Point Range (Normalized)

8 bits = [0x00 – 0xff]

valid range → [0x01 – 0xfe]

Smallest Normalized Value

Sign	Exponent (8 bits)	Significand (23 bits)
b_{31}	0x01	0000....000

Exponent: 0000_0001 = $2^{(-126)_{10}}$

Significand: 1.?

$= \pm (1.0)_2 \times 2^{(-126)_{10}}$

Largest Normalized Value

Sign	Exponent (8 bits)	Significand (23 bits)
b_{31}	0xfe	111111...111

Exponent: 1111_1110 = $2^{254-127} = 2^{+127}$

Significand: 1.?

$\pm (1.11...11)_2 \times 2^{127}$

Floating Point Range (Normalized)

- Symmetric range around `0`
  - Makes sense to talk about **smallest/largest** representable values instead

Smallest Normalized Value

$$(1.000\_0000\_0000\_0000\_0000\_0000)_2 \times 2^{-126} \approx (1.18)_{10} \times 10^{-38}$$

Sign	Exponent (8 bits)	Significand (23 bits)
b_{31}	0000_0001	000_0000_0000_0000_0000_0000

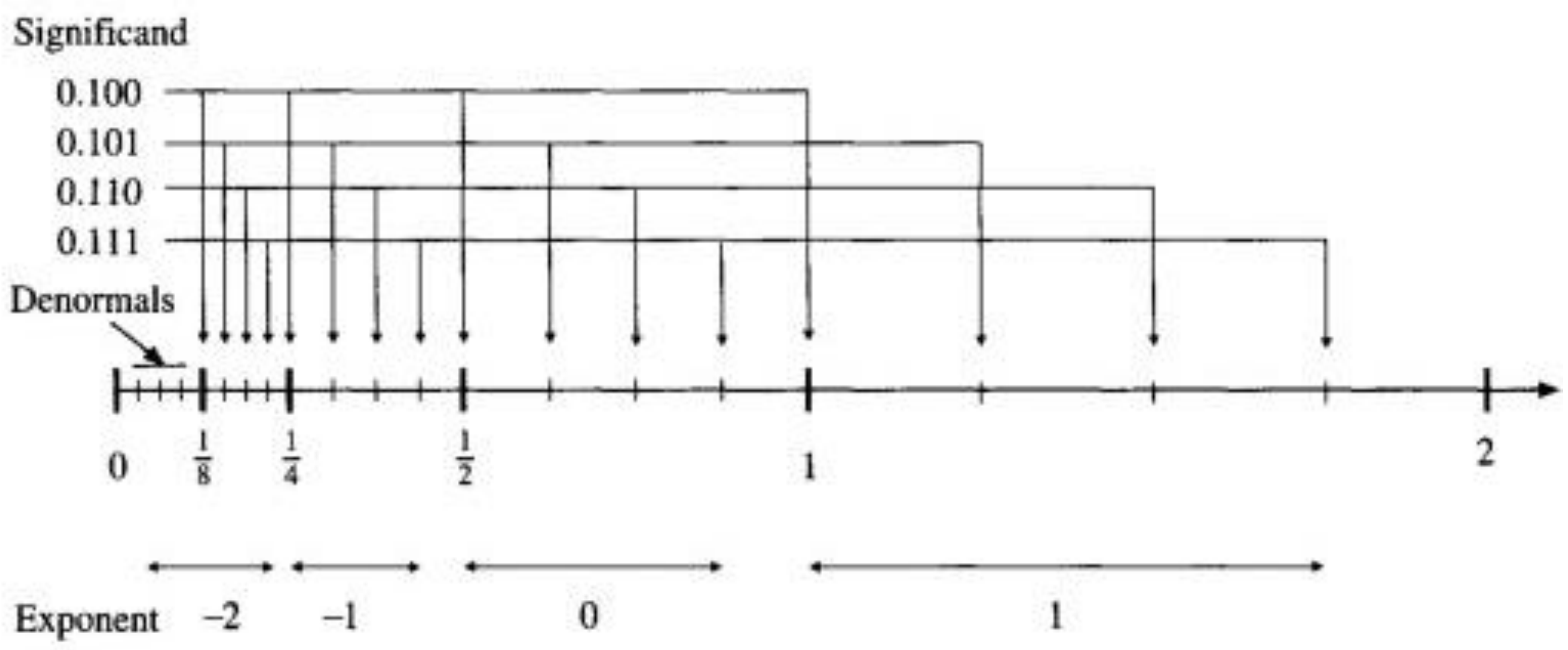
Largest Normalized Value

$$(1.111\_1111\_1111\_1111\_1111\_1111)_2 \times 2^{127} \approx (3.40)_{10} \times 10^{38}$$

Sign	Exponent (8 bits)	Significand (23 bits)
b_{31}	1111_1110	111_1111_1111_1111_1111_1111

Floating Point Value Distribution

- $2^{N=32}$ values are **nonuniformly distributed**



Milos Ercegovac, Tomas Lang, "Digital Arithmetic", Morgan Kaufman, 2004

Other Floating Point Formats

- IEEE Standard 754 specifies other types of float representations

Type	Bits				Exponent bias	Bits precision	Number of decimal digits
	Sign	Exponent	Significand	Total			
Half (IEEE 754-2008)	1	5	10	16	15	11	~3.3
Single	1	8	23	32	127	24	~7.2
Double	1	11	52	64	1023	53	~15.9
x86 extended precision	1	15	64	80	16383	64	~19.2
Quad	1	15	112	128	16383	113	~34.0

- Many other formats exist out in the wild

Type	Sign	Exponent	Trailing significand field	Total bits
FP8 (E4M3)	1	4	3	8
FP8 (E5M2)	1	5	2	8
Half-precision	1	5	10	16
Bfloat16	1	8	7	16
TensorFloat-32	1	8	10	19
Single-precision	1	8	23	32

https://en.wikipedia.org/wiki/Floating-point_arithmetic

NVIDIA H100 Number Formats

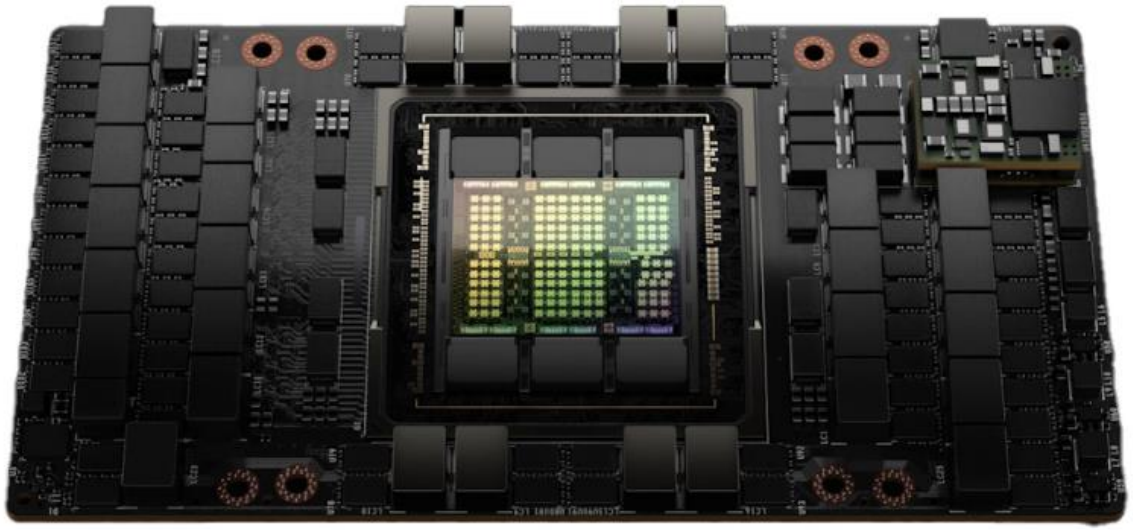


Figure 2 NVIDIA H100 GPU on new SXM5 Module

Table 1. NVIDIA H100 Tensor Core GPU Performance Specs

	NVIDIA H100 SXM5	NVIDIA H100 PCIe
Peak FP64	33.5 TFLOPS	25.6 TFLOPS
Peak FP64 Tensor Core	66.9 TFLOPS	51.2 TFLOPS
Peak FP32	66.9 TFLOPS	51.2 TFLOPS
Peak FP16	133.8 TFLOPS	102.4 TFLOPS
Peak BF16	133.8 TFLOPS	102.4 TFLOPS
Peak TF32 Tensor Core	494.7 TFLOPS | 989.4 TFLOPS[1]	378 TFLOPS | 756 TFLOPS[1]
Peak FP16 Tensor Core	989.4 TFLOPS | 1978.9 TFLOPS[1]	756 TFLOPS | 1513 TFLOPS[1]
Peak BF16 Tensor Core	989.4 TFLOPS | 1978.9 TFLOPS[1]	756 TFLOPS | 1513 TFLOPS[2]
Peak FP8 Tensor Core	1978.9 TFLOPS | 3957.8 TFLOPS[1]	1513 TFLOPS | 3026 TFLOPS[1]
Peak INT8 Tensor Core	1978.9 TOPS | 3957.8 TOPS[1]	1513 TOPS | 3026 TOPS[1]

1. **Effective TFLOPS / TOPS using the Sparsity feature**

"NVIDIA H100 Tensor Core GPU Architecture," NVIDIA, White Paper v1.04.

Toy Floating-Point Format

- Consider an 8-bit floating point format with exponent bias 3

Sign	Exponent (3 bits)	Significand (4 bits)
b_7	$b_6b_5b_4$	$b_3b_2b_1b_0$

$$(-1)^{b_7} \times (1.b_3b_2b_1b_0)_2 \times 2^{b_6b_5b_4-3}$$

Largest Normalized Value

Sign	Exponent (3 bits)	Significand (4 bits)
b7	110	1111

$(-1)^{b_7} \cdot (1.1111)_2 \times 2^{6-3}$

$\times 2^3$

Smallest Normalized Value

Sign	Exponent (3 bits)	Significand (4 bits)
b7	001	0000

$(-1)^{b_7} \times (1.0)_2 \times 2^{1-3}$

$\times 2^{-2}$

Agenda

- Floating Point
  - **Special Values**
- Operations on Representations
  - C vs Java
  - "Hello World" Analysis
  - Compiling C Code

Going Even Smaller: Denormalization

- Exponent value `0x00` indicates a significand of $0.b_{23}b_{22}b_{21}...$
  - Good for **really small numbers** (called a "**denormalized value**")

Smallest Normalized Value

Sign	Exponent (8 bits)	Significand (23 bits)
b_{31}	0000_0001	000_0000_0000_0000_0000_0000

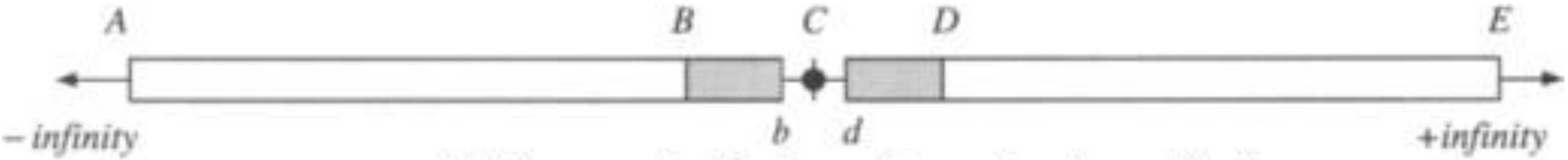
$$(1.000\_0000\_0000\_0000\_0000\_0000)_2 \times 2^{-126} \approx (1.18)_{10} \times 10^{-38}$$

Smallest Denormalized Value

Sign	Exponent (8 bits)	Significand (23 bits)
b_{31}	0000_0000	000_0000_0000_0000_0000_0001

$$(0.000\_0000\_0000\_0000\_0000\_0001)_2 \times 2^{-126} \approx (1.4)_{10} \times 10^{-45}$$

Floating Point Range (Normal + Denormal)



[A, B] — negative floating-point numbers (normalized)
[D, E] — positive floating-point numbers (normalized)
(B, b] & [d, D) — denormals
C — zero
> E — positive overflow
< A — negative overflow
(B, C) — negative underflow (normalized)
(C, D) — positive underflow (normalized)

Milos Ercegovac, Tomas Lang, "Digital Arithmetic", Morgan Kaufman, 2004

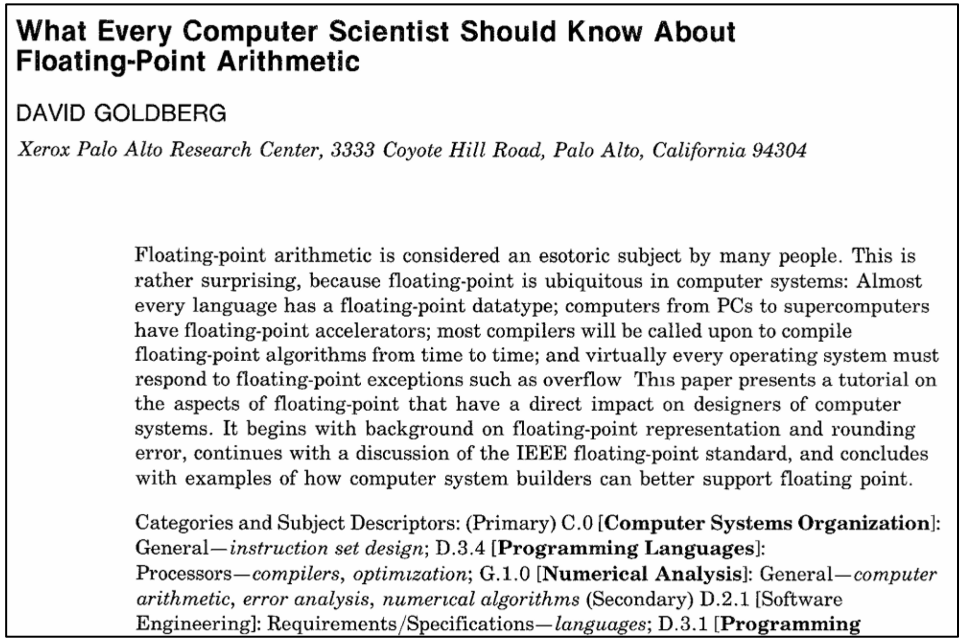
Special Numbers in IEEE 754

Table 5.4 **IEEE 754 floating-point notations for 0, $\pm\infty$, and NaN**

Number	Sign	Exponent	Fraction
0	X	00000000	00000000000000000000000
∞	0	11111111	00000000000000000000000
$-\infty$	1	11111111	00000000000000000000000
NaN	X	11111111	Non-zero

More on Floats

- There's a semester's worth of floating point material to consider
  - Precision, machine epsilon, and ulp
  - Normalized and denormalized numbers
  - Specialized floating-point types (e.g., 8-bit, 16-bit, etc.)
  - Representation error, rounding error, error propagation and bounding



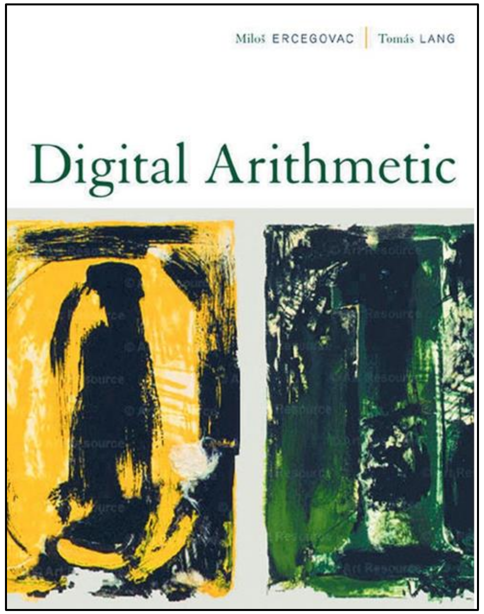
What Every Computer Scientist Should Know About Floating-Point Arithmetic

DAVID GOLDBERG

Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, California 94304

Floating-point arithmetic is considered an esotoric subject by many people. This is rather surprising, because floating-point is ubiquitous in computer systems: Almost every language has a floating-point datatype; computers from PCs to supercomputers have floating-point accelerators; most compilers will be called upon to compile floating-point algorithms from time to time; and virtually every operating system must respond to floating-point exceptions such as overflow This paper presents a tutorial on the aspects of floating-point that have a direct impact on designers of computer systems. It begins with background on floating-point representation and rounding error, continues with a discussion of the IEEE floating-point standard, and concludes with examples of how computer system builders can better support floating point.

Categories and Subject Descriptors: (Primary) C.0 [**Computer Systems Organization**]: General—*instruction set design*; D.3.4 [**Programming Languages**]: Processors—*compilers*, *optimization*; G.1.0 [**Numerical Analysis**]: General—*computer arithmetic*, *error analysis*, *numerical algorithms* (Secondary) D.2.1 [Software Engineering]: Requirements/Specifications—*languages*; D.3.1 [**Programming**



Milos Ercegovac, Tomas Lang, "Digital Arithmetic", Morgan Kaufman, 2004

Number Representations Summary

Unsigned

- ✓ Simple code
- ✓ `C(0) = 0`
- ✓ Sensible arithmetic
- ✗ Negative numbers

Two's Complement

- ✗ Simple code
- ✓ `C(0) = 0`
- ✓ Sensible arithmetic
- ✓ Negative numbers

Bias-K

- ✓ Simple code
- ✗ `C(0) = 0`
- ✗ Sensible arithmetic
- ✓ Negative numbers

Sign Magnitude

- ✓ Simple code
- ✓ `C(0) = 0`
- ✗ Sensible arithmetic
- ✓ Negative numbers

One's Complement

- ✗ Simple code
- ✓ `C(0) = 0`
- ✗ Sensible arithmetic
- ✓ Negative numbers

Fixed Point

- ✓ Simple code
- ✓ `C(0) = 0`
- ✗ Sensible arithmetic
- ✓ Negative numbers

Floating Point

- ✗ Simple code
- ✓ `C(0) = 0`
- ✗ Sensible arithmetic
- ✓ Negative numbers

Agenda

- Floating Point
  - Special Values
- **Operations on Representations**
  - C vs Java
  - “Hello World” Analysis
  - Compiling C Code

Finally.

We're done with number representations.

Or are we?

To a Computer: Everything is a Number(s)

On Screen (For Humans)

Text

Im Namen Gottes des Allmächtigen!

Das Schweizervolk und die Kantone,

in der Verantwortung gegenüber der Schöpfung,

im Bestreben, den Bund zu erneuern, um Freiheit und Demokratie, Unabhängigkeit und Frieden in Solidarität und Offenheit gegenüber der Welt zu stärken,

im Willen, in gegenseitiger Rücksichtnahme und Achtung ihre Vielfalt in der Einheit zu leben,

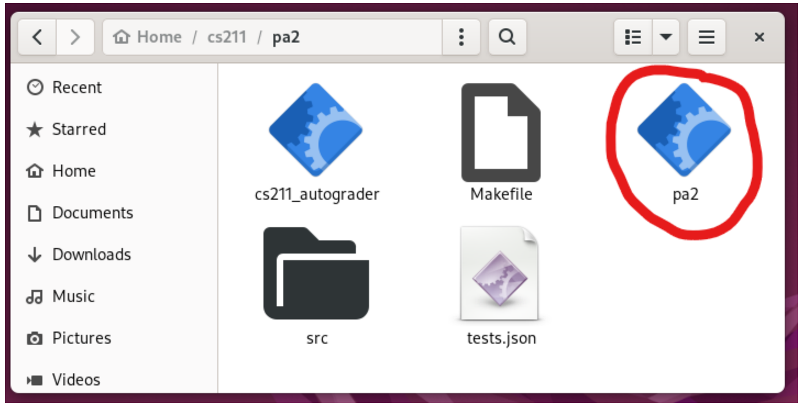
im Bewusstsein der gemeinsamen Errungenschaften und der Verantwortung gegenüber den künftigen Generationen,

gewiss, dass frei nur ist, wer seine Freiheit gebraucht, und dass die Stärke des Volkes sich misst am Wohl der Schwachen,

geben sich folgende Verfassung

Photo

App (PA2 Executable)



In Memory (or Disk) (For Computers)

```
00000000: 01001001 01101101 00100000 01001110 01100001 01101101
00000006: 01100101 01101110 00100000 01000111 01101111 01110100
0000000c: 01110100 01100101 01110011 00100000 01100100 01100101
00000012: 01110011 00100000 01000001 01101100 01101100 01101101
00000018: 11000011 10100100 01100011 01101000 01110100 01101001
0000001e: 01100111 01100101 01101110 00100001 00001010 00001010
00000024: 01000100 01100001 01110011 00100000 01010011 01100011
0000002a: 01101000 01110111 01100101 01101001 01111010 01100101
00000030: 01110010 01110110 01101111 01101100 01101011 00100000
00000036: 01110101 01101110 01100100 00100000 01100100 01101001
0000003c: 01100101 00100000 01001011 01100001 01101110 01110100
00000042: 01101111 01101110 01100101 00101100 00001010 00001010
00000048: 01101001 01101110 00100000 01100100 01100101 01110010
0000004e: 00100000 01010110 01100101 01110010 01100001 01101110
00000054: 01110100 01110111 01101111 01110010 01110100 01110101
0000005a: 01101110 01100111 00100000 01100111 01100101 01100111
00000060: 01100101 01101110 11000011 10111100 01100010 01100101
00000066: 01110010 00100000 01100100 01100101 01110010 00100000
0000006c: 01010011 01100011 01101000 11000011 10110110 01110000
00000072: 01100110 01110101 01101110 01100111 00101100 00001010
00000078: 00001010 01101001 01101101 00100000 01000010 01100101
0000007e: 01110011 01110100 01110010 01100101 01100010 01100101
00000084: 01101110 00101100 00100000 01100100 01100101 01101110
0000008a: 00100000 01000010 01110101 01101110 01100100 00100000
00000090: 01111010 01110101 00100000 01100101 01110010 01101110
```

```
00000000: 11111111 11011000 11111111 11100000 00000000 00010000
00000006: 01001010 01000110 01001001 01000110 00000000 00000001
0000000c: 00000001 00000000 00000000 00000001 00000000 00000001
00000012: 00000000 00000000 11111111 11100001 00000000 01101000
00000018: 01000101 01111000 01101001 01100110 00000000 00000000
0000001e: 01001001 01001001 00101010 00000000 00001000 00000000
00000024: 00000000 00000000 00000010 00000000 00110001 00000001
0000002a: 00000010 00000000 00000111 00000000 00000000 00000000
00000030: 00100110 00000000 00000000 00000000 01101001 10000111
00000036: 00000100 00000000 00000001 00000000 00000000 00000000
0000003c: 00101110 00000000 00000000 00000000 00000000 00000000
00000042: 00000000 00000000 01000111 01101111 01101111 01100111
00000048: 01101100 01100101 00000000 00000000 00000010 00000000
0000004e: 00000000 10010000 00000111 00000000 00000100 00000000
00000054: 00000000 00000000 00110000 00110010 00110010 00110000
0000005a: 00000011 10010000 00000010 00000000 00010100 00000000
00000060: 00000000 00000000 01001100 00000000 00000000 00000000
00000066: 00000000 00000000 00000000 00000000 00110010 00110000
0000006c: 00110010 00110101 00111010 00110000 00110001 00111010
00000072: 00110010 00111000 00100000 00110001 00110000 00111010
00000078: 00110000 00110011 00111010 00110011 00111001 00000000
0000007e: 11111111 11101101 00000000 00111000 01010000 01101000
00000084: 01101111 01110100 01101111 01110011 01101000 01101111
0000008a: 01110000 00100000 00110011 00101110 00110000 00000000
00000090: 00111000 01000010 01001001 01001101 00000100 00000100
```

```
00000000: 01111111 01000101 01001100 01000110 00000010 00000001
00000006: 00000001 00000000 00000000 00000000 00000000 00000000
0000000c: 00000000 00000000 00000000 00000000 00000010 00000000
00000012: 11110011 00000000 00000001 00000000 00000000 00000000
00000018: 01100100 00000001 00000001 00000000 00000000 00000000
0000001e: 00000000 00000000 01000000 00000000 00000000 00000000
00000024: 00000000 00000000 00000000 00000000 11001000 01111100
0000002a: 00000010 00000000 00000000 00000000 00000000 00000000
00000030: 00000000 00000000 00000000 00000000 01000000 00000000
00000036: 00111000 00000000 00000100 00000000 01000000 00000000
0000003c: 00011000 00000000 00010111 00000000 00000011 00000000
00000042: 00000000 01110000 00000100 00000000 00000000 00000000
00000048: 11101001 00001011 00000001 00000000 00000000 00000000
0000004e: 00000000 00000000 00000000 00000000 00000000 00000000
00000054: 00000000 00000000 00000000 00000000 00000000 00000000
0000005a: 00000000 00000000 00000000 00000000 00000000 00000000
00000060: 00011100 00000000 00000000 00000000 00000000 00000000
00000066: 00000000 00000000 00000000 00000000 00000000 00000000
0000006c: 00000000 00000000 00000000 00000000 00000001 00000000
00000072: 00000000 00000000 00000000 00000000 00000000 00000000
00000078: 00000001 00000000 00000000 00000000 00000101 00000000
0000007e: 00000000 00000000 00000000 00000000 00000000 00000000
00000084: 00000000 00000000 00000000 00000000 00000000 00000000
0000008a: 00000001 00000000 00000000 00000000 00000000 00000000
00000090: 00000000 00000000 00000001 00000000 00000000 00000000
```

To a Computer: Everything is a Number(s)

Screen (For Humans)

Text

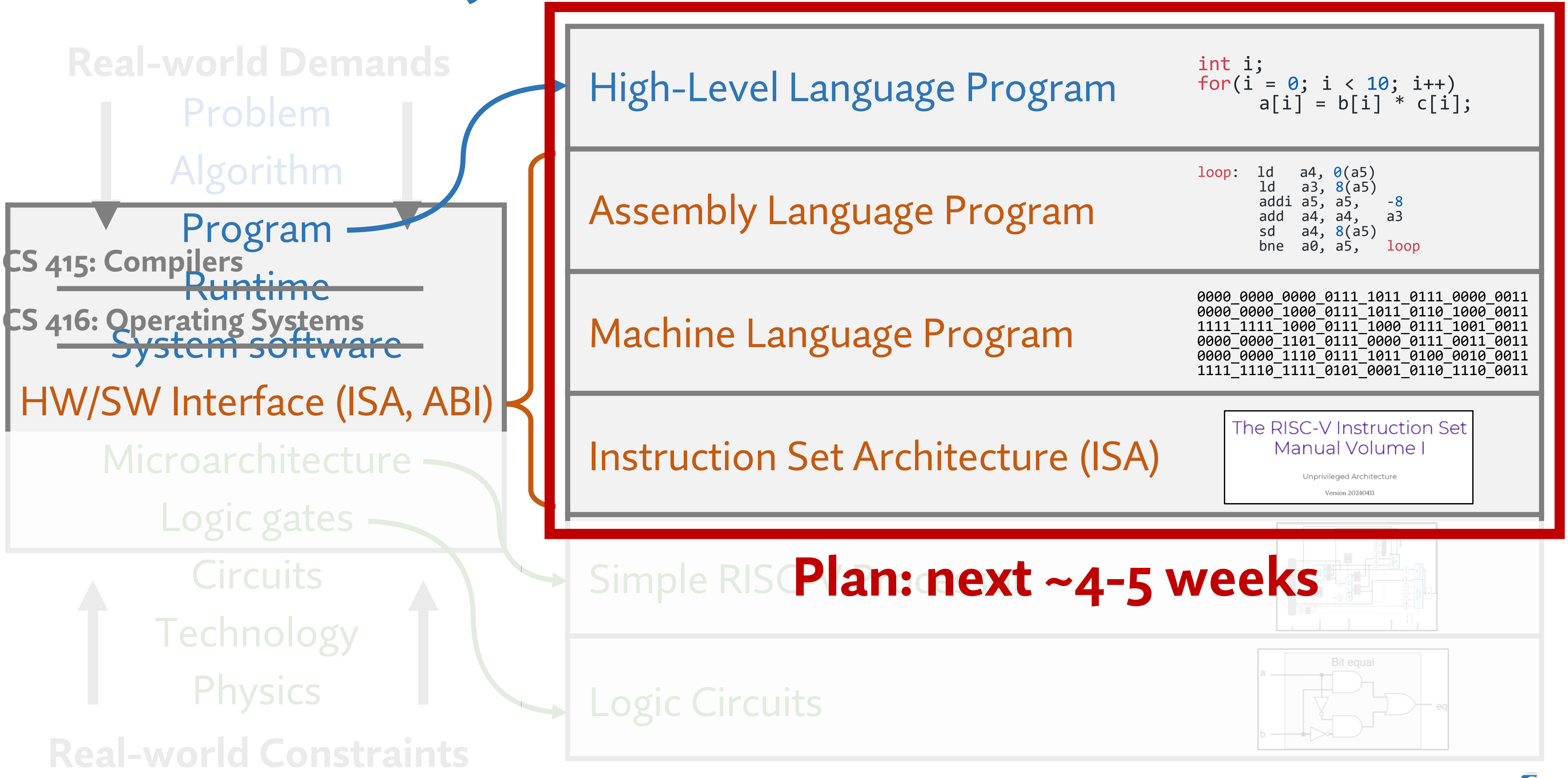
Photo

App (PA2 Executable)

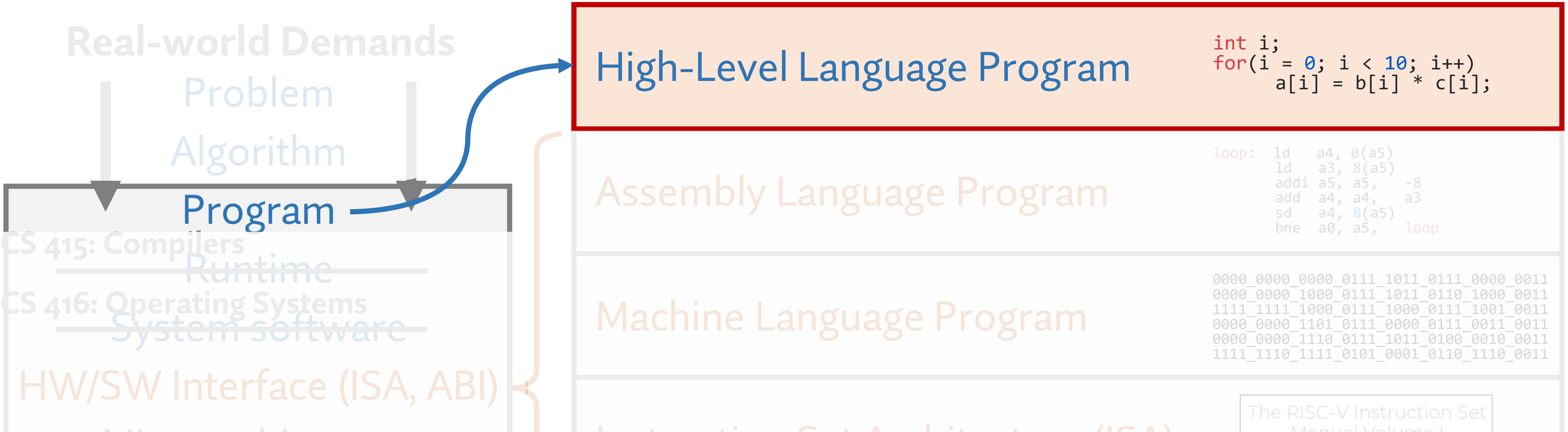
Number representations will be a **recurring theme** throughout this course + future systems courses

In Memory (For Computers)

What’s Next: Layers of Abstraction for CS 211



What's Next: Layers of Abstraction for CS 211



Next ~2-3 weeks:

Using C as a tool to **interact with** and **manipulate** machine representations of data

Agenda

- Floating Point
  - Special Values
- Operations on Representations
  - **C vs Java**
  - “Hello World” Analysis
  - Compiling C Code

C vs. Java: Similarities

- Mostly **syntax-related** topics (Java came from C, after all)
  - You should be able to read C easily, if not understand what's happening

	C	Java
Using a Library	#include <lib>	import java.io.File
Comments	/* ... */ or // rest of line	
Variable Declaration	Always before use	
Operators	Arithmetic, Assignment: +,-,*,/,%, =, +=,-=,… Boolean: !, &&, || Comparators: ==, !=, <, <=, >, >= Increment and decrement: ++ and – Bitwise operators: <<,>>,~,&,|,^ Subgroup expressions: ()	
Conditionals	if, switch	
Loops	for, while, do-while	
Function Calls	int ret = fn(0.0, 1);	
Casting	int as_int = (int)0.0;	

Inspired by Kao Garcia (CS 61C)

C vs. Java: Differences

- Mostly **concept-related** topics
  - We will go over some of these in class
  - You will learn others by doing the programming assignments

	C	Java
Language Type	Procedural	Object Oriented
Programming Unit	Functions	Class (abstract data type)
Memory Management	Manual	Automatic (JVM-managed)
Compilation	Different compiler for each architecture	One compiler for all architectures
Execution	Run on the hardware	Interpreted by the JVM
Variable Usage	Pointers	References
Problems	Error Codes	Exceptions
Standard Libraries	Very limited	Many useful libraries (data structures, I/O, etc.)
Vars/Arrays/Strings	Objects in memory	Abstract data type

C for Java Programmers

- There are many, many resources for Java programmers to learn C
  - George Ferguson (University of Rochester), "*C for Java Programmers*," 2016.
  - Jason Maassen (VU Amsterdam), "*C for Java Programmers*."
  - Tomasz Muldner (Acadia University), "*C for Java Programmers*," 2000.
  - Charlie McDowell (UC Santa Cruz), "*C for Java Programmers*," 2000.
  - And many more…

2.1 What's The Same?

Since Java is derived from C, you will find many things that are familiar:

- Values, types (more or less), literals, expressions
- Variables (more or less)
- Conditionals: `if`, `switch`
- Iteration: `while`, `for`, `do-while`, but not `for`-in-collection ("colon" syntax)
- Call-return (methods in Java, functions in C): parameters/arguments, return values
- Arrays (with one big difference)
- Primitive and reference types
- Typecasts
- Libraries that extend the core language (although the mechanisms differ somewhat)

2.2 What's Different?

On the other hand, C differs from Java in some important ways. This section gives you a quick heads-up on the most important of these.

- No classes or objects: C is not an object-oriented language, although it has structured types (`struct` and `union`) and there are ways of doing things like inheritance, abstraction, and composition. C also has a mechanism for extending its type system (`typedef`).
- Arrays are simpler: there is no bounds checking (your program just dies if you access a bad index), and arrays don't even know their own size!
- Strings are much more limited, although the C standard library helps.
- No collections (lists, hashtables, *etc.*), exceptions, or generics.
- No memory management: You must explicitly allocate (`malloc`) and release (`free`) memory to use dynamic data structures like lists, trees, and so on. C does almost nothing to prevent you from messing up the memory used by your program. *This is the number one cause of problems for new C programmers (and old ones also).*
- Pointer arithmetic: You can, and often have to, use addresses of items in memory (called pointers). C allows you to change the contents of memory almost arbitrarily. This is powerful magic, but also sometimes dangerous magic.

George Ferguson, "*C for Java Programmers*," 2016.

Agenda

- Floating Point
  - Special Values
- Operations on Representations
  - C vs Java
  - **“Hello World” Analysis**
  - Compiling C Code

Lecture Ended Here

To Be Continued

CS 211: Intro to Computer Architecture

4.1: Floating Point and C Data Representations

Minesh Patel

Spring 2025 – Tuesday 11 February