CSC 252: Computer Organization
Spring 2025: Lecture 5

Instructor: Yuhao Zhu

Department of Computer Science
University of Rochester

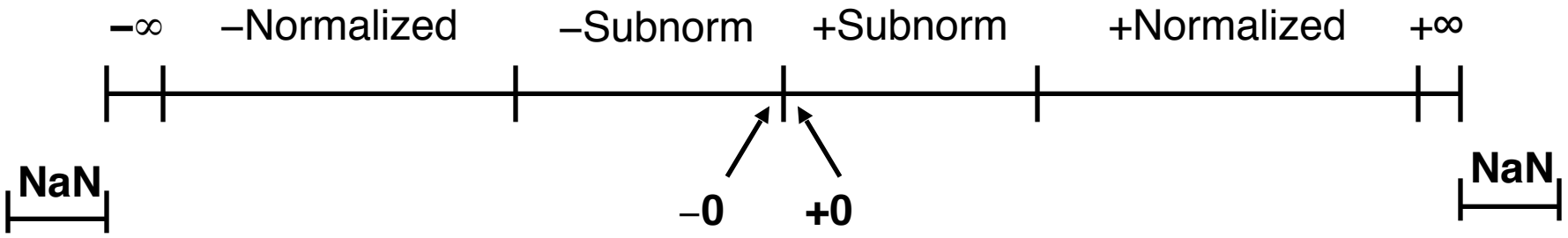
Announcement

- Programming Assignment 1 is out
  - Details: https://cs.rochester.edu/courses/252/spring2025/labs/assignment1.html
  - Due on Feb. 12, 11:59 PM
  - You have 3 slip days

Announcement

- Programming assignment 1 is in C language. Seek help from TAs.
- TAs are best positioned to answer your questions about programming assignments!!!
- Programming assignments do NOT repeat the lecture materials. They ask you to synthesize what you have learned from the lectures and work out something new.
- When emailing TAs, email them all.

Visualization: Floating Point Encodings



Infinite Amount of Real Numbers

Sparse Dense Sparse

Finite Amount of Floating Point Numbers

Today: Floating Point

- Background: Fractional binary numbers and fixed-point
- Floating point representation
- IEEE 754 standard
- Rounding, addition, multiplication
- Floating point in C
- Summary

IEEE 754 Floating Point Standard

- Single precision: 32 bits

- Double precision: 64 bits

IEEE Floating Point

- IEEE Standard 754
  - Established in 1985 as uniform standard for floating point arithmetic
    - Before that, many idiosyncratic formats
  - Supported by all major CPUs (and even GPUs and other processors)

- Driven by numerical concerns
  - Nice standards for rounding, overflow, underflow
  - Hard to make fast in hardware
    - Numerical analysts predominated over hardware designers in defining standard

Single Precision (32-bit) Example

$v = (-1)^s M 2^E$

bias = $2^{(8-1)}-1 = 127$

1 8-bit 23-bit

$15213_{10} = 11101101101101_2$

$= (-1)^0\, 1.11011011011101_2 \times 2^{13}$

$exp = E + bias = 140_{10}$

Today: Floating Point

- Background: Fractional binary numbers and fixed-point
- Floating point representation
- IEEE 754 standard
- Rounding, addition, multiplication
- Floating point in C
- Summary

Floating Point Computations

- The problem: Computing on floating point numbers might produce a result that can't be precisely represented
- Basic idea
  - We perform the operation & produce the infinitely precise result
  - Make it fit into desired precision
    - Possibly overflow if exponent too large
    - Possibly round to fit into frac

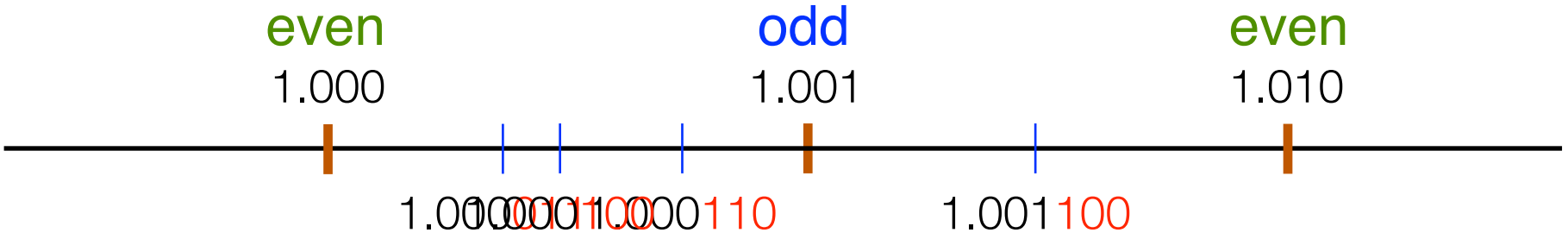
Rounding Modes (Decimal)

- Common ones:
  - Towards zero (chop)
  - Round down (-∞)
  - Round up (+∞)
- Nearest Even: Round to nearest; if equally near, then to the one having an even least significant digit (bit)

Rounding Mode	**1.40**	**1.60**	**1.50**	**2.50**	**-1.50**
Towards zero	1	1	1	2	-1
Round down (-∞)	1	1	1	2	-2
Round up (+∞)	2	2	2	3	-1
Nearest even (default)	1	2	2	2	-2

Rounding Modes (Binary Example)

- Nearest Even; if equally near, then to the one having an even least significant digit (bit)
- Assuming 3 bits for *frac*



Precise Value	Rounded Value	Notes
1.000011	1.000	1.000 is the nearest (down)
1.000110	1.001	1.001 is the nearest (up)
1.000100	1.000	1.000 is the nearest even (down)
1.001100	1.010	1.010 is the nearest even (up)

Floating Point Addition

- $(-1)^{s1}\ M1\ \ 2^{E1}\ \ +\ \ (-1)^{s2}\ M2\ \ 2^{E2}$
- Exact Result: $(-1)^{s}\ M\ \ 2^{E}$
  - Sign *s*, significand *M*:
    - Result of signed align & add
  - Exponent E: E1
    - Assume E1 > E2
- Fixing
  - If $M \geq 2$, shift *M* right, increment *E*
  - If $M < 1$, shift *M* left *k* positions, decrement *E* by *k*
  - Overflow if *E* out of range
  - Round *M* to fit *frac* precision

$1.000 \times 2^{-1} + 11.10 \times 2^{-3}$

align $1.000 \times 2^{-1} + 0.111 \times 2^{-1}$

add 1.111×2^{-1}

Mathematical Properties of FP Add

- Commutative? ***Yes***
- Associative? ***No***
  - Overflow and inexactness of rounding
  - `(3.14+1e10)-1e10 = 0, 3.14+(1e10-1e10) = 3.14`
- 0 is additive identity? ***Yes***
- Every element has additive inverse (negation)? ***Almost***
  - Except for infinities & NaNs
- Monotonicity: $a \geq b \Rightarrow a+c \geq b+c$? ***Almost***
  - Except for infinities & NaNs

Floating Point Multiplication

- $(-1)^{s1}\ M1\ \ 2^{E1}\ \ \times\ \ (-1)^{s2}\ M2\ \ 2^{E2}$
- Exact Result: $(-1)^{s}\ M\ \ 2^{E}$
  - Sign *s*: *s1* ^ *s2*
  - Significand *M*: *M1* x *M2*
  - Exponent *E*: *E1* + *E2*
- Fixing
  - If M ≥ 2, shift M right, increment E
  - If E out of range, overflow
  - Round M to fit frac precision
- Implementation
  - Biggest chore is multiplying significands

Mathematical Properties of FP Mult

- Multiplication Commutative? ***Yes***
- Multiplication is Associative? ***No***
  - Possibility of overflow, inexactness of rounding
  - Ex: `(1e20*1e20)*1e-20= inf, 1e20*(1e20*1e-20)= 1e20`
- 1 is multiplicative identity? ***Yes***
- Multiplication distributes over addition? ***No***
  - Possibility of overflow, inexactness of rounding
  - `1e20*(1e20-1e20)= 0.0,  1e20*1e20 - 1e20*1e20 = NaN`
- Monotonicity: $a \geq b \ \& \ c \geq 0 \Rightarrow a * c \geq b * c$? ***Almost***
  - Except for infinities & NaNs

Today: Floating Point

- Background: Fractional binary numbers and fixed-point
- Floating point representation
- IEEE 754 standard
- Rounding, addition, multiplication
- Floating point in C
- Summary

IEEE 754 Floating Point Standard

- Single precision: 32 bits

- Double precision: 64 bits

- In C language
  - `float` single precision
  - `double` double precision

Floating Point in C

32-bit Machine

Fixed point (implicit binary point)

SP floating point

DP floating point

C Data Type	Bits	Max Value	Max Value (Decimal)
`char`	8	$2^7 - 1$	127
`short`	16	$2^{15} - 1$	32767
`int`	32	$2^{31} - 1$	2147483647
`long`	64	$2^{63} - 1$	$\sim 9.2 \times 10^{18}$
`float`	32	$(2 - 2^{-23}) \times 2^{127}$	$\sim 3.4 \times 10^{38}$
`double`	64	$(2 - 2^{-52}) \times 2^{1023}$	$\sim 1.8 \times 10^{308}$

- To represent 2^{31} in fixed-point, you need at least 32 bits
  - Because fixed-point is a *weighted positional* representation
- In floating-point, we directly encode the exponent
  - Floating point is based on scientific notation
  - Encoding 31 only needs 7 bits in the *exp* field

Floating Point in C

64-bit Machine

Fixed point
(implicit binary point)

SP floating point

DP floating point

C Data Type	Bits	Max Value	Max Value (Decimal)
`char`	8	$2^7 - 1$	127
`short`	16	$2^{15} - 1$	32767
`int`	32	$2^{31} - 1$	2147483647
`long`	64	$2^{31} - 1$	$\sim 9.2 \times 10^{18}$
`float`	32	$(2 - 2^{-23}) \times 2^{127}$	$\sim 3.4 \times 10^{38}$
`double`	64	$(2 - 2^{-52}) \times 2^{1023}$	$\sim 1.8 \times 10^{308}$

Floating Point Conversions/Casting in C

- **`double/float → int`**
  - Truncates fractional part
  - Like rounding toward zero
  - Not defined when out of range or NaN
- **`int → float`**
  - Can't guarantee exact casting. Will round according to rounding mode

- **`int → double`**
  - Exact conversion

Today: Floating Point

- Background: Fractional binary numbers and fixed-point
- Floating point representation
- IEEE 754 standard
- Rounding, addition, multiplication
- Floating point in C
- Summary

Floating Point Review

$$v = (-1)^s \times 1.frac \times 2^E$$

- Denormalized
  - E = (exp + 1) – bias
  - M = 0.frac
- Normalized
  - E = exp – bias
  - M = 1.frac

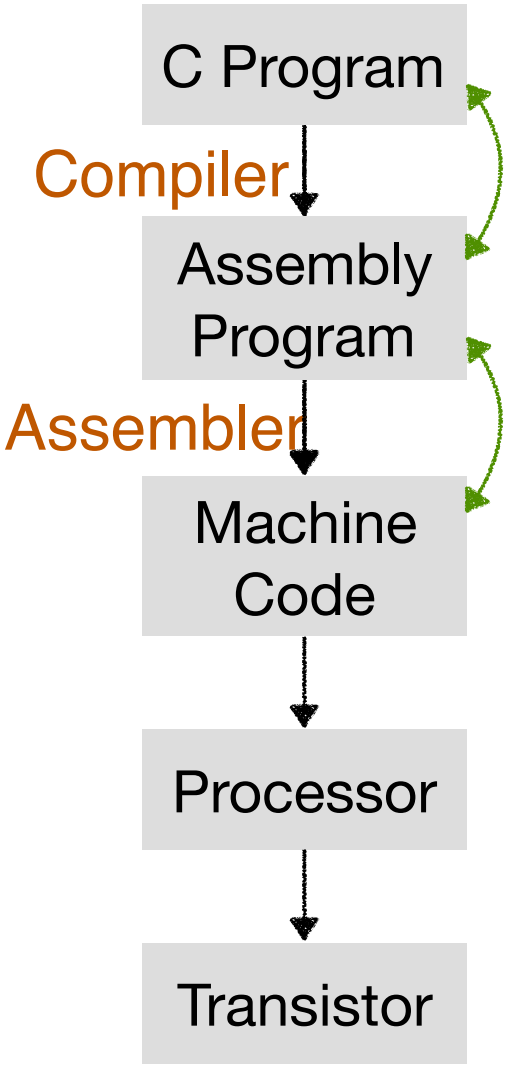
	s	*exp*	*frac*	Value	Value
Denormalized	0	000	00	0.00 x 2^{-2}	0
	0	000	11	0.11 x 2^{-2}	3/16
Normalized	0	001	00	1.00 x 2^{-2}	1/4
	0	001	11	1.11 x 2^{-2}	7/16
	0	010	00	1.00 x 2^{-1}	1/2
	0	010	11	1.11 x 2^{-1}	7/8
	0	100	00	1.00 x 2^{0}	1
	0	100	11	1.11 x 2^{0}	1 3/4
	0	101	00	1.00 x 2^{1}	2
	0	101	11	1.11 x 2^{1}	3 1/2
	0	110	00	1.00 x 2^{2}	4
	0	110	11	1.11 x 2^{2}	7
Special Value	0	111	00	infinite	infinite
	0	111	11	NaN	NaN

Floating Point Review

- If you do an integer increment on a positive FP number, you get the next larger FP number.
- Bit patterns representing non-negative numbers are ordered the same way as integers, so could use regular integer comparison.
- You don't get this property if:
  - *exp* is interpreted as signed
  - *exp* and *frac* are swapped

	s	*exp*	*frac*	Value	Value
Denormalized	0	000	00	0.00×2^{-2}	0
	0	000	11	0.11×2^{-2}	3/16
Normalized	0	001	00	1.00×2^{-2}	1/4
	0	001	11	1.11×2^{-2}	7/16
	0	010	00	1.00×2^{-1}	1/2
	0	010	11	1.11×2^{-1}	7/8
	0	100	00	1.00×2^{0}	1
	0	100	11	1.11×2^{0}	1 3/4
	0	101	00	1.00×2^{1}	2
	0	101	11	1.11×2^{1}	3 1/2
	0	110	00	1.00×2^{2}	4
	0	110	11	1.11×2^{2}	7
Special Value	0	111	00	infinite	infinite
	0	111	11	NaN	NaN

So far in 252…



`int`, **`float`**
`if`, `else`
`+`, `-`, `>>`

`ret`, `call`
`fadd`, **`add`**
`jmp`, `jne`

00001111
01010101
11110000

Fixed-point adder
(e.g., ripple carry),
Floating-point adder

NAND Gate
NOR Gate

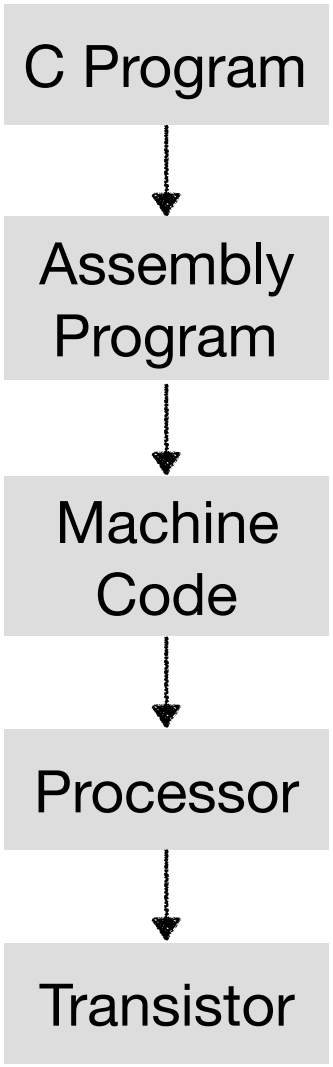
So far in 252…

High-Level Language

Instruction Set Architecture (ISA)

Microarchitecture

Circuit



- ISA: Software programmers' view of a computer
  - Provide all info for someone wants to write assembly/machine code
  - "Contract" between assembly/ machine code and processor
- Processors execute machine code (binary). Assembly program is merely a text representation of machine code
- Microarchitecture: Hardware implementation of the ISA (with the help of circuit technologies)

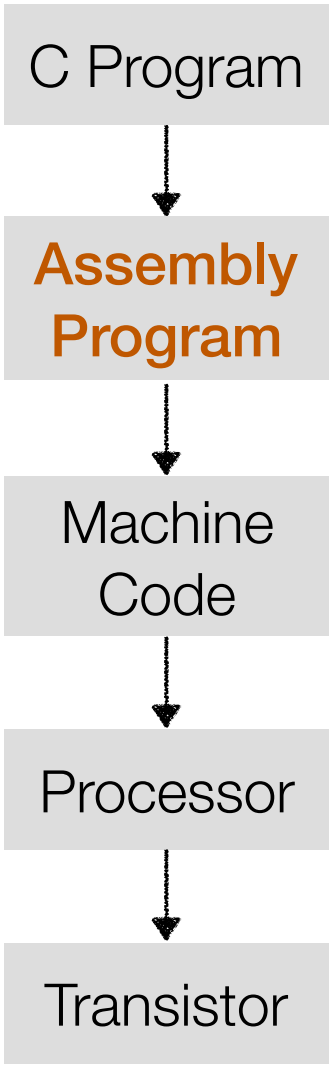
This Module (4 Lectures)

High-Level Language

Instruction Set Architecture (ISA)

Microarchitecture

Circuit



- Assembly Programming
  - Explain how various C constructs are implemented in assembly code
  - Effectively translating from C to assembly program manually
  - Helps us understand how compilers work
  - Helps us understand how assemblers work
- Microarchitecture is the topic of the next module

Today: Assembly Programming I: Basics

- Different ISAs and history behind them
- C, assembly, machine code
- Move operations (and addressing modes)

Instruction Set Architecture

- There used to be many ISAs
  - x86, ARM, Power/PowerPC, Sparc, MIPS, IA64, z
  - Very consolidated today: ARM for mobile, x86 for others
- There are even more microarchitectures
  - Apple/Samsung/Qualcomm have their own microarchitecture (implementation) of the ARM ISA
  - Intel and AMD have different microarchitectures for x86
- ISA is lucrative business: ARM's Business Model
  - Patent the ISA, and then license the ISA
  - Every implementer pays a royalty to ARM
  - Apple/Samsung pays ARM whenever they sell a smartphone

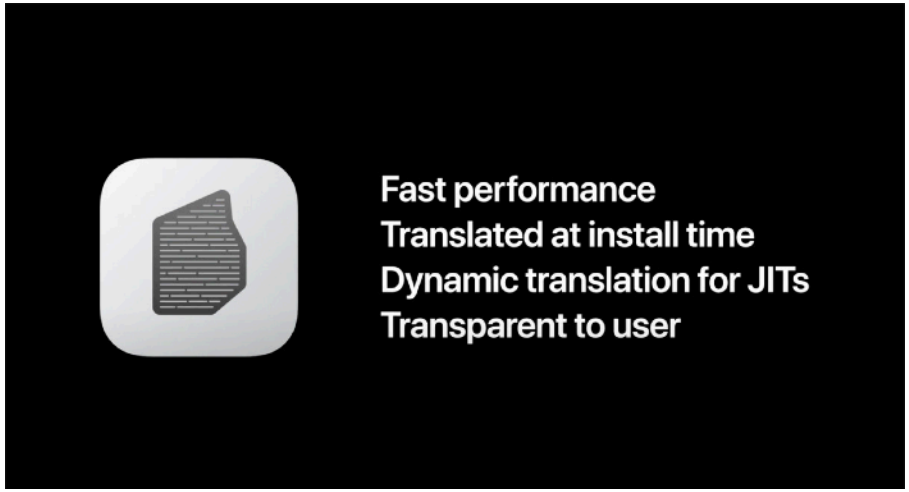
The ARM Diaries, Part 1: How ARM's Business Model Works: https://www.anandtech.com/show/7112/the-arm-diaries-part-1-how-arms-business-model-works

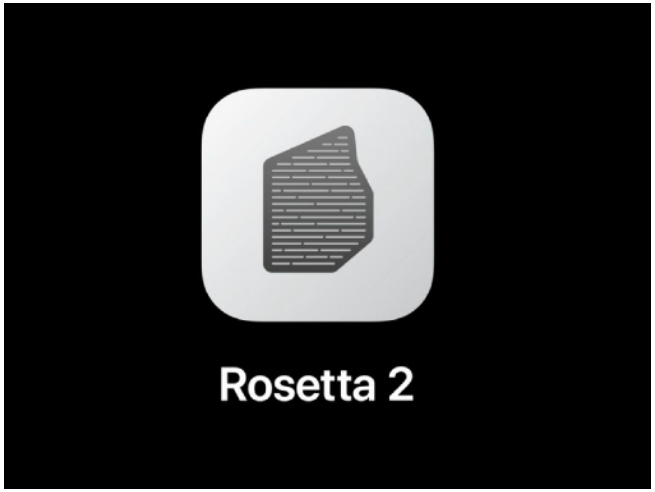
Intel x86 ISA

- Dominate laptop/desktop/cloud market

Aside: Dynamic Binary Translation

- Apple M1 is based on the Arm ISA. A program compiled to x86 ISA is dynamically translated to Arm ISA by Rosetta.
- Not the first time Apple plays this trick.





Aside: Dynamic Binary Translation