Goals for Today

Finish discussion of floats

- Denorms, epsilon, arithmetic error

Introduce assembly language

- What is it? Why do we study it?

The essentials of x86-64

- Registers, memory, the `mov` instruction

Recap: Floating Point

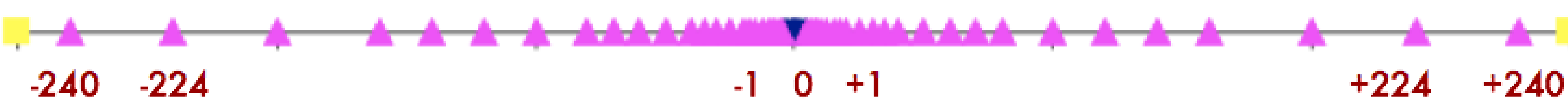
Split number into exponent and significand

- Exponent like the units of the number

Value of float: $\pm 1.\mathrm{xxx}_2 \cdot 2^{\mathrm{yyy}}$

minifloat: 8 bits

- 1 sign, 4 exponent, 3 significand



Denormalized Numbers

Exponent bits all 0

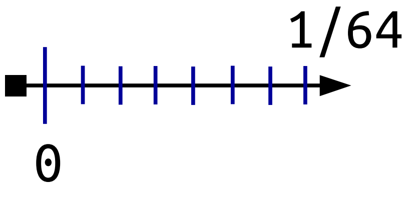
No implicit 1, exponent = `1 - bias`

$\pm 0.SSS_2 \cdot 2^{1\text{-bias}}$

Zero is all bits 0

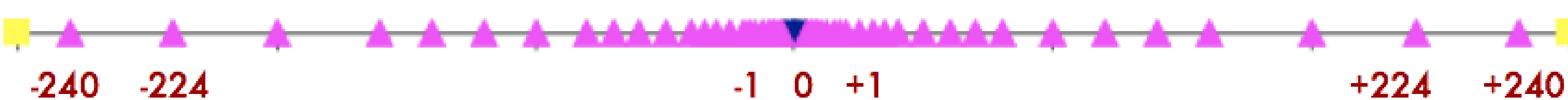
-0: separate bit pattern, hardware handles

Equally spaced



minifloat

	s	exp	frac	*E*	Value	
Denormalized numbers	0	0000	000	-6	0	
	0	0000	001	-6	1/8*1/64 = 1/512	closest to zero
	0	0000	010	-6	2/8*1/64 = 2/512	
	...					
	0	0000	110	-6	6/8*1/64 = 6/512	
	0	0000	111	-6	7/8*1/64 = 7/512	largest denorm
Normalized numbers	0	0001	000	-6	8/8*1/64 = 8/512	smallest norm
	0	0001	001	-6	9/8*1/64 = 9/512	
	...					
	0	0110	110	-1	14/8*1/2 = 14/16	
	0	0110	111	-1	15/8*1/2 = 15/16	closest to 1 below
	0	0111	000	0	8/8*1 = 1	
	0	0111	001	0	9/8*1 = 9/8	closest to 1 above
	0	0111	010	0	10/8*1 = 10/8	
	...					
	0	1110	110	7	14/8*128 = 224	
	0	1110	111	7	15/8*128 = 240	largest norm
	0	1111	000	n/a	inf	



Arithmetic

Addition

- If different exponent, change smaller value to match
- May lose precision (e.g. add 1 to a billion)

Multiplication

- Multiply significand, add exponent

Big Picture Takeaways

Almost all decimal numbers can’t be represented exactly

Relative vs. absolute error

- If checking whether numbers are “close enough,” no single number will do

Error can e precisely quantified

So Far

Finish discussion of floats

- Denorms, epsilon, arithmetic error

Introduce assembly language

- What is it? Why do we study it?

The essentials of x86-64

- Registers, memory, the `mov` instruction

Assembly Language

What happens to C code when we compile?

- C is not directly executed on hardware
- Ultimately translated to sequence of bytes the hardware interprets (machine language)

Assembly language

- One step before machine language
- Can see individual instructions

Why Study Assembly?

Essential for compiler writers

- But most of us aren't

Probably won't need to hand-generate asm

- ...Sometimes (custom hardware, embedded systems)

Reading assembly

- Understand compiler optimizations
- Adapt your C code to hardware

Focus: C <-> assembly translation

ISA: Instruction Set Architecture

Contract between hardware and software

What the hardware can do

- Memory access
- Arithmetic (simple ops)
- Control flow (branch/jump)

How programs/functions interact

- Function calls (parameter passing, return value)
- Memory layout

Brief History of X86-64

Oiriginal 8086 in 1978

- Started out 16-bit, then 32, now 64

Increasing complexity over time

- Can't change/remove instructions from ISA
- Can only add

Only need to learn a very small subset

x86-64 Overview

16 Registers

- 8-byte (64-bit) "boxes"
- Calculations done on registers
- Lots of moving to/from memory

Example: `x = y + 5`

- Move y from memory to register %rax
- Add 5 to %rax
- Move %rax to x in memory

x86-64 Integer Registers

64 bit

%rax
%rbx
%rcx
%rdx
%rdi
%rsi
%rbp
%rsp
%r8

... %r9 through %r15 (just like %r8)

x86-64 Integer Registers

64 bit	32 bit
%rax	%eax
%rbx	%ebx
%rcx	%ecx
%rdx	%edx
%rdi	%edi
%rsi	%esi
%rbp	%ebp
%rsp	%esp
%r8	%r8d

... %r9 through %r15 (just like %r8)

x86-64 Integer Registers

64 bit	32 bit	16 bit	8 bit
%rax	%eax	%ax	%al
%rbx	%ebx	%bx	%bl
%rcx	%ecx	%cx	%cl
%rdx	%edx	%dx	%dl
%rdi	%edi	%di	%dil
%rsi	%esi	%si	%sil
%rbp	%ebp	%bp	%bpl
%rsp	%esp	%sp	%spl
%r8	%r8d	%r8w	%r8b

... %r9 through %r15 (just like %r8)

mov Instruction

`mov src, dst`

- Copy value from src to dst
- src can be constant, register, memory
- dst can be register, memory
- (Both cannot be memory)

Many addresing modes

GCC Explorer: mov

(See link on syllabus page)

Summary

Finish discussion of floats

- Denorms, epsilon, arithmetic error

Introduce assembly language

- What is it? Why do we study it?

The essentials of x86-64

- Registers, memory, the `mov` instruction