

Data Representation: Floating Point Mastery

Yipeng Huang

Rutgers University

October 23, 2026

Data Representation: Floating Point Mastery

Announcements

Programming assignment 3 and midterm exam

Floats: Special cases

Floats: Properties

Floating point multiplication

Floats: Understanding its design

Deep understanding 1: Why is exp field encoded using bias?

Deep understanding 2: Why have denormalized numbers?

Deep understanding 3: Why is bias chosen to be $2^{k-1} - 1$?

Floats: Summary

Computer organization: A primer

Assembly code: Human readable representation of machine code

Instruction set architectures

`swap.s`: Assembly implementation of function that swaps memory contents

Data Representation: Floating Point Mastery

Announcements

Programming assignment 3 and midterm exam

Floats: Special cases

Floats: Properties

Floating point multiplication

Floats: Understanding its design

Deep understanding 1: Why is exp field encoded using bias?

Deep understanding 2: Why have denormalized numbers?

Deep understanding 3: Why is bias chosen to be $2^{k-1} - 1$?

Floats: Summary

Computer organization: A primer

Assembly code: Human readable representation of machine code

Instruction set architectures

`swap.s`: Assembly implementation of function that swaps memory contents

Programming assignment 3 and midterm exam

Programming assignment 3

1. Due Tuesday 10/27.
2. Binary representations of integers and floating point numbers.

In-class midterm exam

1. Friday, October 30 12:10 pm - 13:30 pm
2. Lucy Stone Hall LSH-A102
3. Closed book, closed notes exam
4. Bring: pencils and eraser
5. Don't bring: electronic devices

Epsilon vs. Smallest Positive Number

▶ Smallest Positive Normalized Number (Tiny Float):

- ▶ Encoding: 0 | 0001 | 000
- ▶ Value: $1.0_2 \times 2^{-6} = \frac{1}{64}$

▶ Machine Epsilon (ϵ):

- ▶ *Definition:* The difference between 1.0 and the next representable number greater than 1.0.
- ▶ Encoding for 1.0: 0 | 0111 | 000
- ▶ Next representable number: 0 | 0111 | 001 $\implies 1.001_2 \times 2^0 = 1 + \frac{1}{8}$
- ▶ Value of ϵ in Tiny Float: $\frac{1}{8}$

- ▶ **Key Takeaway:** Epsilon is **not** an infinitesimally small number, nor is it the smallest representable number. It defines the relative spacing/precision of the floating-point system at 1.0.

Data Representation: Floating Point Mastery

Announcements

Programming assignment 3 and midterm exam

Floats: Special cases

Floats: Properties

Floating point multiplication

Floats: Understanding its design

Deep understanding 1: Why is exp field encoded using bias?

Deep understanding 2: Why have denormalized numbers?

Deep understanding 3: Why is bias chosen to be $2^{k-1} - 1$?

Floats: Summary

Computer organization: A primer

Assembly code: Human readable representation of machine code

Instruction set architectures

`swap.s`: Assembly implementation of function that swaps memory contents

Floats: Special cases

number class	when it arises	exp field	frac field
+0 / -0		0	0
+infinity / -infinity	overflow or division by 0	$2^k - 1$	0
NaN not-a-number	illegal ops. such as $\sqrt{-1}$, inf-inf, inf*0	$2^k - 1$	non-0

Table: Summary of special cases

Data Representation: Floating Point Mastery

Announcements

Programming assignment 3 and midterm exam

Floats: Special cases

Floats: Properties

Floating point multiplication

Floats: Understanding its design

Deep understanding 1: Why is exp field encoded using bias?

Deep understanding 2: Why have denormalized numbers?

Deep understanding 3: Why is bias chosen to be $2^{k-1} - 1$?

Floats: Summary

Computer organization: A primer

Assembly code: Human readable representation of machine code

Instruction set architectures

`swap.s`: Assembly implementation of function that swaps memory contents

$$\mathbb{Z} \subset \mathbb{R} \subset \mathbb{C} \subset \mathbb{Q} \dots$$

range, define by bit length
modular

Identity

addition (+)

0.0

multiplication (x)

1.0

Commutative

$$A+B = B+A$$

$$AB = BA$$

Associative

$$(A+B)+C = A+(B+C)$$

tiny float

A B C

$$(240+64)-64 \quad A+(B+C)$$

$$= \infty - 64$$

$$240+(64-64)$$

$$= \infty$$

$$= 240+0$$

$$= 240$$

$$(A \ B) \ C = A \ (B \ C)$$

distribution

$$A(B+C) = AB + AC$$

$$A = \frac{1}{64} \quad B = 240 \quad C = 64 \quad \frac{1}{64} \cdot 240 + \frac{1}{64} \cdot 64$$

$$A(B+C) = \frac{30}{8} + 1$$

$$= A(240+64)$$

$$= 3 \dots + 1$$

$$= A \cdot 300 = \frac{30}{64} = \infty$$

$$= 4$$

$$A = 240 \quad B = 64 \quad C = -64$$

$$A(B+C)$$

$$AB + AC$$

$$= 240(64 - 64)$$

$$240 \times 64 - 240 \cdot 64$$

$$= 240 \cdot 0$$

$$= \infty - \infty$$

$$= 0$$

$$= NaN$$

inverse

additive
inverse
always
exists

multiplicative
inv

Data Representation: Floating Point Mastery

Announcements

Programming assignment 3 and midterm exam

Floats: Special cases

Floats: Properties

Floating point multiplication

Floats: Understanding its design

Deep understanding 1: Why is exp field encoded using bias?

Deep understanding 2: Why have denormalized numbers?

Deep understanding 3: Why is bias chosen to be $2^{k-1} - 1$?

Floats: Summary

Computer organization: A primer

Assembly code: Human readable representation of machine code

Instruction set architectures

`swap.s`: Assembly implementation of function that swaps memory contents

How to multiply scientific notation?

Recall: $\log(x \times y) = \log(x) + \log(y)$

FP Multiplication

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

Data Representation: Floating Point Mastery

Announcements

Programming assignment 3 and midterm exam

Floats: Special cases

Floats: Properties

Floating point multiplication

Floats: Understanding its design

Deep understanding 1: Why is exp field encoded using bias?

Deep understanding 2: Why have denormalized numbers?

Deep understanding 3: Why is bias chosen to be $2^{k-1} - 1$?

Floats: Summary

Computer organization: A primer

Assembly code: Human readable representation of machine code

Instruction set architectures

`swap.s`: Assembly implementation of function that swaps memory contents

Data Representation: Floating Point Mastery

Announcements

Programming assignment 3 and midterm exam

Floats: Special cases

Floats: Properties

Floating point multiplication

Floats: Understanding its design

Deep understanding 1: Why is exp field encoded using bias?

Deep understanding 2: Why have denormalized numbers?

Deep understanding 3: Why is bias chosen to be $2^{k-1} - 1$?

Floats: Summary

Computer organization: A primer

Assembly code: Human readable representation of machine code

Instruction set architectures

`swap.s`: Assembly implementation of function that swaps memory contents

Deep understanding 1: Why is exp field encoded using bias?

exp field needs to encode both positive and negative exponents.

Why not just use one of the signed integer formats? 2's complement, 1s' complement, signed magnitude?

Deep understanding 1: Why is exp field encoded using bias?

exp field needs to encode both positive and negative exponents.

Why not just use one of the signed integer formats? 2's complement, 1s' complement, signed magnitude?

Answer: allows easy comparison of magnitudes by simply comparing bits.

Deep understanding 1: Why is exp field encoded using bias?

exp field needs to encode both positive and negative exponents.

Why not just use one of the signed integer formats? 2's complement, 1s' complement, signed magnitude?

Answer: allows easy comparison of magnitudes by simply comparing bits.

Consider hypothetical 8-bit floating point format (from the textbook)

1-bit sign, $k = 4$ -bit exp, 3-bit frac.

What is the decimal value of
0b1_0110_111?

What is the decimal value of
0b1_0111_000?

Deep understanding 1: Why is exp field encoded using bias?

exp field needs to encode both positive and negative exponents.

Why not just use one of the signed integer formats? 2's complement, 1s' complement, signed magnitude?

Answer: allows easy comparison of magnitudes by simply comparing bits.

Consider hypothetical 8-bit floating point format (from the textbook)

1-bit sign, $k = 4$ -bit exp, 3-bit frac.

What is the decimal value of
0b1_0110_111?

$$-1.875 \times 2^{-1}$$

What is the decimal value of
0b1_0111_000?

$$-2.000 \times 2^{-1}$$

IEEE 754: Why Use a Biased Exponent?

- ▶ **Biased Exponent Formula:** $EXP = E + \text{bias}$, where $\text{bias} = 2^{k-1} - 1$.
 - ▶ 8-bit Tiny Float ($k = 4$): $\text{bias} = 7$.
 - ▶ 32-bit Single Precision ($k = 8$): $\text{bias} = 127$.
- ▶ **Why NOT Two's Complement for the Exponent?**
 - ▶ Biased encoding makes the EXP field an **unsigned integer**.
 - ▶ **Hardware Reuse:** Allows floating-point numbers to be compared bit-by-bit using existing **unsigned integer comparison hardware**.
 - ▶ Larger positive floating-point numbers will always have a 1 appearing earlier in their bit representation.

Data Representation: Floating Point Mastery

Announcements

Programming assignment 3 and midterm exam

Floats: Special cases

Floats: Properties

Floating point multiplication

Floats: Understanding its design

Deep understanding 1: Why is exp field encoded using bias?

Deep understanding 2: Why have denormalized numbers?

Deep understanding 3: Why is bias chosen to be $2^{k-1} - 1$?

Floats: Summary

Computer organization: A primer

Assembly code: Human readable representation of machine code

Instruction set architectures

`swap.s`: Assembly implementation of function that swaps memory contents

Deep understanding 2: Why have denormalized numbers?

Why not just continue normalized number scheme down to smallest numbers around zero?

Answer: makes sure that smallest increments available are maintained around zero.

Suppose denormalized numbers NOT used.

What is the decimal value of obo_0000_001?

$$1.125 \times 2^{-7}$$

What is the decimal value of obo_0000_111?

$$1.875 \times 2^{-7}$$

What is the decimal value of obo_0001_000?

$$2.000 \times 2^{-7}$$

Deep understanding 2: Why have denormalized numbers?

Why not just continue normalized number scheme down to smallest numbers around zero?

Answer: makes sure that smallest increments available are maintained around zero.

Suppose denormalized numbers ARE used.

What is the decimal value of obo_0000_001?

$$0.125 \times 2^{-6}$$

What is the decimal value of obo_0000_111?

$$0.875 \times 2^{-6}$$

What is the decimal value of obo_0001_000?

$$1.000 \times 2^{-6}$$

Data Representation: Floating Point Mastery

Announcements

Programming assignment 3 and midterm exam

Floats: Special cases

Floats: Properties

Floating point multiplication

Floats: Understanding its design

Deep understanding 1: Why is exp field encoded using bias?

Deep understanding 2: Why have denormalized numbers?

Deep understanding 3: Why is bias chosen to be $2^{k-1} - 1$?

Floats: Summary

Computer organization: A primer

Assembly code: Human readable representation of machine code

Instruction set architectures

`swap.s`: Assembly implementation of function that swaps memory contents

Data Representation: Floating Point Mastery

Announcements

Programming assignment 3 and midterm exam

Floats: Special cases

Floats: Properties

Floating point multiplication

Floats: Understanding its design

Deep understanding 1: Why is exp field encoded using bias?

Deep understanding 2: Why have denormalized numbers?

Deep understanding 3: Why is bias chosen to be $2^{k-1} - 1$?

Floats: Summary

Computer organization: A primer

Assembly code: Human readable representation of machine code

Instruction set architectures

`swap.s`: Assembly implementation of function that swaps memory contents

Floats: Summary

	normalized	denormalized
value of number	$(-1)^s \times M \times 2^E$	$(-1)^s \times M \times 2^E$
E	E = exp - bias	E = -bias + 1
bias	$2^{k-1} - 1$	$2^{k-1} - 1$
exp	$0 < \text{exp} < (2^k - 1)$	$\text{exp} = 0$
M	M = 1.frac M has implied leading 1	M = 0.frac M has leading 0
	greater range large magnitude numbers denser near origin	greater precision small magnitude numbers evenly spaced

Table: Summary of normalized and denormalized numbers

Data Representation: Floating Point Mastery

Announcements

Programming assignment 3 and midterm exam

Floats: Special cases

Floats: Properties

Floating point multiplication

Floats: Understanding its design

Deep understanding 1: Why is exp field encoded using bias?

Deep understanding 2: Why have denormalized numbers?

Deep understanding 3: Why is bias chosen to be $2^{k-1} - 1$?

Floats: Summary

Computer organization: A primer

Assembly code: Human readable representation of machine code

Instruction set architectures

`swap.s`: Assembly implementation of function that swaps memory contents

Computer organization

Layer cake: remember the first day of class, we discussed what are parts of a computer?

- ▶ Society
- ▶ Human beings
- ▶ Applications
- ▶ Algorithms
- ▶ High-level programming languages

2026

▶ Interpreters

2020

C ▶ Low-level programming languages

hsm ○ ▶ Compilers

▶ Architectures

HLS ~2010

▶ Microarchitectures

~2010

RTL

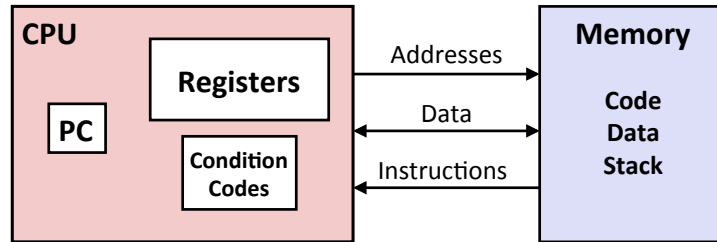
▶ Sequential/combinational logic

▶ Transistors

▶ Semiconductors

Stored program computer

Assembly/Machine Code View



Programmer-Visible State

- **PC: Program counter**
 - Address of next instruction
 - Called “RIP” (x86-64)
- **Register file**
 - Heavily used program data
- **Condition codes**
 - Store status information about most recent arithmetic or logical operation
 - Used for conditional branching

▪ Memory

- Byte addressable array
- Code and user data
- Stack to support procedures

Stored program:

Instructions reside in memory, loaded as needed.

von Neumann architecture:

Data and instructions share same connection to memory.

Memory hierarchy

	Capacity	Access speed
Internet		
Tape	250Pb	
Hard drives	16TB	2Mb/s
Solid state drives	4TB	2Gb/s
DRAM	8Gb - 1Tb+	8Gb/s
Last-level cache	64Mb	
Level-1 cache	1Mb	
Registers	1Kb	

- Registers (.25ns; 4GHz => .25e-9s)

Data Representation: Floating Point Mastery

Announcements

Programming assignment 3 and midterm exam

Floats: Special cases

Floats: Properties

Floating point multiplication

Floats: Understanding its design

Deep understanding 1: Why is exp field encoded using bias?

Deep understanding 2: Why have denormalized numbers?

Deep understanding 3: Why is bias chosen to be $2^{k-1} - 1$?

Floats: Summary

Computer organization: A primer

Assembly code: Human readable representation of machine code

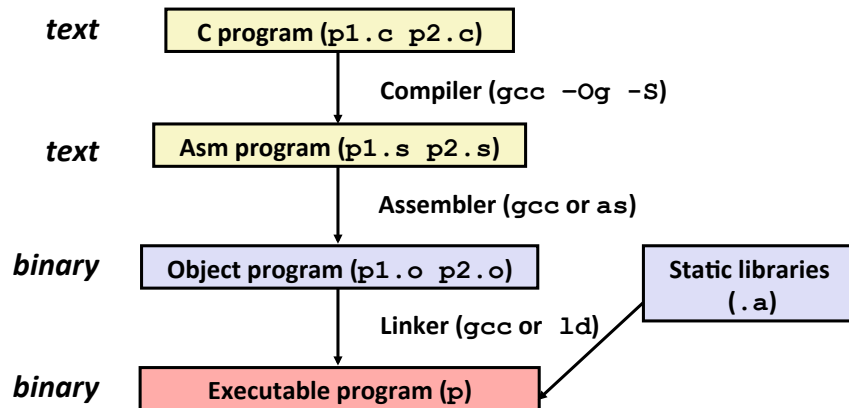
Instruction set architectures

`swap.s`: Assembly implementation of function that swaps memory contents

Unraveling the compilation chain

Turning C into Object Code

- Code in files `p1.c` `p2.c`
- Compile with command: `gcc -Og p1.c p2.c -o p`
 - Use basic optimizations (`-Og`) [New to recent versions of GCC]
 - Put resulting binary in file `p`



Assembly

Human readable machine code

- ▶ Very limited
- ▶ Not much control flow
- ▶ Any more complex functionality is built up
- ▶ for loops, while loops, turn into assembly sequence

Choice of what assembly to experiment with

- ▶ MIPS
- ▶ ARM
- ▶ x86 / x86-64 (not ideal for teaching, but it allows us to experiment on ilab)

Assembly instructions

Instructions for the microarchitecture

- ▶ Binary streams that tell an electronic circuit what to do
- ▶ Fetch, decode, execute, memory, writeback

A preview of microarchitecture

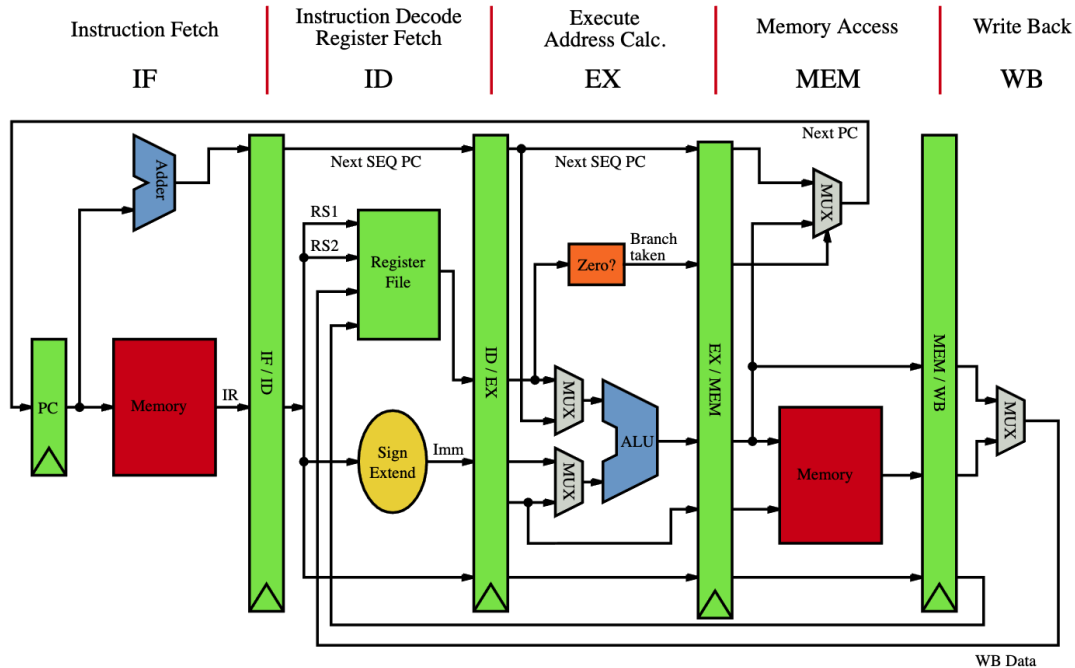


Figure: Stages of compilation. Image credit Wikimedia

Data Representation: Floating Point Mastery

Announcements

Programming assignment 3 and midterm exam

Floats: Special cases

Floats: Properties

Floating point multiplication

Floats: Understanding its design

Deep understanding 1: Why is exp field encoded using bias?

Deep understanding 2: Why have denormalized numbers?

Deep understanding 3: Why is bias chosen to be $2^{k-1} - 1$?

Floats: Summary

Computer organization: A primer

Assembly code: Human readable representation of machine code

Instruction set architectures

`swap.s`: Assembly implementation of function that swaps memory contents

Why are instruction set architectures important

Interface between computer science and electrical and computer engineering

- ▶ Software is varied, changes
- ▶ Hardware is standardized, static

Computer architect Fred Brooks and the IBM 360

- ▶ IBM was selling computers with different capacities,
- ▶ Compile once, and can run software on all IBM machines.
- ▶ Backward compatibility.
- ▶ An influential idea.

CISC vs. RISC

Complex instruction set computer

- ▶ Intel and AMD
- ▶ Have an extensive and complex set of instructions
- ▶ For example: x86's extensions: x87, IA-32, x86-64, MMX, 3DNow!, SSE, SSE2, SSE3, SSSE3, SSE4, SSE4.2, SSE5, AES-NI, CLMUL, RDRAND, SHA, MPX, SGX, XOP, F16C, ADX, BMI, FMA, AVX, AVX2, AVX512, VT-x, VT-d, AMD-V, AMD-Vi, TSX, ASF
- ▶ Can license Intel's compilers to extract performance
- ▶ Secret: inside the processor, they break it down to more elementary instructions

CISC vs. RISC

Reduced instruction set computer

- ▶ MIPS, ARM, RISC-V (can find Patterson and Hennessy Computer Organization and Design textbook in each of these versions), and PowerPC
- ▶ Have a relatively simple set of instructions
- ▶ For example: ARM's extensions: SVE;SVE2;TME; All mandatory: Thumb-2, Neon, VFPv4-D16, VFPv4 Obsolete: Jazelle
- ▶ ARM: smartphones, Apple ARM M1 Mac

Into the future: Post-ISA world

Post-ISA world

- ▶ Increasingly, the CPU is not the only character
- ▶ It orchestrates among many pieces of hardware
- ▶ Smartphone die shot
- ▶ GPU, TPU, FPGA, ASIC

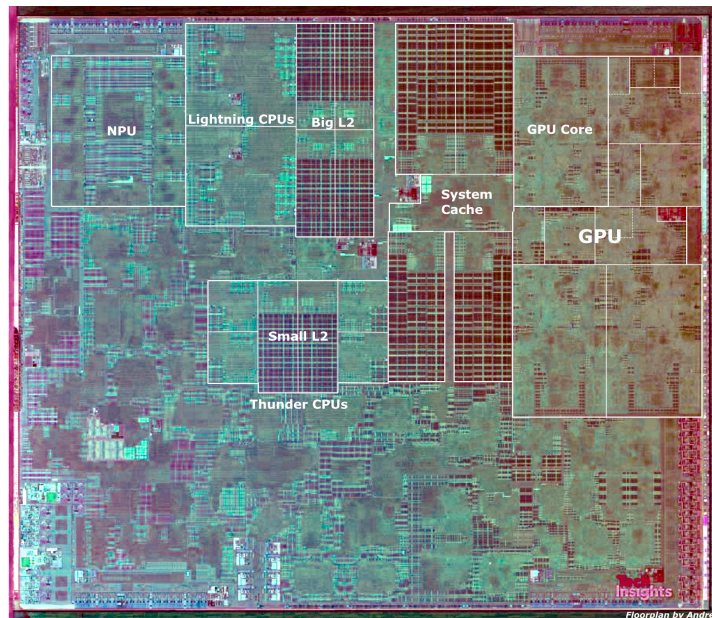


Figure: Apple A13 (2019 Apple iPhone 11 CPU). Image credit AnandTech

Data Representation: Floating Point Mastery

Announcements

Programming assignment 3 and midterm exam

Floats: Special cases

Floats: Properties

Floating point multiplication

Floats: Understanding its design

Deep understanding 1: Why is exp field encoded using bias?

Deep understanding 2: Why have denormalized numbers?

Deep understanding 3: Why is bias chosen to be $2^{k-1} - 1$?

Floats: Summary

Computer organization: A primer

Assembly code: Human readable representation of machine code

Instruction set architectures

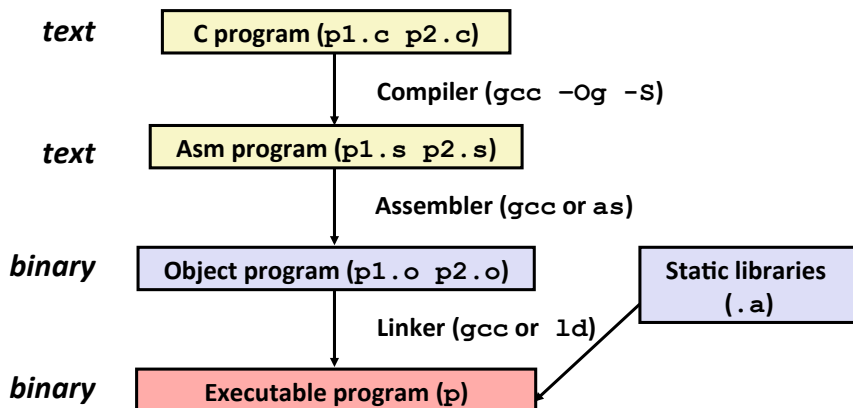
`swap.s`: Assembly implementation of function that swaps memory contents

Unraveling the compilation chain

Carnegie Mellon

Turning C into Object Code

- Code in files `p1.c` `p2.c`
- Compile with command: `gcc -Og p1.c p2.c -o p`
 - Use basic optimizations (`-Og`) [New to recent versions of GCC]
 - Put resulting binary in file `p`



▶ `gcc -Og -S`
`swap.c`

▶ `objdump -d`
`swap`

Let's go to CS:APP
textbook lecture
slides (05-machine-
basics.pdf) slide 28

Data movement instructions

Does unsigned / signed matter?

1. `void swap_uc ( unsigned char*a, unsigned char*b );`
2. `void swap_sc ( signed char*a, signed char*b );`

Swapping different data sizes

1. `void swap_c ( char*a, char*b );`
2. `void swap_s ( short*a, short*b );`
3. `void swap_i ( int*a, int*b );`
4. `void swap_l ( long*a, long*b );`