

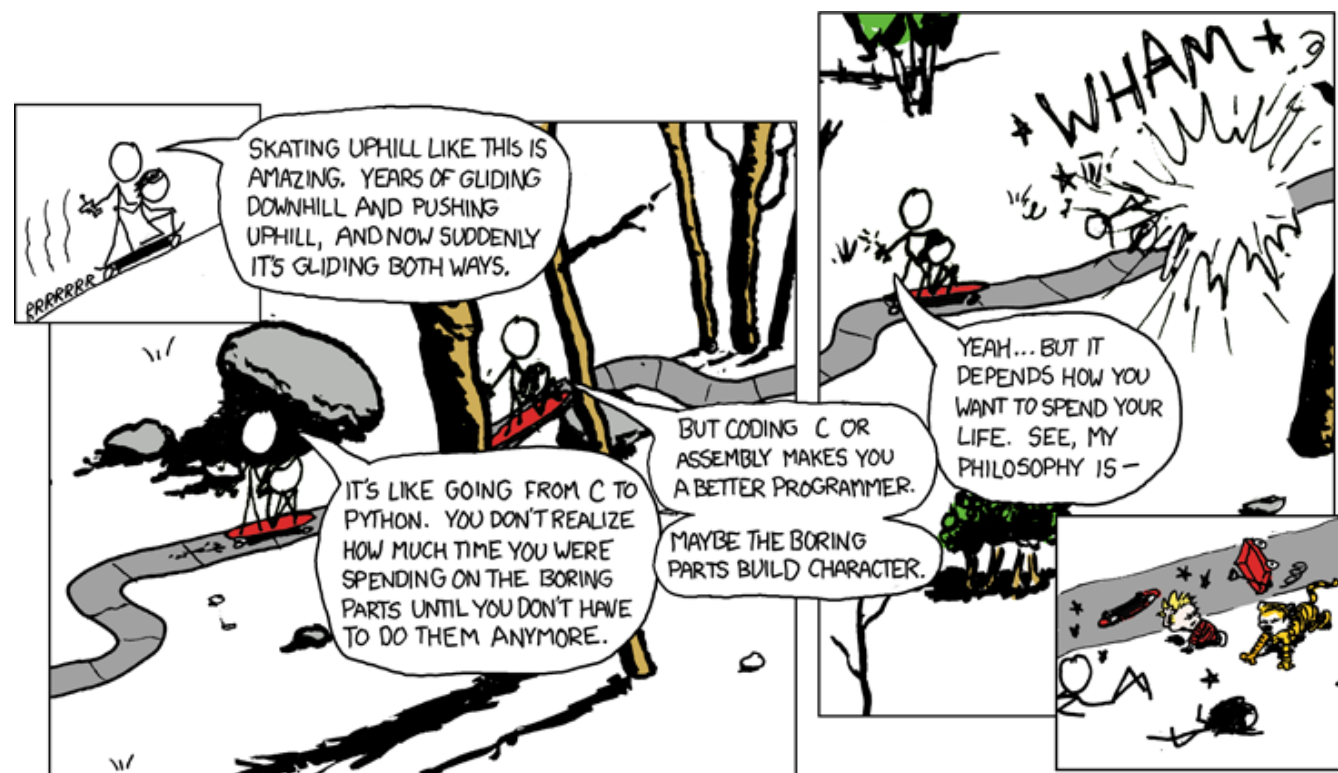
x86-64 Programming I

CSE 351 Winter 2024

Instructor:
Justin Hsia

Teaching Assistants:

Adithi Raghavan
Aman Mohammed
Connie Chen
Eyoel Gebre
Jiawei Huang
Malak Zaki
Naama Amiel
Nathan Khuat
Nikolas McNamee
Pedro Amarante
Will Robertson



<http://xkcd.com/409/>

Relevant Course Information

- ❖ HW5 due tonight, HW6 due Monday, HW7 due Wednesday
- ❖ Lab 1a grades hopefully released by end of Sunday (1/21)
- ❖ Lab 1b due Monday (1/22) at 11:59 pm
 - No major programming restrictions, but should avoid magic numbers by using C macros (`#define`)
 - For debugging, can use provided utility functions `print_binary_short()` and `print_binary_long()`
 - Pay attention to the output of `aisle_test` and `store_test` – failed tests will show you actual vs. expected
 - You have *late day tokens* available

Getting Help with 351

- ❖ Lecture recordings, lessons, inked slides, section worksheet solutions
- ❖ Attend lectures and support hours
 - Can also chat with other students– help each other learn!
- ❖ Form a study group!
 - Good for everything but labs, which should be done in pairs
 - Communicate regularly, use the class terminology, ask and answer each others' questions, show up to SH together
- ❖ Post on Ed Discussion
- ❖ Request a 1-on-1 meeting
 - Available on a limited basis for special circumstances

A detailed, colorful micrograph of a microchip die, showing a complex grid of circuitry and various colored regions. The text 'x86-64 Programming I' is overlaid in the center.

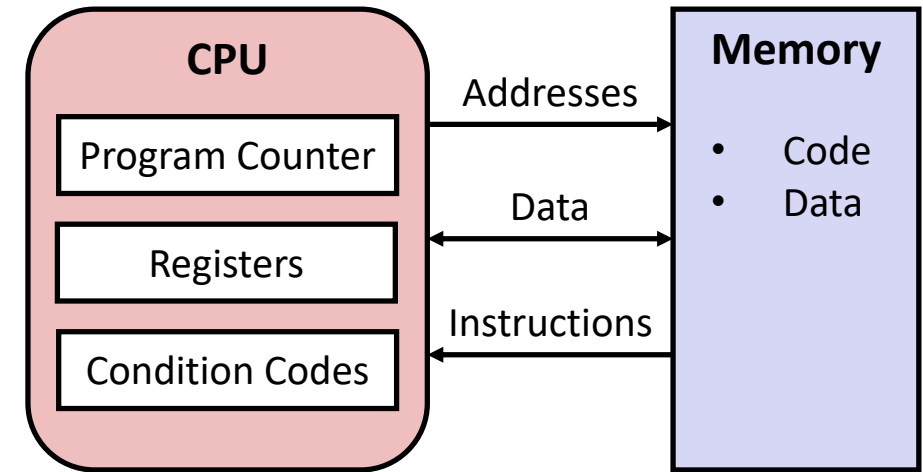
x86-64 Programming I

Lesson Summary (1/2)

❖ Assembly programmer-visible state:

❖ x86-64 is a *complex instruction set computing* (CISC) architecture

- x86-64 integer instruction common forms: **instr op** and **instr src, dst**
 - Fixed width specified by size suffix: b (1 byte), w (2 bytes), l (4 bytes), or q (8 bytes)
- Instruction types:
 - *Data transfer* (e.g., **movq** (%rsi), %rdx)
 - *Arithmetic* (e.g., **imulq** \$3, %rsi)
 - *Control Flow* (e.g., **ret**)



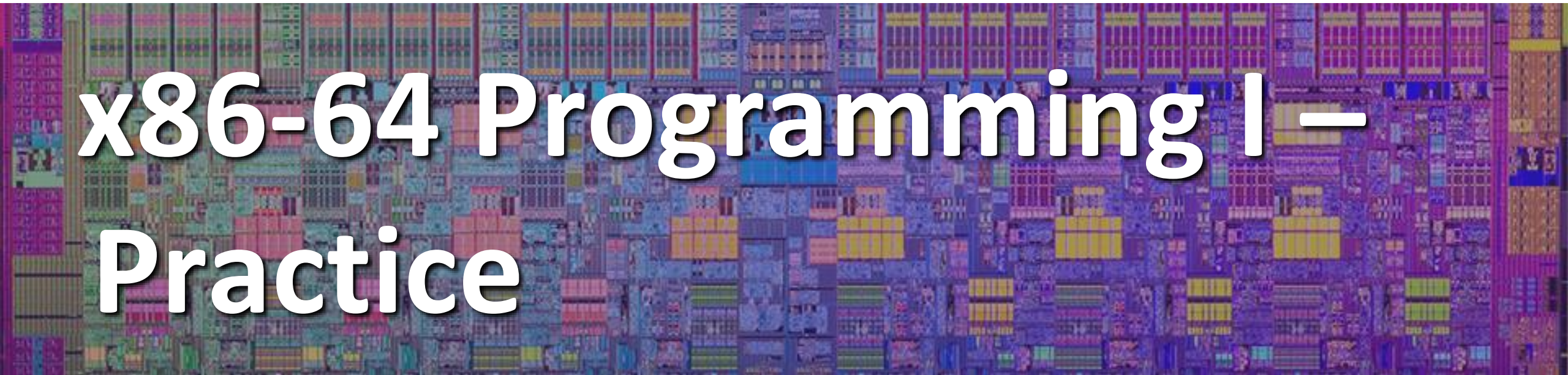
Lesson Summary (2/2)

- ❖ x86-64 is a *complex instruction set computing* (CISC) architecture
 - x86-64 integer instruction common forms: **instr op** and **instr src, dst**
 - Fixed width specified by size suffix: b (1 byte), w (2 bytes), l (4 bytes), or q (8 bytes)
 - Operand types:
 - Immediate (\$) is a literal (e.g., `imulq $3, %rsi`)
 - Register (%) is a general-purpose integer register or sub-register (e.g., `movq (%rsi), %rdx`)
 - Memory (()) is a way to express an address (e.g., `movq (%rsi), %rdx`)

<u>%rax</u>	<u>%eax</u>	<u>%ax</u>	<u>%al</u>	<u>%r8</u>	<u>%r8d</u>	<u>%r8w</u>	<u>%r8b</u>
<u>%rbx</u>	<u>%ebx</u>	<u>%bx</u>	<u>%bl</u>	<u>%r9</u>	<u>%r9d</u>	<u>%r9w</u>	<u>%r9b</u>
<u>%rcx</u>	<u>%ecx</u>	<u>%cx</u>	<u>%cl</u>	<u>%r10</u>	<u>%r10d</u>	<u>%r10w</u>	<u>%r10b</u>
<u>%rdx</u>	<u>%edx</u>	<u>%dx</u>	<u>%dl</u>	<u>%r11</u>	<u>%r11d</u>	<u>%r11w</u>	<u>%r11b</u>
<u>%rsi</u>	<u>%esi</u>	<u>%si</u>	<u>%sil</u>	<u>%r12</u>	<u>%r12d</u>	<u>%r12w</u>	<u>%r12b</u>
<u>%rdi</u>	<u>%edi</u>	<u>%di</u>	<u>%dil</u>	<u>%r13</u>	<u>%r13d</u>	<u>%r13w</u>	<u>%r13b</u>
<u>%rsp</u>	<u>%esp</u>	<u>%sp</u>	<u>%spl</u>	<u>%r14</u>	<u>%r14d</u>	<u>%r14w</u>	<u>%r14b</u>
<u>%rbp</u>	<u>%ebp</u>	<u>%bp</u>	<u>%bpl</u>	<u>%r15</u>	<u>%r15d</u>	<u>%r15w</u>	<u>%r15b</u>
8 bytes	4 bytes	2 bytes	1 byte	8 bytes	4 bytes	2 bytes	1 byte

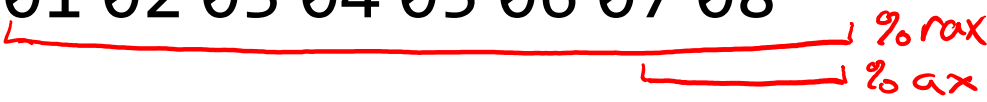
Lesson Q&A

- ❖ Learning Objectives:
 - Without executing, describe the overall purpose of snippets of x86-64 assembly code containing arithmetic, [if-else statements, and/or loops].
- ❖ What lingering questions do you have from the lesson?
 - Chat with your neighbors about the lesson for a few minutes to come up with questions

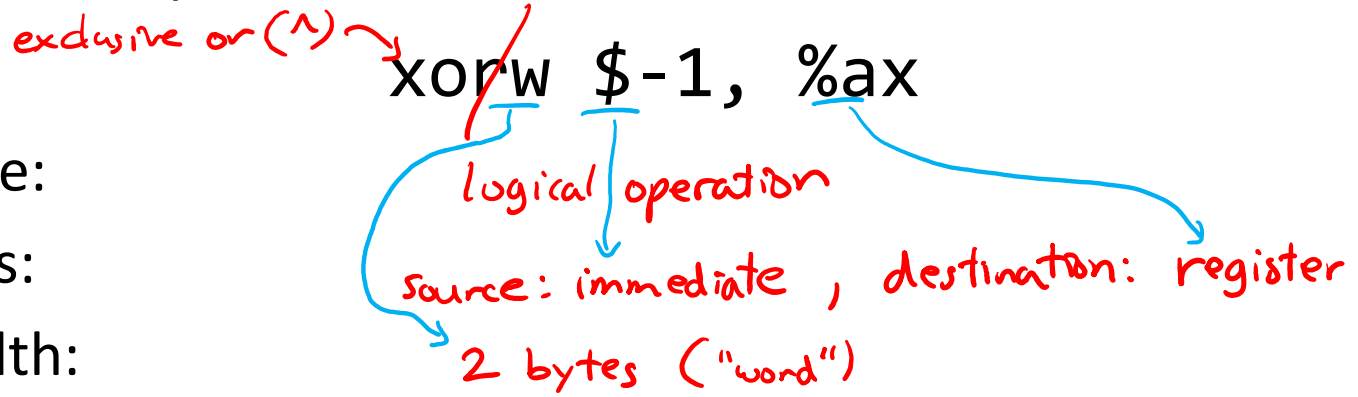
A detailed, colorful micrograph of a microchip die, showing intricate circuit patterns in shades of purple, blue, yellow, and red. The text is overlaid on this background.

x86-64 Programming I – Practice

Polling Questions (1/2)

❖ Assume that the register `%rax` currently holds the value
`0x 01 02 03 04 05 06 07 08`


❖ Answer the questions on Ed Lessons about the following instruction
 (<instr> <src> <dst>):



- Operation type:
- Operand types:
- Operation width:
- (extra) Result in `%rax`:

$$\begin{array}{r}
 0x\ 07\ 08 \\
 \wedge\ 0x\ FF\ FF \\
 \hline
 0x\ F8\ F7 \Rightarrow \%rax: \boxed{0x\ 01\ 02\ 03\ 04\ 05\ 06\ F8\ F7}
 \end{array}$$

Polling Questions (2/2)

- ❖ Which of the following are valid implementations of $rcx = rax + rbx$?

~~❑~~ `addq %rax, %rcx`
`addq %rbx, %rcx`
 $rcx = rcx + rax + rbx$

✓ `movq %rax, %rcx`
`addq %rbx, %rcx`
 $rcx = rax + rbx$

✓ `movq $0, %rcx`
`addq %rbx, %rcx`
`addq %rax, %rcx`
 $rcx = 0 + rbx + rax$

~~❑~~ `xorq %rax, %rax`
`addq %rax, %rcx`
`addq %rbx, %rcx`
 $rcx = rcx + 0 + rbx$ ($rax = 0$)

Homework Setup

❖ Do the following operand types have an implied size?

- An immediate operand is a literal/constant (e.g., \$3)

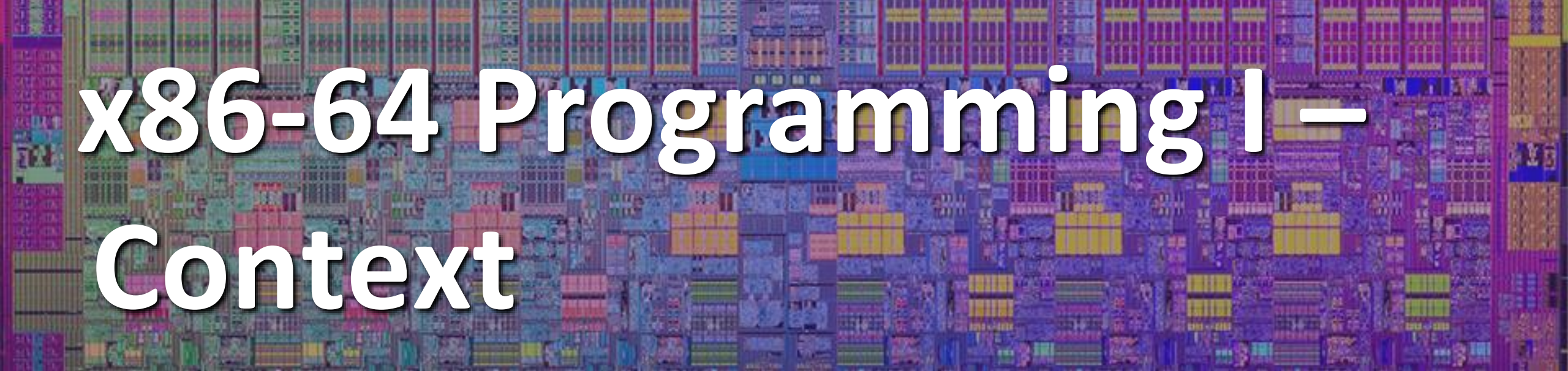
No, could be 0x03, 0x0003, 0x00000003, etc.

- A register operand is the value stored in a register (e.g., %rdx)

Yes, look up in register table (%rdx is 8 bytes wide)

- A memory operand represents an address in memory (e.g., (%rsi))

*No, address just gives us the starting point of the data
(the address itself is a word size, though)*

A detailed, colorful micrograph of a microchip die, showing intricate circuit patterns in shades of purple, blue, yellow, and green. The text is overlaid on this background.

x86-64 Programming I – Context

Instruction Set Philosophies, Revisited

- ❖ *Complex Instruction Set Computing (CISC):*
Add more and more elaborate and specialized instructions as needed
 - **Design goals:** complete tasks in as few instructions as possible; minimize memory accesses for instructions
- ❖ *Reduced Instruction Set Computing (RISC):*
Keep instruction set small and regular
 - **Design goals:** build fast hardware; instructions should complete in few clock cycles (ideally 1); minimize complexity and maximize performance
- ❖ How different are these two philosophies, really?

Instruction Set Philosophies, Revisited

- ❖ *Complex Instruction Set Computing (CISC):*
Add more and more elaborate and specialized instructions as needed
 - **Design goals:** complete tasks in **as few instructions as possible**; **minimize** memory accesses for instructions
- ❖ *Reduced Instruction Set Computing (RISC):*
Keep instruction set small and regular
 - **Design goals:** build **fast** hardware; instructions should complete in **few clock cycles** (ideally 1); **minimize complexity** and **maximize performance**
- ❖ How different are these two philosophies, really?
 - Both pursue **efficiency** (**minimalism** is a means to an end)

Mainstream ISAs, Revisited



x86

Designer	Intel, AMD
Bits	16-bit, 32-bit and 64-bit
Introduced	1978 (16-bit), 1985 (32-bit), 2003 (64-bit)
Design	CISC
Type	Register-memory
Encoding	Variable (1 to 15 bytes)
Branching	Condition code
Endianness	Little

Macbooks & PCs
(Core i3, i5, i7, M)
[x86-64 Instruction Set](#)

ARM

Designer	ARM
Bits	32-bit, 64-bit
Introduced	1985
Design	RISC
Type	Register
Encoding	AArch64/A64 and AArch32/A32 use 32-bit instructions, T32 (Thumb-2) uses mixed 16- and 32-bit instructions
Branching	Condition code, compare and branch
Endianness	Bi (little as default)

Smartphone-like devices
(iPhone, iPad, Raspberry Pi)
[ARM Instruction Set](#)

RISC-V

Designer	University of California, Berkeley
Bits	32 · 64 · 128
Introduced	2010
Design	RISC
Type	Load-store
Encoding	Variable
Endianness	Little ^{[1][3]}

Mostly research
(some traction in embedded)
[RISC-V Instruction Set](#)

Does anything
feel "off" about
this landscape?

Tech Monopolization

- ❖ How many “dominant” ISAs are there?
 - 2: x86, ARM
- ❖ How many “dominant” phone brands are there?
 - 4: Samsung, Apple, Huawei, Xiaomi
- ❖ How many “dominant” operating systems are there?
 - 3/4: Android, iOS/macOS, Windows, Linux (?)
- ❖ How many “dominant” chip manufacturers are there?
 - 3: Intel, Samsung, TSMC
- ❖ It wasn't always this way!
 - Combination of antitrust policies and (lack of) enforcement

Discussion Questions

- ❖ Discuss the following question(s) in groups of 3-4 students
 - I will call on a few groups afterwards so please be prepared to share out
 - Be respectful of others' opinions and experiences

- ❖ How do you feel about tech monopolization?
 - What are the benefits and disadvantages of this landscape for (1) the monopolizing companies and (2) the consumers?

 - These big tech companies are now worth billions of dollars. What might we try if we wanted to break up the monopolization?

Group Work Time

- ❖ During this time, you are encouraged to work on the following:
 - 1) If desired, continue your discussion
 - 2) Work on the homework problems
 - 3) Work on the lab (if applicable)

- ❖ Resources:
 - You can revisit the lesson material
 - Work together in groups and help each other out
 - Course staff will circle around to provide support