

x86-64 Programming II

CSE 351 Autumn 2023

Instructor:

Justin Hsia

Teaching Assistants:

Afifah Kashif

Malak Zaki

Bhavik Soni

Naama Amiel

Cassandra Lam

Nayha Auradkar

Connie Chen

Nikolas McNamee

David Dai

Pedro Amarante

Dawit Hailu

Renee Ruan

Ellis Haker

Simran Bagaria

Eyoel Gebre

Will Robertson

Joshua Tan



<http://xkcd.com/99/>

Relevant Course Information

- ❖ hw7 due Monday, hw8 due Wednesday

- ❖ Lab 1b due Monday (10/16) 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

A detailed, colorful micrograph of a microchip die, showing a complex grid of circuitry and various colored regions (purple, blue, yellow, green, red) representing different functional blocks.

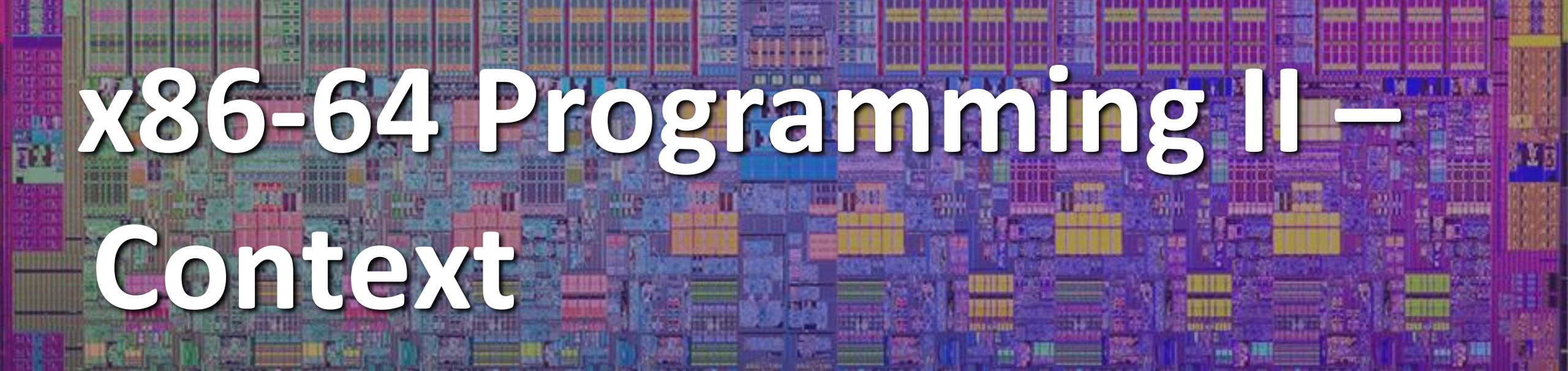
x86-64 Programming II

Lesson Summary (1/2)

- ❖ **Memory Addressing Modes:** The addresses used for accessing memory in `mov` (and other) instructions can be computed in several different ways
 - $D(Rb, Ri, S)$ with *base register*, *index register*, *scale factor*, and *displacement* compute the address $\text{Reg}[Rb] + \text{Reg}[Ri]*S + D$ and is usually dereferenced by instructions
 - These map well to pointer arithmetic operations
- ❖ **Load effective address (`lea`)** instruction used to compute addresses and perform basic arithmetic
 - *Doesn't* dereference the source memory operand, unlike all other instructions!
- ❖ **Extension instructions** (`movz`, `movs`) allow us to zero and sign extend data into longer widths

Lesson Summary (2/2)

- ❖ Terminology:
 - Memory Operand: displacement, base register, index register, scale factor
 - Extension instructions (movz, movs)
 - Address computation instruction (lea)
- ❖ Learning Objectives:
 - Without executing, describe the overall purpose of snippets of x86-64 assembly code containing arithmetic, [if-else statements, and/or loops].
 - Use GDB tools to step through a running program and extract debugging information from a program's disassembly, the state of registers, and values at specific memory locations.
- ❖ What lingering questions do you have from the lesson?

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 II – Context

Extension Instructions (Review)

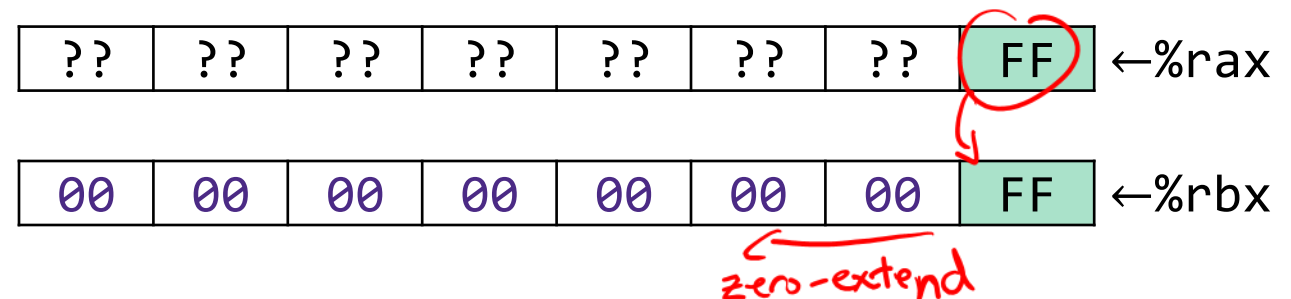
2 width specifiers: b, w, l, q
 1 2 4 8 bytes

- ❖ `movz__ src, dst` # Move with zero extension
- `movs__ src, dst` # Move with sign extension
- Copy from a smaller source value to a larger destination
 - First suffix letter is size of source, second suffix letter is size of destination
 - Recall: zero-extension always fills with 0, sign-extension fills with copy of the sign bit
- `src` can be Mem or Reg; `dst` must be Reg

❖ Example: data shown in hex

■ `movzq %a1, %rbx`

zero-extend ↑
 1 byte → 8 bytes



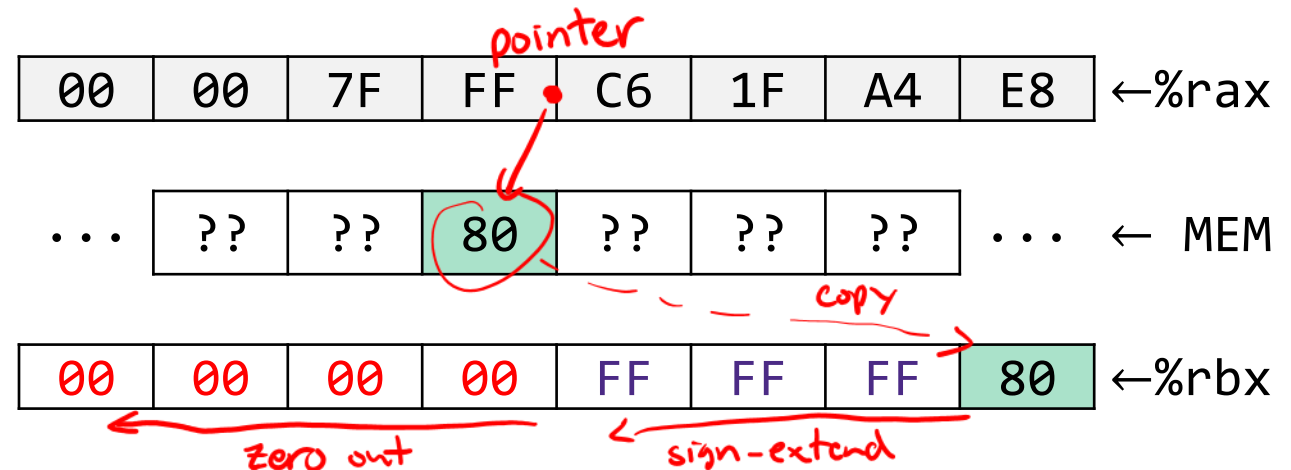
Extension Instructions (Review)

- ❖ `movz__ src, dst` # Move with zero extension
- `movs__ src, dst` # Move with sign extension
- Copy from a smaller source value to a larger destination
 - First suffix letter is size of source, second suffix letter is size of destination
 - Recall: zero-extension always fills with 0, sign-extension fills with copy of the sign bit
- `src` can be Mem or Reg; `dst` must be Reg

❖ Example: data shown in hex

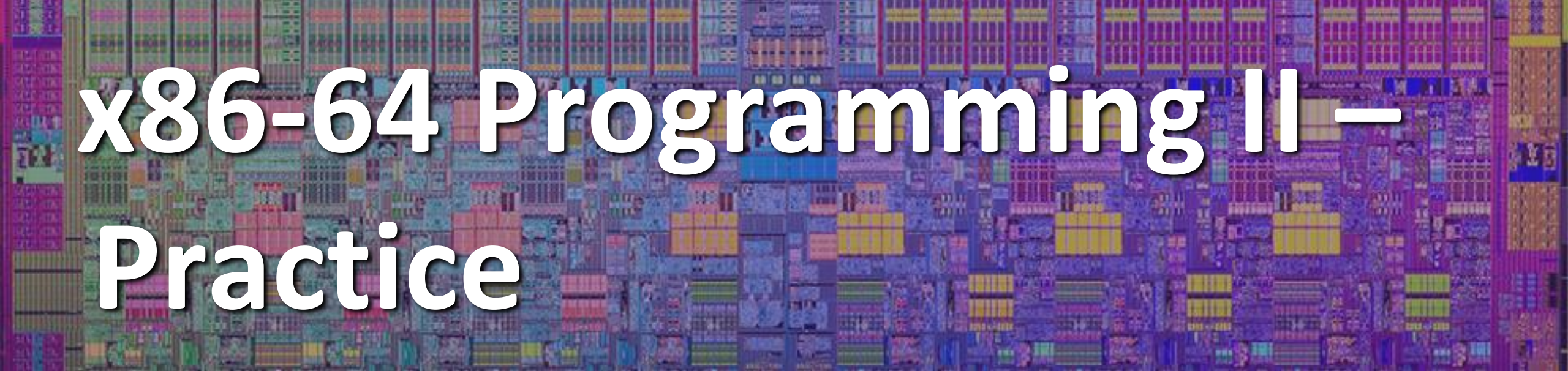
- `movsbl` ^{4 bytes} (%rax), %ebx
sign-extend ↗ *1 byte from memory* ↘

Recall, any x86-64 instruction that stores into a 32-bit (suffix `l`) register zeros out the upper 4 bytes of the register.



GDB Demo

- ❖ The `movz` and `movs` examples on a real machine!
 - `movzbq %a1, %rbx`
 - `movsb1 (%rax), %ebx`
- ❖ You will need to use GDB to get through Lab 2
 - Useful debugger in this class and beyond!
- ❖ Pay attention to:
 - Setting breakpoints (`break`)
 - Stepping through code (`step/next` and `stepi/nexti`)
 - Printing out expressions (`print` – works with regs & vars)
 - Examining memory (`x`)

A detailed, colorful micrograph of a microchip die, showing intricate circuitry and various colored regions in shades of purple, blue, yellow, and red.

x86-64 Programming II – Practice

Group Work Time

- ❖ During this time, you are encouraged to work on the following:
 - 1) If desired, continue your discussion
 - 2) Work on the lesson problems (solutions at the end of class)
 - 3) Work on the homework problems

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

Practice Questions (1/2)

- ❖ $D(Rb, Ri, S)$ computes address $Reg[Rb] + Reg[Ri] * S + D$
 - Likely will get dereferenced, but that's up to the instruction
 - Default values: $D = 0$, $Reg[Rb] = 0$, $Reg[Ri] = 0$, $S = 1$
- ❖ Assuming `%rdx` contains `0xF000` and `%rcx` contains `0x100`, what addresses are computed by the following memory operands?

■ $0x8(\overset{D}{}, \overset{Rb}{\%rdx})$

$$Reg[Rb] + D = 0xf000 + 0x8 = 0xf008$$

■ $(\overset{Rb}{\%rdx}, \overset{Ri}{\%rcx})$

$$Reg[Rb] + Reg[Ri] * 1 = 0xf000 + 0x100 = 0xf100$$

■ $(\overset{Rb}{\%rdx}, \overset{Ri}{\%rcx}, \overset{S}{4})$

$$Reg[Rb] + Reg[Ri] * 4 = 0xf000 + 0x400 = 0xf400$$

■ $0x80(\overset{D}{}, \overset{Ri}{\%rdx}, \overset{S}{2})$

$$Reg[Ri] * 2 + 0x80 = 0x1000 * 2 + 0x80 = 0x1e080$$

$$0xf000 * 2 \\ 0xf000 \ll 1 = 0x1e000$$

Practice Questions (2/2)

❖ Which of the following x86-64 instructions correctly calculates $\%rax = 9 * \%rdi$?

A. ~~leaq~~ (~~,~~ ~~%rdi~~, ~~9~~), ~~%rax~~

B. ~~movq~~ (~~,~~ ~~%rdi~~, ~~9~~), ~~%rax~~

C. leaq (~~%rdi~~, ~~%rdi~~, 8), %rax

D. ~~movq~~ (~~%rdi~~, ~~%rdi~~, 8), %rax

*no memory access, so must be lea
 $S \in \{1, 2, 4, 8\}$
invalid syntax*

invalid syntax

*$\%rax = 9 * \%rdi$*

*$\%rax = \text{Mem}[9 * \%rdi]$*