x86-64 Programming II

CSE 351 Autumn 2023

Instructor:

Justin Hsia

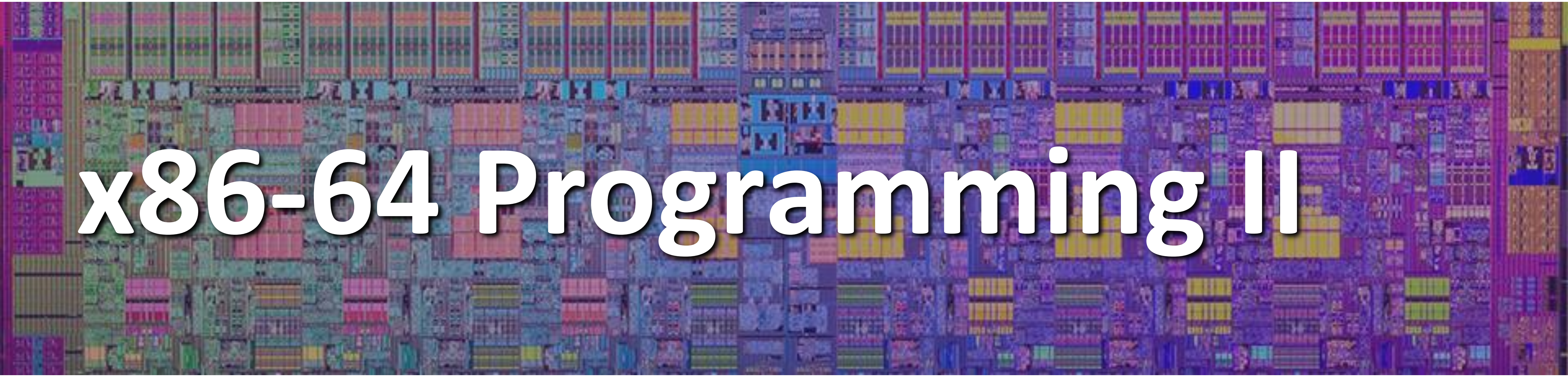
Teaching Assistants:

Afifah Kashif
Bhavik Soni
Cassandra Lam
Connie Chen
David Dai
Dawit Hailu
Ellis Haker
Eyoel Gebre
Joshua Tan
Malak Zaki
Naama Amiel
Nayha Auradkar
Nikolas McNamee
Pedro Amarante
Renee Ruan
Simran Bagaria
Will Robertson

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



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 Reg[Rb] + 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?

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
  - `movzbq %al, %rbx`

1 byte

zero-extend

8 bytes

??	??	??	??	??	??	??	FF	←%rax
00	00	00	00	00	00	00	FF	←%rbx

zero-extend

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 (%rax), %ebx`

4 bytes

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.

pointer

00	00	7F	FF	C6	1F	A4	E8	←%rax

...	??	??	80	??	??	??	...	← MEM

copy

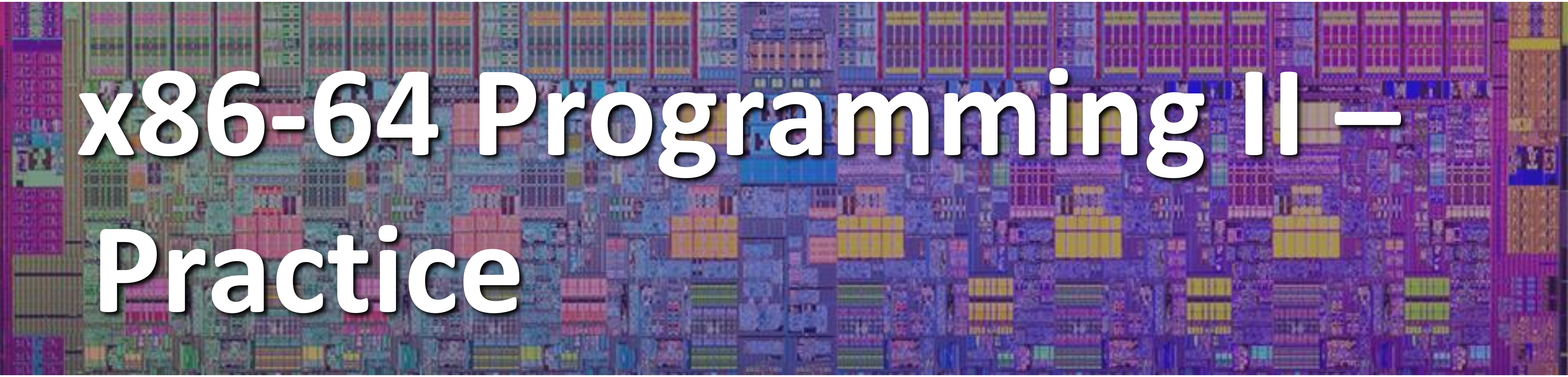
00	00	00	00	FF	FF	FF	80	←%rbx

zero out

sign-extend

GDB Demo

- The `movz` and `movs` examples on a real machine!
  - `movzbq %al, %rbx`
  - `movsbl (%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`)



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(%rdx)`
  - `(%rdx,%rcx)`
  - `(%rdx,%rcx,4)`
  - `0x80(,%rdx,2)`

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`