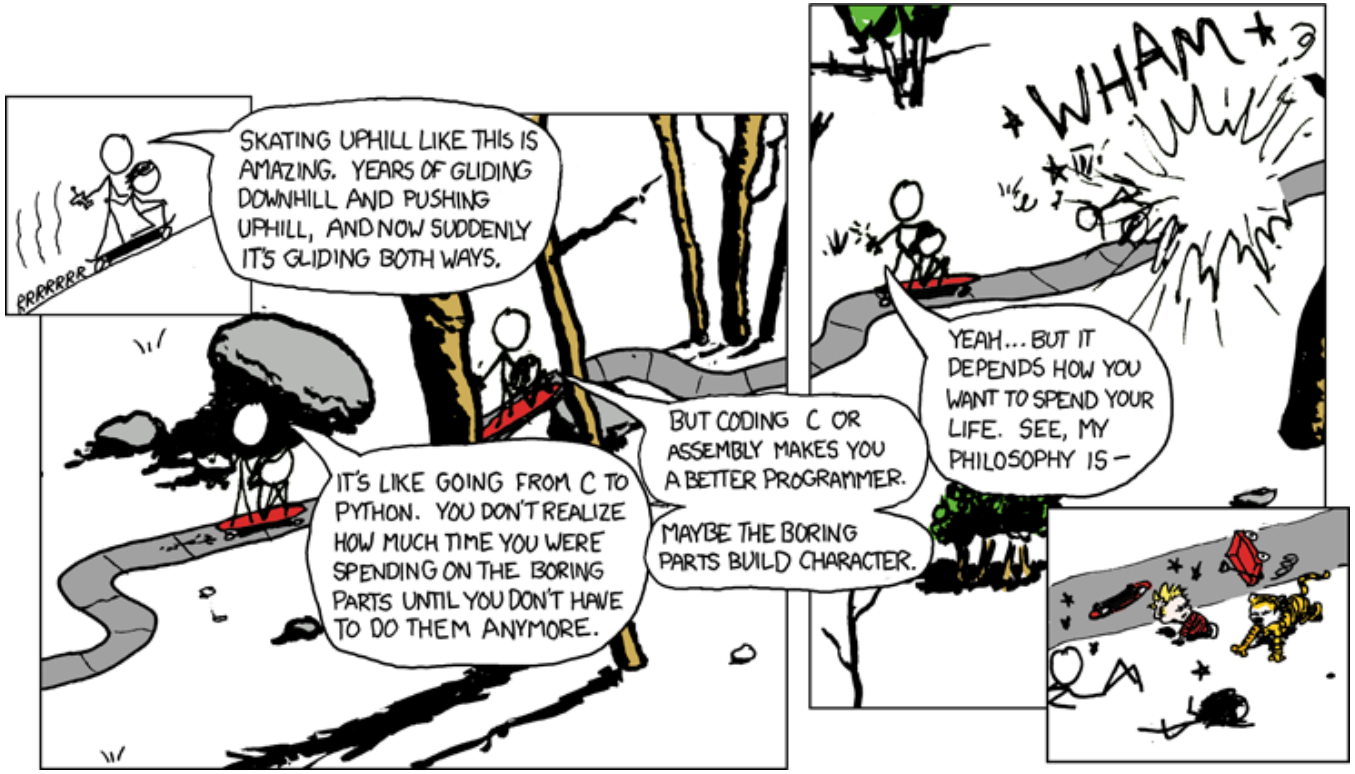
x86-64 Programming I

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

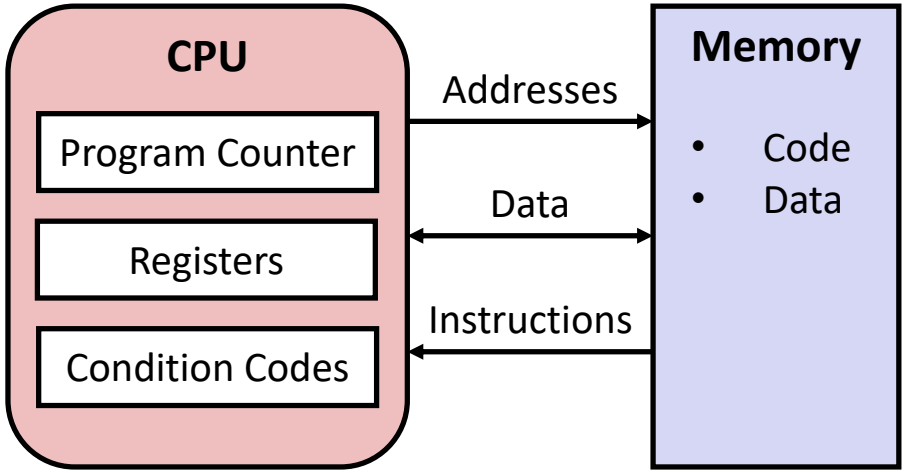
Getting Help with 351

- 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

x86-64 Programming I

Lesson Summary (1/2)

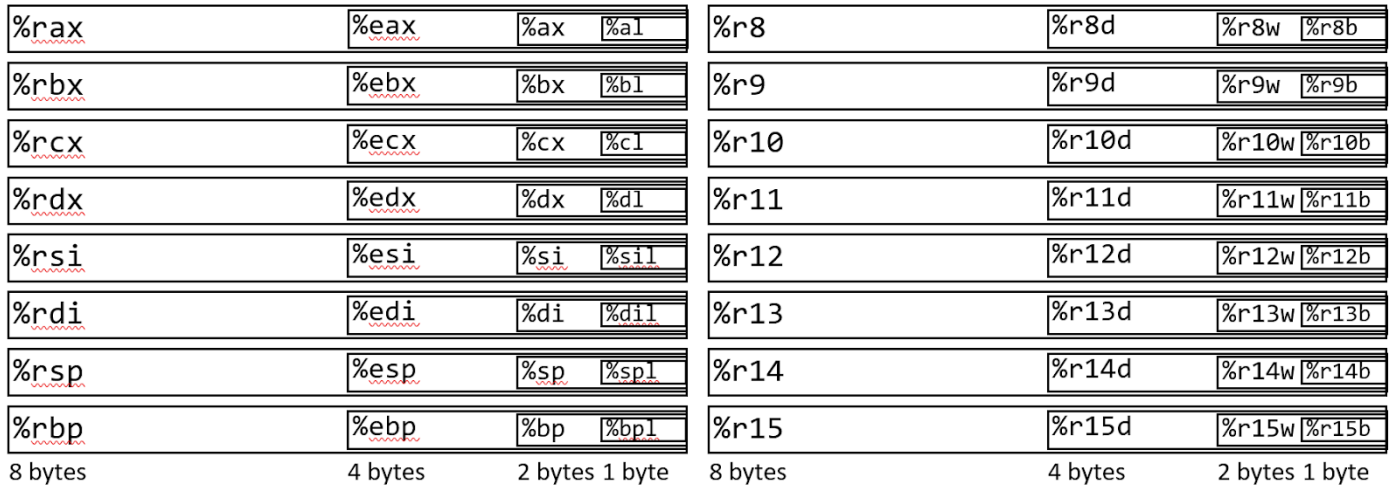
- Assembly programmer-visible state:



- 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`)



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

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>):

```
xorw $-1, %ax
```

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

Polling Questions (2/2)

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

  - ```
    addq %rax, %rcx
    addq %rbx, %rcx
    ```
  - ```
    movq %rax, %rcx
    addq %rbx, %rcx
    ```
  - ```
    movq $0, %rcx
    addq %rbx, %rcx
    addq %rax, %rcx
    ```
  - ```
    xorq %rax, %rax
    addq %rax, %rcx
    addq %rbx, %rcx
    ```

Homework Setup

- Do the following operand types have an implied size?
  - An immediate operand is a literal/constant (*e.g.*, `$3`)
  - A register operand is the value stored in a register (*e.g.*, `%rdx`)
  - A memory operand represents an address in memory (*e.g.*, `(%rsi)`)

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?

Mainstream ISAs, Revisited

intel arm RISC-V

	x86	ARM	RISC-V
Designer	Intel, AMD	A	University of California, Berkeley
Bits	16-bit, 32-bit and 64-bit	32-bit, 64-bit	32 · 64 · 128
Introduced	1978 (16-bit), 1985 (32-bit), 2003 (64-bit)	1985	2010
Design	CISC	RISC	RISC
Type	Register–memory	Register	Load-store
Encoding	Variable (1 to 15 bytes)	AArch64/A64 and AArch32/A32 use 32-bit instructions, T32 (Thumb-2) uses mixed 16- and	Variable
Branching	Condition code	Condition code, compare and branch	
Endianness	Little	Bi (little as default)	Little[1][3]

Does anything feel "off" about this landscape?

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

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

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

Tech Monopolization (blank)

- How many “dominant” ISAs are there?
- How many “dominant” phone brands are there?
- How many “dominant” operating systems are there?
- How many “dominant” chip manufacturers are there?

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