Introduction

Who am I?

▶ Dr Russ Ross
  ▶ From Southern Utah, went to high school in St. George
  ▶ Undergrad at Harvard (AB in Computer Science)
  ▶ Worked in Boston area during the .com boom
  ▶ Graduate school at Cambridge (PhD in Computer Science)
  ▶ At DSU since 2007
▶ Call me “Russ” or “Dr Ross” (how to choose?)
  ▶ Never “Mr Ross”—we are not in high school

Who are you?

▶ CS? SE? Something else? Undecided?
  ▶ Computer Science vs Software Engineering
Overview

What is this class about?

- The hardware-software interface
  - CPUs, ISAs, ABIs
  - Memory layout: segments, the stack, the heap
  - Assembly language and how it relates to high-level languages
- Data representation
  - Binary numbers, bytes, words
  - 2s-complement numbers
  - IEEE-754 floats
  - Arrays, structs, pointers
  - Strings
  - Instructions, functions
- Other low-level details programmers need to know about
  - Cache
  - Virtual memory
  - Memory allocation, garbage collection
  - User mode, kernel mode, system calls
The plan

1. Learn C (why C?)
   ▶ *Not C++*
2. Learn about CPU architecture
   ▶ Our example will be RISC-V
3. Use C to write a CPU simulator
   ▶ We will learn about hardware implementations, too
4. Write assembly language for that CPU
   ▶ Nothing like a little practice
### Attendance, distractions, etc.

- Attendance is not required in that you will not be graded for being here
  - Exception: excessive absence without making arrangements will result in failing (see the syllabus)
- You are responsible for what we talk about in class, and much of what we cover will *not* be available elsewhere
  - Assignment instructions, tips, etc.
  - If you miss class, you may not be able to complete the homework
- I will try to record classes, but the AV system is flaky and will probably fail on some days
  - Use recordings for review; do not depend on them
- You are expected to take notes: bring pen and paper
- Laptops and mobile devices are not allowed in class unless specifically called for
  - Not even for notes or following along with demos
  - Exceptions need documentation
CodeGrinder and the cs2810 server

You should plan to do most of your homework remotely on a server

- All of our programming assignments will use CodeGrinder
- Most will require a special environment that I set up
- You can set it all up on your own machine, but it is probably not worth it

First steps:

- Make sure you have a Unix shell to work in (Linux or Mac OS)
  - WSL works great for this
- Set up SSH to connect to the cs2810.cs.dixie.edu server
- Learn to use command-line tools to do your work
  - Good idea to learn *vim* or *emacs*, but *micro* is a fallback

I will help you get started, but you should plan to spend time throughout the semester learning to better use your tools.
Unsigned binary numbers

- Bits: why?
- $2^n$ combinations for $n$ bits
- Binary vs decimal vs hexadecimal vs octal
- Approximating how big a binary number is (1,000 vs 1024)
- Binary addition
- Binary fractions
- Bytes, words, half words, double words, quad words
  - Programs and data are all represented as sequences of bytes
Steps

You will need convenient access to the machine cs2810.cs.dixie.edu to work on homework. A few steps:

- Set up an ssh tunnel with public-key authentication
- Log in to CodeGrinder
- Learn some vim basics
- Complete “Hello, world!” exercise in C
ssh tunnel

- What is ssh?
- What is public-key authentication?
- What is a tunnel and why is it needed?
- The actual process (see guide on course page)
- Other benefits: scp and rsync
CodeGrinder demo

- Logging in (once per semester per machine)
- Activating an assignment through Canvas
- `grind list`, `grind get`
- Recommendation: one git repo for the course directory
- Reading documentation for a problem
- “Hello, world!” demo
vim basics

- Why vim?
- Why modes? What does vim optimize for?
- Motion commands
- Entering insert mode
- Learn slowly: enough basics to get by, then gradually add on