The Hardware/Software Interface
CSE 351 Spring 2017

Instructor:
Ruth Anderson

Teaching Assistants:
Dylan Johnson
Kevin Bi
Linxing Preston Jiang
Cody Ohlsen
Yufang Sun
Joshua Curtis
Welcome to CSE351!

- See the key abstractions “under the hood” to describe “what really happens” when a program runs
  - How is it that “everything is 1s and 0s”?
  - Where does all the data get stored and how do you find it?
  - How can more than one program run at once?
  - What happens to a Java or C program before the hardware can execute it?
  - And much, much, much more...

- An *introduction* that will:
  - Profoundly change/augment your view of computers and programs
  - Connect your source code down to the hardware
  - Leave you impressed that computers ever work
Who: Course Staff

- **Instructor:** Ruth Anderson

- **6 TAs:**
  - Dylan Johnson, Kevin Bi, Linxing Preston Jiang, Cody Ohlsen, Yufang Sun, Joshua Curtis
  - Available in sections, in office hours, via email, on Piazza
  - Your course navigators

- **Get to know us!**
  - We are here to help you succeed!
  - And to make the course better – with your help
Me (Ruth Anderson)

- **Grad Student at UW** in Programming Languages, Compilers, Parallel Computing
- **Taught Computer Science** at the University of Virginia for 5 years
- **Grad Student at UW**: PhD in Educational Technology, Pen Computing
- **Current Research**: Computing and the Developing World, Computer Science Education
- **Recently Taught**: data structures, architecture, compilers, programming languages, 142 & 143, data programming in Python, Unix Tools, Designing Technology for Resource-Constrained Environments
Introductory Survey, due Thursday

- Major
- Hometown
- Interesting Fact, Hobbies, or what I did over break.
Acknowledgements

- Many thanks to the people whose course content we are liberally reusing with at most minor changes
  - CMU: Randy Bryant, David O’Halloran, Gregory Kesden, Markus Püschel
  - Harvard: Matt Welsh (now at Google-Seattle)
  - UW: Gaetano Borriello, Luis Ceze, Peter Hornyack, Hal Perkins, Ben Wood, John Zahorjan, Katelin Bailey, Justin Hsia, Dan Grossman, Brandon Holt
  - Not listed: hundreds of TAs
Who are You?

- ~120 students registered
  - See me if you are interested in taking the class but are not yet registered
- CSE majors, EE majors, and more
  - Most of you will find almost everything in the course new
- Submit Start-of-Quarter Survey so we can find out more
- Get to know each other and help each other out!
  - Learning is much more fun with friends
  - Working well with others is a valuable life skill
  - Diversity of perspectives expands your horizons
Quick Announcements (see slides at end of lecture)

- Explore website thoroughly:  [http://cs.uw.edu/351](http://cs.uw.edu/351)
  - Schedule (containing slides, readings, office hours) & midterm date coming soon
- Check that you are enrolled in Piazza
- Start-of-Course survey [Catalyst] due Thursday (3/30)
- Section 1 is this Thursday
  - Install the virtual machine (VM) *before* coming to section
  - Bring your computer with you to section
- Lab 0, due Monday (4/3) @ 11:59pm
- Homework 1, due Monday (4/3) @ 11:59pm
- Readings in CSAPP
Outline for today

- Intros
- What is this course about?
- Course logistics
The Hardware/Software Interface

- What do we mean by hardware? software?
- What is an interface?
- Why do we need a hardware/software interface?
- Why do we need to understand both sides of this interface?
C/Java, assembly, and machine code

```c
if (x != 0) y = (y+z)/x;
```

**High Level Language** (e.g. C, Java)

**Assembly Language**

```assembly
cmp %eax, -12(%ebp), %eax
movl %eax, %edx
leal (%edx, %eax), %eax
movl %eax, %edx
sarl $31, %edx
idivl -12(%ebp)
movl %eax, -12(%ebp)
```

**Machine Code**

```
1000001101111100001001000001110000000000
0111010000011000
100001001100010001100100001100011100
10000100110001100010010000010100
1000100111000010
11000001111111101000011111
1110011101111100001001000011100
10001001010001000010010000011000
```

Compiler

Assembler

Machine Code
C/Java, assembly, and machine code

```c
if (x != 0) y = (y+z)/x;
```

- All program fragments are equivalent
- You’d rather write C! (more human-friendly)
- Hardware executes strings of bits
  - In reality everything is voltages
  - The machine instructions are actually much shorter than the number of bits we would need to represent the characters in the assembly language
HW/SW Interface: Historical Perspective

- Hardware started out quite primitive

Jean Jennings (left), Marlyn Wescoff (center), and Ruth Lichterman program ENIAC at the University of Pennsylvania, circa 1946.

Photo: Corbis
HW/SW Interface: Historical Perspective

- Hardware started out quite primitive
  - Programmed with very basic instructions (*primitives*)
  - e.g., a single instruction for adding two integers

- Software was also very basic
  - Closely reflected the actual hardware it was running on
  - Specify each step manually
HW/SW Interface: Assemblers

- Life was made a lot better by assemblers
  - 1 assembly instruction = 1 machine instruction
  - More human-readable syntax
    - Assembly instructions are character strings, not bit strings
  - Can use symbolic names
HW/SW Interface: Higher-Level Languages

- Higher level of abstraction
  - 1 line of a high-level language is *compiled* into many (sometimes very many) lines of assembly language

![Diagram](image-url)
HW/SW Interface: Compiled Programs

Note: The compiler and assembler are just programs, developed using this same process.
Big Theme: Abstractions and Interfaces

- Computing is about abstractions
  - (but we can’t forget reality)
- What are the abstractions that we use?
- What do you need to know about them?
  - When do they break down and you have to peek under the hood?
  - What bugs can they cause and how do you find them?
- How does the hardware relate to the software?
  - Become a better programmer and begin to understand the important concepts that have evolved in building ever more complex computer systems
Roadmap

C:
car *c = malloc(sizeof(car));
c->miles = 100;
c->gals = 17;
float mpg = get_mpg(c);
free(c);

Java:
Car c = new Car();
c.setMiles(100);
c.setGals(17);
float mpg = c.getMPG();

Assembly language:
get_mpg:
    pushq    %rbp
    movq     %rsp, %rbp
    ...
    popq     %rbp
    ret

Machine code:
0111010000011000
100011010000010000000010
1000100111000010
1100000111111010000011111

OS:

Computer system:

Memory & data
Integers & floats
x86 assembly
Procedures & stacks
Executables
Arrays & structs
Memory & caches
Processes
Virtual memory
Memory allocation
Java vs. C
Little Theme 1: Representation

- All digital systems represent everything as 0s and 1s
  - The 0 and 1 are really two different voltage ranges in the wires
  - Or magnetic positions on a disc, or hole depths on a DVD, or even DNA...

- “Everything” includes:
  - Numbers – integers and floating point
  - Characters – the building blocks of strings
  - Instructions – the directives to the CPU that make up a program
  - Pointers – addresses of data objects stored away in memory

- Encodings are stored throughout a computer system
  - In registers, caches, memories, disks, etc.

- They all need addresses (a way to locate)
  - Find a new place to put a new item
  - Reclaim the place in memory when data no longer needed
Little Theme 2: Translation

- There is a big gap between how we think about programs and data and the 0s and 1s of computers
  - Need languages to describe what we mean
  - These languages need to be translated one level at a time

- We know Java as a programming language
  - Have to work our way down to the 0s and 1s of computers
  - Try not to lose anything in translation!
  - We’ll encounter Java byte-codes, C language, assembly language, and machine code (for the x86 family of CPU architectures)
    - Not in that order, but will all connect by the last lecture!!!
Little Theme 3: Control Flow

- How do computers orchestrate everything they are doing?
- **Within one program:**
  - How do we implement if/else, loops, switches?
  - What do we have to keep track of when we call a procedure, and then another, and then another, and so on?
  - How do we know what to do upon “return”?
- **Across programs and operating systems:**
  - Multiple user programs
  - Operating system has to orchestrate them all
    - Each gets a share of computing cycles
    - They may need to share system resources (memory, I/O, disks)
  - Yielding and taking control of the processor
    - Voluntary or “by force”?
Writing Assembly Code? In 2016???

- Chances are, you’ll never write a program in assembly
  - Compilers are much better and more patient than you are
- But understanding assembly is the key to the machine-level execution model
  - Behavior of programs in presence of bugs
    - High-level language model breaks down
  - Tuning program performance
    - Understand optimizations done/not done by the compiler
    - Understanding sources of program inefficiency
  - Implementing system software
    - Operating systems must manage process state
  - Fighting malicious software
  - Using special units (timers, I/O co-processors, etc.) inside processor!
Course Outcomes

- Understanding of some of the abstractions that exist between programs and the hardware they run on, why they exist, and how they build upon each other
- Knowledge of some of the details of underlying implementations
  - Less important later, but cannot “get it” without “doing it” and “doing it” requires details
- Become more effective programmers
  - Understand some of the many factors that influence program performance
  - More efficient at finding and eliminating bugs
  - Facility with more languages that we use to describe programs and data
  - Better understand new hardware
- Prepare for later classes in CSE
CSE351’s role in the CSE Curriculum

- **Pre-requisites**
  - 142 and 143 – Intro Programming I and II
  - Recommended: 391 – System and Software Tools

- **Complementary to:**
  - CSE311→CSE369→CSE371: hardware design “below us”
  - EE/CSE474 embedded systems: CSE351 invaluable but not a pre-req [EE]
  - CSE331/332/341: high-level software design and structures

- **Essential pre-req for:**
  - CSE401 – Compilers: write a *program* to do CSE351 translations
  - CSE333: building well-structured systems in C/C++
  - Beyond 333: OS, networks, distributed systems, graphics, ...
Course Perspective

❖ CSE351 will make you a better programmer
  ▪ Purpose is to show how software really works
  ▪ Understanding the underlying system makes you more effective
    • Better debugging
    • Better basis for evaluating performance
    • How multiple activities work in concert (e.g., OS and user programs)
  ▪ Not just a course for hardware enthusiasts!
    • What every CSE major needs to know (plus many more details)
    • See many patterns that come up over and over in computing (like caching)
  ▪ “Stuff everybody learns and uses and forgets not knowing”

❖ CSE351 presents a world-view that will empower you
  ▪ The intellectual and software tools to understand the trillions+ of 1s and 0s that are “flying around” when your program runs
Outline for today

- Intros
- What is this course about?
- Course logistics
Communication

- **Website:** [http://cs.uw.edu/351](http://cs.uw.edu/351)
  - Schedule, policies, sections, links, assignments, etc.

- **Discussion:** [https://piazza.com/washington/spring2017/cse351](https://piazza.com/washington/spring2017/cse351)
  - Ask and answer questions – staff will monitor and contribute

- **Course mailing list** – check your @uw.edu
  - Low traffic – announcements; you are already subscribed

- **Staff e-mail:** cse351-staff@cse.uw.edu
  - For things that are not appropriate for the discussion board

- **Office Hours:** spread throughout the week
  - Can also e-mail to make individual appointments

- **Anonymous feedback:**
  - Anything course-related where you would prefer not attaching your name
Course Components

- **Lectures (28)**
  - Introduce the concepts; supplemented by textbook

- **Sections (9-10)**
  - Applied concepts, important tools and skills for labs, clarification of lectures, exam review and preparation

- **Written homework assignments (5)**
  - Mostly problems from textbook to solidify understanding

- **Programming lab assignments (6)**
  - Provide in-depth understanding (via practice) of an aspect of system

- **Exams (2)**
  - **Midterm:** date TBA, in lecture
  - **Final:** Wednesday, June 7, 2:30-4:20pm
Policies

- **Exams:** Midterm (15%) and Final (30%)
  - Many old exams on course website
- **Homework:** weighted according to effort (20% total)
  - We’ll try to make these about the same
- **Labs:** weighted according to effort (35% total)
  - These will likely increase in weight as the quarter progresses
- **Other important policies:** (details on [website](#))
  - 3 allowed **late days** for the quarter
  - **Collaboration** and academic integrity
  - Assignment and exam **re-grades**
Textbooks

- **Computer Systems: A Programmer’s Perspective**
  - Randal E. Bryant and David R. O’Hallaron
  - Website: [http://csapp.cs.cmu.edu](http://csapp.cs.cmu.edu)
  - Must be **3rd edition**
    - [http://csapp.cs.cmu.edu/3e/changes3e.html](http://csapp.cs.cmu.edu/3e/changes3e.html)
    - [http://csapp.cs.cmu.edu/3e/errata.html](http://csapp.cs.cmu.edu/3e/errata.html)
  - This book really matters for the course!
    - How to solve labs
    - Practice problems typical of exam problems

- **A good C book – any will do**
  - *The C Programming Language* (Kernighan and Ritchie)
  - *C: A Reference Manual* (Harbison and Steele)
Videos / Online course

- Gaetano Borriello and Luis Ceze made videos in 2013 covering the course content for an online version
  - And self-check quiz questions
- A great resource – I encourage you to watch them
  - Generally optional unless class is cancelled or something
  - Occasionally may “require before class” so you don’t get lost in an example
- Warning: some content has since changed
  - Now “all 64-bit” so some videos may have extra information no longer relevant
  - When in doubt, go with current lectures (but do ask first)
Other details

- Consider taking CSE 391 Unix Tools, 1 credit
  - Useful skills to know and relevant to this class
  - Available to all CSE majors and everyone registered in CSE351

- Everything starts now!
  - Including section and office hours this week
To-Do List

- Explore website thoroughly: [http://cs.uw.edu/351](http://cs.uw.edu/351)
  - Schedule (containing slides, readings, office hours) & midterm date coming soon
- Check that you are enrolled in Piazza
- Start-of-Course survey [Catalyst] due Thursday (3/30)
- Section 1 is this Thursday
  - **Install the virtual machine (VM) *before* coming to section**
  - **Bring your computer with you to section**
- Lab 0, due Monday (4/3) @ 11:59pm
- Homework 1, due Monday (4/3) @ 11:59pm
- Readings in CSAPP
Lab 0, Homework 1, and Readings

- **Lab 0, due Monday (4/3) @ 11:59pm**
  - Basic exercises to *start* getting familiar with C – need the VM
  - Credit/no-credit
  - Do ASAP, attending Section 1 will help

- **Homework 1, due Monday (4/3) @ 11:59pm**
  - 3 canvas quizzes, 20 tries each, you best overall score is kept
    - Course policies – you can do this one now!
    - Unsigned Number Representations – will discuss on Wed
    - Number Bases - will discuss on Wed

- **Readings:**
  - For TODAY: CSPP: § 1.0-1.10 (pp. 1-28)
  - For Wed: CSPP: § 2.0-2.1.3 (pp. 31-48)
  - For Fri: CSPP: § 2.1.3-2.1.5 (pp. 42-50)