The Hardware/Software Interface

CSE 351 Winter 2016

Instructor:

Dan Grossman

Teaching Assistants:

Rajas Agashe, Kevin Bi, Dylan Johnson, Sarang Joshi, Anthony McIntosh, Alfian Rizqi, Yufang Sun

Welcome!

10 weeks to 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 processor can execute it?
- Why is recursion not even slightly magical?
- 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

Concise To-Do List

- **Review syllabus, course goals, collaboration policy, etc.: http://courses.cs.washington.edu/courses/cse351/16wi/**
- **Email-list settings, if necessary**
- **Beginning-of-course survey, "due" Wednesday 5PM**
- **Lab 0, due Monday, January 11 at 5pm**
  - Make sure you get our virtual machine set up and are able to do work
  - Basic exercises to *start* getting familiar with C
  - Credit/no-credit
  - Get this done as quickly as possible
- **Section Thursday**
  - **Please install the virtual machine BEFORE coming to section**
  - BRING your computer with you to section
  - Includes activities to help you get started with Lab 0

Who: Course Staff

- **Dan Grossman: Faculty since 2003, veteran of 341, 332, 331, 373, but first time in 351**
  - Know and love the content, new to the course
  - Not planning "changes" but will be "fresh eyes"
- **TAs: 7 (!), all have taken the course, 3 TA veterans (2 multi-)**
- **Office hours will be figured out ASAP**
- **Get to know us!**
  - We are here to help you succeed
  - And to make the course better
  - And to enjoy showing you a new world

Acknowledgments

Many thanks to the many 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, Ruth Anderson
- Not listed: dozens of TAs

Who are you?

- **~90 registered**
  - My intention: Make it feel like 40; learn all your names
- **CSE majors, EE majors, some want-to-be majors**
  - Most of you will find almost everything in the course “brand new”
- **Please get to know each other**

Staying In Touch

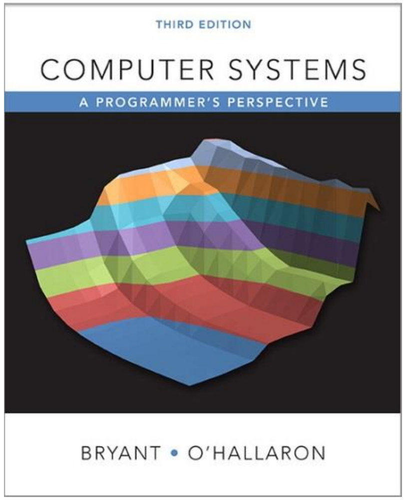
- **Course web page**
  - Schedule, policies, labs, homeworks, and everything else
- **Course discussion board**
  - Keep in touch outside of class – help each other
  - Staff will monitor and contribute
- **Course mailing list cse351a_wi16@u.washington.edu**
  - Low traffic – mostly announcements; your @uw.edu is subscribed
- **Office hours, appointments, drop-ins**
  - We will spread our office hours throughout the week
- **Staff e-mail (Dan + TAs): cse351-staff@cse.uw.edu**
  - For things that are not appropriate for the discussion board
- **Anonymous feedback**
  - Comments about anything related to the course where you would feel better not attaching your name: goes directly to Dan

Course Components

- **Lectures (27)**
  - Introduce the concepts; supplemented by textbook
- **Sections (10)**
  - Applied concepts, important tools and skills for labs, clarification of lectures, exam review and preparation
- **Written homework assignments (4)**
  - Mostly problems from textbook to solidify understanding
- **Programming labs/assignments (5, plus "lab 0")**
  - Provide in-depth understanding (via practice) of an aspect of system
- **Exams (midterm + final)**
  - Test your understanding of concepts and principles
  - Midterm Monday February 8, in class
  - Final time set by the university: Wednesday March 16, 2:30-4:20PM ☹

Textbooks

- **Computer Systems: A Programmer's Perspective, 3rd Edition**
  - Randal E. Bryant and David R. O'Hallaron
  - Prentice-Hall, 2015
  - http://csapp.cs.cmu.edu
  - 3rd edition includes complete rewrite of Chapter 3
    - All code examples in x86-64
    - http://csapp.cs.cmu.edu/3e/changes3e.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**
  - C: A Reference Manual (Harbison and Steele) [instructor preference]
  - The C Programming Language (Kernighan and Ritchie)

Videos / Online course

- **Gaetano Borriello and Luis Ceze made videos in 2013 covering the course content [for an online version of the course]**
  - And self-check quiz questions
- **These are a great resource – 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
- **But the course is now “all 64-bit” so some parts of the course no longer have [relevant] videos available**
  - New videos not yet made – may get some progress on that

Policies: Grading

- **Exams (45%): 15% midterm, 30% final**
  - Many old exams on course website (but now 64-bit and new instructor)
- **Written assignments (20%): weighted according to effort**
  - We'll try to make these about the same
- **Lab assignments (35%): weighted according to effort**
  - These will likely increase in weight as the quarter progresses
- **Late days:**
  - 3 late days to use as you wish throughout the quarter – see website
- **Collaboration:**
  - http://www.cse.uw.edu/education/courses/cse351/16wi/policies.html
  - http://www.cse.uw.edu/students/policies/misconduct
  - Do not cheat!!! It's an affront to the course staff, your fellow students, and yourself. CSE courses are special and valuable – keep it that way!

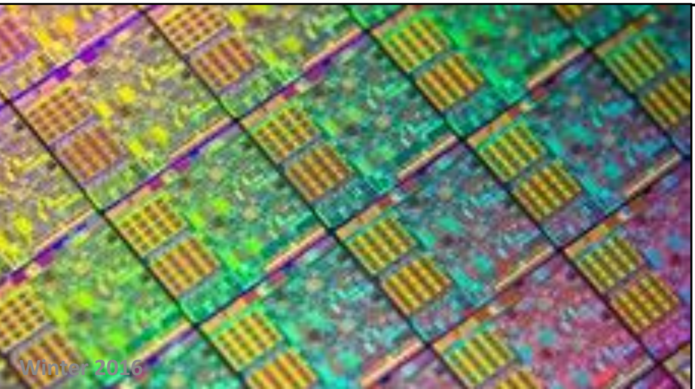
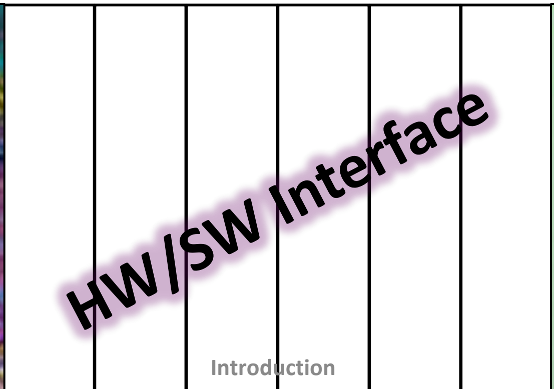
Other details

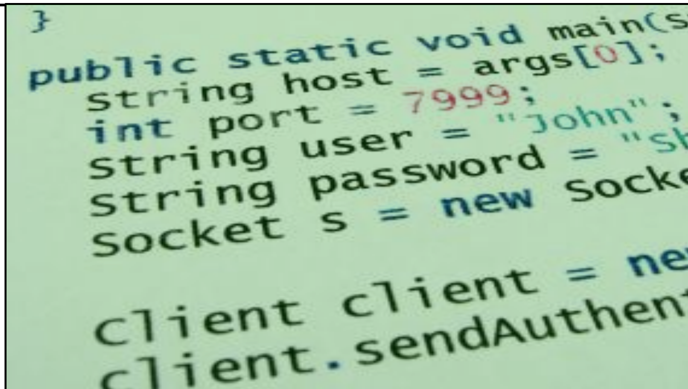
- **Consider taking CSE 390A Unix Tools, 1 credit, useful skills**
  - Available to all CSE majors and everyone registered in CSE351
- **Office hours will be held this week, check web page for times**
- **Remember Lab 0 asap and bring laptop to section**

Anything I forgot about course mechanics before we discuss, you know, hardware and software?

The Hardware/Software Interface

- **What is 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

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

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

```
        cmpl    $0, -4(%ebp)
        je      .L2
        movl    -12(%ebp), %eax
        movl    -8(%ebp), %edx
        leal    (%edx, %eax), %eax
        movl    %eax, %edx
        sarl    $31, %edx
        idivl   -4(%ebp)
        movl    %eax, -8(%ebp)
.L2:
```

Assembly Language

```
1000001101111100001001000001110000000000
0111010000011000
10001011010001000010010000010100
10001011010001100010010100010100
100011010000010000000010
1000100111000010
110000011111101000011111
11110111011111000010010000011100
10001001010001000010010000011000
```

Machine Code

C/Java, assembly, and machine code

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

Compiler

```
        cmpl    $0, -4(%ebp)
        je      .L2
        movl    -12(%ebp), %eax
        movl    -8(%ebp), %edx
        leal    (%edx, %eax), %eax
        movl    %eax, %edx
        sarl    $31, %edx
        idivl   -4(%ebp)
        movl    %eax, -8(%ebp)
.L2:
```

Assembler

```
1000001101111100001001000001110000000000
0111010000011000
10001011010001000010010000010100
10001011010001100010010100010100
100011010000010000000010
1000100111000010
110000011111101000011111
11110111011111000010010000011100
10001001010001000010010000011000
```

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

Assembly Language

Machine Code

C/Java, assembly, and machine code

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

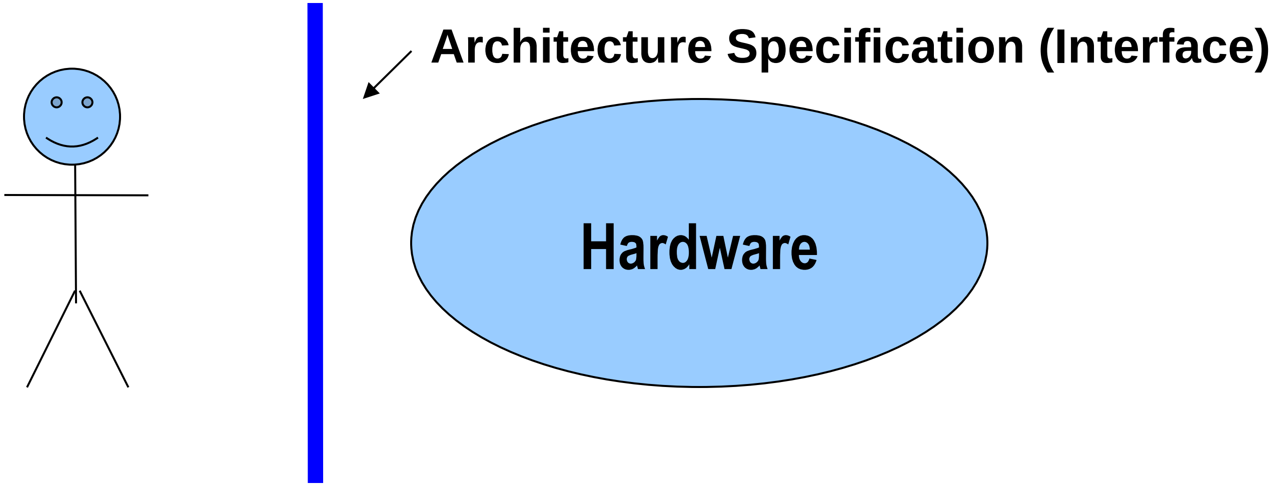
```
        cmpl    $0, -4(%ebp)
        je      .L2
        movl    -12(%ebp), %eax
        movl    -8(%ebp), %edx
        leal    (%edx, %eax), %eax
        movl    %eax, %edx
        sarl    $31, %edx
        idivl   -4(%ebp)
        movl    %eax, -8(%ebp)
.L2:
```

```
1000001101111100001001000000111000000000
0111010000011000
10001011010001000010010000010100
10001011010001100010010100010100
100011010000010000000010
1000100111000010
110000011111101000011111
11110111011111000010010000011100
10001001010001000010010000011000
```

- **The three program fragments are equivalent**
- **You'd rather write C! - a more human-friendly language**
- **The hardware likes bit strings! - 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

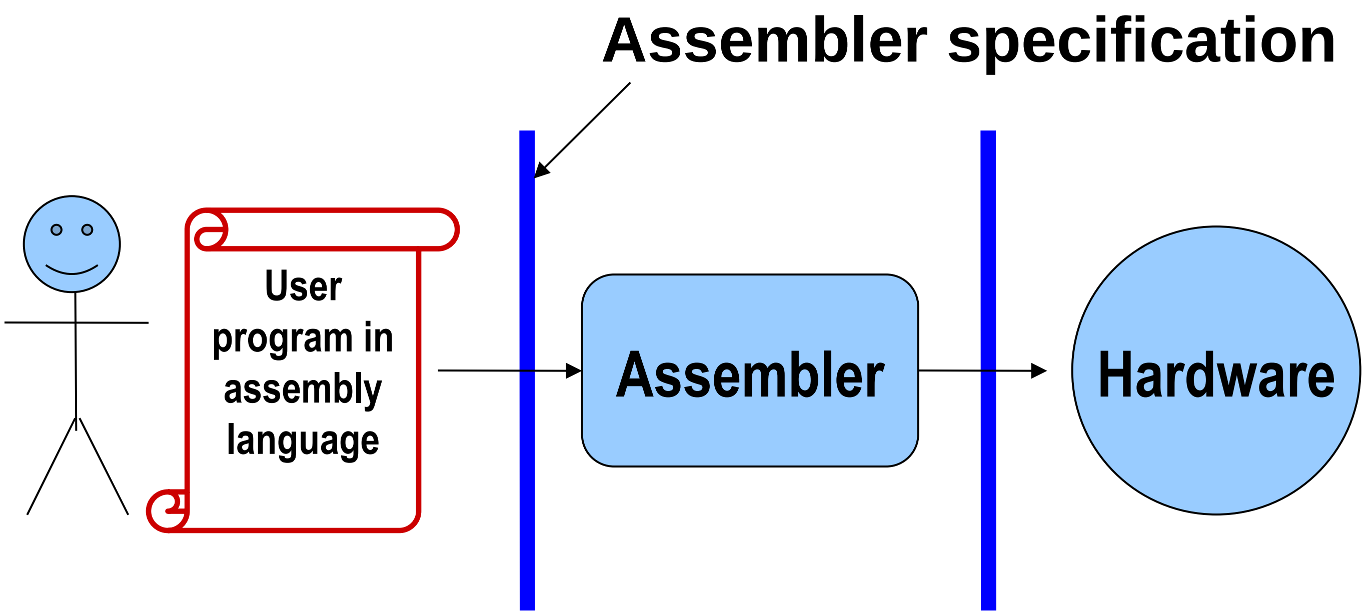
HW/SW Interface: The Historical Perspective

- **Hardware started out quite primitive**
  - Hardware designs were expensive -> instructions had to be very simple – e.g., a single instruction for adding two integers
- **Software was also very basic**
  - Software primitives reflected the hardware pretty closely



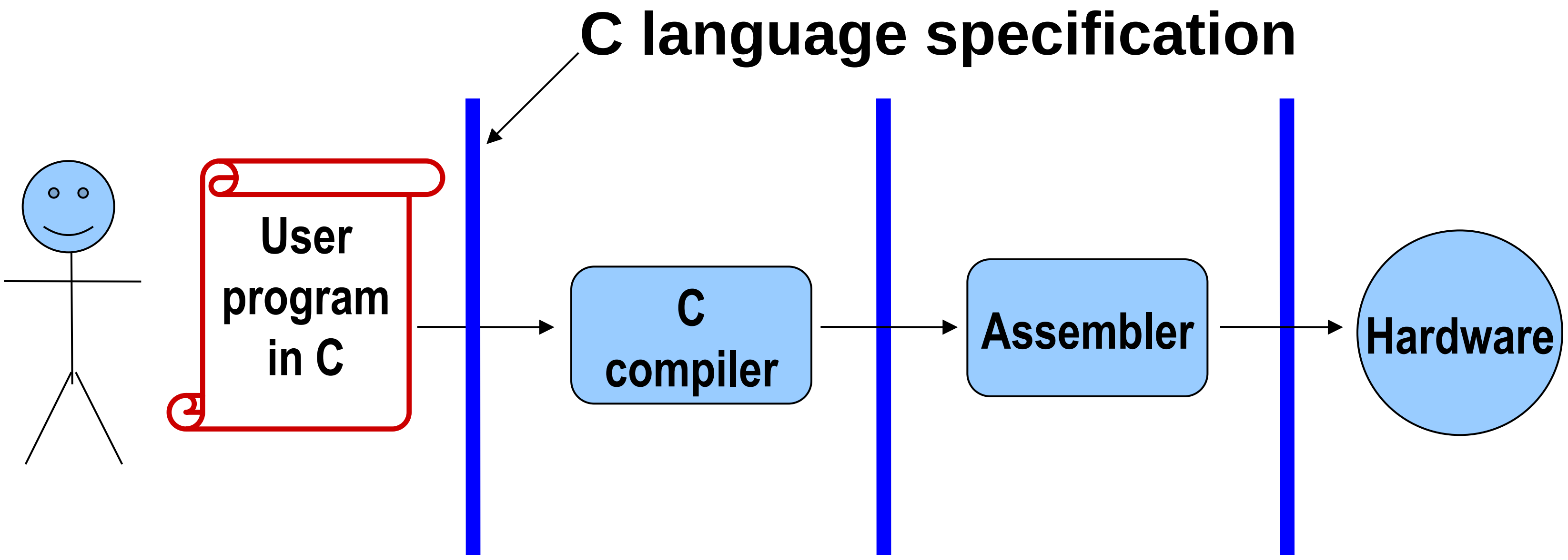
HW/SW Interface: Assemblers

- **Life was made a lot better by assemblers**
  - 1 assembly instruction = 1 machine instruction, but...
  - different syntax: assembly instructions are character strings, not bit strings, a lot easier to read/write by humans
  - can use symbolic names

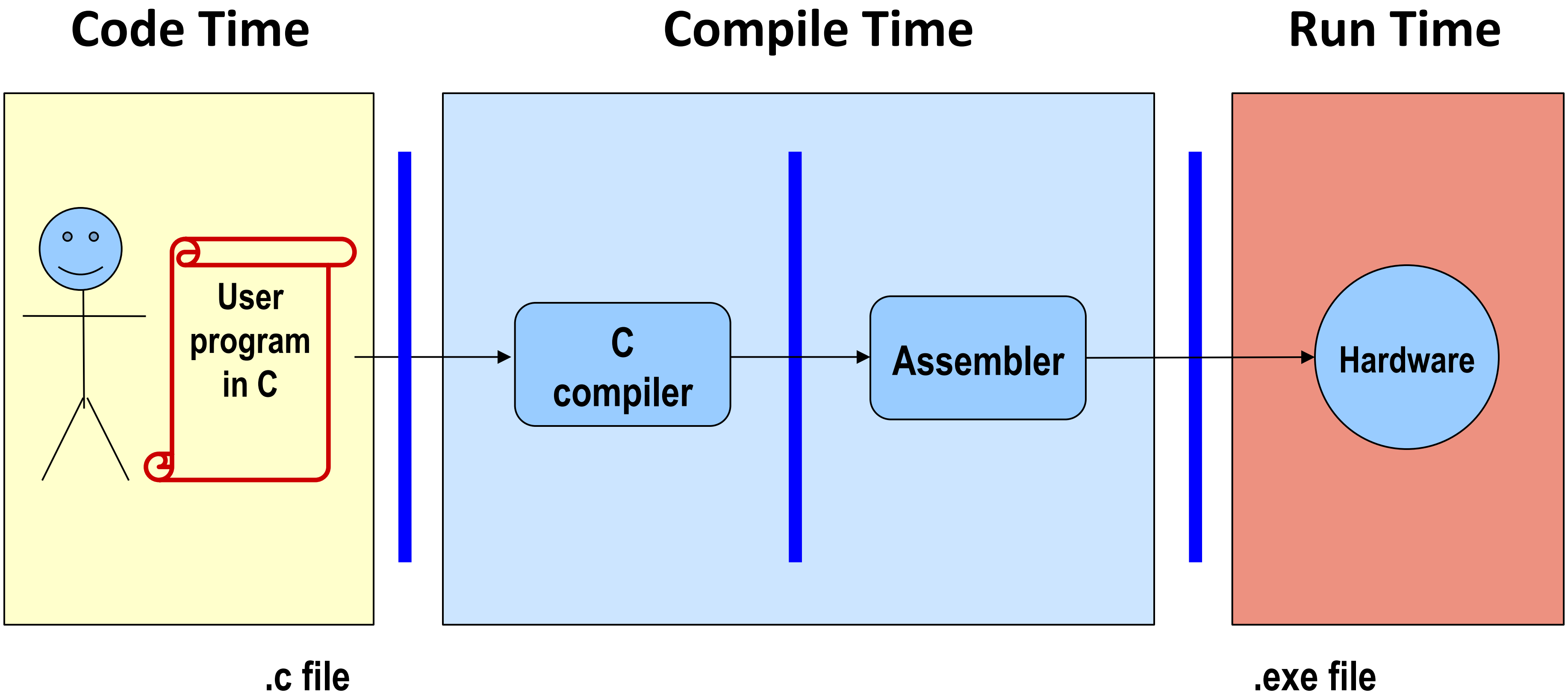


HW/SW Interface: Higher-Level Languages

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



HW/SW Interface: Code / Compile / Run Times



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

The 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 (0s and 1s, processor executing instructions) relate to the software (C/Java programs)?**
  - Become a better programmer and begin to understand the important concepts that have evolved in building ever more complex computer systems

HTTP://XKCD.COM/676/

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
110000011111101000011111
```

OS:

Windows 8 · Mac

Computer system:


- Memory & data
- Integers & floats
- Machine code & C
- x86 assembly
- Procedures & stacks
- 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…
- **“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
- **These encodings are stored throughout a computer system**
  - In registers, caches, memories, disks, etc.
- **They all need addresses**
  - A way to find them
  - 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**
- **Languages need to be translated one step 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 the many things they are doing?**
- **In 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 code**
  - 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 a couple more of the many languages that we use to describe programs and data
- **Prepare for later classes in CSE**

CSE351's role in the CSE Curriculum

- **Pre-requisites**
  - 142 and 143: Intro Programming I and II
  - Also recommended: 390A: System and Software Tools
- **Complementary to:**
  - CSE311->CSE369->EE371 / EE271->EE371: hardware design "below us"
    - "arranging wires to do addition and stuff"
  - 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++
  - Courses after CSE333: 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)
    - Job interviewers love to ask questions from 351!
  - Like other 300-level courses, "stuff everybody learns and uses and forgets not knowing"

- **CSE351 presents a world-view that will empower you**
  - The intellectual tools and software tools to understand the trillions+ of 1s and 0s that are "flying around" when your program runs