The Hardware/Software Interface

CSE 351 Spring 2016

Instructor:

Brandon Holt

Teaching Assistants:

Kevin Bi,
Anton Goncharenko,
Sarang Joshi,
Xi Liu,
Anthony McIntosh,
Alfian Rizqi,
Yufang Sun,
Shan Yang

http://xkcd.com/676/

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 can execute it?
- What is *The* Stack and *The* Heap?
- 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

- **Brandon Holt: 5th year PhD student**
  - Excited to teach what was my favorite class in undergrad.
  - I like: compilers, mountains, and sandwiches (esp. w/ avocado).
  - First time teaching an entire class, so help me out!
- **TAs: 8 (!)**
  - Alfian, Anton, Anthony, Kevin, Sarang, Shan, Xi, Yufang
  - all have taken the course and most have TA'd 351 in the past.
- **Office hours will be figured out ASAP**
- **Get to know us!**
  - We are here to help you succeed
  - And to make the course better – with your help!

Acknowledgments

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, Ruth Anderson, Dan Grossman
- Not listed: dozens of TAs

Who are you?

- **~116(!) registered**
- **CSE majors, EE majors, and more!**
  - Most of you will find almost everything in the course "brand new"
- **Please get to know each other, and help each other out**
- See me if you are interested in taking the class but are not yet registered.

Staying In Touch

- **Course web page: http://cs.uw.edu/351**
  - Schedule, policies, labs, homeworks, and everything else
- **Catalyst discussion board**
  - Keep in touch outside of class – help each other
  - Staff will monitor and contribute
- **Course mailing list `cse351a_sp16@u.washington.edu`**
  - Low traffic – mostly announcements; your @uw.edu is subscribed
- **Office hours, appointments, drop-ins**
  - Spread throughout the week
- **Staff e-mail (Brandon + TAs): `cse351-staff@cs.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, linked from website, goes directly to Brandon.

Course Components

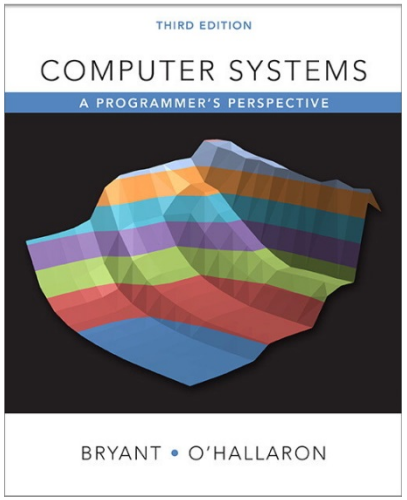
- **Lectures (28)**
  - Introduce the concepts; supplemented by textbook
- **Sections (10)**
  - Applied concepts, important tools and skills for labs, clarification of lectures, exam review and preparation
- **Written homework assignments (4)**
  - Mostly problems from textbook to solidify understanding
- **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 Friday, April 29, in class
  - Final time set by the university: Wednesday June 8, 2:30-4:20PM

Policies: Grading

- **Exams (45%): 15% midterm, 30% final**
  - Many old exams on course website (but now 64-bit, +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/cse351/16sp/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!

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**
  - 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 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 some content has changed**
  - Now "all 64-bit" so some videos may have extra information no longer relevant.
  - This means we may get to new things that class didn't get to.
  - If in doubt, go with what we talked about in this class.

Other details

- **Consider taking CSE 391 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**

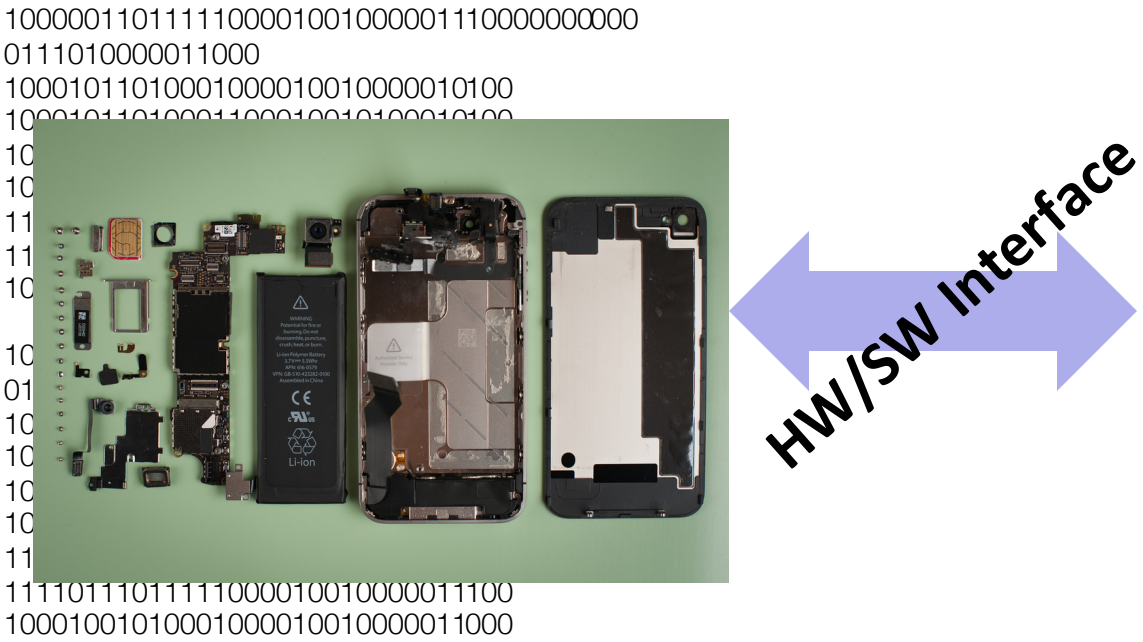
Concise To-Do List

- **Review syllabus, course goals, collaboration policy, etc.: http://cs.uw.edu/351**
- **Make sure you are receiving announcement emails**
- **Beginning-of-course survey, "due" Friday**
- **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

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

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?




```
29 import android.widget.ImageView;
30 import android.widget.LinearLayout;
31 import android.widget.TextView;
32
33 /**
34  * Contains two sub-views to provide a simple stereo HUD.
35  */
36 public class CardboardOverlayView extends LinearLayout {
37   private final CardboardOverlayEyeView leftView;
38   private final CardboardOverlayEyeView rightView;
39   private AlphaAnimation textFadeAnimation;
40
41   public CardboardOverlayView(Context context, AttributeSet attrs) {
42     super(context, attrs);
43     setOrientation(HORIZONTAL);
44
45     Layout
46       Layo
47     params
48
49     leftVi
50     leftVi
51     addVie
52
53     rightV
54     rightV
55     addView
56
57     // Set some reasonable defaults.
58     setDepthOffset(0.01f);
59     setColor(Color.rgb(150, 255, 180));
60     setVisibility(View.VISIBLE);
61
62     textFadeAnimation = new AlphaAnimation(1.0f, 0.0f);
63     textFadeAnimation.setDuration(5000);
```

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:
```

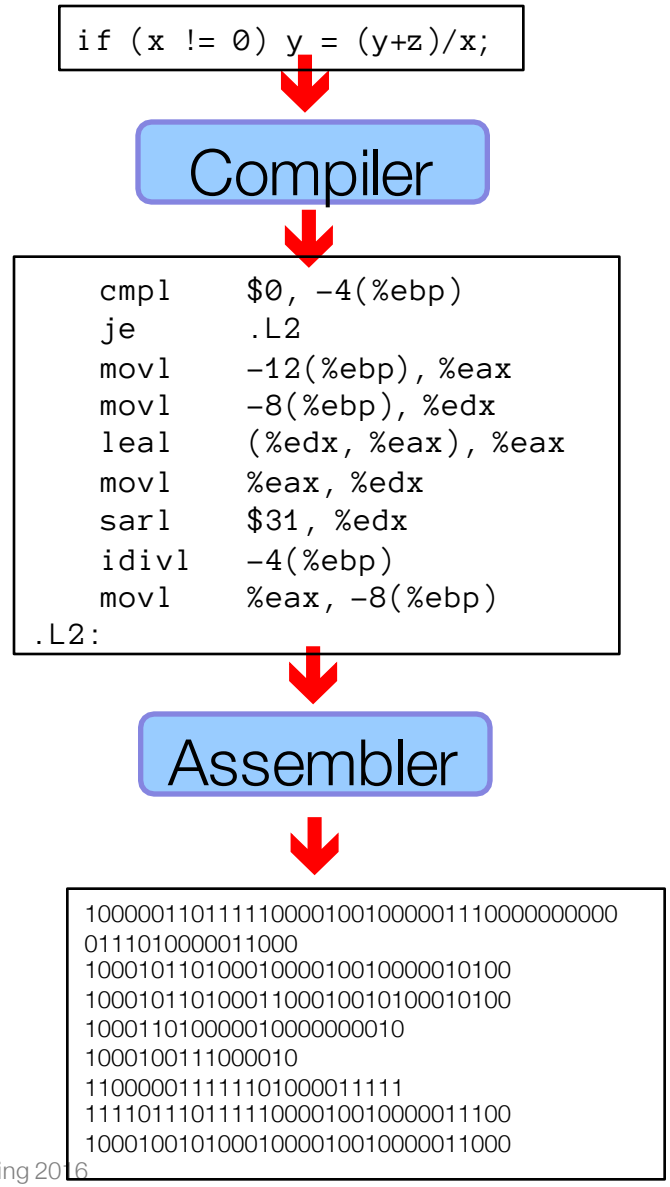
```
10000011011111000010010000011100000000000
0111010000011000
10001011010001000010010000010100
10001011010001100010010100010100
10001101000001000000010
1000100111000010
11000001111110100001111
11110111011111000010010000011100
10001001010001000010010000011000
```

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

Assembly Language

Machine Code

C/Java, assembly, and machine code



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

Assembly Language

Machine Code

C/Java, assembly, and machine code

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

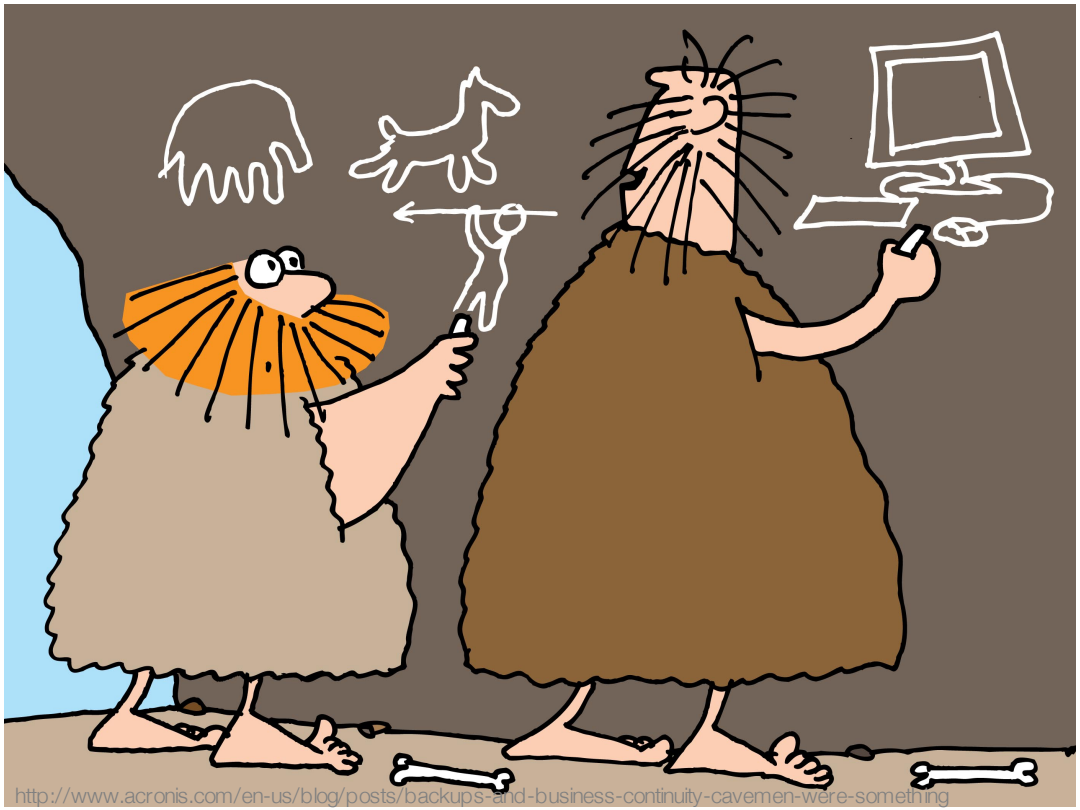
```
    cmpl   $0, -4(%ebp)
    je     .L2
    movl   -12(%ebp), %eax
    movl   -8(%ebp), %edx
    leal   (%edx, %eax), %eax
    movl   %eax, %edx
    sarl   $31, %edx
    idivl  -4(%ebp)
    movl   %eax, -8(%ebp)
.L2:
```

```
10000011011111000010010000011100000000000
0111010000011000
10001011010001000010010000010100
10001011010001100010010100010100
10001101000001000000010
1000100111000010
11000001111111010000011111
11110111011111000010010000011100
10001001010001000010010000011000
```

- The three program fragments are equivalent
- You'd rather write C! (more human-friendly)
- 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

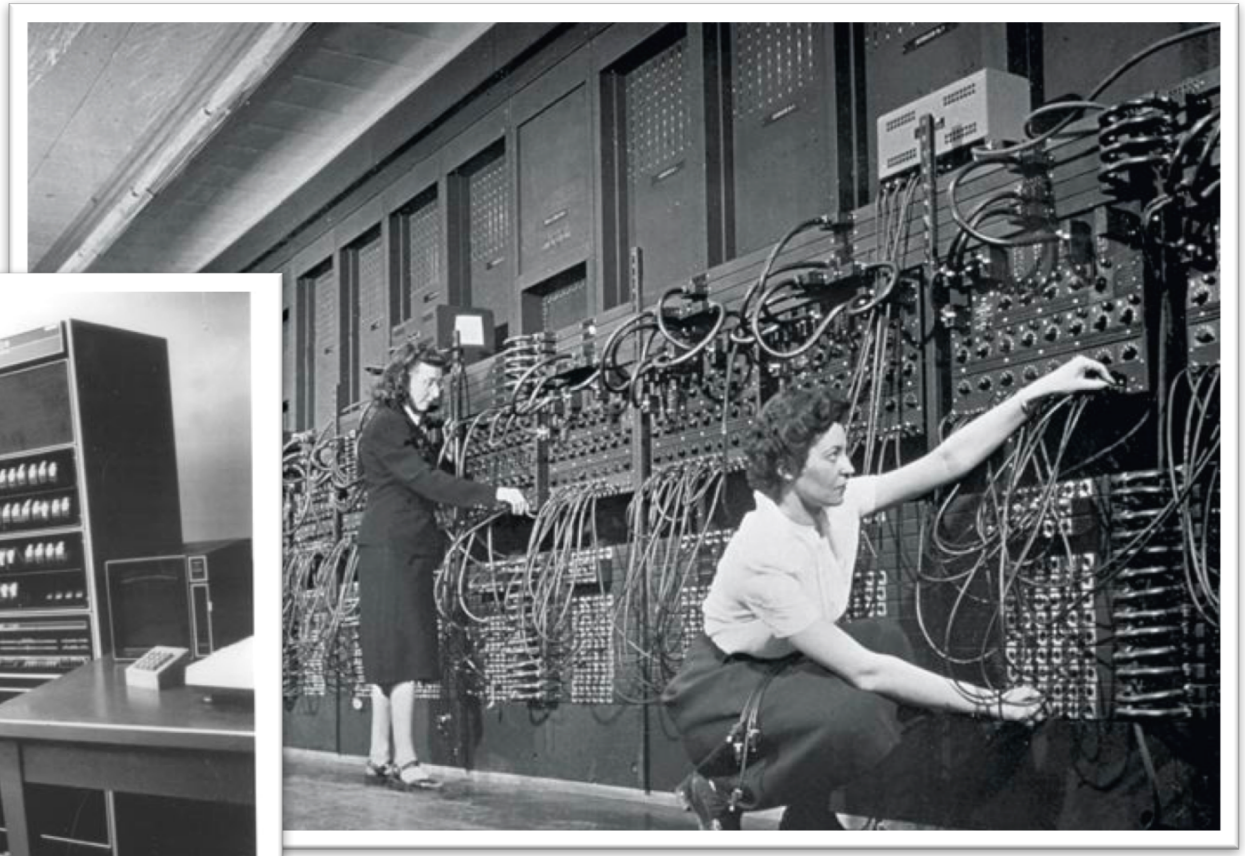
HW/SW Interface: Historical Perspective

- **Hardware started out quite primitive**



HW/SW Interface: Historical Perspective

- **Hardware started out quite primitive**

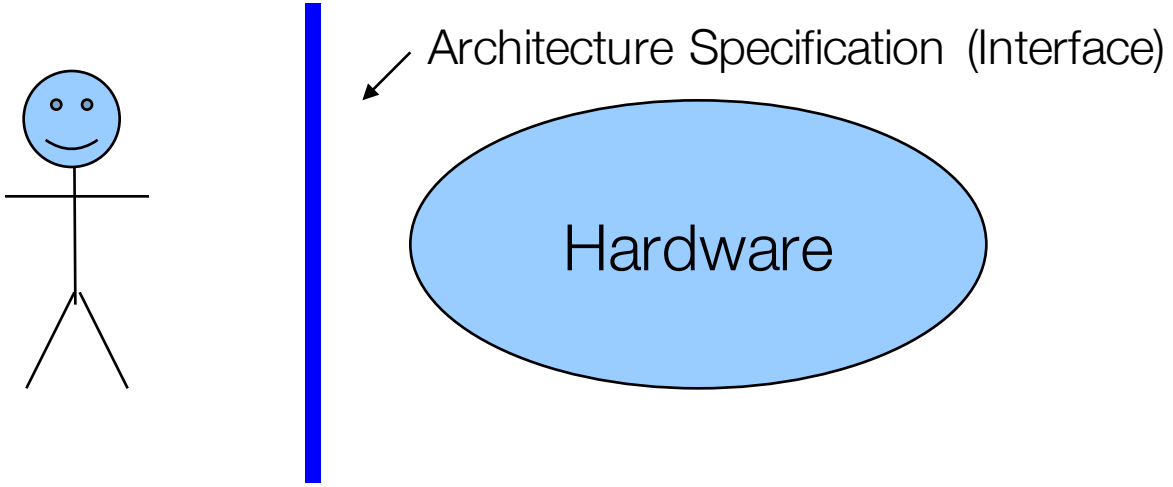


Jean Jennings (left), Marlyn Wescoff (center), and Ruth Lichterman program ENIAC at the University of Pennsylvania, circa 1946. Photo: Corbis
http://fortune.com/2014/09/18/walter-isaacson-the-women-of-eniac/

http://hightechhistory.com/2011/10/24/the-late-dennis-macalistair-ritchie-innovator-of-the-%E2%80%9Cc%E2%80%9D-programming-language-and-

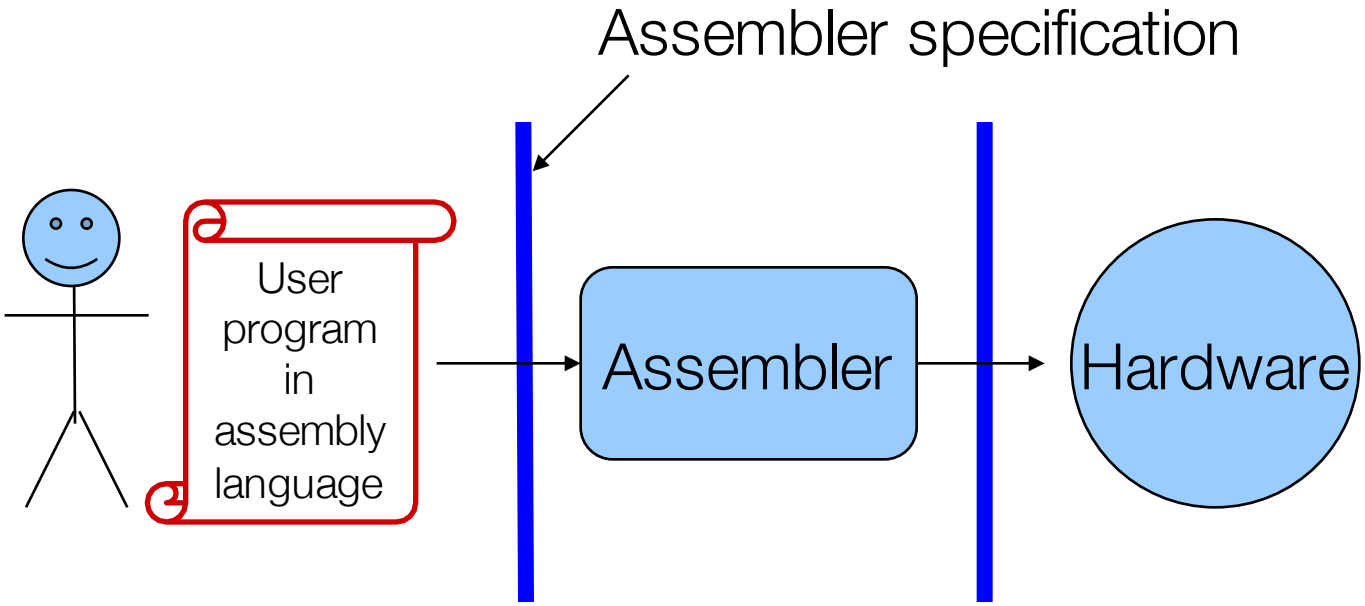
HW/SW Interface: Historical Perspective

- **Hardware started out quite primitive**
  - Programmed with very basic instructions.
  - 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

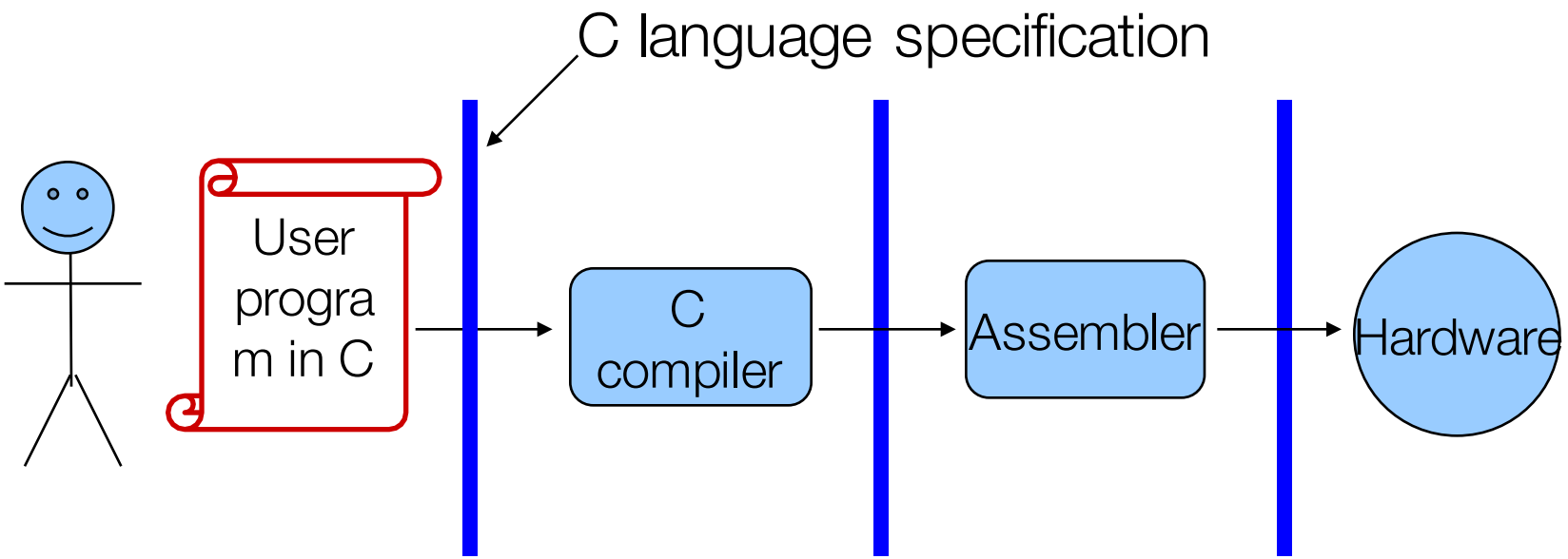
- **Life was made a lot better by assemblers**
  - 1 assembly instruction = 1 machine instruction, but…
  - different syntax: assembly instructions are character strings, not bit strings, a lot easier to read/write by humans
  - can use symbolic names



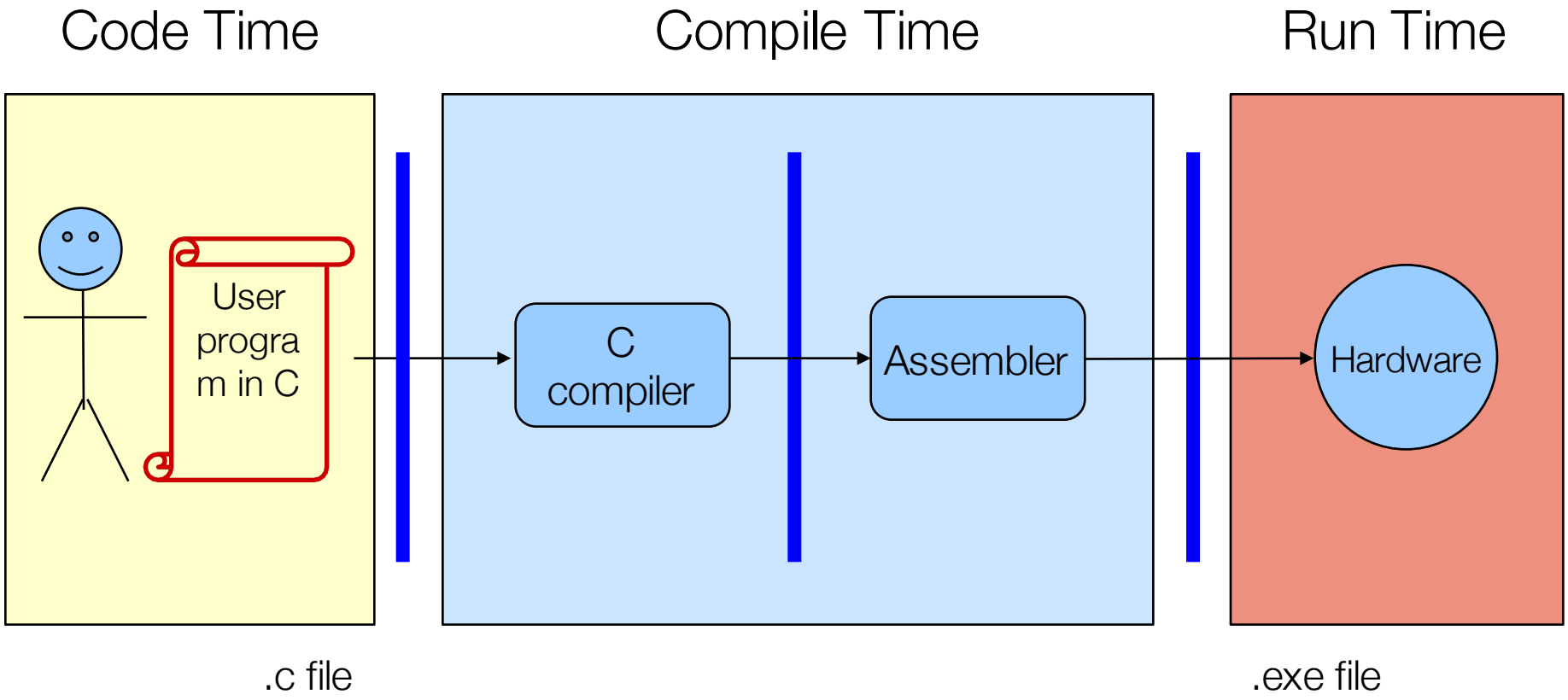
HW/SW Interface:

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

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
11000001111110100011111
```

OS:

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 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
- **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
- 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
  - Also recommended: 391: 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)
    - See many **patterns** that come up over and over in computing (like caching)
  - 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

HTTP://XKCD.COM/676/