The Hardware/Software Interface

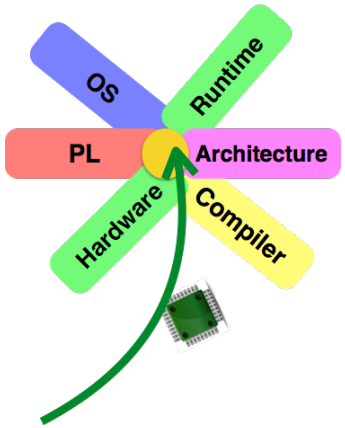
CSE 351 Winter 2017

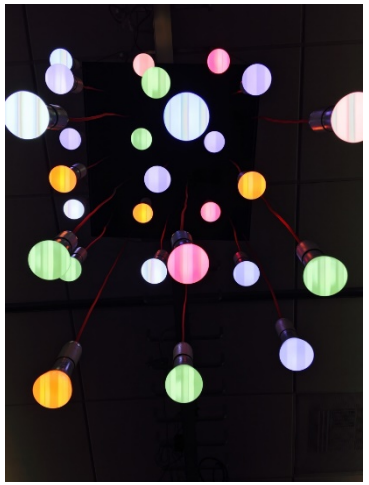
http://xkcd.com/676/

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 is Luis?





Approximate computing
New technologies and applications
DNA storage & computing

And the awesome 351 Wi'17 Staff

Max Willsey
I'm a first-year PhD student focusing on programming languages and architecture. I like reading, running, and southern food (I'm from GA!).

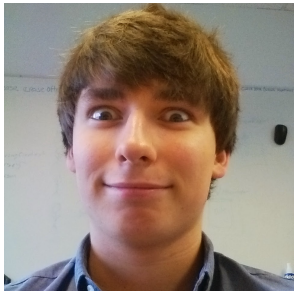
Nick Mooney
I'm a third-year undergrad in the CS department, I'm particularly fond of all things security and low-level, I've been a summer camp counselor for the past three summers, and I think hydration is the fundamental key to success in life

Yufang Sun
I am a senior CS student. Love discussing about security! I like listening to rock and running. (Chop Suey is my absolute favorite!)

And the awesome 351 Wi'17 Staff

Sarang Joshi. I'm a third-year student in CSE, and 351 is my favorite class so far. I enjoy singing, playing and being bad at sports, and staying up late nights in the labs. I'm super excited to be TA'ing, so please come to my office hours to hang out and complain about assembly.

Amrita Mazumdar. I'm a third year PhD student, working at the intersection of computer architecture and computational photography. I like reading books and drinking coffee, and was once described as "actually a really interesting person".



Artem Minyaylov. I'm a Fifth Year Master's Computer Science student working with Luis in the architecture lab. Outside of school, I get great satisfaction out of electronic music, cooking, and painting my own 3D-printed doohickeys.

Who are You?

- ~ 123 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
  - But… anyone written code in C? Multithreaded code?
- Submit Start-of-Quarter Survey
  - https://catalyst.uw.edu/webq/survey/luisceze/321769
- Get to know each other and help each other out!
  - More productive, more fun!

<administrivia>

Communication

- Website: http://cs.uw.edu/351
  - Schedule, policies, sections, links, assignments, etc.
- Discussion: Catalyst Go Post
  - Announcements made here
  - Ask and answer questions – staff will monitor and contribute
- Office Hours: spread throughout the week
  - Can also e-mail to make individual appointments
- Anonymous feedback:
  - Comments about anything related to the course where you would feel better not attaching your name
- Staff mailing list: cse351-staff@cs.washington.edu
- Class-wide mailing list: If you haven't received a welcome message, please contact us.

Course Components

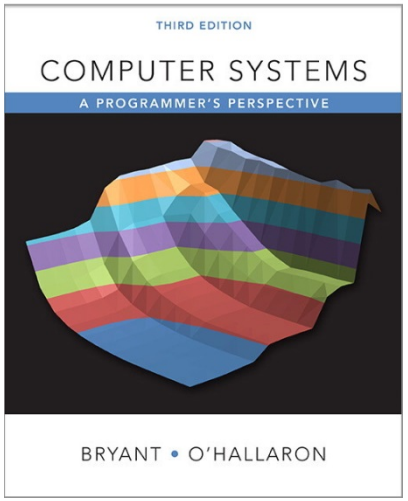
- Lectures (26)
  - Introduce the concepts; supplemented by textbook
- Sections (9-10)
  - Applied concepts, important tools and skills for labs, clarification of lectures, exam review and preparation
- Written homework assignments (4)
  - Mostly problems from textbook to solidify understanding
- Programming lab assignments (6) – **START EARLY!!**
  - Provide in-depth understanding (via practice) of an aspect of system
- Exams (2)
  - **Midterm: Wednesday, Feb 8, in class.**
  - **Final: Wednesday, March 15, 2017,830-1020, JHN 102 (here ☺).**

Policies

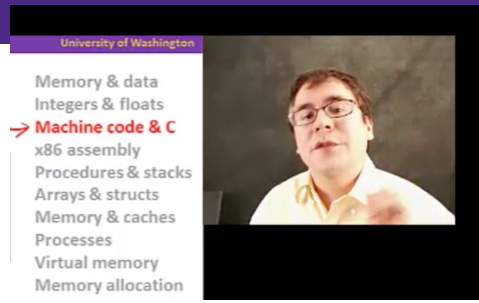
- **Exams:** Midterm (15%) and Final (30%)
  - Many old exams on course website
- **Homework:** weighted according to effort (20% total)
  - We'll try to make these about the same
- **Labs:** weighted according to effort (35% total)
  - These will likely increase in weight as the quarter progresses
- 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
  - Must be 3rd edition
    - http://csapp.cs.cmu.edu/3e/changes3e.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 I made videos in 2013 covering the course content for an online version
  - And self-check quiz questions
- Watch them! [they were a lot of work to make ☺]
  - 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
- Check that you can access Go Post/get notifications
- Start-of-Course survey [Catalyst] due Saturday.
- Section 1 is tomorrow
  - **Install the virtual machine (VM) *before* coming to section**
  - Bring your computer with you to section
- Lab 0 released today, due Monday (1/9) @ 5pm
  - Basic exercises to *start* getting familiar with C – need the VM
  - Credit/no-credit
  - Do ASAP, attending Section 1 will help

- http://tinyurl.com/hz9sxzd

</administrivia>

The Hardware/Software Interface

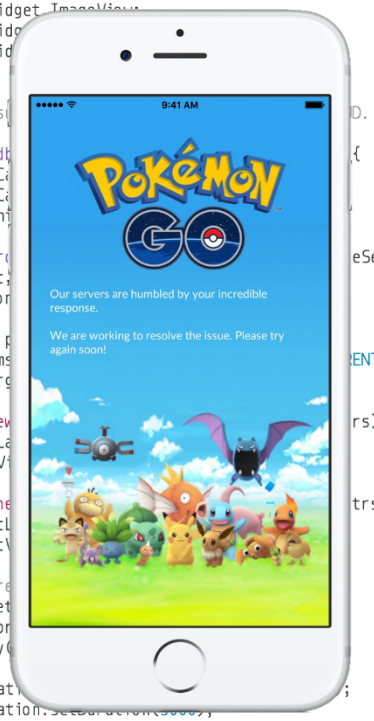
- What do we mean by hardware? software?
- What is an interface?
- Why do we need a hardware/software interface?
- Why do we need to understand both sides of this interface?

1000001101111100001001000001110000000000
0111010000011000

1100000111111010000111111
1111011101111100001001000011100

HW/SW Interface

```
29 import android.widget.ImageView;
30 import android.widg
31 import android.wid
32
33 /**
34  * Contains two s
35  */
36 public class Cardb
37   private final Ca
38   private final Ca
39   private AlphaAni
40
41   public Cardboar
42     super(context)
43     setOrientation
44
45     LayoutParams p
46       LayoutParams
47     params.setMarg
48
49     leftView = new
50     leftView.setLa
51     addView(leftVi
52
53     rightView = ne
54     rightView.setL
55     addView(rightV
56
57     // Set some re
58     setDepthOffset
59     setColor(Color
60     setVisibility(
61
62     textFadeAnimat
63     textFadeAnimation.setDuration(500);
```



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

```
10000011011111000010010000011100000000000111010000011000
10001011010001000010010000010100
10001011010001100010010100010100
10001101000001000000010
1000100111000010
11000001111110100011111
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;
```

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:
```

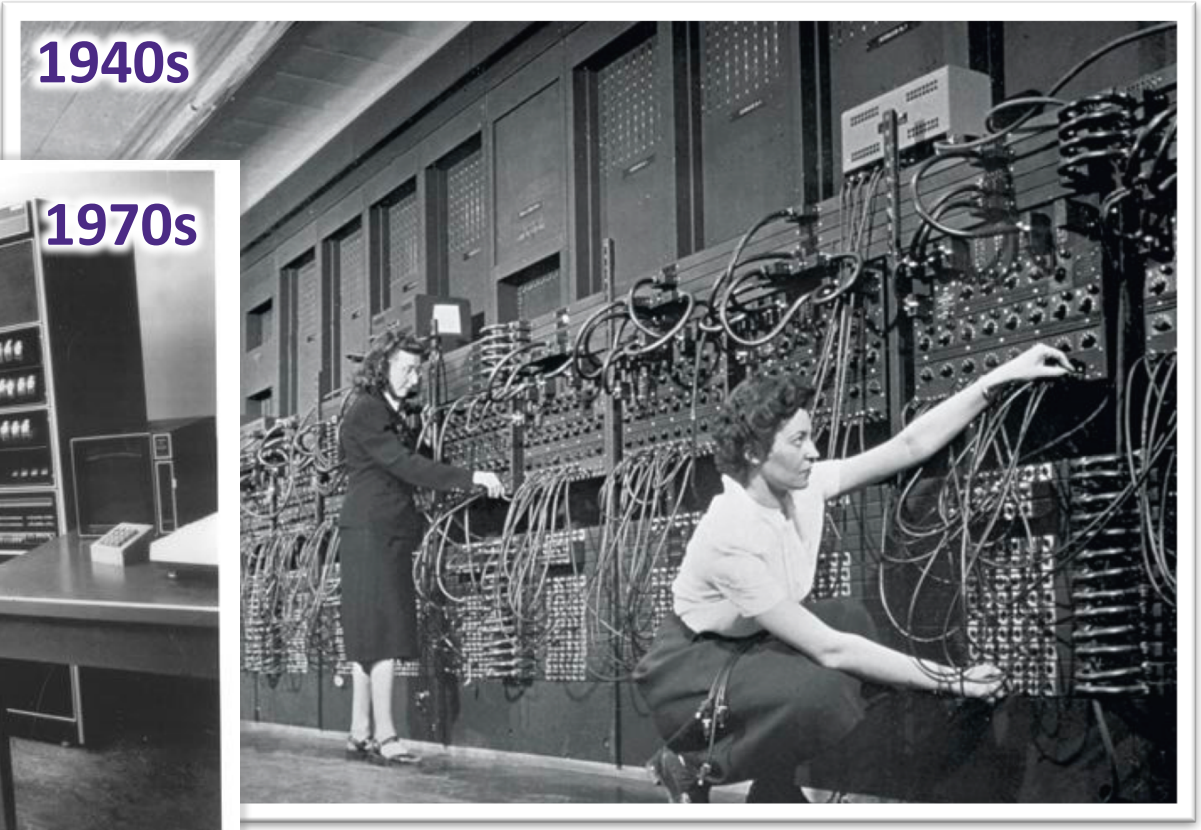
Assembler

```
10000011011111000010010000011100000000000
0111010000011000
10001011010001000010010000010100
10001011010001100010010100010100
10001101000001000000010
1000100111000010
11000001111110100011111
11110111011111000010010000011100
10001001010001000010010000011000
```

- 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

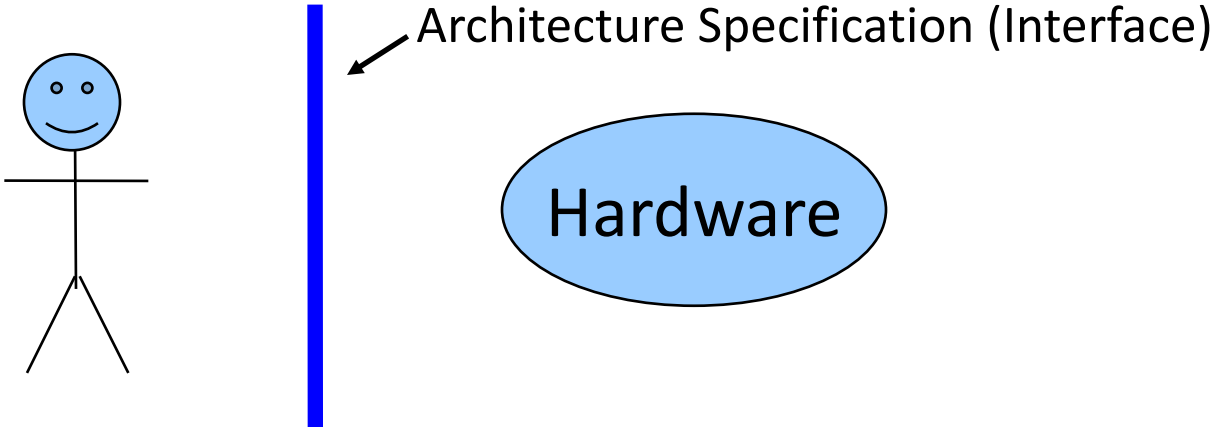


Jean Jennings (left), Marlyn Wescoff (center), and Ruth Lichterman program ENIAC at the University of Pennsylvania, circa 1946.
Photo: Corbis
http://fortune.com/2014/09/18/walter-isaacson-the-women-of-eniac/

https://s-media-cache-ak0.pinimg.com/564x/91/37/23/91372375e2e6517f8af128aab655e3b4.jpg

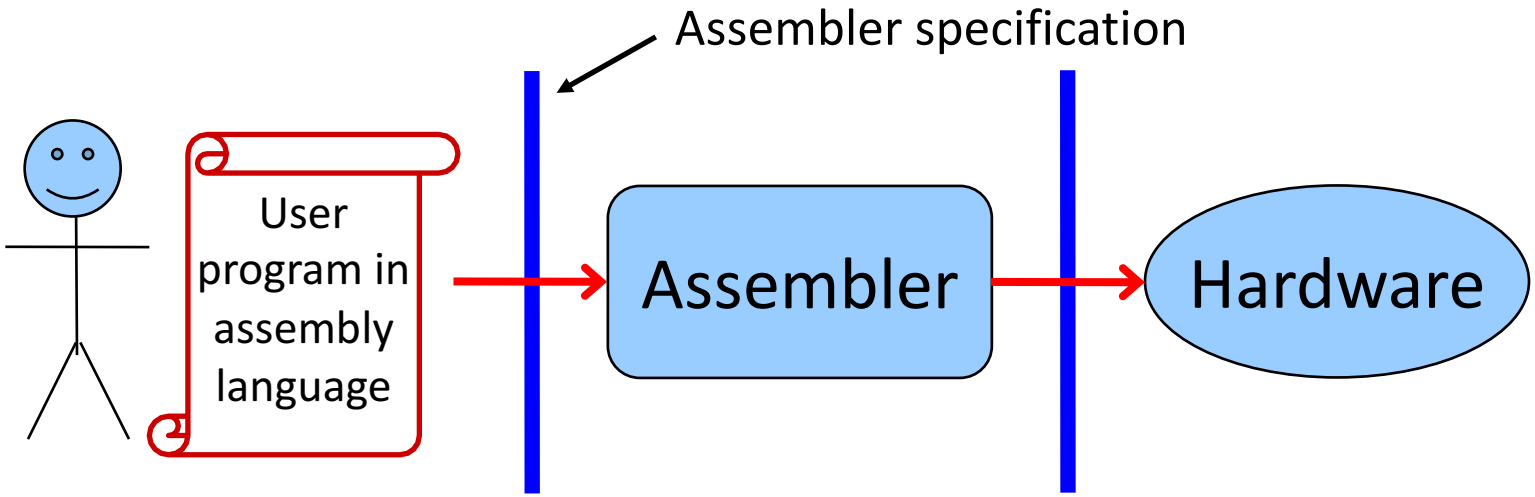
HW/SW Interface: Historical Perspective

- 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



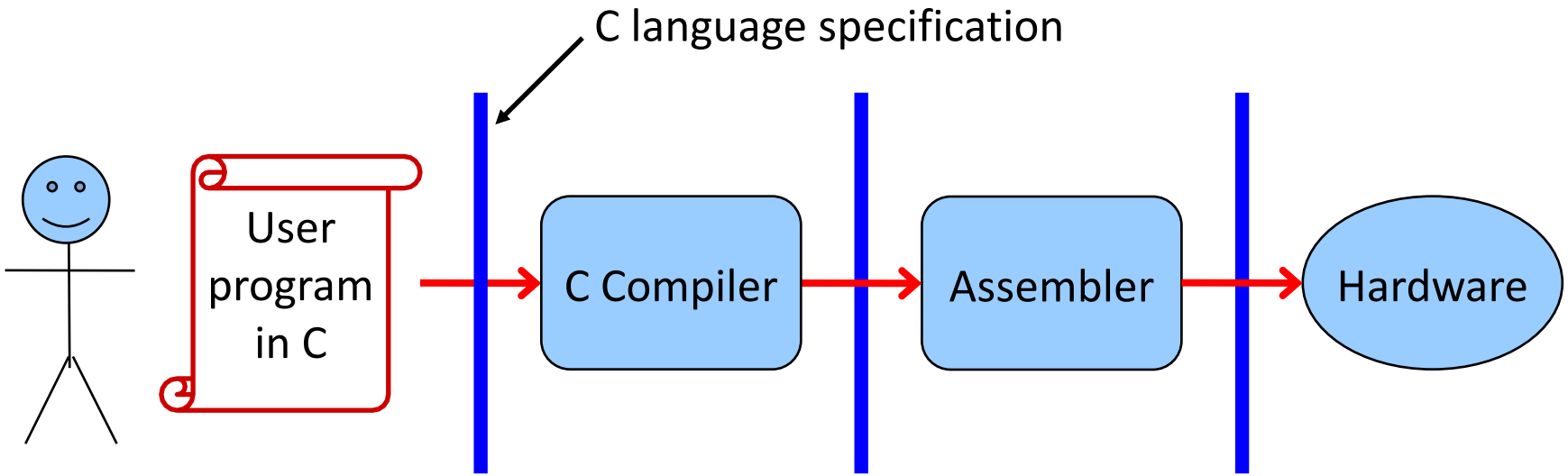
HW/SW Interface: Assemblers

- 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

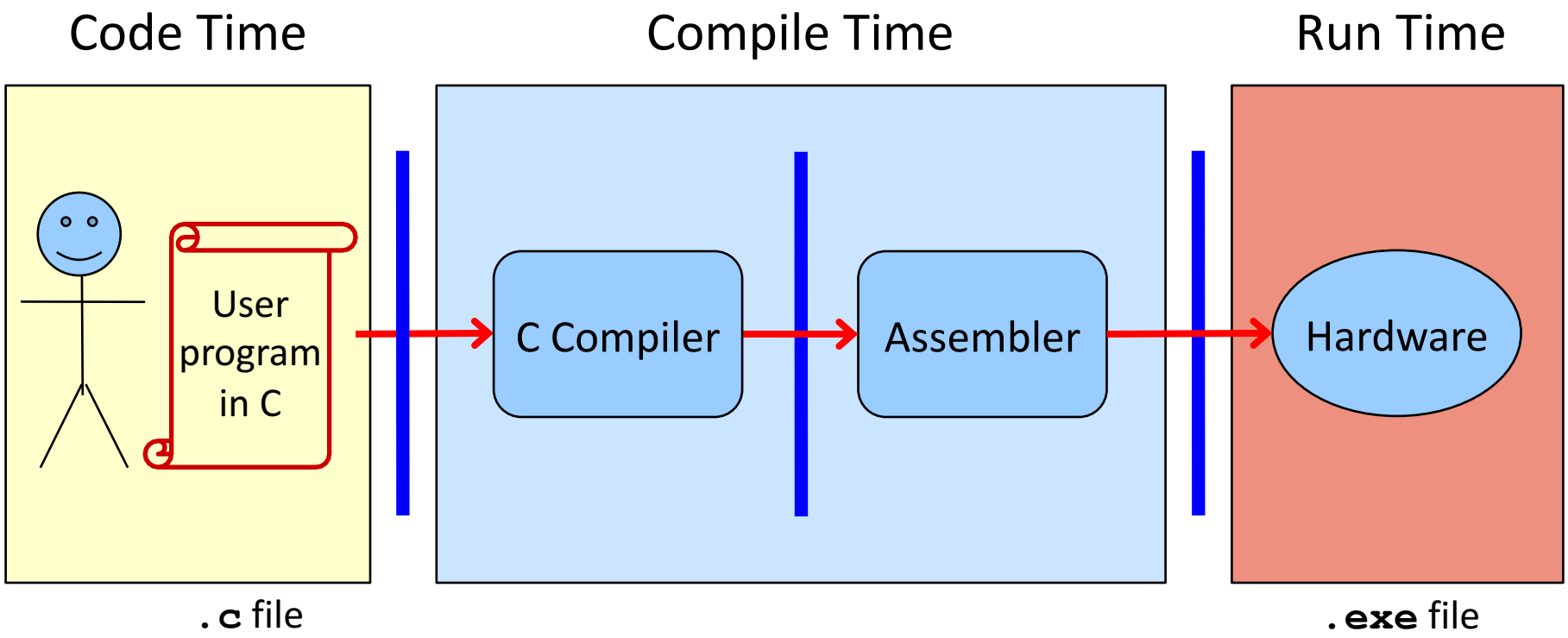


HW/SW Interface: Higher-Level Languages

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

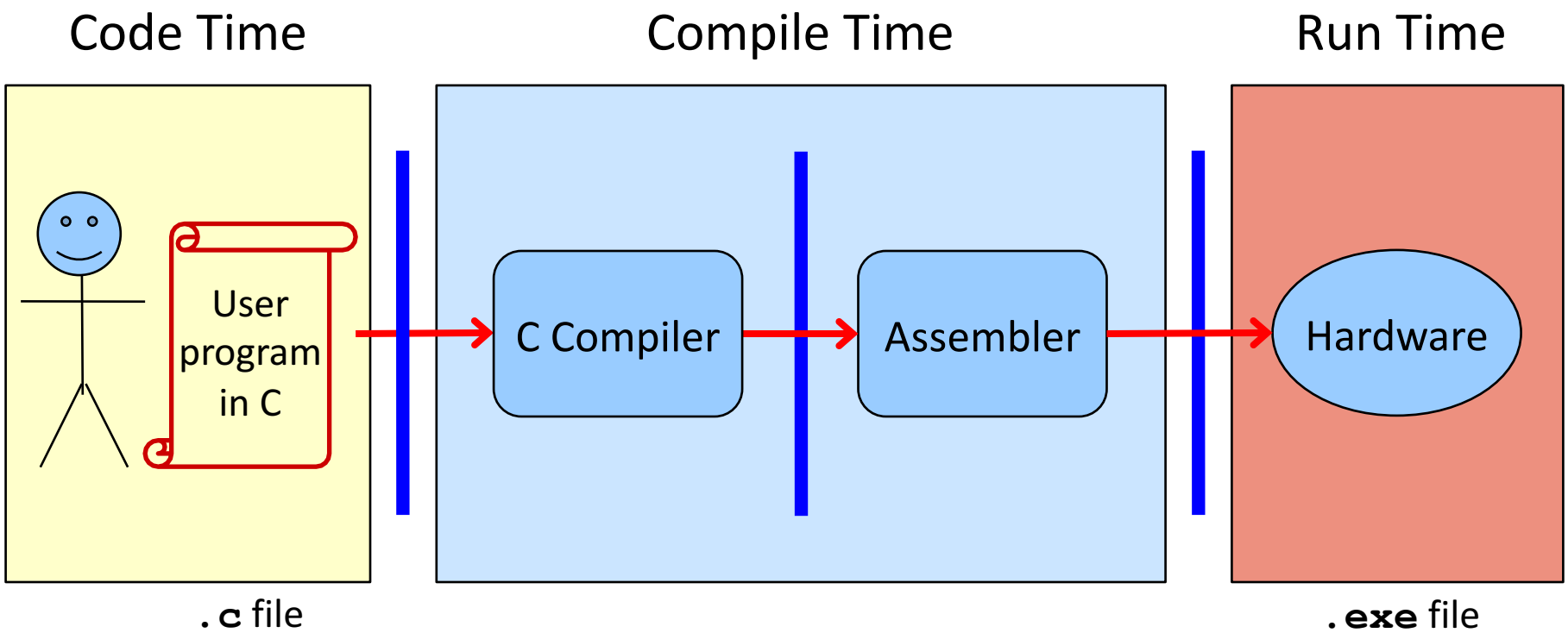


HW/SW Interface: Compiled Programs



How can you make a C program run faster?

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)
- Why do we need abstractions? Which ones do 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:

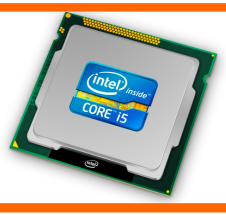
```
get_mpg:
    pushq   %rbp
    movq    %rsp, %rbp
    ...
    popq    %rbp
    ret
```

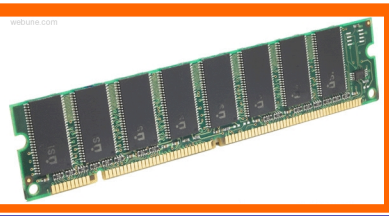
Machine code:

```
0111010000011000
100011010000010000000010
1000100111000010
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
- 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 2017???

- Chances are, you'll never write a program in assembly
  - Compilers are much better and more patient than you are
  - Unless you are writing delicate, "special" code ☺
- 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

Some fun topics that we will touch on

- Which of the following seems the most interesting to you?

a) What is a GFLOP and why is it used in computer benchmarks?
b) How and why does running many programs for a long time eat into your memory (RAM)?
c) What is stack overflow and how does it happen?
d) Why does your computer slow down when you run out of *disk* space?
e) What was the flaw behind the original Internet worm and the Heartbleed bug?
f) What is the meaning behind the different CPU specifications? (e.g. # of cores, # and size of cache, supported memory types)
g) What is going on in computer architecture research?

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, Ruth Anderson, Dan Grossman, Brandon Holt, Justin Shia
  - Not listed: hundreds of TAs