The Hardware/Software Interface

CSE 351 Autumn 2016

Instructor:

Justin Hsia

Teaching Assistants:

Chris Ma

Hunter Zahn

John Kaltenbach

Kevin Bi

Sachin Mehta

Suraj Bhat

Thomas Neuman

Waylon Huang

Xi Liu

Yufang Sun

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?
  - 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

- Your Instructor: just call me Justin
  - Just arrived from California (UC Berkeley and the Bay Area)
  - I like: teaching, the outdoors, board games, and ultimate
  - Excited to be teaching at UW for the first time!
- 10 TAs: 

  - 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

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
  - Not listed: hundreds of TAs

Who are You?

- ~ 220 students registered, split across two lectures
  - 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

Communication

- Website: http://cs.uw.edu/351
  - Schedule, policies, sections, links, assignments, etc.
- Discussion: https://piazza.com/washington/fall2016/cse351
  - 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
  - Can send to individual staff member of whole staff

Course Components

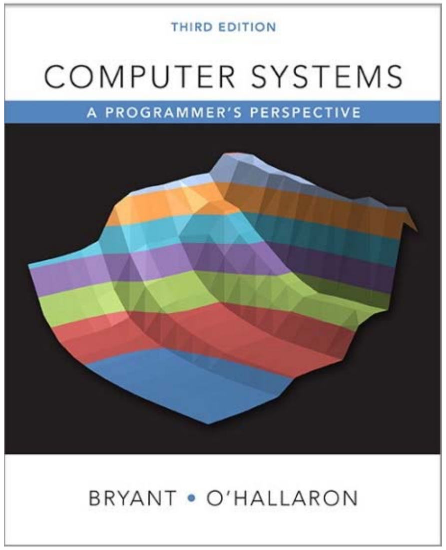
- Lectures (29)
  - Introduce the concepts; supplemented by textbook
- Sections (9-10)
  - Applied concepts, important tools and skills for labs, clarification of lectures, exam review and preparation
- Written homework assignments (4)
  - Mostly problems from textbook to solidify understanding
- Programming lab assignments (6)
  - Provide in-depth understanding (via practice) of an aspect of system
- Exams (2)
  - **Midterm:** Wednesday, November 2, in lecture
  - **Final:** Tuesday, December 13, 12:30-2:20pm (joint)

Policies

- **Exams:** Midterm (15%) and Final (30%)
  - Many old exams on course website (though new instructor)
- **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 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

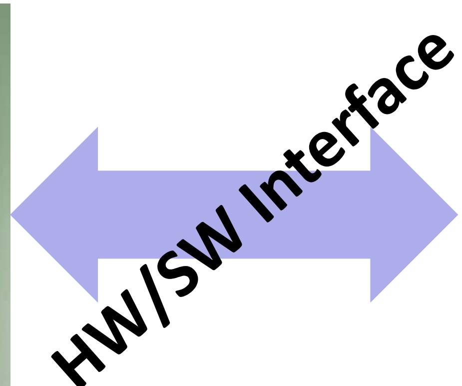
- Explore website thoroughly: http://cs.uw.edu/351
- Check that you are enrolled in Piazza
- Start-of-Course survey [Catalyst] due Friday (9/30)
- Section 1 is tomorrow
  - **Install the virtual machine (VM) *before* coming to section**
  - Bring your computer with you to section
- Lab 0 released today, due Monday (10/3) @ 5pm
  - Basic exercises to *start* getting familiar with C – need the VM
  - Credit/no-credit
  - Do ASAP, attending Section 1 will help

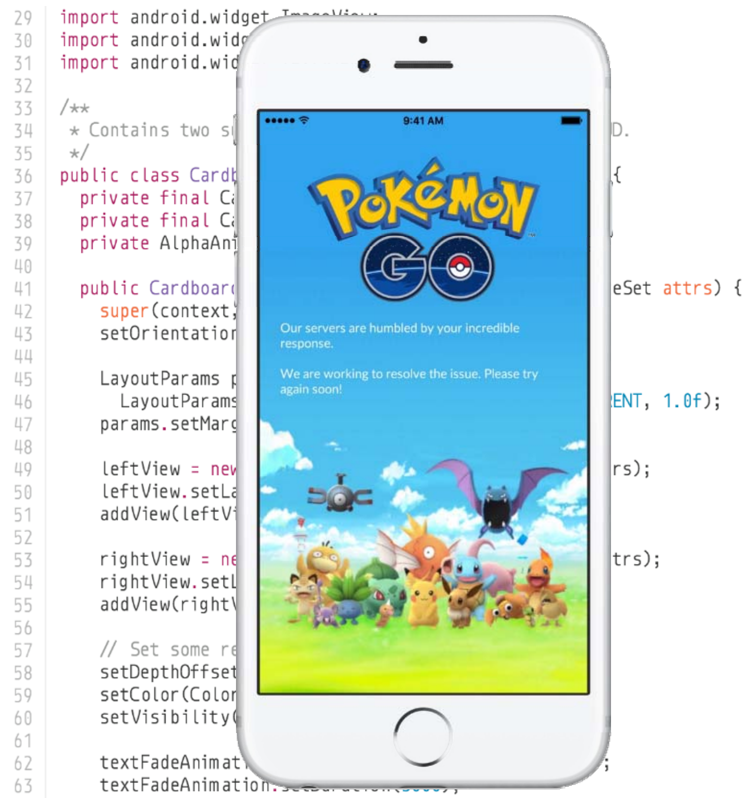
The Hardware/Software Interface

- 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?

10000011011111000010010000011100000000000
0111010000011000

110000011111101000011111
1111011101111100001001000001100





C/Java, assembly, and machine code

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

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

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:
```

Assembly Language

Assembler

```
1000001101111110000100100000111000000000
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

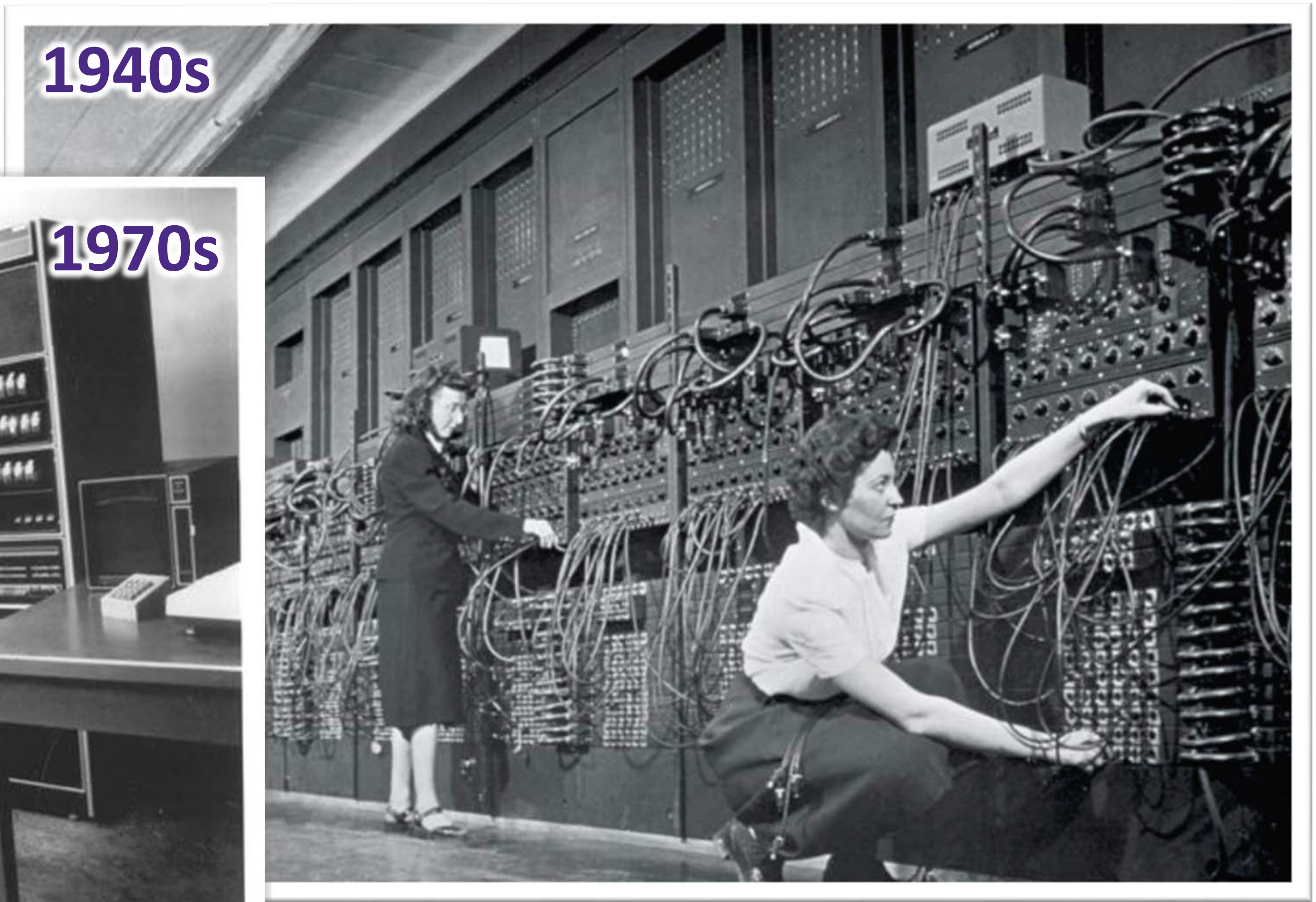
```
1000001101111100001001000001110000000000
0111010000011000
10001011010001000010010000010100
10001011010001100010010100010100
100011010000010000000010
1000100111000010
110000011111101000011111
11110111011111000010010000011100
10001001010001000010010000011000
```

- All program fragments are equivalent
- <u>You'd</u> 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

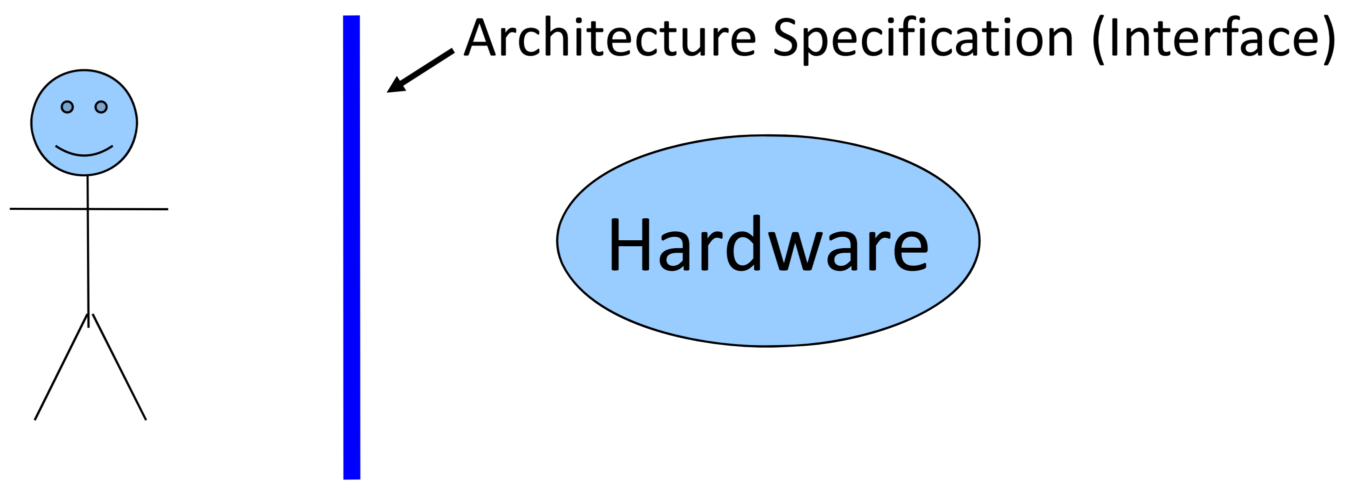
https://s-media-cache-ak0.pinimg.com/564x/91/37/23/91372375e2e6517f8af128aab655e3b4.jpg



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/

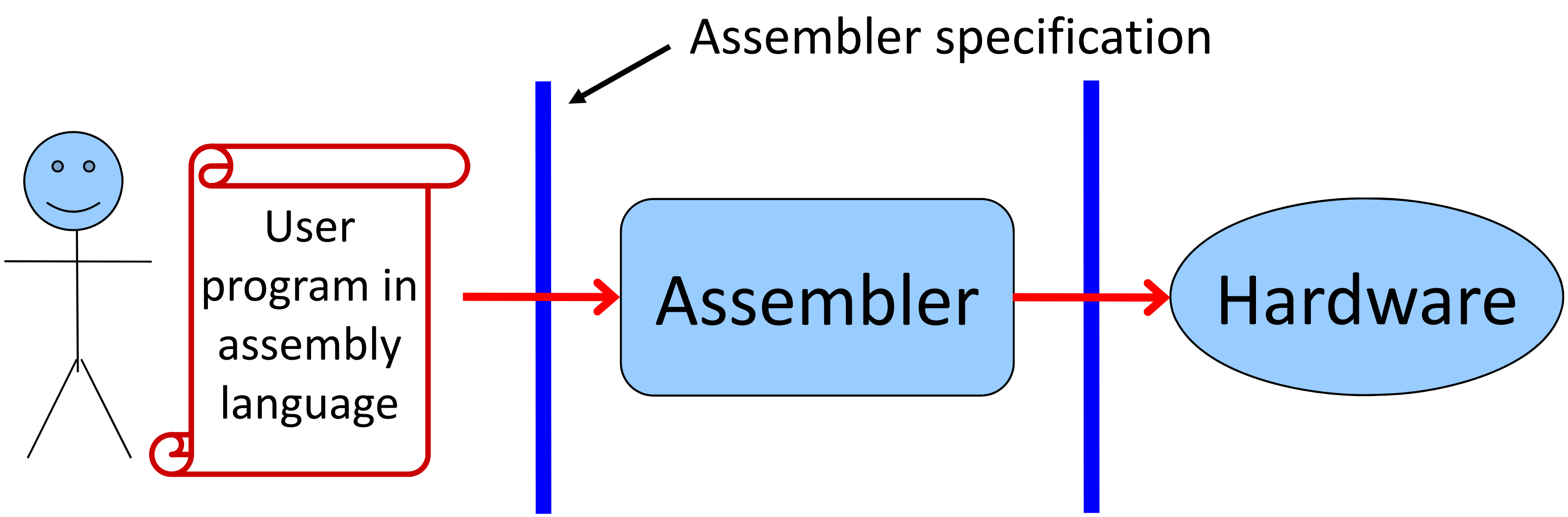
HW/SW Interface: Historical Perspective

- 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



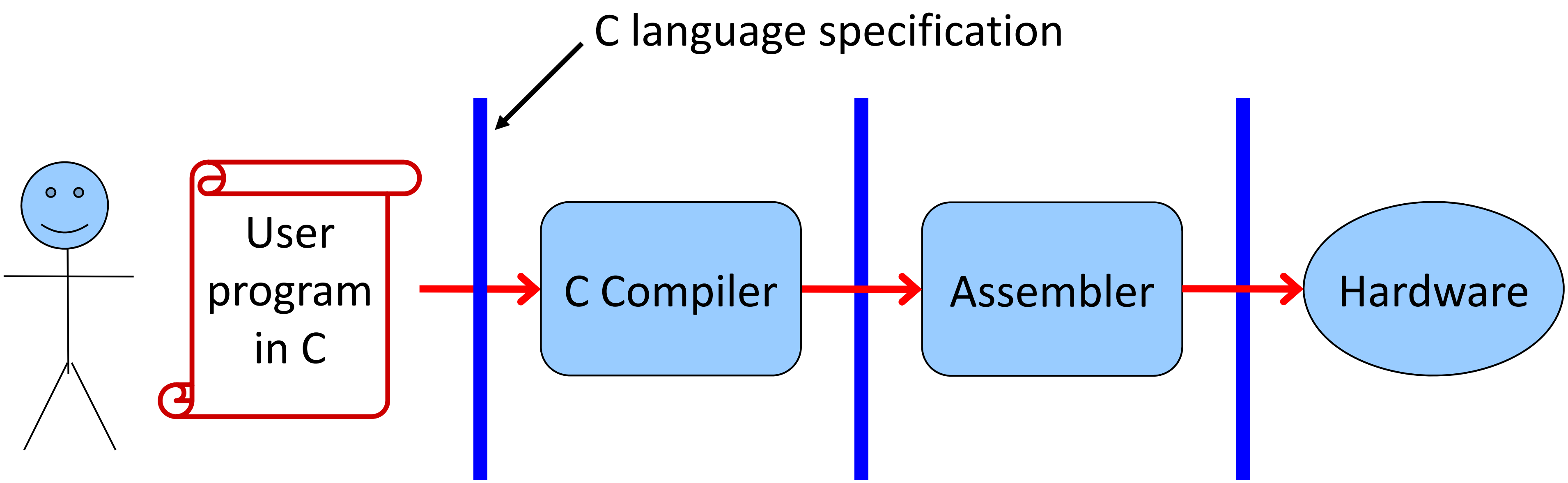
HW/SW Interface: Assemblers

- 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

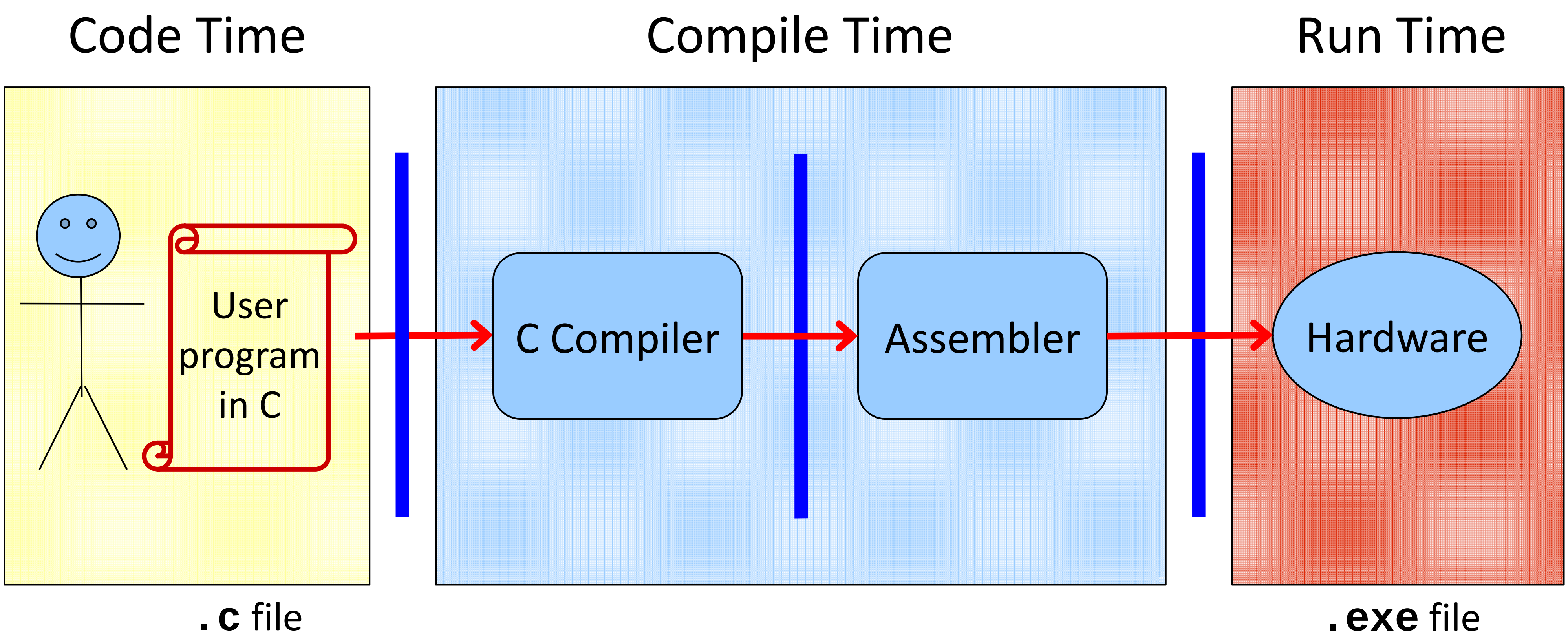


HW/SW Interface: Higher-Level Languages

- 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



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 <u>you</u> 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
```

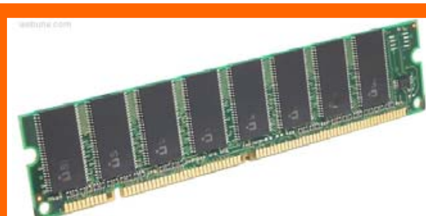
Machine code:

```
0111010000011000
100011010000010000000010
1000100111000010
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 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

Some fun topics that we will touch on

- Which of the following seems the most interesting to you? (vote at http://PollEv.com/justinhsia468)

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)