# The Hardware/Software Interface

CSE 351 Winter 2018

**Instructor:**

Mark Wyse

**Teaching Assistants:**

Kevin Bi

Parker DeWilde

Emily Furst

Sarah House

Waylon Huang

Vinny Palaniappan


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 executes 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

- Your Instructor: call me Mark
- TAs:
  - Available in section, office hours, via email, on Piazza
  - An invaluable source of information and help
- Get to know us
  - We are here to help you succeed!

# About Me

- CSE PhD student, Computer Architecture
- Washington native
- Food lover – I'll try to cook almost anything

- Post-Grad Scholar at AMD Research during 2017
  - Also, 18 of past 24 months
  - Future GPU microarchitecture for compute applications

- Teaching 351 for the first time!
  - TA'd in Wi13, Wi14, and Su14 (Coursera offering)

# Who are You?

- ~ 115 students registered, single lecture
  - 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
- 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

# Registration

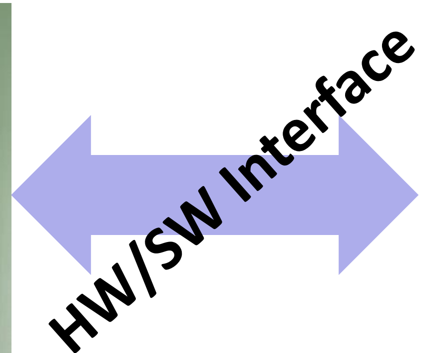
- If you need to register for the course:
  - https://goo.gl/forms/L7dG4a9RYfJzucZ72
- There is an option for EE – 351 only
- Non-majors: select 'Other' for major
- Continue to attend lectures!
- Go to a section with open space tomorrow
- See me after class to write down UW-ID

# The Hardware/Software Interface

- Why do we need a hardware/software interface?
- Why do we need to understand both sides of this interface?

1000001101111100001001000001110000000000
0111010000011000

110000011111101000011111
11110111011111000010010000011100




# 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

```
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:
```

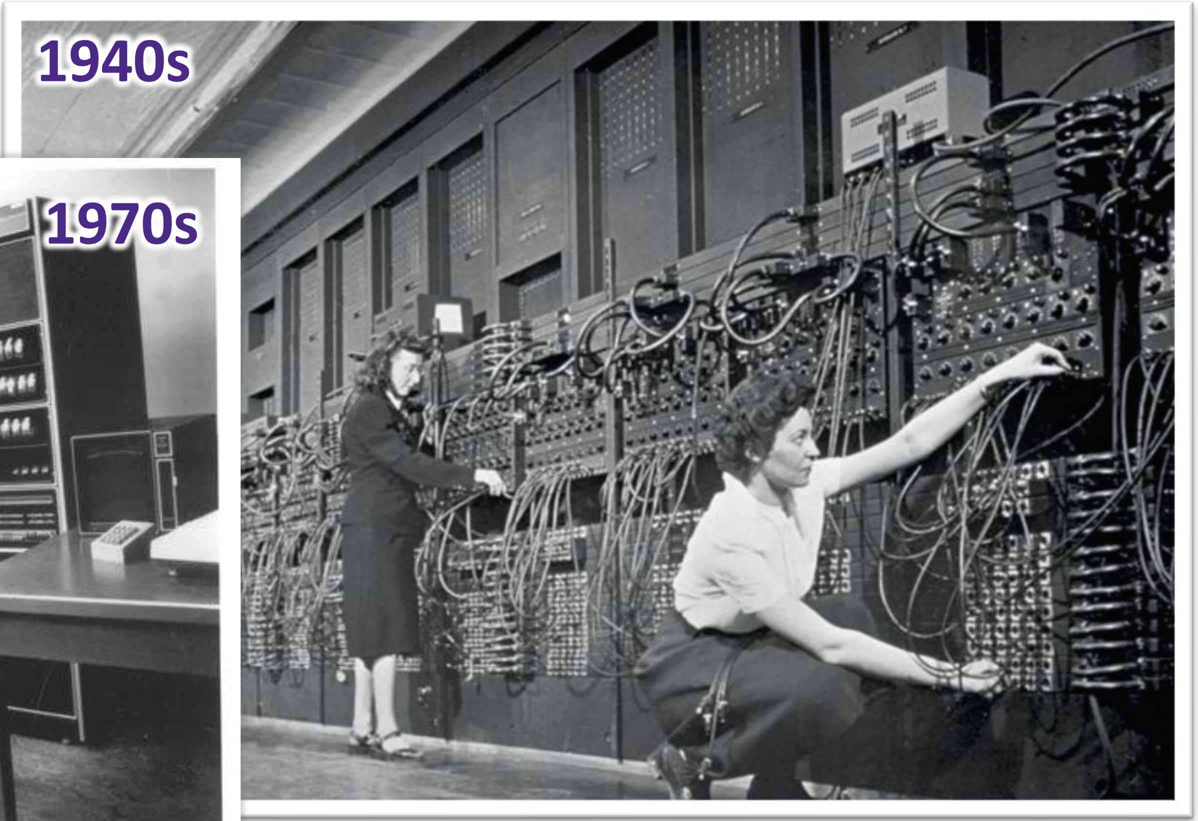
Assembler

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

- All program fragments are equivalent
- You'd rather write C! (more human-friendly)
- Hardware executes strings of bits
  - The machine instructions are actually much shorter than the number of bits we would need to represent the characters in the assembly language
  - In reality everything is voltages and electrical signals

# HW/SW Interface: Historical Perspective

- Hardware started out quite primitive

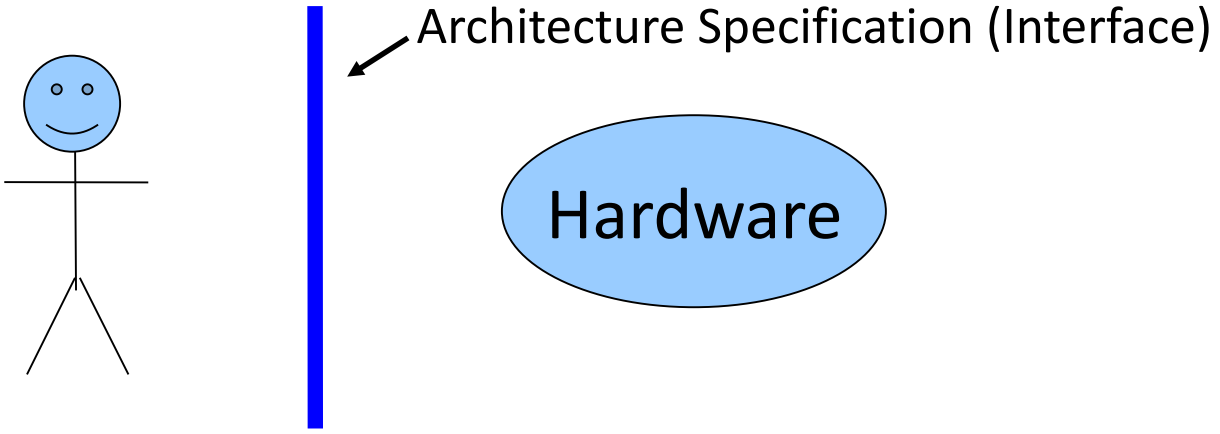



Jean Jennings (left), Marlyn Wescoff (center), and Ruth Lichterman program ENIAC at the University of Pennsylvania, circa 1946.
Photo: Corbis
http://fortune.com/2014/09/18/walter-isaacson-the-women-of-eniac/


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

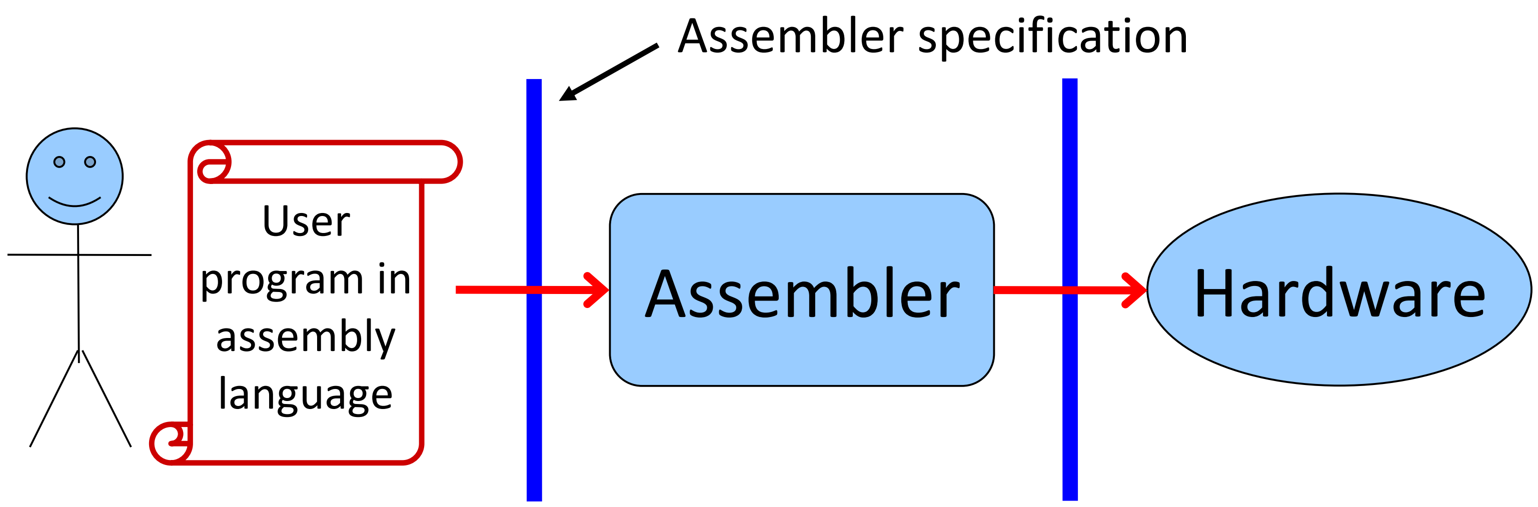
# HW/SW Interface: Historical Perspective

- 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



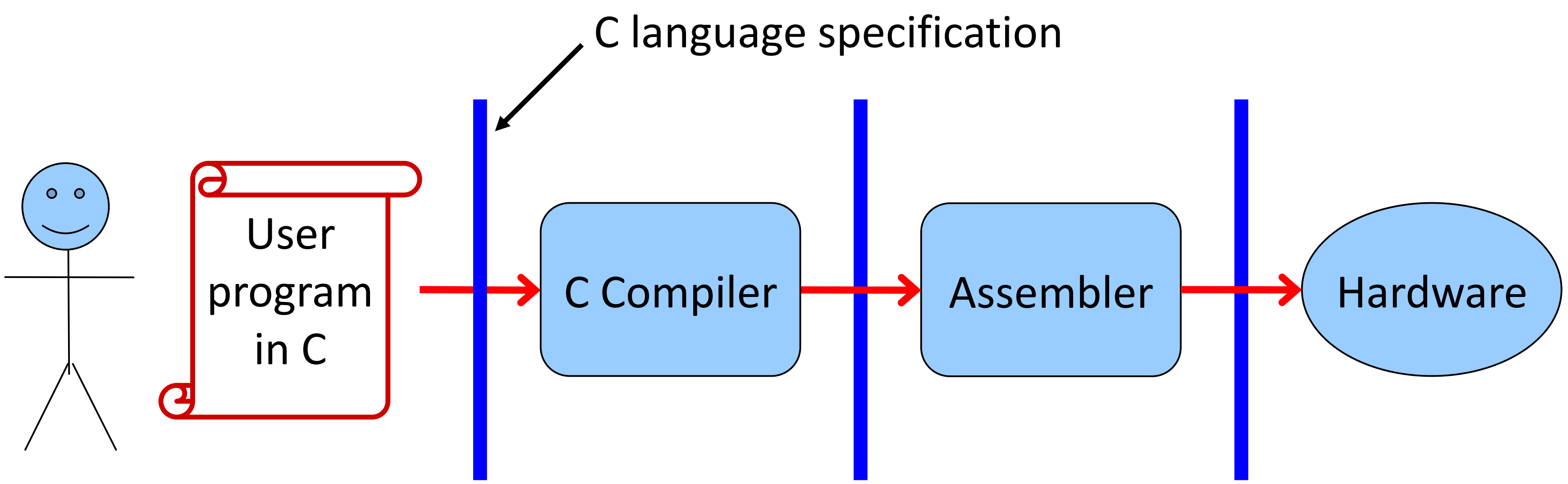
# HW/SW Interface: Assemblers

- 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

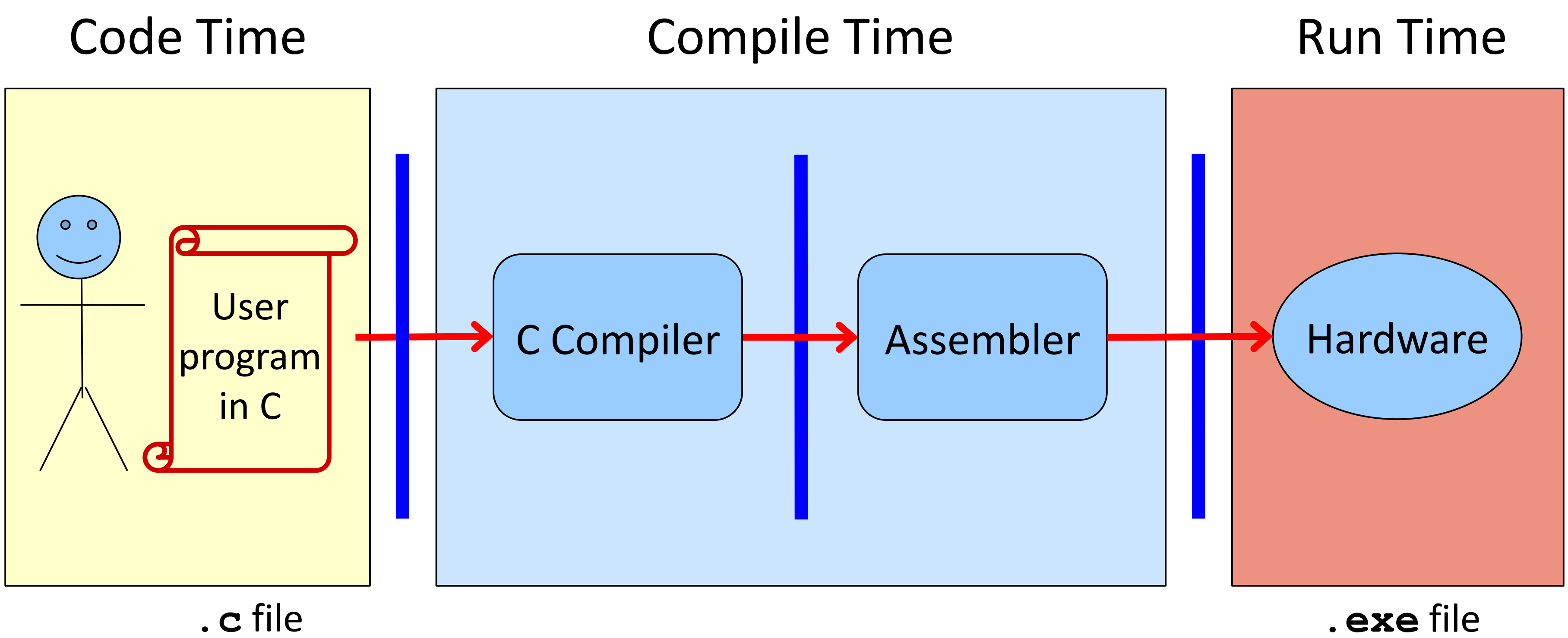


# HW/SW Interface: Higher-Level Languages

- 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




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

# 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();
```

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

Assembly language:

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

OS:

Windows 10
OS X Yosemite

Machine code:

```
0111010000011000
100011010000010000000010
1000100111000010
110000011111101000011111
```

Computer system:


# Course Perspective

- CSE351 will make you a better programmer
  - Purpose is to show how software really works
    - 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
  - 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)
  - "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

# Lecture Outline

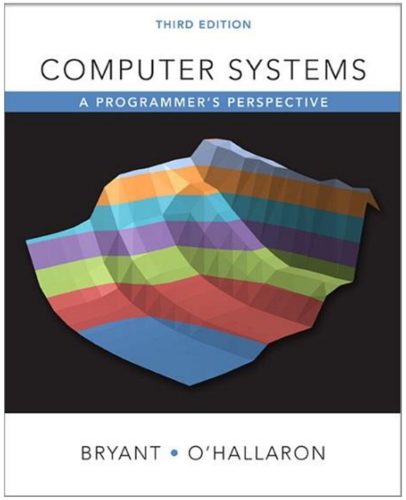
- Course Introduction
- **Course Policies**
  - https://courses.cs.washington.edu/courses/cse351/18wi/syllabus/
- Binary

# Communication

- Website: http://cs.uw.edu/351
  - Schedule, policies, materials, videos, assignments, etc.
- Discussion: http://piazza.com/washington/winter2018/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

# 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 and homework




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

# Course Components

- Lectures (26)
  - Introduce the concepts; supplemented by textbook
- Sections (10)
  - Applied concepts, important tools and skills for labs, clarification of lectures, exam review and preparation
- Online homework assignments (5)
  - Problems to solidify understanding; submitted as Canvas quizzes
- Programming lab assignments (5.5)
  - Provide in-depth understanding (via practice) of an aspect of system
- Exams (2)
  - **Midterm:** Monday, February 5, in class
  - **Final:** Wednesday, March 14, 2:30-4:20pm (UW assigned time/location)

# Grading

- **Homework:** 20% total
  - Autograded; 20 submission attempts
  - *Group work okay*
- **Labs:** 35% total
  - Graded by TAs; last submission graded
  - *Individual work only*
- **Exams:** Midterm (15%) and Final (30%)
  - Many old exams on course website (soon)
- More details on course website

# Due Dates and Late Work Policy

- **Homework**
  - No late days/submissions.
- **Labs**
  - Turn in by the deadline, or 20% per day penalty
  - **No penalty-free late days**
  - 20% off per day, through 4th day after due date
  - Score = min(graded score, 100% - 20% * num_late_days)
    - num_late_days = ceil(hours late / 24)
    - E.g., if you receive 89%, but you turned it in within 24 hours after due date, your score will be 80%
- **Complete assignments by their due date!**

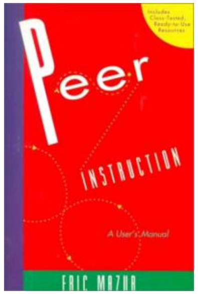
# Collaboration and Academic Integrity

- All submissions are expected to be yours and yours alone
- You are encouraged to discuss your assignments with other students (*ideas*), but we expect that what you turn in is yours
- It is NOT acceptable to copy solutions from other students or to copy (or start your) solutions from the Web (including Github)
- Our goal is that *YOU* learn the material so you will be prepared for exams, interviews, and the future

# Course Environment and Culture

- Simple rules for our course:
  - Respect one another
  - Ask questions
  - Have fun!
- If at any point you feel uncomfortable, disrespected, excluded, etc. by any staff member or another student, please report the incident so we may address the issue and maintain a supportive and inclusive learning environment.
  - Contact: staff (direct or anonymous), CSE undergraduate advising, UW Office of the Ombud

# Peer Instruction




- Increase real-time learning in lecture, test your understanding, increase student interactions
  - Lots of research supports its effectiveness
- Multiple choice question at end of lecture "segment"
  - 1 minute to decide on your own
  - 2-4 minutes in pairs to reach consensus
  - Learn through discussion

- In-person voting during lecture
  - May switch to PollEverywhere if the in-person thing doesn't work well

# Some fun topics that we will touch on

- Which of the following seems the most interesting to you? (show of hands)

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, the Heartbleed bug, and the Cloudbleed bug?
f) What is the meaning behind the different CPU specifications? (e.g. # of cores, # and size of cache, supported memory types)

# Tips for Success in 351

- Attend all lectures and sections
  - Avoid devices during lecture (i.e., listen, engage, and ask questions)
- Learn by doing
  - Can answer many questions by writing small programs
- Visit Piazza often
  - Ask questions and try to answer fellow students' questions
- Go to office hours
  - Even if you don't have specific questions in mind
- Find a study and homework group
- Start assignments early
- Don't be afraid to ask questions

# To-Do List

- Admin
  - Explore/read website *thoroughly*: http://cs.uw.edu/351
  - Check that you are enrolled in Piazza
  - **Get your machine set up for this class (VM or attu) *as soon as possible***
- Assignments
  - Pre-Course Survey due Friday (1/5)
  - Lab 0 due Monday (1/8)
  - HW 1 due Wednesday (1/10)

# Other Details

- Consider taking CSE 391 Unix Tools, 1 credit
  - Useful skills to know and relevant to this course
  - Available to all CSE majors and anyone **registered** in this CSE351
  - If you are interested in taking this, attend the first lecture!!

# Lecture Outline

- Course Introduction
- Course Policies
- **Binary**
  - **Decimal, Binary, and Hexadecimal**
  - **Base Conversion**
  - **Binary Encoding**

# Decimal Numbering System

- Ten symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

- Represent larger numbers as a sequence of digits
  - Each digit is one of the available symbols

- Example: 7061 in decimal (base 10)
  - $7061_{10} = (7 \times 10^3) + (0 \times 10^2) + (6 \times 10^1) + (1 \times 10^0)$

# Octal Numbering System

- Eight symbols: 0, 1, 2, 3, 4, 5, 6, 7
  - Notice that we no longer use 8 or 9
- Base comparison:
  - Base 10: `0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12…`
  - Base 8: `0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14…`
- Example: What is $7061_8$ in base 10?
  - $7061_8 = (7 \times 8^3) + (0 \times 8^2) + (6 \times 8^1) + (1 \times 8^0) = 3633_{10}$

# Peer Instruction Question

- What is $34_8$ in base 10?

  A. $32_{10}$
  B. $34_{10}$
  C. $7_{10}$
  D. $28_{10}$
  E. $35_{10}$

- Think on your own for a minute, then discuss with your neighbor(s)

# Binary and Hexadecimal

- Binary is base 2
  - Symbols: 0, 1
  - Convention: $2_{10} = 10_2 =$ 0b10
- Example: What is 0b110 in base 10?
  - 0b110 = $110_2 = (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) = 6_{10}$

- Hexadecimal (hex, for short) is base 16
  - Symbols? 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, **A, B, C, D, E, F**
  - Convention: $16_{10} = 10_{16} =$ 0x10
- Example: What is 0xA5 in base 10?
  - 0xA5 = $A5_{16} = (10 \times 16^1) + (5 \times 16^0) = 165_{10}$

# Peer Instruction Question

- Which of the following orderings is correct?

  A. 0xC < 0b1010 < 11
  B. 0xC < 11 < 0b1010
  C. 11 < 0b1010 < 0xC
  D. 0b1010 < 11 < 0xC
  E. 0b1010 < 0xC < 11

- Think on your own for a minute, then discuss with your neighbor(s)

# Converting to Base 10

- Can convert from any base *to* base 10
  - 0b110 = $110_2 = (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) = 6_{10}$
  - 0xA5 = $A5_{16} = (10 \times 16^1) + (5 \times 16^0) = 165_{10}$
- We learned to think in base 10, so this is fairly natural for us
- **Challenge:** Convert into other bases (e.g. 2, 16)

# Challenge Question

- Convert $13_{10}$ into binary

- Hints:
  - $2^3 = 8$
  - $2^2 = 4$
  - $2^1 = 2$
  - $2^0 = 1$

**$13_{10}$ = ?**
**13 = 8 + 4 + 1**
**Binary: 0b 1 1 0 1**
**Dec: 8 4 1**

- Think on your own for a minute, then discuss with your neighbor(s)

# Converting from Decimal to Binary

- Given a decimal number N:
  - List increasing powers of 2 from *right to left* until $\geq$ N
  - Then from *left to right*, ask is that (power of 2) $\leq$ N?
    - If **YES**, put a 1 below and subtract that power from N
    - If **NO**, put a 0 below and keep going

- Example: 13 to binary

| $2^4$=16 | $2^3$=8 | $2^2$=4 | $2^1$=2 | $2^0$=1 |
|---|---|---|---|---|
| **0** | **1** | **1** | **0** | **1** |

# Converting from Decimal to Base B

- Given a decimal number N:
  - List increasing powers of B from *right to left* until $\geq$ N
  - Then from *left to right*, ask is that (power of B) $\leq$ N?
    - If **YES**, put *how many* of that power go into N and subtract from N
    - If **NO**, put a 0 below and keep going

- Example: 165 to hex

| $16^2$=256 | $16^1$=16 | $16^0$=1 |
|---|---|---|
| **0** | **A** | **5** |

# Converting Binary ↔ Hexadecimal

- Hex → Binary
  - Substitute hex digits, then drop any leading zeros
  - Example: 0x2D to binary
    - 0x2 is 0b0010, 0xD is 0b1101
    - Drop two leading zeros, answer is 0b101101
- Binary → Hex
  - Pad with leading zeros until multiple of 4, then substitute each group of 4
  - Example: 0b101101
    - Pad to 0b 0010 1101
    - Substitute to get 0x2D

| Base 10 | Base 2 | Base 16 |
|---|---|---|
| 0 | 0000 | 0 |
| 1 | 0001 | 1 |
| 2 | 0010 | 2 |
| 3 | 0011 | 3 |
| 4 | 0100 | 4 |
| 5 | 0101 | 5 |
| 6 | 0110 | 6 |
| 7 | 0111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| 10 | 1010 | A |
| 11 | 1011 | B |
| 12 | 1100 | C |
| 13 | 1101 | D |
| 14 | 1110 | E |
| 15 | 1111 | F |

# Binary → Hex Practice

- Convert 0b100110110101101
  - How many digits? **15**
  - Pad: **0100 1101 1010 1101**
  - Substitute: **0x4DAD**

| Base 10 | Base 2 | Base 16 |
|---|---|---|
| 0 | 0000 | 0 |
| 1 | 0001 | 1 |
| 2 | 0010 | 2 |
| 3 | 0011 | 3 |
| 4 | 0100 | 4 |
| 5 | 0101 | 5 |
| 6 | 0110 | 6 |
| 7 | 0111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| 10 | 1010 | A |
| 11 | 1011 | B |
| 12 | 1100 | C |
| 13 | 1101 | D |
| 14 | 1110 | E |
| 15 | 1111 | F |

# Base Comparison

- Why does all of this matter?
  - *Humans* think about numbers in **base 10**, but *computers* "think" about numbers in **base 2**
  - Binary encoding is what allows computers to do all of the amazing things that they do!
- You should have this table memorized by the end of the class
  - Might as well start now!

| Base 10 | Base 2 | Base 16 |
|---|---|---|
| 0 | 0000 | 0 |
| 1 | 0001 | 1 |
| 2 | 0010 | 2 |
| 3 | 0011 | 3 |
| 4 | 0100 | 4 |
| 5 | 0101 | 5 |
| 6 | 0110 | 6 |
| 7 | 0111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| 10 | 1010 | A |
| 11 | 1011 | B |
| 12 | 1100 | C |
| 13 | 1101 | D |
| 14 | 1110 | E |
| 15 | 1111 | F |

# Numerical Encoding

- **AMAZING FACT: You can represent *anything* countable using numbers!**
  - Need to agree on an encoding
  - Kind of like learning a new language

- <u>Examples</u>:
  - Decimal Integers: 0→0b0, 1→0b1, 2→0b10, etc.
  - English Letters: CSE→0x435345, yay→0x796179
  - Emoticons: 😀 0x0, 😔 0x1, 😎 0x2, 😇 0x3, 😈 0x4, 🙋‍♀️ 0x5

# Binary Encoding

- With N binary digits, how many "things" can you represent?
  - Need N binary digits to represent $n$ things, where $2^N \geq n$
  - Example: 5 binary digits for alphabet because $2^5 = 32 > 26$

- A binary digit is known as a bit
- A group of 4 bits (1 hex digit) is called a nibble
- A group of 8 bits (2 hex digits) is called a byte
  - 1 bit → 2 things, 1 nibble → 16 things, 1 byte → 256 things

# So What's It Mean?

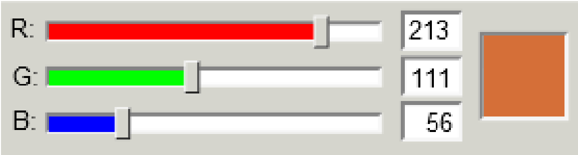
- *A sequence of bits can have many meanings!*

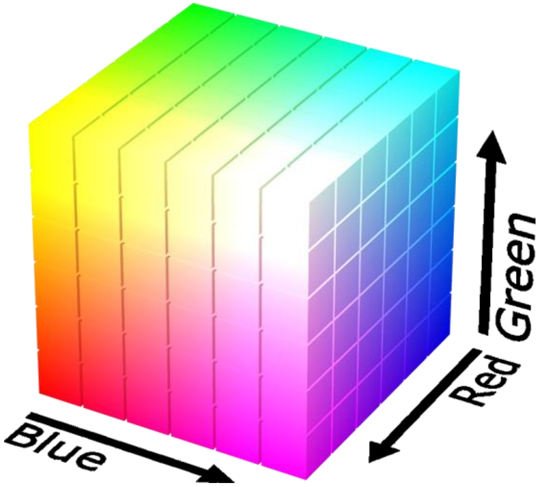
- Consider the hex sequence 0x4E6F21
  - Common interpretations include:
    - The decimal number 5140257
    - The characters "No!"
    - The background color of this slide
    - The real number $7.203034 \times 10^{-39}$

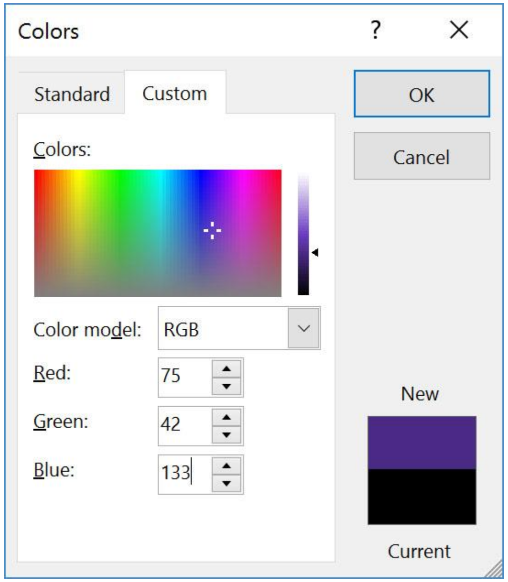
- It is up to the program/programmer to decide how to interpret the sequence of bits

# Binary Encoding – Colors

- RGB – Red, Green, Blue
  - Additive color model (light): byte (8 bits) for each color
  - Commonly seen in hex (in HTML, photo editing, etc.)
  - Examples: **Blue**→0x0000FF, **Gold**→0xFFD700, **White**→0xFFFFFF, **Deep Pink**→0xFF1493



# Binary Encoding – Characters/Text

- ASCII Encoding (www.asciitable.com)
  - American Standard Code for Information Interchange

| Dec | Hx | Oct | Char |
|---|---|---|---|
| 0 | 0 | 000 | NUL (null) |
| 1 | 1 | 001 | SOH (start of heading) |
| 2 | 2 | 002 | STX (start of text) |
| 3 | 3 | 003 | ETX (end of text) |
| 4 | 4 | 004 | EOT (end of transmission) |
| 5 | 5 | 005 | ENQ (enquiry) |
| 6 | 6 | 006 | ACK (acknowledge) |
| 7 | 7 | 007 | BEL (bell) |
| 8 | 8 | 010 | BS (backspace) |
| 9 | 9 | 011 | TAB (horizontal tab) |
| 10 | A | 012 | LF (NL line feed, new line) |
| 11 | B | 013 | VT (vertical tab) |
| 12 | C | 014 | FF (NP form feed, new page) |
| 13 | D | 015 | CR (carriage return) |
| 14 | E | 016 | SO (shift out) |
| 15 | F | 017 | SI (shift in) |
| 16 | 10 | 020 | DLE (data link escape) |
| 17 | 11 | 021 | DC1 (device control 1) |
| 18 | 12 | 022 | DC2 (device control 2) |
| 19 | 13 | 023 | DC3 (device control 3) |
| 20 | 14 | 024 | DC4 (device control 4) |
| 21 | 15 | 025 | NAK (negative acknowledge) |
| 22 | 16 | 026 | SYN (synchronous idle) |
| 23 | 17 | 027 | ETB (end of trans. block) |
| 24 | 18 | 030 | CAN (cancel) |
| 25 | 19 | 031 | EM (end of medium) |
| 26 | 1A | 032 | SUB (substitute) |
| 27 | 1B | 033 | ESC (escape) |
| 28 | 1C | 034 | FS (file separator) |
| 29 | 1D | 035 | GS (group separator) |
| 30 | 1E | 036 | RS (record separator) |
| 31 | 1F | 037 | US (unit separator) |

| Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|
| 32 | 20 | 040 |   | Space |
| 33 | 21 | 041 | ! | ! |
| 34 | 22 | 042 | " | " |
| 35 | 23 | 043 | # | # |
| 36 | 24 | 044 | $ | $ |
| 37 | 25 | 045 | % | % |
| 38 | 26 | 046 | & | & |
| 39 | 27 | 047 | ' | ' |
| 40 | 28 | 050 | ( | ( |
| 41 | 29 | 051 | ) | ) |
| 42 | 2A | 052 | * | * |
| 43 | 2B | 053 | + | + |
| 44 | 2C | 054 | , | , |
| 45 | 2D | 055 | - | - |
| 46 | 2E | 056 | . | . |
| 47 | 2F | 057 | / | / |
| 48 | 30 | 060 | 0 | 0 |
| 49 | 31 | 061 | 1 | 1 |
| 50 | 32 | 062 | 2 | 2 |
| 51 | 33 | 063 | 3 | 3 |
| 52 | 34 | 064 | 4 | 4 |
| 53 | 35 | 065 | 5 | 5 |
| 54 | 36 | 066 | 6 | 6 |
| 55 | 37 | 067 | 7 | 7 |
| 56 | 38 | 070 | 8 | 8 |
| 57 | 39 | 071 | 9 | 9 |
| 58 | 3A | 072 | : | : |
| 59 | 3B | 073 | ; | ; |
| 60 | 3C | 074 | < | < |
| 61 | 3D | 075 | = | = |
| 62 | 3E | 076 | > | > |
| 63 | 3F | 077 | ? | ? |

| Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|
| 64 | 40 | 100 | @ | @ |
| 65 | 41 | 101 | A | A |
| 66 | 42 | 102 | B | B |
| 67 | 43 | 103 | C | C |
| 68 | 44 | 104 | D | D |
| 69 | 45 | 105 | E | E |
| 70 | 46 | 106 | F | F |
| 71 | 47 | 107 | G | G |
| 72 | 48 | 110 | H | H |
| 73 | 49 | 111 | I | I |
| 74 | 4A | 112 | J | J |
| 75 | 4B | 113 | K | K |
| 76 | 4C | 114 | L | L |
| 77 | 4D | 115 | M | M |
| 78 | 4E | 116 | N | N |
| 79 | 4F | 117 | O | O |
| 80 | 50 | 120 | P | P |
| 81 | 51 | 121 | Q | Q |
| 82 | 52 | 122 | R | R |
| 83 | 53 | 123 | S | S |
| 84 | 54 | 124 | T | T |
| 85 | 55 | 125 | U | U |
| 86 | 56 | 126 | V | V |
| 87 | 57 | 127 | W | W |
| 88 | 58 | 130 | X | X |
| 89 | 59 | 131 | Y | Y |
| 90 | 5A | 132 | Z | Z |
| 91 | 5B | 133 | [ | [ |
| 92 | 5C | 134 | \ | \ |
| 93 | 5D | 135 | ] | ] |
| 94 | 5E | 136 | ^ | ^ |
| 95 | 5F | 137 | _ | _ |

| Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|
| 96 | 60 | 140 | ` | ` |
| 97 | 61 | 141 | a | a |
| 98 | 62 | 142 | b | b |
| 99 | 63 | 143 | c | c |
| 100 | 64 | 144 | d | d |
| 101 | 65 | 145 | e | e |
| 102 | 66 | 146 | f | f |
| 103 | 67 | 147 | g | g |
| 104 | 68 | 150 | h | h |
| 105 | 69 | 151 | i | i |
| 106 | 6A | 152 | j | j |
| 107 | 6B | 153 | k | k |
| 108 | 6C | 154 | l | l |
| 109 | 6D | 155 | m | m |
| 110 | 6E | 156 | n | n |
| 111 | 6F | 157 | o | o |
| 112 | 70 | 160 | p | p |
| 113 | 71 | 161 | q | q |
| 114 | 72 | 162 | r | r |
| 115 | 73 | 163 | s | s |
| 116 | 74 | 164 | t | t |
| 117 | 75 | 165 | u | u |
| 118 | 76 | 166 | v | v |
| 119 | 77 | 167 | w | w |
| 120 | 78 | 170 | x | x |
| 121 | 79 | 171 | y | y |
| 122 | 7A | 172 | z | z |
| 123 | 7B | 173 | { | { |
| 124 | 7C | 174 | | | \| |
| 125 | 7D | 175 | } | } |
| 126 | 7E | 176 | ~ | ~ |
| 127 | 7F | 177 | &#127; | DEL |



Source: www.LookupTables.com

# Binary Encoding – Files and Programs

- At the lowest level, all digital data is stored as bits!

- Layers of abstraction keep everything comprehensible
  - Data/files are groups of bits interpreted by program
  - Program is actually groups of bits being interpreted by your CPU

- Computer Memory Demo (if time)
  - From vim: `%!xxd`
  - From emacs: `M-x hexl-mode`

# Summary

- Humans think about numbers in decimal; computers think about numbers in binary
  - Base conversion to go between them
  - Hexadecimal is more human-readable than binary
- All information on a computer is binary
- Binary encoding can represent *anything*!
  - Computer/program needs to know how to interpret the bits