Welcome to

CS 240:
Foundations of
Computer Systems

- Program, Application
- Programming Language
- Compiler/Interpreter
- Operating System
- Instruction Set Architecture
- Microarchitecture
- Digital Logic
- Devices (transistors, etc.)
- Solid-State Physics
Today

1. What is CS 240?
2. Why take CS 240?
3. How does CS 240 work?
4. Dive into foundations of computer hardware.
CS 111, 230, 231, 235, 251:

- What can a program do?
- How can a program solve a problem?
- How do you structure a program?
- How do you know it is correct or efficient?
- How hard is it to solve a problem?
- How is computation expressed?
- What does a program mean?
- ...

A BIG question is missing...
CS 240: How do computers work?

Welcome to DrJava. Working directory is /Users/bpw/courses/cs240/cs240f14
> run HelloWorld
Hello, world!
Big Idea: Abstraction

Layers manage complexity.
Big Idea: Abstraction

with a few recurring subplots

Simple, general interfaces:
– Hide complexity of efficient implementation.
– Make higher-level systems easy to build.
– But they are not perfect.

**Representation** of data and programs

**Translation** of data and programs

**Control flow** within/across programs

---

0s and 1s, electricity

- compilers, assemblers, decoders
- branches, procedures, OS
1800s

Ada Lovelace writes the first computer program

1820s

Charles Babbage designs Analytical Engine

Prototype of Analytical Engine, (was never actually built), Science Museum, London
Image: public domain

1840s

George Boole describes formal logic for computers

Boolean Algebra

1860s

Countess Ava Lovelace, 1840s
George Boole, 1860s
University College Cork, Ireland
Image: public domain
<table>
<thead>
<tr>
<th>Decades</th>
<th>Human Computers</th>
<th>Computing Machines</th>
</tr>
</thead>
<tbody>
<tr>
<td>1890s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1900s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1910s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1920s</td>
<td>Alan Turing, 1940s</td>
<td>Imitation Game, 2014</td>
</tr>
<tr>
<td>1930s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1940s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1950s</td>
<td>NASA computers, 1953</td>
<td>Hidden Figures, 2016</td>
</tr>
<tr>
<td>1960s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1970s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Image: Flikr [mark_am_kramer](https://www.flickr.com/photos/mark_am_kramer/), Imitation Game poster
Image: NASA/JPL/Caltech, Hidden Figures*
**ENIAC** (Electronic Numerical Integrator and Computer),
First Turing-complete all-electronic programmable digital computer.
University of Pennsylvania, 1940s

Image: public domain

<table>
<thead>
<tr>
<th>1940s</th>
<th>1950s</th>
<th>1960s</th>
<th>1970s</th>
<th>1980s</th>
<th>1990s</th>
<th>2000s</th>
<th>2010s</th>
<th>2020s</th>
</tr>
</thead>
<tbody>
<tr>
<td>data represented as electrical signals</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Jean Jennings Bartik and Frances Bilas Spence with part of ENIAC.

*The programmers of ENIAC were six women.*

http://eniacprogrammers.org/, http://sites.temple.edu/topsecretrosies/

Image: public domain
Programming 1940s-style *with switches and cables.*

Image: public domain
programs are data

machine translates instructions to control flow

Manchester “Baby” SSEM (Small-Scale Experimental Machine), replica first stored-program computer -- University of Manchester (UK), 1948
<table>
<thead>
<tr>
<th>1940s</th>
<th>1950s</th>
<th><strong>1960s</strong></th>
<th>1970s</th>
<th>1980s</th>
<th>1990s</th>
<th>2000s</th>
<th>2010s</th>
<th>2020s</th>
</tr>
</thead>
</table>

PDP-11 "minicomputers"

http://simh.trailing-edge.com/

http://www.pcworld.com/article/249951/if_it_aint_broke_dont_fix_it_ancient_computers_in_use_today.html?page=2
<table>
<thead>
<tr>
<th>1940s</th>
<th>1950s</th>
<th>1960s</th>
<th>1970s</th>
<th>1980s</th>
<th>1990s</th>
<th>2000s</th>
<th>2010s</th>
<th>2020s</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="1940s Image" /></td>
<td><img src="image2.png" alt="1950s Image" /></td>
<td><img src="image3.png" alt="1960s Image" /></td>
<td><img src="image4.png" alt="1970s Image" /></td>
<td><img src="image5.png" alt="1980s Image" /></td>
<td><img src="image6.png" alt="1990s Image" /></td>
<td><img src="image7.png" alt="2000s Image" /></td>
<td><img src="image8.png" alt="2010s Image" /></td>
<td><img src="image9.png" alt="2020s Image" /></td>
</tr>
</tbody>
</table>

Images:
- "Ibm pc 5150" by Ruben de Rijcke - Own work. Licensed under Creative Commons Attribution-Share Alike 3.0 via Wikimedia Commons - [Link](http://commons.wikimedia.org/wiki/File:Ibm_pc_5150.jpg)
- "IBM PC Motherboard (1981)" by German - Own work. Licensed under Creative Commons Attribution-Share Alike 3.0 via Wikimedia Commons - [Link](http://commons.wikimedia.org/wiki/File:IBM_PC_Motherboard_(1981).jpg)
- "Macintosh-motherboard" by Shieldforyoureyes Dave Fischer - Own work. Licensed under Creative Commons Attribution-Share Alike 3.0 via Wikimedia Commons - [Link](http://commons.wikimedia.org/wiki/File:Macintosh-motherboard.jpg)
<table>
<thead>
<tr>
<th></th>
<th>ENIAC</th>
<th>iPhone 5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Year</strong></td>
<td>1946</td>
<td>2012</td>
</tr>
<tr>
<td><strong>Weight</strong></td>
<td>30 tons</td>
<td>4 oz</td>
</tr>
<tr>
<td><strong>Volume</strong></td>
<td>2,400 ft³</td>
<td>3.4 in³</td>
</tr>
<tr>
<td><strong>Cost (USD, 2014)</strong></td>
<td>$6,000,000</td>
<td>$600</td>
</tr>
<tr>
<td><strong>Speed</strong></td>
<td>few 1000 ops/sec</td>
<td>2,500,000,000 ops/sec</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>~100 bytes</td>
<td>1,073,741,824 bytes (1 GB)</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td>150,000 W</td>
<td>&lt;5W</td>
</tr>
<tr>
<td><strong>Input/Output</strong></td>
<td>Switches, lights, later punchcards</td>
<td>Touchscreen, audio, camera, wifi, cell, ...</td>
</tr>
<tr>
<td><strong>Production</strong></td>
<td>1</td>
<td>5,000,000 sold in first 3 days</td>
</tr>
</tbody>
</table>
Modern Computer Organization

Processes executes instructions.

Stores program code + data during execution.

Input/Output

Persistent Storage

Network

USB

Display

...
Modern Computer Organization

Processor

Stores program code + data during execution.

Processor repeats:
1. fetch instruction
2. fetch data used by instruction
3. execute instruction on data
4. store result or choose next instruction
Physical implementation of instructions and resources.

Desired computation represented as instructions.

Software/Software Interface

Abstraction!
Microarchitecture (Implementation of ISA)
Instruction Set Architecture (HW/SW Interface)

**Instructions**
- Names, Encodings
- Effects
- Arguments, Results

**Local storage**
- Names, Size
- How many

**Large storage**
- Addresses, Locations

**Computer**
### Machine Instructions

The diagram illustrates a machine instruction in binary format, which adds two values and stores the result. The binary code for this instruction is: 00000010100010101100100000010000.

**Instruction Set Architecture specification**

The instruction is part of an Instruction Set Architecture (ISA) specification. The diagram shows a human figure representing the machine code program, which is translated into machine code by the hardware.

<table>
<thead>
<tr>
<th>1940s</th>
<th>1950s</th>
<th>1960s</th>
<th>1970s</th>
<th>1980s</th>
<th>1990s</th>
<th>2000s</th>
<th>2010s</th>
<th>2020s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Assemblers and Assembly Languages

addl %eax, %ecx           000001010001010110010000010000

Assembly Language specification

assembly program  Assembler  machine code program  Hardware
## Higher-Level Programming Languages

The diagram illustrates the process of converting a high-level language program into machine code, and then into machine instructions for execution on hardware. Here’s a breakdown:

### High-Level Language Program

- **Equation:** \( x = x + y; \)
- **Assembly Code:** `addl %eax, %ecx`
- **Binary:** `00000010100010101100100000010000`

### Compile Time

- **Compilation** takes a high-level language program and generates an assembly program.

### Run Time

- **Assembly program** is then converted to a machine code program using an assembler.

### Hardware

- The machine code program is executed on the hardware.

### Time Periods

<table>
<thead>
<tr>
<th>1940s</th>
<th>1950s</th>
<th>1960s</th>
<th>1970s</th>
<th>1980s</th>
<th>1990s</th>
<th>2000s</th>
<th>2010s</th>
<th>2020s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This timeline highlights the evolution of programming languages from the 1940s to the 2020s.
A-0: first compiler, by Grace Hopper

Early 1950s
Maybe closer to assembler/linker/loader

Later: B-0 ֮ FLOW-MATIC ֮ COBOL, late 50s

Jean Sammet also involved
• headed first sci comp group at Sperry in the ’50s
• Later first female president of ACM
• Mount Holyoke alum, class of 1948
More and more layers...

- Operating systems
- Virtual machines
- Hypervisors
- Web browsers
- ...
CS 240 in 3 acts
(4-5 weeks each)

Hardware *implementation*
  From transistors to a simple computer

Hardware-software *interface*
  From instruction set architecture to C

Abstraction for practical systems
  Memory hierarchy
  Operating systems
  Higher-level languages
I just like to program.

Why study the implementation?

It's fascinating, great for critical thinking.

System design principles apply to software too.

Sometimes system abstractions "leak." Implementation details affect your programs.
int ≠ integer
float ≠ real

int x=...;

x*x >= 0 ?
40000 * 40000 == 16000000000
50000 * 50000 == -1794967296

float a=..., b=..., c=...;

(a + b) + c == a + (b + c) ?
(-2.7e23 + 2.7e23) + 1.0 == 1.0
-2.7e23 + (2.7e23 + 1.0) == 0.0
Reliability?

Ariane 5 Rocket, 1996
Exploded due to cast of 64-bit floating-point number to 16-bit signed number. Overflow.

Boeing 787, 2015
"... a Model 787 airplane ... can lose all alternating current (AC) electrical power ... caused by a software counter internal to the GCUs that will overflow after 248 days of continuous power. We are issuing this AD to prevent loss of all AC electrical power, which could result in loss of control of the airplane."
--FAA, April 2015
void copyji(int src[2048][2048],
             int dst[2048][2048])
{
    int i,j;
    for (j = 0; j < 2048; j++)
        for (i = 0; i < 2048; i++)
            dst[i][j] = src[i][j];
}

void copyij(int src[2048][2048],
             int dst[2048][2048])
{
    int i,j;
    for (i = 0; i < 2048; i++)
        for (j = 0; j < 2048; j++)
            dst[i][j] = src[i][j];
}

several times faster
due to hardware caches
DETECTING GHOST VULNERABILITY

The **GHOST vulnerability** is a buffer overflow condition that can be easily exploited locally or remotely, which makes it extremely dangerous. This vulnerability is named after the GetHOS function involved in the exploit.

All computers are flawed -- and the fix will take years

by Selena Larson  @selenalarson
January 26, 2018: 12:07 PM ET

Meltdown and Spectre

To the long list of objects vulnerable to attack by computer hackers, add the human heart.

The threat seems largely theoretical. But a team of computer security researchers plans to report Wednesday that it had been able to gain wireless access to a combination heart defibrillator and pacemaker.
Why take CS 240?

• Learn how computers execute programs.
• Build software tools and appreciate the value of those you use.
• Deepen your appreciation of abstraction.
• Learn enduring system design principles.
• Improve your critical thinking skills.
• Become a better programmer:
  – Think rigorously about execution models.
  – Program carefully, defensively.
  – Debug and reason about programs effectively.
  – Identify limits and impacts of abstractions and representations.
  – Learn to use software development tools.
• Foundations for:
  – Compilers, security, computer architecture, operating systems, ...
• Have fun and feel accomplished!
https://cs.wellesley.edu/~cs240/

Everything is here.
Please read it.