Welcome to **CS 240**: Intro to Computer Systems

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

**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...
### 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
- 0s and 1s, electricity
- compilers, assemblers, decoders

**Translation** of data and programs
- branches, procedures, CS

**Control flow** within/across programs
- branches, procedures, CS

---

**Layers of virtual machines manage complexity.**

---

**Big Idea:**

*Abstraction*

- Interface
- Implementation

---

**ENIAC** (Electronic Numerical Integrator and Computer),
first Turing-complete all-electronic programmable digital computer:
University of Pennsylvania, 1940s

*Image: public domain*
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/

program controls general-purpose hardware

The 1940s-style programming with switches and cables

physical control flow

programs are data

machine translates instructions to control flow

Manchester “Baby” SSEM (Small-Scale Experimental Machine), replica of first stored-program computer – University of Manchester (UK), 1948

PDP-11 “minicomputers”
<table>
<thead>
<tr>
<th>Year</th>
<th>Weight</th>
<th>Volume</th>
<th>Cost (USD, 2014)</th>
<th>Speed</th>
<th>Memory</th>
<th>Power</th>
<th>Input/Output</th>
<th>Production</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>1946</td>
<td>30 tons</td>
<td>2,400 ft³</td>
<td>$6,000,000</td>
<td>few 1000 ops/sec</td>
<td>~100 bytes</td>
<td>150,000 W</td>
<td>Switches, lights, later punchcards</td>
<td>1</td>
<td>5,000,000 sold in first 3 days</td>
</tr>
<tr>
<td>2012</td>
<td>4 oz</td>
<td>3.4 in³</td>
<td>$600</td>
<td>2,500,000,000 ops/sec</td>
<td>1,073,741,824 bytes (1 GB)</td>
<td>&lt;5W</td>
<td>Touchscreen, audio, camera, wifi, cell, ...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Modern Computer Organization

- **Processor**: Executes instructions.
- **Memory**: Stores program code + data during execution.
- **Bus**: Connects components.
- **Input/Output**: Network, USB, Display, ...
Modern Computer Organization

- Processor: Executes instructions.
- Memory: 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

Desired computation represented as instructions.

Hardware/Software Interface

Physical implementation of instructions and resources.

Instruction Set Architecture (HW/SW Interface)

- Processor:
  - Instruction Logic
  - Registers

- Memory:
  - Encoded Instructions
  - Data

- Local storage:
  - Names, Size
  - How many

- Large storage:
  - Addresses, Locations

Microarchitecture (Implementation of ISA)
Machine Instructions

(adds two values and stores the result)

00000010100010101100100000010000

Instruction Set Architecture specification

Assemblers and Assembly Languages

```
addl $eax, $ecx 00000010100010101100100000010000
```

Assembly Language specification

Higher-Level Programming Languages

```
x = x + y;
```

Programming Language specification

```
addl $eax, $ecx 00000010100010101100100000010000
```

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
More and more layers...

- Operating systems
- Virtual machines
- Hypervisors
- Web browsers
- ...

I just like to program.

2 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.

CS 240: a 3-stage sprint

(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

int ≠ integer
float ≠ real

int x=...;
x*x >= 0 ?
40000 * 40000 == 1600000000
50000 * 50000 == -1794967296

float a=..., b=..., c=...;
(a + b) + c == a + (y + 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

Arithmetic Performance

\[ \frac{x}{973} \quad \frac{x}{1024} \]

Memory Performance

```c
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];
}
```

```c
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

Security

Security

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!
Also: C programming language

• Invented to build UNIX operating system, 1970s
  – OS manages hardware, C close to machine model

• Simple pieces look like Java:
  – if, while, for, local variables, assignment, etc.

• Other pieces do not:
  – structs vs. objects, functions vs. methods
  – addresses, pointers
  – no array bounds checks
  – weak type system

• Important language, still widely used, but many better PL ideas have come along since.

https://cs.wellesley.edu/~cs240/

Everything is here. Please read it.