The Plan

Welcome to
CS 240:
Foundations of
Computer Systems

Plan 1

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

Plan 2

Plan 3

Plan 4

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?

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

Plan 8

Big Idea: Abstraction

interface

Layers manage complexity.

Algorithm, Data Structure, Application

Programming Language

Compiler/Interpreter

Operating System

Instruction Set Architecture

Microarchitecture

Digital Logic

Devices (transistors, etc.)

Solid-State Physics

0s and 1s, electricity

compilers, assemblers, decoders

branches, procedures, OS

CS 111, 230, 231, 235, 251

Programming Language

Compiler/Interpreter

Operating System

Instruction Set Architecture

Microarchitecture

Digital Logic

Devices (transistors, etc.)

Solid-State Physics
Modern Computer Organization

**Processor**
- Executes instructions.

**Memory**
- Stores program code + data during execution.

**Bus**
- Input/Output
  - Persistent Storage
  - Network
  - USB
  - Display
  - ... (omitted)

**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

**Modern Computer Organization**

**Co-processor**
- Modern computer organization

**Hardware/Software Interface**
- Desired computation represented as instructions.

**Hardware**
- 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

---

**Machine Instructions**

(adds two values and stores the result)

000000010100010101101100100000010000

**Instruction Set Architecture specification**

---

**Assemblers and Assembly Languages**

- `addl %eax, %ecx
  00000010100010101100100000010000`

**Assembly Language specification**

---

**Higher-Level Programming Languages**

- `x = x + y;
  00000010100010101101100100000010000`

**Programming Language specification**

---
More and more layers...

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

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.

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

\[ x / 973 \quad x / 1024 \]

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];
}
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

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!

Security

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

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

Meltdown and Spectre

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

Everything is here.
Please read it.

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