Introduction to Computing Systems:

From Bits and Gates to C and Beyond

2nd Edition

Yale N. Patt

Sanjay J. Patel

Original slides from Gregory Byrd, North Carolina State University

Modified slides by Chris Wilcox, Colorado State University

Lecture Goals

- Review course logistics
  - Assignments
  - Policies
  - Organization
  - Grading Criteria
- Introduce key concepts
  - Role of Abstraction
  - Software versus Hardware
  - Universal Computing Devices
  - Layered Model of Computing

Logistics

- Lectures: See syllabus
- Staff: See syllabus
- Recitations: See syllabus
- Help session: See syllabus
- Office hours: See syllabus
- Materials on the website and Canvas:
  - http://www.cs.colostate.edu/~cs270

Assignments

Assignments and quizzes are posted on website:

- Weekly assignments (mostly) alternate between written and programming assignments.
- Homework assignments are due in hardcopy on original handout on Mon. at 10am.
- Programming assignments are submitted in electronic form Sun. at 10pm.
- Late submission varies depending on the difficulty of the assignment.

Policies

- Grading Criteria
  - Assignments (40%)
  - Recitations (10%)
  - Peer Instruction (5%)
  - Midterm Exam (20%)
  - Final Exam (25%)
- Late Policy
  - On-time = full points, late submission= 20% penalty
- Academic Integrity
  - http://www.cs.colostate.edu/~info/student-info.html
  - Do your own work
  - Be smart about Internet resources

Organization

- 1/3 computer hardware: numbers and bits, transistors, gates, digital logic, state machines, von Neumann model, instruction sets, LC-3 architecture
- 1/3 assembly code: instruction formats, branching and control, LC-3 programming, subroutines, memory model (stack)
- 1/3 C programming: data types, language syntax, variables and operators, control structures, functions, pointers and arrays, memory model, recursion, I/O, data structures

Grading Criteria

How to be successful in this class:

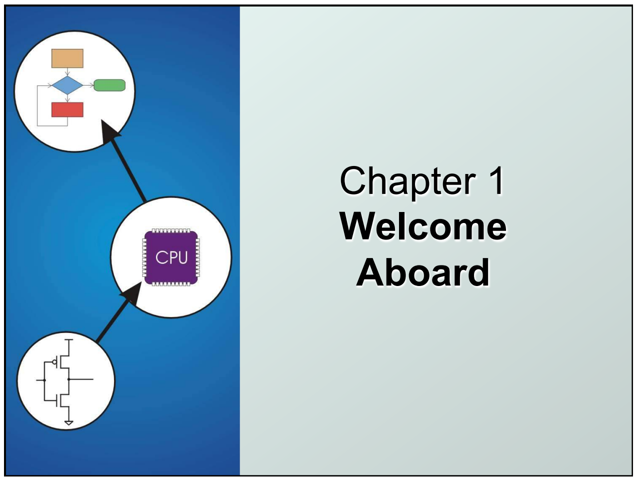
1) Attend all classes and recitations, info will presented that you can't get anywhere else.
2) Do all the homework assignments, ask questions (early!) if you run into trouble.
3) Take advantage of lab sessions where help is available from instructors.
4) Read the textbook, work through the end of chapter problems.

Computer Science Department Picnic

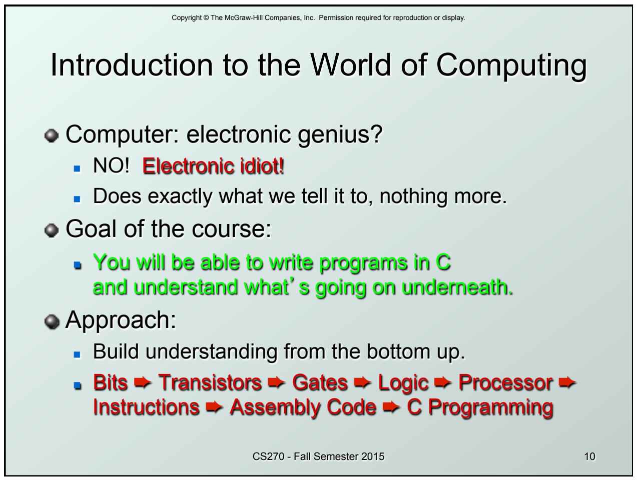
Welcome to the 2015-2016 Academic year !

Meet your faculty, department staff, and fellow students in a social setting. Food and drink will be provided.

When: Saturday, August 29th
Time: 11am – 2pm
Where: City Park Shelter #7

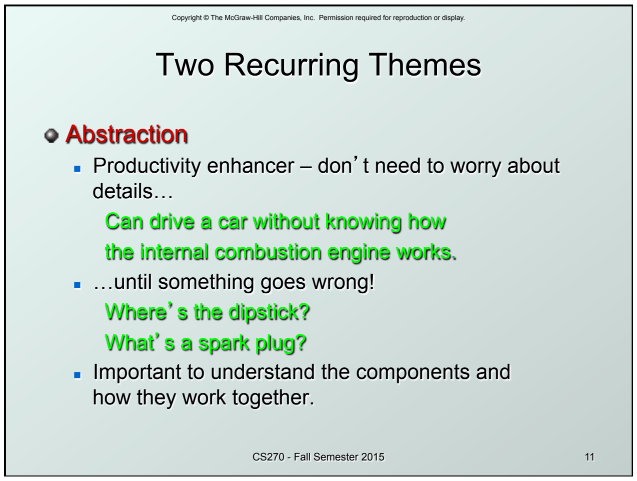


Chapter 1 **Welcome Aboard**



Introduction to the World of Computing

- Computer: electronic genius?
  - NO! Electronic idiot!
  - Does exactly what we tell it to, nothing more.
- Goal of the course:
  - You will be able to write programs in C and understand what's going on underneath.
- Approach:
  - Build understanding from the bottom up.
  - Bits ➨ Transistors ➨ Gates ➨ Logic ➨ Processor ➨ Instructions ➨ Assembly Code ➨ C Programming

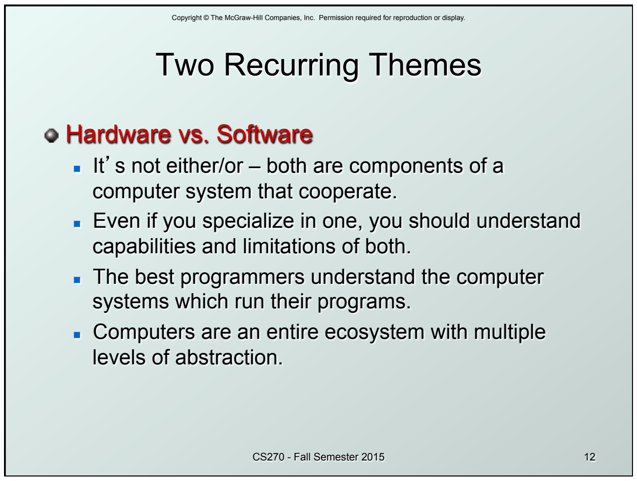


Two Recurring Themes

- Abstraction
  - Productivity enhancer – don't need to worry about details…

    Can drive a car without knowing how
    the internal combustion engine works.
  - …until something goes wrong!

    Where's the dipstick?
    What's a spark plug?
  - Important to understand the components and how they work together.

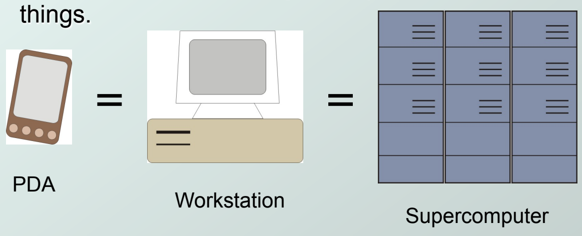


Two Recurring Themes

- Hardware vs. Software
  - It's not either/or – both are components of a computer system that cooperate.
  - Even if you specialize in one, you should understand capabilities and limitations of both.
  - The best programmers understand the computer systems which run their programs.
  - Computers are an entire ecosystem with multiple levels of abstraction.

Big Idea #1: Universal Computing Devices

- All computers, given enough time and memory, are capable of computing exactly the same things.



Turing Machine

- Mathematical model of a device that can perform any computation – Alan Turing (1937)
  - ability to read/write symbols on an infinite “tape”
  - state transitions, based on current state and symbol
- Every computation can be performed by some Turing machine. *(Turing’s thesis)*

$$\begin{tikzcd} a,b \arrow[r] & \boxed{T_{add}} \arrow[r] & a+b \end{tikzcd} \qquad \begin{tikzcd} a,b \arrow[r] & \boxed{T_{mul}} \arrow[r] & ab \end{tikzcd}$$

Turing machine that adds *Turing machine that multiplies*

For more info about Turing machines, see http://www.wikipedia.org/wiki/Turing_machine/

For more about Alan Turing, see http://www.turing.org.uk/turing/

Universal Turing Machine

- A machine that can implement all Turing machines -- this is also a Turing machine!
  - inputs: data, description of computation (other TMs)

$$\begin{tikzcd} T_{add}, T_{mul} \arrow[r] & \boxed{U} \arrow[r] & c(a+b) \\ a,b,c \arrow[ru] & & \end{tikzcd}$$

Universal Turing Machine

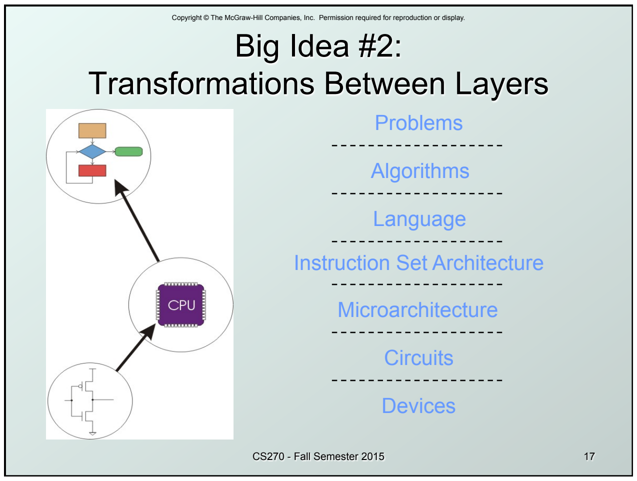
Universal machine is programmable – so is a computer!

- **instructions are part of the input data**
- **a computer can emulate a Universal Turing Machine**

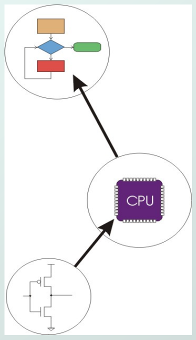
A computer is a universal computing device.

From Theory to Practice

- In theory, computer can ***compute*** anything
- that’s possible to compute
  - given enough *memory* and *time*
- In practice, ***solving problems*** involves computing under constraints.
  - time
    - weather forecast, next frame of animation, ...
  - cost
    - cell phone, automotive engine controller, ...
  - power
    - cell phone, handheld video game, ...



Big Idea #2: Transformations Between Layers



Problems

Algorithms

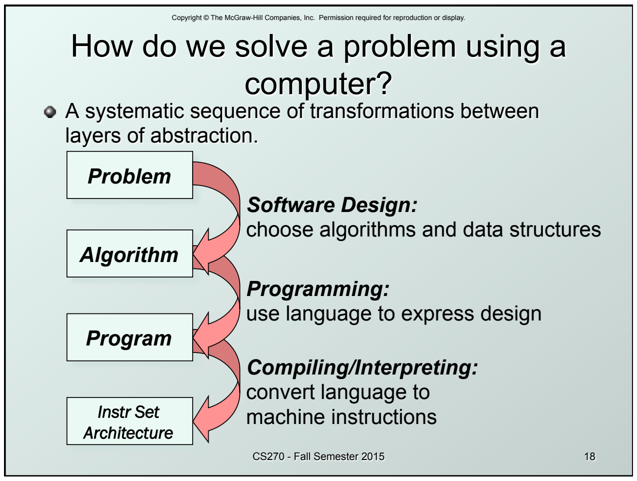
Language

Instruction Set Architecture

Microarchitecture

Circuits

Devices



How do we solve a problem using a computer?

- A systematic sequence of transformations between layers of abstraction.

Problem

Software Design: choose algorithms and data structures

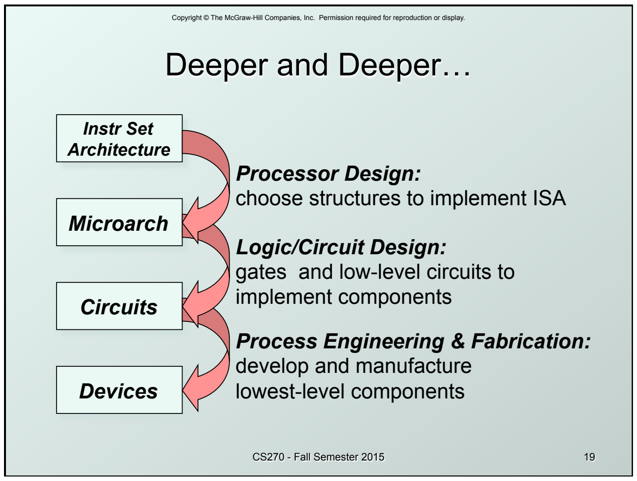
Algorithm

Programming: use language to express design

Program

Compiling/Interpreting: convert language to machine instructions

Instr Set Architecture



Deeper and Deeper…

Instr Set Architecture

Processor Design: choose structures to implement ISA

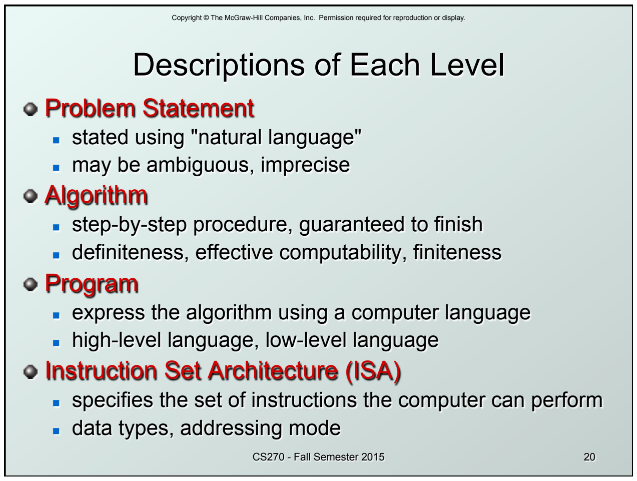
Microarch

Logic/Circuit Design: gates and low-level circuits to implement components

Circuits

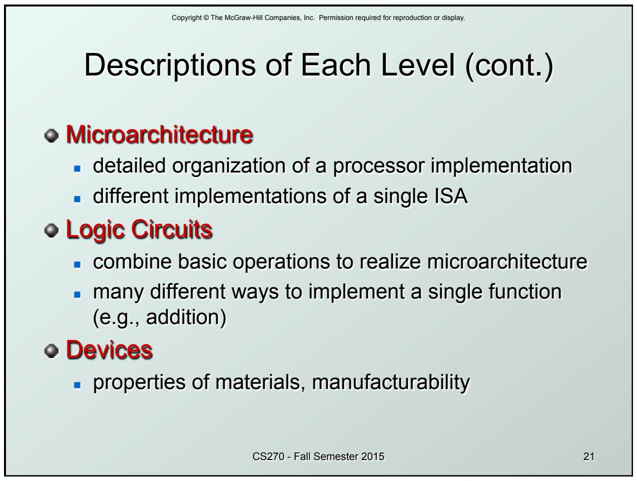
Process Engineering & Fabrication: develop and manufacture lowest-level components

Devices



Descriptions of Each Level

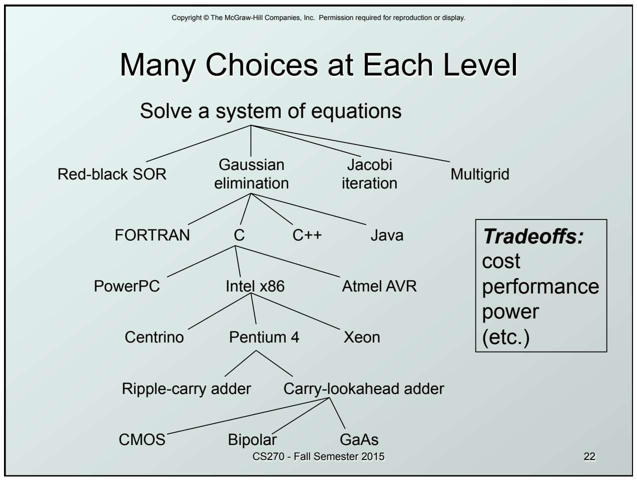
- Problem Statement
  - stated using "natural language"
  - may be ambiguous, imprecise
- Algorithm
  - step-by-step procedure, guaranteed to finish
  - definiteness, effective computability, finiteness
- Program
  - express the algorithm using a computer language
  - high-level language, low-level language
- Instruction Set Architecture (ISA)
  - specifies the set of instructions the computer can perform
  - data types, addressing mode



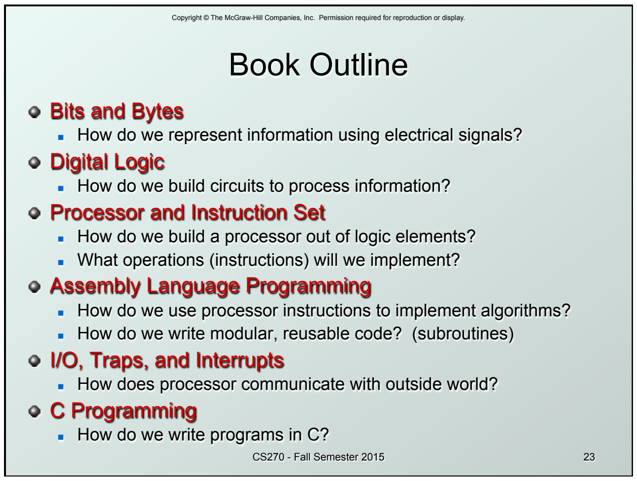
Descriptions of Each Level (cont.)

- Microarchitecture
  - detailed organization of a processor implementation
  - different implementations of a single ISA
- Logic Circuits
  - combine basic operations to realize microarchitecture
  - many different ways to implement a single function (e.g., addition)
- Devices
  - properties of materials, manufacturability

Many Choices at Each Level



Tradeoffs:
cost
performance
power
(etc.)



Book Outline

- Bits and Bytes
  - How do we represent information using electrical signals?
- Digital Logic
  - How do we build circuits to process information?
- Processor and Instruction Set
  - How do we build a processor out of logic elements?
  - What operations (instructions) will we implement?
- Assembly Language Programming
  - How do we use processor instructions to implement algorithms?
  - How do we write modular, reusable code? (subroutines)
- I/O, Traps, and Interrupts
  - How does processor communicate with outside world?
- C Programming
  - How do we write programs in C?