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 by C. Wilcox, S. Rajopadhye, M. Strout, Y. Malaiya

Colorado State University

Lecture Goals

- Review course logistics
  - Logistics
  - Introductions
  - Assignments
  - Grading Criteria and Policies
  - Organization
  - How do I do well in the class?
- Introduce key concepts
  - Role of Abstraction
  - Software versus Hardware
  - Universal Computing Devices
  - Layered Model of Computing

Logistics

- Lectures: Tue, Thu 9:30-10:45 in Glover 130
- Recitations: Tu. 1-1:50, Tu. 11-11:50, Wed. 11-11:50, Thur 11-11:50pm in CSB 225
- Labs: TBD in CSB 120
- Exams:
  - Midterm March 14, Th. during class time,
  - Final May 14 Tu. from 6:20-8:20PM
- Materials on the website and RamCT:
  - http://www.cs.colostate.edu/~cs270
  - http://ramct.colostate.edu
  - Discussions using Piazza

Instruction Team

- Instructor
  - Yashwant Malaiya, Dr. Malaiya, or Prof. Malaiya
  - Office 356 CSB, 11-11:50 Tu, Fri
- Graduate Teaching Assistants
  - Paul Ge Huang
  - Navini Dantanarayana
  - Lab hours: TBD
  - Email/Discussion board

Assignments

Assignments and quizzes are posted on RamCT:

- Weekly assignments roughly alternate between written and programming assignments (Deadlines will be on RAMCT)
- Homework assignments:
  - due in hardcopy on Thu. at the start of class,
  - or by Thu. by 11:59pm if required electronically
- Programming assignments are submitted in electronic form Thu. By 11:59pm
- Quizzes are online on RamCT, due date is usually Sun. at 11:59pm

Grading Criteria

- Grading Weights
  - Assignments: homework and programming (40%)
  - Quizzes (8%)
  - Recitations (5%)
  - Midterm Exam (20%)
  - Final Exam (25%)
  - Participation (2%)
- 50% rule
  - Must earn at least 25 of the 50 assign. points (HWs, PAs, quizzes, parti.) and 25 of the exam points (recits, midterm, final)
- Grading scale:
  - A ≥90%, B ≥80%, C ≥ 70%, D ≥60%, F <60%. Will not cut higher but reserve the prerogative to cut lower.

Policies

- Late Policy
  - During "late period" 75% credit
- Academic Integrity
  - http://www.cs.colostate.edu/~info/student-info.html
  - Do your own work
    - Do not take notes while talking with other students.
    - We will be running Moss on programming assignments.
    - We know how to use google too.
    - If in doubt, ask!
  - Maintain a professional atmosphere in the classroom, recitations, and the lab

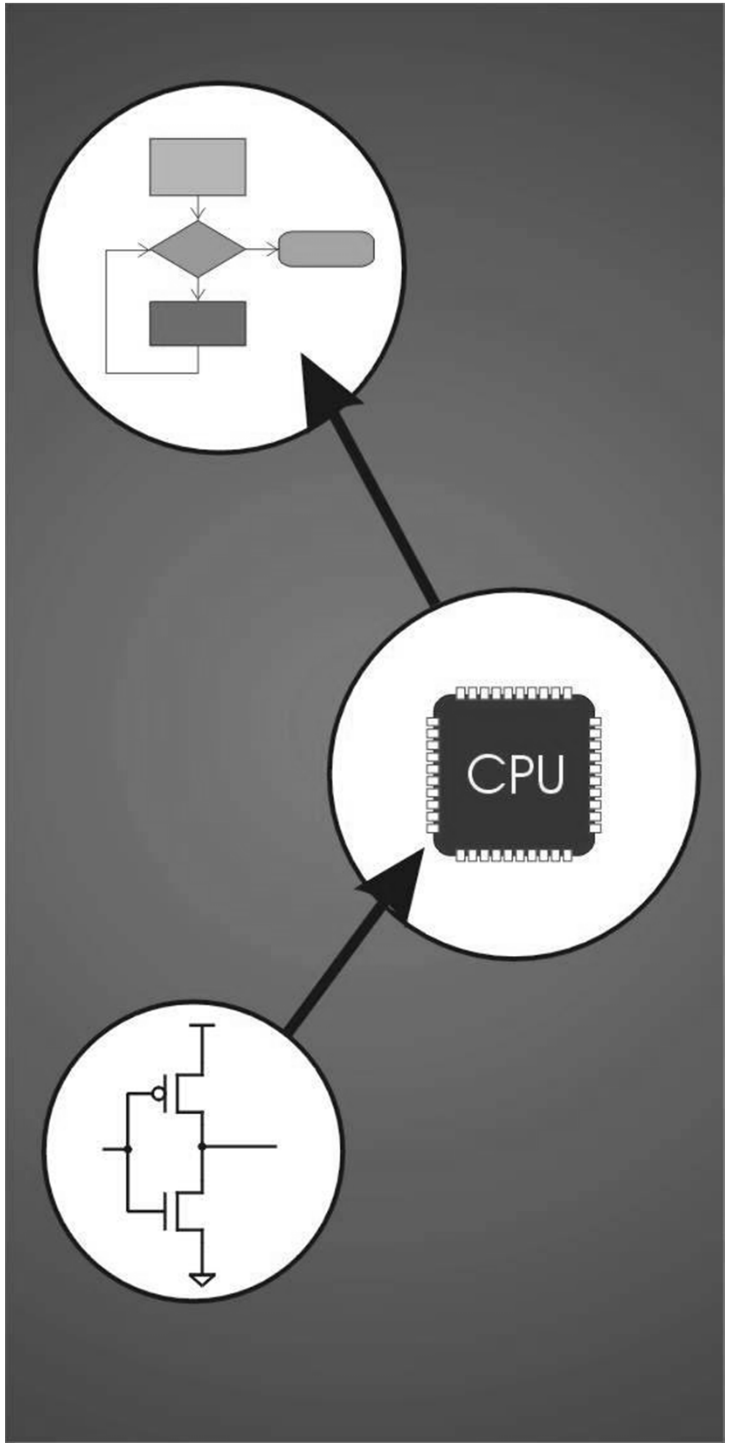
Course Organization

- 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, I/O, subroutines, memory model
- 1/3 C programming: data types, language syntax, variables and operators, control structures, functions, pointers and arrays, memory model, recursion, I/O, data structures
- Study interfaces among these

How do I get an A?

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 instructor and TAs.
4) Read the textbook, take the quizzes, work through the end of chapter problems.



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:
  - Understand computer organization: C, Assembly, Hardware and their interaction
- 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 (cont…)

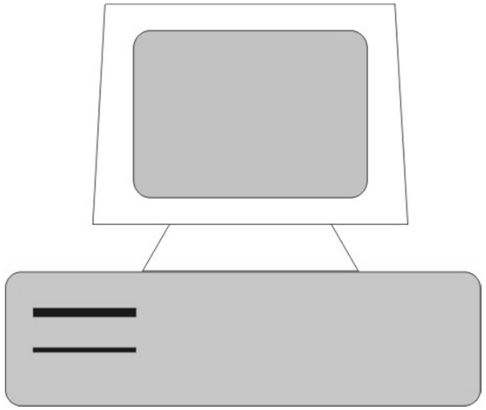
- Hardware vs. Software
  - It's not either/or – both are components of a computer system that cooperate.
  - Even if you specialize in one, you should understand capabilities and limitations of both.
  - The best programmers understand the computer systems that run their programs.
  - Computers are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

Tablet

=



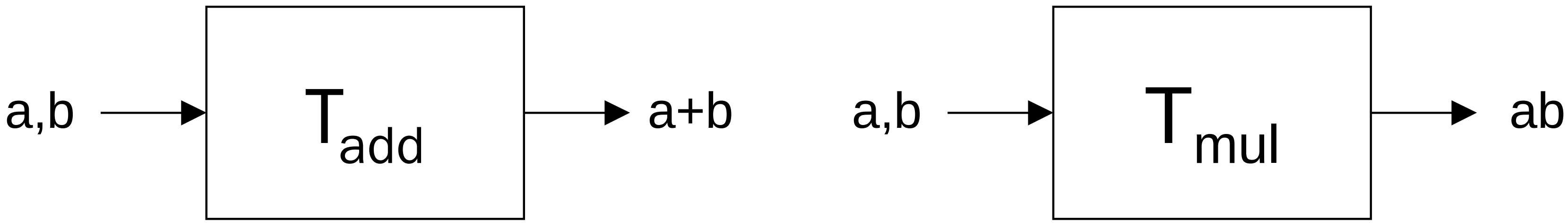
Desktop

=

Supercomputer

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)*



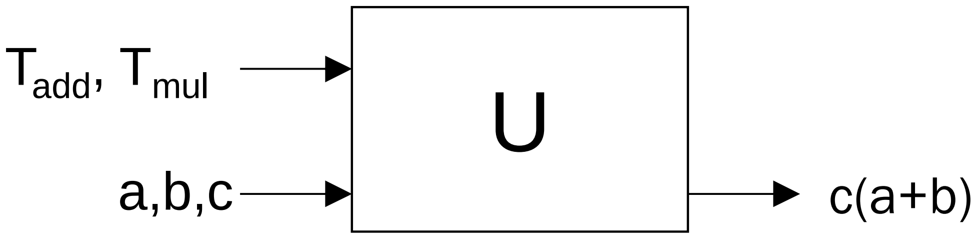
Turing machine that adds *Turing machine that multiplies*

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)



Universal Turing Machine

Universal machine is <u>programmable</u> – so is a computer!

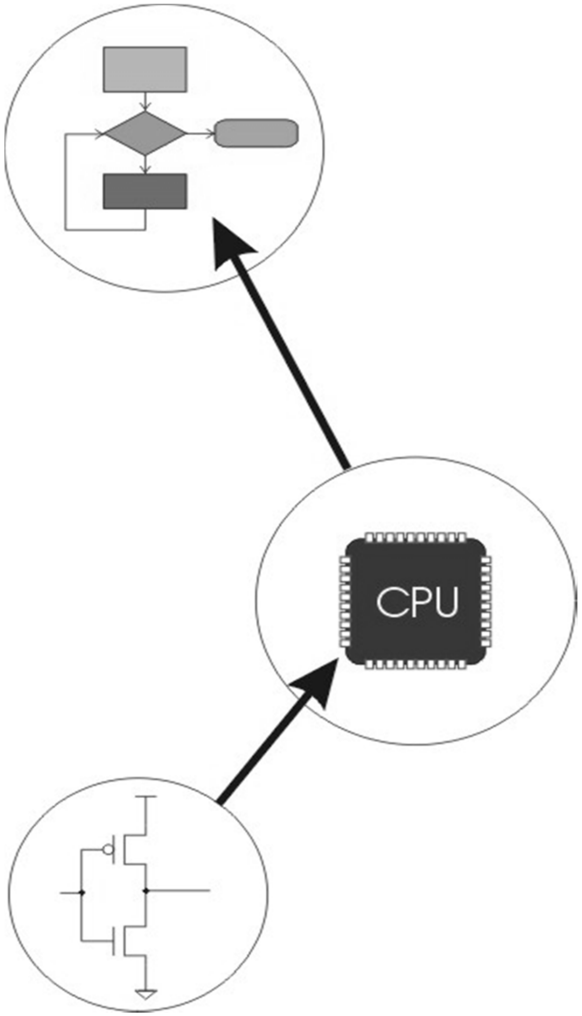
- **instructions are part of the input data**
- **a computer can emulate a Universal Turing Machine**

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 needed
    - weather forecast, next frame of animation, ...
  - Hardware cost
    - cell phone, automotive engine controller, ...
  - Power consumption
    - cell phone, handheld video game, ...

Big Idea #2: Transformations Between Layers



Problems

Algorithms

Language

Instruction Set Architecture

Microarchitecture

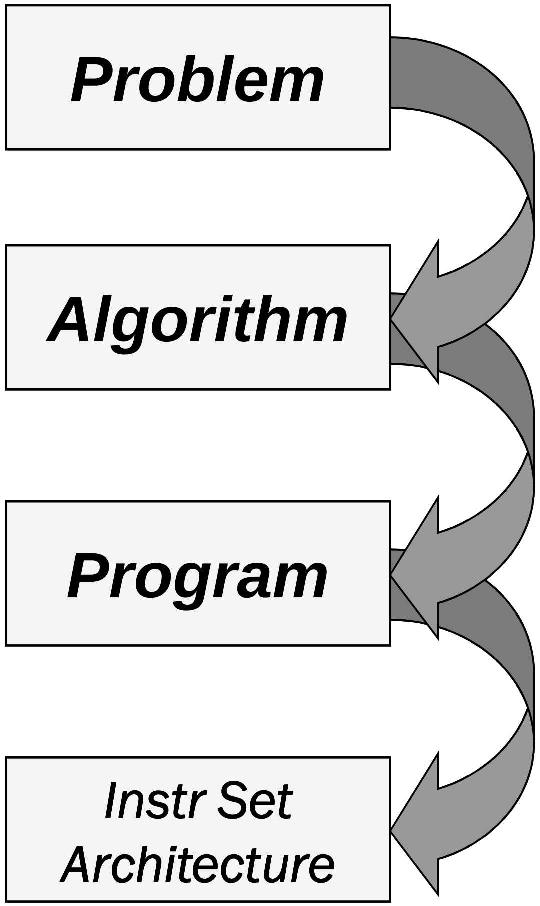
Circuits

Devices

Focus

How do we solve a problem using a computer?

- A systematic sequence of transformations between layers of abstraction.

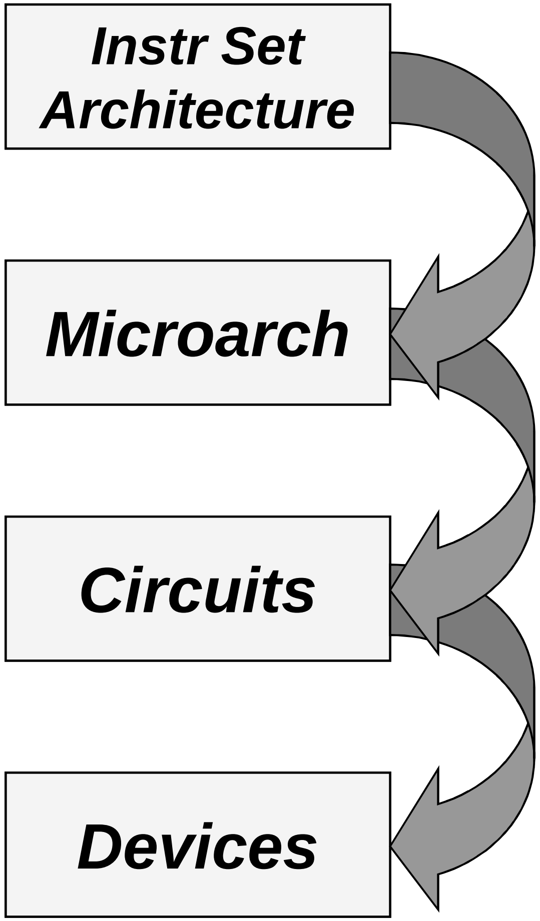


Software Design:
choose algorithms and data structures

Programming:
use language to express design

Compiling/Interpreting:
convert language to
machine instructions

Deeper and Deeper…



Processor Design:
choose structures to implement ISA

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

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

Descriptions of Each Level

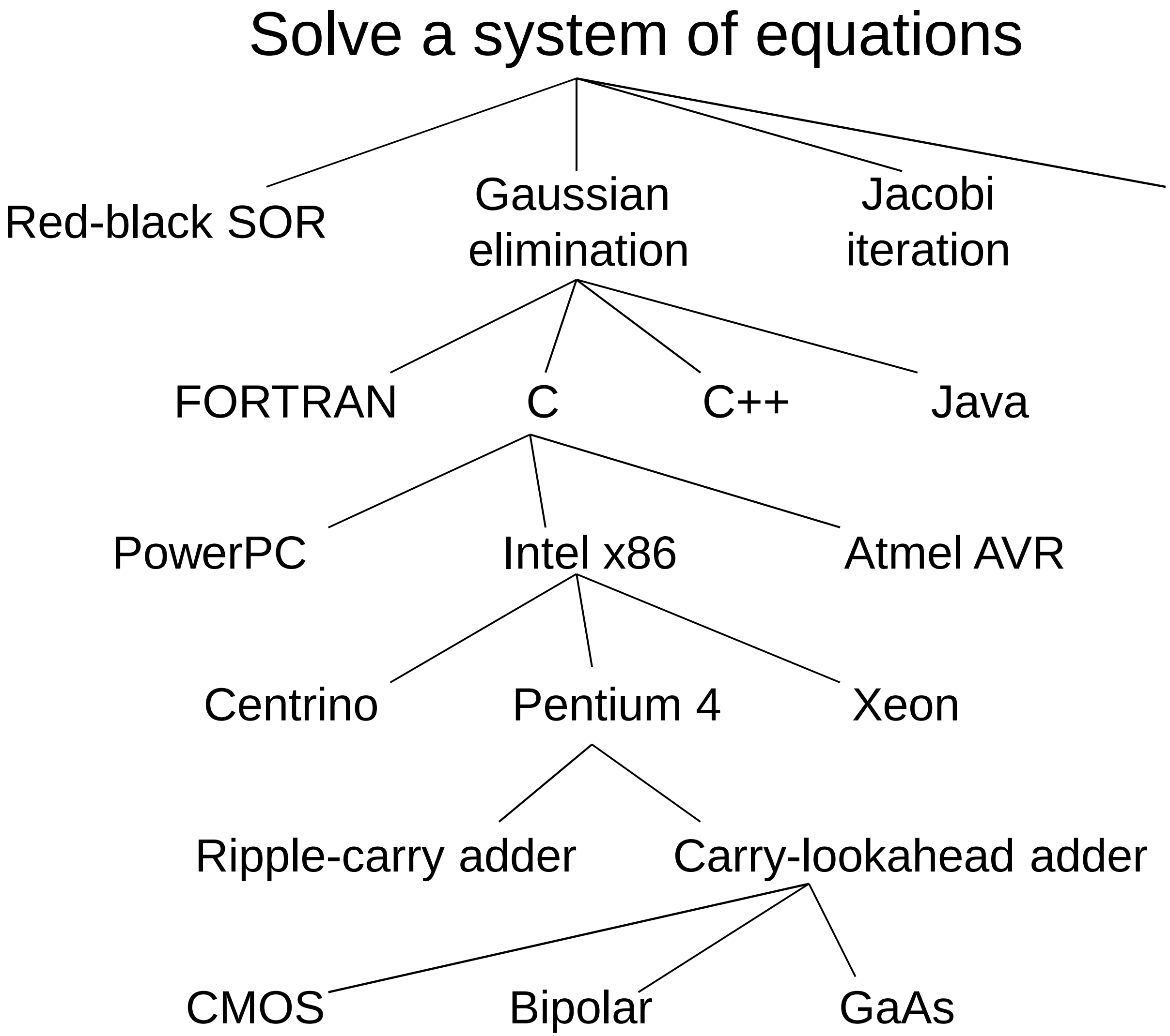
- Problem Statement
  - stated using "natural language"
  - may be ambiguous, imprecise
- Algorithm(s)
  - step-by-step procedure, guaranteed to finish
  - definiteness, effective computability, finiteness
- Program
  - express the algorithm using a computer language
  - high-level language, low-level 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 (transistors)
  - properties of materials, manufacturability

1-3 Billion in a processor chip

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?

We will also study

- C programming most of the semester
- Program development on the command line
  - compiler: gcc
  - build system: make
  - debugging: gdb,
  - revision control: subversion
- Software Engineering
  - Unit testing is important
  - Comments and documentation is important
  - Steady work habits are important