Lecture Goals
Review course logistics
   • Assignments & quizzes
   • 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 desks: See syllabus
Office hours: See syllabus
Materials on the website:
   • http://www.cs.colostate.edu/~cs270
   • Piazza: access through Canvas, or directly
Assignments & Quizzes

Assignments
- Posted on Progress page of the course website
- Programming (C, LC-3) or Logisim circuit designs
- See Canvas for due dates
- Submit via Checkin before 11:59 PM (unless otherwise specified)
- Late period for assignments – posted in assignment, 20% deduction
- Regrading requests in Piazza (see the syllabus for policies).

Quizzes:
- Can be on-line (canvas) or in-class (using iClicker)

Policies

Grading Criteria
- Assignments (20%)
- Recitations (10%)
- Quizzes and iClicker (10%)
- Two Midterm Exams (20% each)
- Final Exam (20%)

Late Policy
- None accepted

Academic Integrity
- http://www.cs.colostate.edu/~info/student-info.html
- Do your own work
- Cannot copy and paste any code, unless provided by us

People

Instructor:
- Phil Sharp

Graduate Teaching assistants:
- N/A

Undergraduate Teaching Assistants:
- Kacey Schulz

Office hours/locations
- See course website
Organization

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

1/3 Instruction set architecture: machine/assembly code, instruction formats, branching and control, LC-3 programming, subroutines, memory model (stack)

1/3 computer hardware: numbers and bits, transistors, gates, digital logic, state machines, von Neumann model, instruction sets, LC-3 architecture

Top Down Perspective

• Multilayered view:
  • Higher layers serves as the specification.
  • Lower layer implements provides the implementation

• We will see
  • How a higher level language (C) is implemented by a processor instruction-set architecture (ISA), LC-3 in our case?
  • How an ISA is implemented using digital circuits?
  • How are digital circuits implemented using transistors?
  • And so on …

Grading Criteria

<table>
<thead>
<tr>
<th>Letter Grade</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>≥90%</td>
</tr>
<tr>
<td>B</td>
<td>≥80%</td>
</tr>
<tr>
<td>C</td>
<td>≥70%</td>
</tr>
<tr>
<td>D</td>
<td>≥60%</td>
</tr>
</tbody>
</table>

• We will not cut higher than this, but we may cut lower.
• Your average score on exams must be ≥65% to receive a passing grade in this course.
How to be successful in this class

1) Read the textbook.
2) Attend all classes and recitations.
3) Take the in-class and on-line quizzes as required.
4) Do the worksheets.
5) Do all the assignments yourself,
   • ask questions (early! [but not too early!]) if you run into trouble.
6) Take advantage of lab sessions where help is available from TAs,
   • but try to do it yourself first, too much help can be harmful.

Text book:
Introduction to Computing Systems:
From Bits and Gates to C and Beyond
2nd Edition
Yale N. Patt and Sanjay J. Patel

Other resources:
• Computer Organization and Design by David Patterson, John Hennessy
• Introduction to the Theory of Computation by Michael Sipser
• C Programming Language by Dennis Ritchie, Brian Kernighan
• MIT Open Courseware 6.004
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.

Hardware vs. Software
• It’s not either/or – both are components of a computer system.
• Even if you specialize in one,
  you should understand capabilities and limitations of both.

Big Idea #1: Universal Computing Device
All computers, given enough time and memory,
are capable of computing exactly the same things.

PDA = Workstation = 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)

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

U 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
 Digital 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...

Processor Design:
choose structures to implement ISA

Instr Set Architecture

Microarch

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

DigCircuits

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 processor (CPU) 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

Solve a system of equations
Red-black SOR
Gaussian elimination
Jacobi iteration
Multigrid

FORTRAN
C
C++
Java

PowerPC
Intel x86
ARM

Centrino
Pentium 4
Core

Ripple-carry adder
Carry-lookahead adder

CMOS
Bipolar
GaAs

Tradeoffs:
cost
performance
power
(etc.)
Course Outline

Bits and Bytes
- How do we represent information using electrical signals?

C Programming
- How do we write programs in C?
- How do we implement high-level programming constructs?

Instruction set architecture/Assembly language
- What operations (instructions) will we implement?
- How do we use processor instructions to implement algorithms?
- How do we write modular, reusable code? (subroutines)
- IO, Traps, and Interrupts: How does processor communicate with outside world?

Digital Logic and processor architecture
- How do we build circuits to process and store information?
- How do we build a processor out of logic elements?

Computer systems: what is next?

Questions

High level language advantages?

Low level language advantages?

Difference between ISA and Microarchitecture?