Introduction

Lecture Topics
- Introduction to the course
- Abstraction
- Computer systems organized as a systemic set of transformations
- Computability
- Representation using bits

Reading assignments
Textbook chapter 1, § 2.1, 2.2

Homework
No

Introduction to the course

Course title
- Introduction to Computing Systems

Instructors
- Sayan Mitra (12:30 – 13:50)
- David Nicol (11:00 – 12:20)
- Volodymyr Kindratenko (14:00 – 15:20)

Teaching Assistants
- TAs
  - Gregory Ross – head TA
  - Keith Campbell
  - Joshua Cochrane
  - Samuel Cornwell
  - Jeremy Greenberg
- Lab assistants
  - Stanley Idesis
  - Cori Johnson
  - Sahil Handa
  - Tomasz Kalbarczyk
  - Genevieve LaBelle
Course objectives
• To introduce basic concepts behind computer organization and computer programming
• To teach how to program in C
• To prepare to take advanced courses in ECE and CS

Course philosophy
• One of a kind course
• Learning to program a computer requires understanding its basic operating principles
• Bottom-up approach to introduce computing concepts
  o Complex systems are built from a large number of simple parts
  o We will learn about the building blocks and how they are put together to form a computer
  o We will also learn how a complex program is transformed from a high-level language into a series of simple instructions and how these instructions are executed at the hardware level
• This is not a typical way to teach programming, but this is the right way to teach programming for computer engineering students

Course structure
• 3 parts
  o binary data representation, binary arithmetic, digital logic
  o computer system organization using LC-3 as an example
  o introduction to C programming
• 29 lectures
  o Tuesday & Thursday, 3 lecture sections
• 14 discussion sections on Fridays
  o Organized as programming studios
  o The same time, but different locations from those advertised:
    ▪ DCL440, DCL 520, and Grainger 57 – check course website for details!
    ▪ Will expose you to many tools that otherwise you will not have a chance to learn and use
• 3 homework assignments
  o Due on Wednesdays at 5pm
  o Drop-off box
• 5 programming assignments with intermediate checkpoints
  o Due on Wednesdays at 6pm
    ▪ Late policy: Projects can be turned up to 48 hours late with a late penalty of 15 points (out of 100 total) per 24 hour period, prorated on hourly basis
  o Electronic submission
• 2 mid-term exams
o  Wednesday evenings
  •  1 final exam

Textbook/Course notes

Website, video recordings, notes

Website
  •  http://courses.engr.illinois.edu/ece190/
  •  check regularly for up to date info
  •  homework and machine problems will be posted on-line
  •  Office hours of instructors and TAs will be posted on-line soon
  •  Online discussion board

Grading
  •  Homework: 5%
  •  Projects: 25%
  •  Exams: Midterm 1: 20%  Midterm 2: 20%  Final: 25%
  •  Other: 5% (subjective evaluation)

Getting help
  •  14 discussion sections on Fridays, four different timeslots
  •  Office hours, will be posted on-line soon

Policies
  •  Will be posted on-line on the course’s website
    o  Make sure you know and follow the course policies!
  •  Collaboration is encouraged, but do not cross the line between collaboration and cheating
  •  Cheating – see policy on-line
    o  if caught first time – a warning
    o  if caught second time – you will fail the course
  •  Cell phones – no cell phones in the class!

Expectations
We expect students to attend all lectures and studios

We expect students to complete all assignments – grades are based on these

We expect students to work hard and do their best
Computer and computer system

What do we mean by computer?

- In general, anything that can compute
- In a more strict sense, it is a mechanism that performs two things:
  - Directs the processing of information
    - Figure out what operation on the information should be done next
  - Performs the processing of information
    - Perform actual calculation
- Computer performs these operations in response to a computer program
- More precise name for this mechanism is CPU or processor
  - In the past processor was constructed out of a large number of integrated circuit packages, each made of few tens of transistors
  - Today, it is just a single chip, made of hundreds of millions of transistors
- In a more loose sense, it is a collection of parts, in addition to the processor, that in combination form a computer system
  - Input devices: keyboard, mouse
  - Output devices: display, audio, printer
  - Storage: disk, CDROM
- Computers take many forms
  - PC, cell phone, MP3 player, GPS, ...

- Why study them?
  - Users, designers, programmers, ...
3 Concepts
1. Abstraction
2. Levels of transformation
3. Computability

Abstraction
A mental shortcut for hiding understood details.

- Everyday example
  - The blue car with orange stripes was parked on the street.

- Mathematics example
  - Definition of a definite integral as an area under a function $f(x)$ on an interval $[a,b]$

$$\lim_{n \to \infty} \sum_{i}^{n} f\left(a + \frac{b - a}{n} i \right) \frac{b - a}{n} = \int_{a}^{b} f(x) \, dx$$

- Black box notation

- We use abstraction as a productivity enhancer
Computer systems organized as a systemic set of transformations

Concept of transformations that bring the problem described in a human language to the transistors that actually execute it.

- Problem statement
  - Typically expressed in a natural, or human language
  - Human languages suffer from ambiguity issue
  - Example: given positive real x, find positive y such that y*y = x

- Algorithm
  - Problem statement is transformed into an algorithm
  - An algorithm is a step-by-step procedure that is guaranteed to terminate, such that each step is precisely defined and can be carried out by a computer
  - Basic algorithm properties are
    - **Definiteness**: each step is precisely stated, or defined
    - **Effective computability**: each step can be carried out by a computer
    - **Finiteness**: the procedure eventually terminates
  - Any problem can be usually solved using a number of different algorithms.
  - Square root example: find a with a*a < x, b with b*b > x, then refine search within [a,b] for solution (sketch on board)

- Program
  - Algorithm is transformed into a computer program using one of computer languages.
  - Programming languages are “mechanical languages,” they were specifically invented to instruct computers in a simple straightforward way
  - There are over 1,000 programming languages, invented for specific computers, problem domains, e.g.:
    - Fortran for scientific calculations
    - COBOL for solving business data-processing problems
    - Prolog for expert systems
  - We will learn C, which was initially designed to manipulate low-level hardware structures
  - 2 kinds of languages
    - **High-level languages** are hardware-independent languages with a high degree of abstraction
    - **Low-level languages** are tied to a specific computer. In general, there is one such language for each computer, called *assembly language*.
  - Square root example---what would we want to express the algorithm? Comparisons, computation (+, -, *, /), branching...
• Instruction Set Architecture (ISA)
  o Program written in a programming language needs to be translated into an instruction set of the computer on which this program will execute
  o An instruction set is a list of all the instructions that a processor can execute
  o The ISA specifies
    ▪ set of instructions
    ▪ data types for operands on which the instructions are executed
    ▪ addressing model for figuring out where the operands are located
    ▪ memory architecture, I/O interface, etc.
  o Examples of ISAs
    ▪ x86
      • has more than 100 operations defined
      • more than a dozen data types
      • more than two dozen addressing models
  o The program written in a programming language, such as C, is translated into the specific ISA is usually done using a translating program called compiler
  o The program written in a low-level computer specific assembly language is translated into ISA using a translating program called assembler.
  o Square root example: what would we want in ISA for this problem?
• Microarchitecture (microprocessor architecture)
  o Microarchitecture is the way a given ISA is implemented on a processor. A given ISA may be implemented with different microarchitectures depending on the design goals.
  o Example x86 ISA:
    ▪ The same instruction set has been implemented by different vendors, Intel and AMD
    ▪ It has been implemented in a number of processors starting from 8086 in 1979 and finishing by today’s Nehalem microarchitecture
  o Car analogy
• Logic circuit (gates)
  o All elements of the microarchitecture are implemented out of simple logic circuits
• Devices
  o Each basic logic element is implemented using a particular device technology, e.g., Complementary metal–oxide–semiconductor (CMOS) technology
  o Of a particular interest are MOSFETs – metal-oxide field effect transistors
  o Devices (transistors) manipulate electrons which do the work for us.

• In this course, we will study all these transformations
  o starting from how transistors are put together to form basic logic elements
  o how a particular ISA is assembled from such logic elements
  o and how a program written in C is compiled into this ISA
Computability

Computers as universal computational devices

Computers are universal computational devices.

- Given an existing computer, one can run on it different instruction sequences, or programs, to execute new kinds of computations
- Computer scientists believe that anything that can be computed, can be computed by a computer
  - Provided that enough time and memory are given
- All computers are capable of computing exactly the same things
  - If they are given enough memory and time
  - Anything a fast computer can do, a slow computer can do as well, provided it is given enough time

Turing machine

The idea of a universal computational device was proposed by Alan Turing in 1936

- Turing reasoned that the most basic features of a mechanical computation are
  - The ability to read and write on a storage medium
    - A tape divided into cells, each cell storing a symbol from a tape alphabet
  - To make some simple logical decisions
• In other words, Turing proposed a model to characterize mechanical computation as a sequence of instructions such as
  o Write a symbol on the tape cell
  o Move to the next cell on the tape
  o Observe the symbol scanned from the tape cell and chose the next step according to the symbol scanned
  o Stop
• Turing’s thesis: every computation can be performed by some Turing machine.

• Additional material
• A Turing machine consists of the following parts:
  o Tape: divided into cells, each cell contains a symbol from some finite alphabet. The alphabet contains a special blank symbol, “B”, and other symbols. The tape is extendable to the left and to the right as much as needed.
  o Head: can read and write symbols on the tape and move the tape left and right one (and only one) cell at a time. At any time the head is positioned over a particular cell, which is said to scan. At time 0 the head is positioned over the start cell.
  o State register: stores the state Turing machine is currently in
  o Action table: a table of instructions such that given the state the machine is currently in, Qi, and a symbol it is scanning, Sj, the head erases or write a symbol Sj or moves the head left or right, and then assumes the same or a new state, Dk. We will write this action as Q, A, → Q, A, D

<table>
<thead>
<tr>
<th>State</th>
<th>Input symbol</th>
<th>New Symbol</th>
<th>New State</th>
<th>Head move</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>R</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>R</td>
</tr>
<tr>
<td>0</td>
<td>B</td>
<td>B</td>
<td>1</td>
<td>S</td>
</tr>
</tbody>
</table>

Example action table for a Turing machine that computes the complement of a given number. Alphabet symbols = \{0, 1, B\}, states = \{0, 1-end state\}

• “Black box” models of Turing machine
  o An abstraction that shows the action of the Turing machine, but does not specify how the action is performed
    • The data on which the machine operates is the input data
    • The results that the machine produces is the output data
  o Add example

```
| a, b | T_{ADD} | a+b |
```


Consider the steps to do binary addition on a Turing machine

- What’s the algorithm
- State captures sequencing, symbols 0,1, b
- How do manage tape input / output?

**Universal Turing machine**

- The problem with Turing machines is that a different one must be constructed for every new computation to be performed, for every input-output relation.
- Turing was able to demonstrate that there is a table U which is such that if a Turing machine is programmed in accordance with U, and if any table, P, is translated and written out on the machine's tape, then the machine will behave as if it is programmed in accordance with P.
- Such a Turing machine with table U is called Universal Turing machine (UTM)
- When started on a tape containing the encoding of another Turing machine, P, followed by the input to P, a UTM produces the same result as P would produce when started on that input.
- Essentially a UTM can simulate the behavior of any Turing machine (including itself.)
- Using the “black box” notation:
• One way to think of a UTM is as a programmable computer. When a UTM is given a program (a description of another machine), it makes itself behave as if it were that machine while processing the input.

• What can be computed with Turing machines
  - Numbers: A number is **Turing-computable** if there exists a Turing machine which starting from a blank tape computes an arbitrarily precise approximation to that number. All of the algebraic numbers (roots of polynomials with algebraic coefficients) and many transcendental mathematical constants are Turing-computable.
  - Functions: Numeric functions, such as the machine for addition multiplication, proper subtraction, exponentiation, factorial and so on.

• What cannot be computed
  - A crucial observation about Turing machine is that there are only countably many machines (a set is countable if it is finite, or may be placed in one-to-one correspondence with the integers.) It follows immediately that there are uncountably many real numbers that are not computable, since the reals are not countable. There are simply not enough Turing machines to compute all of those numbers.
  - Like the real numbers, the number of functions on the natural numbers is not countable. It follows therefore that there are uncountably many such functions that are not computable by any Turing machine.

**Representation using bits**
Real world information can be represented using fundamental data types (integers, reals, characters). These fundamental data types are implemented using **binary digits** (bits) in the digital world.

**Basic concepts**
• Recall that each basic logic element is implemented using a particular device technology
• Digital devices operate by reacting to the presence or absence of voltages, not to the actual voltage
  - This is symbolically represented as “0” – no voltage present, or “1” – voltage present
  - “0” and “1” are referred to as **bits**, or **binary digits**
• Switching from state 1 to state 0 does not necessarily mean that the voltage drops all the way to 0 volt, it simple means that voltage drops below a certain threshold value
• Example, transistor-transistor logic (TTL) based digital circuits operate between 0 and 5 V
  - 0 is any voltage below 0.8 V
  - 1 is any voltage above 2 V
  - 0.8-2 V region is **ambiguous**, a good design makes the transition from one state to the other as short as possible
  - The speed with which transistors can switch from one state to another is related to the speed with which the digital device can operate
• We will use binary digits 0 and 1 to represent numbers, just like in the decimal digits system
• We can combine multiple bits to represent arbitrary large numbers
  o This equals to using multiple wires and multiple digital devices to transmit and react to the presence or absence of voltages on these wires
• Having $k$ bits, each of which can be either 0 or 1, we can represent exactly $2^k$ unique numbers, for example
  o with 2 bits we can have the following permutations: 00, 01, 10, 11, or 4 unique numbers
  o with 8 bits, we can have $2^8$ permutations, thus representing as many as 256 different values
• If $n$ bits can represent $2^n$ values, how many bits are needed to represent $m$ things?
  o $2^n = m; \quad \log_2 2^n = \log_2 m; \quad n = \log_2 m$
  o Example: $m=42; \log_2 42=5.39; \quad n=6$
    ◦ Note however that $2^6 = 64$
Binary representation of integer numbers

- Let’s look how integers, such as 1, 2, 3, and so on can be represented using binary digits 0 and 1
- Decimal numbers use base 10 positional notation, e.g., 564 means that there are 5 hundreds, 6 tenths, and 4 ones. Each digit’s position in the number defines if this number is ones, or tenths, or hundreds...
  - 564 can be written as $5 \times 10^2 + 6 \times 10^1 + 4 \times 10^0$
  - Here 10 is the base of the numerical system used
  - Alternatively, 564 can be written as $564_{10}$
- Binary numbers work the same way using just two digits, 0 and 1, using base 2 positional notation
  - 00110 can be written as $0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
  - Notation $00110_2$ means that the number is written in binary form
  - By the way, $00110_2 = 6_{10}$
  - With k bits we can represent $2^k$ positive integers, ranging from 0 to $2^k-1$

- Finite capacity of binary representation
- Q1: with 6 digits, what is my range of representation?
  - Why would this question even make sense?
    - Odometer example
- A1: 0 to 999999
- Q2: with 6 bits, what is my range of representation?
- A1: 0 to $2^6-1$

- Notable concepts
  - Data width
  - MSB
  - LSB
  - Word size