Introduction

CSE 1302

Introduction to Computer Engineering

Instructional Staff

- Instructor – James Orr
  - Office: Urbauer 228
  - Email: james.orr@wustl.edu
- Head TAs – Ayla Burba and Chiagozie Okoye
  - Contact via Piazza
- Webpage: cse132.engineering.wustl.edu
- Office hours: TBD (see web page)
- Appointments for James: contact via Piazza

Course Web Page

- cse132.engineering.wustl.edu
- Calendar, links to studios and assignments
- Documents grading, collaboration, and late policies
- Contains documentation on languages (Java, C) and tools (Eclipse, Git, Arduino IDE)

What is this class about?

- Organization will be like CSE 13(0)1
  - 1.5 hrs/wk lecture (on Wednesdays)
  - 1.5 hrs/wk studio (on Mondays)
- The material includes
  - Basic computer capabilities (I/O, esp. custom I/O)
  - Demystifying how computer systems operate
  - More than one machine, more than one type of machine
  - Design decisions that include both software and hardware

Some High-level Goals for CSE 1302

- Introduce CoE concepts (so those who should be CoE students know what that is)
  - Do this while ensuring relevance to CS students
- Introduce the concept that not all computers are desktop/laptop class machines
  - Computing happens in many different form factors
  - Vehicle for 1302 will be an 8-bit microcontroller + standard desktop environment (Java/Eclipse from CSE 1301)
- Introduce distributed concurrency (more than one thing going on at a time)
- Recurring theme throughout semester will be the representation of information

Typical Module Sequence

- Lecture
  - Here in Brauer 12
- Studio
  - In Urbauer labs (attendance is required!)
- Assignment
  - Demos in office hours or beginning of studio
- Help
  - A number of help sessions will get scheduled and be staffed by TAs
  - Piazza (all the TAs have instructor access)

Demos in Lab

- Lab time on Monday is primarily for studio
  - Therefore, we reserve most of that time
- Demos
  - Must be in the first 15 min. of lab time
  - You must be ready to demo when you walk in
  - Help will be available ahead of class, but not in lab
  - No “re-demo” options
    - Have your assignment ready to demo when you request a TA to check it out

Two Compute Platforms

- Java on laptop or lab machines, using Eclipse as the development environment (just like 1301)
- ``C'' on Arduino machine
  - Actually a subset of C, and subset is *very* close to the Java you are familiar with
  - Physical computer is 8-bit machine running at only 16 MHz (over 100 times slower than desktop PC)
    - 16 Kbytes of program memory
    - 2 Kbytes of data memory
    - No keyboard or display
  - Wonderful community of users, doing lots and lots!

Arduino Programs

- Community calls them “sketches”
- Composed of the basic structure below

```
void setup() {
    // insert startup code here, will execute once
}

void loop() {
    // insert main code here, will execute over and over
}
```

Hello World

- First complete Arduino program

```
void setup() {
    Serial.begin(9600); //startup comm. link to PC
    Serial.println("Hello world!");
}
void loop() {
}
```

Arduino Timing

- Use delay() library routine
  - Argument is integer number of milliseconds
- Use millis() library routine
  - Returns the number of milliseconds since last reset of Arduino
  - Return type is ` unsigned long int`, which is 32 bits or 4 bytes
- Later in semester we will use micros()
  - Returns number of microseconds since last reset

Arduino Printing

- Printing goes to Serial Monitor in Arduino IDE
  - Serial.begin(9600) in setup() initializes port and sets baud rate (communication speed)
- How do we print?
  - Use Serial.print() and Serial.println()
  - Argument can be any type
    - Serial.println("String to print");
    - Serial.print(14); // no newline included
    - NOTE: cannot do this – Serial.println("X = " + x); because string concatenation is not supported
    - Do this instead –

```
Serial.print("X = ");
Serial.print(x);
```

Quiz Time

- Go to Canvas and answer the single question for Quiz 1A

- True or False:

Work cannot be "redemoed" and you can't stop a demo session once is has begun. For example, if you find an error in your work during the demo, you will not be allowed to fix the error and have your grade based on the "fixed" work.

Let’s Get Started

- Computational Abstraction
  - Finite-state machines
- Information Representation
  - In the digital world, this means binary

Finite-State Machine

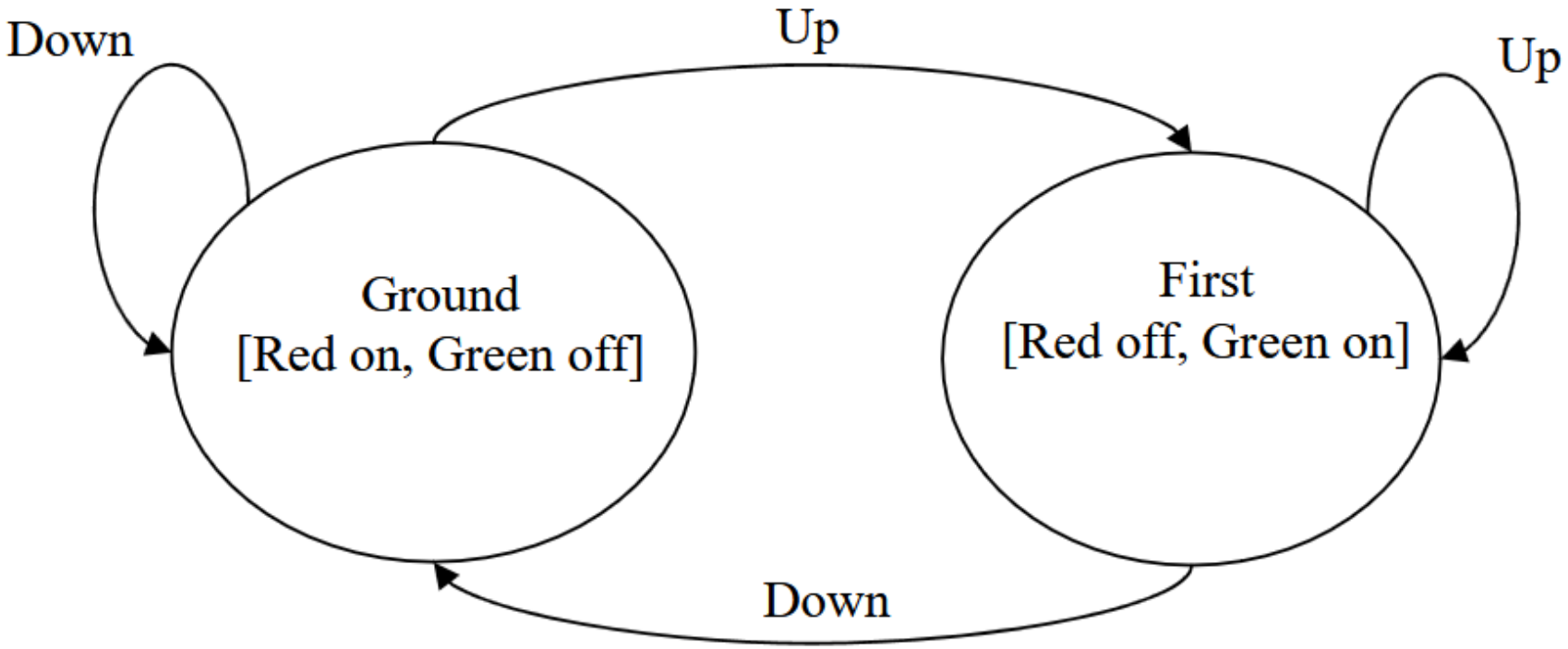
- Abstract machine
  - A specification of what is supposed to happen
- Finite number of “states” (hence the name)
  - A state *remembers* (i.e., keeps track of) whatever the designer wants the system to remember

Example Finite-State Machine

- Elevator control
  - Two inputs: UP button and DOWN button
  - Two states: Ground floor and First floor
  - Two outputs: Two lights in elevator, red indicating ground floor and green indicating first floor
  - Transition trigger: button press
- FSM bubble diagram
  - One bubble for each state
  - Edges (transitions) labeled with inputs
  - Outputs labeled either on state bubbles or edges

FSM Bubble Diagram

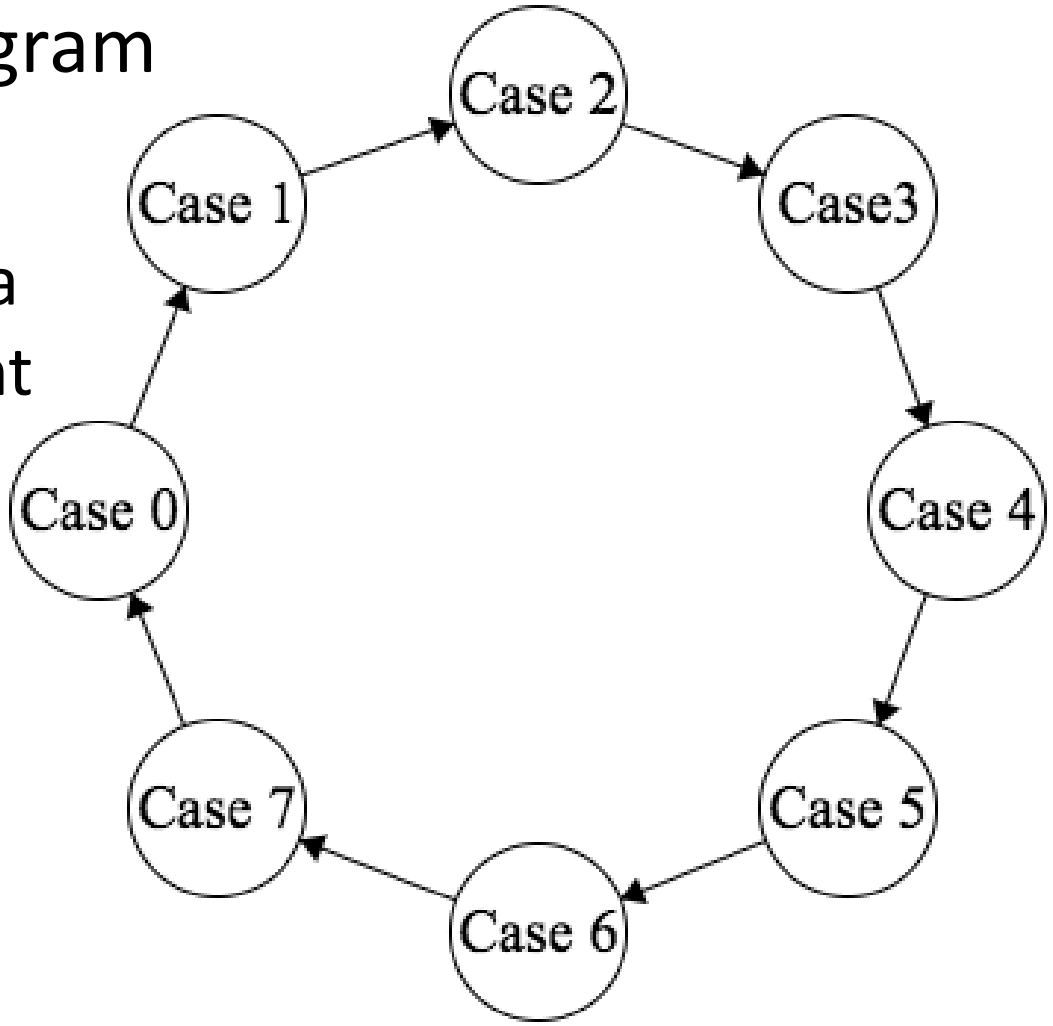
- Elevator control FSM diagram



https://www.cs.princeton.edu/courses/archive/spr06/cos116/FSM_Tutorial.pdf

Counter FSM Bubble Diagram

- Studio 1 FSM diagram
- 8 states
  - Each represents a value of the count
- No inputs
- Output is state
- Transition trigger
  - Elapsed time



Implementing FSMs

- Bubble diagram is just specification
  - Abstract
  - Independent of implementation
- Possible implementations
  - Directly in hardware (take CSE/ESE 260M to learn)
  - Via software (this is what we will do)
- Important considerations
  - How to represent state (we'll use a single variable)
  - What triggers state transitions (many options)

Summarizing FSMs

- Useful abstraction of computation
  - Says what needs to be done, not how to do it
  - Independent of implementation
  - Easier to reason about
  - Diagrams easier to read/edit/understand than implementation (no matter how you implement)
- We'll use FSMs many times

Let’s continue

- Computational Abstraction
  - Finite-state machines

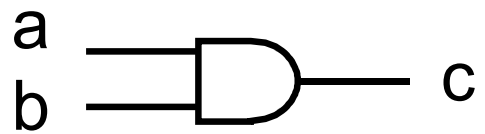
- Information Representation
  - In the digital world, this means binary

What is Binary?

- Underlying base signals are two-valued:
  - 1 or 0
  - true or false (T or F)
  - high or low (H or L)
- One “bit” is the smallest unambiguous unit of information
- Propositional calculus helps us manipulate (operate on) these base signals

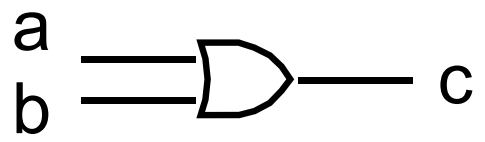
Operations in Propositional Calculus

AND $a \cdot b = c$



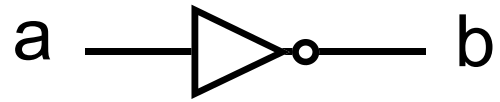
c is true if a is true and b is true

OR $a + b = c$



c is true if a is true or b is true

NOT $a' = b$



b is true if a is false

An Example

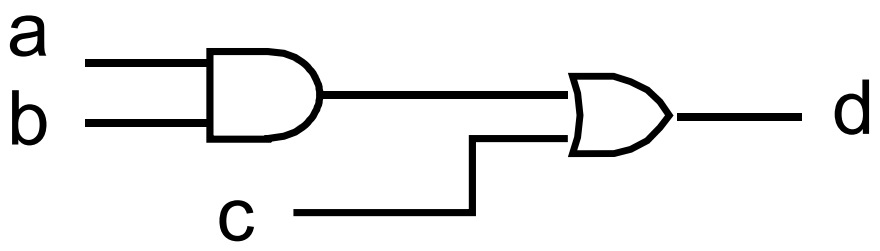
a passed microeconomics course

b passed macroeconomics course

c passed economics survey course

d met economics requirement

$d = a \cdot b + c$



Boolean Algebra

- Boolean algebra (named after 19th century mathematician George Boole) lets us manipulate and reason about expressions of propositional calculus
- Systems based on this algebraic theory are called “digital logic systems”
- All modern computer systems fall in this category

Physical Representation

- Positive logic convention
  - Binary value (1 or 0) is represented by the voltage on a wire (H or L)
  - true, 1 voltage greater than threshold V_H
  - false, 0 voltage less than threshold V_L
  - Voltage gap between V_H and V_L provides safety margin to limit errors

That’s Not Enough!

- We are interested in representing signals that have more than just two values
  - numbers
  - text
  - images
  - audio
  - video
  - and much more

How do we represent numbers?

- A positional number system lets us represent integers. E.g., in base 10:

$$xyz_{10} = x \cdot 10^2 + y \cdot 10^1 + z \cdot 10^0$$

$$= x \cdot 100 + y \cdot 10 + z$$

x, y, z can each have 10 possible values: 0 to 9

Base 2 (binary) works the same way

$$xyz_2 = x \cdot 2^2 + y \cdot 2^1 + z \cdot 2^0$$
$$= x \cdot 4 + y \cdot 2 + z$$

x, y, z can each have 2 possible values: 0 or 1

each digit is called a “bit”

e.g.,

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Negative numbers

- With a fixed number of bits, one can represent negative numbers in a variety of ways.

  E.g., 4-bit binary number system:

- **unsigned** range 0 to 15 (0000 to 1111)

  unsigned integers with n bits range 0 to $2^n - 1$

- **offset** or **bias** (e.g., -7) range -7 to 8

  subtract fixed amount (such as midpoint value)

  generally bad for computation

4-bit Sign-Magnitude

1^{st} bit encodes sign (0 = positive, 1 = negative)

bits 2, 3, 4 magnitude $\Rightarrow$ range 0 to 7 (000 to 111)

overall range -7 to +7

what about 1000? -0!

with n bits, use n-1 bits for magnitude

range $-(2^{n-1} - 1)$ to $+(2^{n-1} - 1)$

issues:

- two representations for “0”, +0 and -0
- need significant hardware to support add, subtract

2's (radix) complement

- Use negative weight for 1st bit:

$$wxyz = w \cdot -(2)^3 + x \cdot 2^2 + y \cdot 2^1 + z \cdot 2^0$$
$$= w \cdot -(8) + x \cdot 4 + y \cdot 2 + z$$

- overall range -8 to +7
- 1st bit is still sign bit,
  with 0 = positive and 1 = negative
- only one zero: 0000

Properties of 2's complement

- least significant n-1 bits have unaltered meaning (i.e., standard positional notation and weights apply)
- most significant bit has weight negated (instead of weight 2^{n-1}, it is weight -2^{n-1})
- range $-(2^{n-1})$ to $+(2^{n-1}-1)$
- negation operation: flip all bits, add 1, throw away carry
- addition/subtraction function normally

Make binary more human friendly

- Hexadecimal representation – base 16
- Commonly called “hex” but don’t be confused, it is not base 6, it is base 16
- Character set 0-9, a-f (alternately A-F)
  - a=10, b=11, c=12, d=13, e=14, and f=15
- C notation is to prefix hex with symbol `0x` (e.g., `0x12`, `0xa3`)

Positional notation applies

$$xyz_{16} = x \cdot 16^2 + y \cdot 16^1 + z \cdot 16^0$$
$$= x \cdot 256 + y \cdot 16 + z$$

So $02c_{16} = 0 \cdot (256) + 2 \cdot (16) + 12 = 44_{10}$

or 0x02c, which is the shorthand I will typically use in class

Benefits of Hex

- Real beauty of hex notation is ease with which one can move back and forth between hex and binary, since $16 = 2^4$

- To transform hex number (e.g., 0x3d50) to binary we expand each hex digit to 4 bits of binary:

3	d	5	0
0011	1101	0101	0000

Binary to Hex Transformation

- To transform binary number (e.g., 1001000) to hex we group into 4-bit groups (starting from right) and rewrite each group in hex

  100 1000

  4 8 = 0x48

- Or, e.g., 110101110

  1 1010 1110

  1 a e = 0x1ae

What about fractions?

- Later…

Logistics

- Assignment 1 released
  - Due on Studio 2 day (demo in office hours or first 15 minutes of studio)
  - Quiz 1B same due dates as Assignment 1
- Module 2 starts with lecture next week
- Studio 2 on January 26