Introduction

CSE 132

Instructional Staff

• Instructor – Roger Chamberlain
  – Office: Bryan 405C
  – Email: roger@wustl.edu
• Head TA – Josh Gelbard
  – Email: gelbard@wustl.edu
• Office hours: TBD (see web page)
• Appointments for Roger: send an email

Course Web Page

• http://classes.cec.wustl.edu/~cse132
• It is a work in progress ...
  – Will contain calendar
  – Will contain studio and lab assignments
• Documents grading, collaboration, and late policies
• Contains documentation on languages (Java, C) and tools (Eclipse, Subversion, Arduino)

What is this class about?

• Organization will be like CSE 131
  – 1.5 hrs/wk lecture
  – 1.5 hrs/wk studio
  – 1.5 hrs/wk lab
• 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 132

• 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 132 will be an 8-bit microcontroller + standard desktop environment (Java/Eclipse from CSE 131)
• 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 Simon
• Studio
  – In Urbauer labs (attendance is required!)
• Lab
  – Lab demos in Urbauer labs
• Help
  – A number of help sessions will get scheduled and be staffed by TAs
  – Piazza (all the TAs have instructor access)
Two Compute Platforms
- Java on laptop or lab machines, using Eclipse as the development environment (just like 131)
- “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!

Let’s get started
- Information Representation
- In the digital world, this means binary

What is Binary?
- Underlying base signals are two-valued:
  - 0 or 1
  - 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:
  \[ \text{xyz}_{10} = x \cdot 10^2 + y \cdot 10^1 + z \cdot 10^0 \]
  \[ = x \cdot 100 + y \cdot 10 + z \]
  \( x, y, z \) can each have 10 possible values: 0 to 9

Base 2 (binary) works the same way

\[ \text{xyz}_2 = x \cdot 2^2 + y \cdot 2^1 + z \cdot 2^0 \]
\[ = x \cdot 4 + y \cdot 2 + z \]
\( x, y, z \) can each have 2 possible values: 0 or 1

e.g.,

\[
\begin{align*}
000 & : 0 \\
001 & : 1 \\
010 & : 2 \\
011 & : 3 \\
100 & : 4 \\
101 & : 5 \\
110 & : 6 \\
111 & : 7
\end{align*}
\]

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

1st bit encodes sign (0 = positive, 1 = negative)
bits 2, 3, 4 magnitude \( \Rightarrow \) range 0 to 7 (0000 to 1111)

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 \(2^n\), 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

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

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:
  \[
  \begin{array}{cccc}
  3 & d & 5 & 0 \\
  0011 & 1101 & 0101 & 0000
  \end{array}
  \]

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
  \[
  \begin{array}{cccc}
  100 & 1000 \\
  4 & 8
  \end{array}
  \]
  \(= 0x48\)
- Or, e.g., 110101110
  \[
  \begin{array}{cccc}
  1 & 1010 & 1110 \\
  1 & a & e
  \end{array}
  \]
  \(= 0x1ae\)
Text – Characters and Strings

• ASCII – American Standard Code for Information Interchange
  – 7-bit codes representing basic Latin characters and numbers [A-Z, a-z, 0-9], some common punctuation, and control characters
  – There are a number of extensions to 8 bits, but only the 7-bit codes really standard.
• Unicode – 8- or 16-bit codes extending to a much wider set of languages
  – The first 128 codes are equivalent to the 7-bit ASCII standard

C Strings

• Strings are sequences of ASCII characters, stored one byte per character (8 bits), terminated by a NULL (zero) character
  • E.g., “Hello!”
    01001000 'H' 0x48
    01100101 'e' 0x65
    01101100 'l' 0x6c
    01101100 'l' 0x6c
    01101111 'o' 0x6f
    00100001 '!' 0x21
    00000000 NULL 0x00

ASCII Facts

• Numerical digits are assigned in order of increasing value
  i.e., '0' = 0x30
            '1' = 0x31
            '2' = 0x32
            '9' = 0x39
  • For single character, value conversion is simply a difference of 0x30

More ASCII Facts

• Letters are also assigned in lexicographical order:
  'A' = 0x41
  'B' = 0x42
  'Z' = 0x5a
  'a' = 0x61
  'b' = 0x62
  'z' = 0x7a
  • Upper/lower case conversion is simply a difference of 0x20

Still More ASCII Facts

• First 32 characters (0-0x1f) are control codes:
  0x00 ^@ null (C string terminator)
  0x07 ^G bell
  0x0a ^J line feed
  0x0c ^L form feed
  0x0d ^M carriage return

Line breaks are not standardized

• End of line conventions differ by operating system:
  – In MS Windows: 0x0a, 0x0d is end of line
  – In Unix/Linux: 0x0a is end of line
  – 0x0a, linefeed, is sometimes called ‘newline’
• In C, ‘\n’ is mapped to OS end of line termination convention
Java Strings

- Strings are represented via the class “String”
- String objects are immutable
- The character encoding is system specific, e.g., either UTF-8 or UTF-16 (typical).
- The length is an instance variable in the object (in most implementations)
- The characters are stored in a char[] array (again, in most implementations)

Unicode

- Standard for character representation
  - Supports wide variety of languages, symbols
- UTF-8
  - Variable length code with 8-bit code units
  - U+0000 to U+007F are the same as ASCII
- UTF-16
  - Uses 16-bit code units, also variable length
  - Latin + Greek + Cyrillic + Coptic + Armenian + Hebrew + Arabic + Syrian + Täna + N’ko fit in 16 bits
- UTF-32
  - Uses 32-bit code units, fixed length

Studio Today

- Come to Urbauer labs (load balance as on Wed)
- Form groups of 2 to 4
- Do the exercises
  - Some on whiteboard (no computer required)
  - Authoring simple C programs on the PC
- Get signed out by a TA
- Assignment 1 will start on Wed and be due Wed of next week (Feb 3) in lab
  - Don’t assume lab time next week to complete it!!!!