15213 Recitation 1
In case you didn’t get it (or you were too lazy to check)

• Class Web page: http://www.cs.cmu.edu/~213
• No Blackboard, no Piazza
• Questions? Email 15-213-staff@cs.cmu.edu
• Office hours: MTWR, 5:30-8:30pm, WeH 5207
• Need help? 1:1 appointments available
• Sharks Machines: ssh shark.ics.cs.cmu.edu
Fun Stuff

- 7 labs, 1 midterm, 1 final
- Labs 1-6 are individual. Lab 7 is a partner lab.
- All assignments due 11:59 pm on their respective due dates
- Conflicts – talk to us AHEAD of time
- Grade appeals only good for 7 days after grade release – formal procedures in syllabus
- 5 grace days, max of 2 per lab
- No grace days? 15% per day penalty afterwards
- No handin after 3 late days
Just in case

• Cheating is bad
• Don’t cheat
• Cheating is bad
Bits/Bytes/Ints Overview

• Integers stored in binary representation
• Byte = 8 bits
Different Bases

- Decimal: $100_{10} = 100_{10}$
- Binary: $100_{10} = 1100100_{2}$
- Hexadecimal: $100_{10} = 64_{16}$
# Data Representations

<table>
<thead>
<tr>
<th>C Data Type</th>
<th>Typical 32-bit</th>
<th>Intel IA32</th>
<th>x86-64</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>long long</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>long double</td>
<td>8</td>
<td>10/12</td>
<td>10/16</td>
</tr>
<tr>
<td>pointer</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>
Boolean Algebra

- AND, OR, NOT, XOR
- Nice figures on slides on website
Representing Sets

- Width $w$ bit vector represents subset of \{0,1,..., w-1\}
- 11011 = \{0,1,3,4\}
- 0101 = \{0,2\}
Bit-Level Operators

• AND = &
• OR = |
• NOT = ~
• XOR = ^
• Applies to integer types: char, short, int, etc
• ~0x0 = 0xF
• 0x4 | 0x6 = 0x6
• 0xA & 0X6 = 0X2
Logical Operators

- AND = &&
- OR = ||
- NOT = !
- 0 = “False”
- Nonzero = “True”
- (!0x0) = 0x1
- (!0x4) = 0x0
- 0xBEEF && 0xDEAF = 0x1
- 0xFEED && 0xDEED = 0x1

- Short-circuit. If second expression does not need to be calculated, the machine does not. For example, if c = 0,
  - (++c || ++c)
  - Value of expression above is 0, but c is now 1, not 2
Fun Fact to Keep You From Insanity

• && and & are NOT the same thing. && applies to logical expression while & applies to bit (or bitwise vectors)
• Similarly for || & |
• If funny stuff is happening in controls, check your conditionals for these mistakes
Shifts

• Left shift: $x << y$. Shift bit vector $x$ left by $y$ positions. Discard the extra bits on the left. Fill new bits on right with 0’s

• Right shift: $x >> y$. Shift bit vector $x$ right by $y$ positions. Discard extra bits on right.
  – Logical: Fill new bits on left with 0’s
  – Arithmetic: Fill new bits on left with most significant bit of $x$

• If $y < 0$ or $y >=$ word size, undefined behavior
Encoding Integers

- int =/= integers
- Unsigned: \( B2U(X) = \sum_{i=0}^{w-1} x_i 2^i \)
- Signed: \( B2T(X) = -2^{w-1} + \sum_{i=0}^{w-2} x_i 2^i \)
- Sign bit: most significant bit indicates sign for two’s complement numbers
  - 0 for non-negative
  - 1 for negative
Two’s Complement

- $-x = \sim x + 1$
- $x = 00000110_2 = 6$
- $\sim x = 11111001_2 = -7$
- $-x = 11111010_2 = -6$
Range

• Unsigned
  – $UMin = 0 = 000 \ldots 000_2$
  – $UMax = 2^w - 1 = 111 \ldots 111_2$

• Signed
  – $TMin = -2^{w-1} = 100 \ldots 000_2$
  – $TMax = 2^{w-1} - 1 = 011 \ldots 111_2$
Casting

• The small details are on the website pdf
• Main points to take away:
  – There is a unique nice bijective mapping between signed and unsigned words
  – Bit pattern maintained, only reinterpreted
More fun fact to save you from insanity

• In expressions mixing ints and unsigned ints, ints are casted to unsigned ints first!
  – (unsigned) 0 > (signed -1) returns 0 (false)
  • :O
Expanding

• Unsigned: 0s added
• Signed: sign extension
• Both yield expected results
Truncating

- Unsigned/signed: bits truncated
- Results reinterpreted
- Unsigned: mod operation
- Signed: similar to mod
Addition

• Unsigned:
  – Ignore carry: addition modulo $2^w$
  – (unsigned char): $240 + 56 = 40$

• Signed
  – Again ignore carry and treat remaining number as signed
  – (signed char): $127 + 1 = -1$
Multiplication

• Similar rule of addition apply
Lab 1: Data Lab

• Not a very hard lab
• Still, start early to prepare for any unforeseen issues with code/Autolab/Shark/Andrew/The World
• More fun facts to save you from insanity:
  – Declare all your variables at the very beginning of each function, or the dlc will cause you much pain.
  – Work on the shark machines. Or else the dlc may not work and cause you much pain.
Tips for Starting Off 15213
Shark Machines

• Use SSH to login to the machines. You can pick which shark you like.
  – Choose your favorite shark: angelshark, bamboo shark, baskingshark, blueshark, carpetshark, catshark, hammerheadshark, houndshark, lemonshark, makoshark
  – Then login to the host, replacing (shark) with your favorite shark: (shark).ics.cs.cmu.edu

• If you are ambivalent, or indifferent, or indiscriminant, you can simply login to the following host:
  – shark.ics.cs.cmu.edu
Get a REAL Operating System

• How to login to the Shark machines
  – If you’re on Linux, use the following command in terminal
    • ssh <andrew id>@<shark>.ics.cs.cmu.edu
    • Replace <andrew id> with your Andrew ID
    • Replace shark with your favorite shark
  – If you’re on a Mac, get a REAL operating system. Like Ubuntu. Or Arch. Or Debian. Or follow the same command above.
  – If you’re on a Windows, get a REAL operating system. Like Ubuntu. Or Arch. Or Debian. Or:
    • Get Putty for free online
    • Type the host name into the host name box (demo needed?)
    • Connect, and enter password when prompted
Running make

• On linux or Mac, in the directory in which you extracted your lab handout, simply run make to “make” and build your files.
  – cd lab_directory
  – make
GCC

• Compile C code
• `gcc -o <executable name> -Wall -Werror <.o files> <.h files> <.c files>`
• `gcc -c -o <object name> -Wall -Werror <.hfiles> <.cfiles>`
Coding

• 80 characters per line – we will check! And deduct style points.
• Make your variables meaningful. Bad variable names are a, b, c, temp, tmp, fat (unless you are measuring fat content of some substance)
Productivity

• Use vim, emacs, gedit, nano, pico, notepad+
• You could also use Wordpad, Microsoft Word, OpenOffice, LibreOffice.
  — Hint: Don’t
• Syntax Highlighting is good
• Compile and test your code on your machine!
  Don’t repeatedly submit to Autolab. It’s slow for you and for us and for everybody else.