CSE 351: Week 2

Tom Bergan, TA
Today

• Lab 1
  - refresher on binary and hexadecimal
  - tips and tricks

• Debugging with gdb
  - this will be useful for lab 2!
Binary Numbers

0 0 0 1 0 1 1 0

$2^7 \ 2^6 \ 2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0$

$= 2^4 + 2^2 + 2^1$

$= 16 + 4 + 2$

$= 22$
Two’s Complement

(figure stolen from lecture slides)
Two’s Complement

Computing negative one (-1):

this is 1: 0001

~  invert each bit

   1110

+1  add one

this is -1: 1111

(figure stolen from lecture slides)
Two’s Complement

How to negate a number:

```c
// Negate x without using -
int negate(int x) {
    return (~x+1);
}
```

(figure stolen from lecture slides)
Hexadecimal Numbers

0 0 0 4 B E E F

\[ \begin{align*}
&= 4 \cdot 16^4 + 11 \cdot 16^3 + 14 \cdot 16^2 + 14 \cdot 16^1 + 15 \cdot 16^0 \\
&= 311,023
\end{align*} \]
Binary To Hexadecimal

1101 0110  =  0xD6

This is really easy! 😊

4-digit Binary to Hex

0 = 0000  8 = 1000
1 = 0001  9 = 1001
2 = 0010  A = 1010 (= 10)
3 = 0011  B = 1011 (= 11)
4 = 0100  C = 1100 (= 12)
5 = 0101  D = 1101 (= 13)
6 = 0110  E = 1110 (= 14)
7 = 0111  F = 1111 (= 15)
Lab 1 Hints: The ! operator

!x means “not x”
- As in, “x is not true”

In C:
- 0 becomes 1
- everything else becomes 0

Examples:

!0 = 1
!1 = 0
!42 = 0
!99 = 0
Lab 1 Hints: The ! operator

A trick in C:
- Say you want to return 1 if x is positive, and otherwise 0
- Double-! does that:

  \[ \begin{align*}
  \text{!!0} &= \text{0} \\
  \text{!!1} &= \text{1} \\
  \text{!!42} &= \text{1} \\
  \text{!!99} &= \text{1}
  \end{align*} \]
Lab 1 Hints

Use DeMorgan’s Laws

! (A & B) = !A | !B

! (A | B) = !A & !B

What does $2^n$ look like?
all zeros except for one bit: 0000010000000
computing $2^n$: 1 $<<$ n

What does $2^n-1$ look like?
all zeros then all ones: 000000111111111
Lab 1 Hints

Do the easy problems first

isZero(), getByte()

Decompose into an easier problem

example: isMinusOne(x) {
  return isZero(x + 1);
}

eample: isOne(x) {
  return isZero(x + (~1+1));
  or:  return isZero(x ^ 1);
  or:  return isZero(x >> 1) & !isZero(x);
}

use subroutines like this while you’re figuring out the problem
Lab 1 Hints

Decompose into groups of bits

example: To solve

Try

isFoo(00110101),
isFoo(00) isFoo(11) isFoo(01) isFoo(01)
Lab 1 Hints

Take advantage of overflow/wraparound

Example: this is a big positive number $\text{0x7FFFFFFF}$

$\text{0111111111111111}$

what happens when you add two of them?

$\text{0x7FFFFFFF} + \text{0x7FFFFFFF}$

it overflows to a negative number $\text{0xFFFFFFFFFFE}$ (this is $-2$)
Today

• Lab 1 tips

• Debugging with gdb
  - this will be very useful for lab 2!

Demo
## gdb cheat sheet

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>help cmd</code></td>
<td>Get help about command “cmd”</td>
</tr>
<tr>
<td><code>run x y z</code></td>
<td>Run the program with command line arguments x, y, and z</td>
</tr>
<tr>
<td><code>Ctrl-c</code></td>
<td>Stop a program (e.g., in an infinite loop)</td>
</tr>
<tr>
<td><code>backtrace</code></td>
<td>Print a stack backtrace</td>
</tr>
<tr>
<td><code>break foo</code></td>
<td>Add a breakpoint at function foo</td>
</tr>
<tr>
<td><code>break foo.c:24</code></td>
<td>Add a breakpoint at line 24 of file foo.c</td>
</tr>
<tr>
<td><code>next</code></td>
<td>Execute one statement, then stop</td>
</tr>
<tr>
<td><code>step</code></td>
<td>Execute one statement, then stop</td>
</tr>
</tbody>
</table>

*next* and *step* treat function calls differently:

- **next** executes the entire function and then stops at the statement after the call
- **step** “steps into” the function, so it stops at the first statement inside the function
gdb cheat sheet

print x
print x+2
call foo(x)
x /4b 0xbeef
x /4b &first
x /1w &first
watch x

- print variable x
- print expression \( (x+2) \)
- call foo with argument \( x \) and print the return value
- print the first four bytes of memory at address 0xbeef
- print the first four bytes of memory at the address (&first)
- print to first word of memory at the address (&first)
- same as previous, except prints as one 32-bit number instead of four 8-bit numbers
- add a watchpoint on \( x \)
- will stop the program when \( x \) changes