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

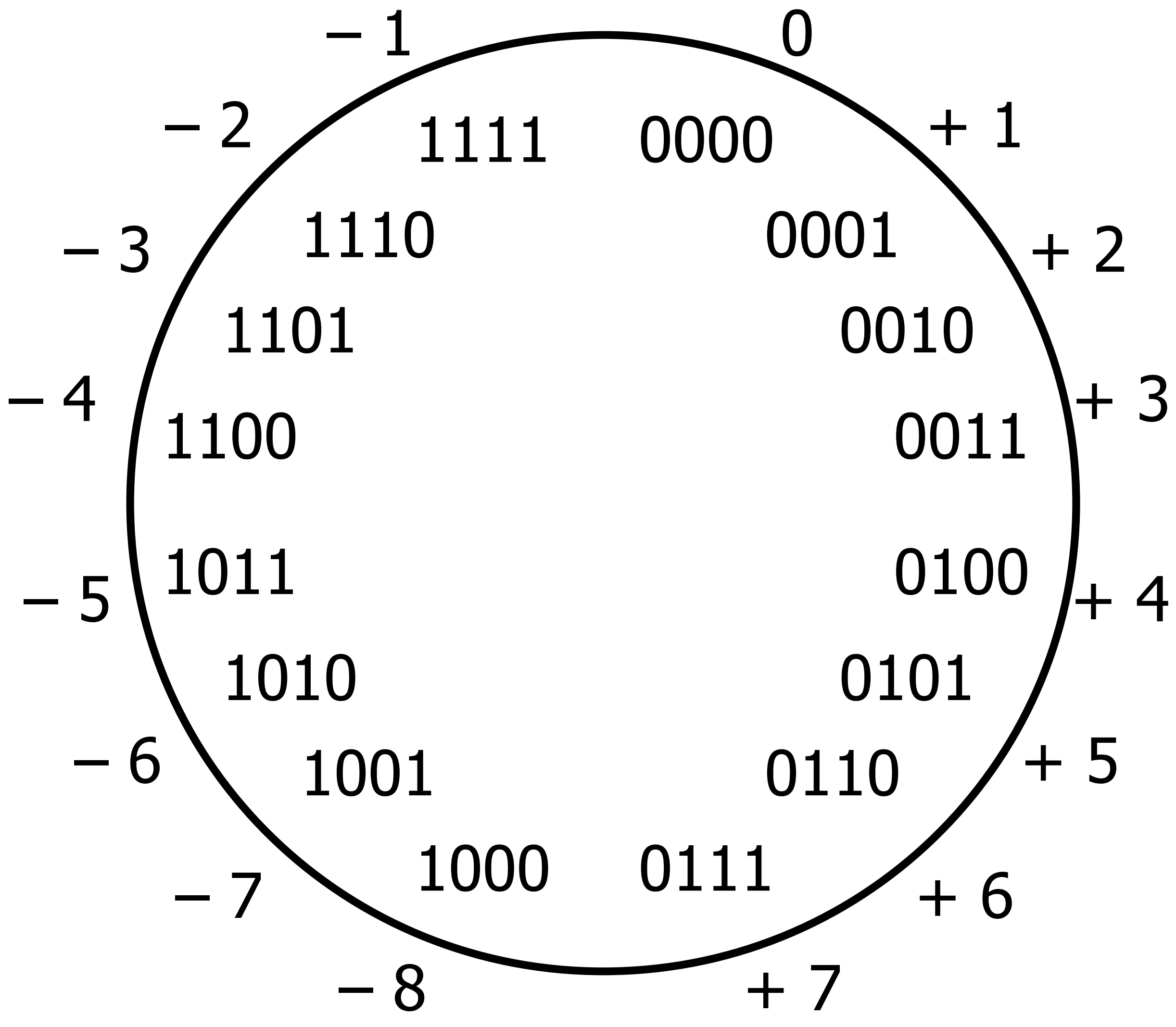
0	**0**	**0**	**1**	**0**	**1**	**1**	**0**
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

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

$$= 16 + 4 + 2$$

$$= 22$$

Two’s Complement



(figure stolen from lecture slides)

Two's Complement

Computing negative one (-1):

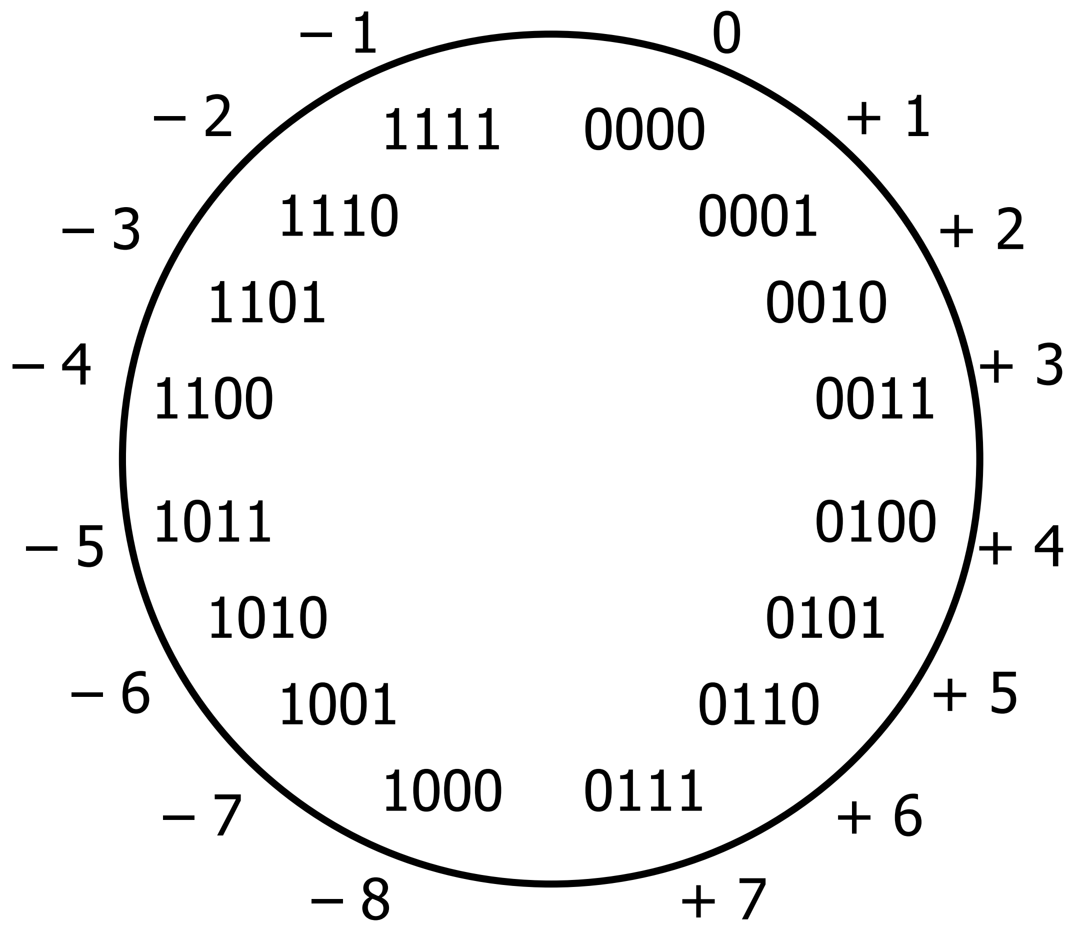
this is 1: **0001**

~ ↓ invert each bit

1110

+1 ↓ add one

this is -1: **1111**

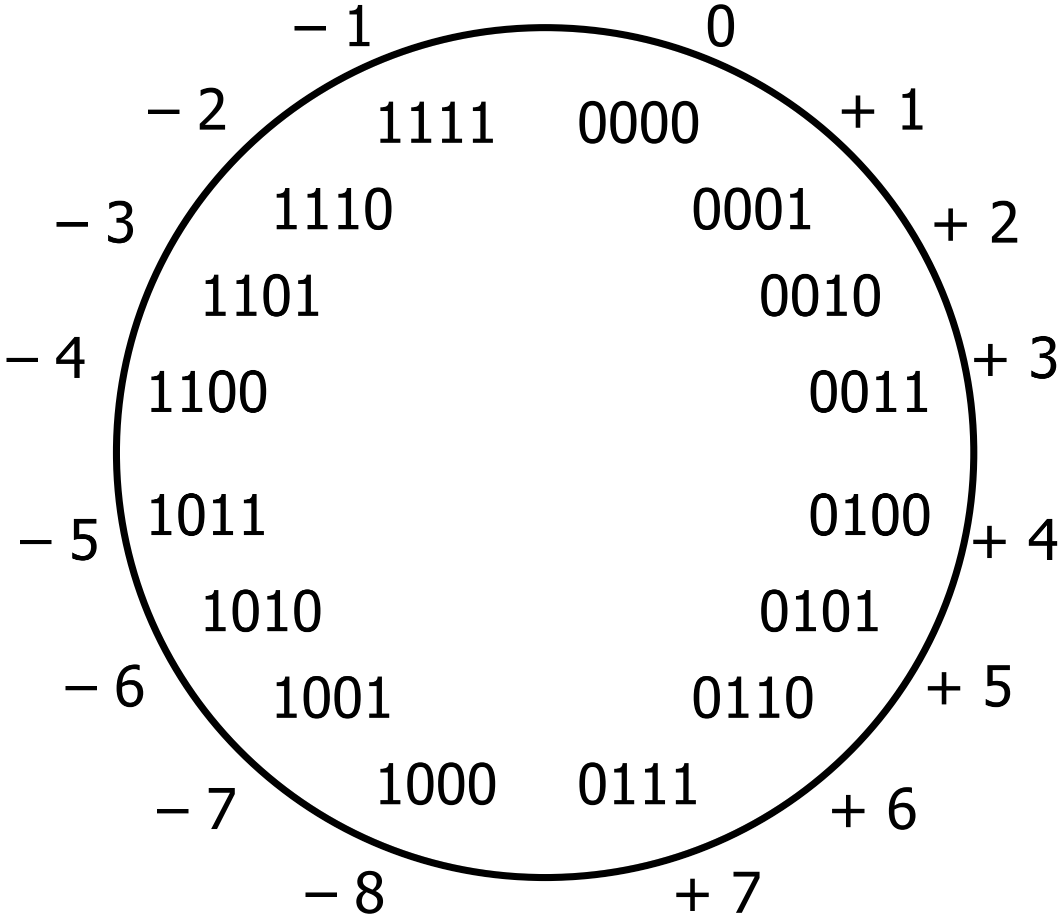


(figure stolen from lecture slides)

Two’s Complement

How to negate a number:

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



(figure stolen from lecture slides)

Hexadecimal Numbers

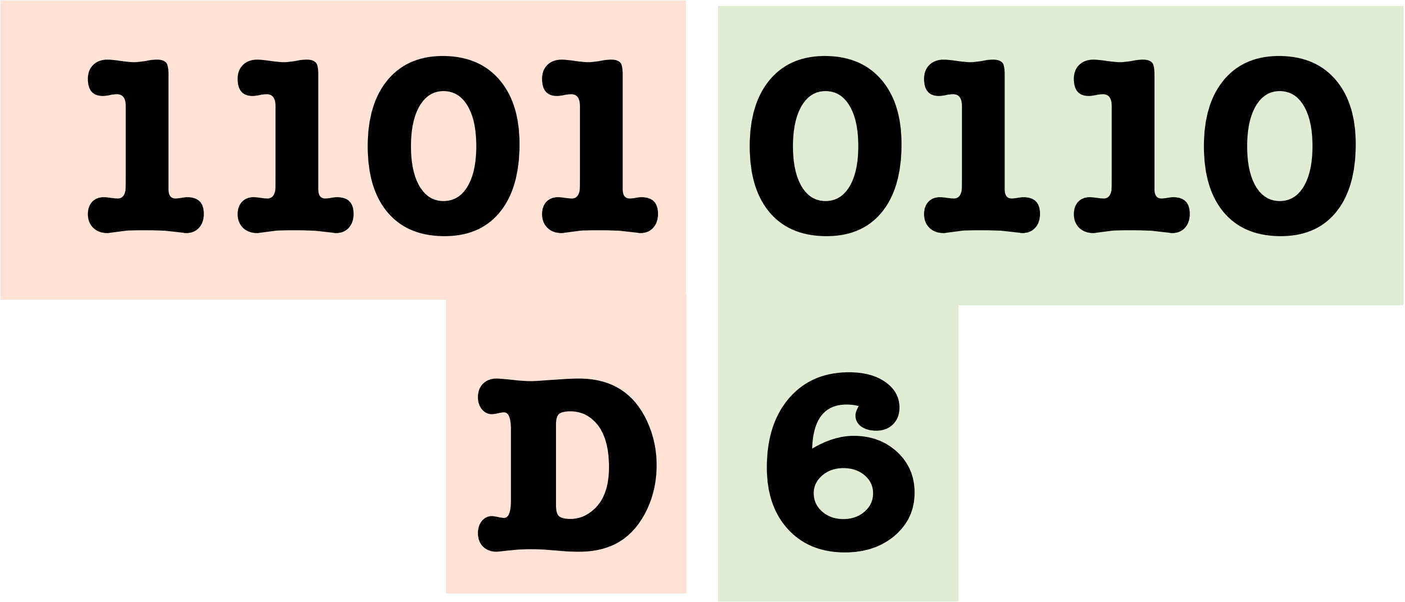
0 0 0 4 B E E F

16^7 16^6 16^5 16^4 16^3 16^2 16^1 16^0

$= 4 \cdot 16^4 + 11 \cdot 16^3 + 14 \cdot 16^2 + 14 \cdot 16^{1\ +}\ 15 \cdot 16^0$

$= 311{,}023$

Binary To Hexadecimal



1101 0110 = 0xD6

D 6

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:

```
 !!0 = 0
 !!1 = 1
!!42 = 1
!!99 = 1
```

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);
         }

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

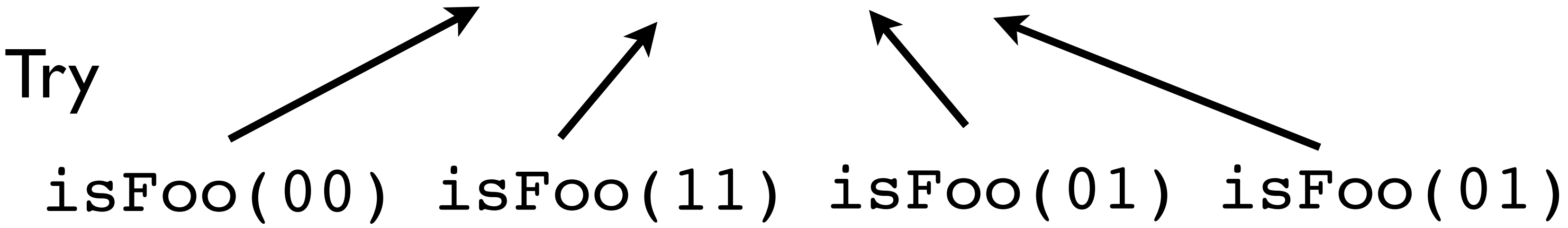
use subroutines like this while
you're figuring out the problem

Lab 1 Hints

Decompose into groups of bits

example: To solve `isFoo(00110101),`

Try



`isFoo(00) isFoo(11) isFoo(01) isFoo(01)`

Lab 1 Hints

Take advantage of overflow/wraparound

Example: this is a big positive number **0x7FFFFFFF**

01111111....1111

what happens when you add two of them? **0x7FFFFFFF**

+ 0x7FFFFFFF

it *overflows* to a negative number **0xFFFFFFFE** (this is -2)

Today

- ~~Lab 1 tips~~
- Debugging with gdb
    - this will be very useful for lab 2!

Demo

gdb cheat sheet

`help cmd`	get help about command “cmd”
`run x y z`	run the program with command line arguments x, y, and z
`Ctrl-c`	stop a program (e.g., in an infinite loop)
`backtrace`	print a stack backtrace
`break foo`	add a breakpoint at function foo - will stop the program when function foo is called
`break foo.c:24`	add a breakpoint at line 24 of file foo.c - will stop the program when it reaches line 24 of foo
`next`	execute one statement, then stop
`step`	execute one statement, then stop
	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 variable x
`print x+2`	print expression (x+2)
`call foo(x)`	call foo with argument x and print the return value
`x /4b 0xbeef`	print the first four bytes of memory at address 0xbeef
`x /4b &first`	print the first four bytes of memory at the address (&first)
`x /1w &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
`watch x`	add a watchpoint on x - will stop the program when x changes