(01)Today

- **This week**
  - Getting started on the Data Lab
  - Reading quizzes

- **Lecture Today:**
  - Finish up the Intro lecture
  - Few words about the Data Lab
  - Representing information as bits

(02) Getting Started on the Data Lab

Here is an *example workflow* for getting going on the data lab.

I. Find the lab invitation link on Moodle and accept the assignment. Your own GitHub repo will automatically get generated.

II. Go to coding.csel.io and clone your newly created repo (you will have a unique URL from the previous step). Now you can do things like compile, run the provided programs, and begin working on the lab.

1. Open up the Terminal, and begin issuing commands.
2. Issue the clone command, using the SSH version of your repo URL: `$ git clone git@github.com:cu-csci-2400-spring-2022/lab1-datalab-username.git`
3. Navigate to new directory

   `$ cd lab1-datalab-username`

Data Lab tips:

A. Run make to compile and run your code (Linux command that runs Makefile)

```
$ make grade
```

B. `bits.c` is where you solve the puzzles

C. Run **`btest`** to test individual functions from **`bits.c.`**
Example to test **`bitNor`** command:

```
$ ./btest -f bitOr -1 0x6 -2 0x5
```

D. Run dlc to compile and test for illegal operations

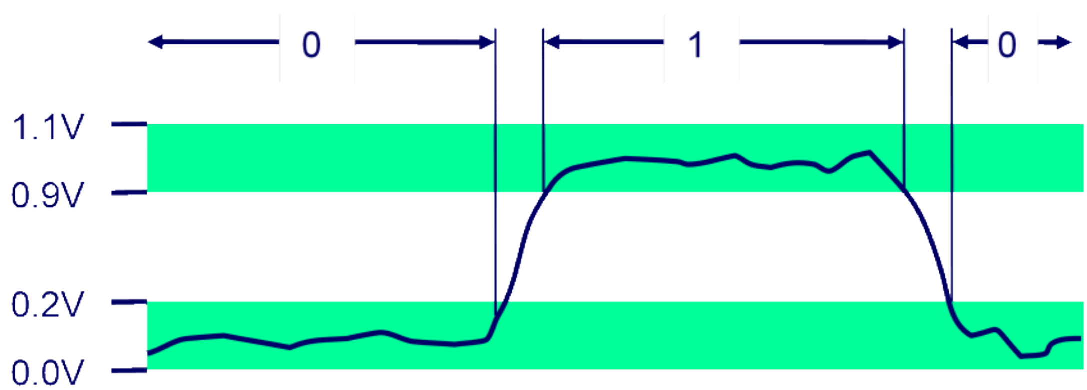
```
$ ./dlc bits.c
```

int bitOr(int x, int y)

(03) Everything is bits

- **Each bit is 0 or 1**
- **By encoding/interpreting sets of bits in various ways**
  - Computers determine what to do (instructions)
  - … and represent and manipulate numbers, sets, strings, etc…
- **Why bits? Electronic Implementation**
  - Easy to store with bi-stable elements

Reliably transmitted on noisy and inaccurate wires



(04) Encoding Byte Values

- One byte has 8 bits = 1 byte
- How many possible combinations can we generate?

  2^8 = 256   0 to 255

- If we take these combinations, we could use them to represent integer values
  - e.g $0000\ 0000_2 = 0_{10}$

    $0000\ 0001_2 = 1_{10}$

    $0000\ 0010_2 = 2_{10}$

    ...

    $0000\ 1001_2 = 9_{10}$

    $0000\ 1010_2 = 10_{10}$

- What about converting from binary to decimal?

`0110 0111`

(05) Hex number base

- Yet another number base
- Makes for a nice compact representation of binary
- Base 16 number representation
- Use characters ‘0’ to ‘9’ and ‘A’ to ‘F’
- Convert 11101101101101_2

  $11\ 1011\ 0110\ 1101_2$

  $3 \quad B \quad 6 \quad D_{16}$
- C syntax exmple: FA1D37B16

```
int x = 0xFA1D37B;
int x = 0xfa1d37b;
```

- Byte = 8 bits = 2 hex digits
  - Binary 00000000_2 to 11111111_2
  - Decimal: 0_{10} to 255_{10}

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

(06) Example Data Representations

How are bits used to represent data by the computer? Depends on the data type.

- Each data type has a set number of bytes at its disposal

C Data Type	Typical 32-bit	Typical 64-bit	x86-64
`char`	1	1	1
`short`	2	2	2
`int`	4	4	4
`long`	4	8	8
`float`	4	4	4
`double`	8	8	8
`long double`	–	–	10/16
pointer	4	8	8

# of bytes

(07) Boolean Algebra

What sort of operations can we peform on binary (logical) value?

- **Developed by George Boole in 19th Century**
- Algebraic representation of logic
  - Encode “True” as 1 and “False” as 0
- **Similar rules to integer numbers, but not exactly same. Examples:**
  - a*(b+c) = a*b + a*c
    - true for both
  - a+(bc) = (a+b)(a+c)
    - true in boolean algebra
    - not true in integer algebra

And, &, *

A	B	A&B
0	0	0
0	1	0
1	0	0
1	1	1

Or, |, +

A	B	A | B
0	0	0
0	1	1
1	0	1
1	1	1

Xor, ^ , ⊕

A	B	A^ B
0	0	0
0	1	1
1	0	1
1	1	0

Not, ~

A	~A
0	1
1	0

(08) General Boolean Algebras

- **Operate on Bit Vectors**
  - Operations applied bitwise
- **All of the Properties of Boolean Algebra Apply**
  - (bitwise)

```
  01101001     01101001     01101001
& 01010101   | 01010101   ^ 01010101   ~ 01010101
  01000001     01111101     00111100     10101010
```

(09) Bit-Level Operations in C

- **Operations &, |, ~, ^ Available in C**
  - Apply to any “integral” data type
    - long, int, short, char, unsigned
  - View arguments as bit vectors
  - Arguments applied bit-wise
  - **Examples (char data type - 8 bits)**
    - ~0x41
    - ~0x00
    - 0x69 & 0x55
    - 0x69 | 0x55

(10) Contrast: Logic Operations in C

- **Contrast to Logical Operators**
  - &&, ||, !
    - View 0 as “False”
    - Anything nonzero as “True”
    - Always return 0 or 1

      early termination

```
int x, y, z
....
if(!((x==0) && (x>y) || (z<256)){
...
z= ~(x&y)|z;
}
```

- **Examples (char data type)**
  - !0x41
  - !0x00
  - !!0x41
  - 0x69 && 0x55
  - 0x69 || 0x55

(11) Masks and Shifting Bit Vectors

- **Bit vectors are commonly used for *masks***
- **Typically involves shifting bit vectors**
  - $1011\ 1110_2$ << 3 becomes $1111\ 0000_2$
  - $1011\ 1110_2$ >> 3 becomes $0001\ 0111_2$

    or 1111_0111_2
  - Logical or arithmetic shift depends on the “integer representation”
    - Will need to understand difference between encoding unsigned vs signed integers in C

(12) Bit-wise Programming

Extract Last Byte

- **Task:** Given hex value like
  0xb01dface, extract last byte ('ce')

  0xb01dface & 0x000000ff

Extract All but Last Byte

- **Task:** Given hex value like
  0xb01dface, extract all but last byte ('ce'), e.g. 0xb01dfa00

  0xb01dface & ~0x00000ff

Extract Byte w/ Shift & Mask

- **Task:** Given hex value like
  0xb01dface, extract 2nd to last byte (0x**fa**)

  ((0xb01dface) >> 8) & 0xff

Change byte w/Shift

- **Task:** Given hex value like
  0xb01dface, change 2nd to last byte (0xfa) to 0xbd
-

  0xb01dface & (~(0xff <<8))

  0xb01d00ce | (0xbd<<8)