EECS 370

Lab 1: Binary & C

Lab Logistics

Attendance is **required**, and you must attend your section (waived for this week)

For occasional conflicts or sickness/emergencies, 2 lowest lab scores are dropped **automatically**

If there will be an ongoing conflict, please switch sections. Check the form on our website for help.

Recordings of content without assignments are on our YouTube.

You can find these slides in our Google Drive, usually the night before lab.

Upcoming Assignments

- Lab 1 due Wednesday @ 11:55 pm ET
  - Labs will always be due Wednesday @ 11:55 pm ET
  - No late submissions. Ideally, labs should be submitted during lab section.
- Check the syllabus for details about drops & late submissions.

Grade Breakdown

- 20% - Asynchronous lecture quiz
  - Due 11:55 pm Wednesday **BEFORE** lab
  - Covers lecture material from previous week
  - We'll give everyone credit for first lab
- 40% attendance
  - Expected to be here at lab start (10 minute grace period, your responsibility to check that you are marked present)
  - Required to stay until the end, unless you finish assignment and check with me first
- 40% group assignment
  - Submit to Gradescope (and sometimes autograder) by 11:55 pm Wednesday **AFTER** lab
  - **EVERYONE**'s responsibility to ensure assignment is submitted
  - Assignments will be done individually for first lab - later ones will be group assignments

This Lab will cover:

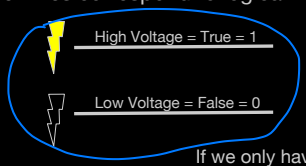
1. **Decimal, Binary, and Hexadecimal (Hex)**
2. Differences between C and C++
3. Visual Debugger Setup Tutorial

What's going in inside computers?

Computers are primarily made up of wires and ***logic gates.***

Follow the instructions in the lab assignment to read more on gates.

Voltages in the wires correspond to logical values.



If we only have 2 values per wire, we need new representations of numbers and other data…

Decimal, Binary, and Hexadecimal (Hex)

	Each digit represents a power of:	Each digit can take the values:	Example: 94_{10}	Declare the value in C/C++
Decimal	10	0-9 (digit)	94_{10}	x = 94
Binary	2	0, 1 (bit)	1011110_2	x = 0b1011110
Hex	16	0-9, A-F (hexdigit)	$5E_{16}$	x = 0x5E

Expanded form for Binary and Hexadecimal

	Decimal	Binary	Hex
Digits	9 4	1 0 1 1 1 1 0	5 E
Values	10^1 10^0	2^6 2^5 2^4 2^3 2^2 2^1 2^0	16^1 16^0
Conversion Back to Decimal	$9*10^1 + 4*10^0 = 94$	$2^6 + 2^4 + 2^3 + 2^2 + 2^1 = 94$	$5*16^1 + 14*16^0 = 94$

Problem 1: Conversion from Binary

Convert the following Binary into Decimal and Hex and submit:
`0101 1010 1101`

Decimal: Sum each of the representations

$0 + 2^{10} + 0 + 2^8 + 2^7 + 0 + 2^5 + 0 + 2^3 + 2^2 + 0 + 2^0 = 1453$

Hex: Sum representations *OR* Group bits into "nibbles" of 4 and convert

`0101 -> 5, 1010 -> A, 1101 -> D          = 0x5AD`

Octal (for fun): Can you figure out how to convert into base 8?

Basic Operations in Binary

Since bits represent T/F values, we can apply truth tables bitwise:

X	Y	X & Y
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	X | Y
0	0	0
0	1	1
1	0	1
1	1	1

Addition in Decimal and Binary

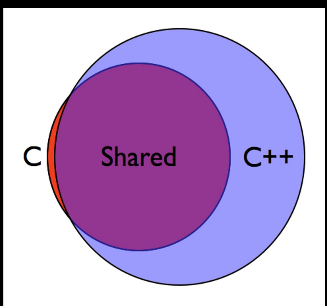
Binary addition is the same as decimal addition, just with 2 digits.

```
 11                       1   11
        37
   0101001
                +   65
+  0100011
                  102
1001100
```


This Lab will cover:

1. Decimal, Binary, and Hexadecimal (Hex)
2. **Differences between C and C++**
3. Visual Debugger Setup Tutorial

C vs C++: Important Differences



C++	C
Classes, Objects, Structs	Only Structs
`new, delete`	`malloc(), free()`
`std::string`	`char*, strcmp(), strcpy()`
`std::vector, std::set, std::map`	Only arrays
`std::cout, std::cin, std::fstream`	`printf(), scanf(), fopen(), fgets()`

The Size of Common Data Types

Data Type	Typical Size
char	1 Byte
short	2 Bytes
int, float	4 Bytes
double	8 Bytes
long *(32-bit Architecture)*	4 Bytes
long *(64-bit Architecture)*	8 Bytes
pointer *(32-bit Architecture)*	4 Bytes
pointer *(64-bit Architecture)*	8 Bytes

More about types in C

C doesn't have `bool` types (without `#include <stdbool.h>`)!

We have to use `int`s instead.

(Also watch out for `int32_t` - you must `#include <stdint.h>` to use this.
But `int` works just fine too.)

Decimal, Binary, and Hexadecimal in C

These formats are just different ways of **displaying the data for humans**. They are stored the same way in C, so there is no need to convert between formats in C.

```
int main () {
    int a = 15;          // a = 15
        int b = 0b1111;    // b = 15, b == a
        int c = 0xF;             // c = 15, c == b
}
```

They are only different when printing or reading in (we took care of reading for you).

We will show later how to debug with hex.

This is useful since every nibble (4 bits) in binary corresponds to a hexdigit.

Bit Manipulation in C: Bitwise AND (&)

The Bitwise AND operator (&) performs a logical AND *bitwise:* between each bit of two numbers

```
int a = 26  // … 0001 1010
int b = 15  // … 0000 1111
a = a & b   // … 0000 1010
```

X	Y	X & Y
0	0	0
0	1	0
1	0	0
1	1	1

This is extremely useful for *masking* bits (changing all but some useful bits to 0).

For b = 2^n - 1, a & b == a % 2^n. (P4 tip).

Bit Manipulation in C: Bitwise OR (|)

The Bitwise OR operator (|) performs a logical OR bitwise between each bit of two numbers

```
int a = 42  // … 0010 1010
int b = 4   // … 0000 0100
a = a | b   // … 0010 1110
```

X	Y	X | Y
0	0	0
0	1	1
1	0	1
1	1	1

This is extremely useful for *setting* bits without changing any other values

Bit Manipulation in C: Shifting

Left Shift << and Right Shift >>

```
int a = 18;       // … 0001 0010
int b = a >> 2;   // … 0000 0100 10
```

After assignment: b == 4. (18 / 2^2)

```
int a = 18;       // … 0001 0010 00
int b = a << 2;   // … 0100 1000
```

b == 72. (18 * 2^2)

Left shift multiplies by powers of 2, and right shift divides by powers of 2.

Note: Shifting doesn't actually change **a** here. If you wanted to modify **a**, you'd need to reassign (`a = a >> 2;`)

Decimal, Binary, and Hexadecimal in C

Know the difference between the following:

Logical/Boolean Operators		Bitwise Operators
&&	and	&
||	or	|
!	not	~
Used for conditions		Used for setting/extracting fields

Both work with C, but for different things!

Decimal, Binary, and Hexadecimal in C

Example: What's wrong with this code?

```
int i = 2;
if(i & (i<3)){              //If i is not 0 but less than 3
    printf("It works!!"); //This won't print on CAEN*.
}
// Don't forget the parentheses around (i<3).
    //See C order of operations to see why
```

*C booleans are handled differently by different systems. This is undefined behavior. The condition should use && for defined behavior.

Operator Precedence

- Basically PEMDAS for C and C++

(https://en.cppreference.com/w/c/language/operator_precedence)

1: Parentheses, Brackets
2: Unaries: Negative -, Logical not !, Bitwise not ~, casts, derefs *, refs &
3: Multiplication *, Division /, Modulus %
4: Addition +, Subtraction -
5: Shifts <<, >>
6: Comparisons <, >, <=, >=
7: Comparisons ==, !=
8, 9, 10: Bitwise: and &, xor ^, or | respectively
11, 12: Logical: and &&, or || respectively
13: Ternary ?:
14: Assignments with =
15: Comma

Translating from C++ to C

```
/**
 * Goal:
 * Print "Hello world! Jello world!"
 */

string hello = "Hello world!";
string jello;

jello = hello;
jello[0] = 'J';

cout << hello << " " << jello;
```

This won't work!!
It can be dangerous to think in C++ when writing C code, let's talk why

```
char* hello = "Hello world!";
char* jello;

jello = hello;
jello[0] = 'J';

printf("%s %s", hello, jello);
```

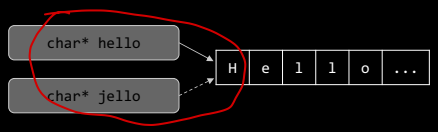

Why This Doesn't Work in C

```
char* hello = "Hello world!";
char* jello;

jello = hello;
jello[0] = 'J';

printf("%s %s", hello, jello);
```

Both **jello** and **hello** are *pointers* to a **char** array (C-string)

When performing the assignment, we will point **jello** to **hello**

This does not create any new data!



So what is the output of this code?

```
char* hello = "Hello world!";
char* jello;

jello = hello;
jello[0] = 'J';

printf("%s %s", hello, jello);
```

Solution:

```
Jello world! Jello world!
```

Take 15 seconds to come up with an idea.

How to Fix Our Implementation

```
char* hello = "Hello world!";
~~char* jello;~~
char jello[strlen(hello) + 1];

strcpy(jello, hello);
jello[0] = 'J';

printf("%s %s", hello, jello);
```

`jello` was initialized without a value!

If we try to copy into `jello`, we will **SegFault**

We must allocate space to store the data

`strlen()` gives us the length of an input string

C Strings and Structs

```
struct doublestring{
        char* a;
        char* b;
};
//This contains only pointers!
//The actual c-strings are
    elsewhere
//Can go out of scope before or
    after the c-string
```

```
struct doublestring{
        char[8] a;
        char[8] b;
};
//This contains actual c-strings!
//Each char is inside the struct.
//Goes out of scope with struct
//Makes copying really easy
```


printf and fprintf

printf and fprintf use *formatted strings* to produce output. The formatted string is then followed by the ordered list of values to insert...

Formatted strings contain *format specifiers* to insert useful values into output. These begin with %:

- `%c`: character
- `%s`: string (null-terminated)
- `%d`: decimal number
- `%x`: hexadecimal number //Great for P1a debugging

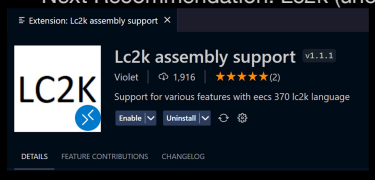
Example: say we want to print the value of an integer x.
We see lots of `printf(x)` instead of `printf("%d", x)`. The former is wrong.

Download 370 P1a Starter Code (Ex: Bash)

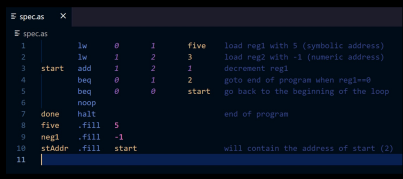
```
mkdir -p ~/eecs370/p1-assembler      #Can be different
wget https://eecs370.github.io/project_1_spec/starter_1a.tar.gz
tar -xvzf starter_1a.tar.gz
mv starter_1a/* ~/eecs370/p1-assembler/
cd ~/eecs370/p1-assembler
code . #Launch VSCode
```


Let's Look Around: Extensions

Next Recommendation: Lc2k (unofficial)

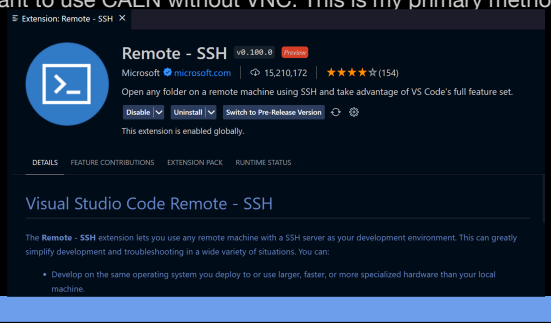


Now assembly code looks like this:



Let's Look Around: Extensions

If you want to use CAEN without VNC: This is my primary method.



Let's Look Around: Extensions

Themes! This is VITAL

There are a bunch: official + unofficial