ECE 241

Logical Equations:

In the first lab we are using one of the most basic programming construct, the "if" statement. The basic form of an if is as follows

```
if( LOGICAL )
{
  // The code between the two curly bracket {}
  // is run if LOGICAL is true.
}
```

So what is LOGICAL? This is a statement that evaluates to being true or false. The most common are the comparisons. Examples of the basic comparison operations are shown in the following table

Example of Operation	Description: True if ...	Notes
A > B	A is greater than B	
A >= B	A is greater than or equal to B	
A == B	A is equal to B	Beware since A = B will actual return a true, but also will assign B into A.
A <= B	A is less than or equal to B	
A < B	A is less than B	
A != B	A is not equal to B	
A	any bit in a is non-zero.	Equivalent to A != 0

Now what if there is more than one comparison, such as checking that a variable is greater than 0 and less than 100. In this case we will need to use logical operators (Not Bit-Wise Logical Operators). They have similar names, but can have a extremely different responses. The following table shows the operators.

Name of Operation	Written as	Notes / Example	Description of Example
AND	&&	if (A > 0 && A < 100) {...}	Runs if A is between 1 and 99
OR	||	if(A > 0 || B > 0) {...}	Runs if A or B is positive (> 0)
NOT	!	if(!A) {...}	Runs if A is false
XOR	!=	if (A != B)	Runs if A is true or B is true, but not both

Using logical operators, elaborate decisions can be made, but also rather subtle changes can have a large effect on a programs operation. Perhaps the best thing is to show a couple examples of how the operations are different.

First recall that an integer variable can be used as a logical, and is considered true if any bit is a one, and is false if and only if every bit is zero. Thus if we have the following two statements

```
int A = 0x22, B = 0x11;
if( A & B )
{
   // Will not run since
   //   A = 0010  0010
   // & B = 0001  0001
   //       0000  0000
   // Which is zero or false.
}

if( A && B )
{
   // Will run since
   // A is non-zero, which makes it true
   // and B is non-zero, which makes it also true
   // Thus both are true.
}
```

Masking can be employed as part of a logical expression. In Lab 2, we accessed the external pins (pin 12) of the Arduino using an internal variable called PORTB. The bits in PORTB are connected to pins 13, 12, 11, 10, 9 and 8. When looking at inputs on these pins, the variable PINB will reflect the value on these pins if they are set as INPUT.

Consider the case of wanting code to run if an input signal on pin 12 is high. We could set up an if inside the loop that looks like

```
if( PINB & 0x10 )  // All bits are forced to zero,
{                  // except for the bit representing pin 12
   // Will run when pin 12 has a high input.
     ...
}
```

Note that using an OR operation would be a mistake

```
if( PINB | 0x10 ) // The or operation will force one bit
{                 // high in the result and the code will
     …            // will always run.
}
```

ECE 241

A common way to test your ability to read and understand logical equations is to set up a series of logical equations or if statements and ask for what values the code would run. The following are examples of these, and are especially designed to demonstrate the subtly of these equations.

```
int A, B;
…
// Unknown batch of code.
…
if( A && !B )
{
   // Will run when A is non-zero and B is zero
   …
}

if( A && ~B )
{
   // Will run when A is non-zero and B is not equal to 0xffff
   // since ~ inverts all the bits, as long as B is not all ones
   // its inverse will have at least one bit that is one and
   // therefore true.
   …
}

if( A < 0 && A > 0 )
{
   // Will never run since no number
   // can be less than and greater than 0
   …
}

if( A < 0 || A > 0 )
{
   // Will run as long as A is not equal to 0
   …
}
```