# ECE 341 - Lab #0

Collin Heist

January 23, 2019

## Contents

## Listings

## 1 Introduction

The goal of this lab was to learn some of the important tools available to us through the MPLAB X IDE. This lab was also a quick refresh on a few basics of the C Programming Language.

We first launched MPLAB X, our chosen IDE for interfacing with the Microcontrollers. We then created a new project, selected the **PIC32MX795F512L** from the list of available 32-bit MCUs available, and selected the **CerebotMX7ck** licensed debugger so that we can use the available debugging tools provided by the IDE. We chose the X32 compiler, named the project, and were finally ready to actually add code to said project.

Since all the code was provided already, I simply used the *Add Existing Item* option on the Header Files subfolder for the project. I selected the provided .h files, and then repeated this process for the Source Files folder and .c files.

At this point, the 'setup' was largely completed, so I began the actual work of the lab.

## 2 Implementation

The only real work necessary was using the provided Debugger to step through the code inside lab0.c. To make my life easier, and avoid miscounting the number of times the main program loop iterated, I did create a local variable called **count** that incremented each loop. That code is shown in **Listing 1** below.

Listing 1: Counting Variable

```c
int count = 0;
a = ac;
b = bc;
while (1) {
  no_swap(a, b);
  c = swap(&a, &b);
  a = c - d;
  count++;
}
```

This code comes directly after the hardware initialization, and the only change I made was the addition of the count code. This variable allowed me to very easily check how many times the while loop had iterated, and how many swaps were performed.

## 3 Testing and Validation

Testing this week's code was very simple, especially since it was all provided for us. I was able to test each loop without even using a breakpoint, I simply put my cursor on the final line, and selected the debugging option to **Run to Cursor**, letting each series of swaps happen only once. Beyond performing that iteration four times to see what the values of each variable became, there were no necessary tests for code validation.

## 4 Questions

1. The difference between a **#define** statement, and declaring a variable outright is twofold. The **#define** is quite simple in implementation, as it just replaces all instances of the defined name with its definition, it is also defined at compile time and can never be changed. A variable, on the other hand, is stored in memory at a specific address. Because of that, you can change its value, ask for its size, etc. It also takes up space in memory that the define statement does not (as there is nothing to store).

2. If the code was changed to reassign **d** and not **a**, the code would not run. I am fairly certain the Compiler would throw an error as well. This is because **d** is defined as a constant upon its initialization. This means that at no point during the programs run-time can its value be changed at all.

3. Local variables are created in a few different ways:

    - Static Local Variables - A local variable with a fixed memory address.
    - Automatic Local Variables - A local variable with a temporary memory address.
    - Function-level Local Variables - A local variable created within a function call and de-allocated when the function returns.

4. The required information for a function prototype is the function's return type (even if **void**), the function's name as written in it's implementation, and the argument(s) variable types / names. I believe

in C++, the names of the variables are unimportant and even ignored by some compilers, but I am not sure whether C is the same.

5. With the changed code inside the main **while(1)** loop (so that **a = ac; b = bc;** is *above* the loop), it took 80 cycle counts to perform one execution of the code.

## 5 Conclusion

This introductory lab was quite enlightening. So far, the IDE is quite intuitive (although I do have prior experience), and all of the available Debugging tools are sure to be very useful in future labs. Reviewing the differences between passing by value and reference was also useful, as that level of C mastery is easy to forget, but very important. There is no real 'design' for this lab, so there are no inherent limitations present.