# Lab 1: Laboratory Report

**Presented to Professor Donald Peter Davis**
**For the course ECSE 324: Computer Organization**
**On 02/15/2019**

**Presented by:**


| | |
|---|---|
| **Roger Zhang** | **260805143** |
| **Antoine Hamasaki-Belanger** | **260800991** |

# 1. Learning the basics

### 1.1 Description

The goal of the first part was simply to familiarize ourselves with the Intel FPGA Monitor program and the DE1-SoC board. The lab documentation walks through the process of creating a new project. As an example, it used a simple program that finds the maximum and the code was already written. The documentation showed how to use the editor, the different components on the screen, the disassembly tab, the debugging process, etc.

### 1.2 Approach

The pre-written code in the lab documentation uses a register to point to where the result is stored. It then uses that same register and adds 4 bytes to retrieve the number of numbers to look through and the start of the list of numbers.

The set-up finished, the program enters a loop. Each time the loop starts, a register storing the number of numbers is decremented and acts a loop counter. The register pointing to the checked number is then moved forward by 4 bytes to check the next number in the list. This number is compared to the current maximum and replaces is if it is found to be bigger. When all the numbers have been checked (loop counter gets to 0), the program ends and the max number is stored in memory.

### 1.3 Challenges

The biggest challenge was to understand registers and memory interacted together. The difference between LDR and MOV, for example, were not instinctive to us and we had to play with the program multiple times before understanding. Since all the code was already pre-written, there were no other significant challenges.

# 2. Standard deviation

### 2.1 Description

The goal of this exercise was to find the standard deviation of a list of values using the following formula: $\frac{max - min}{4}$ . As input values, the program accepts the number of values and the list of values.

Therefore, during the lab, the max and the min were successfully found. The min was then subtracted from the min and the difference divided by 4. Finally, the result was stored in memory.

### 2.2 Approach

This program starts with a few set-up lines of codes just like in part 1. One register is set to point to the number of values (and will also act as a loop counter) and another is set to point the first value in the list.

Then follows 2 loops. The first one finds the max value in the list and is copied from the first part. This first loop is followed by a small reset phase where the max is moved to another register, the register pointing to the number being checked is reset to the first value and the loop counter is reset. The second loop then executes and finds the min in a similar fashion as in the previous loop. Since compare is simply a subtraction and checking if the result is negative, switching the values from CMP R0, R1 to CMP R1, R0 allows finding the min instead of the max.

Finally, the max is subtracted from the min and divided by 4 by shifting the difference twice to the left.

### 2.3 Challenges

The biggest challenge for this exercise was to figure out how some operations impacted the flags in the CPSR. For example, it took some time to understand that the program has no idea what happened in the previous line. It has to check the flags in the CPSR to refer to a previous result. Understanding that notion made it a lot easier to finish this exercise and the ones after. It allowed us to understand and properly use operations such as BGE and SUBS.

One of the problems with our code is that it is inefficient. The major improvement we could do is to combine the 2 loops into 1.

## 3. Centering

### 3.1 Description

The goal of this exercise was to calculate the average of an array of numbers and subtract that average from each number in order to obtain a signal which is centred. As input values, the program accepts the number of integers and the list of integers.

### 3.2 Approach

This program starts by storing the address of a variable called AVERAGE into register 0. Afterwards, a loop goes through the list and calculates the sum which is stored in R1. Another label called DIVIDE divides the number of integers by 2 n times until the result reaches 0. This is the conditional case which determines how many times the sum will be divided by 2 and thus shifted right. Finally, the average is obtained and it is subtracted from all integers of the list.

### 3.3 Challenges

The biggest challenge was to determine how many times to divide the sum by 2 in order to obtain the average. Since the only procedure for a division that exists in assembly language is shifting right, we needed to find a way to determine the number of times to execute a right shift. Fortunately, the number of elements in the list was constrained to be a power of 2. As a result, we established that the number of iterations needed to divide the number of elements by 2 until it reached 0 would be equal to the number of right shifts instructions needed to calculate the average.

## 4.Sorting

### 4.1 Description

The goal of this exercise was to sort a list of numbers ranging from minimum to maximum. As input values, the program accepts the number of integers and the list of integers.

### 4.2 Approach

We used the simple bubble sort algorithm which was provided to us. We first represented a boolean value false using 0 and stored that value in the R0. We then have an outer loop which checks if the value of R0 is false. If it is, end the program. If not, then the array is not completely sorted and it enters the inner loop. The inner loop then sets the boolean value to true and checks the current number and compares it to the following; if it is bigger, swap it with the one following it and set the boolean to false and return to the beginning of the inner loop. The inner loop ends when the loop has iterated through all the numbers so when the counter R1 reaches 0.

### 4.3 Challenges

The biggest challenge was to execute a swap and return the current iteration of the inner loop. Unlike in high-level programming where we have stack frames, assembly language does not have stack frames so going from one method to another proves quite complicated. For example, calling the label SWAP from the inner loop allows to execute the swap, but we cannot go back directly to where we were in the inner loop. As a result, once in the swapped label, we decided to increment the counter of the inner loop by 1 and go back to the beginning of the inner loop. Thus, since the state of the array would remain unchanged, it was a way to return to where we were when we initially called the swapped label.