ECSE 324 Lab 1 Report

Archit Gupta (260720757)

Part 1: Finding Maximum Number

The first part was just an introduction to ARM and assembly language. We didn't have to code anything. All the code was provided to us. We just had to implement it correctly and get familiar with the software used for this lab: the Altera Intel FPGA Program.

The code was structured efficiently which made it easy for us to learn the language. The regular syntax came first which indicated to the FPGA where the start of the program is located. Then came the loading instructions which assigned the various registers to the provided constants, result location, loop counter etc. After that was the main body in which the loop was situated, which effectively allowed us to loop over the given list of numbers, compare the numbers and thus find the maximum number. This maximum number was finally stored in the register allocated to this purpose specifically.

This part was pretty straightforward as all the code was given to us. We didn't face any problems and it was a good starting point for learning assembly.

PART 2: Standard Deviation

The second part required us to find the standard deviation of a list of numbers. Because the actual formula required an intricate knowledge of assembly and involved a much higher complexity, we used an approximation. This approximation required us to find the max., the min. and divide the difference by 4.

Much of the code was reused from part 1, as the process of finding the minimum number is very similar to finding the maximum number. The steps are the same; loading instructions which assigns the various registers to different values, main body (looping) and the final calculation. In the final step, when we have both the min. and max. Numbers, we need to divide the difference by 4. We achieve this by logically shifting the number of bits by 2 (as 4 = 2^2). Hence, we calculate the final standard deviation, which was stored in memory at a pre-defined location.

This part was also simple to execute as we just had to add a few more instructions to find the standard deviation. We also had to use logical shifting instructions which we looked up from the reference material. The best way to increase the accuracy of this program would have been to use the actual formula of standard deviation, but it would have been more complex at the same time.

PART 3: Centering an array

This part required us to centre the array by first calculating the average of the numbers in the array and then subtracting the array from each of the numbers so that the new list is centred around 0.

To achieve this, we used two loops. The first loop calculates the average of the numbers by adding the numbers and dividing by the size of the list. After that, the second loop comes into play where we subtract the average from each of the numbers to get a new list. We knew that the list length is a power of 2. This made things easier as we just needed to logically shift the sum by that power to get the average. The framework was a bit different though as after the regular loading instructions and constants into registers, we had to include two loops and a calculation in between those loops where we calculated the average. After getting the average, we had to reset the initial register values so that we can assign the new numbers into them. We can then check the new centred values by going into the memory tab of the FPGA software.

While the concept of this part was not hard to understand, implementation was a bit tricky as we would get confused between what the registers are supposed to store, the actual values or the memory addresses. It took us a while to fully understand this part and implement it correctly. One way to improve this program would be to accept as input a variable list length, instead of the list length necessarily being a power of 2.

PART 4: Sorting a list

This was the toughest program out of all the parts. It was because this problem required bubble sort which is implemented using nested loops and repeated iteration.

First step was similar as we loaded the registers to initialize the program. In this program, we created two loops with two different loop counters. The inner loop would iteratively select the numbers of the list. We then stored the first two numbers in two registers and compared the values. If sorting was required, we switched the values of the registers by using a temporary register. If there was no sorting to be done, we just decremented the loop counter and moved ahead with the iteration. After finishing the first pass, we exit the inner loop, come to the outer loop and decrement the outer loop counter. We then repeat all over again. After the final loop, all elements which needed to be swapped would have been swapped. We can just check the first number in the memory tab and see that successive memory locations contain the numbers in ascending order.

This part was the trickiest as we had to implement two nested loops and keep track of the outside and inside loop counters. We can improve the efficiency of this bubblesort algorithm by stopping at the loop in which no swaps took place, because that effectively means that our list is sorted. But all in all, we learnt a lot about how assembly language works from this particular program and the lab overall in general.