Introduction to ARM Programming

Lab Report 1

**Group #30**

Quentin Norris - 260621167

Ilana Haddad - 260687661

Computer Organization - ECSE 324

February 8th, 2017

Introduction

The purpose of Lab 1 was to serve as an introduction to the ARM processor and learn the basics of ARM assembly by programming three different assignments; the fast standard deviation computation, the centering of a signal, and the sorting algorithm bubble sort. We were given the first assignment, a max value calculator, to test the ARM processor in the Intel FPGA Monitor Program.

Part 1 Challenges

The first part of the laboratory assignment was to test code given to us through the Intel FPGA Monitor Program. The code itself was used to loop through a list of numbers and identify the largest number in the list. It did this by initializing registers to hold the value for the length of the list, a pointer to the first number in the list, and the maximum number found so far in the list. It would then compare the current maximum number with the next number in the list, and if the number was smaller it would loop to the next number and continue comparing. If not, it would replace the current maximum number with the new maximum number in the register and continue comparing again.

Although the code was already written for us, there were some slight challenges getting acquainted with the Intel FPGA Monitor Program and all of its functions. These included understanding the syntax necessary (specifically the spacing) for all assembly programs, as the program will not run if it does not have the exact syntax necessary. This is different compared to various other programming languages where syntax is command specific and will still run if the spacing isn’t consistent.

In addition, we were shown how to use the “debugging mode” provided by the software. This was a new and welcomed change to our habitual programming environment, but was still challenging to learn how to effectively make use of it!

Part 2: Standard Deviation

In this next part of the lab, we wrote our first ARM assembly program that computes the standard deviation of a signal using a simplified formula: . We used the program given to us from part 1 to calculate the maximum value and store it in memory for later use. Using similar logic, we iterated through the array to find its minimum value and stored it in memory. Then, we used the command SUB to subtract the minimum value from the maximum value and store the result in a separate register. The most challenging part came next, having to divide our result by 4. We used the command ASR, arithmetic shift right, that shifts the bits to the right by a value k, which equivalently means dividing a value by 2k. In our case, we wanted to divide by 4 so we shifted right by k = 2. Our final result for the standard deviation was stored in a register.

This being our first attempt at writing code in assembly, there are definitely some improvements we could have made. For example, after finding the max and the min, it was unnecessary to store those values into memory as it takes more time. We could’ve reduced the runtime of the program by using the temporary registers at our disposal to store our values.

Part 3: The Challenges of Centering an Array

This goal of this problem was to “enter a signal by calculating the average value of the signal and subtracting the average from every sample of the signal.” The program starts off by finding the average value of the inputted signal in a similar fashion to the previous assignments where we found the maximum and minimum values of the input. To do this we used a loop to go through the signal and add each value to a predetermined register. The next part was to identify how many numbers were in the signal so we could figure out the correct number to actually divide our total sum by. In order to do this we used the command LSR (Logical Shift Right) which divides the register we have chosen by 2^N. The challenge was to figure out what number the N value was so it could work for any type of signal we had inputted. In order to do this we used a loop that took the register that held the total length of our signal and continuously used a LSR to divide that register by two. We used a counter to keep track of how many times we divided by two, and we had the function stop once that same register [the one the previously held the length of the signal] was equal to one. This is how we determined the value of N to use in our LSR to calculate the average. Finally, we used another loop and the STR command to go through every value in our signal and not only subtract the average from it, but also replace that value in our signal. This resulted in a centered array, just as the problem asked.

The first problem we encountered which led to immediate hesitation was surprisingly not something to do with the actual coding of the lab. We were first confused about what the question meant by accepting “the signal length as an input parameter.” We thought that we had to take an actual signal and read it as an input to our program - this would’ve been outside the scope of what we had already learned in class so we quickly asked for help from our TA, Jerry. He explained that we could input the signal by recording it in memory utilizing the .word command. This misunderstanding with the concept of an inputted signal led us to our second big challenge during the coding of this part, which was calculating the average of the signal. The second challenge we faced was right before we demoed the program - our TA told us we couldn’t hardcode the value for N which we used in our LSR to determine the average value of the signal. We did not know this beforehand, so we had to devise a system to calculate this for any length of signal inputted. To do this we used a loop that took the length of the signal, and divided it by two, each time incrementing the counter we had by 1. We had the program branch out of the loop once the register that contained the value for the length of the signal reached one. We then stored the value of the counter into a register to use as our N, as the LSR divides the value of the chosen register by 2^N!

Improvements we could have made to our code would be to not hardcode anything like that again. It did not even occur to us what we were doing at the time, until the TA told us. We were very excited to devise a new system to identify the correct value of N to divide by. Another improvement, to match the wording of the question asked, would be to create a code to actually input the numbers you want in your signal at the start of the program, instead of choosing arbitrary values.

Part 4: Sorting

For the final section of this lab, we had to write an ARM assembly program which sorts an array in ascending order. For simplicity reasons, we used the bubble sort method to sort our array. Bubble sort iterates through the array and compares the value of two consecutive numbers, swapping them if the first is bigger than the second. This checking and swapping is done in a loop until the array is sorted.

The first challenge we faced was implementing an outer loop that checked if our array was sorted, and an inner loop that checked two neighboring elements of the array and swapped the values if necessary. For the outer loop, we needed to use some sort of boolean to keep track of whether the array was sorted, so instead we used the numbers 1 and 0 for true and false respectively. Then for the inner loop, using a pointer to the elements in the array, we compared the value of a first number with the one following it. If the first was greater, we swapped them, and if not, we moved on to the comparing the second number with the one following it. For swapping, we used two extra registers to temporarily store the values from the array and then stored them back into the array in memory having switched their order.

Finally, an improvement for this algorithm would’ve been to use a sorting algorithm that has a better runtime than bubble sort. In both best case and worst case scenarios, bubble sort has a runtime of ). If we had used merge sort for example, it has a runtime of ) for both best and worst case scenarios.