Introduction to ARM programming

Lab #1

Group 45
Giuseppe Lipari (260742811)
Alex Masciotra (260746829)

February 9, 2018

Introduction:

In this lab, we learned how to use the ARM processor and further advanced our knowledge in ARM assembly by programming three short routines. These routines were largest integer program, standard deviation computation, centering an array and sorting. This report aims to further explain the methods taken to complete these ARM assembly programs and the challenges faced.

Part 1: Largest Integer Program:

This section was the easiest of the four. It was an introduction to the Altera Monitor program. The task that was assigned to us was to copy and run a code given to us which computes the largest integer. This was a simple program that quite simply took in multiple integers then returned the greatest value. The code compares the current max value with the next value in the list. If the next value is greater, than it replaces it in the register. This first code was very helpful for us to learn the basic principles of the software and how the memory worked.

There weren't many challenges in this section since for the most part, it was just copying code. However, we realized the Altera Monitor program is a little tricky to use when it comes to debugging. It is a very meticulous program, which meant we had to revise our code several times to ensure that our spacing was right.

Part 2: Standard Deviation Program:

This section of the lab got significantly more difficult, since we were required to write this part on our own. In order to calculate the "fast standard deviation computation", we used the following formula:

$$\hat{\sigma} \approx \frac{x_{max} - x_{min}}{4}$$

Basically, the way this works, is that it uses the same algorithm from the "Largest Integer Program" in order to find the x_{max}. The challenges we were faced with were to find the x_{min} and be able to divide their difference by four.

In order to find $x_{min,}$ we needed to iterate through the array and store the smallest integer. We did this by creating a second loop which found the minimum element in the array. Once that was discovered, we subtracted the min from the max using the SUB command and divided it by four (using a shift instruction) which gave us our standard deviation.

The greatest challenge faced in this section of the lab was to divide by four. In all prior coding courses, dividing by four was quite simply just typing in the number. However, this was challenging because the register needed to be shifted by 2^k bits, where k=2 in this case, giving four.

One adjustment that could have been made is to have the max and min loop in one loop. This would have decreased the run time. Another way this could have been done would have been to sort the list, then take the first and last number to do the standard deviation computation. Finally, another improvement that could have been made would be to have the program check if the input array is empty and if it is, do nothing instead of blindly running through the program.

Part 3: Centering the array

In this section of the lab, we were tasked with creating a program that took in an array, found its size to then later center it. This was done by taking the average of the numbers within the array then subtracting each number from the average. However, a restriction was set for the program. The program was restricted to using the original memory locations and to not create a new array that contained the centered values. Negative numbers were also allowed in the centering calculations. They were represented in the memory using 2's complement.

The first task was to add all the values of the array. A register was used to store the total sum of the array. Another two registers are initialized with the first two values of the array and summed together. A loop was then created to continuously keep adding the next values of the array to the current sum register. The loop ended when the number of elements in the array minus one reaches zero, which means all the numbers in the array were added. In order to find the average, the number of bits that needs to be shifted is determined by the size of the array. To find the size of the array, a second loop was created. This loop took log base 2 of the sum and counted each time this happened until the sum was equal to 1, in order to find which power of 2 to divide by.

One challenge faced was the fact that the array can be of any size which made calculating the average difficult. This is because the program only ran flawlessly when the array size was a perfect power of 2. If this was not the case, as the loop counter was essentially taking the floor of the sum (check this math English). In turn, the average was not properly calculated. This part of the lab was quite tricky. Tests at each loop needed to be done in order to ensure that the logic was correct. By going step by step it became clearer where the errors were.

Part 4: Sorting:

For this section of the lab, the responsibility was to create a sorting method of our choice which would put the elements in the array in ascending order. We decided to use the bubble sort method. The basic principle of our sorting algorithm followed this method:

```
// Given an array A of length N
sorted = false
while not sorted:
    sorted = true
    for i = 2 to N:
        if A[i] < A[i-1], swap A[i] with A[i-1] and set sorted = false
```

In order to do this, we created two nested loops. The outer loop (first loop) quite simply scans from the beginning of the array to see if the elements are in order. The moment the outer loop realizes something is not in order, the inner loop is initialized. The inner loop (or the second loop) iterates through the array and if two neighboring elements are not in order it swaps them. The swapping is almost like a swap in Java. Two temporary registers are created. It stores both elements in the temporary registers, then swaps them and places them back into the original (real) registers.

The biggest challenge faced in this section was not knowing how to represent a Boolean in ARM assembly. After researching and discussing how to do this, we realized that we can achieve true or false statements by storing a 1 or 0 in a register. The 1 means the array is sorted and the inner loop would not need to be called. The 0 means it is not sorted and the inner loop, which does the swapping would need to be called. In terms of creating the swapping method, that was not too complicated. It followed identical logic to a Java or C swap that we learned in previous courses like COMP 250 or ECSE 202. One improvements that could have been made to shorten runtime is that one temporary register could have been created to do the swapping, instead of two. The only big challenge was understanding how to write it in ARM assembly.

One improvement that could have been made is to use a different type of sorting algorithm which would shorten the run time. From previous knowledge in COMP 250, we learnt that bubble sort has a runtime of O(n^2). Other sorting algorithms such as merge sort or heapsort have runtimes of O(nlogn). However, for the purposes of this lab, it was easier to understand how to write a bubble sort algorithm.