Lab 1 Report

ECSE 324

Introduction to ARM Programming

Group 13

Boqi Chen 260727855
Fandi Yi 260722217

LARGEST INTEGER PROGRAM

The first program is the code we simply copied from the lab instruction. The aim of the first program is letting us familiar with ARM instructions.

The code starts by assign values to four registers (R4, R3, R2, R0), The R4 points to the address used to store the result. The R2 holds the number of elements list in NUMBERS and note that the #4 means adding the address of R4 by 4 bytes (32 bits), which is the length of a “word” (same as directly read the label “N”). The R3 points to the address indicated by R4adding address by 8 bytes. Therefore, the R3 points to the first number in the list. R0 is used to hold the value of the first number in the list.

For the loop label, SUB R2, R2, #1 means decreasing the loop counter (from N=7 to N=0). The following code means if the N=0, then we enter the DONE label. Next, R3 will points to the next number in the list. Then store the number pointed by R3 to R1. After that, we indicated CMP R0, R1 to compare these two number, to check R0-R1 greater or equal to 0. If R0 greater or equal to R1 we continue the loop, otherwise, we update the current max in R0 to the value stored in R1. Branch backs to the top at the end of the loop.

As for the Done label, we store the maximum number to the Result label.

In the END label, we keep going loop (infinite loop) to keep the program up and running.

STANDARD DEVIATION PROGRAM

The second program is the code to find the standard deviation for the list of number by using the equation of $\frac{x_{max}-x_{min}}{4}$. We are inspired from the first program and following the ideas of the first program.

Firstly, the code starts by storing values to five registers. The R4 holds the address of the result location and the R2 holds the number of elements in the list. R3 points to the first number in the list. Both of them are achieved by adding constants to R4. We use R0 to hold the maximum number in the list and the to hold the minimum number. The general idea is similar to the Start label in the first program.

For the Loop Label, we indicate the loop counter (from N to 0) firstly. Let R5 hold the next number in the list. Then, checking if the number in R0 is greater than the number in R5. If R0 bigger than R5, we continue to check the minimum number. If R0 smaller than R5, we update R0 to R5.

As the screenshot shown above, we name two Labels in the code. They are called NOT_BIG and NOT_LESS separately. The code processes the check of the minimum number in the NOT_BIG label.

For NOT_BIG label, we compare R1 with R5 and check if the current number is smaller than the R1. If number stored in R1 smaller than R5's, we continue the loop by redirecting to NOT_LESS label. If R1 greater than R5, we update the number stored in R5 to R1.

In the Done label, the code do the calculation for standard deviation. At mentioned before, we use R6 to store the difference value between maximum number and minimum number (R0=max, R1=min). Then we use Mov R6, R6, ASR #2 to divided 4. Since it is binary number, when we move two bits right, the value will be divided by 2^2.
Finally, we save the standard deviation value to R4. And continue the program to an infinite loop to keep it running.

CENTERING PROGRAM

In this program, we ensure that the signal is "centered". Which means the average value of the list of elements is 0. In order to achieve that, we need to calculate the average value of the signal and subtracting the average from every sample of the signal.

For the first two labels, they are pretty similar to the previous two programs. We store some values to some registers at "_start" label and then use the loop label to check the elements in the list one by one (from first to the last). And in the lable, we use R3 as an accumulator to store the sum of all elements.

Then, in the CENTER label, we first restore some list information to R0 and R1. Then we use the POWER label to find out n where $2^n = L$, and L is the length of the list. We do this since because the specification ensures that length will be a power of 2. So in the UPDATE label we can use arithmetic right shift on the sum to achieve division by L to get the average. Lastly, in NORM, we just subtract every number by the average then store the number back to its original address.

SORTING PROGRAM

In this case, we need to sort the elements in list from minimum to maximum. We applied bubble sort as the general idea to write the code.

Firstly, we create two register to store the value 1 and 0 separately, while R1 is used to record how many times we go through the loop, R6 is used as an indicator whether the arrayhas been sorted
In the SORT label, we firstly get the length of the list. By bubble sort, we know that everytime we go through the array, the last element in the list will be the maximum value, so that next time we just need to go through $i - 1^{th}$ element. Then for the next loop we can just go though $i - 2^{th}$ element.

For the LOOP label, we compare the i^{th} element with $i - 1^{th}$ element and put the larger element to the right side. In the code, If R3 is larger than R3, we will swap the two numbers by storing them to each others' address.. Note that, we use R6 as boolean, which means when R6 =0, continue the loop and when R6 =1, stop this loop since we know that if we never swap numbers in one iteration, this means the list has already been sorted and we do not need to proceed anymore

OBSTACLES

As for the obstacles we met during this lab, firstly, it was the first time ever that we actually programmed in assembly language. And ARM assembly was the brand new language for both of us. Thus, we were not really familiar with the programing format and the name of the operations. We checked the *ARM Instruction Set Quick Reference Card* frequently during the lab. Also, since we had to jump a lot in the code using labels, it was sometimes not trivial that what exactly the instructions were supposed to do. Actually, when we were doing part 3, we forgot to branching to a label, which led to wrong result. And we spent a very long time to find this issue. But this also made us familiar with the debugging system in the lab.

IMPROVEMENT

As for the improvement, the first two parts are pretty trivial and there is no really a so called better way to solve the problem. But for centering the list, which is part 3, if the time allows we could manually create a division method to make it work for lists with general length. Also, the code becomes hard to read quickly as the complexity increases. After learnt the concept of subroutine, we could have use subroutines to modularize the code to make it both more readable and maintainable. As for the sort part, we used one of the simplest sorting algorithms, which was somehow computational expensive. By using a more sophisticated sorting algorithm like Merge Sort or Quick sort, we could improve the computation time significantly when the list is long. However, the complexity of the program will also increase in order to achieve this time efficiency.