ECSE 324 G02 Lab 1 Report

Adrian Wang 260769387
Chen Cheng 260775674

Part 1 Largest Integer Program

At the beginning of the lab, a snippet of code is given to students for familiarization of the syntax structure, and basic operations in the assembly language. The code mainly consists of a loop that looks for the largest number in an array stored in the memory. At the beginning of the program, two registers R4 and R3 are used as pointers that point to the result location and the array correspondingly for the incrementation in the loop. One register R2 is used to hold the number of elements to stop the loop and another, R0 is used to store the array elements that are incremented. Finally, the largest number is updated in register R0 and stored into register R4. The program does not have any problem since it is provided, but it helps students to understand how the assembly language runs and dynamically changes in each step. A summary of the resources used is as follows:

- Pointer register: R3, R4
- General register: R0, R1, R2
- Lines of code: 25
- Commands used: LDR, ADD, SUBS, BEQ, CMP, BGE, MOV, B, STR

Part 2 Standard Deviation Program

In the program challenge part, a standard deviation is firstly asked to be written in the hardware-friendly way of (X_max-X_min)/4. In this way, based on the code provided in part 1, only the selection of the minimum number and a simple calculation are needed to be implemented. We choose to simply add a second loop with a doubled number of registers and pointers to choose the minimum number in the array. After the two numbers are chosen, their difference is calculated and then divided by 4 using logical shifting right for four bits.

In the beginning, the registers were used for both loops and it caused some problems, after a quick analysis, it was because the new part of the code changed the address in the memories and then the pointers could not point to the correct location. To solve this, separate registers are applied as a straight-forward way which simply adds another loop. Therefore, the program still has great potential for improvement after meeting the objectives. With more carefully designed algorithms in the loops and functions, the selection of the extreme number could be done within one loop which can make the program shorter in length and thus less time it takes to run.

- Pointer register: R3, R4, R5, R7
- General register: R0, R1, R2, R6, R8, R9, R10
- Lines of code: 25
- Commands used: LDR, ADD, SUBS, BEQ, BLT, CMP, BGE, MOV, B, STR

Part 3 Centering Program

In this part, the same array is used as the previous parts to simulate the input. The input is needed to be centered around zero by shifting the entire input by its average value. To fulfill this goal, the program first calculates the sum of the entire array by traversing through the array and summing the elements. This is done by the first loop and pointers used as before, and the sum is stored in one register. After that, the average is calculated by a simple division and the program enters the second loop to traverse through the array again to subtract the average from each of the elements. During the traversal, the elements are updated in the original address after the subtraction. This program is lightweight and straight forward and was finished quickly and smoothly.

- Pointer register: R0, R1
- General register: R2, R3, R4, R5
- Lines of code: 25
- Commands used: LDR, ADD, SUBS, BEQ, CMP, BGE, MOV, B, STR, ADR

Part 4 Sorting Program

As the last part of the lab, a sorting program using the bubble sort algorithm is implemented with the assembly language. The program is restructured with two loops and one branch as an equivalent of the pseudocode given in the lab description. At the beginning of the program, loop 2 functions as the while loop and sets R10 as the indicator of the sorting status with boolean encoding. Then, the code continues to loop 1 to iterate through the array to compare, swap, set the indicator, and go back to loop 2. Every time the loop 2 runs, the indicator is checked and R10 = 1 means the array is sorted and the code ends. If R10 = 0, then loop 1 will reset and run again until the array is fully sorted. By the nature of the bubble sort, the times that the loops are run may vary from 1 to n^2, which is not efficient. Therefore, other algorithms like quicksort could be implemented and compare the cost between them. If the bubble sort is indeed the easiest to implement, CPSR flags could be considered to replace the “register flag” R10 to improve the program.