G21_Lab1_Report

Lab 1 consists multiple tasks, focusing on finding the maximum number of an array, finding the standard deviation of an array, centering an array and sorting an array respectively.

Part 1: Largest integer program

Part 1 is the necessary process to realize the rest of this lab. The code has given us a method to find the maximum number in a list which has a fix number of element. A loop is created to iterate through the array. Every time the loop gets executed, it compares R0 and R1 and if R0 is greater than R1, goes back to the loop. Otherwise, change R0's value to R1. Basically, we want to keep R0 the largest integer in the array. A counter R2 is built to check whether every number in this array is compared with each other. Once one comparison is finished, R2 decreases by one and when the counter reaches zero, finally store the result to the memory location.

Part 2: Standard deviation program

The process to get the approximate standard deviation according to the range rule is nothing more than find the difference between maximum and minimum and divide by 4. Based on part 1, finding the minimum is pretty much the same logic:

- Pointer iterates to the next number
- Update the current minimum if the number is smaller than current min
- Branch back to the start of the loop

After this process, the minimum can be found. A SUB command can get the difference of max and min. Copy the difference to another register, then use a LSR command by 2 to divide the difference by 2^2, which is 4, and finally store the answer to the RESULT memory location.

Centering program

This part is asked to find the average of an array, subtract it from each element of the array, and then store them to the same memory slot they were stored; with the assumption that only lengths that are powers of 2 can be passed to the program. The process is:

- Find the sum of the whole array
- Get the average by dividing the sum by the number of elements in the array
- Subtract the average from each element and store them in place

Although the number of elements **n** is assumed to be the powers of 2, there is still no way to directly finding the square root but write a separate loop. First, LSR the number of elements by 1 each time, this is equals to divide n by 2. And add the counter(R5) by 1.

The loop stops when the digits are all shifted and left with 0. The value of R5 is desired in this part.

After this, we update R2's value to avoid using to many new registers.
Next, we find summation of the array using a loop. R2, as a counter, decreases by one every time we add a number to the previous number. When the counter reaches zero, end loop. When the summation is done, R0 should hold the sum of all the numbers in this array at this time.
The average number is simply found by dividing sum by the size of the array which is done by LSR R0 by R5.
The last step is using a loop, to subtract the average from the first element to the last element, and store each to the memory location instantly after subtraction. The centered array is now stored in place.

Sorting program

This part requires an algorithm to bubble sort the array in ascending order and store the array in place as well. The logic is given by the following:

- Set two pointers pointing the first and the second elements in the list
- Compare the two pointers
- If needed to swap, exchange the two pointers by using an intermediate register (without changing the pointer values), store the two elements in memory, add both pointers by 1
- If no need to swap, branch to directly add the pointers by 1, reset counter, then return to loop
- The loop runs several times, which equals to the number in the array. After this many times of iteration, the array must be in a sorted ascending order.

We use a nested loop to realize this question. The outer loop counter R1 is the number of the element in the array and the inner loop counter R5 indicates for each number, there is N-1 comparisons (N is the number of the elements). We first set a counter (R5) to start the inner loop, load the first two elements in the array to two registers and compare them. If the first element is larger, then there is no need to swap and begin a new comparison (counter -1). If the first element is smaller, we swap two element's positions. In this way, each element gets to compare with each other. When R5 reaches zero, it means that the inner loop ends and branch to "ENDINNER" to subtract 1 from R1 and reset the inner loop counter and then start a new iteration. If R1 reaches zero, it means that the array is sorted and branch to end.
The flaw of our code is that the loop will always be executed N times even if the array is already sorted.