ECSE 324
LAB 1

Sameen Mahtab

Tianyi Zou

Part 1

The code begins with "_start". Over here the registers R0, R2, R3, R4 are loaded with different values. Register R4 points to the address of RESULT. Register R2 holds the number of elements in the list. Register R3 is initially used to point towards the address of the list of numbers and register R0 is loaded with the first number in the list. R0 will be used to store the maximum number; initially that is the first number in the list.

After doing that we proceed with "LOOP". Over here we first initialize the counter. SUBS is used to decrease the value stored in R2 by one after each iteration of the loop. The we execute the branch instruction BEQ which basically branches to done if the value if greater than or equal to 0. We branch is done to end the loop. After that, the next number in the list is loaded to register R3 and then that is compared to the number stored in R0 using the CMP instruction. The instruction BGE is used to see if the value stored in R3 is greater than that stored in R0. If it is bigger, then the MOV instruction is used to store the value stored in R3 in R0. R0 now contains the new maximum. After than we branch back to LOOP and this keeps in iterating until we have compared R0 to all the numbers in the list and have stored the largest number in R0.

After the loop counter has reached 0, DONE is executed to store the value in R0 into RESULT. And then the program is ended at END.

Stddev

In this part we had to find the standard deviation between two numbers. Since the original formula for standard deviation is very complicated and would be lengthy to implement in assembly and simplified version was provided for us to implement. The standard deviation could be found by simply adding the minimum and maximum numbers in a list of numbers and then dividing them by 4.

To do this out code took a lot of inspiration form part 1. We re-used the code in part 1 to find the maximum value and added a small modification to also find the minimum value.In this code R0 was used to hold the current minimum, and register R1 was used to hold the successive numbers to compare to R0. In this code we also added another register R6 to hold the maximum number in the list. During an iteration a number is first compared to R0, if the number is greater than the value stored in R0, we branch to MIN. Inside MIN, we compare to see if R1 is less than R6. If the number is not less than R6, then R6 is updated with that number. Is the number is smaller then we are branched back to LOOP.

This keeps on iteration until we find the minimum and maximum numbers. After we find the two numbers the values stored in R0 and R6 are added using the ADD instruction. And then they are divided by 4 by doing a logical shift using the instruction LSR #2.

And then finally then code end in the same way as in part 1 when it reaches END.

Center

This program consists of three parts, calculating the average of signals , subtracting each average by each signal and storing signals into the same address as the original signal.

We first use R0 pointing to the location which contains the number of the signals. Then we load the content in the address to R4, now R4 has the value of 8. We let R1 points to the address which is the address of the first digits of the list, then we load contents of R1to R2. We set R7 as the counter for getting the sum of the list of the digits and R9 as the counter for putting the difference between the signal and the average back to the address of the original signal. We set R4 + 1 as the value of R7 and R9 because we let R7 and R9 subtract 1 before we do the addition or subtraction. If we just use the number of the signal to compute, the number of times we go through the loop is going to be 7 not 8 so we add 1 to it to go through 8 times computation.

For the addition part, we just subtract our counter by 1 every time we add one of the digit to R3 and let R2 to be the next signal we need to add to R3 which is the sum.

We use the logical shift left(LSR) to accomplish the division.

At last we need to put the differences back to the address of the subtrahends (the signals). We subtract R9 by 1 every time we begin the store and use R8 as the medium to store the value. Every time we finish a store operation we add R0 by 4 which is just the same way as we used R0 to point each signal's address as before.

Sort

For this program, first register R10 is used to points towards the number N, register R9 is used to point towards FALSE and register R1 is used to store the value in FALSE. Register R10 points towards N and register R2 stores the value stored in N. FALSE holds a value 0 and it is used as a boolean to say if the array is sorted. Register R12 is used to point to NUMBERS and register R3 is used to hold the actual values stored in NUMBERS.

If our first loop LOOP, we check to see if the boolean FALSE is greater than 0. If it is we branch to DONE. if not then we change it to 1 and branch to LOOP2.

In out second loop LOOP2, we start of by initiating a counter which decreases by 1 after each iteration of LOOP2. Then we load two numbers, the first number in the list and the next number in the list to registers R4 and R5 respectively, then we update the pointer R3 to point to the next number. Then we compare the two numbers stored in R4 and R5. If the value stored in R5 is less than the value stored in R4, we branch back to LOOP2 and start another iteration this time comparing the first number to the new number pointed to by R3. When comparing the two numbers stored in R4 and R5, if the value stored in R5 is less than the value stored in R4, we are taken to SWAP.

Inside SWAP, we store the value stored in R5 in the previous address of R3 and store R4 in the current location of R3. This puts the number in R5 before the number in R4 in the list. SWAP also changes the boolean from 0 to 1.

In this way the program compares two numbers at a time and sorts them accordingly. When the array is sorted the boolean is changed to 1.

The program ends when SWAP does not have to be entered anymore. The boolean is not changes back to 0 and LOOP 1 branches to END.

Finally improvements that can be made to the code include using post indexing instead of writing an extra line to increment a pointer. MOV could be used implement the boolean instead of using memory.