ECSE 324 Lab Report 1

Marie-Lynn Mansour - 260770547
Haluk Calin - 260790895

Part 1: Finding the Maximum Number from a List of N Numbers

The first part of the lab was to find the maximum number from a list of N numbers. In order to complete this part, we used the code that was provided to us in the lab description on MyCourses. This code was re-used and refactored in various applications throughout the lab as the loop structure was a great template to base our code off for the other parts. This part of the code was crucial as it helped us gain an understanding of how the assembly language works, including its syntax and style.

The algorithm of actually finding the maximum number from a list of N numbers was a process of looping through the numbers in the list that were stored in memory. The first element in the list is stored in the register that acts as the holder of the maximum number of the list, regardless of whether this value is actually the maximum value. Another register holds each following number in the list as the pointer loops.

The program is constantly checking whether the element stored in the maximum number register is larger than the value in the register that holds the following number in the list. If the comparison is false, the number in the list register is moved to the register that holds the maximum number, since it is now the new maximum. If the comparison is true, it simply moves onto the next number in the list. After it has gone through the entire list, it will move on to the DONE loop branch of the assembly code and store the maximum number of the list in permanent memory.

One possible improvement to this code would be to implement a method that detects whether the array is empty prior to attempting to find the largest number in the list in order to avoid errors. Implementing this method would include using the CMP function to compare N to 0. If N is smaller or equal to 0, the assembly code could directly branch to the end of the code.

Part 2: Finding the Minimum Number from a List of N Numbers

The second part of the code was to find the minimum number from a list of N numbers. The completion of this part required slightly modifying the code for finding the maximum number from Part 1.

The first change we made was loading the register R5, that holds the first number in the list which will eventually be the minimum value of the list, into the address R3.

```
LDR R5, [R3]
```

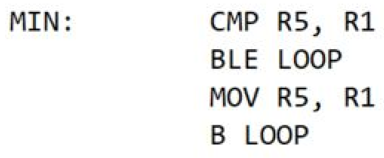
figure 1: loading R5 into the address R3

Another change we made was slightly modifying the branching condition in the loop. We changed the BGE (branch if greater than or equal to 0) LOOP from the Part 1 maximum number code to BGE MIN. If the value being compared to R0, which is R1 in this case, is smaller, we branch to MIN.

```
BGE MIN
```

figure 2: branch to MIN if greater than or equal to 0

We also added MIN to calculate the minimum value of the list when it is branched to in LOOP. This compares R5, which is the first number of the list, and R1, which holds the next number in the list. If R5 is smaller than R1, we go back to the LOOP to iterate over another element. If R1 is smaller than R5, it becomes the new global minimum and it is stored in the address R5. We then branch back to the LOOP and continue iterating through the elements in the array.

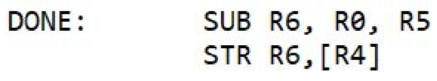


```
MIN:        CMP R5, R1
            BLE LOOP
            MOV R5, R1
            B LOOP
```

figure 3: assembly code to find the minimum value of the list

Part 3: Find the Range of the Set of Data

The third part of the assembly code was to compute the range of a set of data. This part was completed by combining the functionalities of Part 1 and Part 2 with a minor addition. The difference between the maximum and minimum (Max - Min) was computed by using the SUB instruction, as illustrated in the figure below:



```
DONE:       SUB R6, R0, R5
            STR R6,[R4]
```

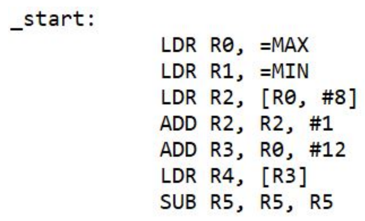
figure 4: computing the range of the maximum and minimum values of the list

DONE simply calculates the range of the set of data once the maximum and minimums have been determined. R5 (the minimum) is subtracted from R0 (the maximum) and put in R6. R6 is then stored in the address of R4.

One improvement could have been to remove the B LOOP at the end of LOOP as we included one in the MIN part of the code. By removing the loop, we would've reduced the running time of our algorithm.

Part 4: Finding the Maximum and Minimum Values of an Algebraic Expression

In order to complete this task, we split it into 3 parts. At the very beginning, we assigned multiple registers for some key elements that were reused throughout the code. R0 was loaded to hold the address of the maximum value of the algebraic expression. R1 was loaded to hold the minimum value of the algebraic expression. R2 was loaded to hold the number of elements in the list. R3 was loaded to point to the first number in the list while R4 holds the actual value of the first number. Finally, the R5 register was cleared in order to hold the sum of all the numbers for the first part of the code.

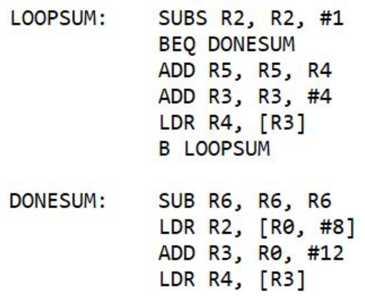


```
_start:
            LDR R0, =MAX
            LDR R1, =MIN
            LDR R2, [R0, #8]
            ADD R2, R2, #1
            ADD R3, R0, #12
            LDR R4, [R3]
            SUB R5, R5, R5
```

figure 5: initializing the registers

4.1: Finding the Sum of the List

The first step was to sum up all the numbers in the list. We created a loop, LOOPSUM, that computes this sum. It starts by decrementing the register that counts the number of iterations, ending the loop if/when the counter reaches 0. R4, which holds the actual value of the first number of the list, is added to the sum in R5, which holds the sum of the numbers. R4 is then loaded into R3. The loop keeps iterating through each element of the list until all the numbers are added to R5. The R6 register, which is used later to hold the maximum/minimum values, is cleared, R2 is reinitialized for the next loop, and R4 is loaded into memory register R3.



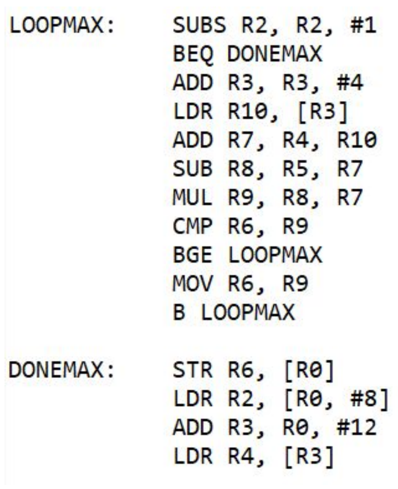
```
LOOPSUM:    SUBS R2, R2, #1
            BEQ DONESUM
            ADD R5, R5, R4
            ADD R3, R3, #4
            LDR R4, [R3]
            B LOOPSUM

DONESUM:    SUB R6, R6, R6
            LDR R2, [R0, #8]
            ADD R3, R0, #12
            LDR R4, [R3]
```

figure 6: loop to compute the sum of all the numbers in the list

4.2: Finding the Maximum of an Algebraic Expression

The second step was to find the maximum of the algebraic expression, which was also done using a loop (LOOPMAX). The counter is decremented, ending the loop if/when it reaches 0, and R3, which points to the first number in the list, is added, shifted by 4 bytes. Then, the value of the next number is loaded from R3 into R10. R4, which is the actual value of the first number in the list, and R10 are added together and placed in R7. In order to find the sum of the other 2 numbers in the list, we compute S-X (as discussed in the Lab 1 document on MyCourses). The result of the addition calculated earlier (X) is subtracted from the sum computed previously (R5) and placed in R8. Then, the first sum (X) and the second sum (S-X) are multiplied together and placed in R9. R9 is then compared to R6, which holds the current maximum. Using BGE (branch is greater than or equal to 0), if R9 is larger than R6, we update the current max value to R9. If R9 is smaller than R6, we branch back to the loop and start again. Finally, the result R6 is stored in the R0 register, which holds the MAX of the algebraic expression. The counter R2 is reinitialized for the next loop.



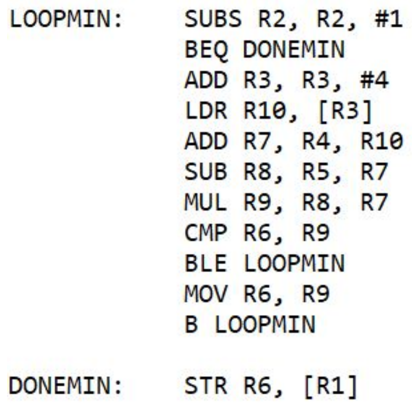
```
LOOPMAX:    SUBS R2, R2, #1
            BEQ DONEMAX
            ADD R3, R3, #4
            LDR R10, [R3]
            ADD R7, R4, R10
            SUB R8, R5, R7
            MUL R9, R8, R7
            CMP R6, R9
            BGE LOOPMAX
            MOV R6, R9
            B LOOPMAX

DONEMAX:    STR R6, [R0]
            LDR R2, [R0, #8]
            ADD R3, R0, #12
            LDR R4, [R3]
```

Figure 7: loop to compute the maximum of the algebraic expression

<u>4.2: Finding the Minimum of an Algebraic Expression</u>

The final step was to find the minimum of the algebraic expression, which was done using a loop (LOOPMIN). The methodology was exactly the same as finding the maximum of the algebraic expression. The difference is that instead of using BGE (branch if greater than or equal to 0), we use BLE (branch if less than or equal to 0). If R9 is less than R6, it becomes the new current minimum. At the end of the loop, the minimum R6 is stored in the memory address R1.



```
LOOPMIN:    SUBS R2, R2, #1
            BEQ DONEMIN
            ADD R3, R3, #4
            LDR R10, [R3]
            ADD R7, R4, R10
            SUB R8, R5, R7
            MUL R9, R8, R7
            CMP R6, R9
            BLE LOOPMIN
            MOV R6, R9
            B LOOPMIN

DONEMIN:    STR R6, [R1]
```

Figure 8: loop to compute the minimum of the algebraic expression