ECSE 324 , Group 17

Lab 1 Report

Part1: Largest integer programming

Here the code was provided along with the lab instruction, all we need to do is to understand the given code and then elaborate from the code to do the following parts of this lab.

Approach taken

```
_start:
            LDR R4, =RESULT     //R4 points to the result location R4=0x38
            LDR R2, [R4, #4]    //R2 holds the number of elements in the list R4+4 = 0x3c; R2=7
            ADD R3, R4, #8      //R3 points to the first number; R3= R4+8=0x38 + 8 = 0x40
            LDR R0, [R3]        //R0 holds the first number in the list; R0 = 4

LOOP:       SUBS R2, R2, #1     // decrement the loop counter
            BEQ DONE            // end loop if counter has reached 0
            ADD R3, R3, #4      // R3 points to next number in the list
            LDR R1, [R3]        // R1 holds the next number in the list
            CMP R0, R1          // check if R0 is g/equal than R1( greater than the maximum)
            BGE LOOP            // if yes, branch back to the loop
            MOV R0, R1          // if no, update the current max
            B LOOP              // branch back to the loop
```

Figure 1: Screenshot of part of the code given by the instructor in the lab description for part1.s

To better understand the code, we asked the TA during the lab section and added our own comments (see *Figure 1*), since we were asked to calculate the range, it is crucial to understand the logic and do further implementation. One efficient way used here was writing down the memory locations and values stored in the register after compiling and running the code.

Challenges faced

We were confused about the add #4 , #8 operations in LDR and ADD at the beginning (which was covered in the lecture a couple of days later), which turned out acted as a “pointer” and helped to move to the next location.

Possible improvements made/could have made

None

Part 2: Finding a range of a set of data

Here we were given a set of data in which we had to find the difference between the max and min values of the set.

Approach taken

```
LOOP:       SUBS R2, R2, #1     // decrement the loop counter
            BEQ DONE            // end loop if counter has reached 0
            ADD R3, R3, #4      // R3 points to next number in the list
            LDR R1, [R3]        // R1 holds the next number in the list
            CMP R0, R1          // check if R1 greater than current Max
            BGE MIN             // if no, branch to MINIMUM
            MOV R0, R1          // if yes, update the current max (content of R1 moved into R0)

MIN:        CMP R5, R1          //check if R1 less than current Min
            BLE LOOP            //if NO, branch back to LOOP
            MOV R5, R1          // if YES, update the current Min (content of R1 moved into R5)
            B LOOP              // Branch back to LOOP
```

Figure 2: Screenshot of the loop code written by group members for rangecal.s

In our approach, initially, R0 and R5 store the value of the first number of the set, this number is assumed to be the max value and the min value. R1 stores the next number in the set. The loop shown above (see *Figure 2*) compares each number X with the max value, if X > max, then X is placed inside R0. Otherwise, we check whether X is < min. If so, X is placed inside R5.

```
DONE:       SUB R6, R0, R5      // Calculate Max-Min = Range, put Range into R6
```

Figure 3: Screenshot of the part of the code (calculate range from max and min) written by group members for rangecal.s

Once we have iterated through all of the numbers within the set, we find the range by subtracting the min value from the max value (see *Figure 3*).

Challenges faced

No challenges faced. The problem was quite simple to do as we built upon the given code in the instructions for Lab 1.

Possible improvements made/could have made

We believe that we made the code as small and efficient as possible. Another implementation could have been to have multiple loops in which one loop find outs the max value and the other loop finds the min value – This approach would have resulted in the assembly code being longer.

Part 3: Maximum and minimum values of an algebraic expression

Here we had to calculate max and min of the arithmetic expression (x1 + x2)*(y1 + y2) where xi, yi ∈ Z (set of integers) where i ∈ {1,2}.

Approach taken

Suppose S is the sum of all the (n+m) numbers in expression and X is the sum of the n numbers on the left side of the expression. The sum of the m numbers on the right side of the expression would then be (S-X).

```
SUM:
            LDR R8, [R4]        // R8 stores value of first num
            ADD R7, R7, R8      // First num added to R7 = R7 + num
            ADD R4, R4, #4      // R4 points to address of second num
            SUBS R6, R6, #1     // Counter decrement
            BEQ RESET           // Branch to RESET if R6-R6=0 or counter = 0
            B SUM               // Branch to SUM
```

Figure 4: Screenshot of the code for calculating the sum written by group members for maxmin.s

So firstly, we calculated the sum S as shown in the picture above (see *Figure 4*). We iterated through the set of numbers using a counter, during each iteration, we added the number to the sum – R7 stores the sum (R7 initially stored the value 0). We then reset the values of R4(which points to the first number of the list) and the counter (R6) as we have limited registers and so that we can calculate the max and min of the arithmetic expression.

```
LOOP:
            SUBS R6, R6, #1     // counter decrement
            BEQ DONE            // Exit if counter = 0
            LDR R11, [R5]       // Second number loaded into R11 as R5 points to second num
            ADD R5, R5, #4      // R5 points to next number (initially from 2nd num to 3rd num)
            ADD R12, R8, R11    // First num + next num loaded into R12 (X = sum of numbers on LHS)
            SUB R2, R7, R12     // S - X
            MUL R3, R12, R2     // X*(S-X)
            CMP R9, R3          // Compare R9(current MAX value) and R3[X*(S-X)]
            BLE CHANGEMAX       // If current MAX lower or equal to R3[X*(S-X)], then change MAX
            B MINCHECK          // Or else, check for MIN
```

Figure 5: Screenshot of the looping code written by group members for calculating X of maxmin.s

The loop above(see *Figure 5*) was the key part of the solution to the programming challenge. It basically calculates X, and then uses X and S to calculate the sum of the m numbers on the right side of the expression as stated above. So, allow me to explain how we calculated X.

1. We stored the first number of the set in R8. We then added the next number in the set to the first number, this gave us X for one of the many combinations of the integers.
2. Using X, we calculated X*(S-X).
3. We then compared X*(S-X) with the initial values of max and min stored in R0 and R1.
4. Step 1-3 repeated through iteration by using a counter.

<u>Challenges faced</u>

The biggest challenge we faced was figuring out how to calculate the sum of all possible combinations of n numbers and remaining m numbers. Not only that, reusing the limited number of registers was also somewhat of a confusing procedure.

<u>Possible improvements made/could have made</u>

Our initial thought of implementation was to first calculate all the possible values of the expression (x1 + x2)*(y1 + y2). Then compare each of the values to find the max and min values. However, this expression would have resulted in more than one loop and so would not have been the most efficient implementation. Therefore, we believe that our current implementation (using one loop) is an improvement to our initial idea of implementation