EE306 - Microprocessors

Laboratory Exercise 3
Subroutines

March 17, 2020

Erdal Sidal Dogan
#041702023

Alp Gokcek
#041701014

1 Sigma Sum

In this experiment we wrote an assembly program that finds the mean of given set of numbers, length of the number sequence is defined as N. Listing 1 below calculates the result of the following expression $\sum_{i=0}^{N} i$

First step is to store the value that we are going to sum up to (N) in the R0 register. Then, we load 0 to the R1. It will be used as temporary register during the summation process, later on we will move the value of the R1 to the R0 since it has been specifically requested to store the result in R0.

Subroutine `FINDSUM` adds current value of R1 to R0, then decreases the value of the R0 by 1. Finally, checks if the R0 is equal to 0. If no; repeats the same sequence. If yes; it's job is done. After `FINDSUM` finishes its operation, we move the result from R1 to R0 and end the program.

Listing 1: Assembly Code for Sigma sum & calculating average

```
.include "address_map_arm.s"
.text
.global _start
_start:
        LDR R0, N // load the data word into R0
        MOV R1, #0 // temp register
        BL FINDSUM
        LSR R1, R1, #5 // average of 32 numbers
        MOV R0, R1 // move result from R1 to R0
        B END
FINDSUM:
        ADD R1, R1, R0
        SUBS R0, R0, #1 // count down
        BXEQ LR // branch if 0
        B FINDSUM
END: B END
N: .word 0x9
.end
```

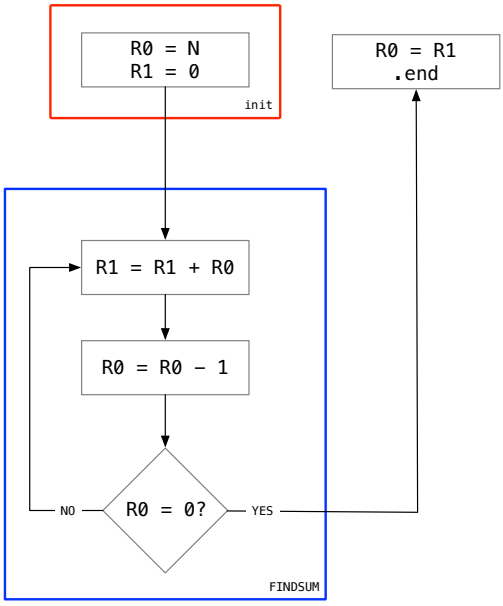

Figure 1: Flowchart of the Listing 1

2 Part IV - Bubble Sort

In this experiment we wrote an assembly program that sorts the given numbers using bubble sort algorithm. First step is to create a pointer to the memory location that will be sorted and after that load the number of values that will be sorted which placed as the first element of that memory location. Then, we create a pointer to the first element of the list that will be sorted and then we started the sorting subroutine.
In the sorting subroutine's initialization part, we save the link register's value for one time then save the content of R0 and R1 and update the value of flag register as 0. Flag register is used for checking whether any item is sorted on that traverse. After initialization, algorithm checks whether we traversed the whole list or not. If we traversed the whole list, then we branch to checking part. If not, algorithm loads 2 items to be compared from memory and check whether first element is greater or equal to second element. If yes, continue to traverse, else, start swapping subroutine. If no, continue to traverse.
In swapping subroutine, we store the first element in second elements memory location and store second element in first elements memory location. Then, we update the status flag and return to sorting subroutine.
On checking part, we restore the content of R0 and R1 and decrement R1 which holds the number of elements. After that, we check whether we swapped anything on that traverse. If yes, branch to sorting subroutine, else, restore link register and update R0 to hold the first elements position on memory.

Listing 2: Assembly code for bubble sort algorithm

```
.text
.global start
start: LDR R8, =LIST //holds the address of the list
       ADD R0, R8,#4 // points to first element
       LDR R1, [R8] // number of elements
       MUL R4, R1, #4 // for calculating the latest address
       BL INIT_SORT
       SUB R0, R0, R4 //returing to the first element
       B END
INIT_SORT:PUSH {LR}
SORT: PUSH {R0, R1} // save number of elements and pointer
       MOV R12, #0 // flag
BUBBLE_SORT:SUBS R1, R1,#1 // decrement number of elements
       BEQ CHECK //branch if number of elements is 0
       LDR R2, [R0] //element 1
       LDR R3, [R0, #4]! //element 2
       CMP R2, R3 //compare two elements and swap them if needed
       BGE BUBBLE_SORT // if r2 is greater than or equal to r3, continue to traverse
       BL SWAP // if element 1 is less than element 2 then swap
       B BUBBLE_SORT //continue to traverse after sorting
SWAP: STR R2, [R0]
       STR R3, [R0, #-4]
       MOV R12, #1 // update flag
       BX LR
CHECK: POP {R0, R1} // restore number of elements and pointer
       SUB R1, R1,#1 // decrement number of elements
       CMP R12, #1 // checking if anything sorted
       POPNE {LR} // restore link register if list is sorted
       BXNE LR // return to link register if sorted
       B SORT //continue to sort if not sorted
END: B END
LIST: .word 10,0,1,2,3,4,5,6,7,8,9
.end
```

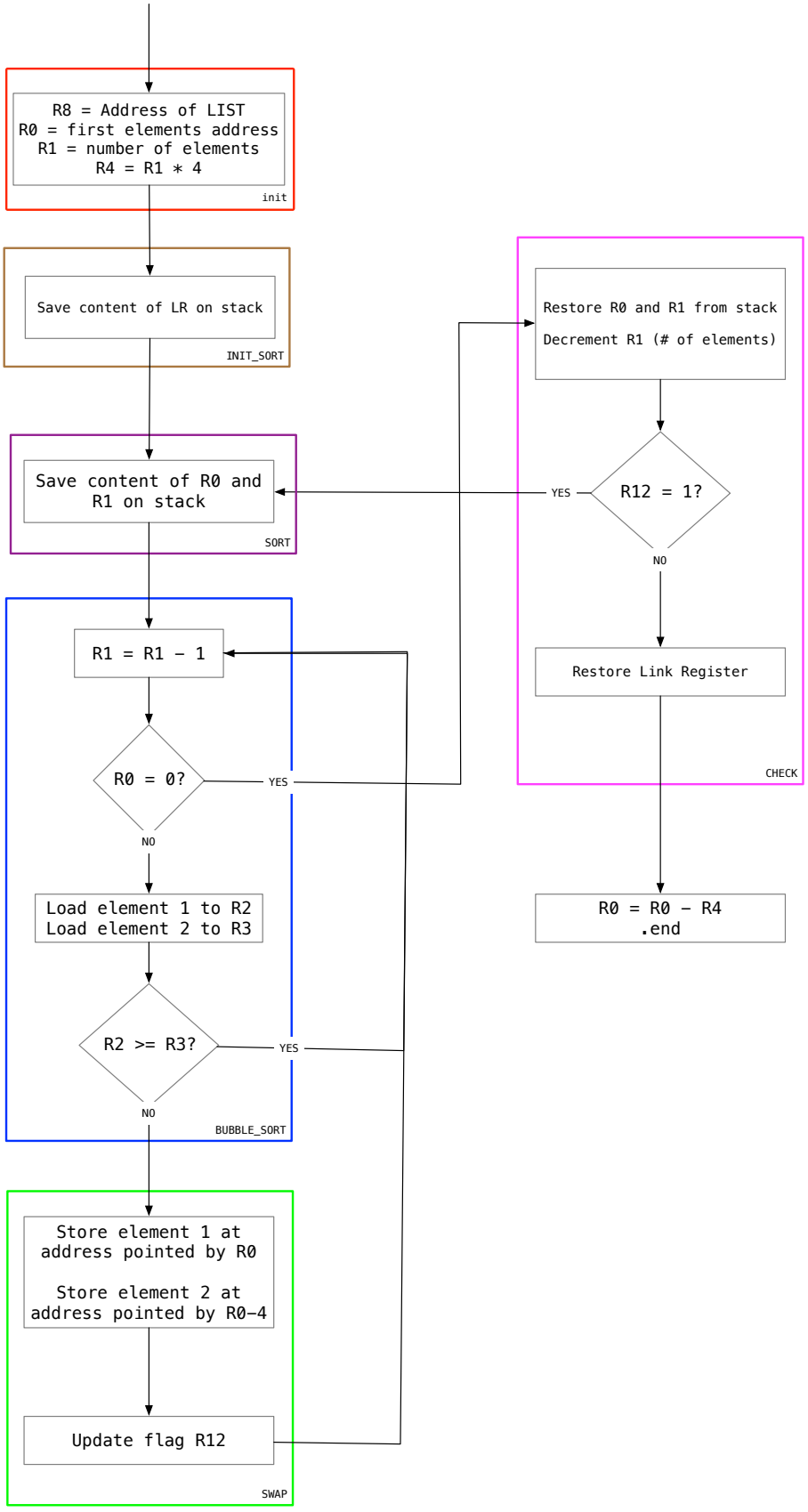

Figure 2: Flowchart of the Listing 2