ECE 212
LAB-3 REPORT

Introduction to Subroutines

Name: Misbah Ahmed Nauman
CCID: misbahah
Student ID: 1830574

Partner: Muhammad Ayaan Hafeez
CCID: mhafeez1
Student ID: 1805075

Section: H25
Lab Instructor: Manuchehr Fathi
Lab Number: 3
Lab Date: 03.11.2025

Table of Contents

1. Introduction

This lab focuses on using subroutines in ARM Assembly to modularize and structure code effectively. The central program is a **bubble sort algorithm**, which has been broken down into three key subroutines:

- WelcomePrompt handles user input and validation.
- Sort executes the bubble sort.
- Display outputs the sorted results.

The lab emphasizes **stack management**, **register preservation (callee convention)**, and **parameter passing**, aligning with structured assembly programming practices.

2. Design

WelcomePrompt Subroutine

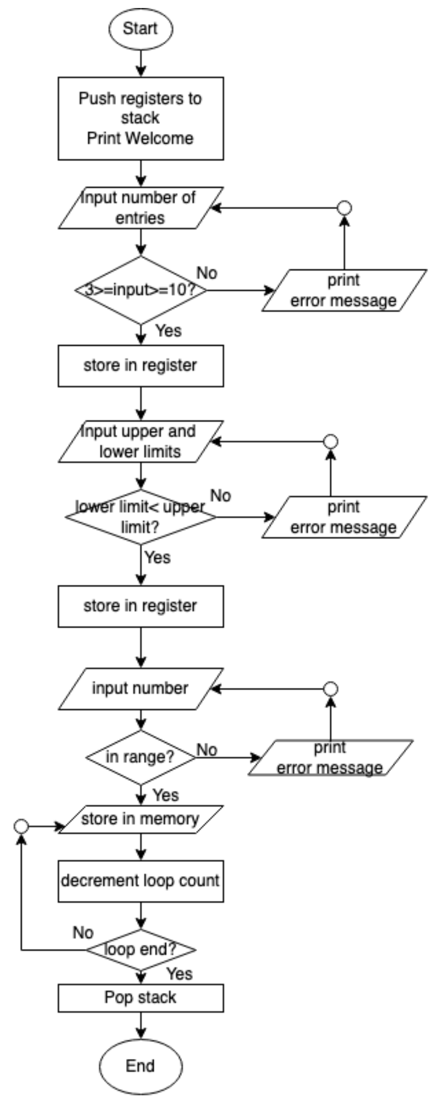
The WelcomePrompt subroutine prompts the user to input:

- Number of entries to be sorted (between 3–10).
- Lower and upper bounds for acceptable input values.
- The actual numbers to be sorted, with validation for range and count.

Errors such as out-of-range limits, equal upper/lower bounds, or invalid entries are re-prompted with appropriate messages. Values are stored at the address pointed to by R0.

Key Features:

- Stack: Inputs/outputs use the stack to maintain modularity.
- Validation: Includes bounds checking and re-prompt logic.
- Subroutines used: printf, getstring, cr, value.



Flowchart for part A

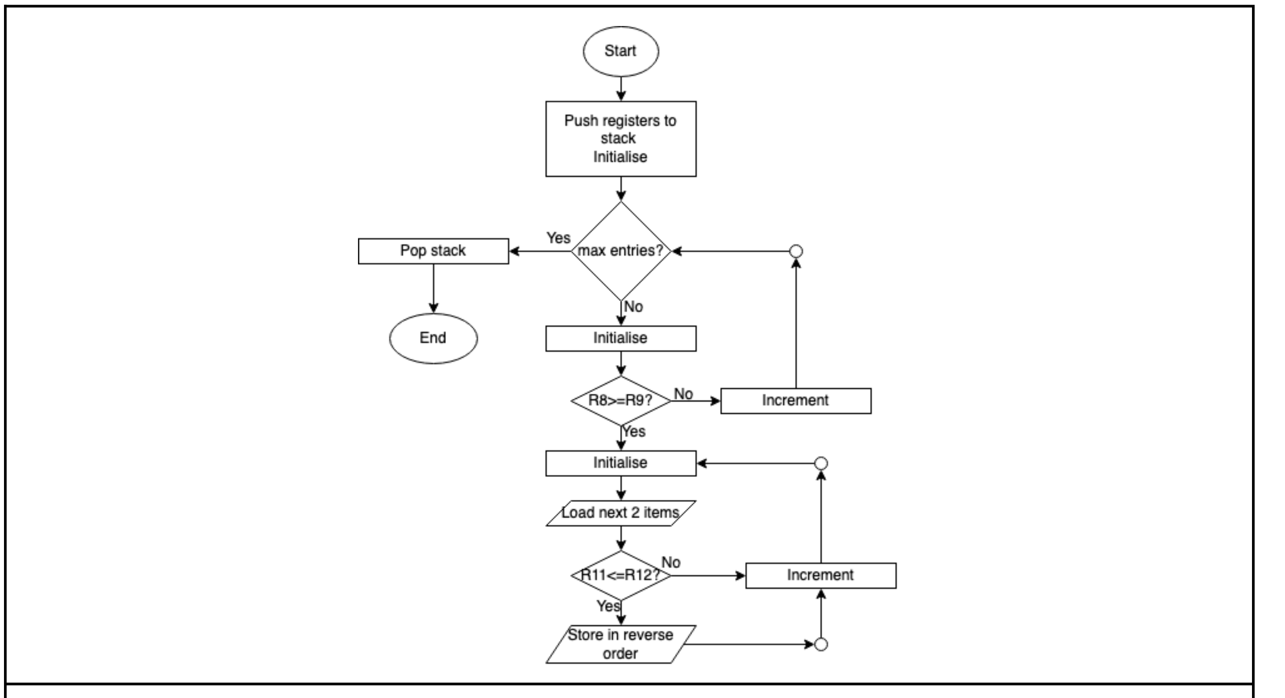
Sort Subroutine

The Sort subroutine implements the bubble sort algorithm. It:

- Loads the number of elements from the stack.
- Iteratively compares and swaps adjacent elements.
- Uses R0 for the base address of the numbers.

Key Points:

- All temporary registers are saved and restored (callee-save).
- Stack remains unmodified (entries preserved).
- Sorting happens in-place.



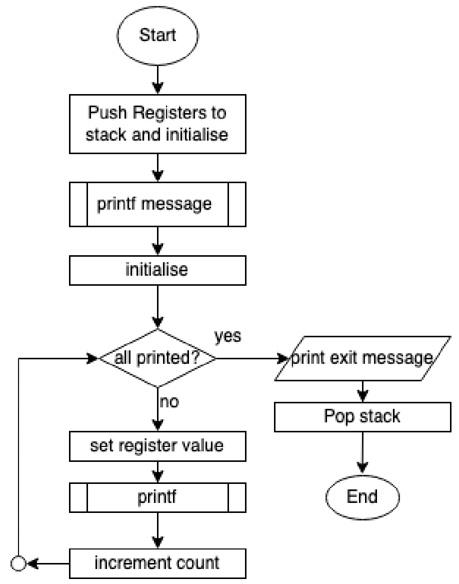
Flowchart for part B

Display Subroutine

The Display subroutine presents the sorted array:

- First prints the number of elements.
- Displays "Sorted from smallest to biggest..." and each value using value and cr.
- Ends with "Program ended".

Parameters are passed through the stack (number of entries and memory address).



Flowchart for part C

3. Testing Strategy and Results

3.1 Testing Approach:

Part A – WelcomePrompt Subroutine

For Part A, our testing approach involved simulating various user inputs via getstring and observing how the program handled validation and storage. We checked for correct stack updates, valid input range handling, and memory allocation. The tests included:

- **Valid Input Case:** Entering a number of entries within the range (e.g., 5), then specifying valid lower and upper limits (e.g., 10 and 50), and inputting values within this range.
- **Entry Count Bounds:** Testing edge cases such as inputting fewer than 3 entries (e.g., 2) or more than 10 (e.g., 11) to trigger the appropriate error messages.
- **Range Rejection:** Inputting a lower limit greater than or equal to the upper limit to ensure the prompt resets correctly with the message "Error. Please enter the lower and upper limit again."

- **Out-of-Range Value Check:** Entering values outside the specified range to test whether the error message "Invalid!!! Number entered is not within the range" appears.

- **Final Entry Prompt:** Verifying that the last number displays the "Please enter the last number" string as expected.

This approach ensures all validations work as expected and values are properly stored in memory starting at the address pointed to by R0.

Part B – Sort Subroutine

To test Part B, we focused on confirming that the bubble sort algorithm worked correctly for all supported array lengths and that values were sorted **in-place** as expected. Our testing strategy was as follows:

- **Pre-sorted Arrays:** Arrays that were already sorted to ensure the algorithm didn't unnecessarily swap values.

- **Reverse Order Arrays:** Inputs in strictly descending order to confirm the algorithm performs the maximum number of required swaps.

- **Mixed Order Arrays:** Arrays with random unsorted values to simulate a real sorting scenario and ensure the correct output.

- **Edge Case - Duplicate Values:** Arrays with multiple identical values to confirm they are preserved correctly in the sorted result.

- **Minimum and Maximum Array Sizes:** Sorting with 3 entries and 10 entries to validate handling of size limits.

Memory was inspected after the sort to verify that the sorted data correctly overwrote the original data stored at the R0 address.

Part C – Display Subroutine

The Display subroutine was tested by observing the printed output and confirming that:

- The number of entries was displayed correctly.

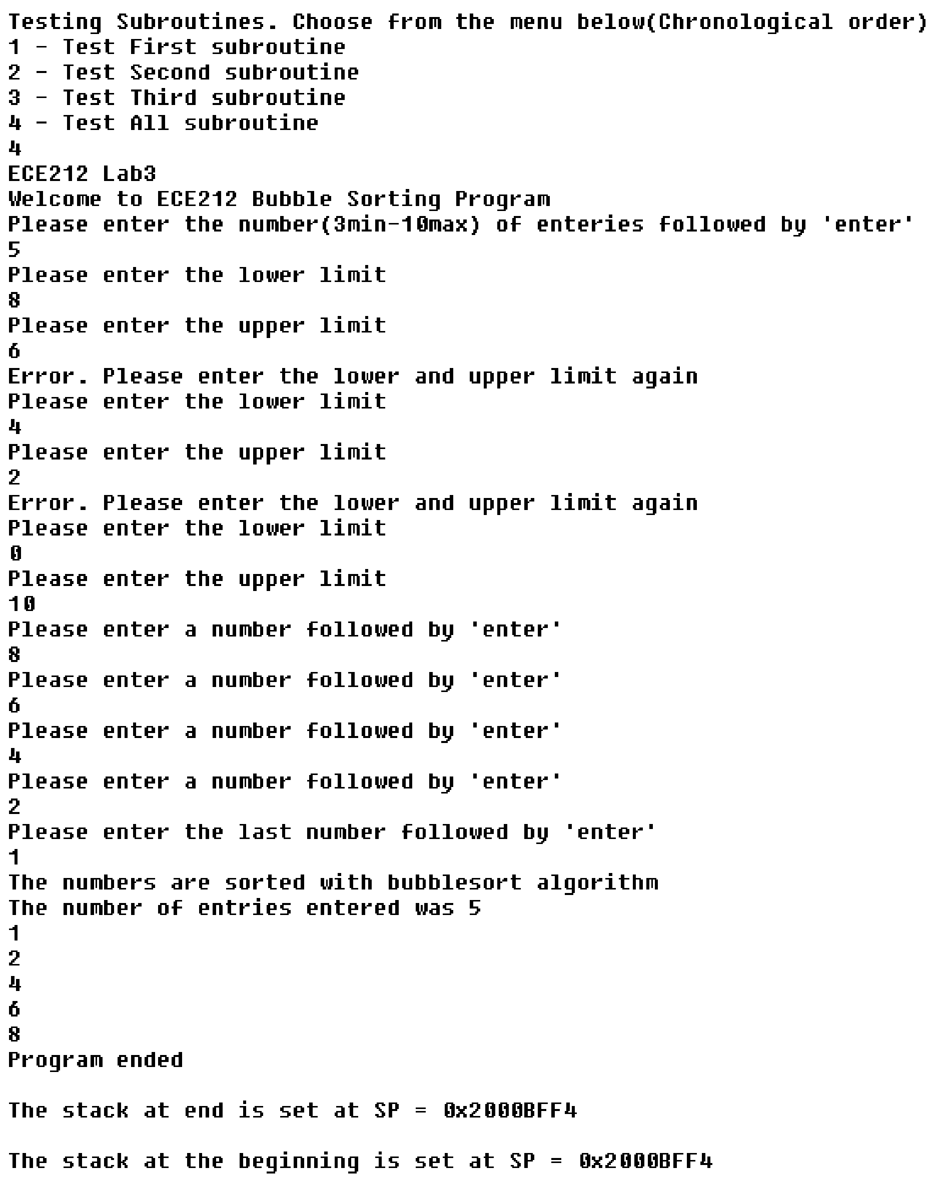
- The "Sorted from smallest to biggest, the numbers are:" message appeared before output.

- Each sorted number was printed on a new line using the value and cr subroutines.

- The program ended with "Program ended" after the final value was printed.

We also checked the order of printed values against the sorted memory to ensure the Display subroutine iterated correctly through the stored list. Tests were repeated with various array sizes and content to verify that nothing was missed or out of order.

3.2 Results:



```
Testing Subroutines. Choose from the menu below(Chronological order)
1 - Test First subroutine
2 - Test Second subroutine
3 - Test Third subroutine
4 - Test All subroutine
4
ECE212 Lab3
Welcome to ECE212 Bubble Sorting Program
Please enter the number(3min-10max) of enteries followed by 'enter'
5
Please enter the lower limit
8
Please enter the upper limit
6
Error. Please enter the lower and upper limit again
Please enter the lower limit
4
Please enter the upper limit
2
Error. Please enter the lower and upper limit again
Please enter the lower limit
0
Please enter the upper limit
10
Please enter a number followed by 'enter'
8
Please enter a number followed by 'enter'
6
Please enter a number followed by 'enter'
4
Please enter a number followed by 'enter'
2
Please enter the last number followed by 'enter'
1
The numbers are sorted with bubblesort algorithm
The number of entries entered was 5
1
2
4
6
8
Program ended

The stack at end is set at SP = 0x2000BFF4

The stack at the beginning is set at SP = 0x2000BFF4
```

Fig 3.1: Results for the entire program.

4. Questions

1. **Is it always necessary to implement either callee or caller preservation of registers when calling a subroutine? Why?**
   Yes. If the callee or caller does not preserve the used registers, critical data may be overwritten, leading to unpredictable behavior. In this lab, the callee method ensures the main program can trust its register contents after subroutine calls.

2. **Is it always necessary to clean up the stack? Why?**
   Yes. If the stack is not cleaned properly, leftover values can corrupt the logic of future subroutines, leading to runtime errors and incorrect results. The stack should be restored to its prior state on exit.

3. **If a proper check for the getstring function was not provided and you have access to the buffer, how would you check to see if a valid number was entered?**
   We would iterate through each character in the buffer to check if it's between ASCII values of '0' to '9' or a minus sign ('-') at the beginning. If any other character is present, it would be invalid. Once verified, we would convert the string to an integer by multiplying the existing number by 10 and adding the next digit's value.

5. Conclusion

This lab solidified the concepts of subroutines, stack usage, and register preservation. Each subroutine was designed to be modular, reusable, and aligned with ARM assembly best practices. Key takeaways include:

- How to safely pass/return data through the stack.
- Using subroutines to structure assembly code clearly.
- Implementing validation and handling user interaction effectively.

This modular approach not only improved readability but also made debugging and testing far easier.

6. Appendix

6.1 Assembler Code

Part A – WelcomePrompt Subroutine

```
/* Author: Misbah Ahmed Nauman, Student ID: 1830574 */
/* Author: Muhammad Ayaan Hafeez, Student ID: 1805075*/

Welcomeprompt:
        // Save return address and used registers
        PUSH {LR}
        PUSH {R9}
        PUSH {R8}
        PUSH {R7}
        PUSH {R6}
        PUSH {R5}
        PUSH {R4}

        MOV R7, R0 // Print Welcome
        LDR R0, =Label1
        BL printf
        BL cr
        LDR R0, =Label2
        BL printf
        BL cr

Start:
        LDR R0, =PromptEntry // Ask user for number of entries
        BL printf
        BL cr
        BL getstring // Get the number of entries
        MOV R4, R0 // Store number of entries in R4
        BL value // Prints decimal inputted

        CMP R4,#3 // Check if it's within the range 3-10
        BLT InvalidNumEntries1
        CMP R4,#10
        BGT InvalidNumEntries2

        STR R4,[SP,#28] // Push the number of entries to stack

AskLimits:
        BL cr
        LDR R0, =LowerPrompt
        BL printf
        BL cr
        BL getstring
        MOV R5,R0 // Store lower limit in R5
        BL value

        BL cr
        LDR R0, =UpperPrompt
        BL printf
        BL cr
        BL getstring
        MOV R6,R0 // Store upper limit in R6
        BL value

        CMP R5,R6 // Check if lower limit < upper limit
        BLT AskNumbers // If valid, move to number input
        Bl cr
        LDR R0, =InvalidLim
        BL printf
        BAL AskLimits // If invalid, re-enter only limits

AskNumbers:
        MOV R8,R4 // Copy number of entries to R8 for loop counter

NumberInputLoop:
        BL cr
        CMP R8,#1 // Check if this is the last number to enter
        BEQ LastNumberPrompt // If it's the last number, show a special message

        LDR R0, =EnterNum
        BL printf
        Bl cr
        BL getstring // Get user input
        MOV R9, R0 // Store input number in R9
        BL value

        CMP R9, R5 // Check if number >= lower limit
        BLT InvalidNumber
        CMP R9, R6 // Check if number <= upper limit
        BGT InvalidNumber

        STR R9,[R7],#4 // Store number in memory, move address forward
        SUB R8,#1 // Decrement loop counter
        CMP R8,#0 // Check if all numbers entered
        BNE NumberInputLoop // Repeat until all numbers are entered
        B Done // Jump to the end if no more numbers to input

LastNumberPrompt:
        LDR R0, =LastNum
        BL printf // Display message for the last number
        BL cr
        BL getstring // Get user input for the last number
        MOV R9,R0 // Store input number in R9
        BL value
        BL cr

        CMP R9,R5 // Check if number >= lower limit
        BLT InvalidNumber
        CMP R9,R6 // Check if number <= upper limit
        BGT InvalidNumber

        STR R9,[R7],#4 // Store number in memory, move address forward
        SUB R8,#1 // Decrement loop counter
        B Done

InvalidNumEntries1: // Error message for entries < 3
        BL cr
        LDR R0, =InvalidEntry1
        BL printf
        BL cr
        BAL Start // Restart the subroutine if entries are invalid

InvalidNumEntries2: // Error message for entries > 10
        BL cr
        LDR R0, =InvalidEntry2
        BL printf
        BL cr
        BAL Start // Restart the subroutine if entries are invalid

InvalidNumber:
        BL cr
        LDR R0, =InvalidRange
        BL printf
        BAL NumberInputLoop // Re-enter only the current number

Done:
        POP {R4}
        POP {R5}
        POP {R6}
        POP {R7}
        POP {R8}
        POP {R9}
        POP {PC} // Restore return address and used registers
/*-------Code ends here ---------------------*/

/*-----------------Add your strings here in the data section--------*/
.data
Label1:
.string "ECE212 Lab3"
Label2:
.string "Welcome to ECE212 Bubble Sorting Program"
PromptEntry:
.string "Please enter the number(3min-10max) of enteries followed by 'enter'"
InvalidEntry1:
.string "Invalid entry, Please enter more than 2 entry"
InvalidEntry2:
.string "Invalid entry, Please enter less than 11 entry"
LowerPrompt:
.string "Please enter the lower limit"
UpperPrompt:
.string "Please enter the upper limit"
InvalidLim:
.string "Error. Please enter the lower and upper limit again"
EnterNum:
.string "Please enter a number followed by 'enter'"
InvalidRange:
.string "Invalid!!! Number entered is not within the range"
LastNum:
.string "Please enter the last number followed by 'enter'"
```

Part B – Sort Subroutine

```
/* Author: Misbah Ahmed Nauman, Student ID: 1830574 */
/* Author: Muhammad Ayaan Hafeez, Student ID: 1805075*/

Sort:
        PUSH {LR}
        PUSH {R12}
        PUSH {R11}
        PUSH {R10}
        PUSH {R9}
        PUSH {R8}
        PUSH {R7}
        PUSH {R6}                           // Save return address and used registers
        PUSH {R5}
        PUSH {R4}
        LDR R4,[SP,#40]          // Load number of entries from stack
        MOV R5,R0              // Move R0=0x20001000 to R5
        SUB R6, R4, #1
        MOV R7, #0
OuterLoop:
        CMP R7, R4
        BGE ExitSort
        MOV R8, #0         // Number of comparisons in this pass
InnerLoop:
        SUB R9, R6, R7
        CMP R8, R9
        BGE Done
        ADD R10, R8, #1
        LDR R11, [R5, R8, LSL #2]     // Load current element
        LDR R12, [R5, R10, LSL #2]    // Load next element
        CMP R11, R12                 // Compare
        BLE NoSwap
        // Swap R0 and R1
        STR R12, [R5, R8, LSL #2]     // Store smaller value in first position
        STR R11, [R5, R10, LSL #2]    // Store larger value in second position
NoSwap:
        ADD R8, #1                 // Move to next pair
        B InnerLoop
Done:
        ADD R7,#1
        B OuterLoop
ExitSort:
        MOV R10, R4
        POP {R4}
        POP {R5}
        POP {R6}
        POP {R7}
        POP {R8}
        POP {R9}
        POP {R10}
        POP {R11}
        POP {R12}
        POP {PC}        // Restore return address and used registers
```

```
/*-------Code ends here ---------------------*/
```

Part C – Display Subroutine

```
/* Author: Misbah Ahmed Nauman, Student ID: 1830574 */
/* Author: Muhammad Ayaan Hafeez, Student ID: 1805075*/

Display:
        PUSH {LR} // Save return address
        PUSH {R10}
        PUSH {R6}
        PUSH {R5}
        PUSH {R4}

        LDR R5, [SP,#24] // Load base address of array
        LDR R0, =Label1
        BL printf
        BL cr

        LDR R0, =LabelNumEntries // Load label for "Number of entries"
        BL printf

        LDR R4, [SP,#20] // Load number of elements
        MOV R0, R4
        BL value // Display number of entries
        BL cr
        MOV R6, #0 // Index counter

DisplayLoop:
        CMP R6, R4 // Check if we've printed all elements
        BGE DoneDisplay // If done, exit

        LDR R0, [R5, R6, LSL #2] // Load current array element
        // Call print subroutine
        BL value
        BL cr

        ADD R6, R6, #1 // Move to next element
        B DisplayLoop // Repeat

DoneDisplay:
        LDR R0, =Exit
        BL printf
        BL cr
        POP {R4}
        POP {R5}
        POP {R6}
        POP {R10}
        POP {PC} // Restore return address
/*-------Code ends here ---------------------*/

/*-----------------Add your strings here in the data section--------*/
.data
Label1:
.string "The numbers are sorted with bubblesort algorithm"
LabelNumEntries:
.string "The number of entries entered was "
NumOrder:
.string "Sorted from smallest to biggest, the numbers are:"
Exit:
.string "Program ended"
```

6.2 Lab Work Distribution

- **Code Development -** Misbah 50%, Ayaan 50%
- **Demo Presence -** Misbah 50%, Ayaan 50% (Both were present and answered any questions or changes made to the code)
- **Lab Report -** Misbah 50%, Ayaan 50%