Prelab Report: Design and Analysis of Embedded Systems

Arye Mindell, LMU EECE 3200

***Abstract*—This report outlines the development and design of a program in ARM Assembly Language for the Raspberry Pi that counts the occurrences of the substring ”LMU” in a given string. The output is displayed on the terminal as well as visually with connected LEDs, showcasing the integration of software and hardware functionality for embedded systems. The experimental procedure involves testing the program with predefined input strings to validate its functionality and observing the results displayed on both the terminal and LEDs.**

I. Introduction

THIS lab leverages ARM Assembly Language to implement a functional embedded system on a Raspberry Pi. The primary objective is to process user input, specifically a 15-character ASCII string, to detect and count instances of the substring ”LMU.” This information is displayed both on the terminal and visually on GPIO-connected LEDs in binary form. The program operates in an infinite loop, continuously requesting new input and providing feedback. [1]

II. System Description

THE hardware system consists of a Raspberry Pi microcontroller connected to three LEDs via GPIO pins 17, 18, and 27, each paired with a 220-ohm resistor to limit current. These LEDs visually display binary output corresponding to the processed data, while the system interfaces with a keyboard and monitor for input and output.

The software implementation involves modular assembly routines designed to ensure clarity, maintainability, and accuracy in real-time operation. The main loop of the program operates as follows:

1) Display the prompt message: ”Please type text of 15 characters.”
2) Read a 15-character string from the keyboard and store it in memory.
3) Loop through the string to count occurrences of ”LMU”:
   - Compare each character to detect ”L,” followed by ”M,” and then ”U.”
   - Increment the count whenever ”LMU” is found.
4) Store the count in memory and convert it to ASCII format for display.
5) Output the count in decimal format on the monitor with the message: ”There are [number] instances of LMU.”
6) Convert the count to binary and display it using GPIO-connected LEDs on PINs 17, 18, and 27.
7) Repeat the process indefinitely, allowing for continuous input and processing.

The counting algorithm is detailed in the flowchart shown in Fig. 1.

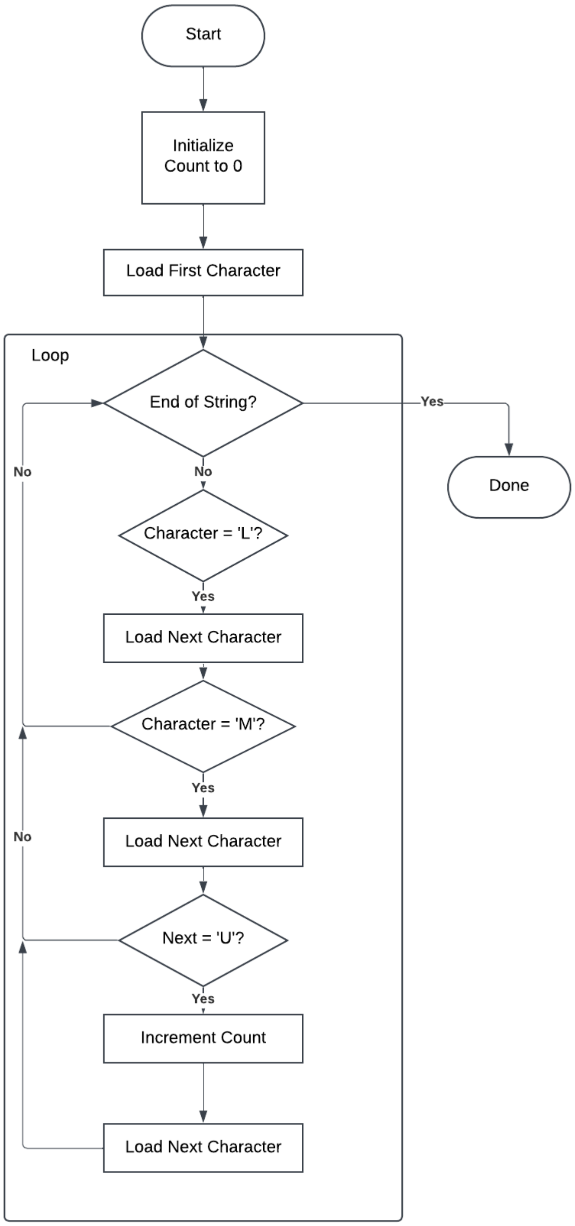


Fig. 1. Flowchart of the program algorithm.

III. Program Code

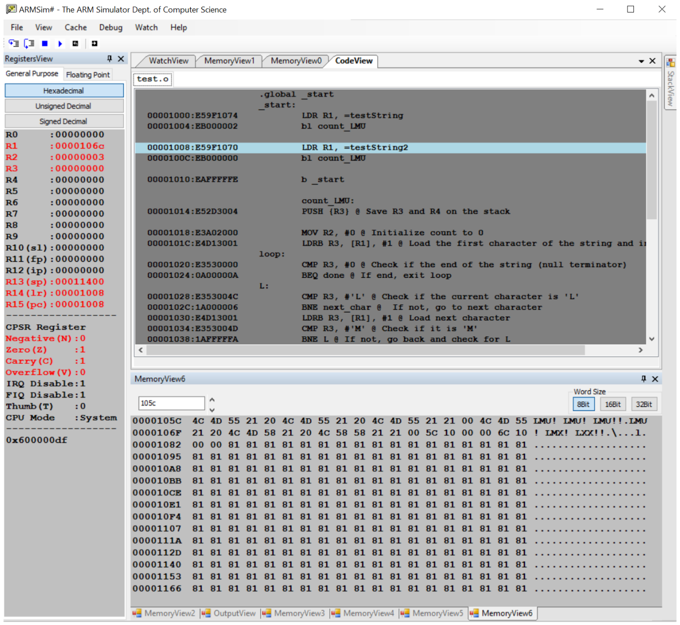
The following listing shows the ARM Assembly code implemented for the program, adhering to the design outlined in the flowchart and system description.

```

@ Define my Raspberry Pi
        .cpu    cortex-a53
        .fpu    neon-fp-armv8
        .syntax unified         @ modern syntax

@ Useful source code constants
        .equ    STDIN,0
        .equ    STDOUT,1
        .equ    aLetter,-16
        .equ    local,16

@ Constant program data
        .section  .rodata
        .align  2
promptMsg:
        .asciz   "Please type text of 15 characters: "
        .equ    promptLngth,.-promptMsg
responseMsg:
        .asciz   " instances of LMU\n"
        .equ    responseLngth,.-responseMsg

@ Program code
        .text
        .align  2
        .global main
        .type   main, %function
main:
        sub     sp, sp, 8       @ space for fp, lr
        str     fp, [sp, 0]     @ save fp
        str     lr, [sp, 4]     @   and lr
        add     fp, sp, 4       @ set our frame pointer
        sub     sp, sp, local   @ allocate memory for local var

        mov     r0, STDOUT      @ prompt user for input
        ldr     r1, promptMsgAddr
        mov     r2, promptLngth
        bl      write

        mov     r0, STDIN       @ from keyboard
        add     r1, fp, aLetter @ address of aLetter
        mov     r2, 15           @ one char
        bl      read

        add r1, fp, aLetter
        bl count_LMU

        mov     r0, STDOUT      @ nice message for user
        LDR R1, =response
        mov R2, 1
        bl write

        ldr     r1, responseMsgAddr
        mov     r2, responseLngth
        bl      write

        b main

count_LMU:
        PUSH {R3} @ Save R3 and R4 on the stack

        MOV R2, #0 @ Initialize count to 0
        LDR R3, =response
        str R2, [R3]
        LDRB R3, [R1], #1 @ Load the first character of the string and increment R1
loop:
        CMP R3, #0 @ Check if the end of the string (null terminator)
        BEQ done @ If end, exit loop
L:
        CMP R3, #'L' @ Check if the current character is 'L'
        BNE next_char @  If not, go to next character
        LDRB R3, [R1], #1 @ Load next character
        CMP R3, #'M' @ Check if it is 'M'
        BNE L @ If not, go back and check for L
        LDRB R3, [R1], #1 @ Load next character
        CMP R3, #'U' @ Check if it is 'U'
        BNE L @ If not, go back and check for L
        ADD R2, R2, #1 @ Increment count
next_char:
        LDRB R3, [R1], #1 @ Load the next character and increment R1
    B loop @ Repeat the loop
done:
        add R2, R2, #48 @ convert count to ascii
        LDR R3, =response
        str R2, [R3]
        POP {R3} @ Restore R3 and R4 from the stack
        BX LR                   @ Return from the function
@ Addresses of messages
        .align  2
promptMsgAddr:
        .word   promptMsg
responseMsgAddr:
        .word   responseMsg

.section .bss
        .align 2
        response:
                .space 4
```

Listing 1. ARM Assembly Code for the Program

IV. Test Cases

The following figures illustrate the output of the implemented program for two test cases, showcasing its functionality and correct operation.

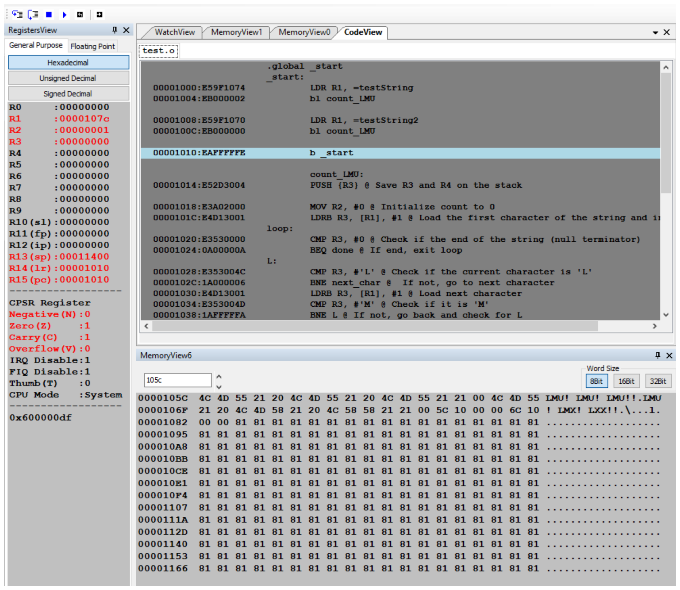


```
R0        :00000000
R1        :0000106c
R2        :00000003
R3        :00000000
R4        :00000000
R5        :00000000
R6        :00000000
R7        :00000000
R8        :00000000
R9        :00000000
R10(sl):00000000
R11(fp):00000000
R12(ip):00000000
R13(sp):00011400
R14(lr):00001008
R15(pc):00001008
------------------
CPSR Register
Negative(N):0
Zero(Z)    :1
Carry(C)   :1
Overflow(V):0
IRQ Disable:1
FIQ Disable:1
Thumb(T)   :0
CPU Mode   :System
------------------
0x600000df
```

```
                    .global _start
                    _start:
00001000:E59F1074           LDR R1, =testString
00001004:EB000002           bl count_LMU

00001008:E59F1070           LDR R1, =testString2
0000100C:EB000000           bl count_LMU

00001010:EAFFFFFE           b _start

                            count_LMU:
00001014:E52D3004           PUSH {R3} @ Save R3 and R4 on the stack

00001018:E3A02000           MOV R2, #0 @ Initialize count to 0
0000101C:E4D13001           LDRB R3, [R1], #1 @ Load the first character of the string and i
                    loop:
00001020:E3530000           CMP R3, #0 @ Check if the end of the string (null terminator)
00001024:0A00000A           BEQ done @ If end, exit loop
                    L:
00001028:E353004C           CMP R3, #'L' @ Check if the current character is 'L'
0000102C:1A000006           BNE next_char @  If not, go to next character
00001030:E4D13001           LDRB R3, [R1], #1 @ Load next character
00001034:E353004D           CMP R3, #'M' @ Check if it is 'M'
00001038:1AFFFFFA           BNE L @ If not, go back and check for L
```

```
105c

0000105C  4C 4D 55 21 20 4C 4D 55 21 20 4C 4D 55 21 21 00 4C 4D 55 LMU! LMU! LMU!!.LMU
0000106F  21 20 4C 4D 58 21 20 4C 58 58 21 21 00 5C 10 00 00 6C 10 ! LMX! LXX!!.\...l.
00001082  00 00 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
00001095  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
000010A8  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
000010BB  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
000010CE  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
000010E1  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
000010F4  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
00001107  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
0000111A  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
0000112D  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
00001140  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
00001153  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
00001166  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
```

Fig. 2. Output for Test Case 1: Input string ”LMU! LMU! LMU!!”



```
R0        :00000000
R1        :0000107c
R2        :00000001
R3        :00000000
R4        :00000000
R5        :00000000
R6        :00000000
R7        :00000000
R8        :00000000
R9        :00000000
R10(sl):00000000
R11(fp):00000000
R12(ip):00000000
R13(sp):00011400
R14(lr):00001010
R15(pc):00001010
------------------
CPSR Register
Negative(N):0
Zero(Z)    :1
Carry(C)   :1
Overflow(V):0
IRQ Disable:1
FIQ Disable:1
Thumb(T)   :0
CPU Mode   :System
------------------
0x600000df
```

```
                    .global _start
                    _start:
00001000:E59F1074           LDR R1, =testString
00001004:EB000002           bl count_LMU

00001008:E59F1070           LDR R1, =testString2
0000100C:EB000000           bl count_LMU

00001010:EAFFFFFE           b _start

                            count_LMU:
00001014:E52D3004           PUSH {R3} @ Save R3 and R4 on the stack

00001018:E3A02000           MOV R2, #0 @ Initialize count to 0
0000101C:E4D13001           LDRB R3, [R1], #1 @ Load the first character of the string and i
                    loop:
00001020:E3530000           CMP R3, #0 @ Check if the end of the string (null terminator)
00001024:0A00000A           BEQ done @ If end, exit loop
                    L:
00001028:E353004C           CMP R3, #'L' @ Check if the current character is 'L'
0000102C:1A000006           BNE next_char @  If not, go to next character
00001030:E4D13001           LDRB R3, [R1], #1 @ Load next character
00001034:E353004D           CMP R3, #'M' @ Check if it is 'M'
00001038:1AFFFFFA           BNE L @ If not, go back and check for L
```

```
105c

0000105C  4C 4D 55 21 20 4C 4D 55 21 20 4C 4D 55 21 21 00 4C 4D 55 LMU! LMU! LMU!!.LMU
0000106F  21 20 4C 4D 58 21 20 4C 58 58 21 21 00 5C 10 00 00 6C 10 ! LMX! LXX!!.\...l.
00001082  00 00 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
00001095  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
000010A8  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
000010BB  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
000010CE  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
000010E1  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
000010F4  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
00001107  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
0000111A  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
0000112D  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
00001140  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
00001153  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
00001166  81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ...................
```

Fig. 3. Output for Test Case 2: Input string ”LMU! LMX! LXX!!”

References

[1] Loyola Marymount University, Department of Electrical Engineering and Computer Science, “EECE 3200: Lab 1 - Introduction to ARM Assembly Language Programming Using Raspberry Pi,” Spring 2025.