

Lab 4: Arrays, Loops, and Functions

Victor Yuan

Microcontrollers and Embedded Systems

February 23, 2017

Introduction:

The Goal of this lab was to write a program in assembly read in a line of ASCII characters add those variables up storing the sum value in a checksum variable, while using a loop.

Procedure:

Given a sample of code written in C, write an assembly code that mimics or reproduce the same functions the C program is doing.

First declare an array called buffer, and allocate it 4096 bytes of memory. Then declare the variable(s) needed for the program, the ASCII prompt(s) and the str1 needed for scanf. Then declare a function called checksum leave this empty for now and come back later, it serves as a place holder for now.

Move onto main, begin by loading the prompt into register r0. Print with bl printf. Then prepare r0 and r1 with locations of str1, and buffer for scanf. Then use bl scanf to read the the user's input into the buffer using str1's format Now call the function checksum, call a branch link for checksum to return to the spot.

Writing the checksum, first load a register with the location of sum variable, this variable will be the variable that holds the sum combination ASCII value of the user input. Load a different register with location of tracking byte, this will be used later to determine how many times a loop has to be executed. Load this value with the constant 0 at the start of the checksum function. Load buffer with a constant 0 as well. Begin the loop with an indented "loop:" label. Use ldrb to load a single byte to a different register, r4 for example, load the value stored in r3, r3 should contain location of array buffer, every time load is called increase by 1 so that the next element in the array can be loaded into r4. Add r4 and r2 and store it back into r2 this lets value stored in r2 slowly accumulate. End the loop with bne loop which loops for a comparison inside the loop. The loop ends when r7, the register holding the tracking byte, equals the max size of the buffer array, in this case it is 4096. Move the program counter back to when the checksum function was first called.

Call the function again and test the code.

Code:

```
.data
buffer: .space 4096
i:      .word 0
sum:    .word 0
prompt1: .asciz "Enter text (Esc to end):\n"
str1:   .asciz "%s"
        .text
res:    .asciz "the checksum is %d\n"
.global main
```

Declaration of variables buffer holds in user input.

I is the tracking byte.

Sum holds the checksum value.

Prompt1 prompts the user to enter the string.

Str1 is the format.

And res is the line for displaying results.

```
main:

        ldr r0, =prompt1
        bl printf          @prints out user prompt

        ldr r0, =str1
        ldr r1, =buffer
        bl scanf           @reads first input into array
        bl checksum
        ldr r1, =sum
        ldr r1, [r1]
        ldr r0, =res
        bl printf
```

Main code, highlighted portion is when checksum is first called.

Which moves the program to the following function

```
checksum:
    ldr r9, =sum
    ldr r7, =i
    mov r7, #0
    ldr r3, =buffer
    mov r2, #0

    loop:
        ldrb r4, [r3], #1

        add r2, r2, r4
        str r2, [r9]
        add r7, r7, #1
        cmp r7, #4096 @max size
        bne loop

    ldr r0, =str1
    mov pc,lr
```

In checksum locations of sum, i, and buffer are moved into appropriate registers, and then the loop is initiated. Once the loop is done the program counter is moved back to the branch link so that the sum is stored in r1 then the value of r1 is called and stored into r1 at which point r0 is called to store location of the result message and bl printf prints the first result.

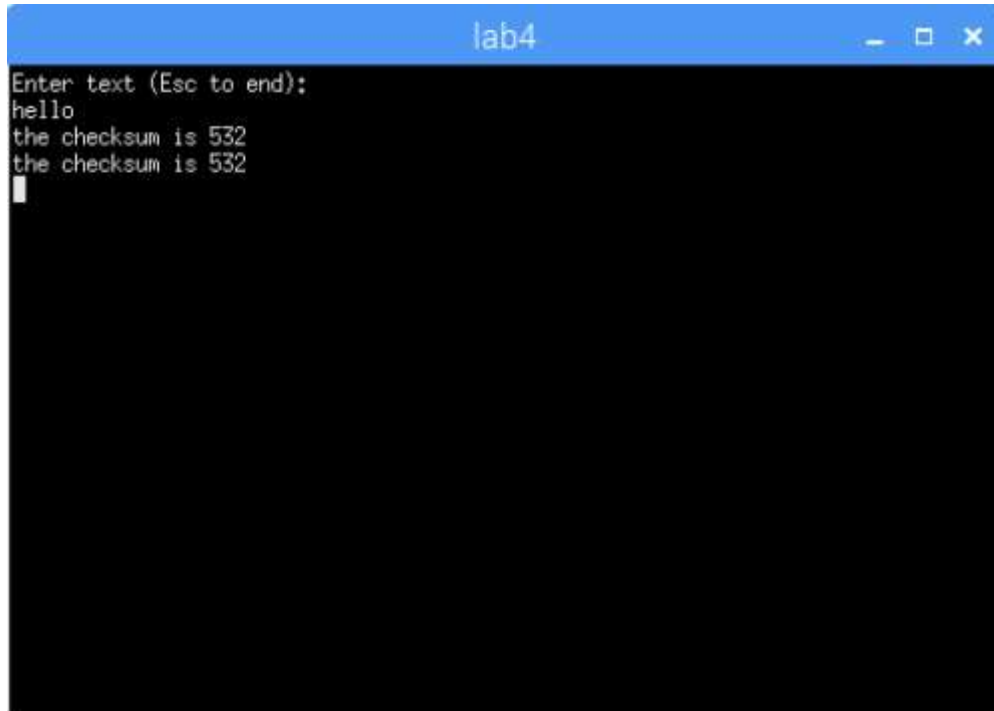
Then the checksum is called once again

```
bl checksum
    ldr r1, =sum
    ldr r1, [r1]
    ldr r0, =res
    bl printf

    mov r0, #0
    mov pc,lr
```

The last two lines are there so that the program doesn't run into an error while in execution.

Results:



```
lab4
Enter text (Esc to end):
hello
the checksum is 532
the checksum is 532
█
```

Shown above is the checksum value of hello displayed twice, both values matched each other.

Question:

If your assembly code is correctly implemented, the second checksum will be incorrect. WHY?

Since my code reset the sum to the constant 0 every time it runs it does not remember the values it used last time.

Conclusion:

One thing I learned is that to end a bne loop is that you have to use cmp command to compare two values if the first value is greater than the 2nd then the loop ends.