EE277 Embedded System Design Course

LAB 2: Implement and Debug String Reverse Function in ARM Assembly Language and C

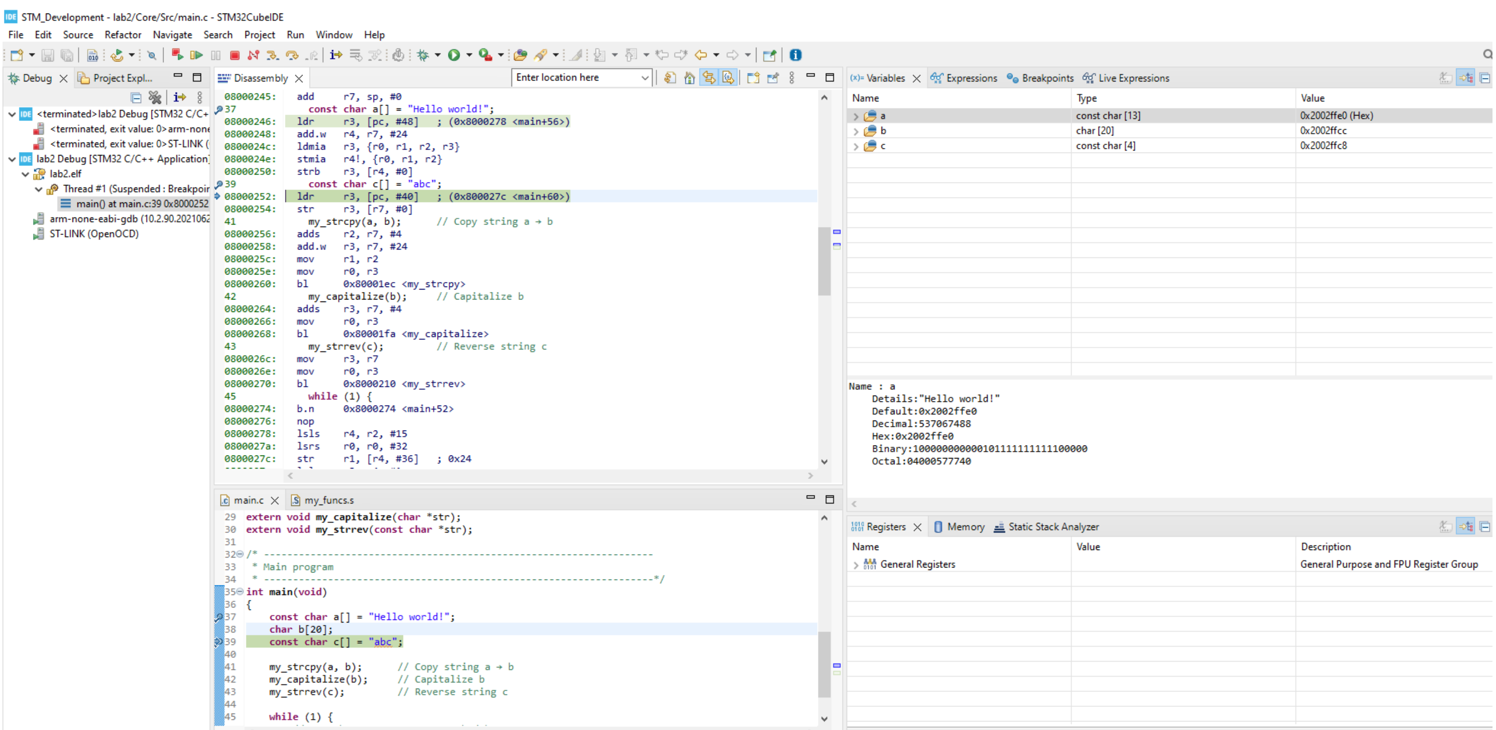
Baron Eiley

EE 277 Embedded SoC Design

1.1 Implement a string reversal function in assembly and call it in C (30 points)

Note to professor:

I was unable to attain the correct license from the arm sales-person, however, I found a workaround to complete the lab using STM32CubeIDE software and my NucleoSTM-F429Z1 microcontroller board (https://os.mbed.com/platforms/ST-Nucleo-F429ZI/). The STM32CubeIDE software has the same base infrastructure as the ARM DS IDE. See figure below. It supports debugging of the stack register and supports the compilation of inline assembly using custom startup files (.s). I will explain more in the video submission on how I was able to run lab two using this software and this hardware.



```
    .global my_strrev
my_strrev:
    PUSH    {r1-r5, lr}
    MOV     r1, r0
    MOV     r2, r0
find_end:
    LDRB    r3, [r2]
    CMP     r3, #0
    BEQ     end_found
    ADD     r2, r2, #1
    B       find_end
end_found:
    SUB     r2, r2, #1
rev_loop:
    CMP     r1, r2
    BHS     done
    LDRB    r3, [r1]
    LDRB    r4, [r2]
    STRB    r4, [r1]
    STRB    r3, [r2]
    ADD     r1, r1, #1
    SUB     r2, r2, #1
    B       rev_loop
done:
    POP     {r1-r5, pc}
```

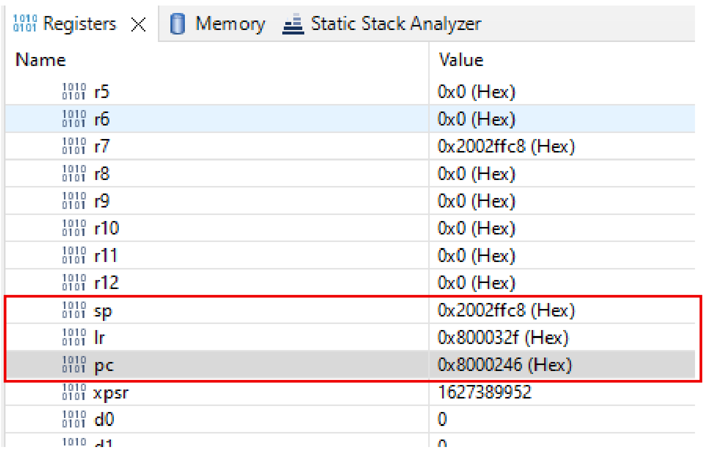
This is my implementation of the string reverse function in inline assembly.

The reverse string function operates by using two pointers to scan forward to find the null terminator and the steps back to point to the last char. The program continuously swaps the character they point to from one end to the other. Once they meet in the middle, the string has been reversed and the stores the value into the register and the branch returns.

1.2 Please answer the following questions based on your program debugging

1. Run the program until the opening brace in the main function is highlighted. Open the Registers window (Window->Show View->Registers). What are the values of the stack pointer (SP), LR, and the PC? (Insert screenshot of your editor.) **(3 points)**

The value of the stack point (SP) is 0x2002ffc8, the value of the link register (LR) is 0x800032f and the value of the PC is 0x8000246.



2. Open the Disassembly window (Window->Show View->Disassembly). Which instruction is highlighted, and what is its address? How does this address relate to the value of PC? (Insert screenshot of your disassembly window.) **(3 points)**

The instruction that is highlighted is "08000246: ldr r3, [pc, #48] ; (0x8000278 <main+56>)" This makes sense as this is the first part of the assembly instruction that will load "Hello World" into const char a[]: The address at this point is 0x8000246, which is the same value as the program counter as the PC is responsible for keeping track of where the current execution of the code.

3. Switch to "Step by Instruction" mode by clicking in the Debug Control window. Step one machine instruction using the F5 key while the Disassembly window is selected. Which two registers have changed (they should be highlighted in the Registers window), and how do they relate to the instruction just executed? (Insert the screenshot.) **(3 points)**

Since I am using a different software for this lab, I have a similar button that lets me step by instruction. When I press the F5 key, the two registers that changed were r0, and r1, in addition to the PC. They relate to the instruction being executed because they are loading in the arguments from the function to the desired registers. (More registers are highlighted because the software compiler I am using compiles the c code in my main.c different than the ARM DE IDE). Nonetheless, this shows the arguments from the function being loaded in.

r0	0x6c6c6548 (Hex)
r1	0x6f77206f (Hex)
r2	0x21646c72 (Hex)
r3	0x0 (Hex)
r4	0x2002ffec (Hex)

4. Look at the instructions in the Disassembly window. Do you see any instructions that are four bytes long? If so, what are the instructions? (Insert screenshot.) **(3 points)**

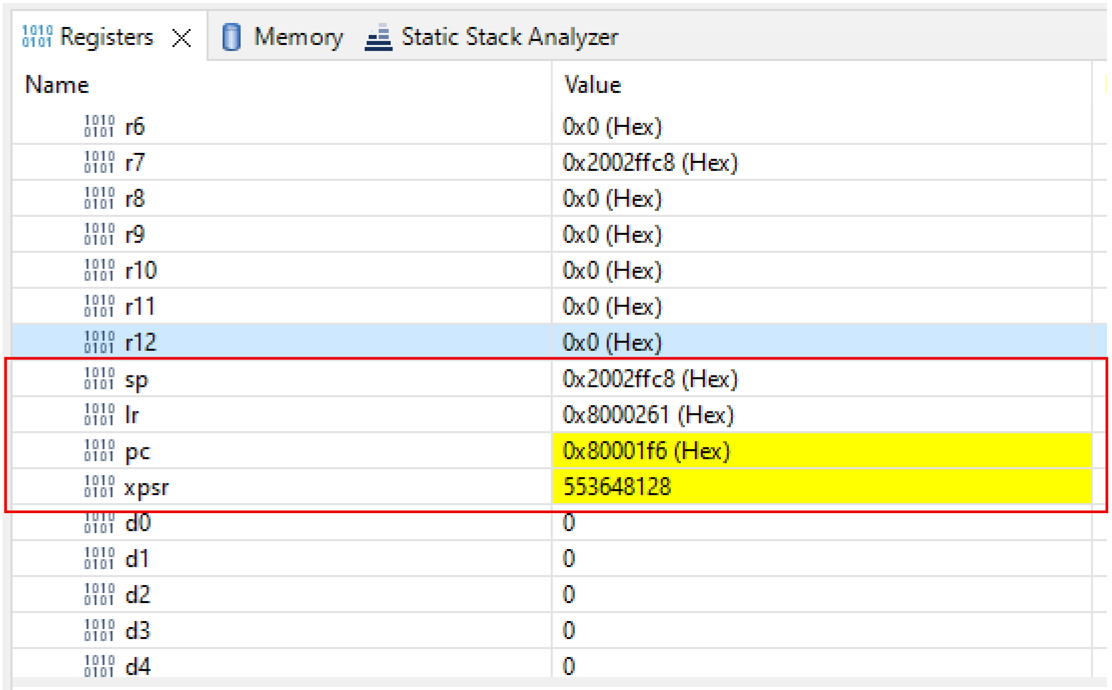
According to my disassembly window, there is one instruction that is 4 bytes long as it relates to my_strcpy function: 0800025c: bl   0x80001ec <my_strcpy>

```
Disassembly
08000250:   strb    r3, [r4, #0]
41            my_strcpy(a, b);      // Copy string a → b
08000252:   adds    r2, r7, #4
08000254:   add.w   r3, r7, #24
08000258:   mov     r1, r2
0800025a:   mov     r0, r3
0800025c:   bl      0x80001ec <my_strcpy>
42            my_capitalize(b);     // Capitalize b
08000260:   adds    r3, r7, #4
08000262:   mov     r0, r3
08000264:   bl      0x80001fa <my_capitalize>
43            my_strrev(b);         // Reverse string c
08000268:   adds    r3, r7, #4
0800026a:   mov     r0, r3
0800026c:   bl      0x8000210 <my_strrev>
45            while (1) {
08000270:   b.n     0x8000270 <main+48>
08000272:   nop
08000274:   lsls    r4, r1, #15
08000276:   lsrs    r0, r0, #32
          NMI_Handler:
08000278:   push    {r7}
0800027a:   add     r7, sp, #0
 75         while (1)
0800027c:   b.n     0x800027c <NMI_Handler+4>
 85       {
          HardFault_Handler:
0800027e:   push    {r7}
08000280:   add     r7, sp, #0
```

5. Continue execution (using F5) until reaching the BL my_strcpy instruction. What are the values of the SP, PC, and LR? (Insert screenshot.) **(3 points)**

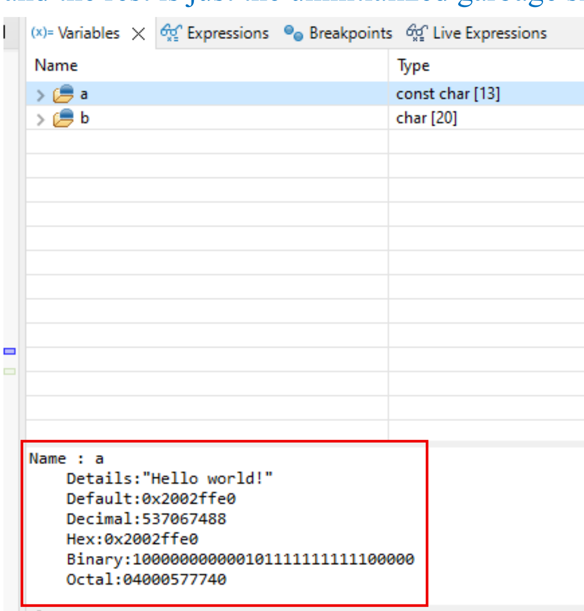
Once we loop through the end of my_strcpy, the values of the SP is 0x2002ffc8, the values of the PC is 0x80001f8, and the value of the LR register is 0x8000261. The SP did not change during these execution and this is because the function did not use the command "PUSH" or "POP".

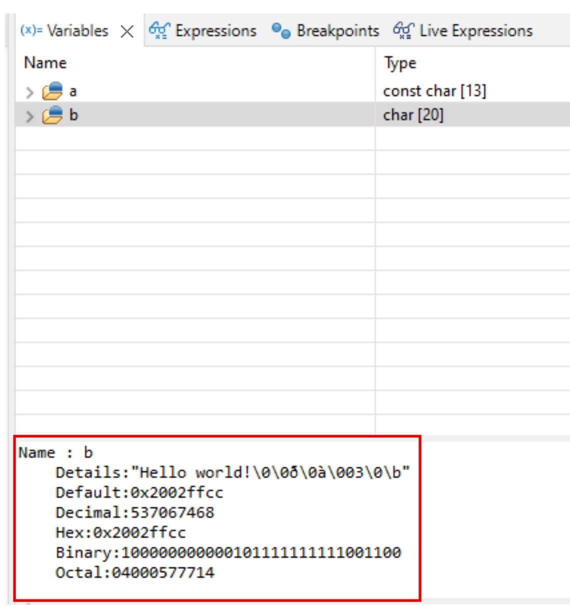
The SP point always points to the top of the stack and if these commands aren't used with the compiler within this STM software, therefore the SP won't change.



6. Watch the Variables window (Window->Show View-> Variables) to analyze the variables "a" and "b" (Insert screenshot). What is the value of "a"? What is the value of "b"? **(3 points)**

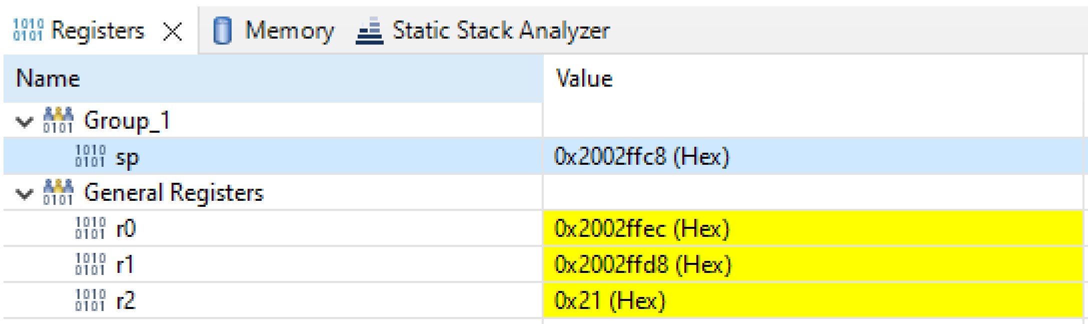
At this point in the execution, the values of variable a is "Hello world!" and the value of variable b is "Hello world!\0\0ð\0à\003\0\b". The value of b makes sense as the as it copied the value of a and the rest is just the uninitialized garbage since the char b is set to a element array of 20.





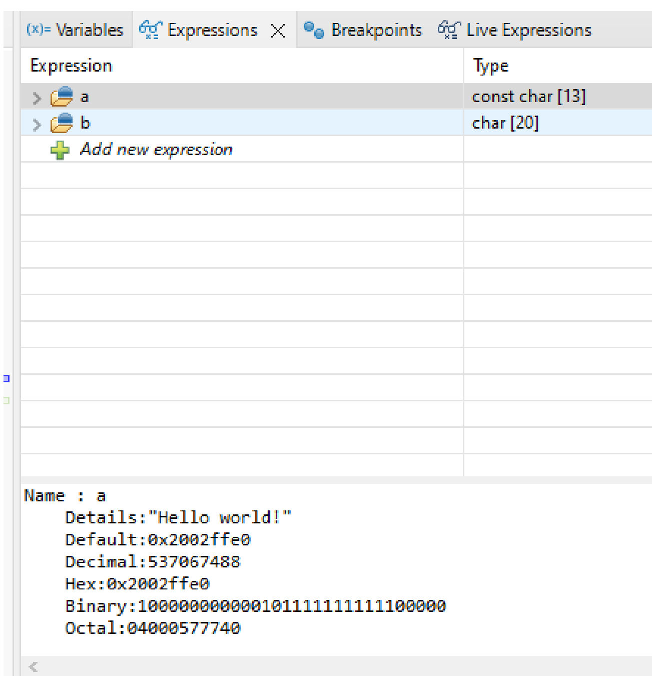
7. Which registers hold the arguments to my_strcpy, and what are their contents? (Insert screenshot.) **(3 points)**

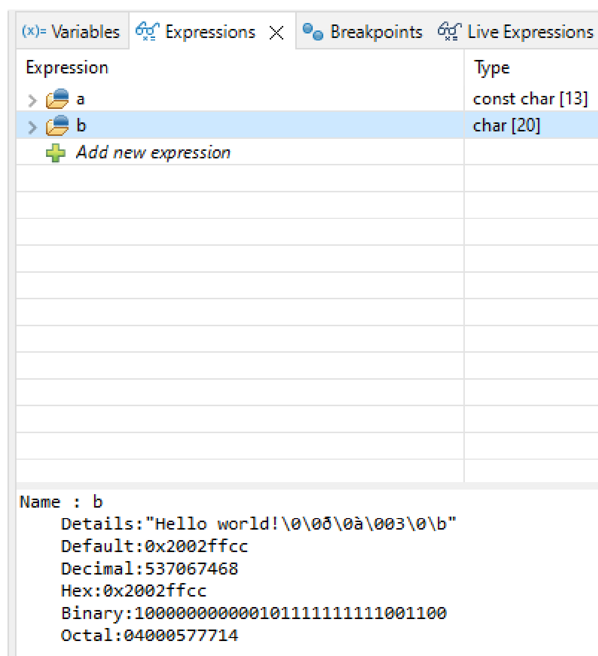
According to my software version, the registers that hold the arguments of my_strcpy is r0 and r1.



8. Use the Expressions window to watch the values in the address held in R0 and R1. Do the values match variables "a" and "b"? (Insert screenshot.) **(3 points)**

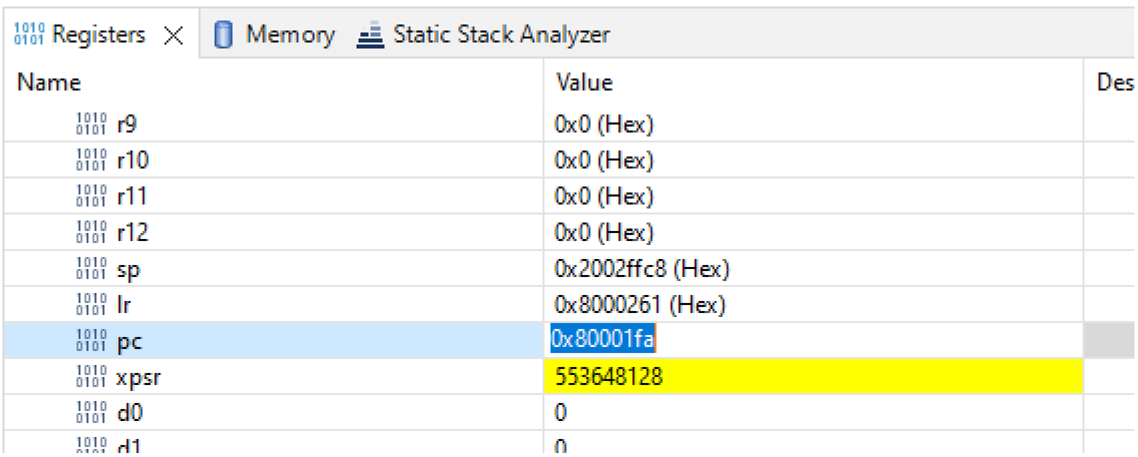
Yes, the values match.

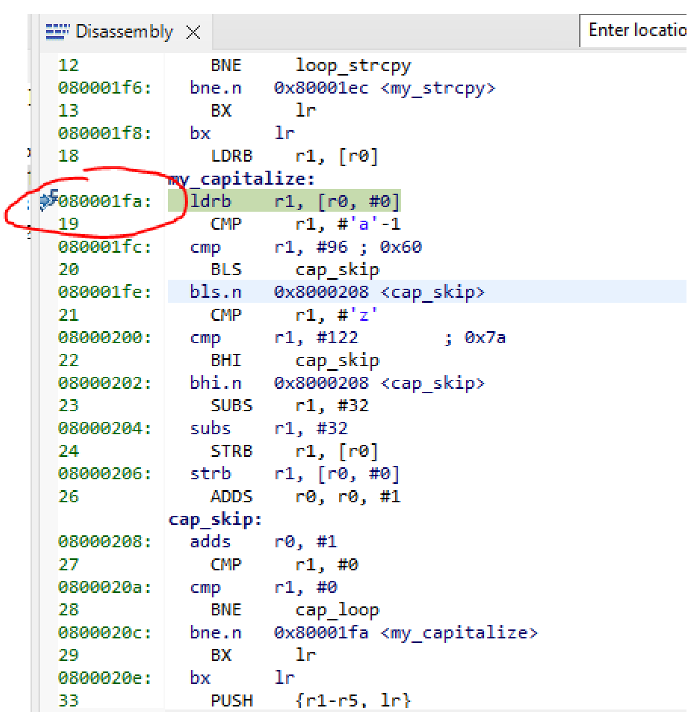




9. Execute the BL instruction. What are the values of the SP, PC, and LR? What has changed and why? Does the PC value agree with what is shown in the Disassembly window? (Insert screenshot.) **(3 points)**

After the execution of the BL instruction , the values of the SP is 0x2002ffc8, the PC is 0x80001fa, and the value of the LR is 0x8000269. The program counter changed because the we looped through to the next instruction. The link register is pointing to the location we need to jump back to after the function finishes. In this scenario, the program counter matches that of the disassembly window. Note: You must click on the value to see the real value of the register. This software uses a different representation.





10. Single step through the assembly code watching the "Expressions" window to see the string being copied character by character from a to b. Which register holds the character? **(3 points)**

In step 5 of this lab, we branched through the my_strcpy function already. So now we are in the my_capitalize function. I will loop through this function and analyze the Expressions window from there. At this point, the register that holds the character is R1.

General Registers	
r0	0x2002ffec (Hex)
r1	0x2002ffd8 (Hex)
r2	0x21 (Hex)

12. Execute the BX lr instruction. Now what is the value of the PC? **(3 points)**

After execution of BX lr, the value of the of the PC is 0x8000268.

Registers | Memory | Static Stack Analyzer

Name	Value
r6	0x0 (Hex)
r7	0x2002ffc8 (Hex)
r8	0x0 (Hex)
r9	0x0 (Hex)
r10	0x0 (Hex)
r11	0x0 (Hex)
r12	0x0 (Hex)
sp	0x2002ffc8 (Hex)
lr	0x8000261 (Hex)
pc	0x8000268

13. What is the relationship between the PC value and the previous LR value? Explain. **(3 points)**

As we learned from the previous lab, we learned that the relationship between the PC and the LR is that the LR holds the place where the program needs to jump back to in the main subroutine and once it does, the value of the PC will be the same as the LR because that is now where the current execution is.

14. Now step through the str_reverse subroutine and verify it works correctly, converting from "Hello world!" to "!dlrow olleH". ((Insert final screenshot.) **(11 points)**

I ran the code with the my_capitalize function still active but this question assumes that we run the copied lower case char b through my str_reverse. I will include both screenshots and recompile the code to do #14