CPE 390 – Microprocessor Systems
Lab 3
Michael Savo

Your purpose with this lab is to explore more advanced programs using the ARM assembly language while making connections to high-level programming languages you are (likely) more familiar with.

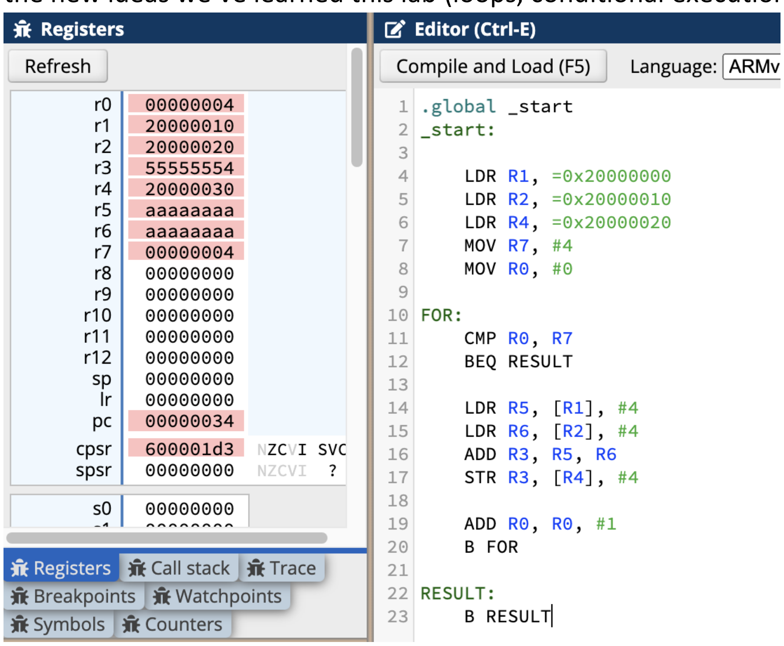
Instruction:

You will use https://cpulator.01xz.net/?sys=arm for simulated testing. After you've finished the code, compiled it, and successfully tested it, you can use “File -> Save” at the top to save your files to your computer and include those files in your submission!

Submission Requirements:

- Please download and submit the asm files on Canvas.
- For 3(a), please submit code in the corresponding high-level language (e.g. ‘.py’ file or ‘.cc’ file)

1. Last time we recreated matrix addition in assembly. Try to do this again using one or more of the new ideas we’ve learned this lab (loops, conditional execution, etc.)



This time around, I used loops and breaks to slightly condense down my original code. It also uses a compare function (using R0 as an index) to check if all the elements of the matrix have been processed.

2. Consider the following high-level code snippet. Assume the (signed) integer variables g and h are in r0 and r1, respectively.

if (g >= h)

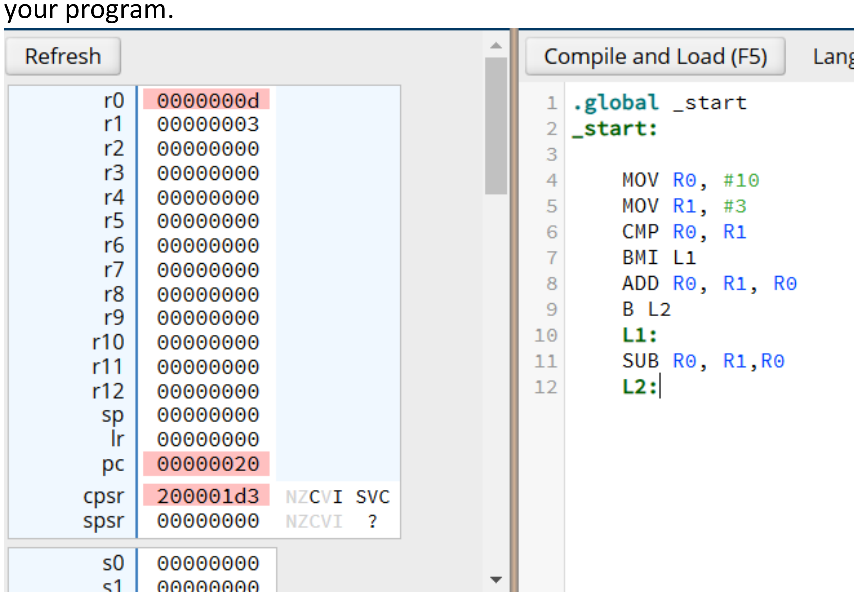
h = g + h;

else

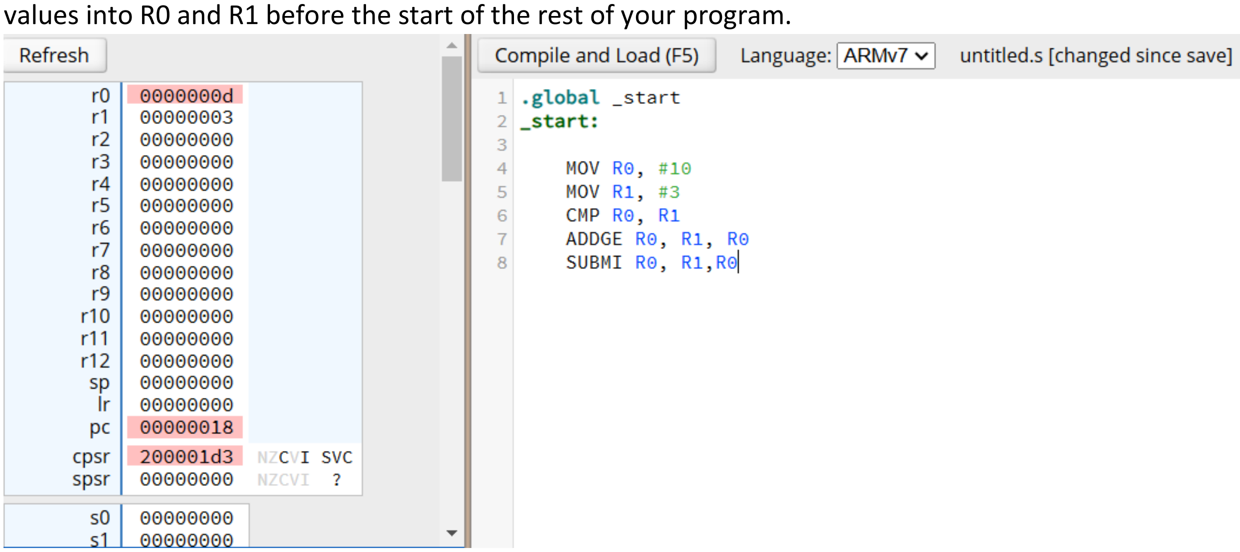
h = g – h;

//R0 = h, R1 = g

(a) Write the code snippet in ARM assembly language assuming conditional execution is available for branch instructions only. Use as few instructions as possible (within these parameters). Try the code for yourself by moving example values into R0 and R1 before the start of the rest of your program.



(b) Write the code snippet in ARM assembly language with conditional execution available for all instructions. Use as few instructions as possible. Try the code for yourself by moving example values into R0 and R1 before the start of the rest of your program.



3. For this problem, we are going to use the stack to allow us to sum up an "array" of values. Your program should meet the following requirements:
   a. You should start with a **push** instruction that saves the link register as well as any other registers that you end up using in the rest of your code, **outside of those that are expected to be used as temporary variables (refer to the slides for that list)**
   b. Initialize a few permanent registers of your choosing to keep track of the starting location of the array (we'll use 0x20000000 as the default value), the number of items to add up (we'll use 5 as the default value), and the sum (which should start at 0).
   c. Create one loop that loads a value from the array and pushes it onto the stack. This should continue until you finish going through each item (corresponding to the number of items specified in step b)
   d. After finishing the step c loop, start another loop. Each time through the loop, pop a value off the stack that you add to the sum. This loop should once again stop once you have completed all items. You may need to be careful about which registers you are using to keep track of your "loop counter" – it is easily possible to overwrite the number you need as part of step c.
   e. At the end of both loops, a final few steps should have you put your final result into the defined "return" register, pop back any saved registers (including the link register) that you pushed at the beginning, and execute a return instruction as specified in the slides.

Your actual addition results will look fairly meaningless in this case, and that is ok. As currently written your code should also loop infinitely – that is also ok. We just want to see your thought process and ability to apply these new concepts.

Final code after following all the steps:

Editor (Ctrl-E)

Compile and Load (F5) Language: ARMv7 untitled.s

```
.global _start
_start:

DIFFOFSUMS:
    PUSH {R1, R2, R3, R4, LR}
    LDR R1, =0x20000000
    MOV R2, #5
    MOV R3, #0
    MOV R4, R2

FOR:
    CMP R2, #0
    BEQ FOR2

    LDR R0, [R1], #4
    PUSH {R0}
    SUB R2, R2, #1
    B FOR

FOR2:
    CMP R4, #0
    BEQ RESULT

    POP {R0}
    ADD R3, R3, R0
    SUB R4, R4, #1
    B FOR2

RESULT:
    MOV R0, R3
    POP {R1, R2, R3, R4, LR}
    MOV PC, LR
```