Semal Shah
I pledge my honor that I have abided by the Stevens Honor System.

Code for Taylor.s

```
.text
.global main
.extern printf

 main:
    SUB sp, sp, #16
    STR x30, [sp]
    ldr x8, =terms
    ldr x9, =x
    ldr x0, [x8] // terms
    ldr d1, [x9] // x
    fmov d0, #1.0 // keeps the running count of answer
    fmov d2, #1.0 // initializes the x keeper with 1 because of multiplication
    fmov d4, #1.0 // factorial keeper
    fmov d5, #1.0 // iteration counter
    fmov d6, #1.0 // keeper for each iteration
    fmov d9, #1.0 // a 1 to use
    mov x10, #1
    bl compute
    ldr x30, [sp]
    ldr x0, =prt_str1
    add sp, sp, #16
    bl printf
    br x30
compute:
    cmp x0, #0 // if terms is 0 then done
    beq done
    cmp x10, #1
    beq store
    fmul d2, d2, d1 // for the exponent so multiplies x by the previous values
    fmul d4, d4, d5 // multiplies the iteration by the previous factorial
    fdiv d3, d2, d4
    fadd d0, d0, d3
    fadd d5, d5, d9 // adding one to the iteration counter
    sub x0, x0, #1 // subtracting one from the terms
    bl compute
done:
    ldr x30, [sp, #8]
    br x30
store:
    str x30, [sp, #8]
    add x10, x10, #1
    bl compute
 .data
terms:
    .word 6
x:
    .double 5
prt_str1:
    .ascii "The approximation is %f \n\0"

 .end
```

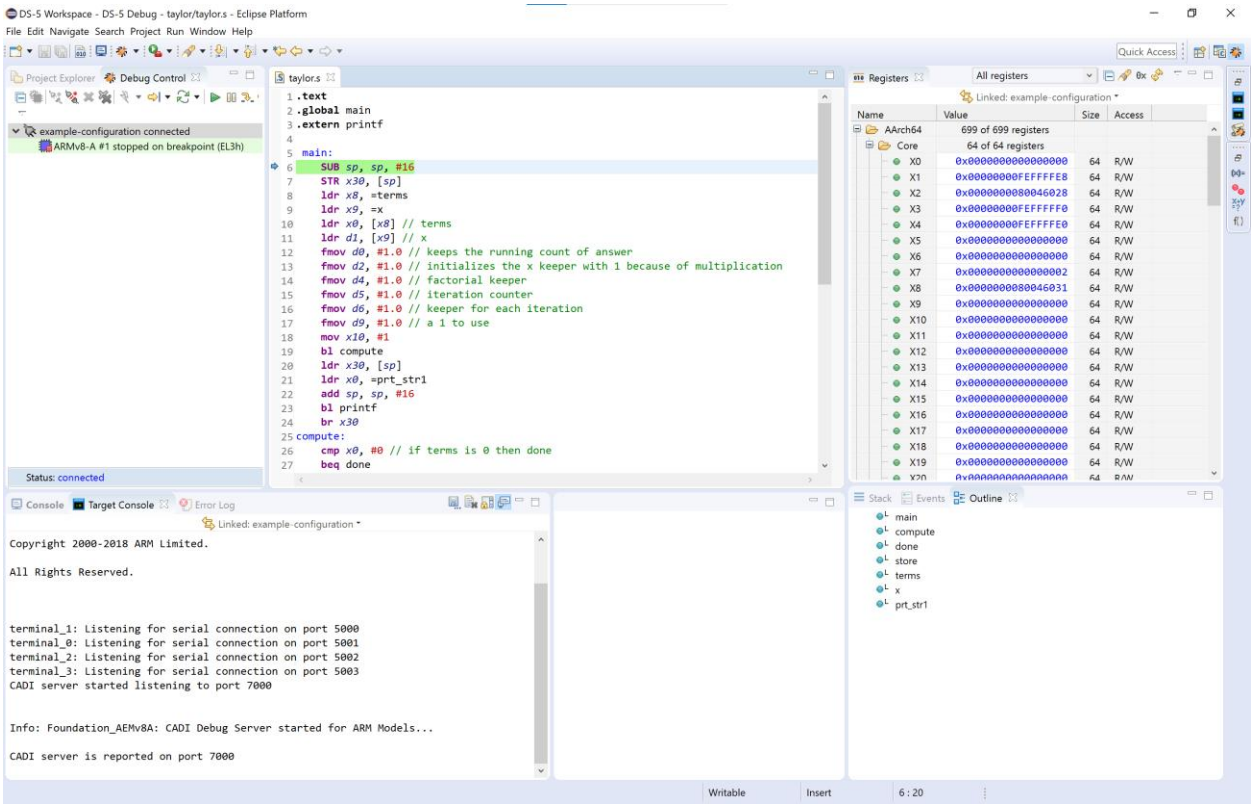
I tried to keep my approach to taylor as simple as possible, while also managing the link register and function calling using bl. In my main function I load the 2 “inputs” into x0 and d1. X0 is the terms and d1 is the x value. I then initialized many registers to keep track of what is going on in compute. I also initialized d0 to 1 because the function would return 1 in its base case where n = 0. I could not do this in compute because it is running in a loop and it would’ve reinitialized the values every time giving me the wrong answer.

In compute a problem I ran into was not being able to return to the main after my calculations were done. It would only return correctly to main if the terms were 0. So, I figured out that I needed to manage the link register and load it on to the stack before running compute in a loop. To accomplish this I initialized x10 to 1 and if it was 1 when compute ran I would store x30 on to the stack then increment x10 by 1 so it would never be run again then, call compute. In compute on line 26-27 I make sure the amount of terms left are not 0. My general thinking in compute was to go bottom up instead of top down because that would be faster and arguably easier to code. Since, every iteration you would not have to call a factorial loop or function and a power loop or function. On line 30 I multiply the whatever is in the register d2 by d1 because my d2 register is there for keeping the value of the current exponent value before that iteration. So for example in the first iteration if x was 2 it would be 1x2, next it would be 2x2, 4x2 and so on so I would not have to compute what I already computed before in terms of exponents. On line 31 I multiply the previous factorial needed by the iteration of the loop. I start the iteration variable at 1. And add one to it every time compute runs. This is for the same reason as before in where I would have the previous factorial answer so all I would have to do to get the new factorial needed for this iteration is to multiply it by the next number in line. On line 32 I now do the division of x^n/n!. Then, I add this to the running count of the answer in d0. After the calculation is done in each iteration I increment the iteration counter by 1 and subtract 1 from the amount of terms left. When terms finally equals 0 it will branch out to done and return to main.

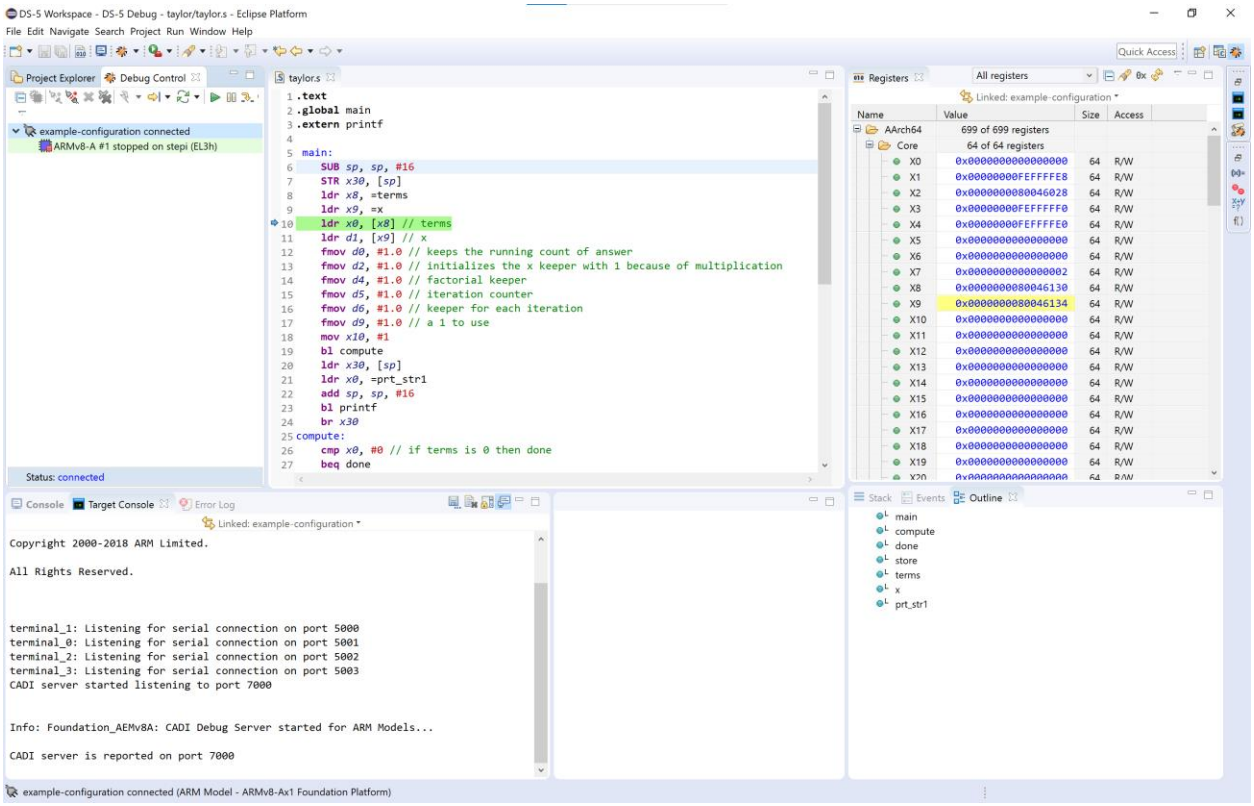
After I return to the main I load x30 from the stack then load the string I want to print in x0. I then add the spots back on the stack. Then, I execute bl printf. Finally on line 24 I do br x30 and I am finished.

Pictures of the taylor.s printing the correct value to console and link register changing are below if needed.

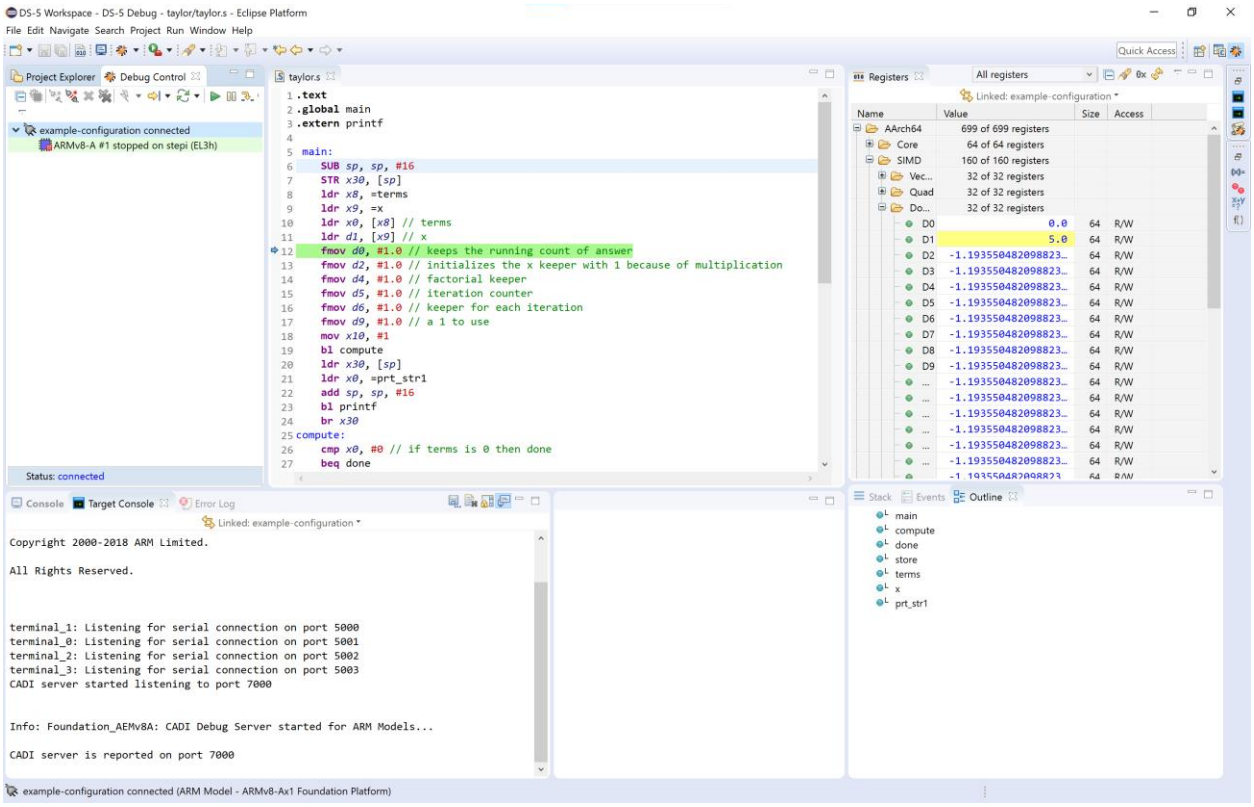
Before change in x registers



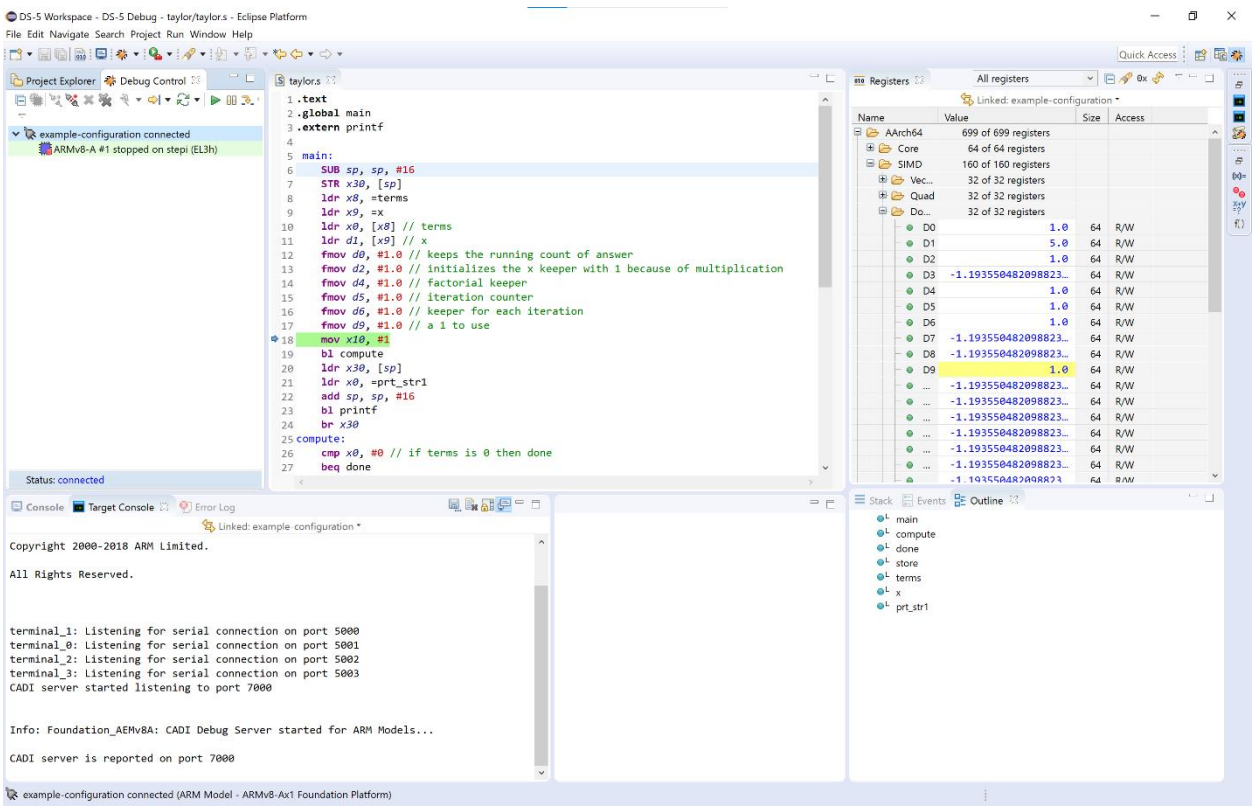

After change in x registers



Before change in d registers




After change in d registers



Printing the correct value to console. Values for x and n shown in the picture