Matthew Vaysfeld
I pledge my honor that I have abided by the Stevens Honor System.

Code:

```
/*
 * fibonacci.s
 *
 *  Created on: Oct 27, 2020
 *      Author: Matthew Vaysfeld
 */

 .text
 .global main
 .extern printf

main:
 .global fibonacci
ldr x0, =stack //load the stack pointer
mov sp, x0
sub sp, sp, #16
str x30, [sp]
bl fibonacci
.func fibonacci

fibonacci:

ldr x3, = num //address of &n
ldr x3, [x3] // derefrence
mov x1, #1 //fibCurrent
mov x2, #0 //fib previous
bl ffib // call recursive functiom
mov x1, x0
ldr x0, =string
stp x0,x1, [sp,#-16]!
bl printf
ldp x0, x1, [sp], #16

fibonacci_end:

ldr x30, [sp]
add sp, sp, #16
br x30 //return to caller

.endfunc

.func ffib


ffib:
stp x1,x2, [sp,#-16]! // save x1 and x2 on the stack
stp x3,x30, [sp,#-16]! // save x3 and x30 on the stack


ffibrec:
cmp x3, #1 // checks if x3 value is 1 (base case)
beq fib_end // if x3 is 1 then end the recursion (base case)
add x9, x1, x2 // adds fibCurrent to fibPrevious
mov x2, x1  // make fibPrevious = fibCurrent
mov x1, x9  // makes fibCurrent the sum of fibPrevious and fibCurrent
sub x3, x3, #1 //decrement n
bl ffibrec


fib_end:
mov x0, x1
ldp x3, x30, [sp], #16 //reload x3 and x30
ldp x1, x2, [sp], #16 //reload x1 and x2
br x30 //jump to caller
.endfunc

.data
string:
 .ascii "ans: %d \n\0"
num:
.dword 8
.bss
.align 8

out:
.space 8// space for result
.align 16
.space 4096 //space for stack
stack:
.space 32 //space for base address
.end

```

Code Explanation:

Note:My fibonacci starts from 1

I divided my code into 4 parts, the main function, the initial fibonacci function, and the fibonacci recursion, and the data. I will go into detail about each part.

Main:
In the main function load the stack pointer in the first line, allocate space for the stack, store the stack pointer on the link register, and branch off to the 2nd part of my code the function fibonacci.

```
12 main:
13  .global fibonacci
14 ldr x0, =stack //load the stack pointer
15 mov sp, x0
16 sub sp, sp, #16
17 str x30, [sp]
18 bl fibonacci
19 .func fibonacci
```

Initial Fibonacci:
This code is the equivalent of calling return fibo(1, 0, n); in the java code. In the function fibonacci I load the address of n into x3 and then dereference the address of n. Then I set x1 and x2 to 1 and 0, which essentially makes x1 fibCurrent and x2 fibPrevious in the java code. I then branch off to the recursive call.

After the recursive function finishes, I set x0 to the string value i specified in data and then save x0 and x. After I print the value ans: fib(n) and load the values of x0 and x1. I then go onto fibonacci_end which just loads the stack pointer onto the link register, removes the stack and then returns to the caller by doing br x30.

```
19 .func fibonacci
20
21 fibonacci:
22
23 ldr x3, = num //address of &n
24 ldr x3, [x3] // derefrence
25 mov x1, #1 //fibCurrent
26 mov x2, #0 //fib previous
27 bl ffib // call recursive functiom
28 mov x1, x0
29 ldr x0, =string
30 stp x0,x1, [sp,#-16]!
31 bl printf
32 ldp x0, x1, [sp], #16
33
34 fibonacci_end:
35
36 ldr x30, [sp]
37 add sp, sp, #16
38 br x30 //return to caller
39
40 .endfunc
```

Fibonacci recursion:

Firstly I save the values of x1, x2, x3 and x30 on the stack. Then I go onto the actual recursion, ffibrec, and I start with checking for the base case. If the x3 (n) is equal to 1, I end the function and go onto fib_end. If not, I continue the recursion and basically do what the java code does. I add FibCurrent to FibPrevious, saving that value to x9. Then I make fibPrevious = fibCurrent and fibCurrent = x9 (sum of fibPrevious and fibCurrent). I then decrement n so eventually n will reach the base case (n==1)

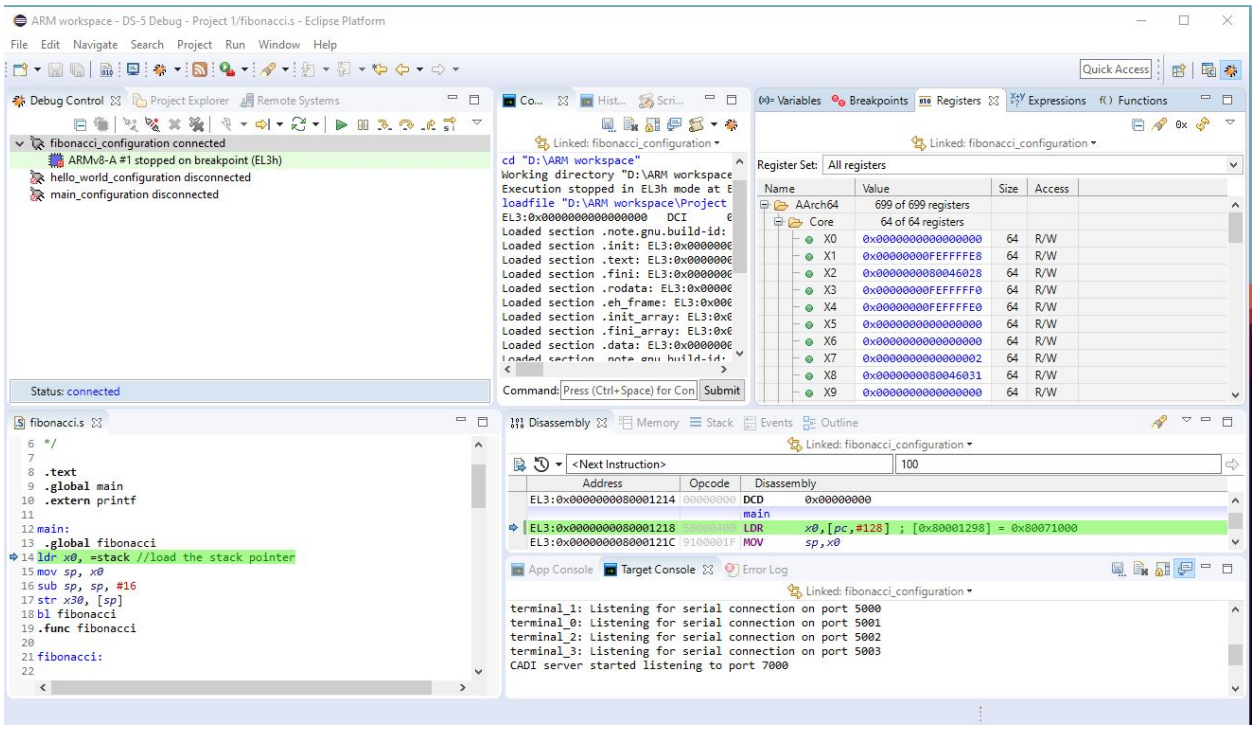
When the function ends I make the value of x1 which is FibCurrent equal to x0 and I reload x1, x2, x3 and x30. Then I branch to the link register, which takes me back to the initial fibonacci function

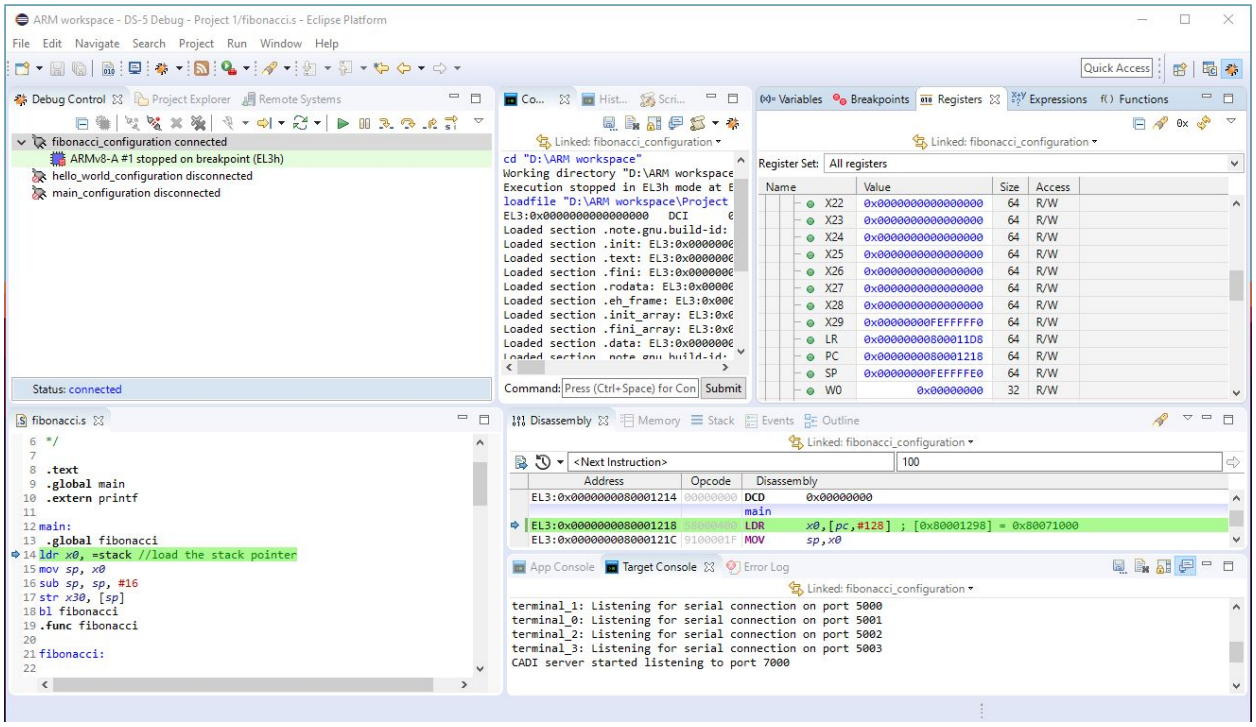
```
42 .func ffib
43
44
45 ffib:
46 stp x1,x2, [sp,#-16]! // save x1 and x2 on the stack
47 stp x3,x30, [sp,#-16]! // save x3 and x30 on the stack
48
49
50
51 ffibrec:
52 cmp x3, #1 // checks if x3 value is 1 (base case)
53 beq fib_end // if x3 is 1 then end the recursion (base case)
54 add x9, x1, x2 // adds fibCurrent to fibPrevious
55 mov x2, x1  // make fibPrevious = fibCurrent
56 mov x1, x9  // makes fibCurrent the sum of fibPrevious and fibCurrent
57 sub x3, x3, #1 //decrement n
58 bl ffibrec
59
60
61 fib_end:
62 mov x0, x1
63 ldp x3, x30, [sp], #16 //reload x3 and x30
64 ldp x1, x2, [sp], #16 //reload x1 and x2
65 br x30 //jump to caller
66 .endfunc
67
```

Data:
In the data I give the value to a string so I can print f, and set num (n) to a dword. I also allocate space for the result, stack, and base address.

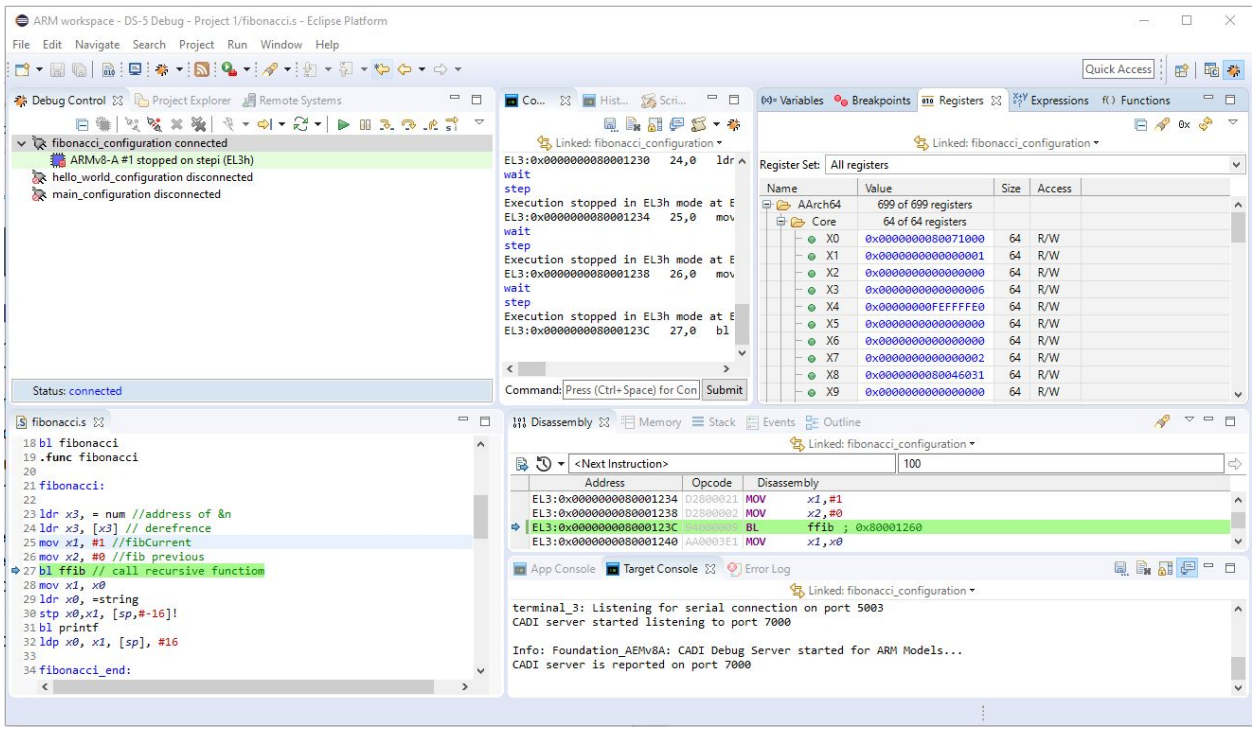
```
68 .data
69 string:
70  .ascii "ans: %d \n\0"
71 num:
72 .dword 6
73 .bss
74 .align 8
75
76 out:
77 .space 8// space for result
78 .align 16
79 .space 4096 //space for stack
80 stack:
81 .space 32 //space for base address
82 .end
83
```

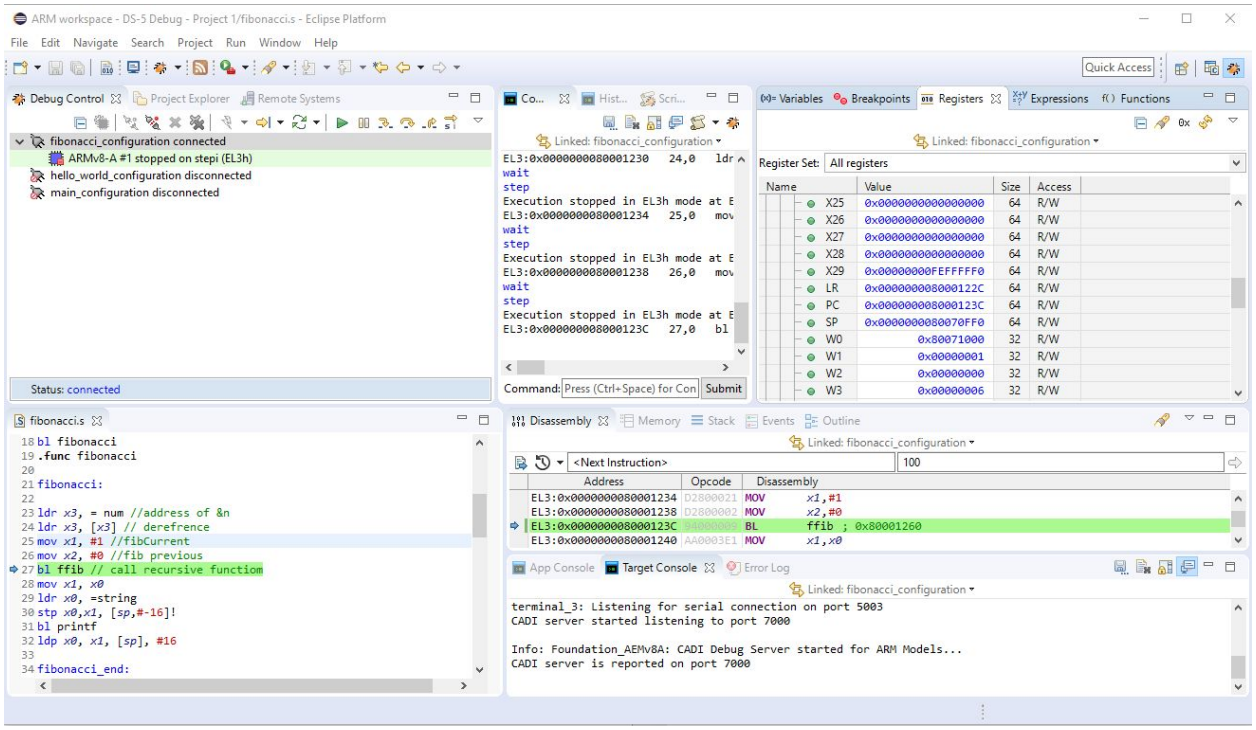
Screenshots
(First screenshots are all of the value registers, Second screenshot is the value of the Link Register)



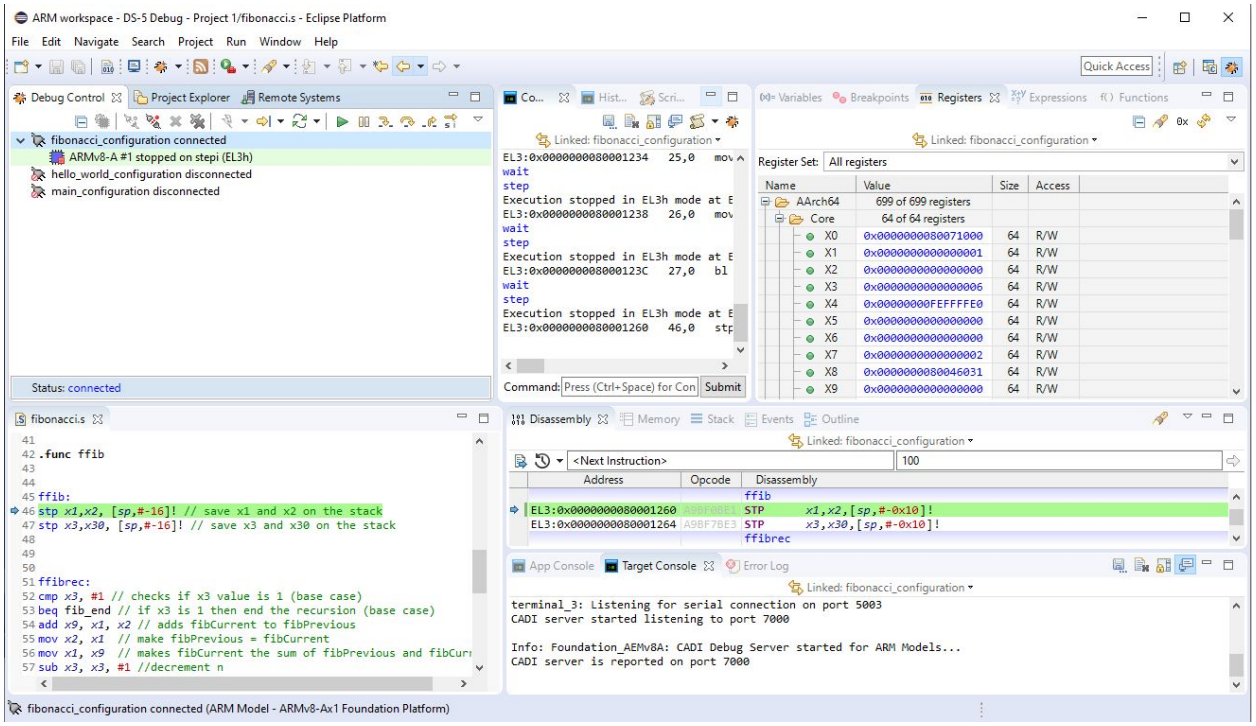


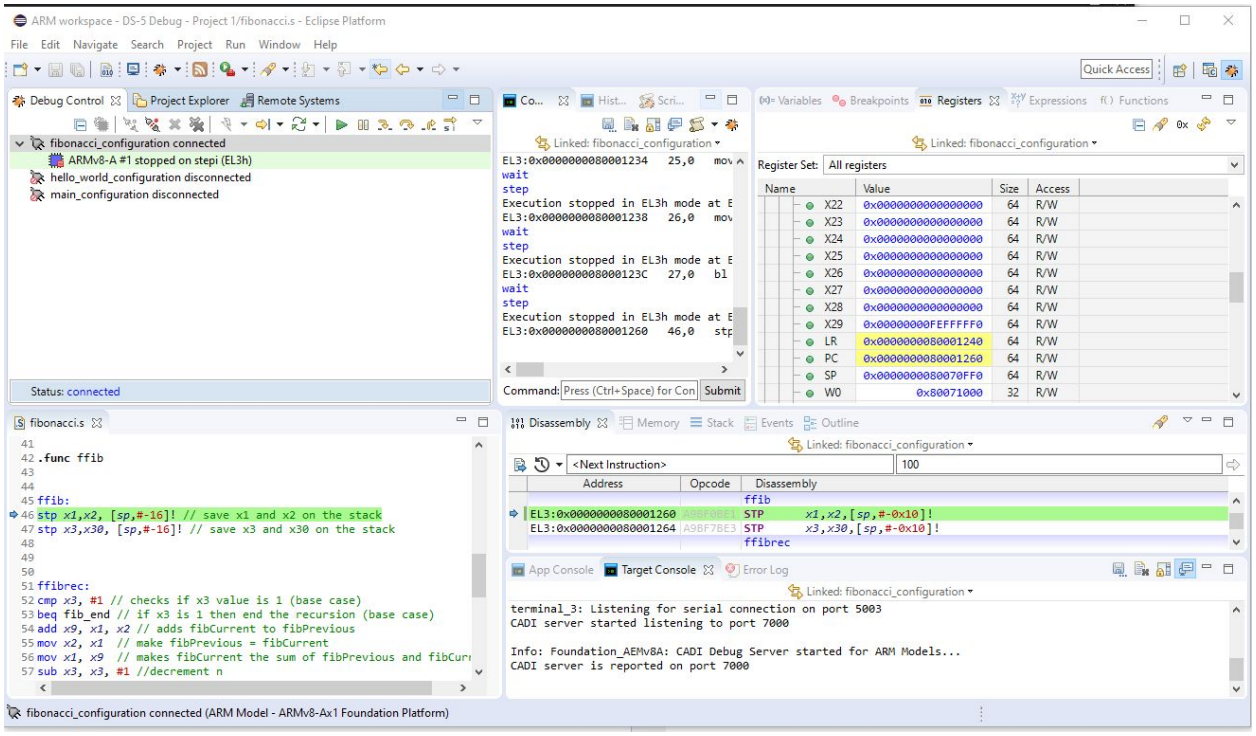
These are the register values in the beginning of the code



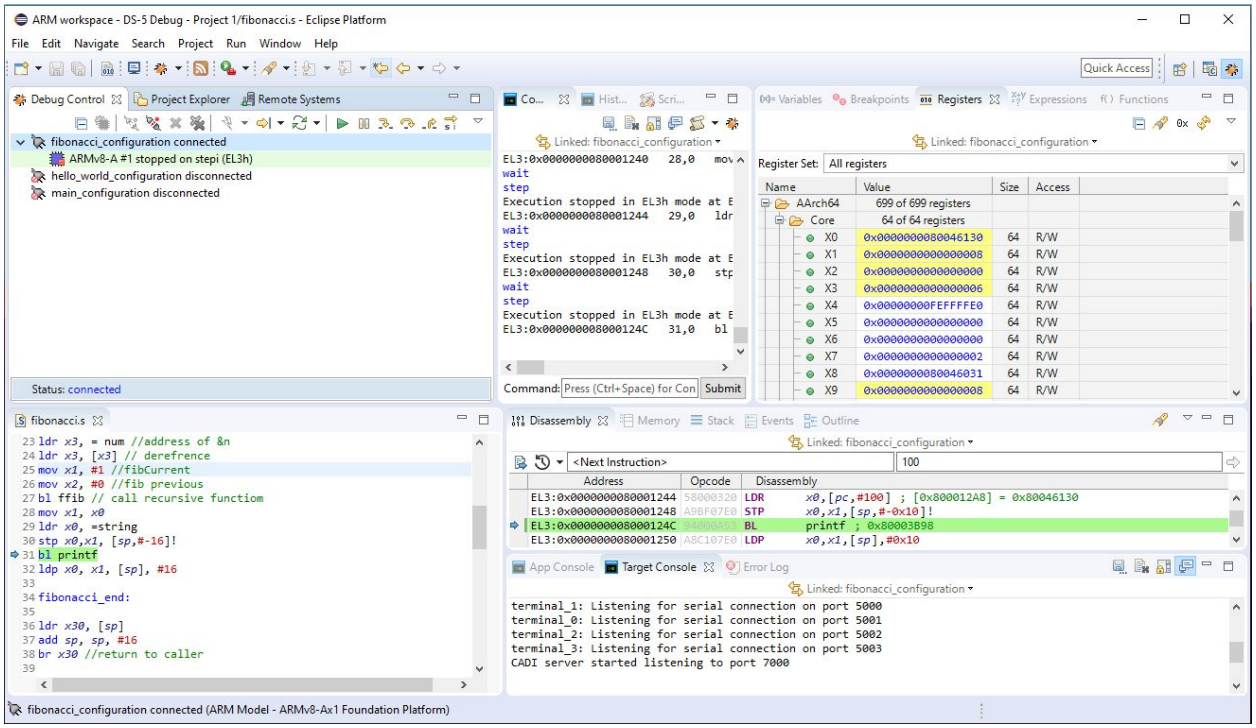


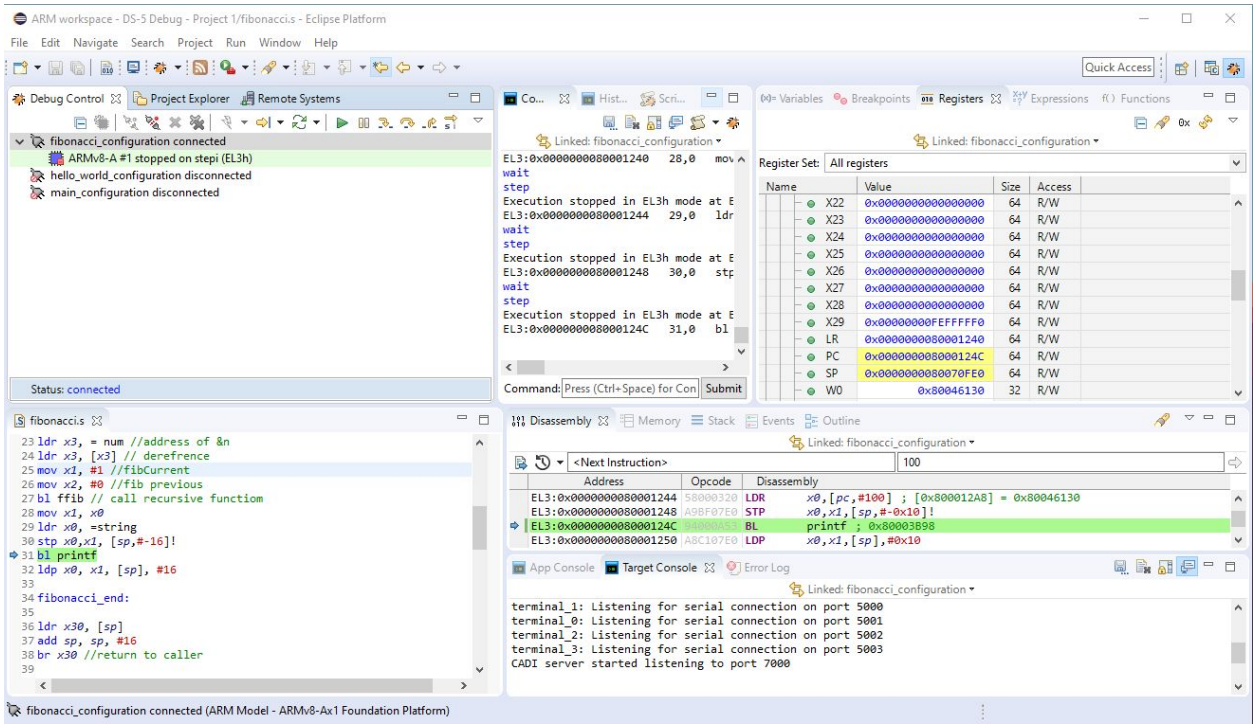
Link Register and values before the recursive call. x1 and x2(which are FibCurrent and FibPrevious) are set to 1 and 0, and x3 is set to 6, which in this case is the value of n.



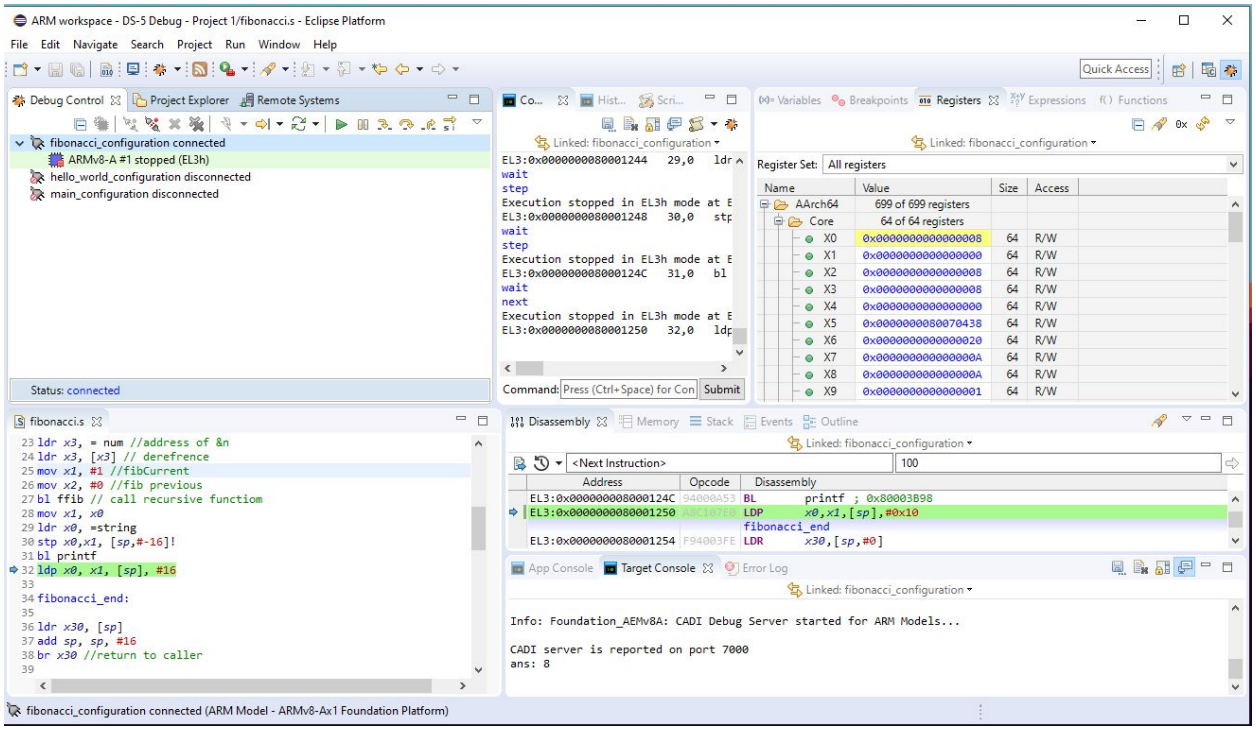


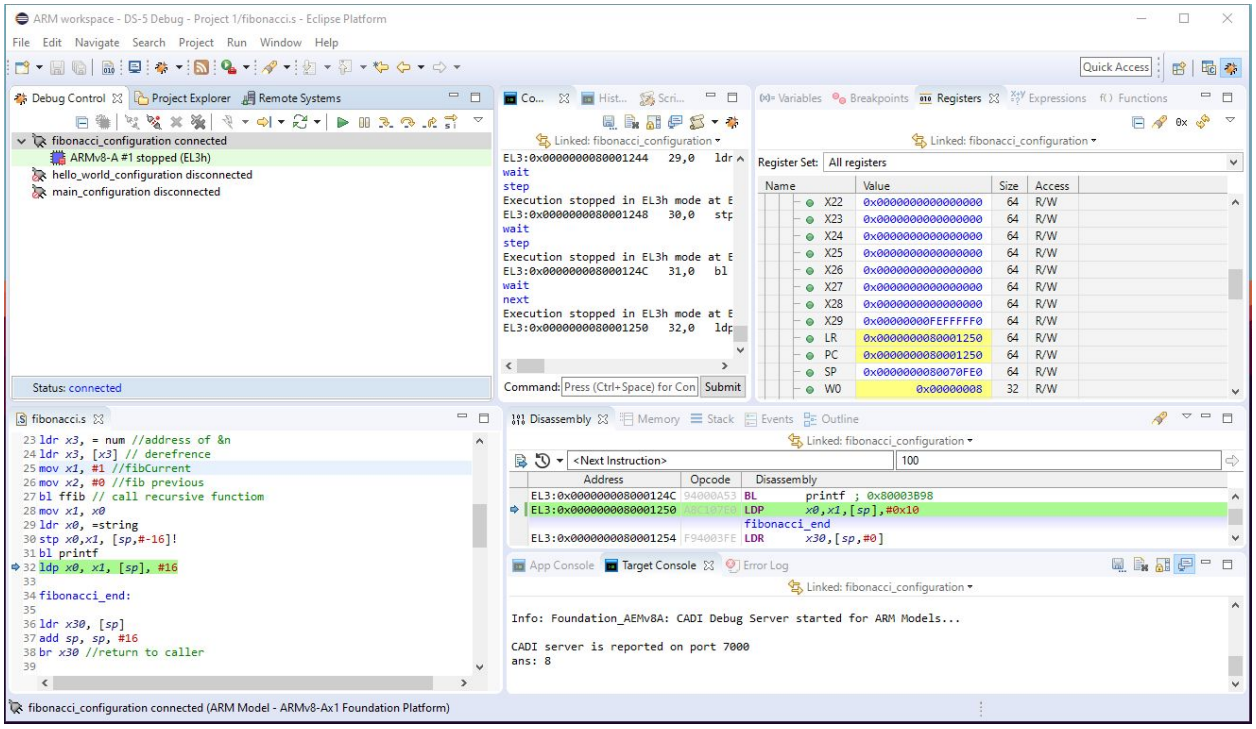
After branching to the recursion the value in the Link Register changes.



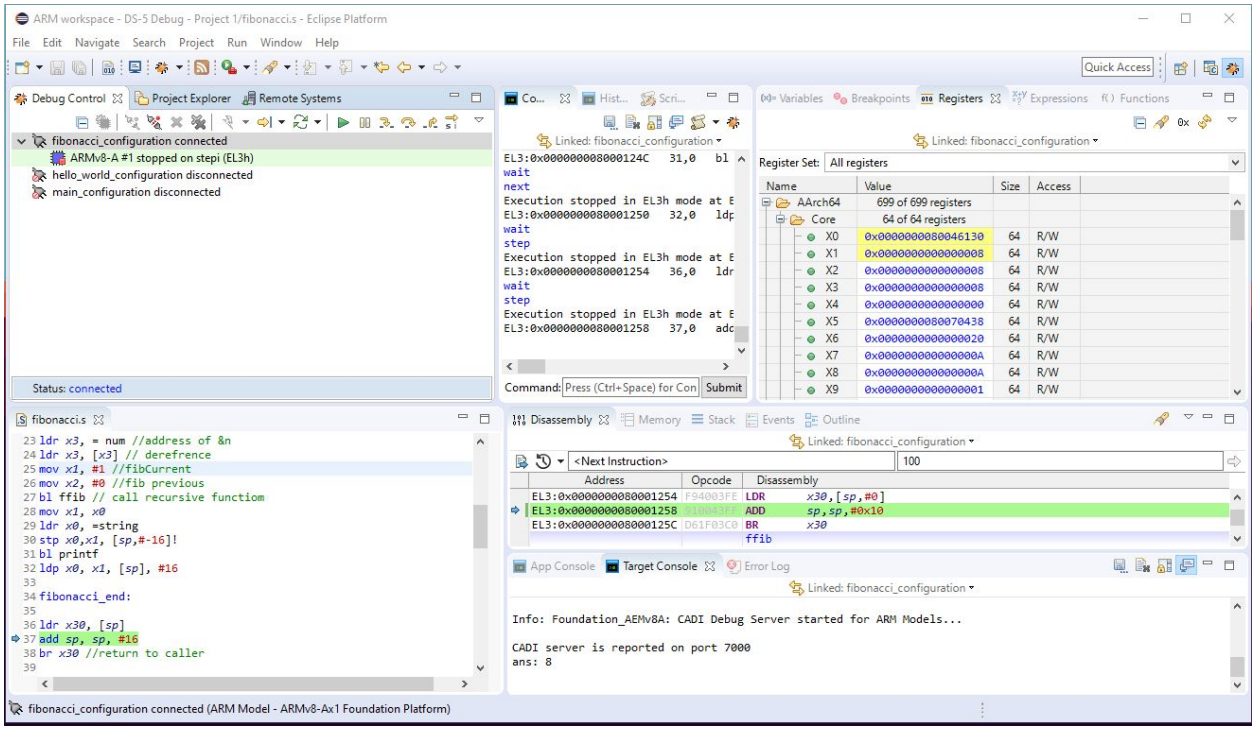


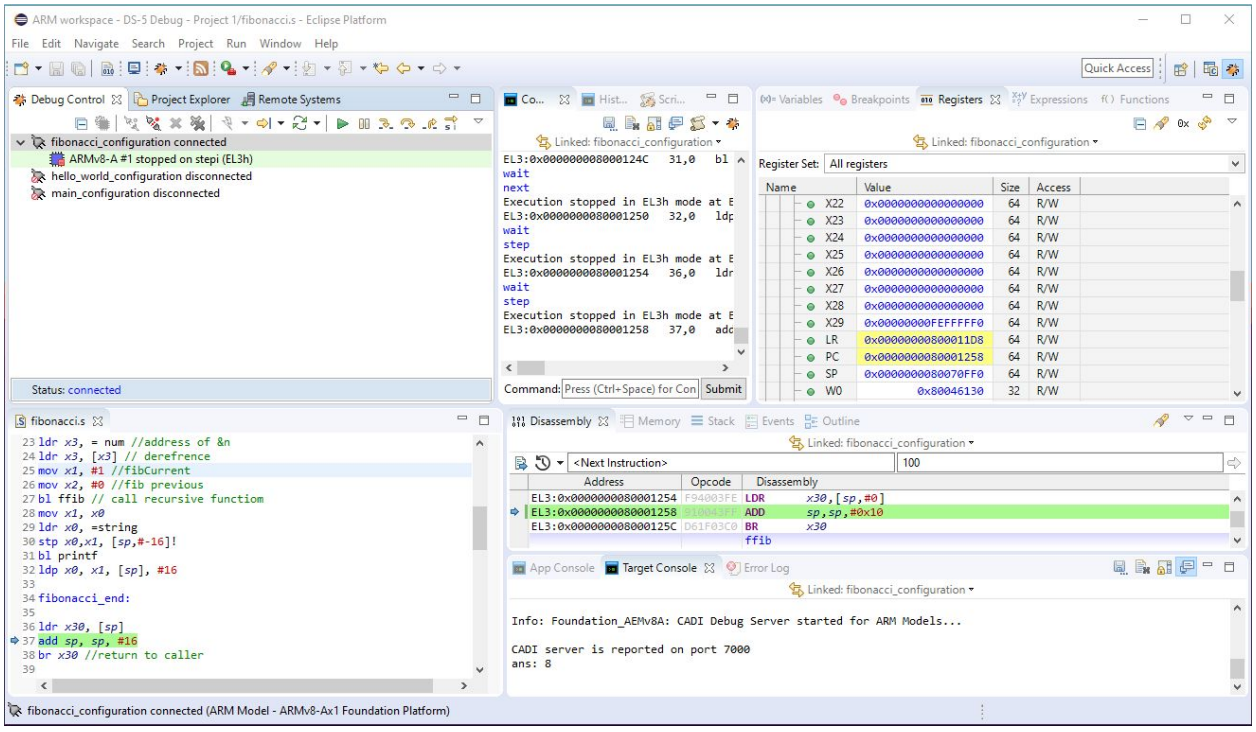
These are the registers before I printf (print out the value). The value of x1 is also 8, which is the answer to the fibonacci of 6.





This is after the printf line. The LR (x30) has changed values, and so have the rest of the registers because of the printf.





I restore x30 from the stack, which then becomes the original value of x30. The value of x1 is also 8, which is the answer to the fibonacci of 6.