Benjamin Rosen – Student ID: 1790339 – Tower of Hanoi Recursion problem

Whole Code:

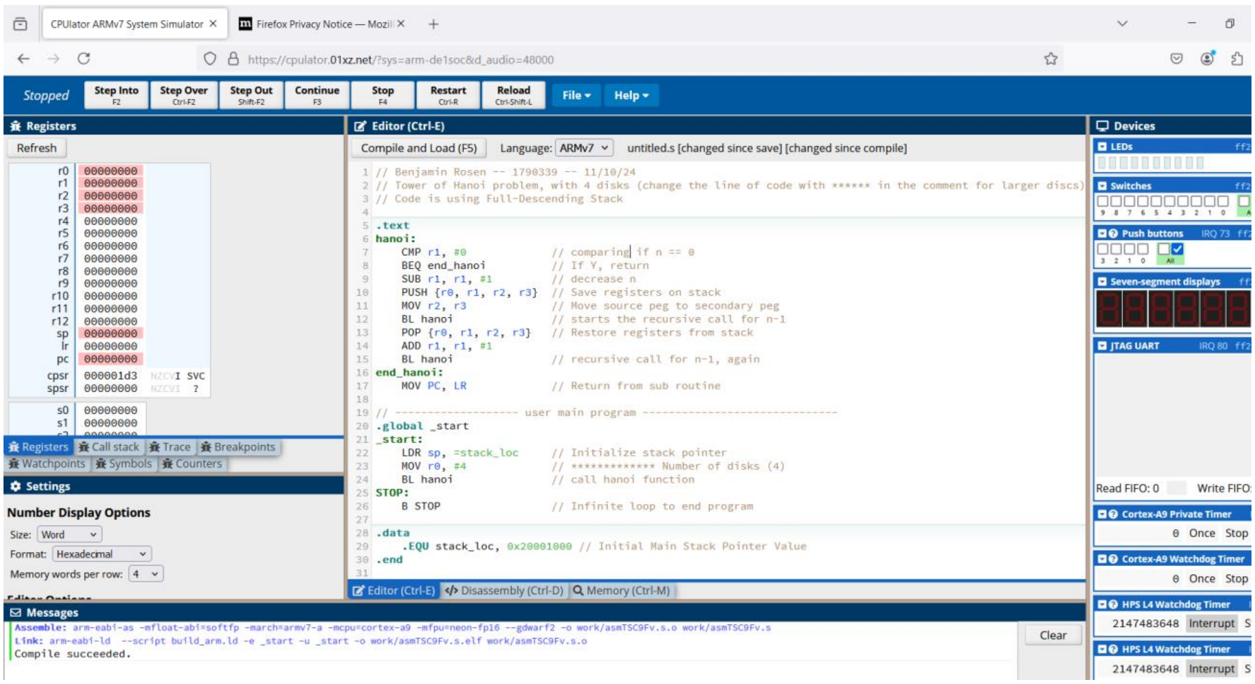
```
// Benjamin Rosen -- 1790339 -- 11/10/24
// Tower of Hanoi problem, with 4 disks (change line 23 for higher disc numbers)
// Code is using Full-Descending Stack

.text
hanoi:
//your code below
   CMP r1, #0          // check if n == 0
   BEQ end_hanoi       // if Y, return
   SUB r1, r1, #1      // decrease n
   PUSH {r0, r1, r2, r3}  // save registers on stack
   MOV r2, r3          // move source peg to secondary peg

   BL hanoi            // starts the recursive call for n-1
   POP {r0, r1, r2, r3}   // restore registers from stack
   ADD r1, r1, #1      // restore n
   BL hanoi            // recursive call for n-1, again
end_hanoi:
   MOV PC, LR          // return from sub routine
// end of your code

// ------------------- user main program ------------------------------
.global _start
_start:
   LDR sp, =stack_loc
   MOV r0, #4          // ************* Number of disks (4)
   BL hanoi            // call hanoi function

STOP:    B STOP
```

.data

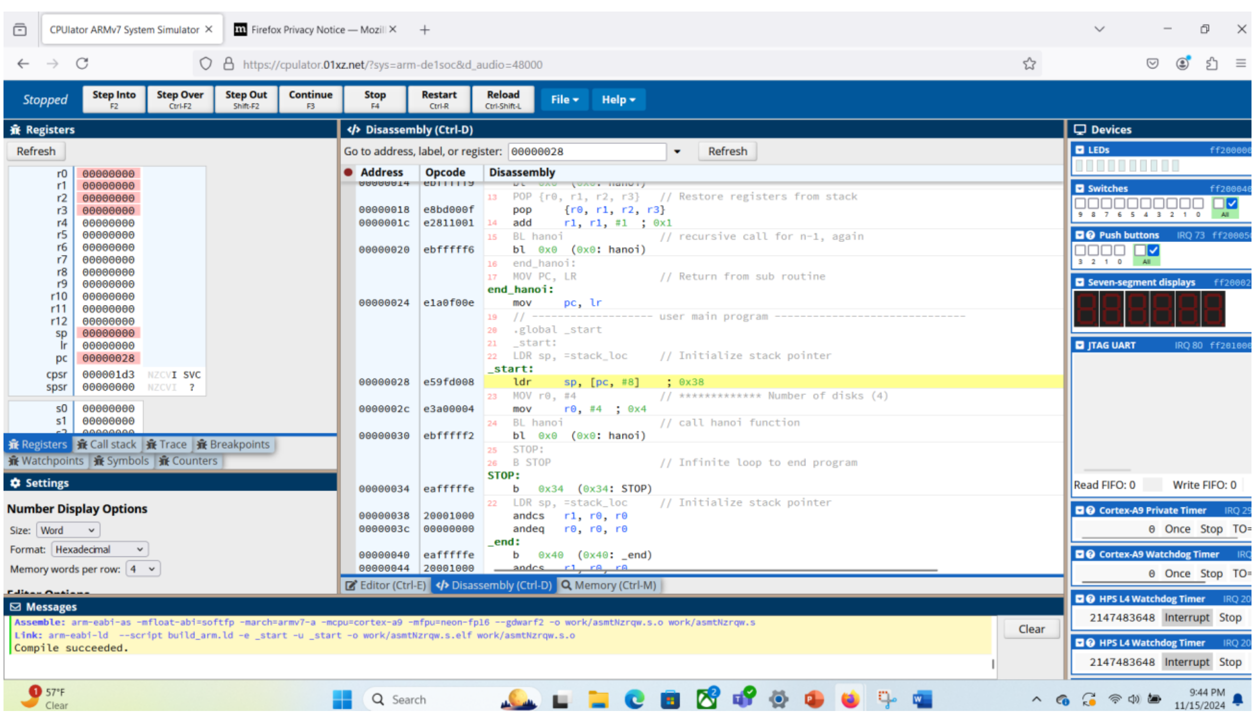
.EQU stack_loc, 0x20001000 // Initial Main Stack Pointer Value

.end

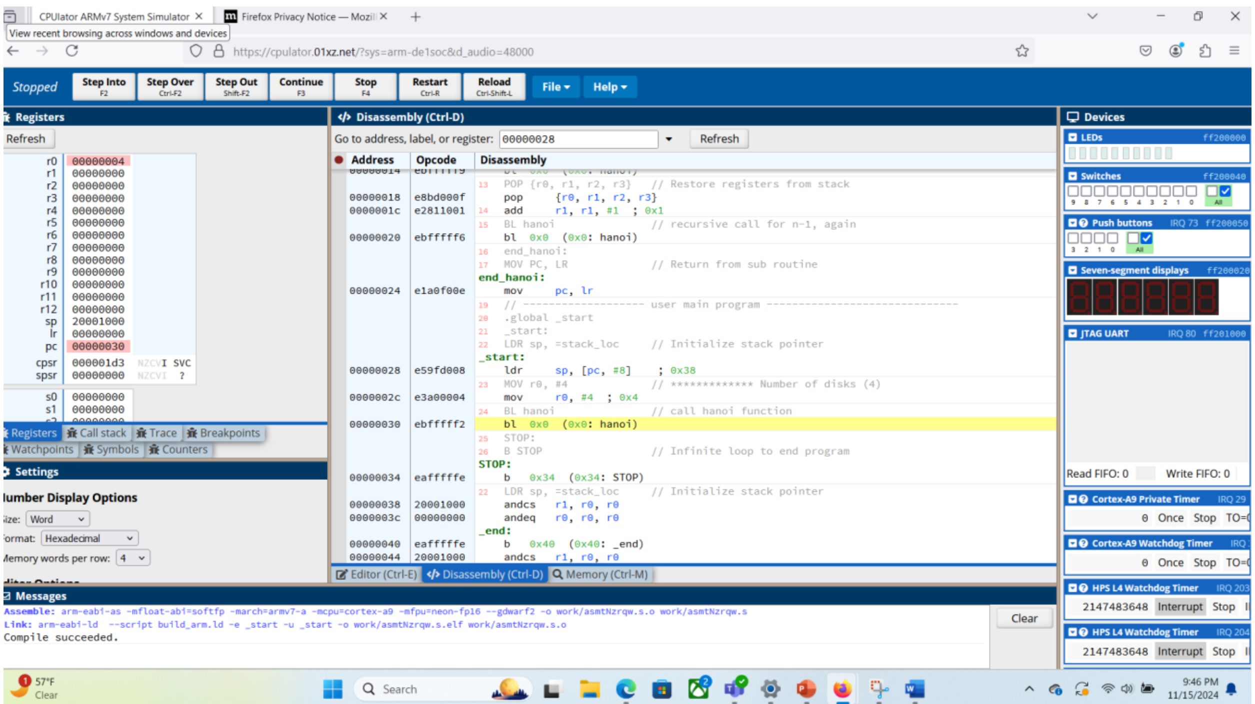
Code Overview:



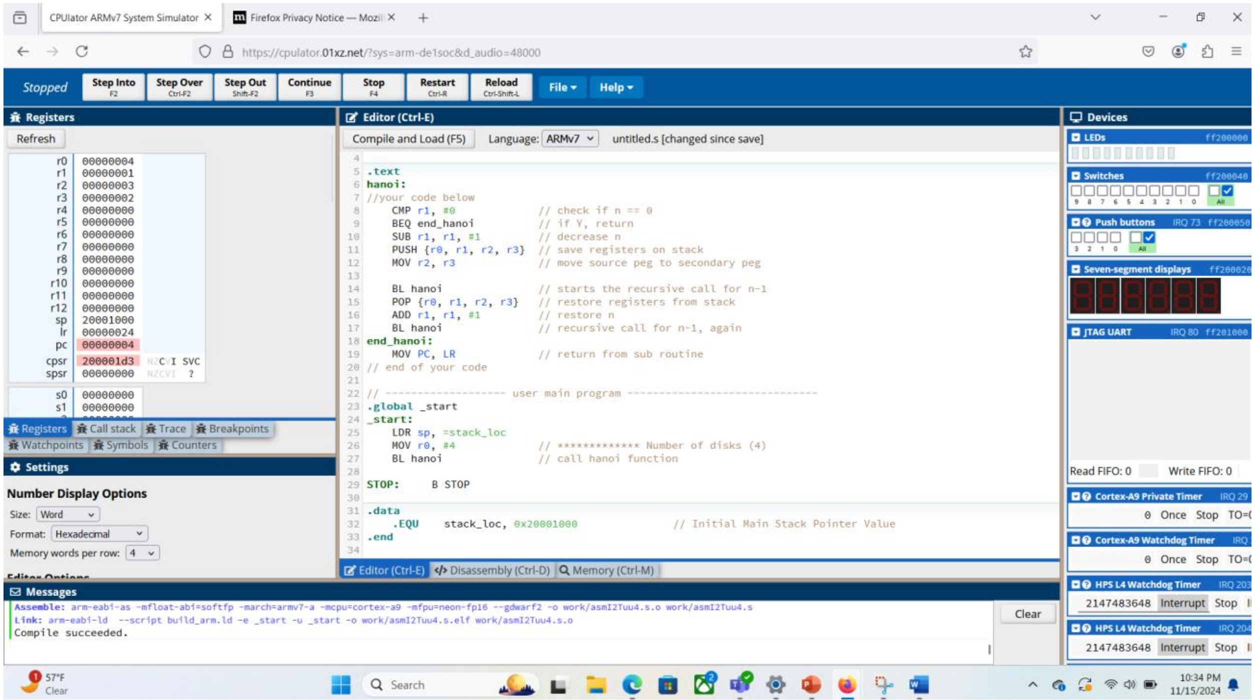
Successful Compilation Screenshot



Screenshot showing the start of the tower, stack pointer at 2001000 & r0 = 4:



Final screenshot showing the final values of the registers



Code Overview:

so this is a recursive implementation of the Tower of Hanoi problem. It uses the stack to manage the state during recursion, checking for the base case, and preserving the context with push and pop instructions.

Line by line breakdown:

CMP r0, #0: Compares the value in register r0 (which holds the number of disks) to 0.

BEQ end_hanoi: If the result of the comparison is equal (meaning there are no disks to move), it branches to the label end_hanoi, effectively ending the recursion.

SUB r0, r0, #1: Decreases the disk count in r0 by 1. This prepares for the recursive call where the function will handle one less disk.

PUSH {r0, r1, r2}: Pushes the current values of r0, r1, and r2 onto the stack to preserve their state for when the function returns.

MOV r1, r0: Moves the decremented number of disks into r1 for the next recursive call.

MOV r2, r1: This line is a placeholder;

BL hanoi: Branches to the hanoi subroutine, effectively calling it with the updated number of disks.

POP {r0, r1, r2}: Restores the saved values of r0, r1, and r2 from the stack after the recursive call returns. This ensures that the function continues with the correct state.

ADD r0, r0, #1: Increments the disk count back to the original value after the recursive call has processed the smaller problem.

BL hanoi: Calls the hanoi subroutine again to move the disks from the auxiliary peg to the target peg.

end_hanoi: This label marks the end of the recursive function.

MOV PC, LR: Moves the value in the link register (LR) to the program counter (PC), effectively returning control to the calling function.

MY NOTES:

1 **Full-Ascending Stack**: Grows upwards in memory; this is not typical for ARM.

2 **Full-Descending Stack**: Grows downwards; this fits the ARM model.

3 **Empty-Ascending Stack**: Starts at a higher address and grows upwards; not typical.

4 **Empty-Descending Stack**: Starts at a lower address and grows downwards; not typical in this context.

Hanoi code recursion problem should be done using Full-Descending Stack. This type of stack grows downwards in memory. When you push values onto the stack, the stack pointer (sp) decreases, and when you pop values, it increases. The ARM architecture typically uses a full-descending stack model, meaning that as you push data, you move the stack pointer to lower memory addresses.