

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:

CPulator ARMv7 System Simulator

Firefox Privacy Notice — Mozilla

https://cpulator.01xz.net/?sys=arm-de1soc&d\_audio=48000

Stopped Step Into Step Over Step Out Continue Stop Restart Reload File Help

Registers Refresh

Registers: r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, sp, lr, pc, cpsr, spsr, s0, s1

Editor (Ctrl-E)

Compile and Load (F5) Language: ARMv7 untitled.s [changed since save] [changed since compile]

```
1 // Benjamin Rosen -- 1790339 -- 11/10/24
2 // Tower of Hanoi problem, with 4 disks (change the line of code with ***** in the comment for larger disks)
3 // Code is using Full-Descending Stack
4
5 .text
6 hanoi:
7     CMP r1, #0 // comparing if n == 0
8     BEQ end_hanoi // If Y, return
9     SUB r1, r1, #1 // decrease n
10    PUSH {r0, r1, r2, r3} // Save registers on stack
11    MOV r2, r3 // Move source peg to secondary peg
12    BL hanoi // starts the recursive call for n-1
13    POP {r0, r1, r2, r3} // Restore registers from stack
14    ADD r1, r1, #1
15    BL hanoi // recursive call for n-1, again
16 end_hanoi:
17     MOV PC, LR // Return from sub routine
18
19 // ----- user main program -----
20 .global _start
21 _start:
22     LDR sp, =stack_loc // Initialize stack pointer
23     MOV r0, #4 // ***** Number of disks (4)
24     BL hanoi // call hanoi function
25 STOP:
26     B STOP // Infinite loop to end program
27
28 .data
29 .EQU stack_loc, 0x20001000 // Initial Main Stack Pointer Value
30 .end
31
```

Messages

Assemble: arm-eabi-as -mfloat-abi=softfp -march=armv7-a -mcpu=cortex-a9 -mfpu=neon-fp16 -gdwarf2 -o work/asmTSC9Fv.s.o work/asmTSC9Fv.s  
Link: arm-eabi-ld -script build\_arm.ld -e \_start -u \_start -o work/asmTSC9Fv.s.elf work/asmTSC9Fv.s.o  
Compile succeeded.

## Successful Compilation Screenshot

CPulator ARMv7 System Simulator

Firefox Privacy Notice — Mozilla

https://cpulator.01xz.net/?sys=arm-de1soc&d\_audio=48000

Stopped Step Into Step Over Step Out Continue Stop Restart Reload File Help

Registers Refresh

Registers: r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, sp, lr, pc, cpsr, spsr, s0, s1

Disassembly (Ctrl-D)

Go to address, label, or register: 00000028 Refresh

Address	OpCode	Disassembly
00000018	e8bd000f	pop {r0, r1, r2, r3} // Restore registers from stack
0000001c	e2811001	add r1, r1, #1; 0x1
00000020	ebfffff6	bl 0x0 (0x0: hanoi) // recursive call for n-1, again
00000024	e1a0f00e	mov pc, lr // Return from sub routine
00000028	e59fd008	ldr sp, [pc, #8]; 0x38
0000002c	e3a00004	mov r0, #4; 0x4
00000030	ebfffff2	bl 0x0 (0x0: hanoi) // call hanoi function
00000034	ebfffff2	STOP: B STOP // Infinite loop to end program
00000038	20001000	andcs r1, r0, r0
0000003c	00000000	andeq r0, r0, r0
00000040	eaffffff	b 0x40 (0x40: _end)
00000044	20001000	andcs r1, r0, r0

Messages

Assemble: arm-eabi-as -mfloat-abi=softfp -march=armv7-a -mcpu=cortex-a9 -mfpu=neon-fp16 -gdwarf2 -o work/asmTzrqw.s.o work/asmTzrqw.s  
Link: arm-eabi-ld -script build\_arm.ld -e \_start -u \_start -o work/asmTzrqw.s.elf work/asmTzrqw.s.o  
Compile succeeded.

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

The screenshot displays the CPUTator ARMv7 System Simulator interface. The top bar shows the status as "Stopped". The left panel lists registers, with r0 at 00000004 and sp at 2001000. The main panel shows the disassembly of the program, starting with the initialization of the stack pointer (sp) to 2001000 and the setting of register r0 to 4. The right panel shows the state of various hardware components, including LEDs, switches, push buttons, and seven-segment displays.

**Registers:**

Register	Value
r0	00000004
r1	00000000
r2	00000000
r3	00000000
r4	00000000
r5	00000000
r6	00000000
r7	00000000
r8	00000000
r9	00000000
r10	00000000
r11	00000000
r12	00000000
sp	2001000
lr	00000000
pc	00000030
cpsr	000001d3
spsr	00000000

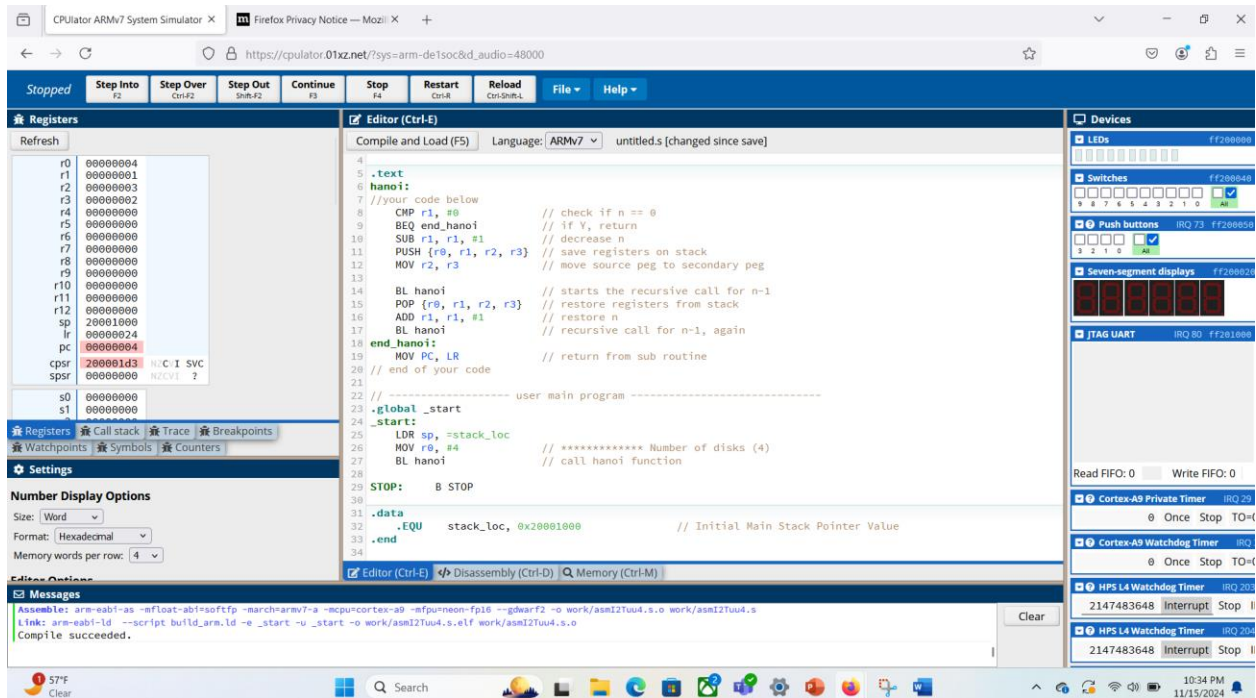
**Disassembly:**

Address	Opcode	Disassembly
00000030	e59fd008	ldr sp, [pc, #8] ; 0x38
00000032	00000000	andcs r1, r0, r0
00000034	00000000	andeq r1, r0, r0
00000036	00000000	andcs r1, r0, r0
00000038	00000000	andeq r1, r0, r0
0000003a	00000000	andcs r1, r0, r0
0000003c	00000000	andeq r1, r0, r0
0000003e	00000000	andcs r1, r0, r0
00000040	00000000	andeq r1, r0, r0
00000042	00000000	andcs r1, r0, r0
00000044	00000000	andeq r1, r0, r0
00000046	00000000	andcs r1, r0, r0
00000048	00000000	andeq r1, r0, r0
0000004a	00000000	andcs r1, r0, r0
0000004c	00000000	andeq r1, r0, r0
0000004e	00000000	andcs r1, r0, r0
00000050	00000000	andeq r1, r0, r0
00000052	00000000	andcs r1, r0, r0
00000054	00000000	andeq r1, r0, r0
00000056	00000000	andcs r1, r0, r0
00000058	00000000	andeq r1, r0, r0
0000005a	00000000	andcs r1, r0, r0
0000005c	00000000	andeq r1, r0, r0
0000005e	00000000	andcs r1, r0, r0
00000060	00000000	andeq r1, r0, r0
00000062	00000000	andcs r1, r0, r0
00000064	00000000	andeq r1, r0, r0
00000066	00000000	andcs r1, r0, r0
00000068	00000000	andeq r1, r0, r0
0000006a	00000000	andcs r1, r0, r0
0000006c	00000000	andeq r1, r0, r0
0000006e	00000000	andcs r1, r0, r0
00000070	00000000	andeq r1, r0, r0
00000072	00000000	andcs r1, r0, r0
00000074	00000000	andeq r1, r0, r0
00000076	00000000	andcs r1, r0, r0
00000078	00000000	andeq r1, r0, r0
0000007a	00000000	andcs r1, r0, r0
0000007c	00000000	andeq r1, r0, r0
0000007e	00000000	andcs r1, r0, r0
00000080	00000000	andeq r1, r0, r0
00000082	00000000	andcs r1, r0, r0
00000084	00000000	andeq r1, r0, r0
00000086	00000000	andcs r1, r0, r0
00000088	00000000	andeq r1, r0, r0
0000008a	00000000	andcs r1, r0, r0
0000008c	00000000	andeq r1, r0, r0
0000008e	00000000	andcs r1, r0, r0
00000090	00000000	andeq r1, r0, r0
00000092	00000000	andcs r1, r0, r0
00000094	00000000	andeq r1, r0, r0
00000096	00000000	andcs r1, r0, r0
00000098	00000000	andeq r1, r0, r0
0000009a	00000000	andcs r1, r0, r0
0000009c	00000000	andeq r1, r0, r0
0000009e	00000000	andcs r1, r0, r0
000000a0	00000000	andeq r1, r0, r0
000000a2	00000000	andcs r1, r0, r0
000000a4	00000000	andeq r1, r0, r0
000000a6	00000000	andcs r1, r0, r0
000000a8	00000000	andeq r1, r0, r0
000000aa	00000000	andcs r1, r0, r0
000000ac	00000000	andeq r1, r0, r0
000000ae	00000000	andcs r1, r0, r0
000000b0	00000000	andeq r1, r0, r0
000000b2	00000000	andcs r1, r0, r0
000000b4	00000000	andeq r1, r0, r0
000000b6	00000000	andcs r1, r0, r0
000000b8	00000000	andeq r1, r0, r0
000000ba	00000000	andcs r1, r0, r0
000000bc	00000000	andeq r1, r0, r0
000000be	00000000	andcs r1, r0, r0
000000c0	00000000	andeq r1, r0, r0
000000c2	00000000	andcs r1, r0, r0
000000c4	00000000	andeq r1, r0, r0
000000c6	00000000	andcs r1, r0, r0
000000c8	00000000	andeq r1, r0, r0
000000ca	00000000	andcs r1, r0, r0
000000cc	00000000	andeq r1, r0, r0
000000ce	00000000	andcs r1, r0, r0
000000d0	00000000	andeq r1, r0, r0
000000d2	00000000	andcs r1, r0, r0
000000d4	00000000	andeq r1, r0, r0
000000d6	00000000	andcs r1, r0, r0
000000d8	00000000	andeq r1, r0, r0
000000da	00000000	andcs r1, r0, r0
000000dc	00000000	andeq r1, r0, r0
000000de	00000000	andcs r1, r0, r0
000000e0	00000000	andeq r1, r0, r0
000000e2	00000000	andcs r1, r0, r0
000000e4	00000000	andeq r1, r0, r0
000000e6	00000000	andcs r1, r0, r0
000000e8	00000000	andeq r1, r0, r0
000000ea	00000000	andcs r1, r0, r0
000000ec	00000000	andeq r1, r0, r0
000000ee	00000000	andcs r1, r0, r0
000000f0	00000000	andeq r1, r0, r0
000000f2	00000000	andcs r1, r0, r0
000000f4	00000000	andeq r1, r0, r0
000000f6	00000000	andcs r1, r0, r0
000000f8	00000000	andeq r1, r0, r0
000000fa	00000000	andcs r1, r0, r0
000000fc	00000000	andeq r1, r0, r0
000000fe	00000000	andcs r1, r0, r0

**Messages:**

```
Assemble: arm-eabi-as -efloat-abi=softfp -march=armv7-a -mcpu=cortex-a9 -mfpu=neon-fp16 --gdwarf2 -o work/asmblzrqw.s.o work/asmblzrqw.s
Link: arm-eabi-ld --script build_arm.ld -e _start -u _start -o work/asmblzrqw.s.elf work/asmblzrqw.s.o
Compile succeeded.
```

## 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.