Assignment-1 (29-10-2025)

Topic: ARM assembly

Q1) Write an ARM Assembly Language Program (ALP) to check whether a given number is a palindrome.

If a palindrome r1 = 1 if not r1 = 0

CODE:

Editor (Ctrl-E)

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

```
.global main
main:
    ldr     r0, =12321  @ INPUT NO. (USING ldr pseudo instruction to be able to input upto 32 bit values. MOV only allows upto 16 bit)
    mov     r3, r0          @ copy of input for calculations
    mov     r2, #0          @ reversed number
    mov     r4, #10         @ the divisor 10

reverse_loop:
    cmp     r3, #0
    beq     check_palindrome

 @ DIVISION BY REPEATED SUBTRACTION METHOD as UDIV is not supported here

    mov     r6, #0          @ stores quotient
    mov     r9, r3          @ stores remainder

sub_loop:
    cmp     r9, r4
    blt     div_done

    sub     r9, r9, r4      @ Subtracting 10 each time
    add     r6, r6, #1      @ increment quotient by 1 for each iteration
    b       sub_loop

div_done:
    mul     r7, r2, r4
    add     r2, r7, r9      @ reversed_num = previous*10 + remainder
    mov     r3, r6          @ now quotient is the next dividend
    b       reverse_loop


check_palindrome:
    cmp     r0, r2          @ Comparing original number with reversed
    moveq   r1, #1
    movne   r1, #0

end:
    b end
```

Disassembly (Ctrl-D) Memory (Ctrl-M) Editor (Ctrl-E)

Method Used

- Implemented manual division by 10 using repeated subtraction (since DIV and UDIV was not supported in this simulator).
- The reversed number is built digit by digit by the following:

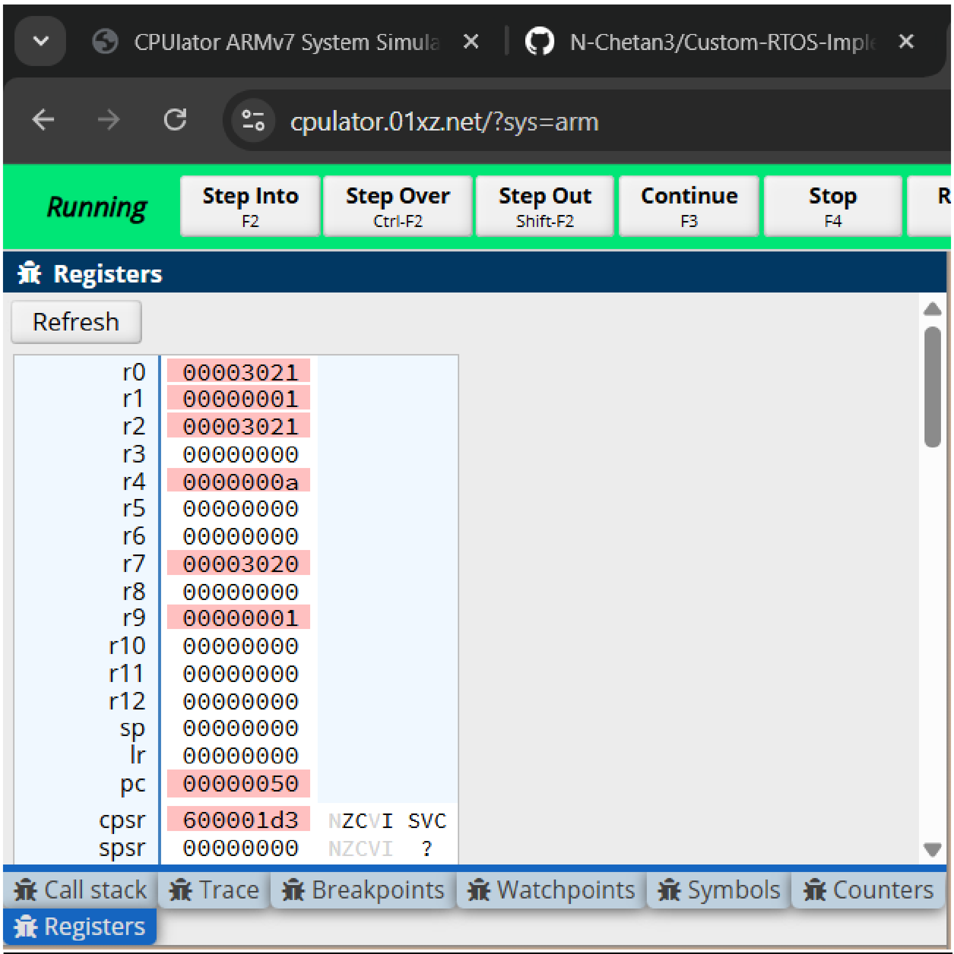
$$\text{rev} = (\text{rev} \times 10) + (\text{n} \, mod \, 10)$$

- Finally, the reversed value is compared with the original to determine if it's a palindrome.

Register	Purpose
r0	Original input number
r1	Output flag (1 = palindrome, 0 = not palindrome)
r2	Reversed number
r3	Copy of input used for calculations
r4	Constant divisor (10)
r6	Quotient (result of division)
r7	Temporary for multiplication (reversed × 10)
r9	Remainder after division

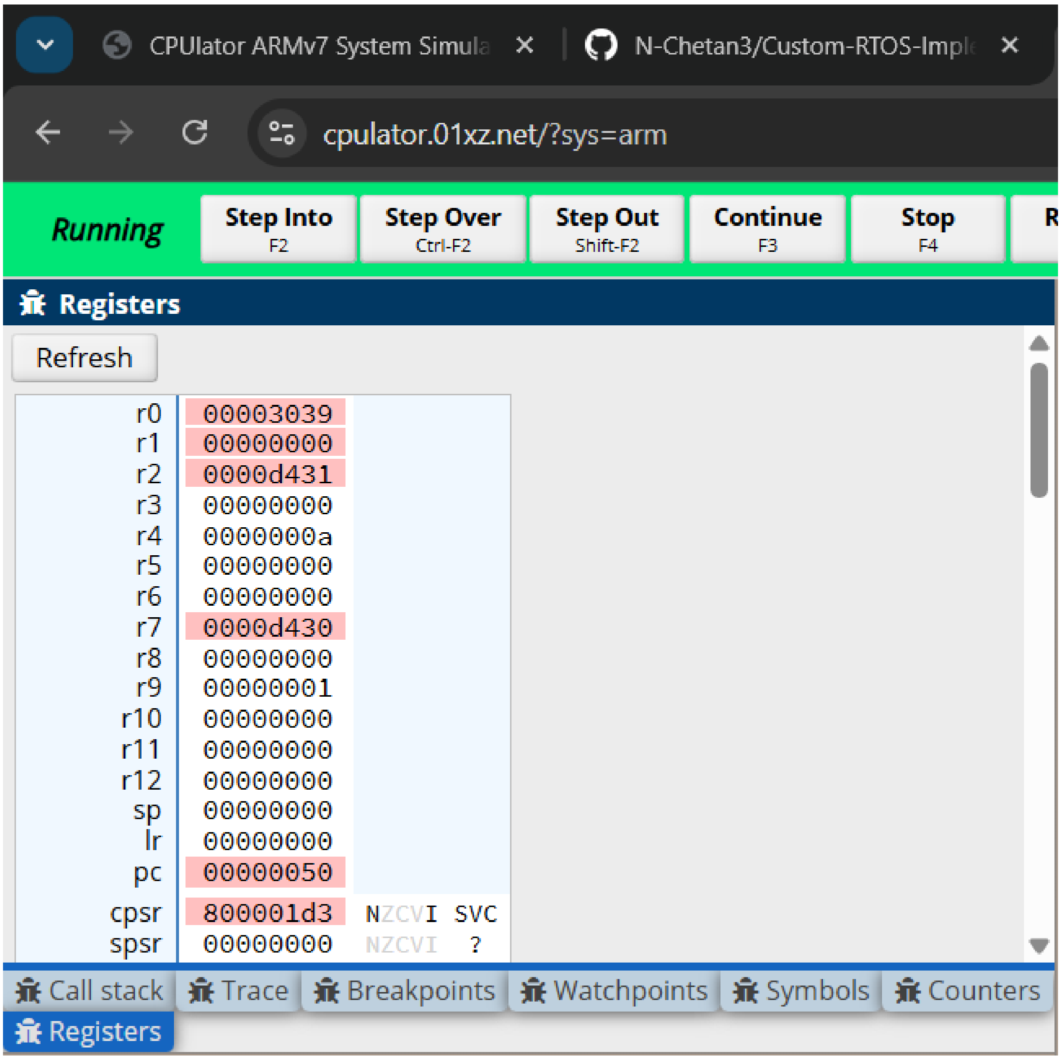
OUTPUT:

1) Palindrome no: 12321   (0x3021)



R1 stores value 1 after execution.

2) <u>**Non palindrome No. 12345 (0x3039)**</u>



R1 correctly stores value 0.

Similarly this code works for all positive numbers upto 32 bits.

Q2) Write an ARM Assembly Language Program (ALP) to generate the Fibonacci series and store the result in memory.

Code:

Editor (Ctrl-E)

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

```
.global text
text:
    mov r0,#2           @ Initializing counter r0 = 2 (first two Fibonacci numbers already stored)
    ldr r1, =fib_series @ Loading the address of fib_series array into register r1
    mov r2,#0           @ first Fibonacci number =0
    mov r3,#1           @ second Fibonacci number=1
    mov r5,#10          @ 10, to generate the 1st 10 numbers in fib_series

    str r2,[r1],#4      @ Storing F0 at fib_series[0], incrementing address by 4 bytes (32 bits)
    str r3,[r1],#4

fibonacci:
    add r4,r2,r3        @ r4 = r2 + r3 (next Fibonacci number)
    str r4,[r1],#4      @ Storing r4 in fib_series, in the memory address stored in register r1
    mov r2,r3
    mov r3,r4
    add r0,#1
    cmp r0,r5           @ Comparing counter with total terms (10)
    blt fibonacci

end:
    b end

.data
fib_series:            @ Memory space to store the Fibonacci sequence
```

Editor (Ctrl-E) Disassembly (Ctrl-D) Memory (Ctrl-M)

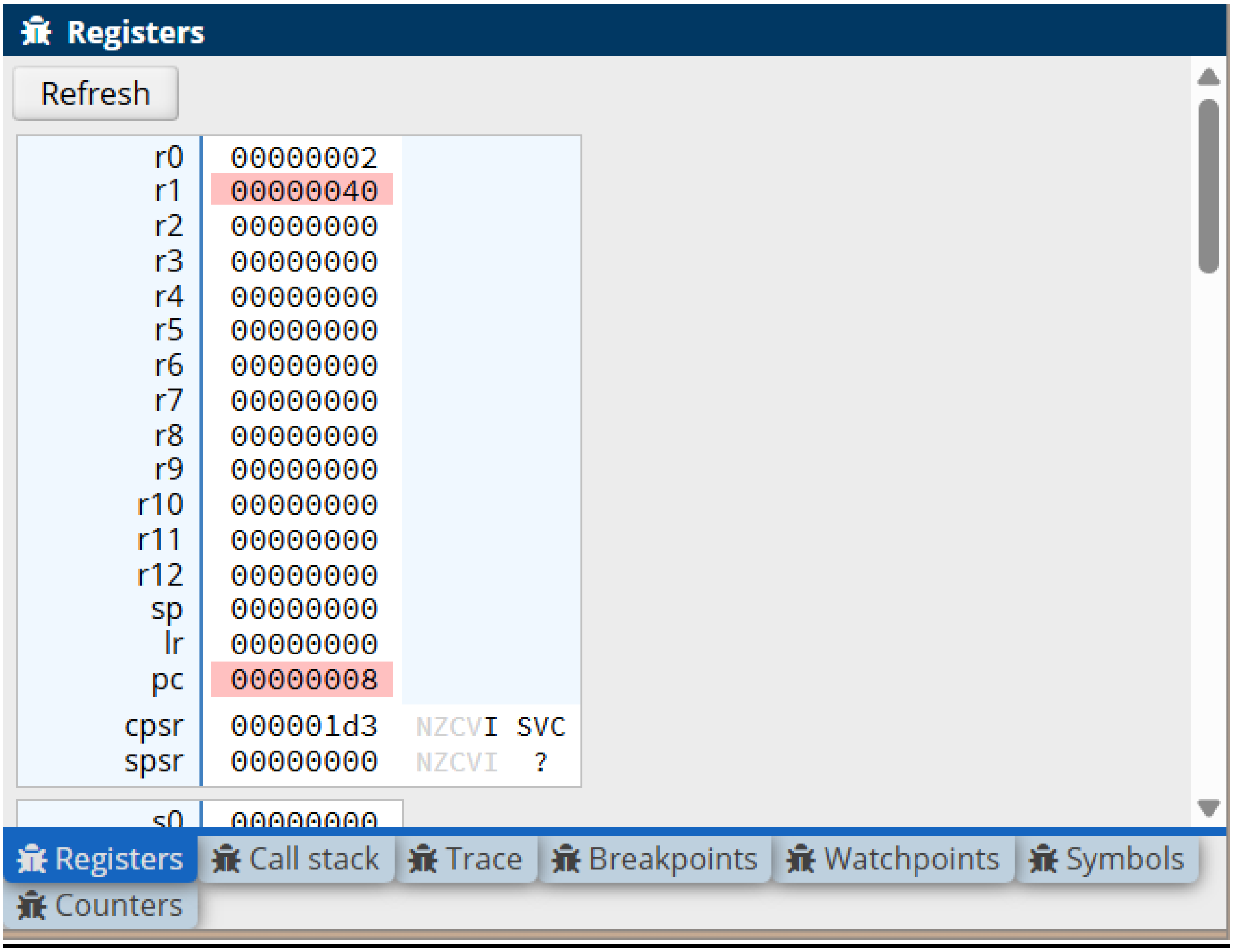
Description:

This program generates the first 10 Fibonacci numbers and stores them sequentially in memory (fib_series).

It uses simple addition-based iteration — each new term is the sum of the previous two, without using recursion or complex operations.

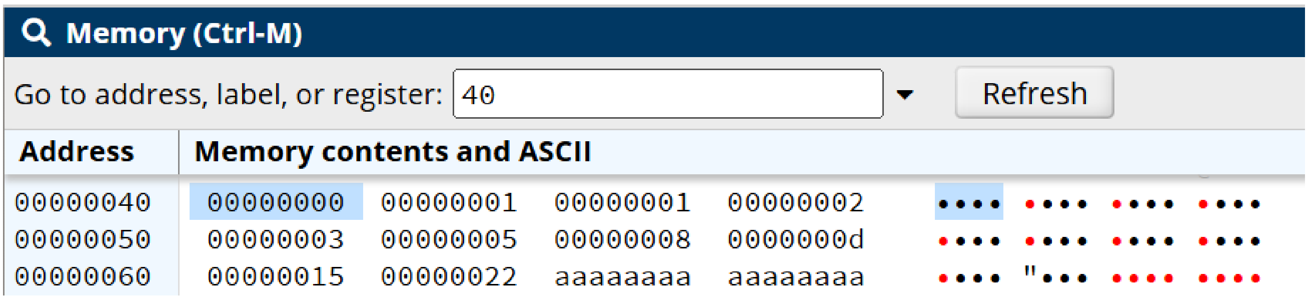
Register	**Purpose**
r0	Counter for number of terms generated
r1	Base address of fib_series array
r2	First previous Fibonacci number (F_{n-2})
r3	Second previous Fibonacci number (F_{n-1})
r4	Current Fibonacci number (F_n)
r5	Limit (total number of terms to generate = 10)

OUTPUT:



The address of the 1st element in the Fibonacci series is 0x40 as seen in the registers window (R1). It can vary .

Memory Window:



We can see that the fibonacci series has been stored in consecutive memory locations starting from 0x40 upto 0x68.