

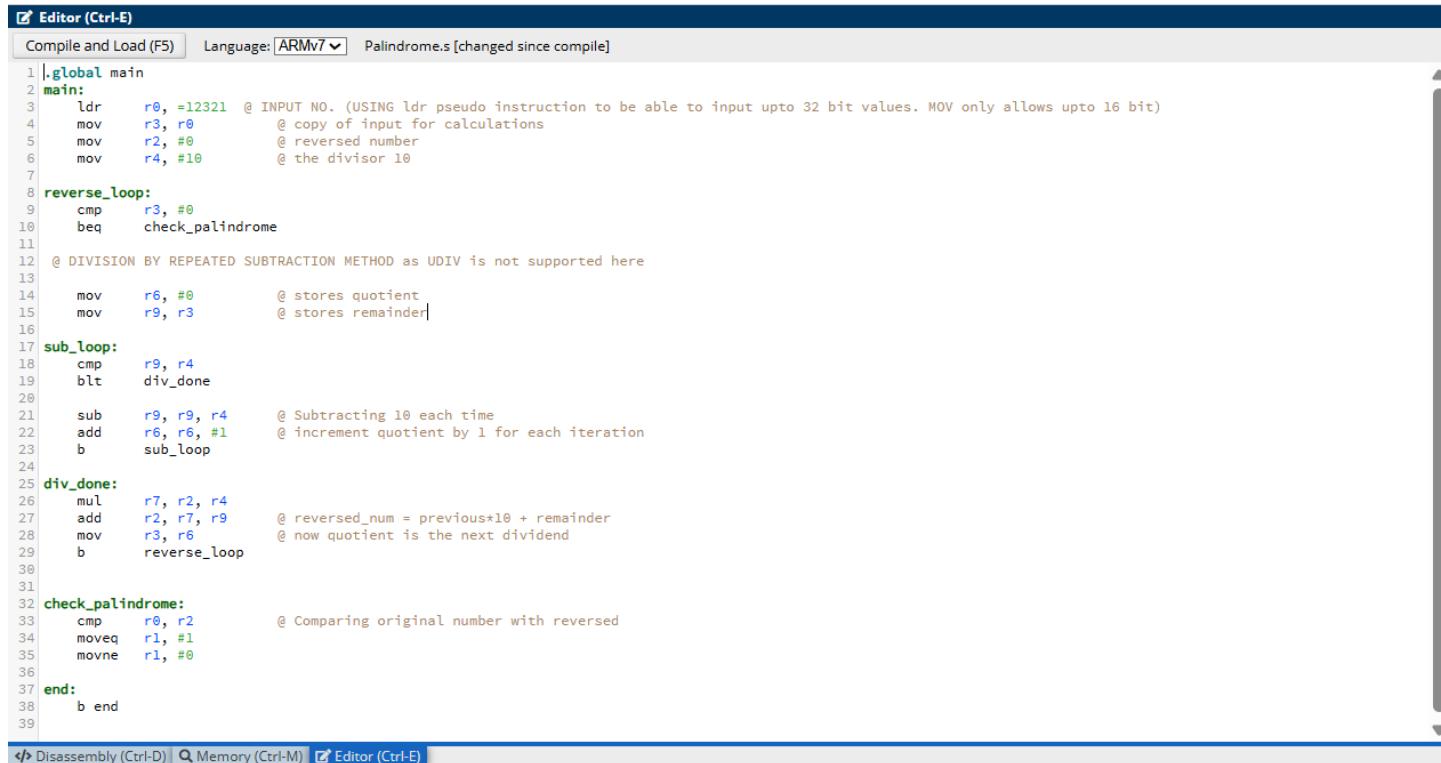
# Assignment-1 (29-10-2025)

## 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:



The screenshot shows an ARM assembly editor window. The code is written in ARM assembly language and is designed to check if the number 12321 is a palindrome. The code uses manual division by 10 through repeated subtraction and stores the reversed number in r3. It then compares r3 with the original number r0 to determine if it's a palindrome. The code includes labels for main, reverse\_loop, sub\_loop, div\_done, and check\_palindrome, along with various assembly instructions like ldr, mov, cmp, blt, sub, add, and b.

```
1 |.global main
2 |main:
3 |    ldr    r0, =12321    @ INPUT NO. (USING ldr pseudo instruction to be able to input upto 32 bit values. MOV only allows upto 16 bit)
4 |    mov    r3, r0          @ copy of input for calculations
5 |    mov    r2, #0           @ reversed number
6 |    mov    r4, #10          @ the divisor 10
7 |
8 |reverse_loop:
9 |    cmp    r3, #0
10 |   beq   check_palindrome
11 |
12 |@ DIVISION BY REPEATED SUBTRACTION METHOD as UDIV is not supported here
13 |
14 |    mov    r6, #0           @ stores quotient
15 |    mov    r9, r3            @ stores remainder
16 |
17 |sub_loop:
18 |    cmp    r9, r4
19 |    blt   div_done
20 |
21 |    sub    r9, r9, r4      @ Subtracting 10 each time
22 |    add    r6, r6, #1       @ increment quotient by 1 for each iteration
23 |    b     sub_loop
24 |
25 |div_done:
26 |    mul    r7, r2, r4
27 |    add    r2, r7, r9      @ reversed_num = previous*10 + remainder
28 |    mov    r3, r6            @ now quotient is the next dividend
29 |    b     reverse_loop
30 |
31 |
32 |check_palindrome:
33 |    cmp    r0, r2
34 |    moveq  r1, #1
35 |    movne  r1, #0
36 |
37 |end:
38 |    b end
39 |
```

### 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.

<b>Register</b>	<b>Purpose</b>
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)

Register	Value
r0	00003021
r1	00000001
r2	00003021
r3	00000000
r4	0000000a
r5	00000000
r6	00000000
r7	00003020
r8	00000000
r9	00000001
r10	00000000
r11	00000000
r12	00000000
sp	00000000
lr	00000000
pc	00000050
cpsr	600001d3 NZCVI SVC
spsr	00000000 NZCVI ?

R1 stores value 1 after execution.

## 2) Non palindrome No. 12345 (0x3039)

The screenshot shows the CPUulator ARMv7 System Simulator interface. The top bar displays the title "CPUulator ARMv7 System Simula" and the project name "N-Chetan3/Custom-RTOS-Impl". Below the title is a navigation bar with icons for back, forward, and search, followed by the URL "cpulator.01xz.net/?sys=arm". A green toolbar at the top has buttons for "Running" (highlighted), "Step Into" (F2), "Step Over" (Ctrl-F2), "Step Out" (Shift-F2), "Continue" (F3), "Stop" (F4), and a "Reset" button. The main window is titled "Registers" and contains a table of register values. The registers listed are r0 through r12, sp, lr, pc, cpsr, and spsr. The values are as follows:

Register	Value
r0	00003039
r1	00000000
r2	0000d431
r3	00000000
r4	0000000a
r5	00000000
r6	00000000
r7	0000d430
r8	00000000
r9	00000001
r10	00000000
r11	00000000
r12	00000000
sp	00000000
lr	00000000
pc	00000050
cpsr	800001d3
spsr	00000000

Below the table, the CPSR status bits are shown: NZCVI SVC. The SPSR status bits are: NZCVI ?.

At the bottom of the window, there are tabs for "Call stack", "Trace", "Breakpoints", "Watchpoints", "Symbols", "Counters", and "Registers". The "Registers" tab is currently selected.

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:

The screenshot shows an ARM assembly editor interface. The code is written in ARM assembly language and performs the following steps:

- Initializes counter `r0` to 2.
- Loads the address of the `fib_series` array into register `r1`.
- Initializes `r2` to 0 and `r3` to 1.
- Initializes `r5` to 10, the total number of terms to generate.
- Stores `F0` at `fib_series[0]`, incrementing the address by 4 bytes (32 bits).
- Enters a loop labeled `fibonacci:`:
  - Adds `r2` and `r3` to get the next Fibonacci number (`r4`).
  - Stores `r4` in `fib_series` at the address stored in `r1`.
  - Moves `r2` to `r3`.
  - Moves `r3` to `r4`.
  - Adds `r0` and `r5` to compare the counter with the total terms (10).
  - If the counter equals the limit, it branches to `end`.
- Exits the loop and branches to `end`.
- Defines the `fib_series` data section as memory space to store the Fibonacci sequence.

## 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.

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

## OUTPUT:

The screenshot shows the Registers window of a debugger. The registers listed are r0 through r12, sp, lr, pc, cpsr, and spsr. The values for r1, r10, and pc are highlighted in red. The cpsr and spsr rows show their bit fields: CPSR has 000001d3 NZCVI SVC and SPSR has 00000000 NZCVI ?. Below the registers is a memory location s0 with value 00000000. At the bottom, tabs for Registers, Call stack, Trace, Breakpoints, Watchpoints, and Symbols are visible, with Registers being the active tab.

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

Memory Window:

The screenshot shows the Memory window with address 40 selected. The table displays memory contents at addresses 0x00000040, 0x00000050, and 0x00000060. The first row shows 00000000, 00000001, 00000001, 00000002, followed by four dots and three more bytes. The second row shows 00000003, 00000005, 00000008, 0000000d, followed by four dots and three more bytes. The third row shows 00000015, 00000022, followed by four 'aaaa' bytes and three more bytes. To the right of the memory dump, there are ASCII representations of the data.

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