

# **EC-210**

# **MICROPROCESSORS LAB**

## **LAB-5**



**UTKARSH MAHAJAN 201EC164**

**ARNAV RAJ 201EC109**

Objective: To study defining memory area, constant in the assembly program.

**Exercise:**

5.2] Check if the given number is odd or even.

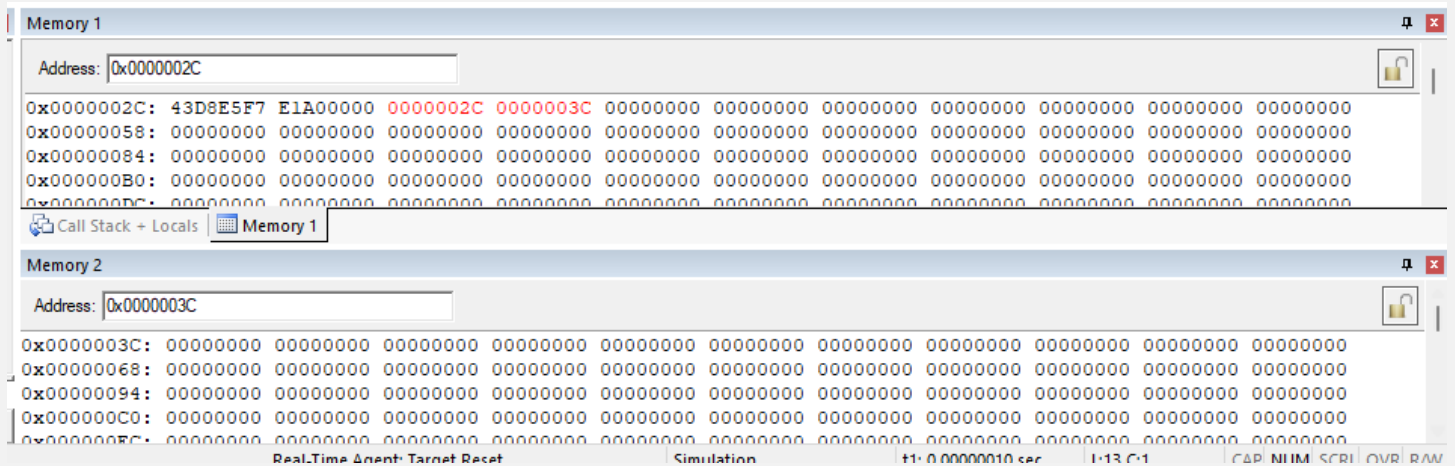
->Source Code:

```
    AREA AllocSpace, DATA, NOINIT, READWRITE
    ;Space for storing result
result SPACE 1;
;Actual Code
    AREA hmm, CODE, READWRITE
    EXPORT Reset_Handler
Reset_Handler
    ;address to the input number
    LDR R1, =inputnumber ;
    ;address to the result
    LDR R2, =result ;
    ; load the input number from memory
    LDR R3, [R1];
    ; Getting the last bit
    AND R4, R3, #1;
    ; Checking if it is zero
    CMP R4, #0;
    ;if not zero-> not divisible.
    MOVNE R4, #0xFF;
    ; storing result as 0xFF;
    STRBNE R4, [R2];
    BNE stop
    ; if zero ->divisible
    ADD R4, R4, #1;
    ; storing result as 1;
    STR R4, [R2]; storing 0
stop BAL stop
; input_number
inputnumber DCD 0x43D8E5F7;
    NOP
    END
```

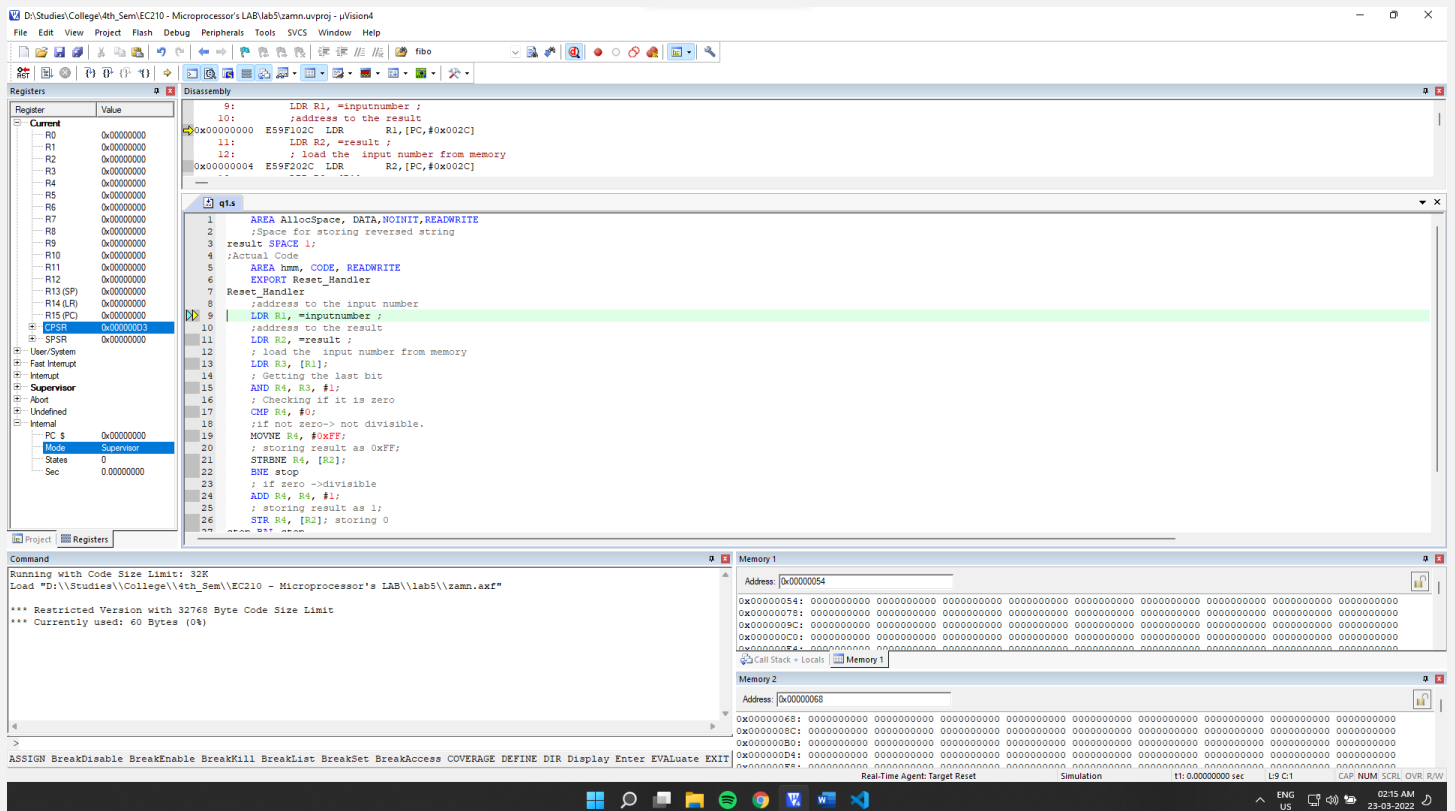
# Debugging:

Initial Memory: (after getting the address through register)

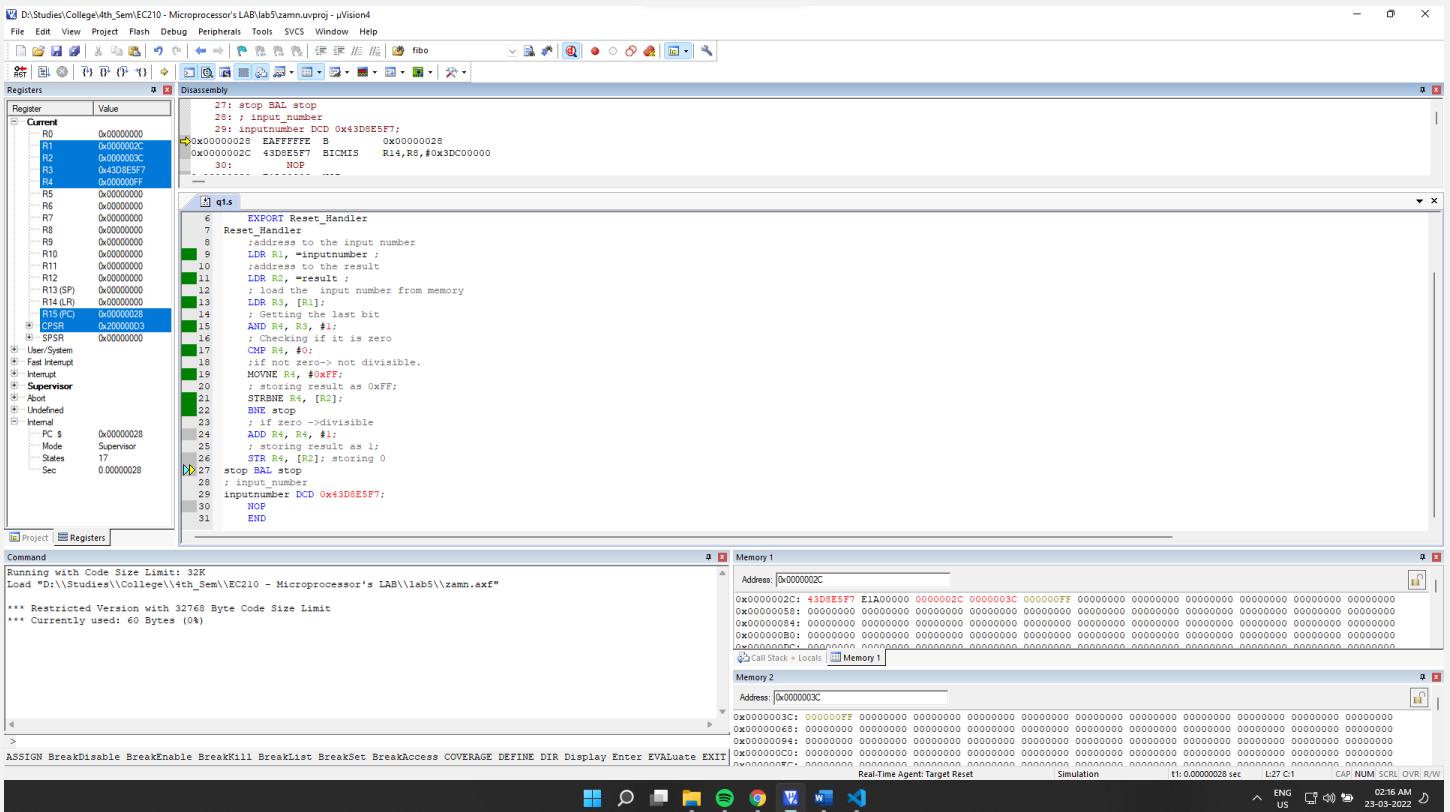
Memory 1 shows input and memory 2 will store output



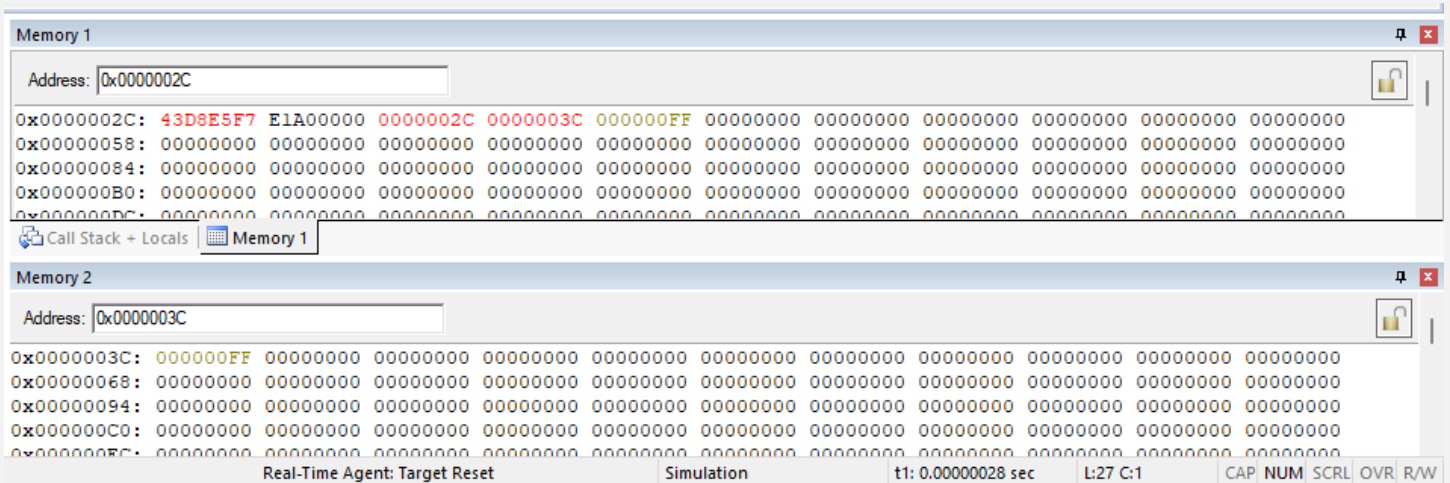
## Setup:



Final Output:



## Final Memory:



## Observation:

We can see that the output 0xFF for the given input is correct.

0x43D8E5F7 Is not divisible by 2.

For input number: 0x4FF02ED6

## Initial Memory:

Memory 1												
Address: 0x0000002C												
0x0000002C:	4FF02ED6	E1A00000	0000002C	0000003C	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x00000058:	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x00000084:	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x000000B0:	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x000000DC:	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
Call Stack + Locals Memory 1												
Memory 2												
Address: 0x0000003C												
0x0000003C:	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x00000068:	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x00000094:	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x000000C0:	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x000000FC:	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
Real-Time Agent: Target Reset Simulation t1: 0.00000010 sec L:13 C:1 CAP NUM SCRI OVR R/W												

## Final memory:

Memory 1												
Address: 0x0000002C												
0x0000002C:	4FF02ED6	E1A00000	0000002C	0000003C	00000001	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x00000058:	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x00000084:	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x000000B0:	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x000000DC:	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
Call Stack + Locals Memory 1												
Memory 2												
Address: 0x0000003C												
0x0000003C:	00000001	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x00000068:	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x00000094:	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x000000C0:	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x000000FC:	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
Real-Time Agent: Target Reset Simulation t1: 0.00000028 sec L:27 C:1 CAP NUM SCRI OVR R/W												

**Observation:** we can see the output for the given input 0x4FF02ED6 is correct. The input is divisible by 2.

5.3] Find the number of occurrences of a digit in a number.

->

Source Code:

```
AREA AllocSpace, DATA, NOINIT, READWRITE
;Space for storing result
result SPACE 1;
;Actual Code
AREA hmm, CODE, READWRITE
EXPORT Reset_Handler
Reset_Handler
```

```

;address to the input number
LDR R1, =inputnumber ;
;address to the input digit to cmp
LDR R2, =inputdigit;
;address to the result
LDR R3, =result ;
; load the input number from memory
LDR R4, [R1];
LDRB R9, [R2];
; Getting the last bit
MOV R5, R4; R5 = R4
MOV R6, #0; for quotient
;dividing and finding nth digit.
;division by the use of subtracting
; until a remainder is found.
; where remainder < 10
div SUB R5, R5, #10; subtracting by 10
CMP R5, #10; checking if remainder < 10
;incrementing iterator to store quotient
ADD R6, R6, #1;
; if remainder >=10 then loop to sub further.
BPL div
;least digit stored in R5
;checking if remainder is equal to the input digit.
CMP R9, R5;
;If yes then increment count
ADDEQ R10, R10, #1;
;comparing quotient and 10.
CMP R6, #10;
;if quotient >10 then store quotient in remainder
MOVPL R5, R6;
; && reset quotient
MOVPL R6, #0;
; && loop again
BPL div;
; the last digit will be in R6
; so compare that to input again.
CMP R6, R9; for the last digit
;if yes increment counter;
ADDEQ R10, R10, #1;
; storing result as 1;
STR R10, [R3]; storing answer
stop BAL stop

```

```
; input_number
inputnumber DCD 142534;
inputdigit DCB 4;
END
```

## Debugging:

Initial Memory: (after getting the address through register)

Memory 1 shows input number and memory 2 shows input digit while memory 3 will hold result (no of occurrences).

The screenshot displays three memory windows in the uVision IDE:

- Memory 1:** Address 0x00000054. The value at this address is 0000142534, which is the input number.
- Memory 2:** Address 0x00000068. The value at this address is 0000000004, which is the input digit.
- Memory 3:** Address 0x00000058. The value at this address is 04 00 00 00 54 00 00 00 58 00 00 00 68 00 00, which represents the result 58.

At the bottom of the IDE, the status bar shows: Real-Time Agent: Target Reset, Simulation, t1: 0.00000015 sec, L15 C:1, CAP NUM SCRL OVR R/W.

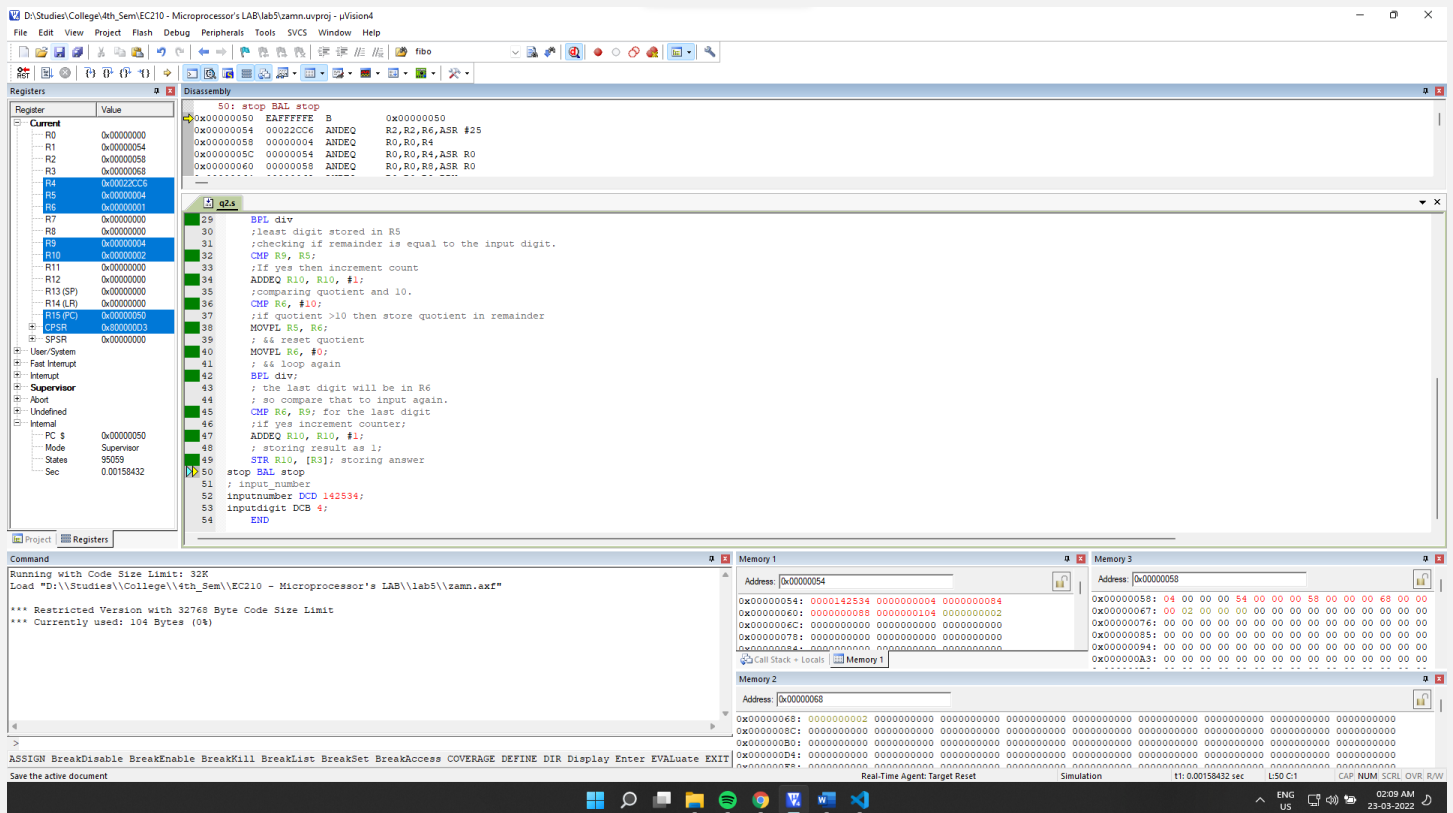
## Setup:

The screenshot shows the uVision IDE interface with the following components:

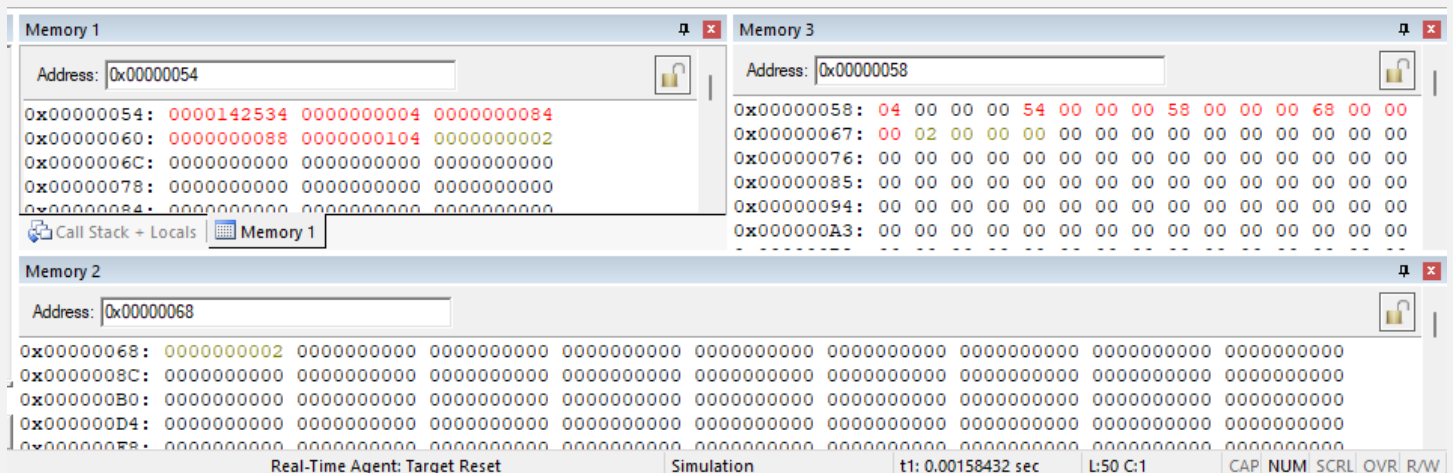
- Project:** D:\Studies\College\4th\_Sem\EC210 - Microprocessor's LAB\lab5\zamn.uvproj - uVision4
- Registers:** The 'Current' register is R0, with a value of 0x00000000. Other registers (R1-R15, SP, PC, etc.) are also listed.
- Disassembly:** The assembly code is displayed, showing the initialization of registers and memory.
- Memory:** The memory windows show the initial memory setup, with Memory 1 at address 0x00000054 and Memory 2 at address 0x00000068.

At the bottom of the IDE, the status bar shows: Real-Time Agent: Target Reset, Simulation, t1: 0.00000000 sec, L9 C:1, CAP NUM SCRL OVR R/W.

## Final Output:



## Final Memory:



## Observation:

We can see that our result stored in memory2 is 2 which is correct as digit 4 repeats twice in the input 142534.

## 5.4] Reverse the given number.

->



## Source Code:

```
    AREA AllocSpace, DATA, NOINIT, READWRITE
    ;Space for storing reversed string
array SPACE 1024;
result SPACE 1024;
;Actual Code
    AREA hmm, CODE, READWRITE
    EXPORT Reset_Handler
Reset_Handler
    ;address to the input number
    LDR R1, =inputnumber ;
    ;address for space used in operation
    LDR R2, =array;
    MOV R12, R2;
    ;address to the result
    LDR R3, =result ;
    ; load the input number from memory
    LDR R4, [R1];
    LDRB R9, [R2];
    ; Getting the last bit
    MOV R5, R4; R5 = R4
    MOV R6, #0;
    MOV R7, #0; for storing no of nos.
    ;dividing and finding nth digit.
    ;division by the use of subtracting
    ; until a remainder is found.
    ; where remainder < 10
div SUB R5, R5, #10; subtracting by 10
    CMP R5, #10; checking if remainder < 10
    ;incrementing iterator to store quotient
    ADD R6, R6, #1;
    ; if remainder >=10 then loop to sub further.
    BPL div
    ADD R7, #1; incrementing nos.
    STRB R5, [R2], #1; storing digits
    ;comparing quotient and 10.
    CMP R6, #10;
    ;if quotient >10 then store quotient in remainder
    MOVPL R5, R6;
    ; && reset quotient
    MOVPL R6, #0;
```

```

; && loop again
BPL div;
; check if quotient is zero
CMP R6, #0;
; increment no of digits counter if no
ADDNE R7, #1;
; && store it in R2 if no.
STRBNE R6, [R2]; storing last digit
;
MOV R9, #0; i for iteration
MOV R5, #0; for storing result
MOV R8, #1; for multiple of 10
MOV R4, #0; for storing digit;
MOV R6, #0;
MOV R10, #10;
;loading ith digit in R4
mult LDRB R4, [R2], #-1;
; multiplying it with 10^i
MUL R6, R4, R8;
; adding it to Result register
ADD R5, R5, R6;
;incrementing iterator
ADD R9, #1;
; multiplying multiplying register with 10.
MUL R13, R8, R10;
; storing it back into multiplkying register
MOV R8, R13;
; checking if i< n (n= no of digits)
CMP R7, R9;
; if yes then loop
BPL mult
;store final result in [R3] memory.
STR R5, [R3];
stop BAL stop
; input_number
inputnumber DCD 142534;
END

```

## Debugging:

Initial Memory: (after getting the address through register)

Memory 1 shows input number and memory 2 will hold reversed no.

Memory 1

Address: 0x00000098

0x00000098:	0000142534	0000000152	0000000168	0000001192	0000000000	0000000000	0000000000	0000000000	0000000000
0x000000BC:	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000
0x000000E0:	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000
0x00000104:	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000
0x00000128:	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000

Call Stack + Locals Memory 1

Memory 2

Address: 0x000004A8

0x000004A8:	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000
0x000004CC:	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000
0x000004F0:	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000
0x00000514:	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000
0x00000538:	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000

Real-Time Agent: Target Reset Simulation t1: 0.00000017 sec L:17 C:1 CAP NUM SCRL OVR R/W

Setup:

D:\Studies\College\4th\_Sem\EC210 - Microprocessor's LAB\lab5\zamn.uvproj - uVision

File Edit View Project Flash Debug Peripherals Tools SVCS Window Help

Registers

Register	Value
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000000
PCSR	0x00000000
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC \$	0x00000000
Mode	Supervisor
States	0
Sec	0.00000000

Disassembly

```
10: LDR R1, =inputnumber ;
11: ;address for space used in operation
0x00000000 E59F1094 LDR R1,[PC,#0x0094]
12: LDR R2, =array;
0x00000004 E59F2094 LDR R2,[PC,#0x0094]
13: MOV R12, R2;
14: ;Space for storing reversed string
15: array SPACE 1024;
16: result SPACE 1024;
17: ;Actual Code
18: AREA hmn, CODE, READWRITE
19: EXPORT Reset_Handler
20: Reset_Handler
21: ;address to the input number
22: LDR R1, =inputnumber ;
23: ;address for space used in operation
24: LDR R2, =array;
25: MOV R12, R2;
26: ;address to the result
27: LDR R3, =result ;
28: ; load the input number from memory
29: LDR R4, [R1];
30: LDRB R9, [R2];
31: ; Getting the last bit
32: MOV R5, R4; R5 = R4
33: MOV R6, #0;
34: MOV R7, #0; for storing no. of nos.
35: ;dividing and finding nth digit.
36: ;division by the use of subtracting
37: ; until a remainder is found.
38: ; where remainder < 10
39: SUB R5, R5, #10; subtracting by 10
40: ; where remainder < 10
```

Command

Running with Code Size Limit: 32K  
Load "D:\Studies\College\4th\_Sem\EC210 - Microprocessor's LAB\lab5\zamn.axf"

\*\*\* Restricted Version with 32768 Byte Code Size Limit  
\*\*\* Currently used: 168 Bytes (0K)

Memory 1

Address: 0x00000048

0x00000048:	1526726645	3814064128	0310865921	0365060096	3818950656	3818934272	3818946561	3818930176	3818938368
0x0000006C:	3818954762	3830595585	3758491796	3766833158	3800666113	3758951064	3785392141	3780575241	1526726647
0x00000090:	3850588160	3942645758	0000142534	0000000152	0000000168	0000000192	0000000000	0000000000	0000000000
0x000000B4:	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000
0x000000D8:	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000

Memory 2

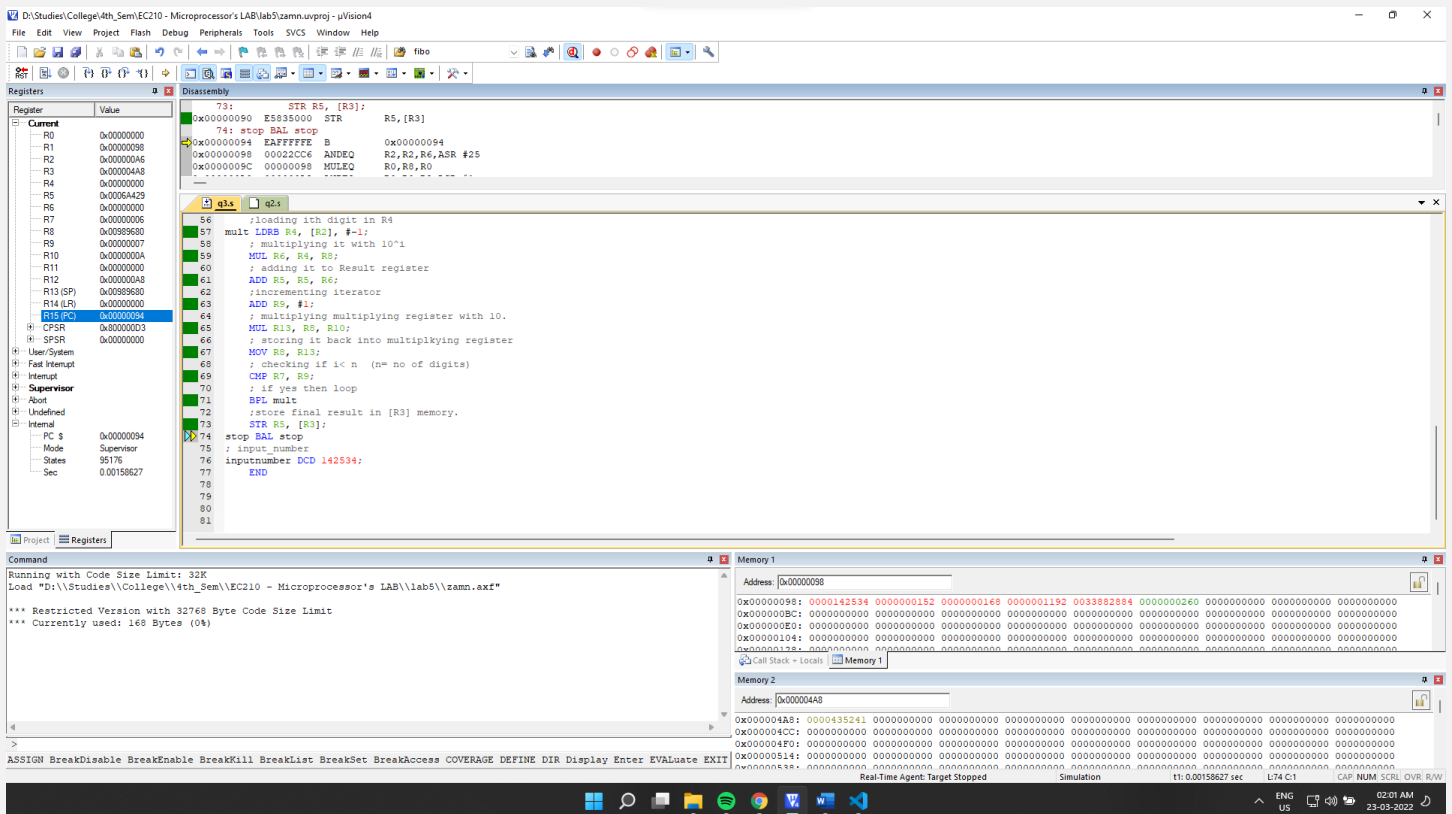
Address: 0x00000054

0x00000054:	00 60 C2 15 00 90 A0 E3 00 50 A0 E3 01 80 A0 E3 00 40 A0 E3 00 60 A0 E3 0A A0 A0 E3 01 40 52 E4 94 08
0x00000076:	06 E0 06 50 88 E0 01 90 59 E2 98 0A 0D E0 80 A0 E1 09 00 57 E1 F7 FF 5A 00 50 83 E5 FE FF EA
0x00000098:	C6 2C 02 00 98 00 00 00 A8 00 00 00 A8 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000000BA:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000000DC:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

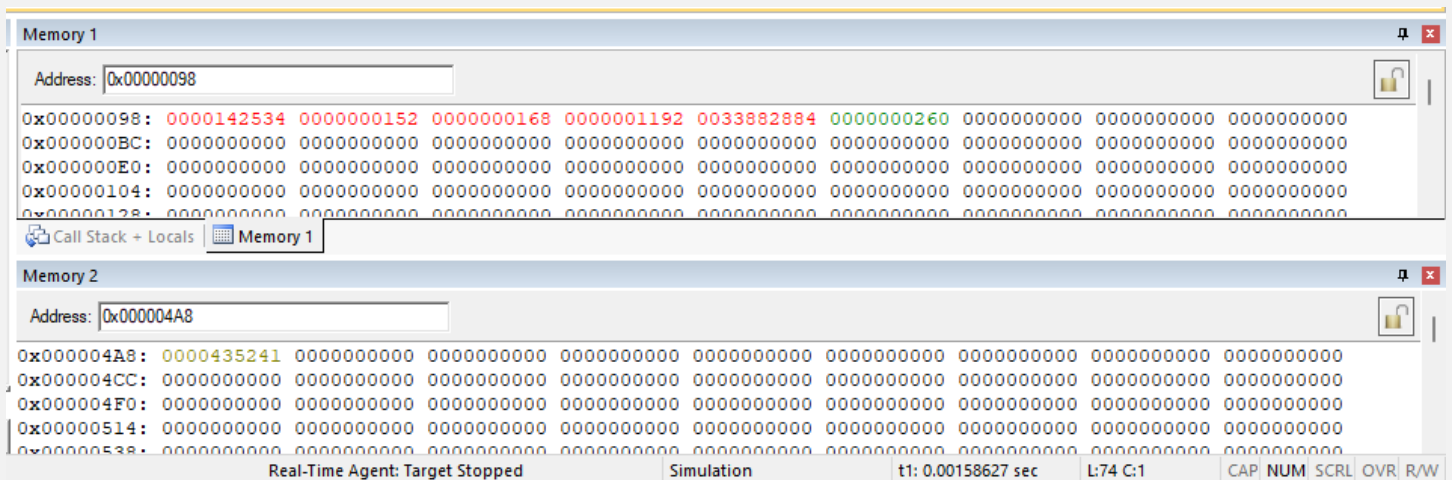
Real-Time Agent: Target Reset Simulation t1: 0.00000000 sec L:10 C:1 CAP NUM SCRL OVR R/W

00:00 AM 23-03-2022

Final Output:



## Final Memory Values:



**Observation:** We can see that for input 142534, we have achieved its reverse 435241 stored in result memory location.

5.5] Check whether the given number is a Fibonacci number.

-> For input 144, which is a Fibonacci Number

Source Code:

```

    AREA AllocSpace, DATA, NOINIT, READWRITE
    ;Space for storing result
result SPACE 1;
;Actual Code
    AREA hmm, CODE, READWRITE
    EXPORT Reset_Handler
Reset_Handler
    ;address to the input number
    LDR R1, =inputnumber ;
    ;address to the result
    LDR R2, =result ;
    MOV R4, #1; (n-1)th term
    MOV R5, #1; nth term
    ; load the input number from memory
    LDR R3, [R1];
    ; base case check if input ==0;
    CMP R3, #0;
    ; as 0 is a fib no for f(0), store & jump
    MOVEQ R6, #1;
    BEQ dun
    ;for other cases n>0;
    ;as f(1)==f(2)==1;
    ; it doesn't matter in iteration.
    ; while comparing.
    ;comparing nth fib, initially R5, f(1)=f(2)
chk CMP R3, R5;
    ; storing 1 in result register if it matches any fibb.
    MOVEQ R6, #1;
    ; jumping to store result in memory
    BEQ dun
    ; if input > current F(n),
    ;We will calculate F(n+1) and store it in R5
    ; And Current F(n) in R4;
    ADDPL R7, R4, R5;
    MOVPL R4, R5;
    MOVPL R5, R7;
    ; LOOP to check again.
    BPL chk;
    ; if input < F(n) then its not a fibonacci number
    ; store 0xFF in result register;
    MOV R6, #0xFF; :0
    ; store result in memory.
dun STRB R6, [R2];

```

```
stop BAL stop
; input_number
inputnumber DCD 144;
END
```

## Debugging:

Initial Memory: (after getting the address through register)

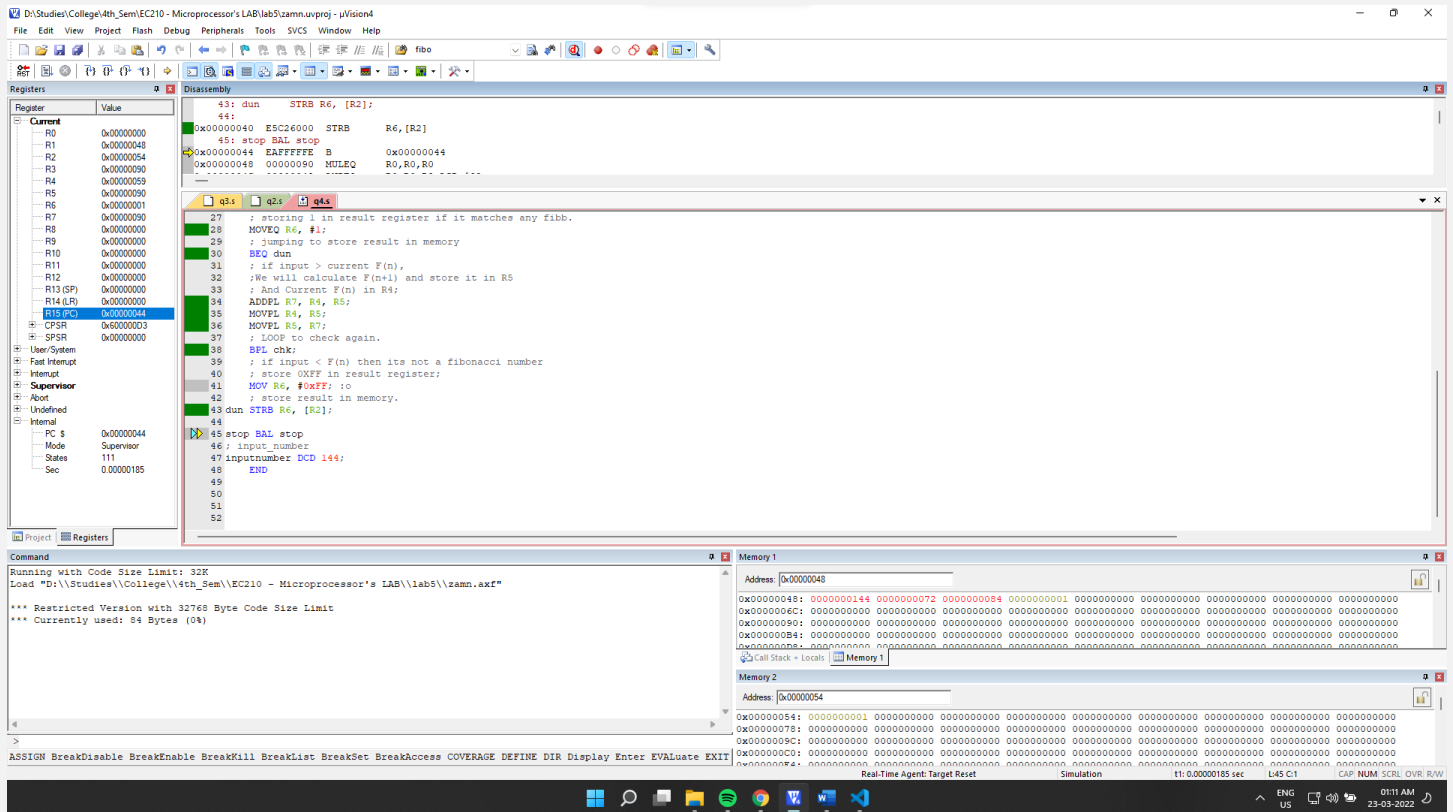
Memory 1 shows input, memory 2 will hold output.

The screenshot shows the uVision IDE with two memory windows open. Memory 1 is at address 0x00000048 and contains the value 0000000144. Memory 2 is at address 0x00000054 and is empty. The status bar at the bottom indicates 'Real-Time Agent: Target Reset' and 'Simulation'.

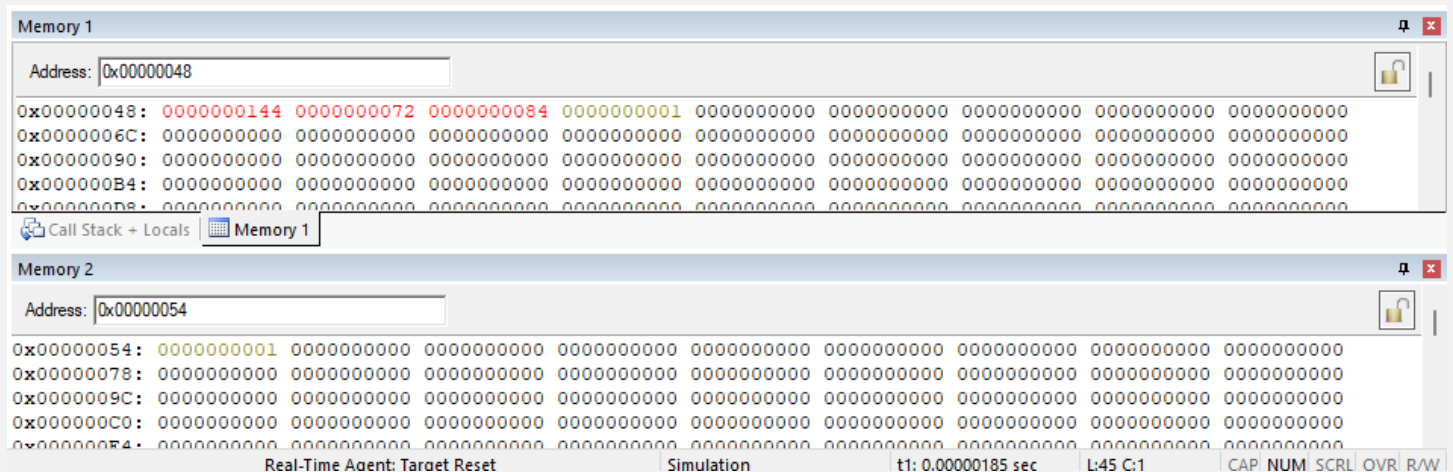
## Setup:

The screenshot shows the uVision IDE with the Registers and Disassembly windows open. The Registers window shows the current values of the registers. The Disassembly window shows the assembly code for the program. The Command window shows the command to run the program.

# Final Output:



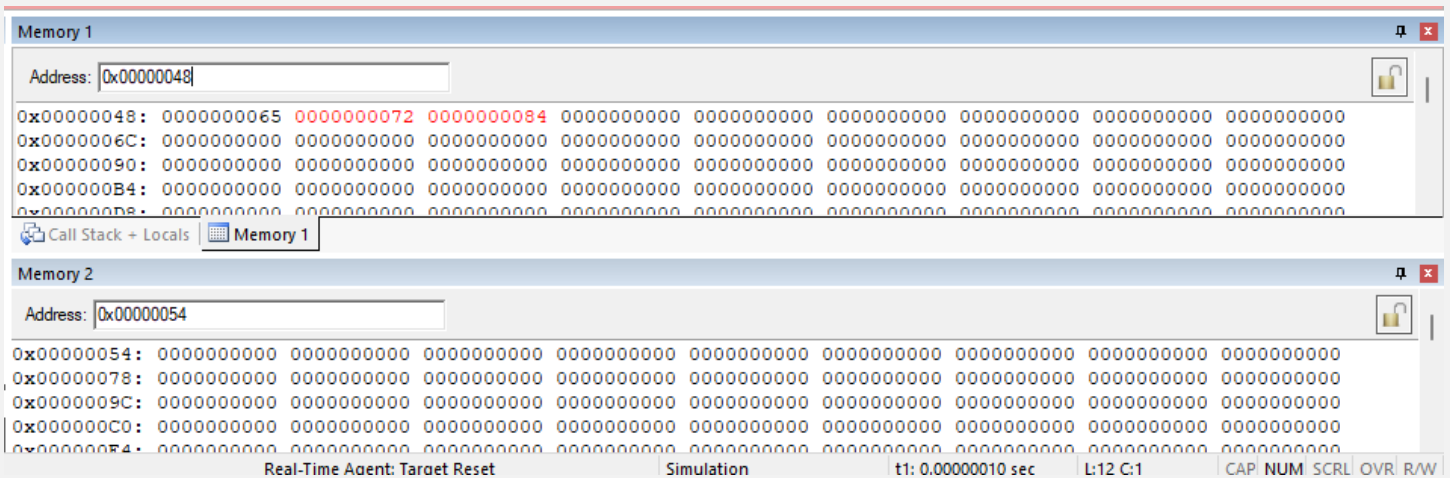
# Final Memory:



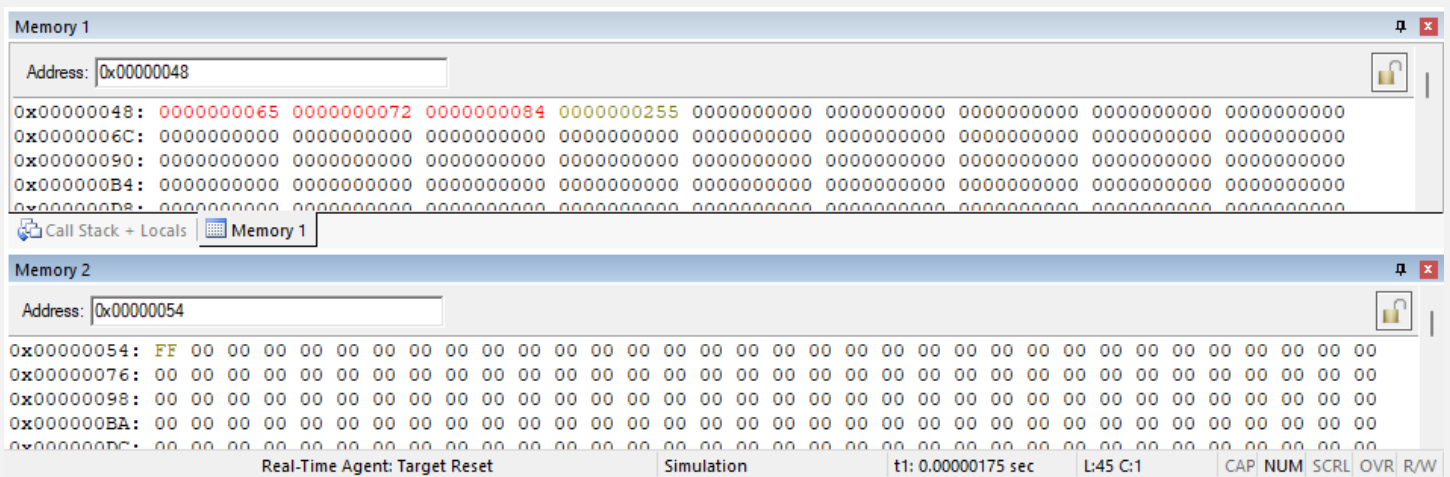
**Observation:** We can see that in memory2, 1 is stored. Output generated for the input 144 is correct as it is a Fibonacci number.

For Input 65:

Initial Memory:



## Final Memory:



Observation: we can see that the output is 0xFF is correct as the input 65 is not a Fibonacci number.

5.6] To generate the Fibonacci numbers up to the given number

->

Taking input =11

Source Code:

```

AREA AllocSpace, DATA, NOINIT, READWRITE
;Space for storing result
result SPACE 1024;
;Actual Code
AREA hmm, CODE, READWRITE
EXPORT Reset_Handler
Reset_Handler

```



```

;address to the input number
LDR R1, =inputnumber ;
;address to the result
LDR R2, =result ;
MOV R4, #1; (n-1)th term
MOV R5, #1; nth term
; load the input number from memory
LDR R3, [R1];
; base_case if the input number is 0;
CMP R3, #0;
MOVEQ R6, #0;
BEQ dun
;base_case if the input number is 1;
CMP R3, #1;
MOVEQ R6, #1;
BEQ dun
; for input number >1
MOV R9, #2; iterator i from 2 to n.
chk CMP R3, R9; checking if i==n
BEQ dun ; jump to store if done.
ADD R9, #1; increment iterator
; nth term = (n-1)th + (n-2)th
ADD R7, R4, R5;
; storing nth term in R5
MOV R4, R5;
;storing (n-1)th term in R4
MOV R5, R7;
; jump to chk to loop till i==n
BPL chk;
;store the result in [R2]
dun STR R5, [R2];

stop BAL stop
; input_number
inputnumber DCD 11;
END

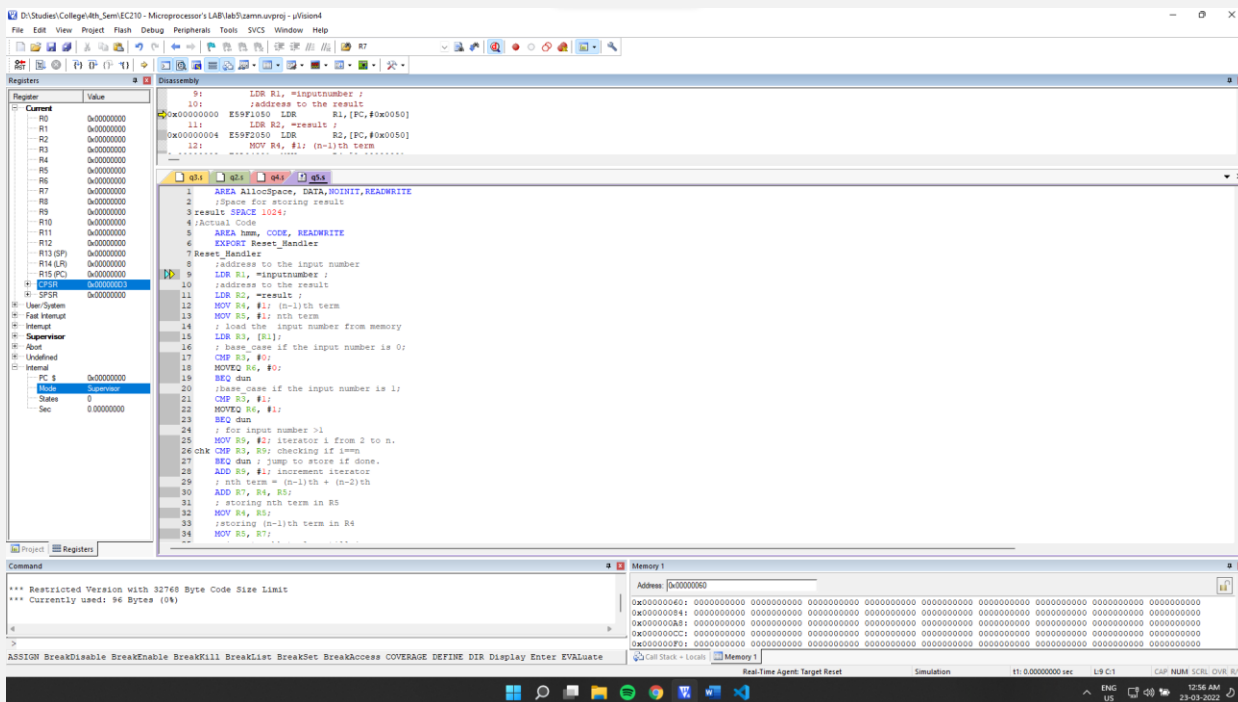
```

**Initial Memory:** (after getting the address through register and loading the value into the reserved space)

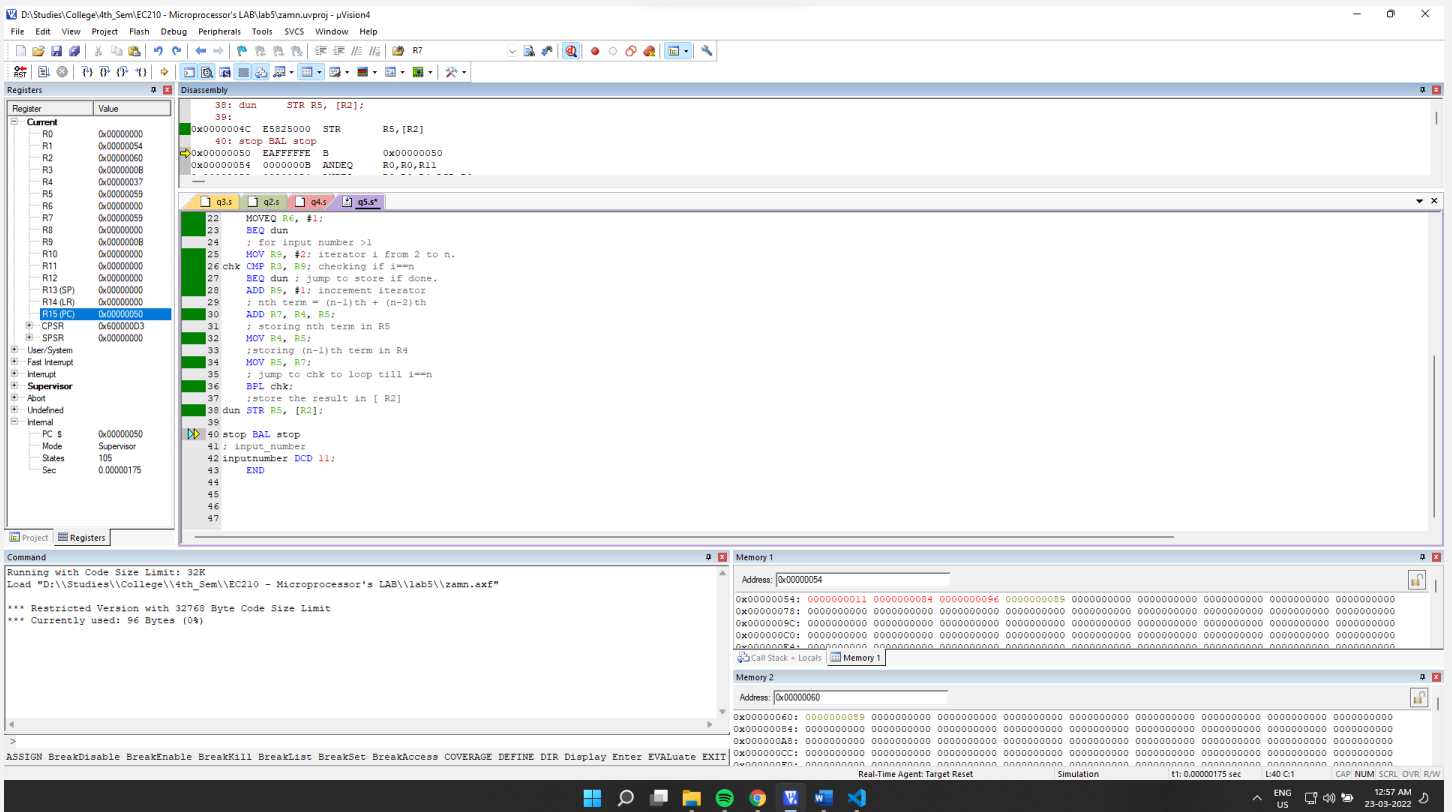
Memory 1 will hold the input no and memory 2 will hold result (Fibonacci no).

Memory 1									
Address: 0x00000054									
0x00000054:	0000000011	0000000084	0000000096	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000
0x00000078:	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000
0x0000009C:	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000
0x000000C0:	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000
0x000000F4:	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000
Call Stack + Locals Memory 1									
Memory 2									
Address: 0x00000060									
0x00000060:	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000
0x00000084:	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000
0x000000A8:	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000
0x000000CC:	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000
0x000000F0:	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000
Real-Time Agent: Target Reset Simulation t1: 0.00000010 sec L:12 C:1 CAP NUM SCRL OVR R/W									

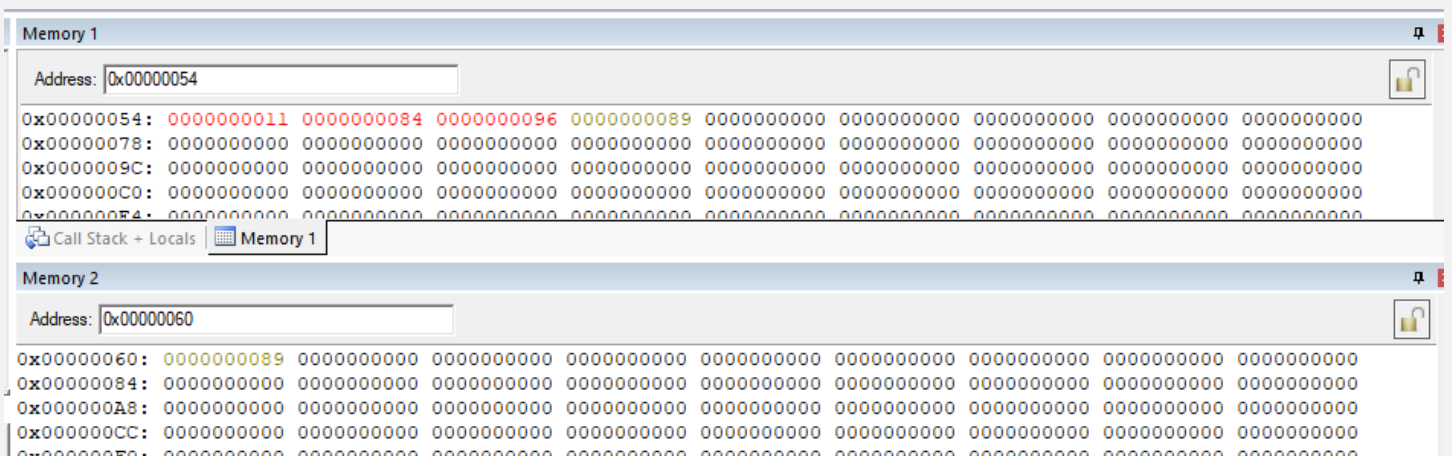
## Setup:



## Final Output:



## Final Memory:



**Observation:** We can see that the output stored in result memory location 89 is correct as 11<sup>th</sup> Fibonacci no is 89.