EC-210

MICROPROCESSORS LAB

LAB-5

UTKARSH MAHAJAN 201EC164

ARNAV RAJ 201EC109

<u>Objective:</u> To study defining memory area, constant in the assembly program.

Exercise:

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

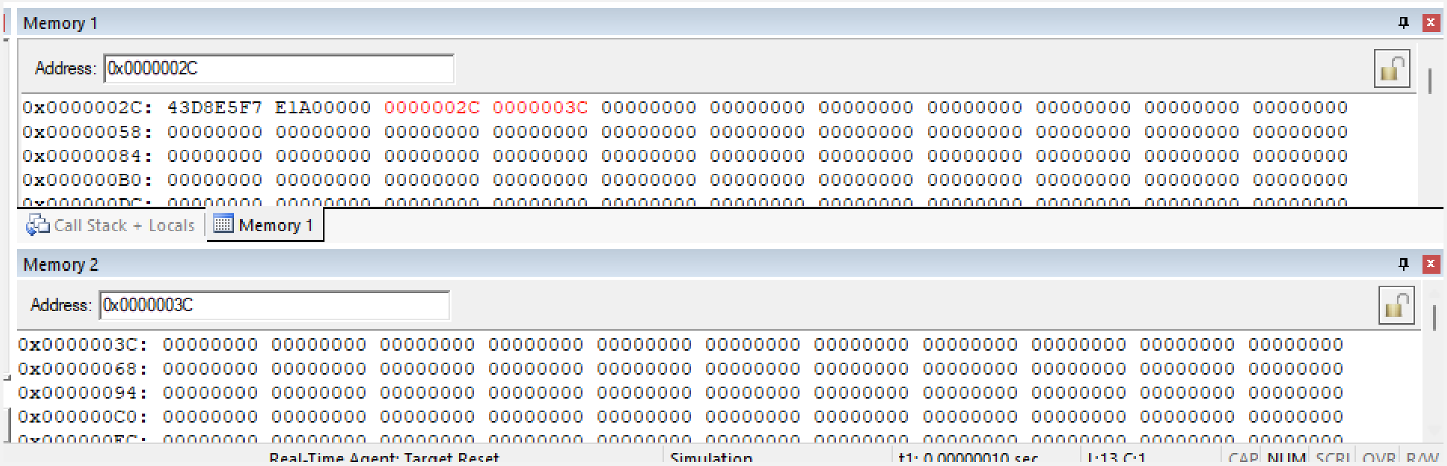
->Source Code:

```
    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
```

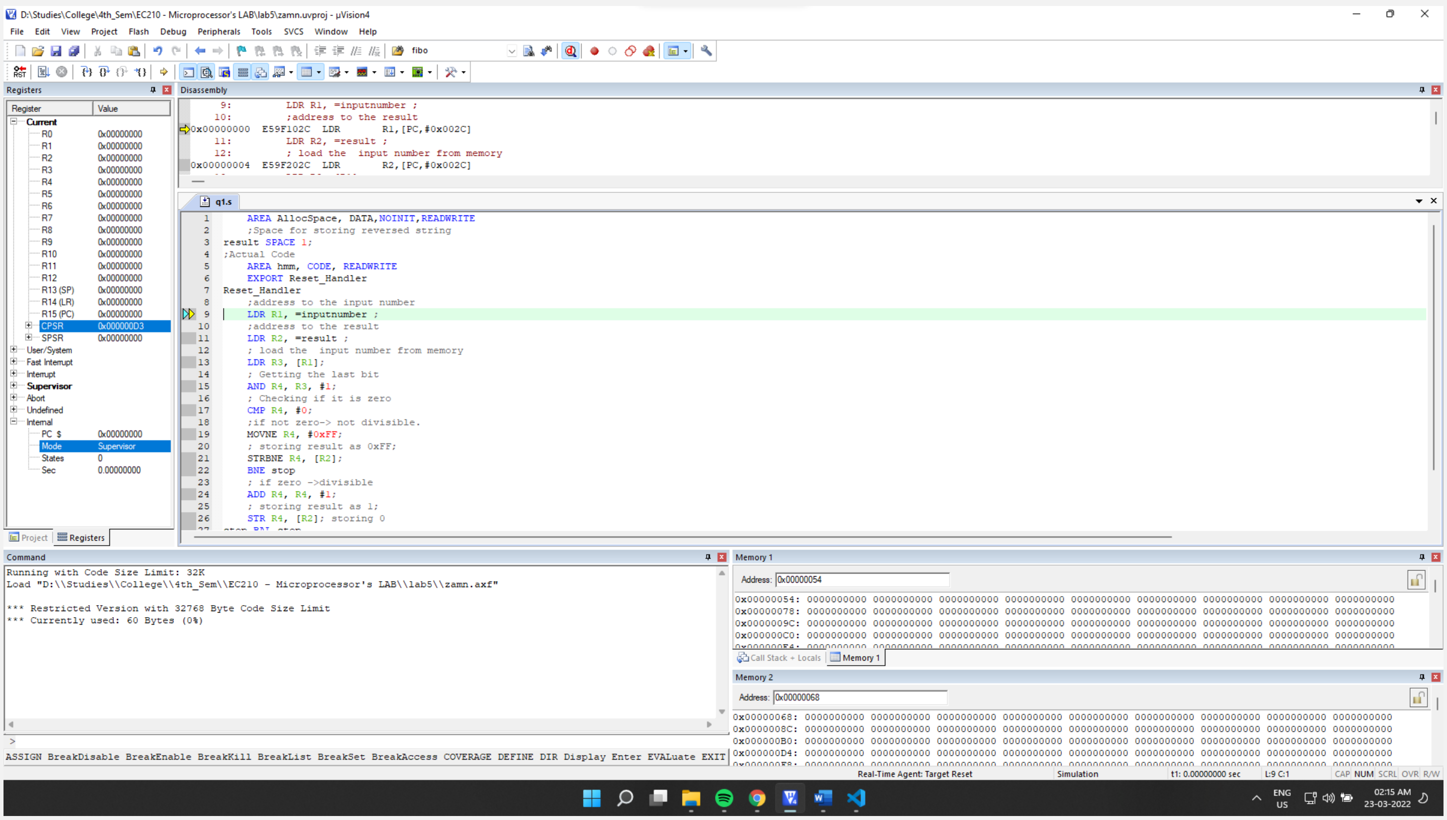
<u>Debugging:</u>

Initial Memory: (after getting the address through register)

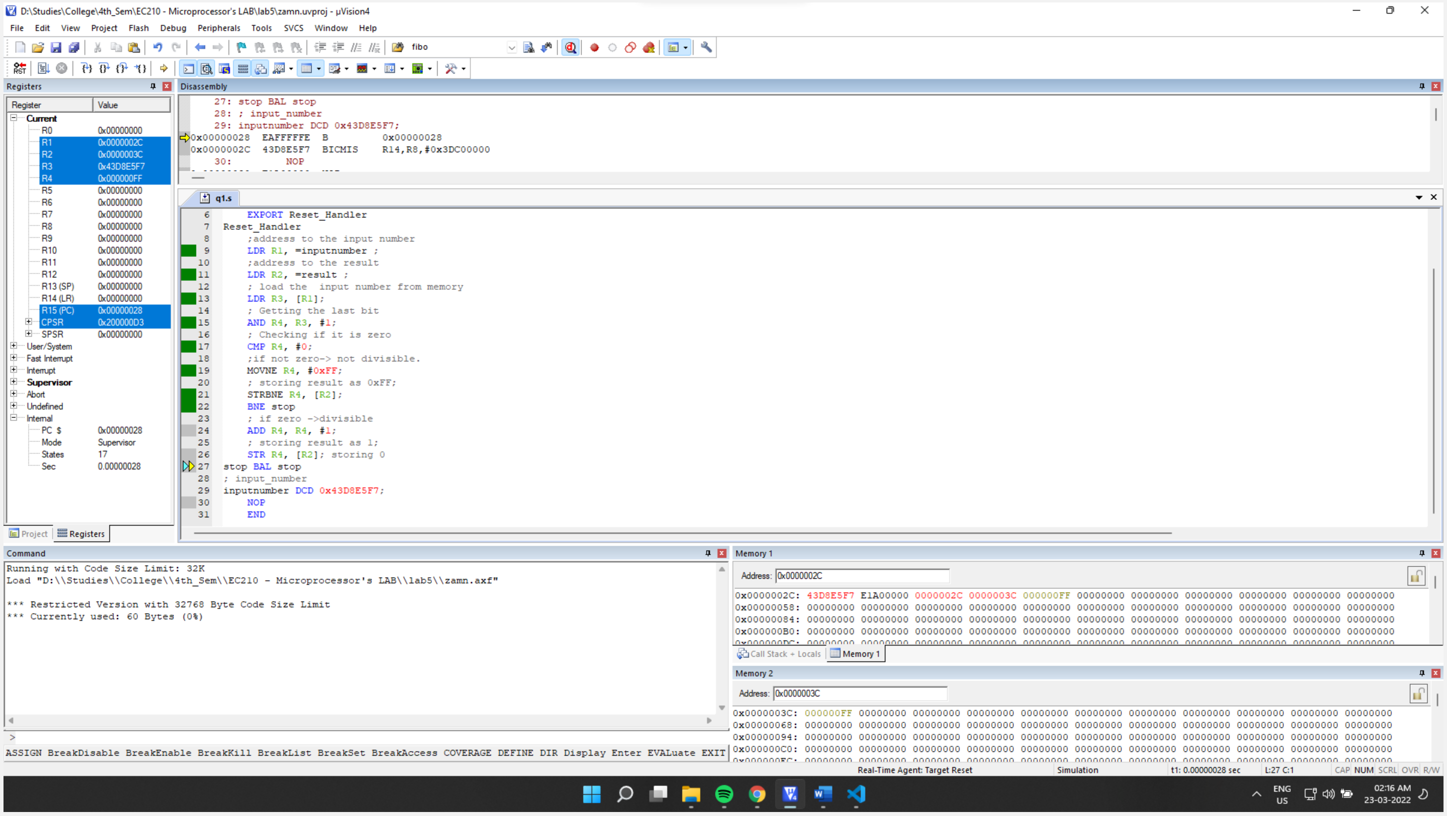
Memory 1 shows input and memory 2 will store output



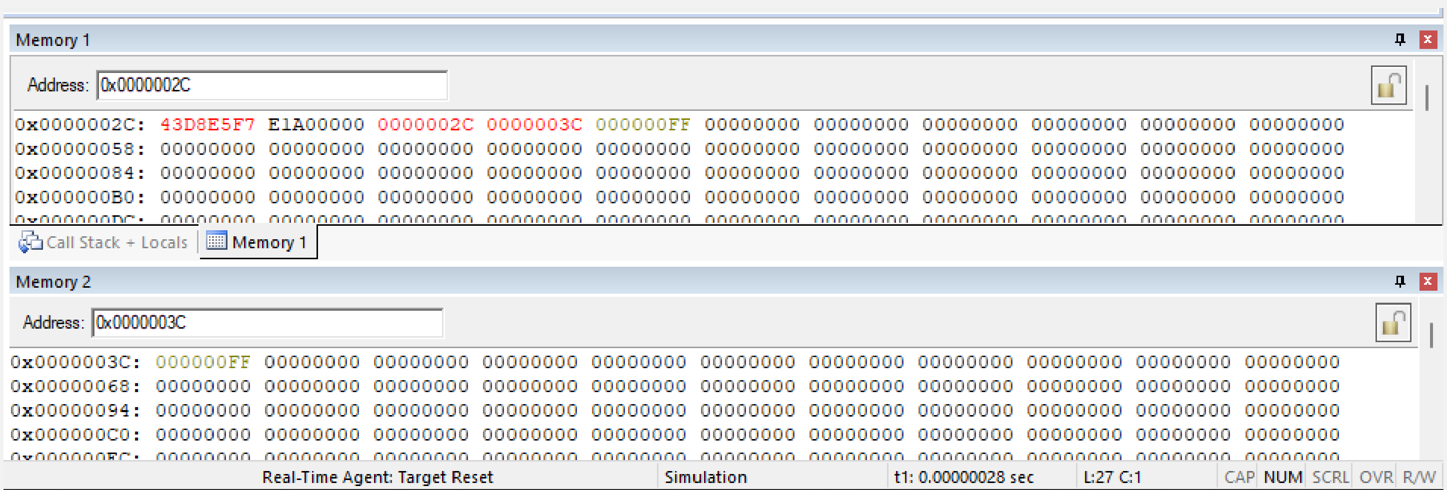
Setup:



Final Output:



Final Memory:

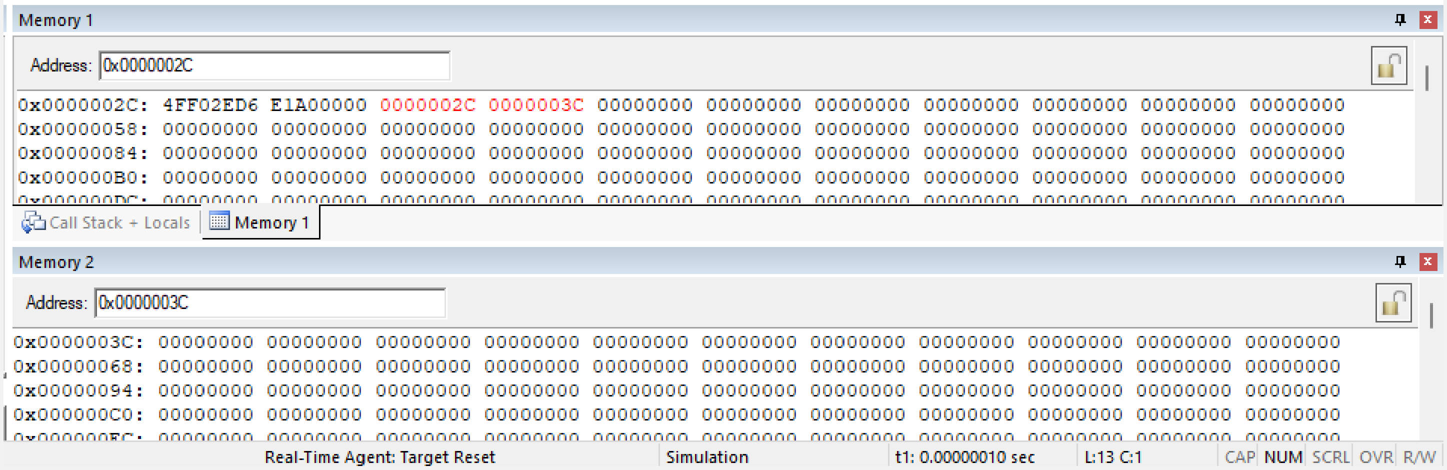


Observation:

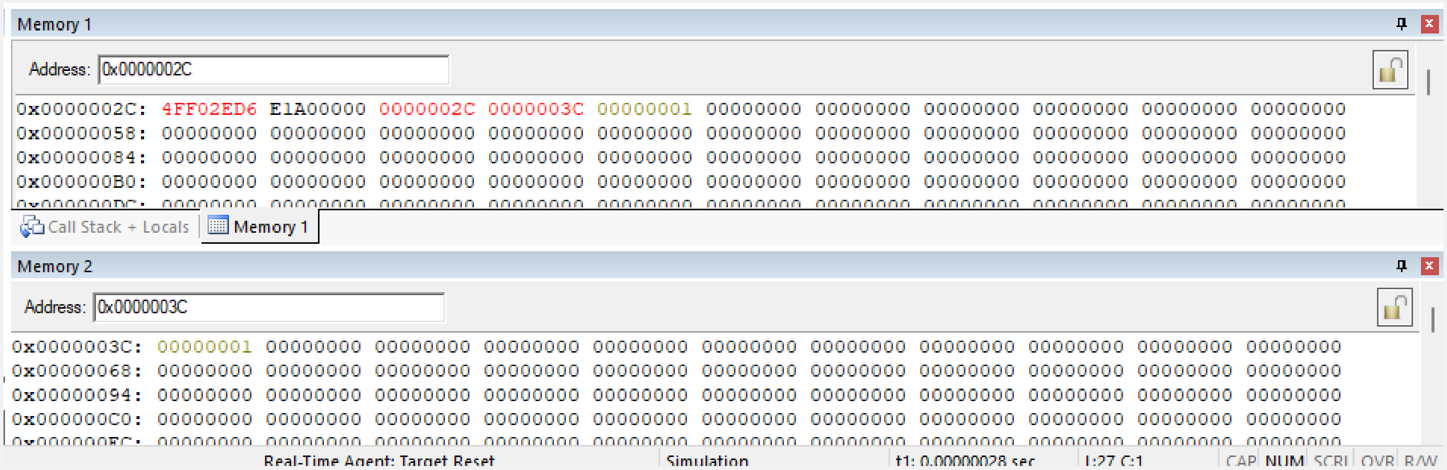
We can see that the output 0xFF for the given input is correct. `0x43D8E5F7` Is not divisible by 2.

For input number: 0x4FF02ED6

Initial Memory:



Final memory:



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
```

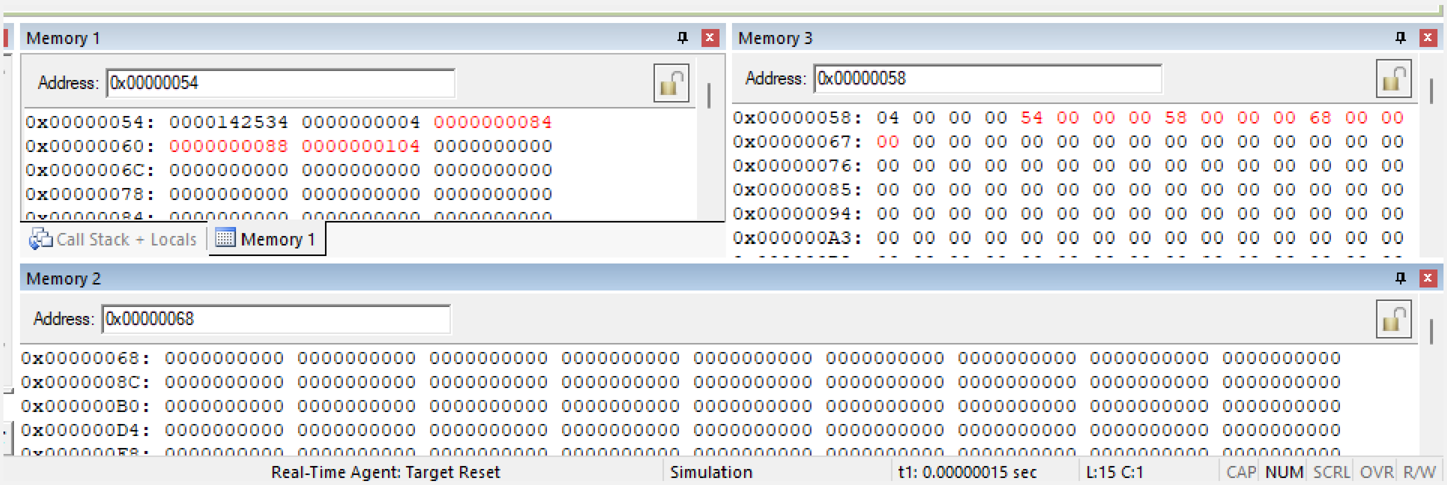
```
    ;address to the input number
    LDR R1, =inputnumber ;
    ;address to the input digit to cmp
    LDR R2, =inputdigit;
    ;address to the result
    LDR R3, =result ;
    ; load the  input number from memory
    LDR R4, [R1];
    LDRB R9, [R2];
    ; Getting the last bit
    MOV R5, R4; R5 = 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
```

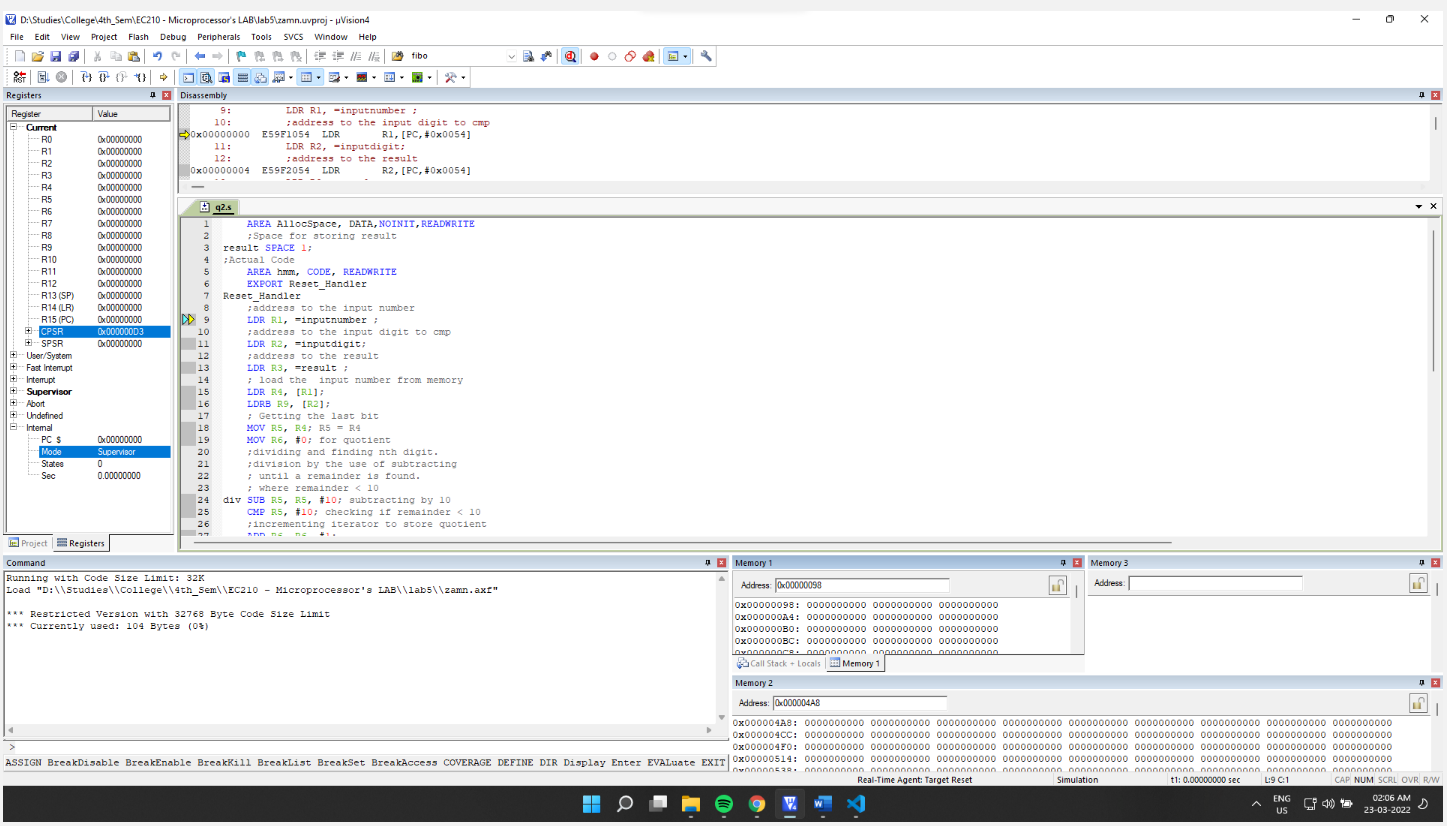
Debugging:

Initial Memory: (after getting the address through register)

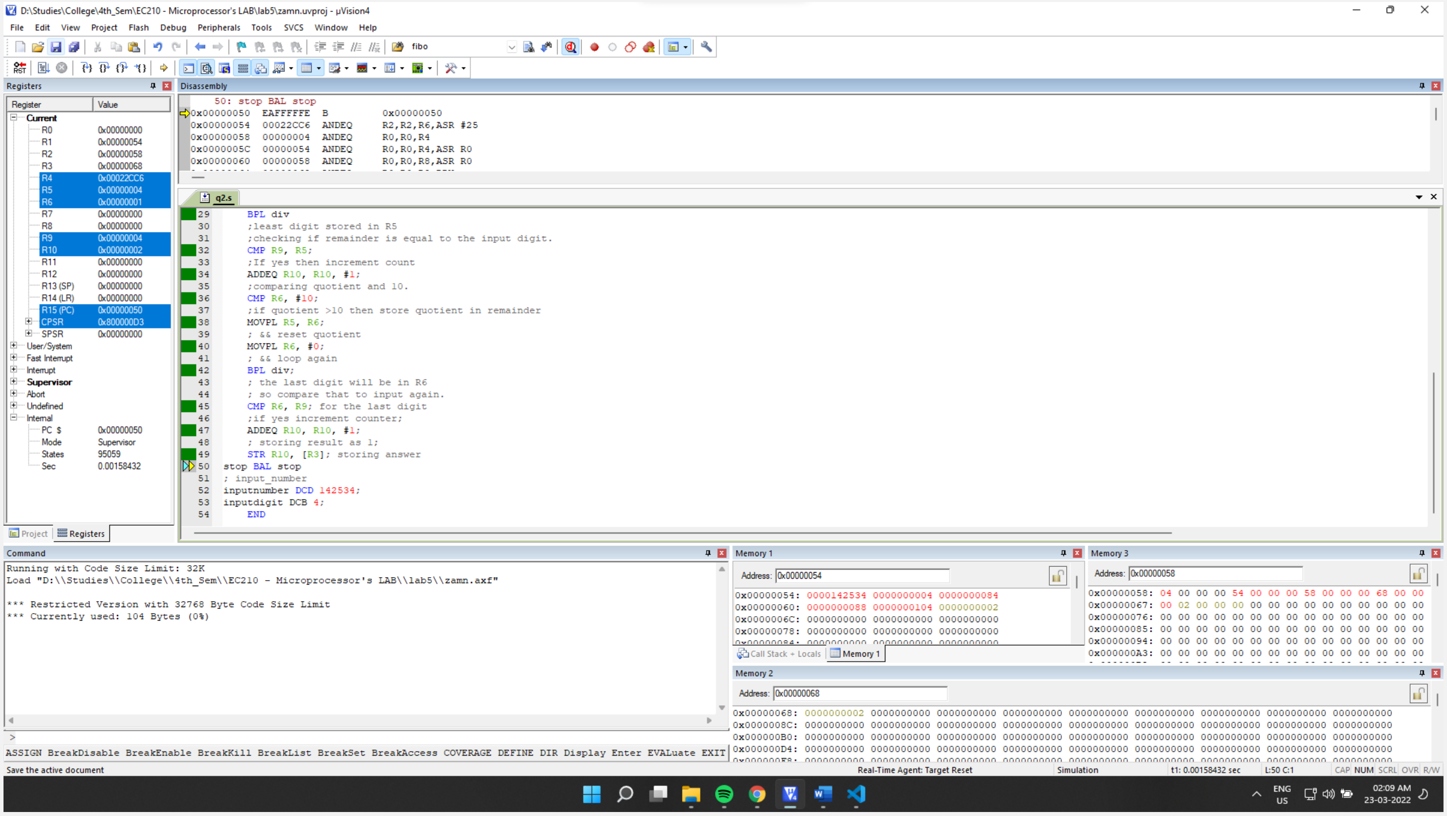
Memory 1 shows input number and memory 2 shows input digit while memory 3 will hold result (no of occurrences).



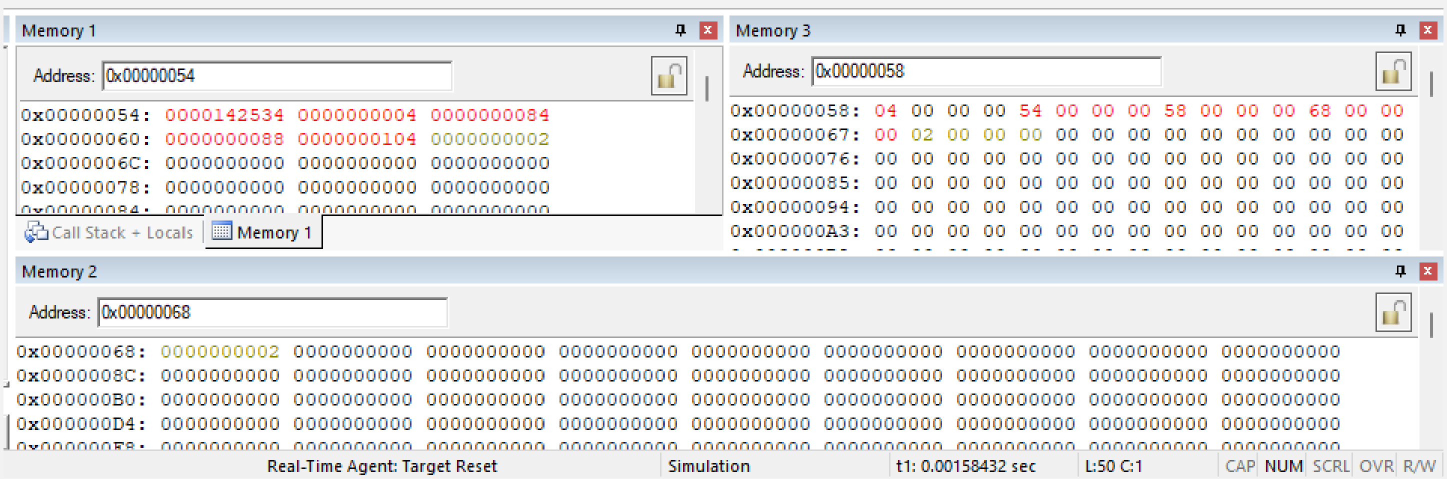
Setup:



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:

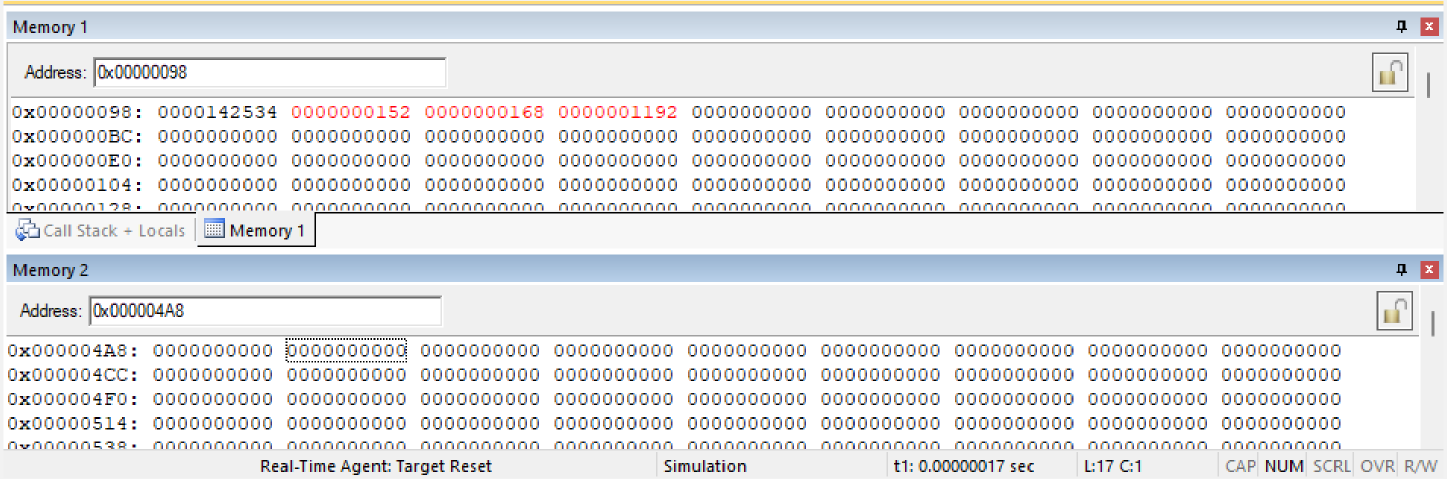
```
    AREA AllocSpace, DATA,NOINIT,READWRITE
    ;Space for storing reversed string
array SPACE 1024;
result SPACE 1024;
;Actual Code
    AREA hmm, CODE, READWRITE
    EXPORT Reset_Handler
Reset_Handler
    ;address to the input number
    LDR R1, =inputnumber ;
    ;address for space used in operation
    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
```

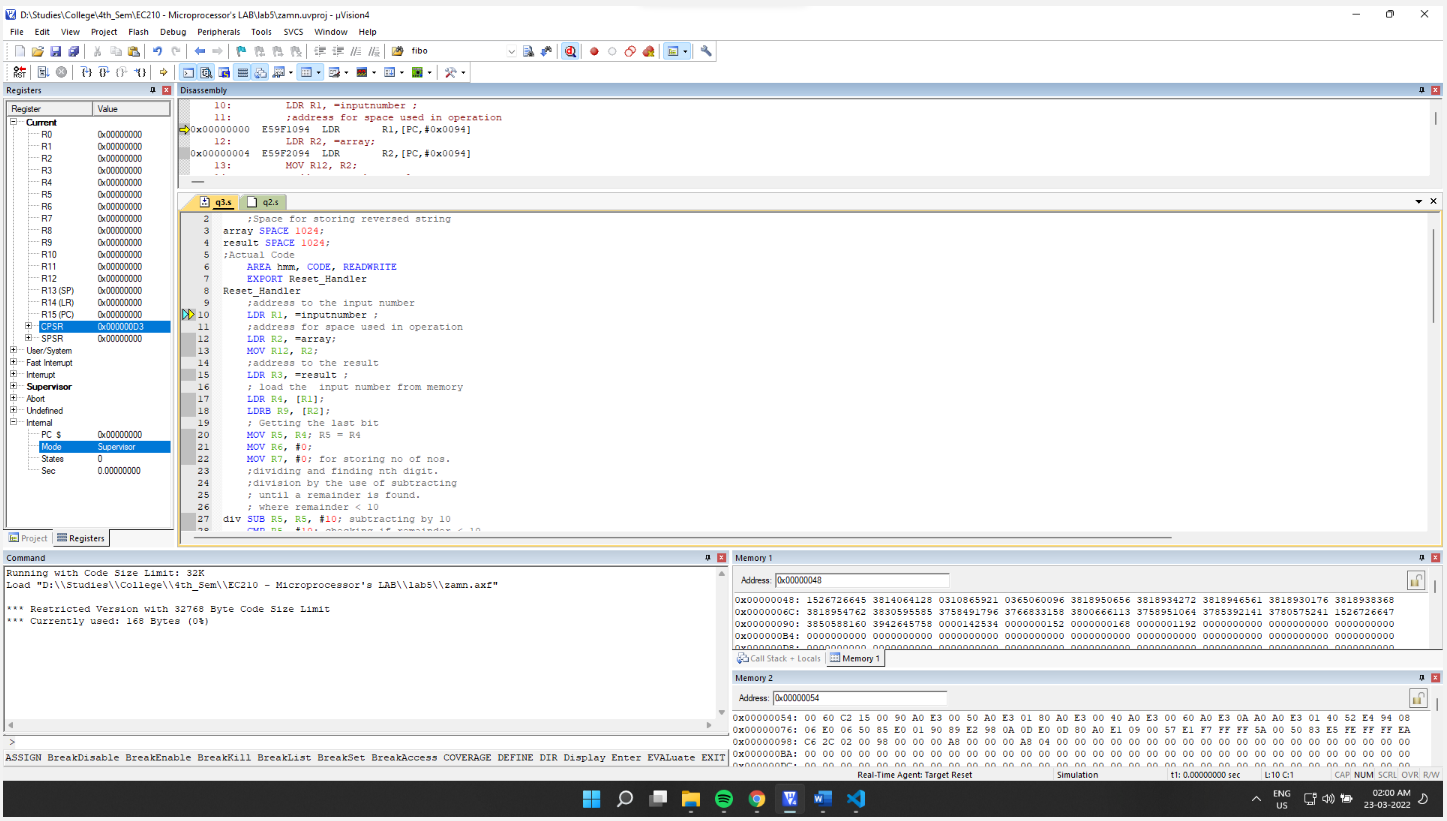
Debugging:

Initial Memory: (after getting the address through register)

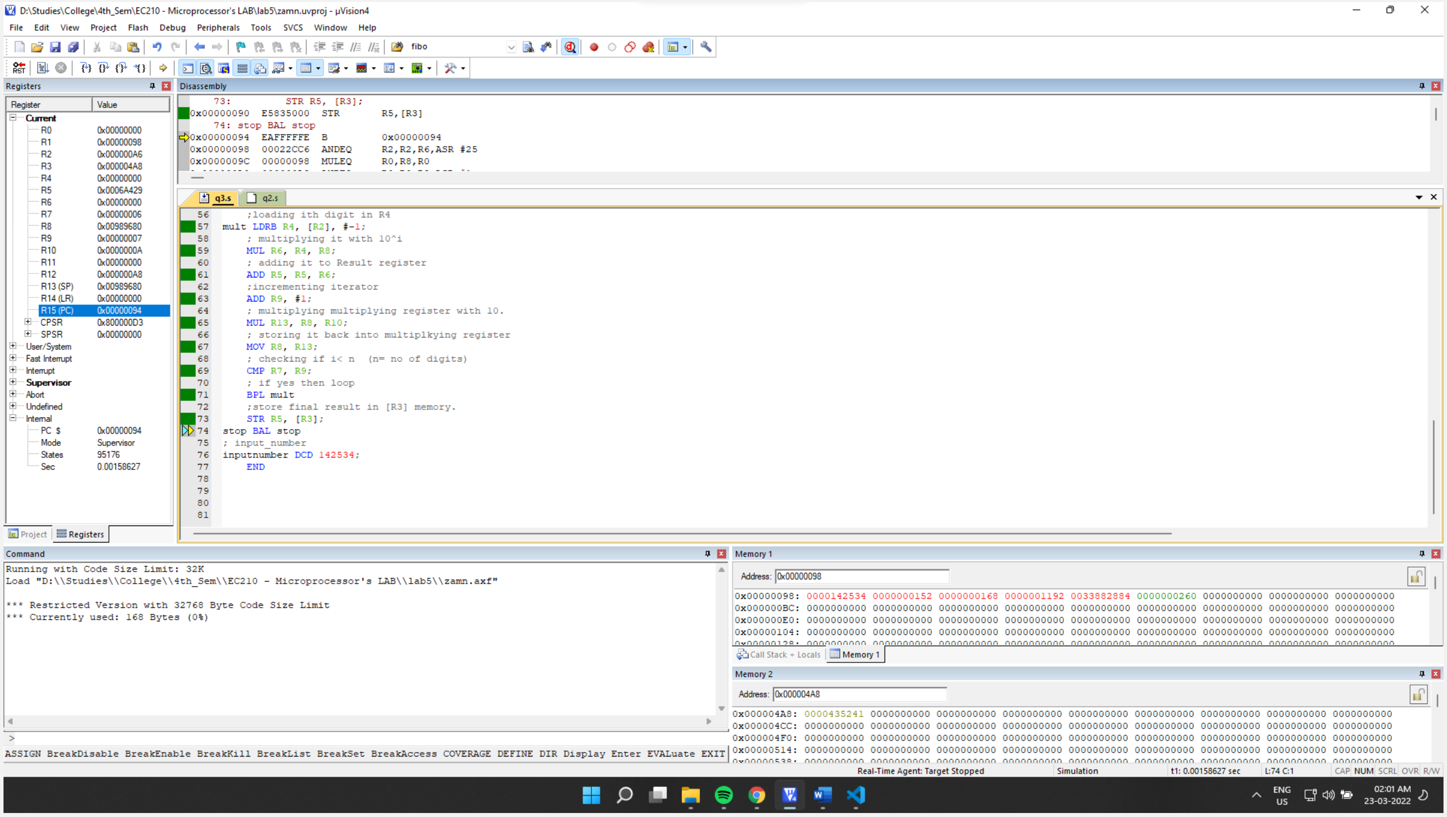
Memory 1 shows input number and memory 2 will hold reversed no.



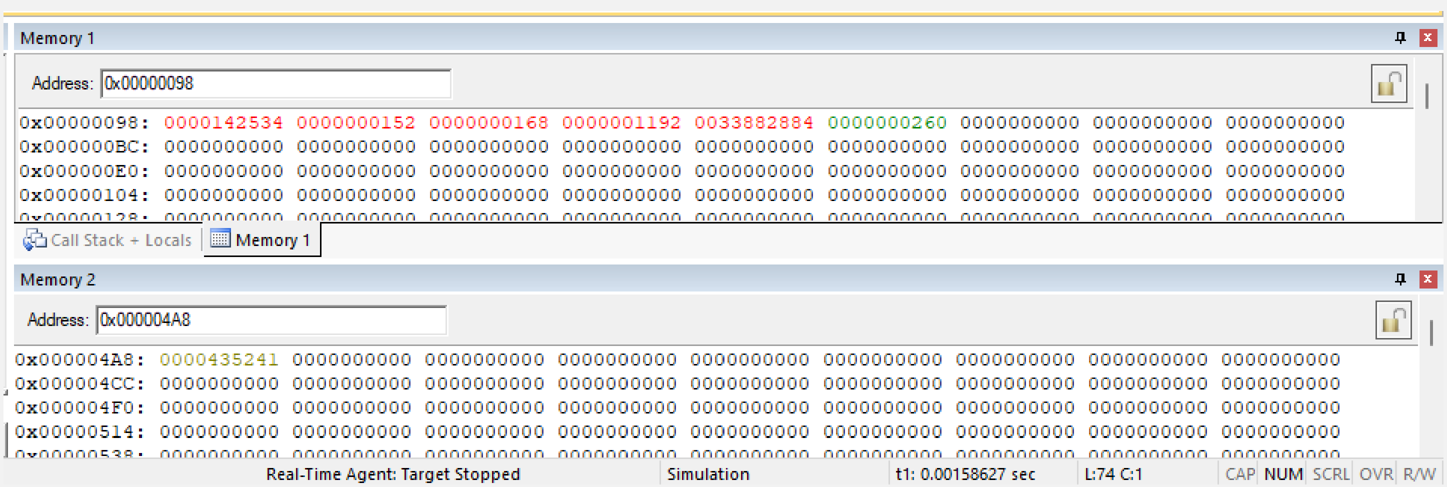
Setup:



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:

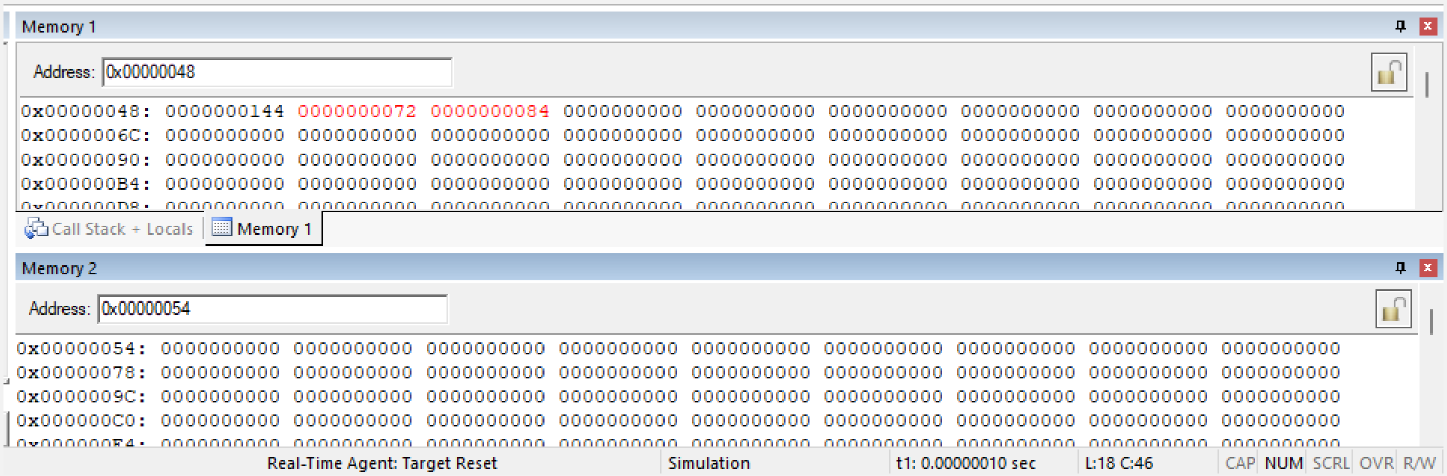
```
    AREA AllocSpace, DATA,NOINIT,READWRITE
    ;Space for storing result
result SPACE 1;
;Actual Code
    AREA hmm, CODE, READWRITE
    EXPORT Reset_Handler
Reset_Handler
    ;address to the input number
    LDR R1, =inputnumber ;
    ;address to the result
    LDR R2, =result ;
    MOV R4, #1; (n-1)th 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; :o
    ; store result in memory.
dun STRB R6, [R2];
```

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

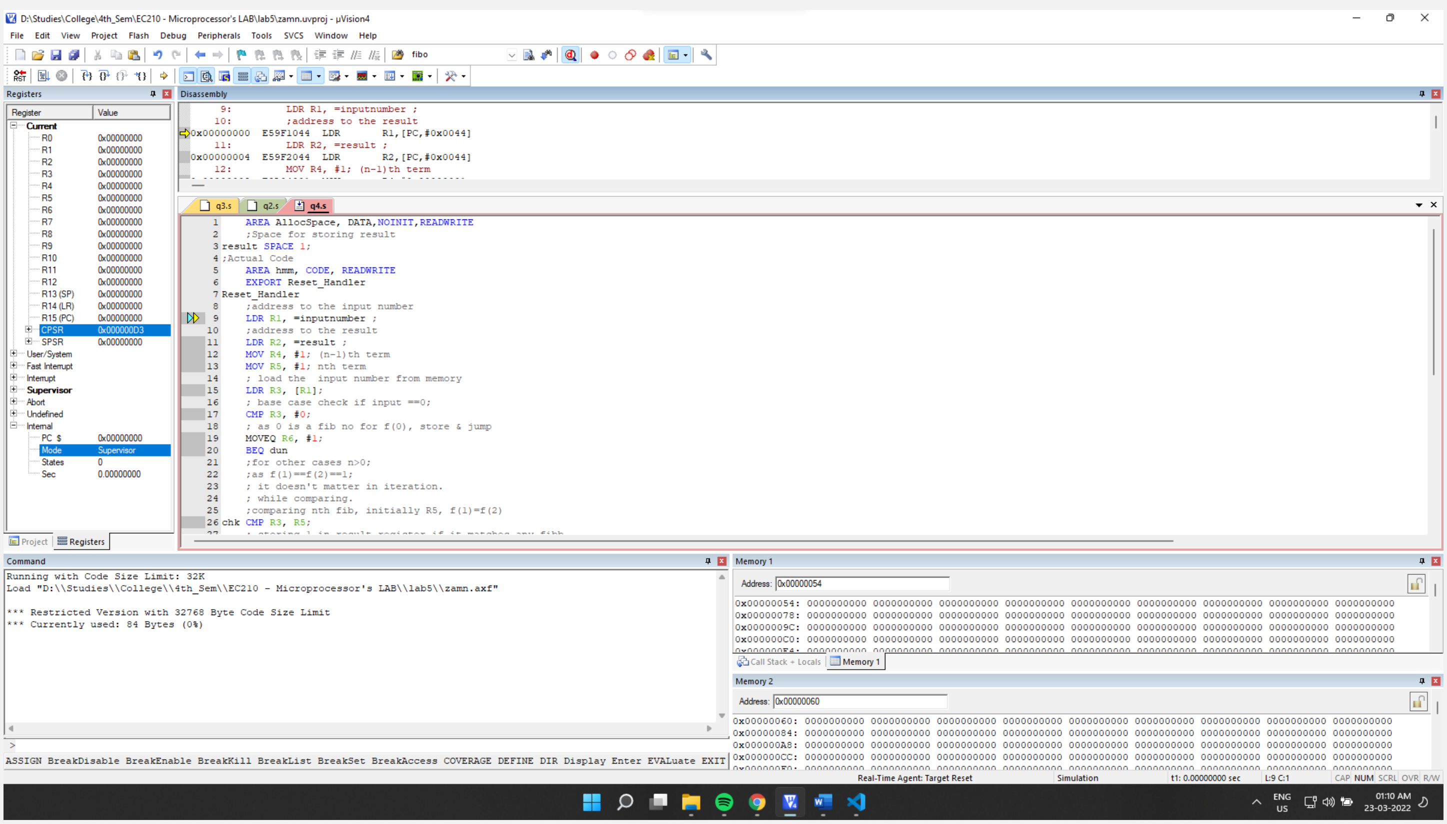
Debugging:

Initial Memory: (after getting the address through register)

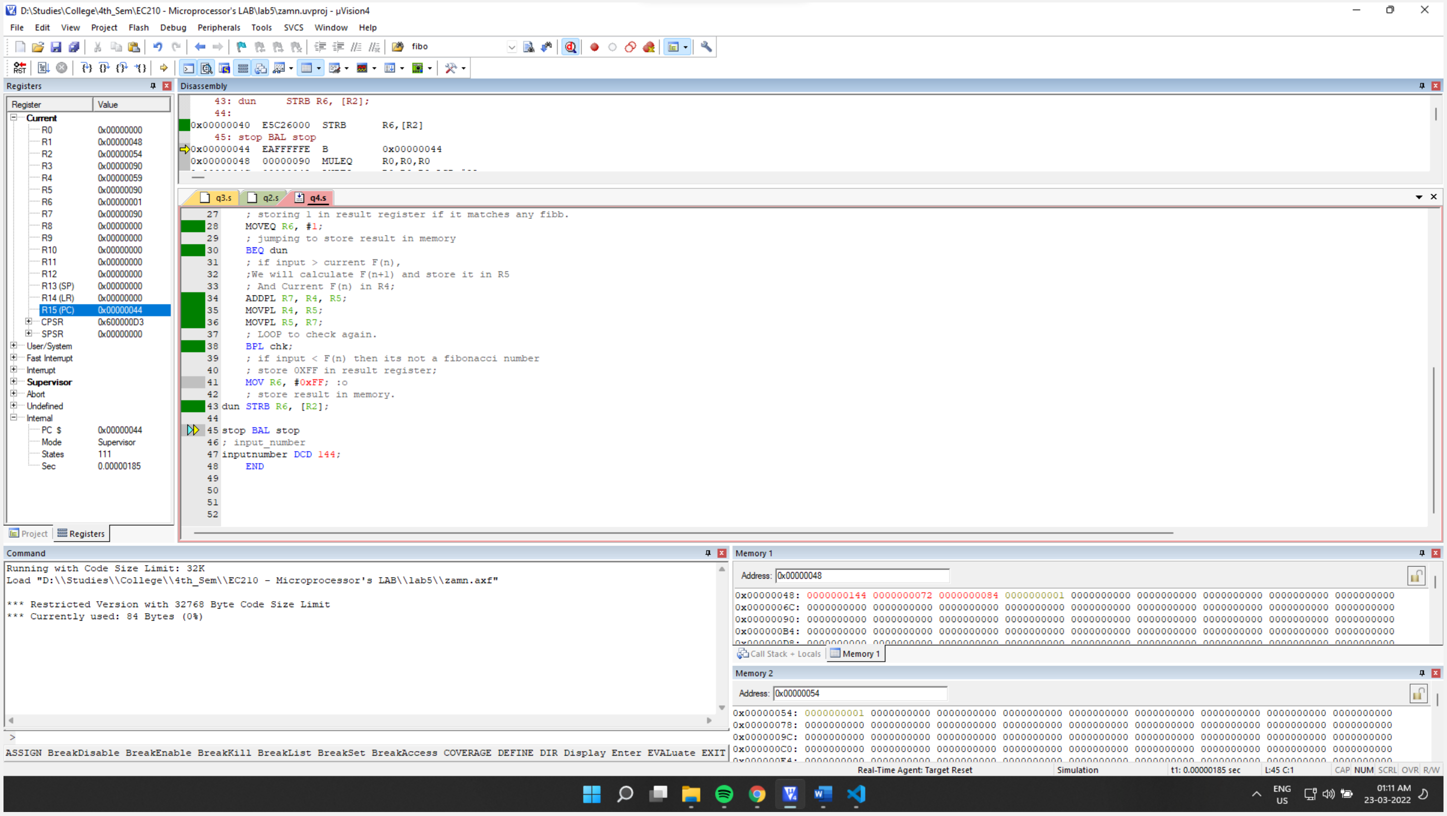
Memory 1 shows input, memory 2 will hold output.



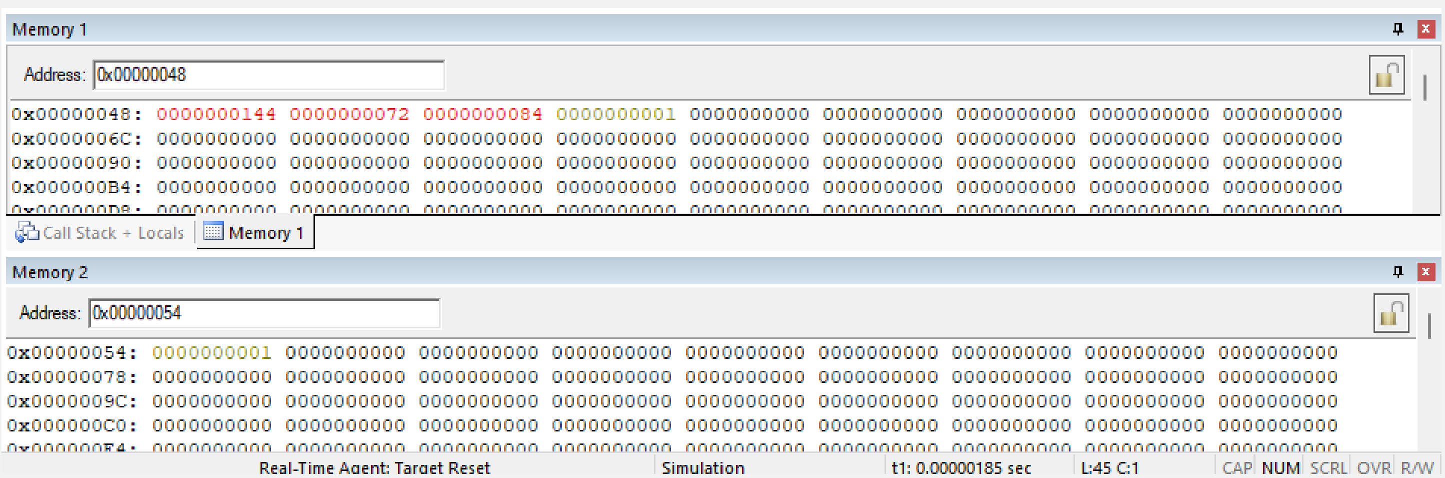
Setup:



Final Output:



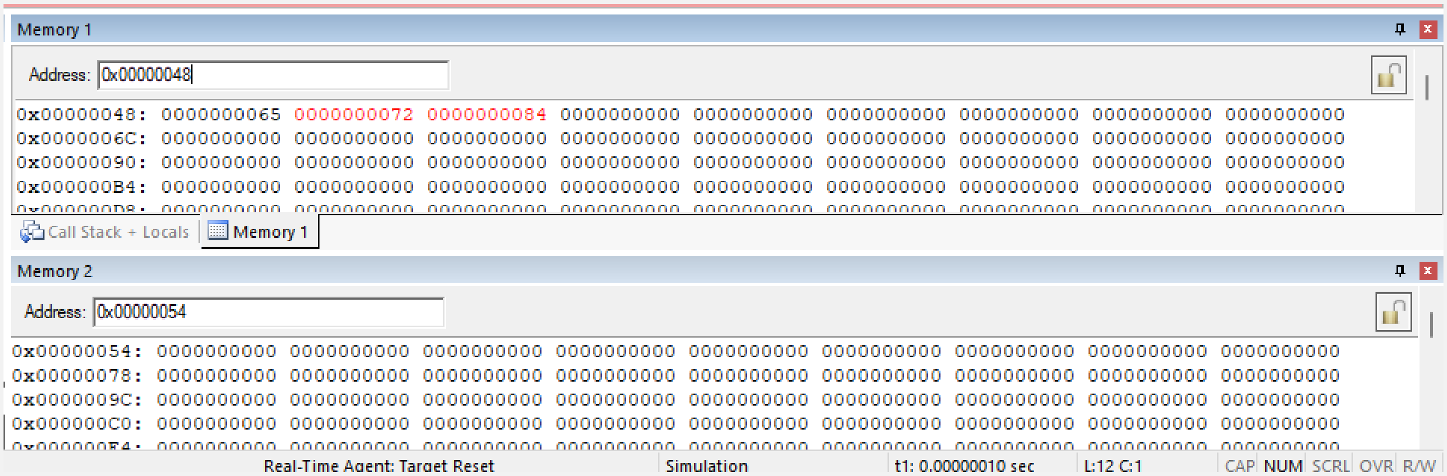
Final Memory:



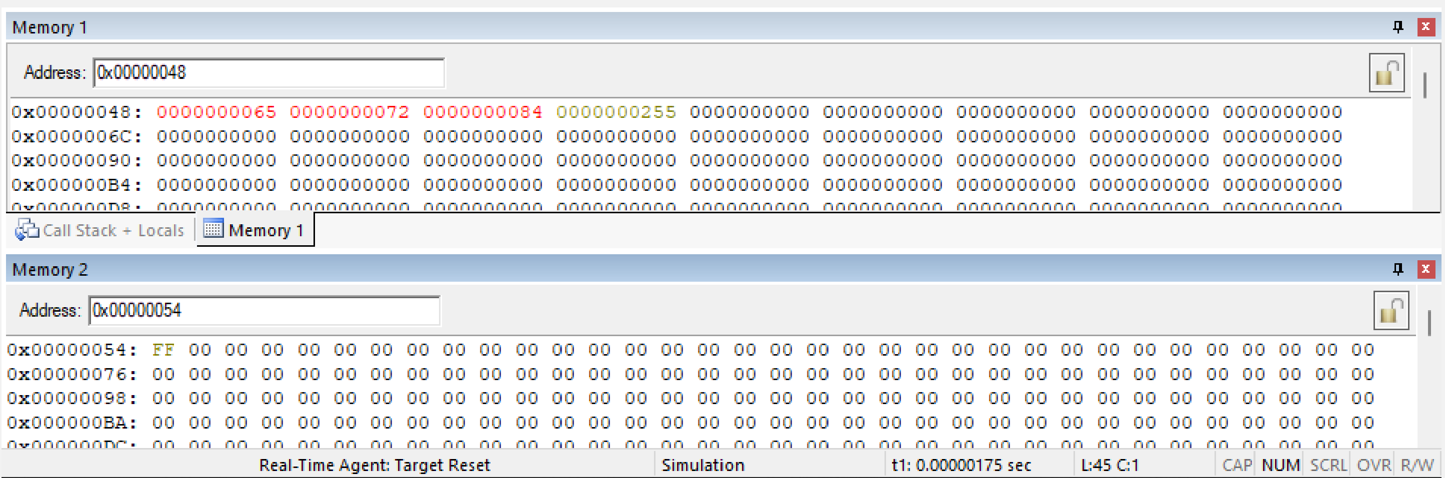
Observation: We can see that in memory2, 1 is stored. Output generated for the input 144 is correct as it is a Fibonacci number.

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

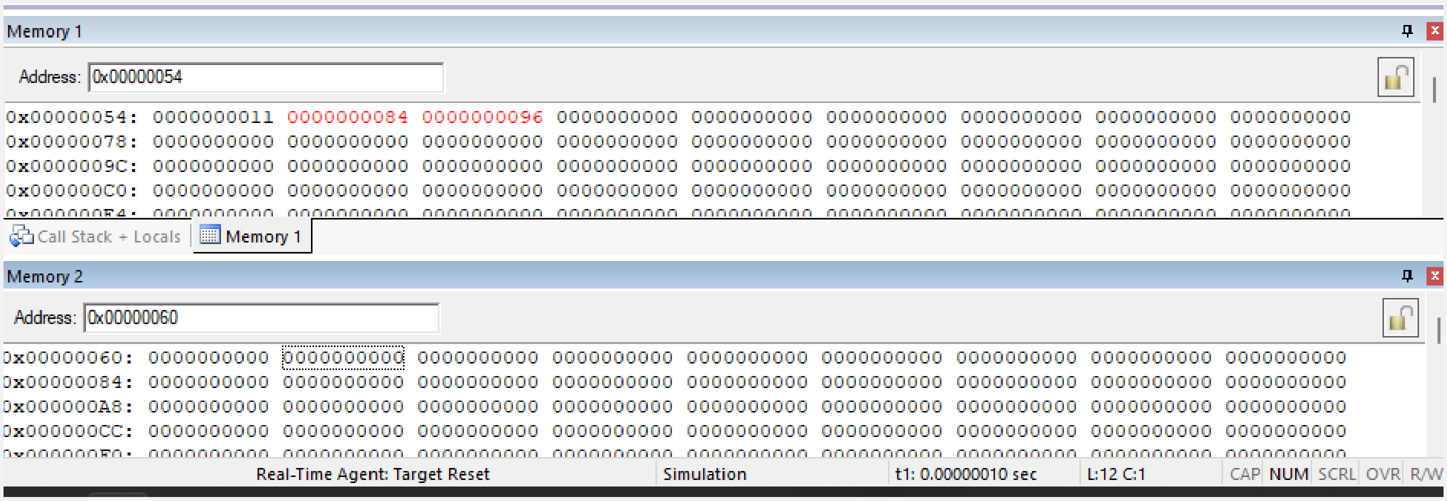
Source Code:

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

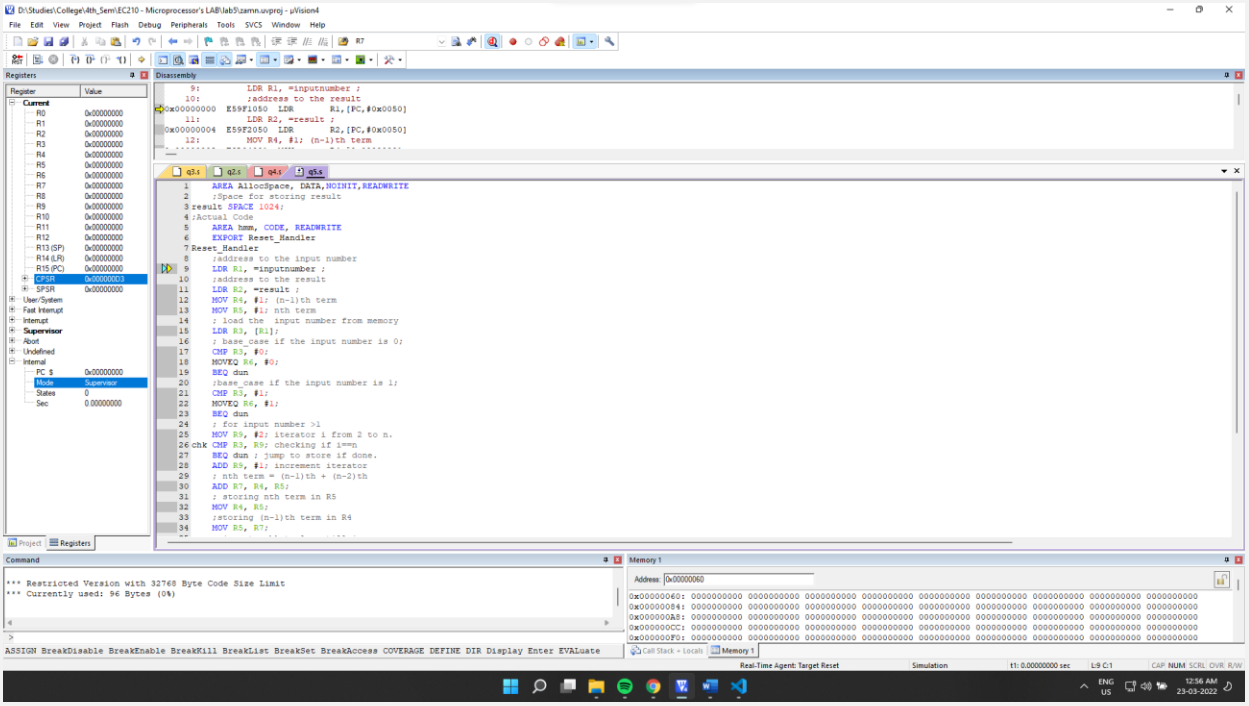
```
    ;address to the input number
    LDR R1, =inputnumber ;
    ;address to the result
    LDR R2, =result ;
    MOV R4, #1; (n-1)th term
    MOV R5, #1; nth term
    ; load the  input number from memory
    LDR R3, [R1];
    ; base_case if the input number is 0;
    CMP R3, #0;
    MOVEQ R6, #0;
    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)

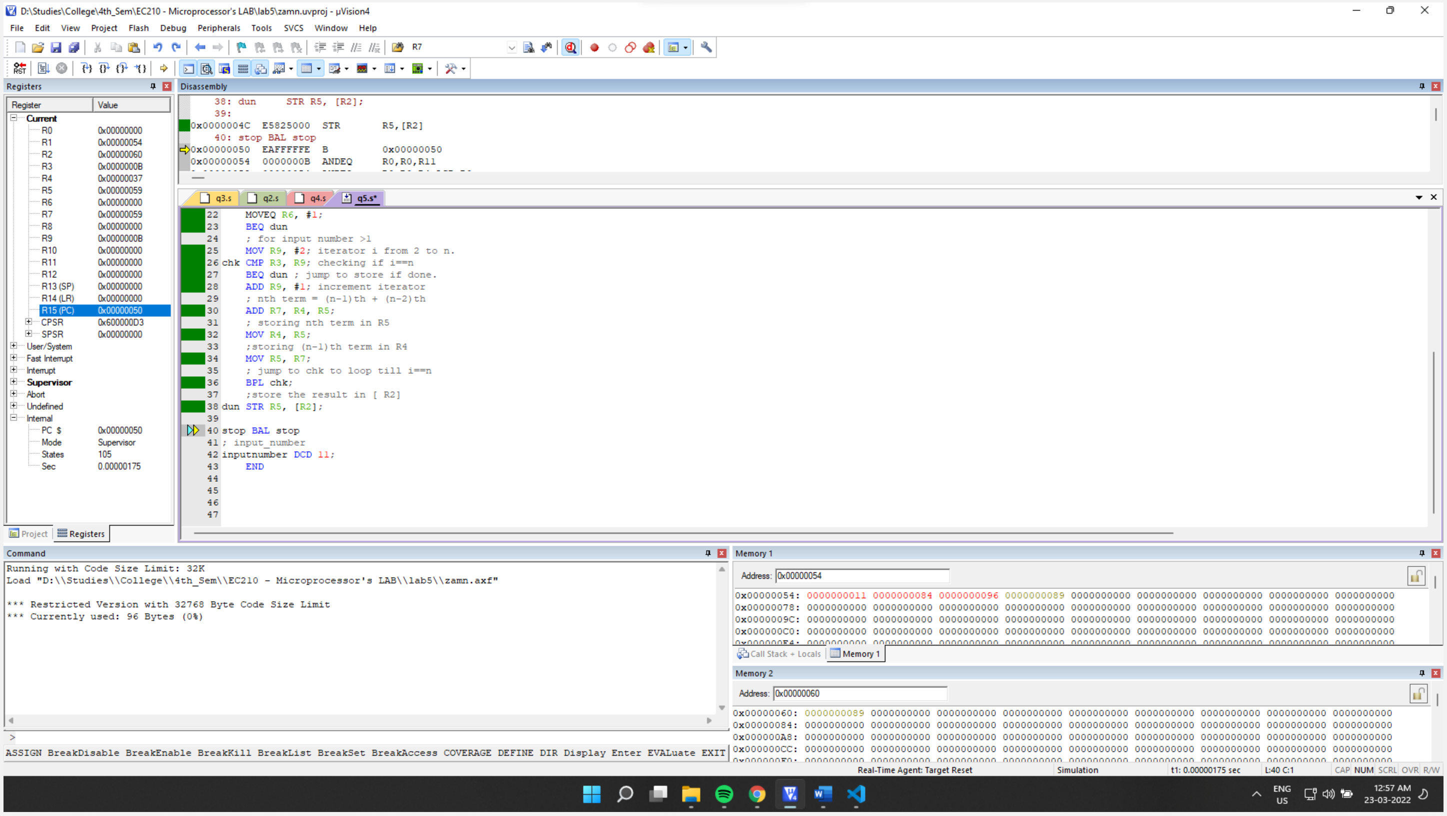
Memory 1 will hold the input no and memory 2 will hold result (Fibonacci no).



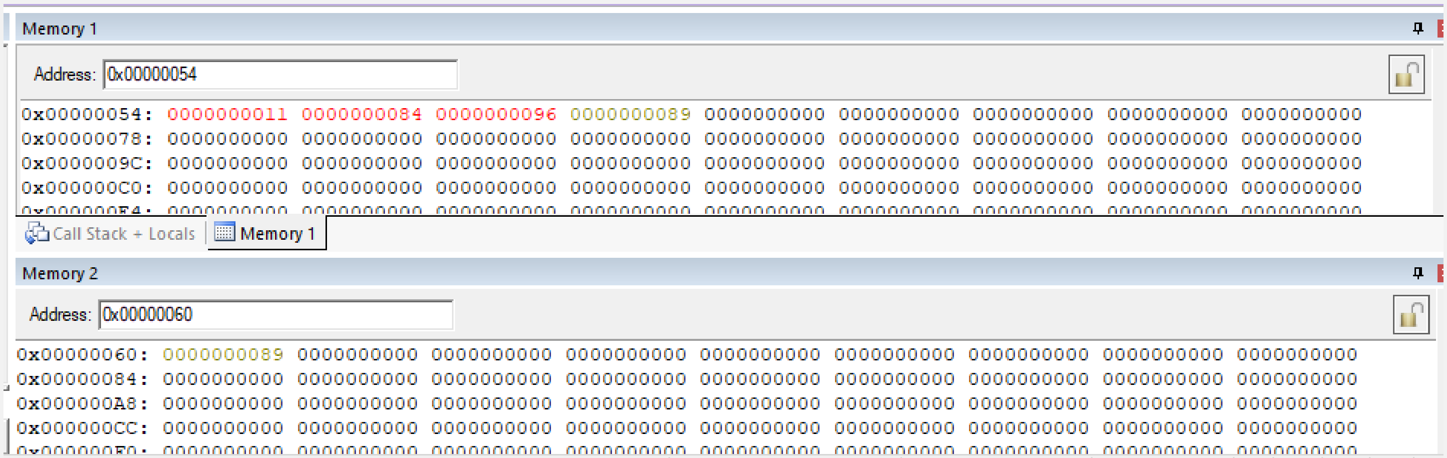
Setup:



Final Output:



Final Memory:



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