EC-210

MICROPROCESSORS LAB

LAB-4

UTKARSH MAHAJAN 201EC164

ARNAV RAJ 201EC109

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

Exercise:

4.2] Reverse the string and check if the string is a palindrome.

->

Source Code:

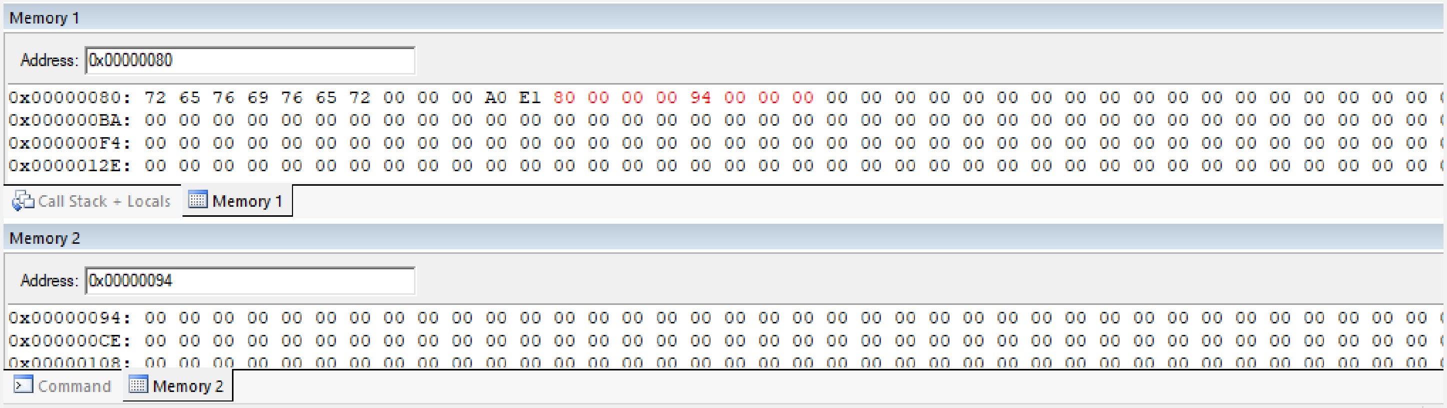
```
        AREA AllocSpace, DATA,NOINIT,READWRITE
        ;Space for storing reversed string
str_rev SPACE 1024;
;Actual Code
        AREA hmm, CODE, READWRITE
        EXPORT Reset_Handler
Reset_Handler
        ;Pointer to the input string
        LDR R1, =str_src ;
        ;Pointer to the reversed string
        LDR R2, =str_rev ;
        ;storing the input's pointer in R8
        ADD R8, R1, #0;
        ;Storing the result's pointer in R9
        ADD R9, R2, #0;
        ; load a byte from input string and update the pointer
lth     LDRB R3, [R8],#1;
        ; counting length (total = length +1)
        ADD R4, #1;
        ; Check for End of string
        CMP R3, #0 ;
        ; If not go back to lth to count length
        BNE lth;
        ; total = (length +1) - 1
        SUB R4, #1;
        ;storing length in R5
        ADD R5, R4, #0;
        ; R8 to point to the last value just before the 0 (NULL).
        SUB R8, #2;
        ; for iteration count R5 = R5 -1;
res SUB R5, #1;
```

```
        ; Loading the value in reverse order in R3
        LDRB R3, [R8],#-1 ; load a byte and update the pointer
        ; Storing the loaded value;
        STRB R3, [R9],#1 ; store byte and update the pointer
        ; checking if R5 is 0. which indicated we are done reversing.
        CMP R5, #0 ; Check for End of string
        ; If its not then loop again.
        BNE res
        ; storing null in R3
        MOV R3, #0;
        ; adding NULL at the end of the reversed string
        STRB R3, [R9];
        ;Restoring length in R5
        ADD R5, R4, #0;
        ;storing starting addresses of input & reversed string.
        ADD R8, R1, #0;
        ADD R9, R2, #0;
        ;checking if the input string is palindrome
        ; by comparing same index bits of input and reversed string.
pal LDRB R3, [R8],#1 ;
        LDRB R4, [R9],#1 ;
        ; checking for end of string
        CMP R3, #0;
        ; As once it reaches end means it's a palindome.
        BEQ ipa;
        ;comparing bytes.
        CMP R3, R4;
        ; if equal -> it can be a palindrome hence looping for next index
        BEQ pal;
        ; If not equan then its not a palindrome
        ;storing 0 in R0 (output register)
np      MOV R0, #0;
        ; jump to stop;
        BAL stop;
        ;storing 1 in R0 (output register) if input is a palindrome
ipa     MOV R0, #1;
        BAL stop;
stop BAL stop
; input_string
str_src DCB "reviver",0;
        NOP
        END
```

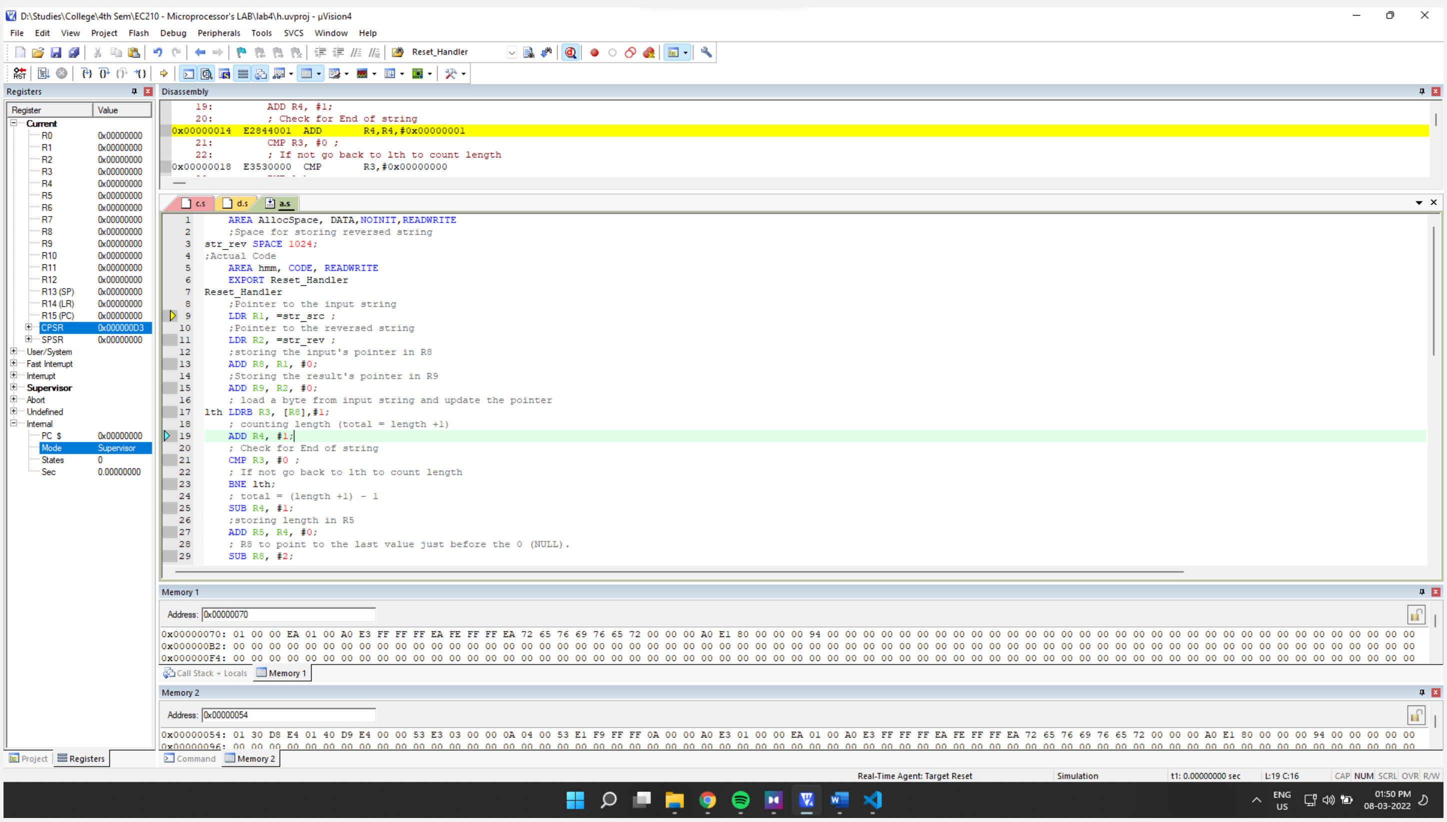
<u>Debugging:</u>

Initial Memory: (after getting the address through register)

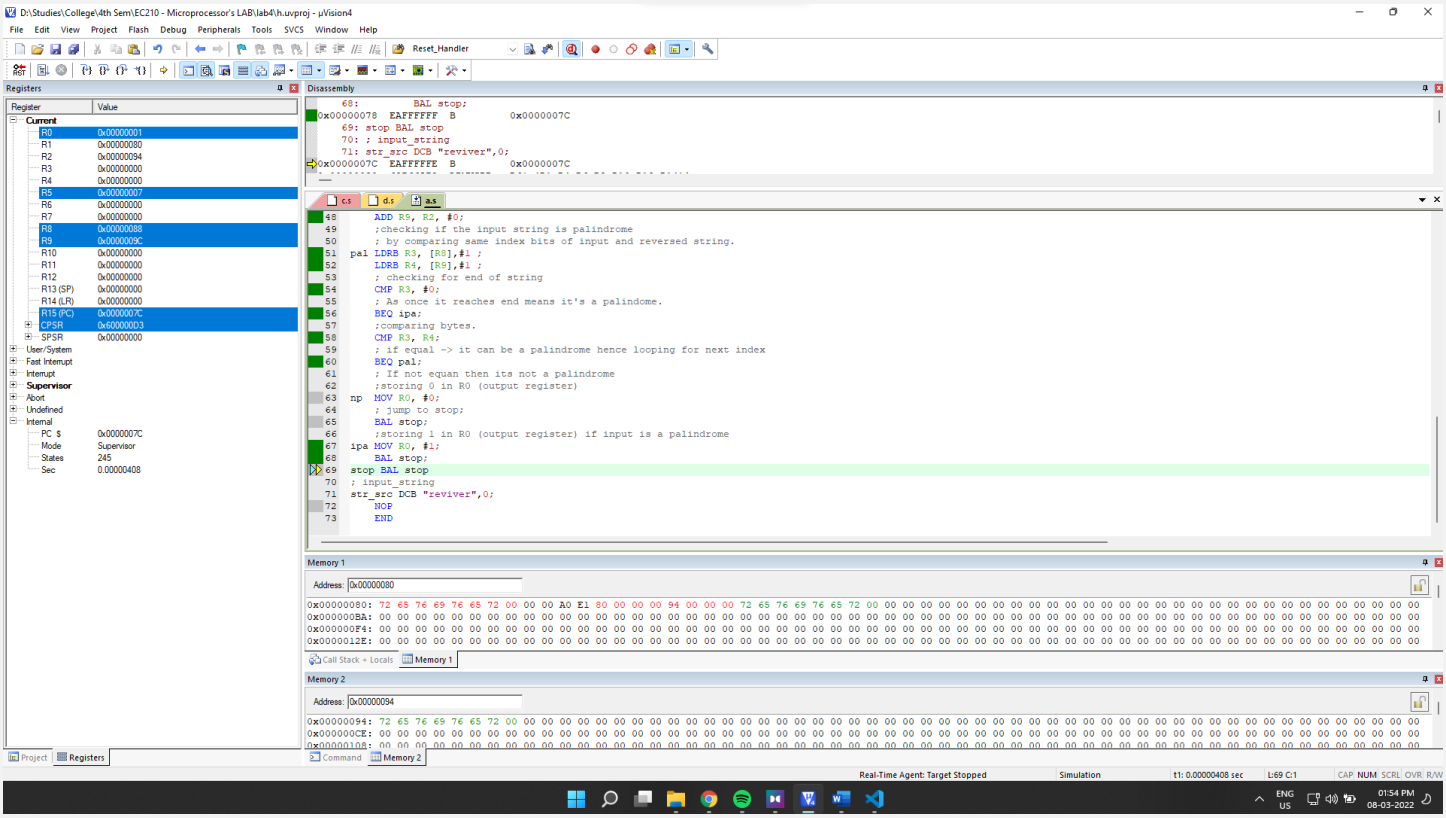
Memory 1 shows input memory and memory 2 shows output(reversed)



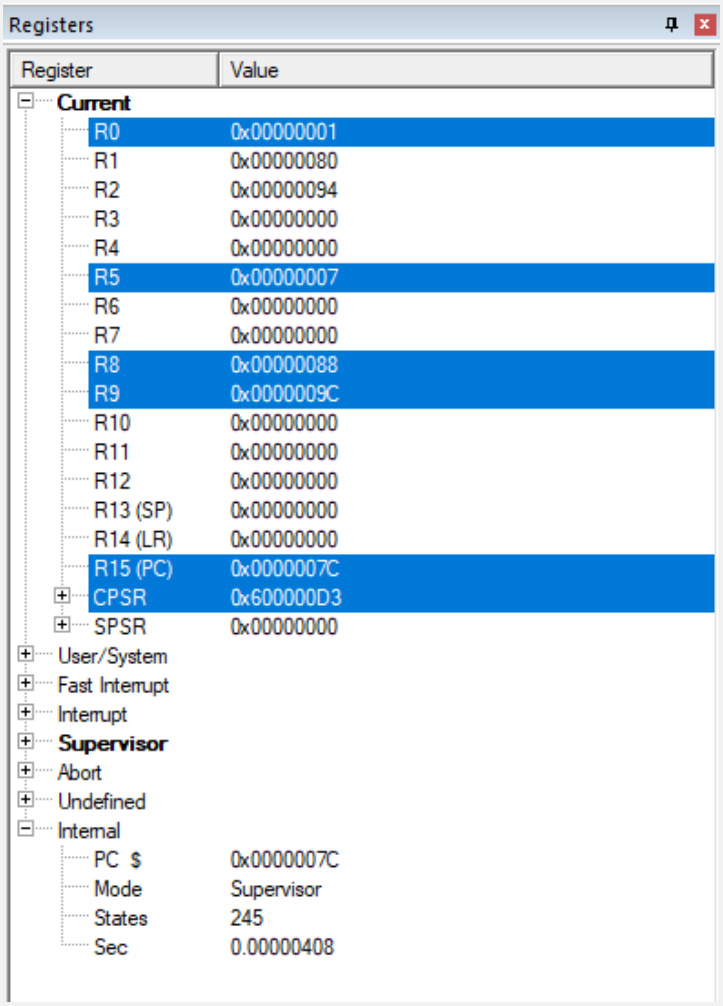
Setup:



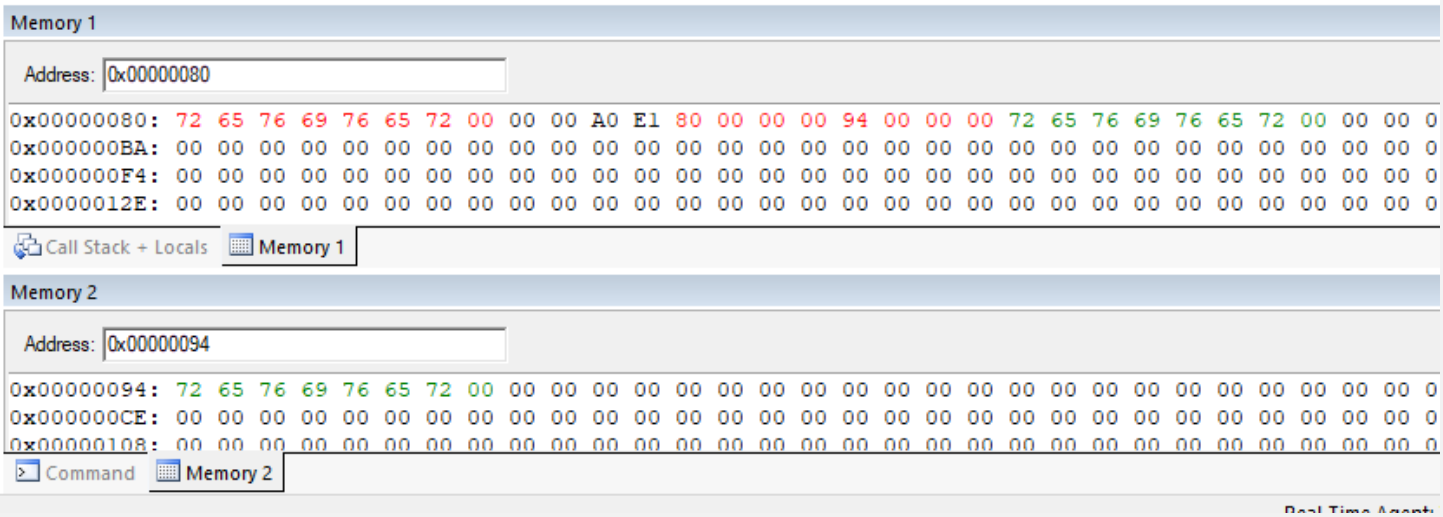
Final Output:

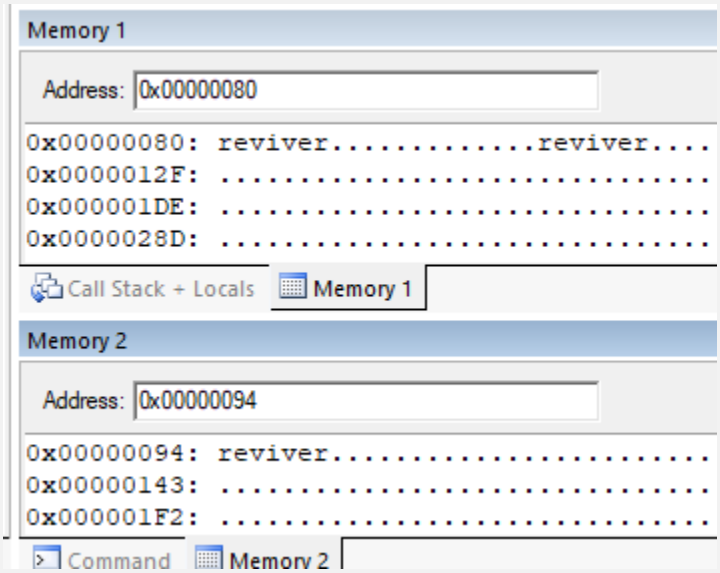


Final Register Values:



Final Memory:

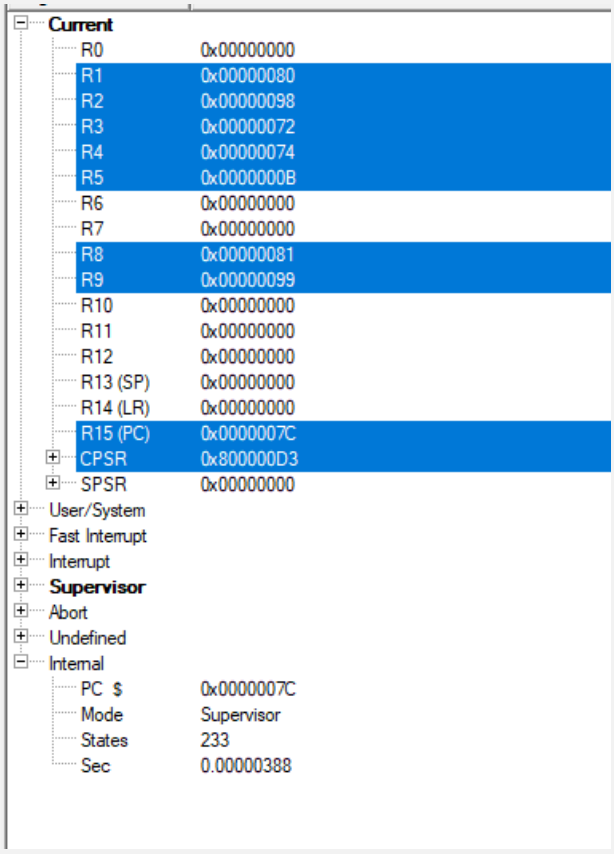




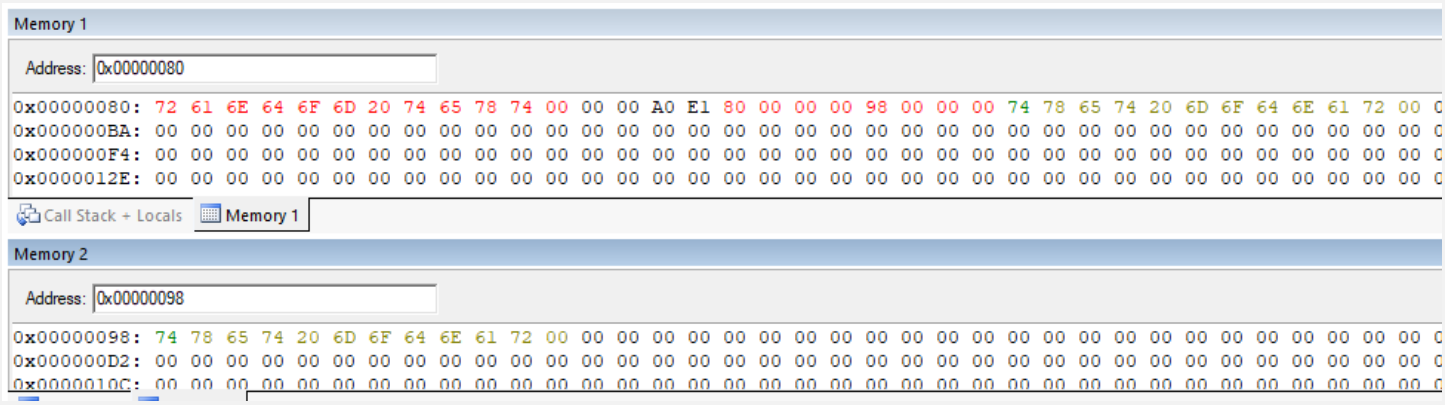
Observation: The output of if the string was palindrome is stored in R0. We can see that its 1 and our input string is actually also an palindrome. We can also see the output memory in memory2. It is reverse of the string that we entered.

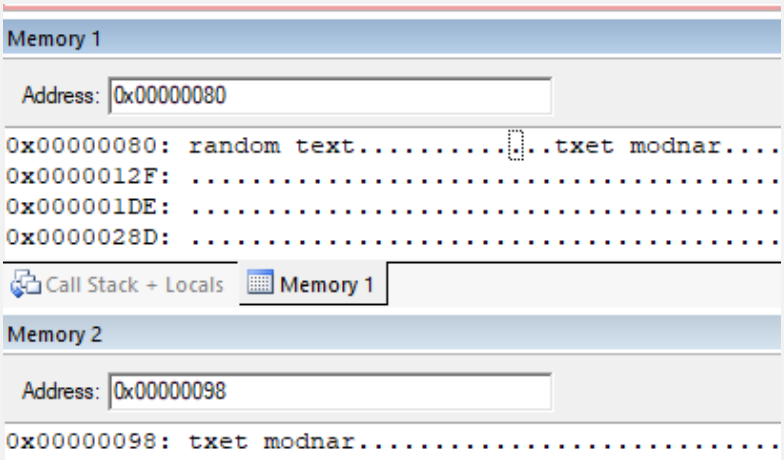
For input string: "random text"

Output register:



Output memory:





Observation: we can see that R0 is indicating the input string is not a palindrome and we can also see the reversed string in output memory (2).

4.3] Find the substring in Main string

->

Source Code:

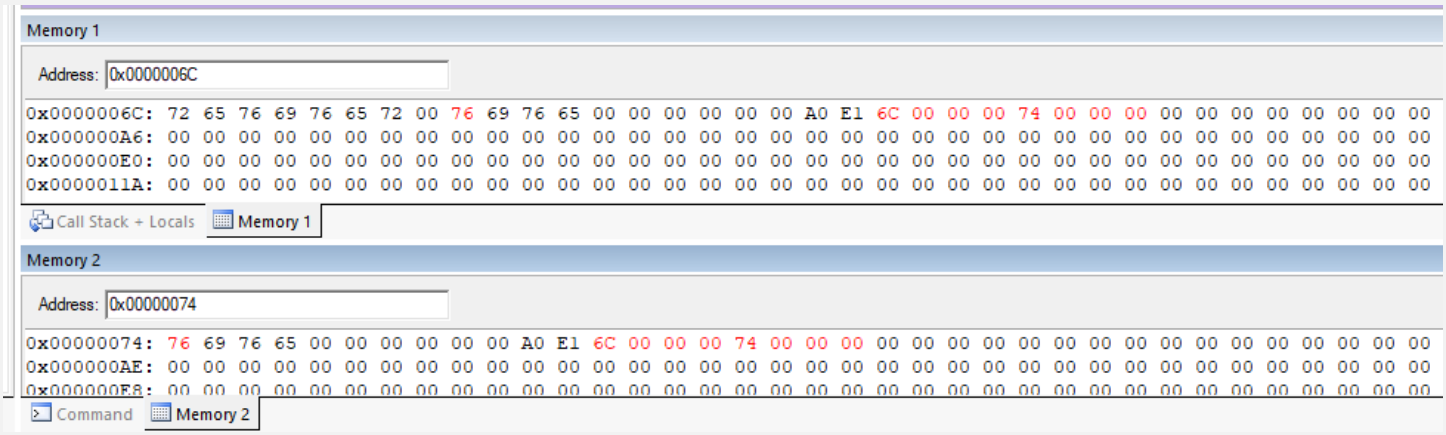
```
        AREA hmm, CODE, READWRITE
        EXPORT Reset_Handler
Reset_Handler
        ; Pointer to the input string
        LDR R1, =str_src ;
        ; Pointer to the input substring
        LDR R2, =sub_str;
        ;storing the input string's pointer in R8
        ADD R8, R1, #0;
        ;storing the input substring's pointer in R9
        ADD R9, R2, #0;
        ; Loading the first byte(char) of the substring
        LDRB R4, [R9],#1 ;
        ; for storing the index.
        MOV R5, #0;
        ;loading the chars of input string.
ind     LDRB R3, [R8],#1 ;
        ; incrementing the index.
        ADD R5, #1;
        ; Check for End of string
        CMP R3, #0 ; Check for End of string
        ; if it reaches the end of string -> not found
        BEQ nf;
        ;comparing 1st char of substring and i'th char of string
        CMP R3, R4;
        ;back to iteration if not equal.
        BNE ind;
        ; else
        ; storing index in R10 for inner loop.
```

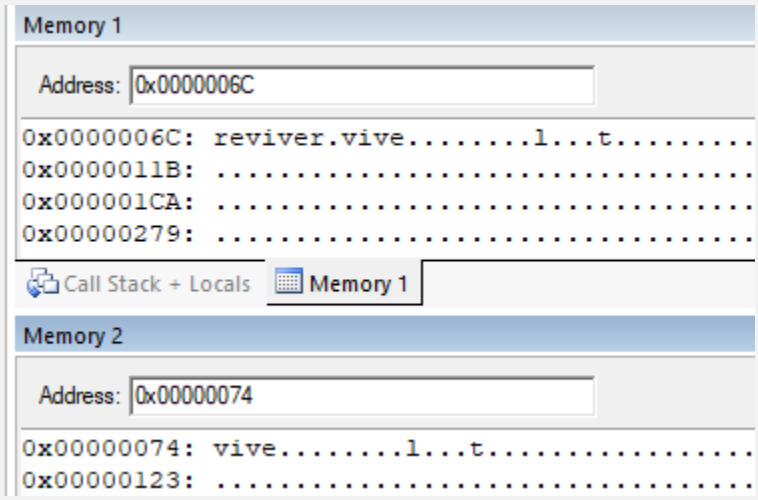
```
        ADD R10, R8, #0;
        ; comparing subsequent characters
ver LDRB R3, [R10],#1 ; load a byte and update the pointer
        LDRB R4, [R9],#1 ; load a byte and update the pointer
        CMP R4, #0 ; Check for End of sub string
        ; if entire substring satisfies, we return.
        BEQ iss;
        ;Check if char's are equal
        CMP R3, R4 ;
        ;If not equal, point R9 to start of string and reload R4
        ADDNE R9, R2, #0;
        LDRBNE R4, [R9],#1 ;
        ;back to main iteration if not equal.
        BNE ind;
        ; if current chars are equal loop further.
        BEQ ver
        ;storing -1 if substring not found.
nf      MOV R0, #-1;
        BAL stop;
        ; storing index of substring in R0 if substring found.
iss     ADD R0, R5, #-1;
        BAL stop;
stop BAL stop
; input string
str_src DCB "reviver",0;
; input substring
sub_str DCB "vive",0;
        NOP
        END
```

Debugging:

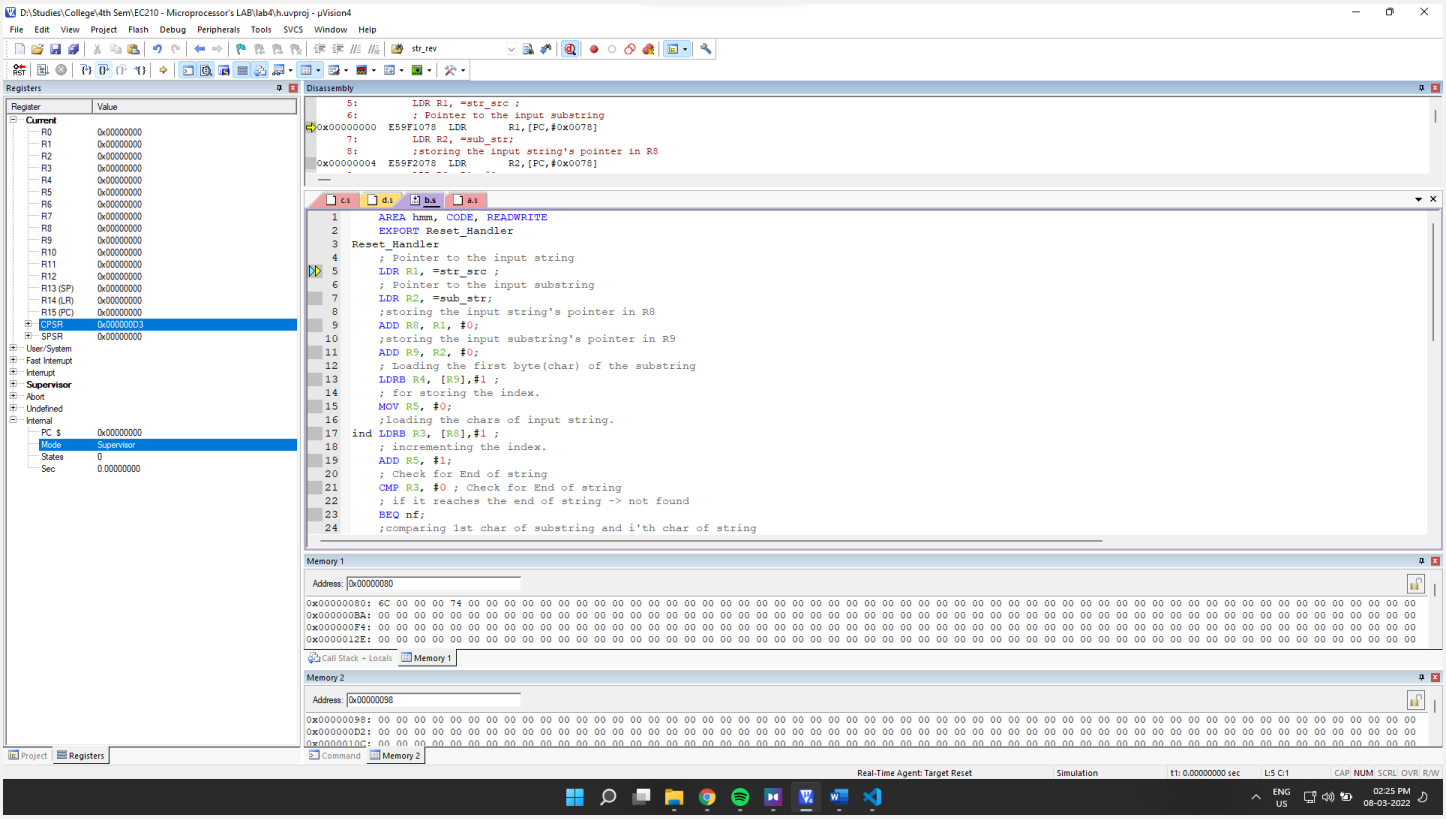
Initial Memory: (after getting the address through register)

Memory 1 shows input string and memory 2 shows input substring

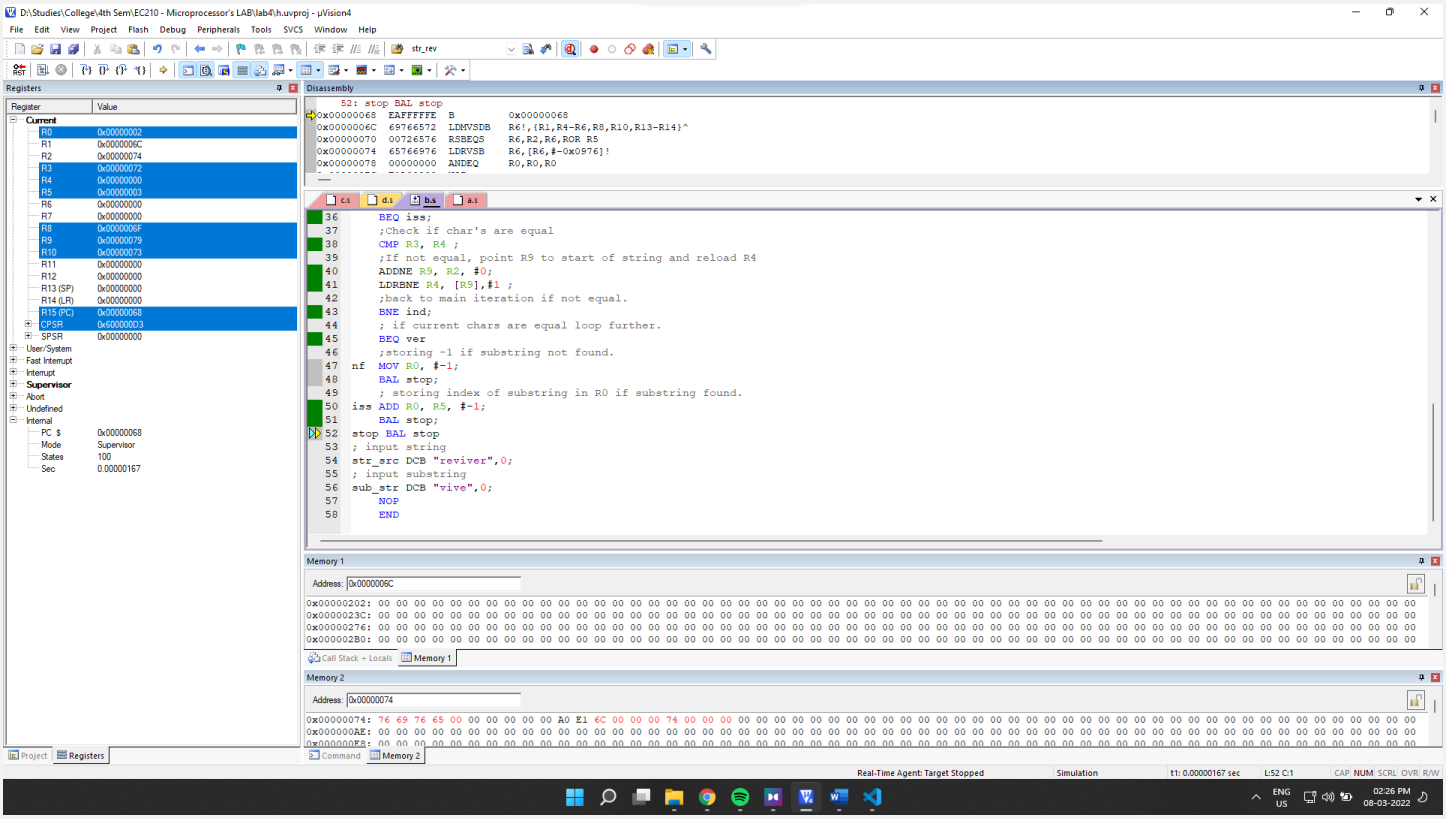

Memory 1
Address: 0x0000006C
0x0000006C: 72 65 76 69 76 65 72 00 76 69 76 65 00 00 00 00 00 00 A0 E1 6C 00 00 00 74 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000000A6: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000000E0: 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 00 00 00 00 00 00 00 00 00 00
0x0000011A: 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 00 00 00 00 00 00 00 00 00 00
Call Stack + Locals
Memory 1
Memory 2
Address: 0x00000074
0x00000074: 76 69 76 65 00 00 00 00 00 00 A0 E1 6C 00 00 00 74 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000000AE: 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 00 00 00 00 00 00 00 00 00 00
0x000000E8: 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 00 00 00 00 00 00 00 00 00 00
Command
Memory 2


Memory 1
Address: 0x0000006C
0x0000006C: reviver.vive.......l...t.........
0x0000011B: ....
0x000001CA: ....
0x00000279: ....
Call Stack + Locals
Memory 1
Memory 2
Address: 0x00000074
0x00000074: vive.......l...t.................
0x00000123: ....

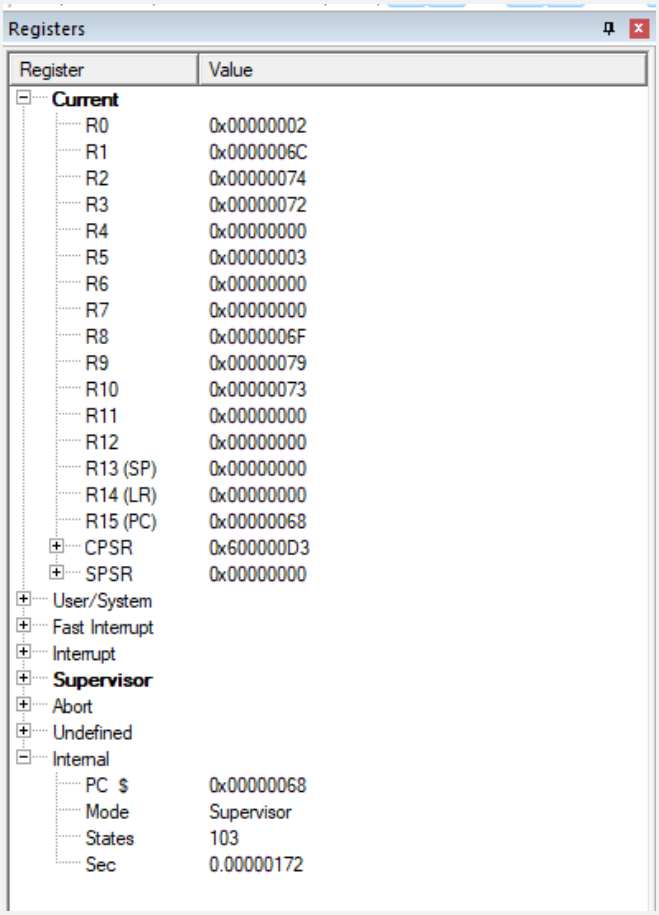
Setup:


D:\Studies\College\4th Sem\EC210 - Microprocessor's LAB\lab4\h.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Disassembly
Memory 1
Memory 2
Project
Registers
Command
Memory 1
Real-Time Agent: Target Reset
Simulation

Final Output:



Final Register Values:



Observation: We can see that the output stored in R0 is 2. Which is correct as our input string a substring were “reviver” and “vive” respectively from which we can see that vive starts at index 2 (counting from 0).

4.4] Insert a substring in main string at given position.

->

Source Code:

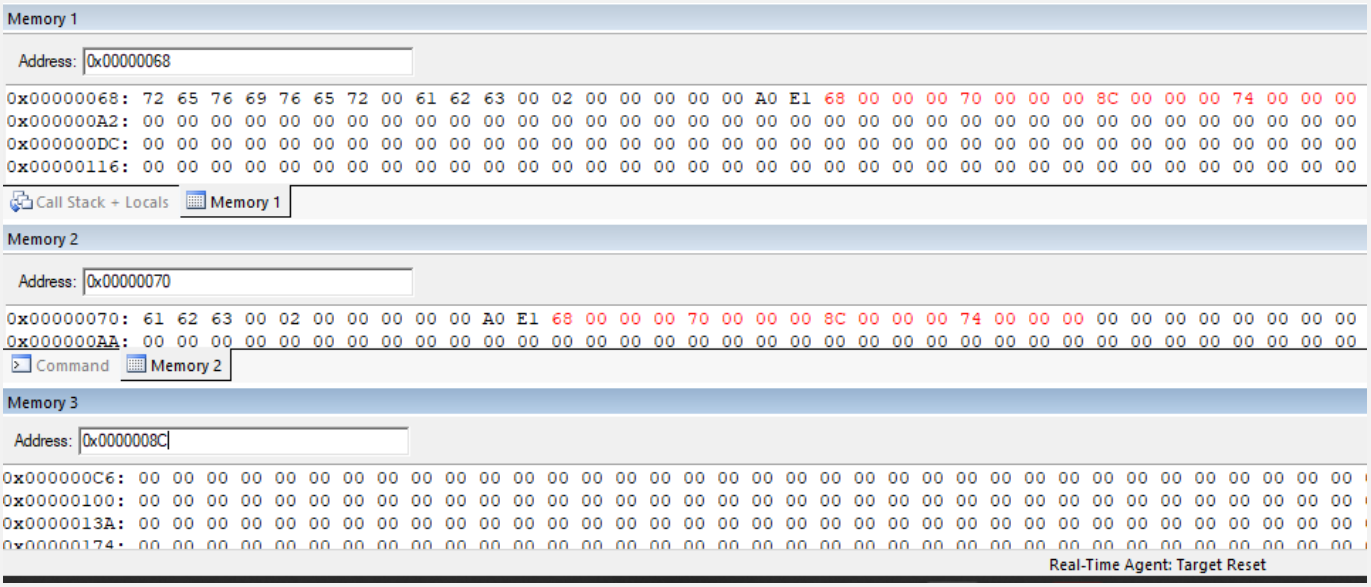
```
        AREA AllocSpace, DATA,NOINIT,READWRITE
;Space for storing the final string
str_rev SPACE 1024

        AREA hmm, CODE, READWRITE
        EXPORT Reset_Handler
Reset_Handler
        ;Pointer to the input string
        LDR R1, =str_src ;
        ;Pointer to the substring
        LDR R2, =sub_str ;
        ;Pointer to result string
        LDR R12, =str_rev ;
        ;Pointer to index
        LDR R11, =sub_index ;
        ;loading index in R10;
        LDRB R10, [R11];
        ;storing the input string's pointer in R8
        ADD R8, R1, #0;
        ;storing the input substring's pointer in R9
        ADD R9, R2, #0;
        ;for iteration till the insertion index
        MOV R7, #0;
        ;result's pointer in R12.
        ADD R6, R12, #0;
;storing the prefix part of string in result(before the insertion index)
        ;load a byte and update the pointer
```

```
pref LDRB R3, [R8],#1 ;
        ;store byte and update the pointer
        STRB R3, [R6], #1;
        ;incrementign R7.
        ADD R7, #1;
        ;check if the insertion index is reached.
        CMP R7, R10;
        ; if not then loop. again to pref.
        BNE pref
        ; storing the substring in consequent index in result.
        ; load a byte and update the pointer
sbst LDRB R3, [R9],#1 ;
        ;store byte and update the pointer
        STRB R3, [R6], #1;
        ;Check for End of string
        CMP R3, #0 ;
        ; If not, then loop.
        BNE sbst
        ; subtract result pointer, so the null value can be rewritten
        SUB R6, #1;
;storing the suffix part, the part that existed at right of insertion index
        ;load a byte and update the pointer
sufx LDRB R3, [R8],#1 ;
        ; store byte and update the pointer
        STRB R3, [R6], #1;
        ; Check for End of string
        CMP R3, #0 ;
        ; if not , loop again
        BNE sufx
done MOV R0, #1;
        BAL stop;
stop BAL stop

;input string
str_src DCB "reviver",0;
;input substring
sub_str DCB "abc",0;
;input insertion index
sub_index DCB 2;
        NOP
        END
```

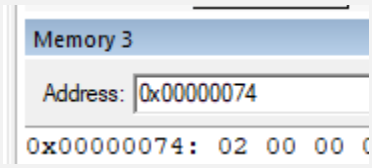
Debugging:

Initial Memory: (after getting the address through register)

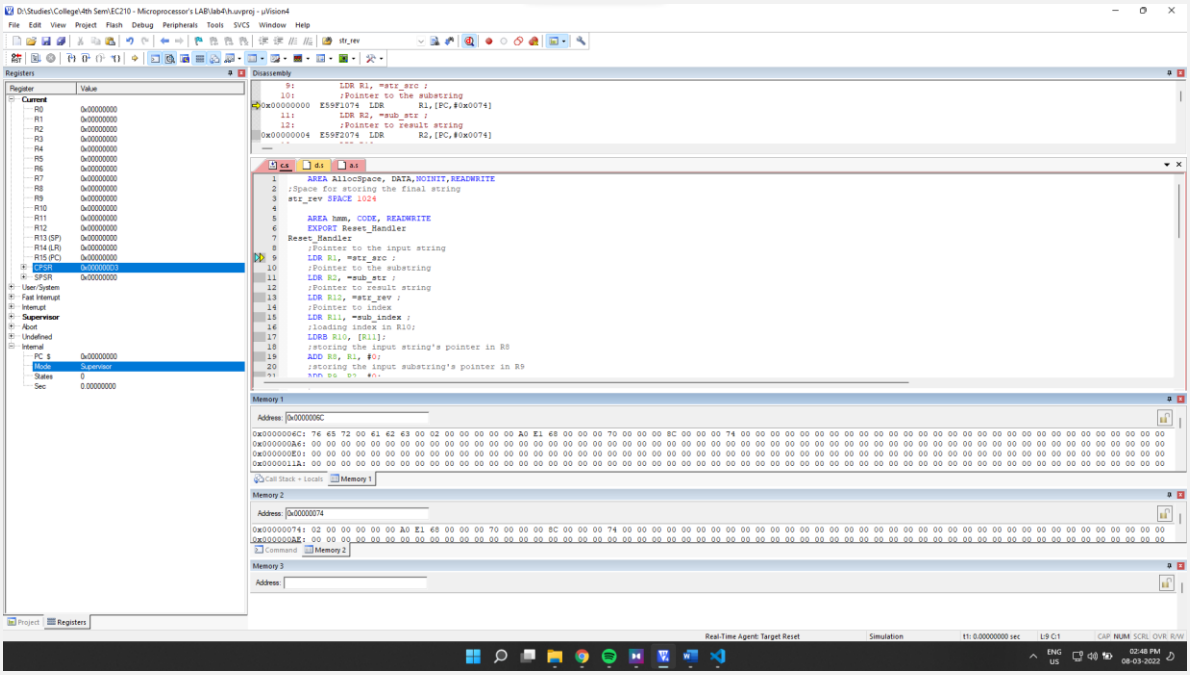
Memory 1 shows input string, memory 2 input substring and memory 3 will hold output result string.



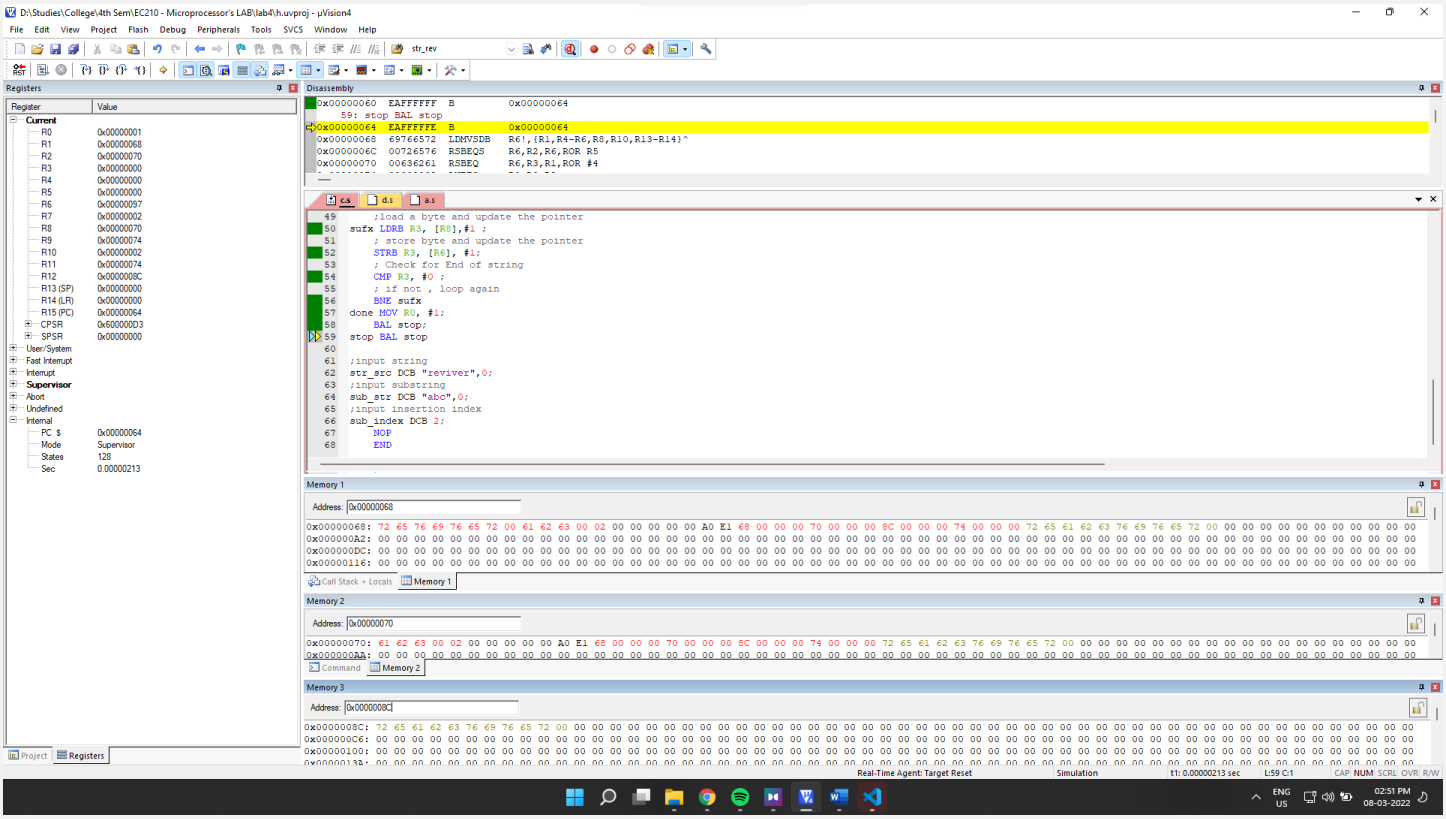
Index:



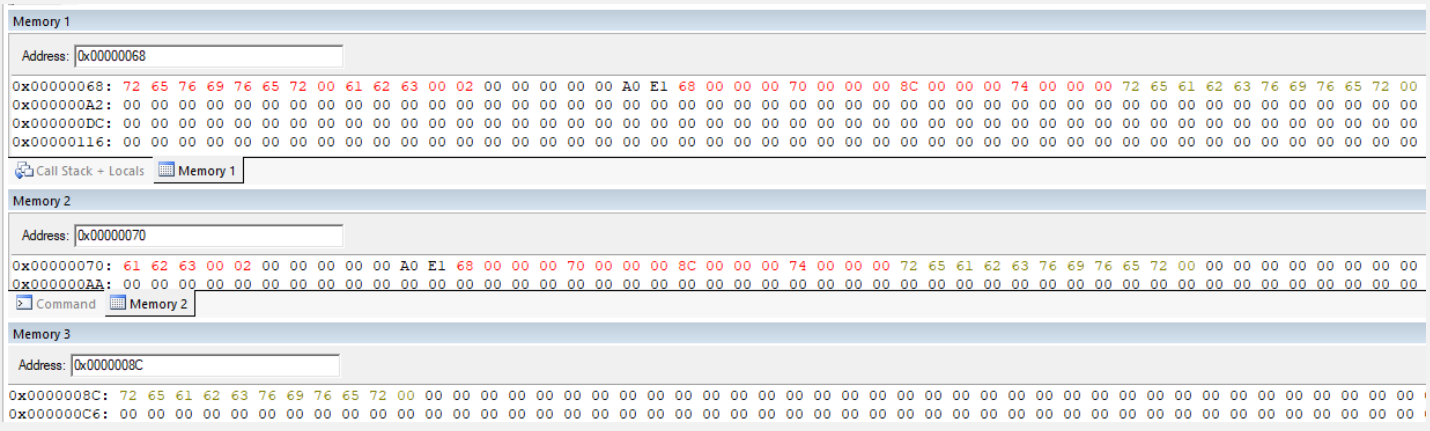
Setup:

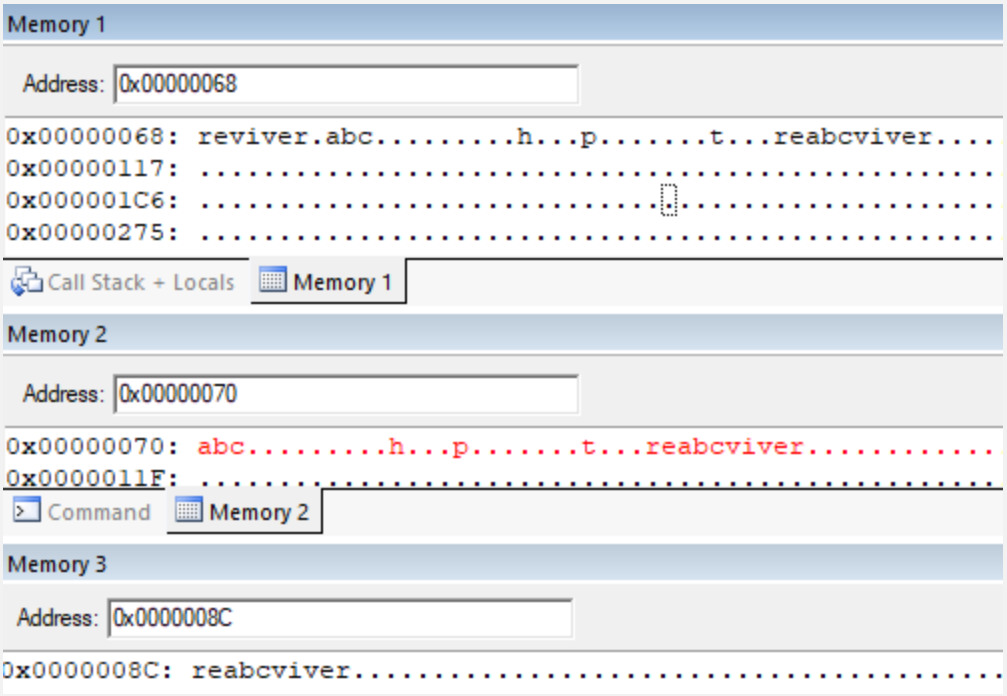


Final Output:



Final Memory:





Observation: We can see that in memory3, we have our final string. We can compare both the input string and substring with that of in memory 3 and verify that it contains the string with substring at insertion index.

4.5] Squeeze a string removing all the blank spaces and store it in the same location

-> We will copy the input string to the space. And then we will operate on it. This way we can directly operate on the same memory.

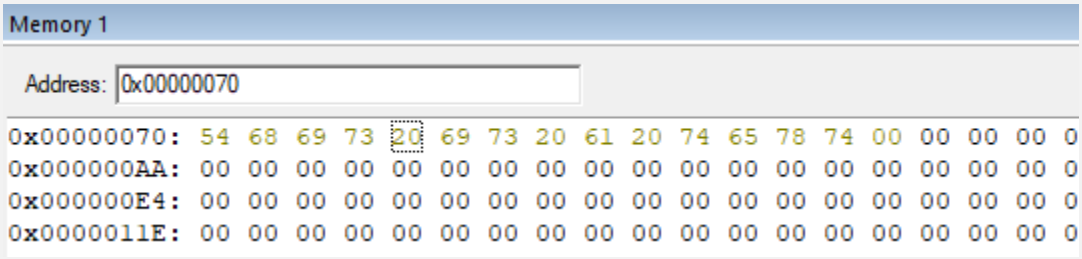
Source Code:

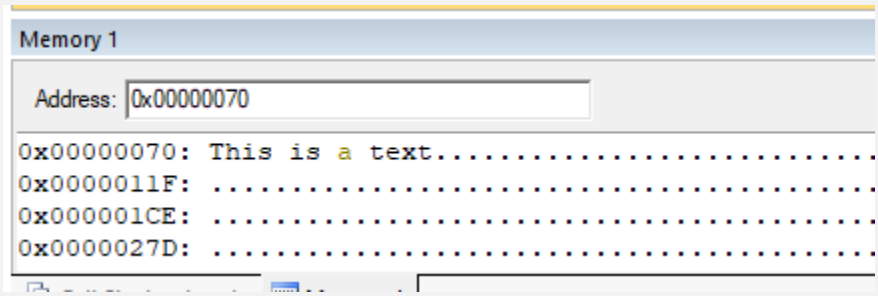
```
        AREA AllocSpace, DATA,NOINIT,READWRITE
;Space for the input string
str_main SPACE 1024

        AREA hmm, CODE, READWRITE
        EXPORT Reset_Handler
Reset_Handler
        ;Pointer to the string
        LDR R1, =str_main ;
```

```
        ;Pointer to the input string
        LDR R2, =str_src ;
        ;storing the string's pointer in R8
        ADD R4, R1, #0;
        ;storing the input's pointer in R8
        ADD R5, R2, #0;
        ;for storing the input String;
        ; loading characters from input string.
fill LDRB R3, [R5], #1;
        ;storing the character in result space
        STRB R3, [R4], #1; store byte and update the pointer
        ;checking if end of line is reached
        CMP R3, #0;
        ; if not then loop.
        BNE fill;
        ; for storing no of zeroes encountered
        MOV R4, #0;
        ; Pointer to the string in R5
        MOV R5, R1;
        ;loading character from string
sqz LDRB R3, [R5];
        ; checking if its a space
        CMP R3, #32;
        ; if its a space then add 1 to R4.
        ADDEQ R4, #1;
        ; if not then calculate index to move towards left.
        SUBNE R6, R5, R4;
; and also store the current value towards left at that index.
        STRBNE R3, [R6];
        ; update the string's iterator
        ADD R5, #1;
        ;check if string ends.
        CMP R3, #0;
        ; if not then loop.
        BNE sqz;
        ;R0 to change to 1 at the end of the loop.
done MOV R0, #1;
        BAL stop;
stop BAL stop;
str_src DCB "This is a text",0;
        NOP
        END
```

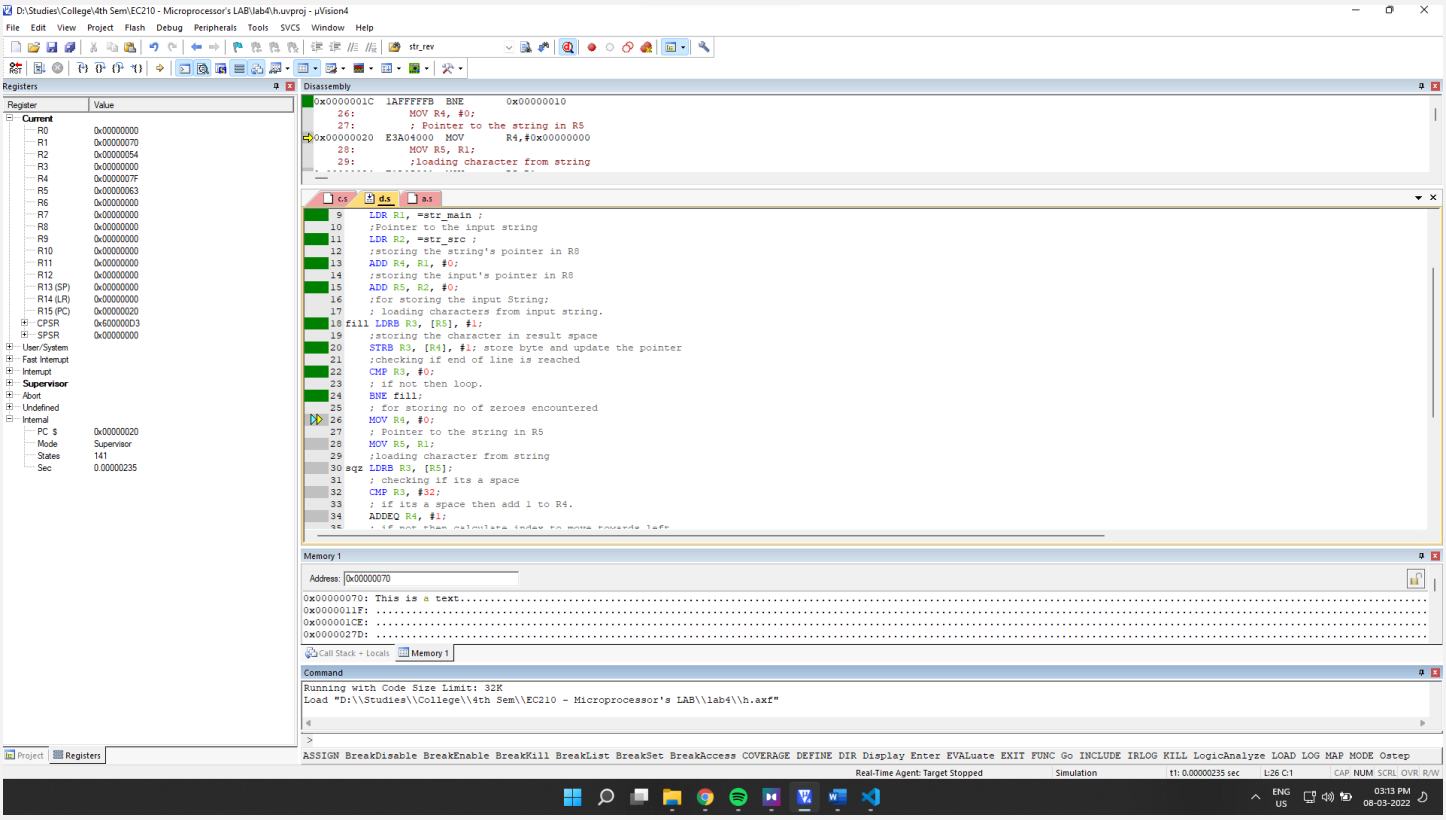
Initial Memory: (after getting the address through register and loading the value into the reserved space)

Memory 1 will hold the input string and later also the result.

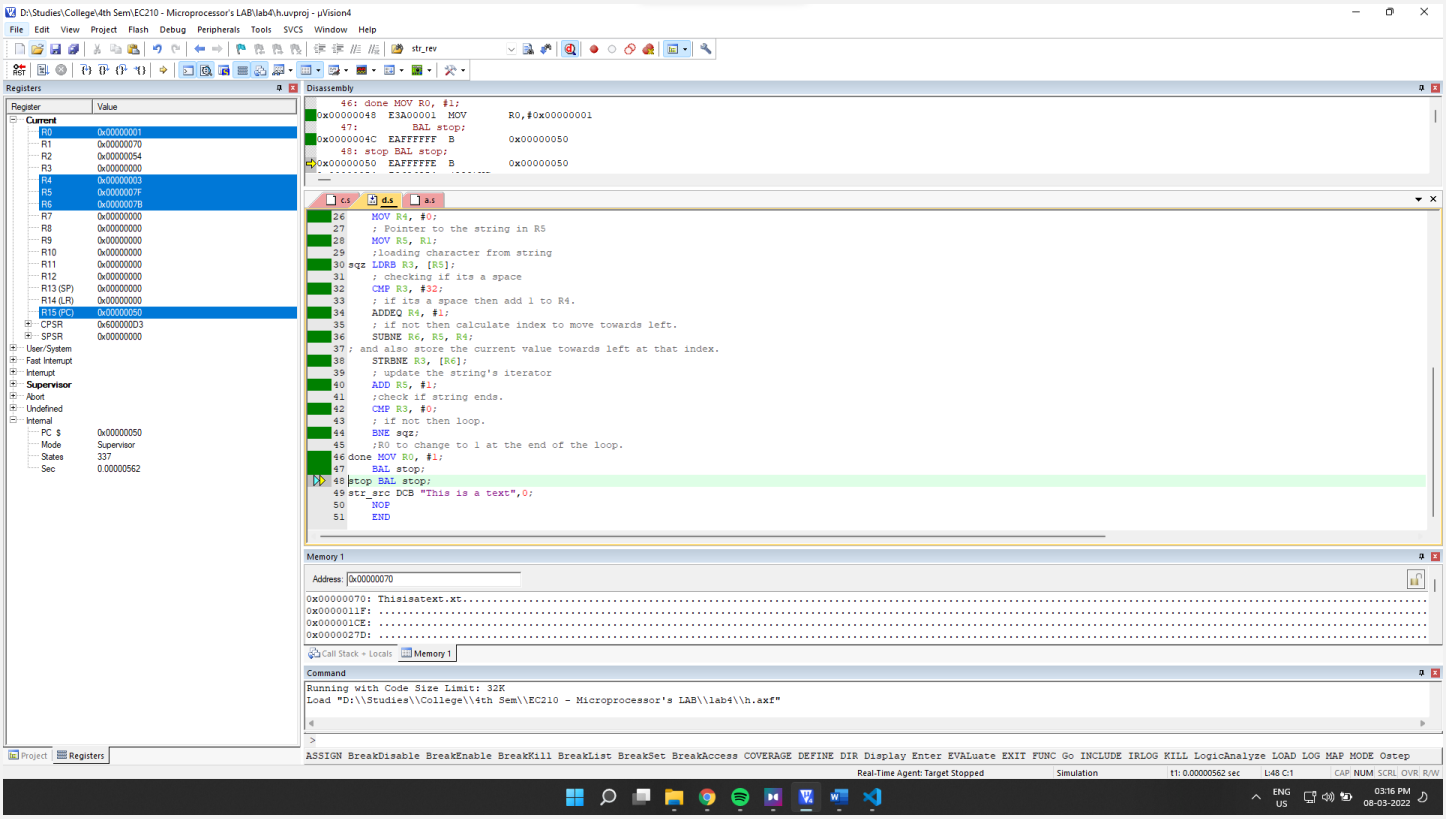




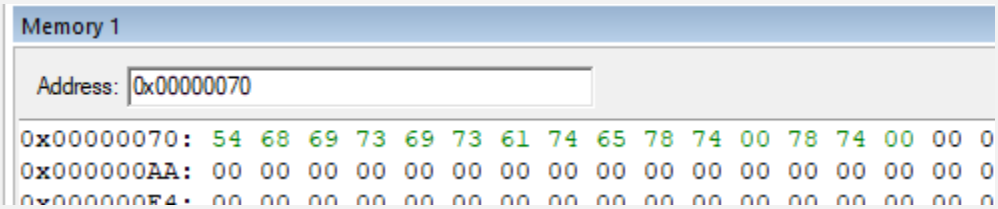
Setup:

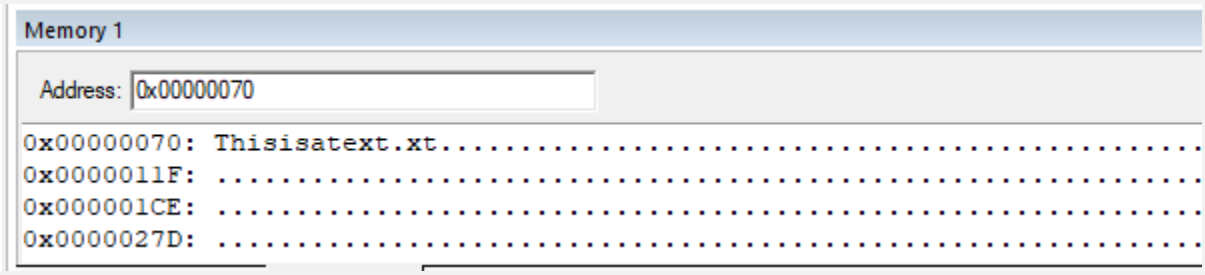


Final Output:



Final Memory:





Observation: We can see that the final string is squeezed and all spaces are removed.