

# **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;
```

```

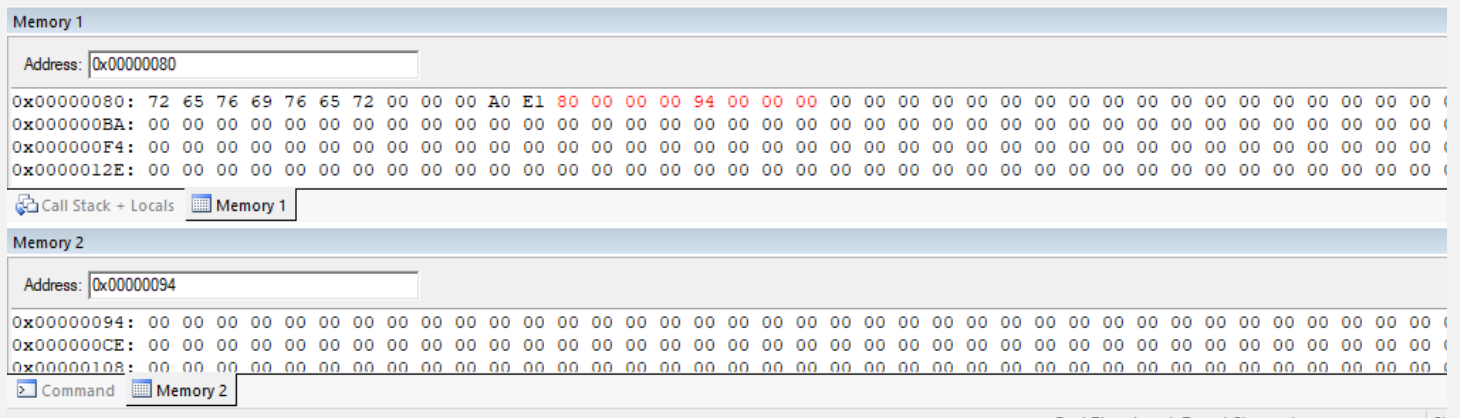
; Loading the value in reverse order in R3
LDRB R3, [R8], #-1 ; load a byte and update the pointer
; Storing the loaded value;
STRB R3, [R9], #1 ; store byte and update the pointer
; checking if R5 is 0. which indicated we are done reversing.
CMP R5, #0 ; Check for End of string
; If its not then loop 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 equal 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

```

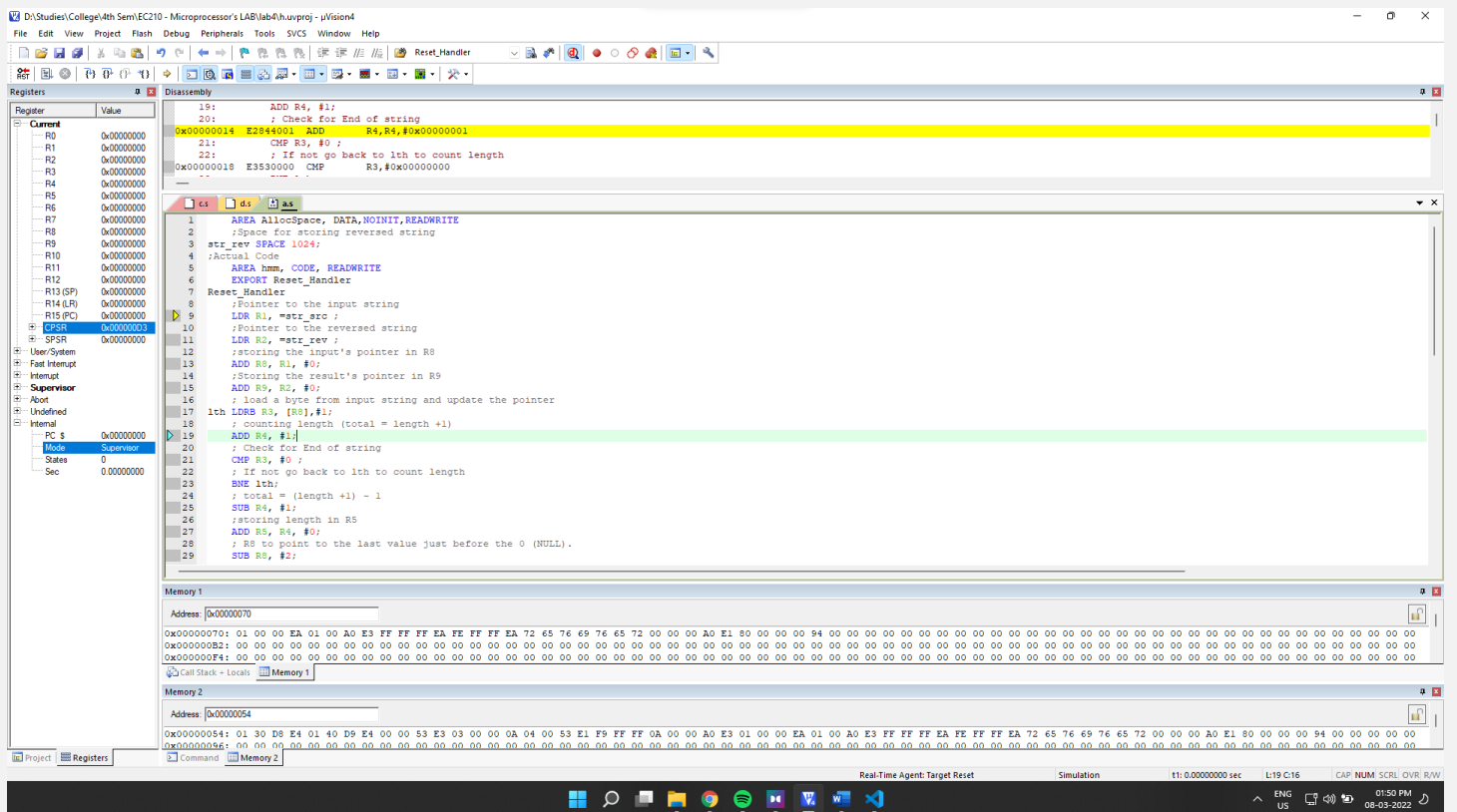
# Debugging:

Initial Memory: (after getting the address through register)

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



# Setup:



# Final Output:



## Final Memory:

Memory 1

Address:

0x00000080: 72 65 76 69 76 65 72 00 00 00 A0 E1 80 00 00 00 94 00 00 00 72 65 76 69 76 65 72 00 00 00 0

0x000000BA: 00 0

0x000000F4: 00 0

0x0000012E: 00 0

Call Stack + Locals Memory 1

Memory 2

Address:

0x00000094: 72 65 76 69 76 65 72 00 0

0x000000CE: 00 0

0x00000108: 00 0

> Command Memory 2

Real Time Assembl...

Memory 1

Address:

0x00000080: reviver.....reviver....

0x0000012F: .....

0x000001DE: .....

0x0000028D: .....

Call Stack + Locals Memory 1

Memory 2

Address:

0x00000094: reviver.....

0x00000143: .....

0x000001F2: .....

> Command Memory 2

**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:

Current	
R0	0x00000000
R1	0x00000080
R2	0x00000098
R3	0x00000072
R4	0x00000074
R5	0x0000000B
R6	0x00000000
R7	0x00000000
R8	0x00000081
R9	0x00000099
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x0000007C
CPSR	0x800000D3
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC \$	0x0000007C
Mode	Supervisor
States	233
Sec	0.00000388

## Output memory:

Memory 1

Address: 0x00000080

0x00000080: 72 61 6E 64 6F 6D 20 74 65 78 74 00 00 00 A0 E1 80 00 00 00 98 00 00 00 74 78 65 74 20 6D 6F 64 6E 61 72 00 00  
0x000000BA: 00  
0x000000F4: 00  
0x0000012E: 00

Call Stack + Locals Memory 1

Memory 2

Address: 0x00000098

0x00000098: 74 78 65 74 20 6D 6F 64 6E 61 72 00  
0x000000D2: 00  
0x0000010C: 00

Memory 1

Address: 0x00000080

0x00000080: random text.....0..txet modnar....  
0x0000012F: .....  
0x000001DE: .....  
0x0000028D: .....

Call Stack + Locals Memory 1

Memory 2

Address: 0x00000098

0x00000098: txet modnar.....

**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



# Final Output:

The screenshot displays the uVision4 IDE interface. The **Registers** window on the left shows the current register values. The **Disassembly** window in the center shows the assembly code with the current instruction highlighted. The **Memory** window at the bottom shows the memory contents.

Register	Value
R0	0x00000002
R1	0x0000006C
R2	0x00000074
R3	0x00000072
R4	0x00000000
R5	0x00000003
R6	0x00000000
R7	0x00000000
R8	0x0000006F
R9	0x00000079
R10	0x00000073
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000068
CPSR	0x600000D3
SPSR	0x00000000

```
51: stop BAL stop
0x00000068 EAffFF7F B 0x00000068
0x0000006C 69766572 LDMVSD B R6!,R4-R6,R8,R10,R13-R14!
0x00000070 00726576 RSBEQS R6,R2,R6,ROR R5
0x00000074 65766976 LDRVSB R6,[R6,#-0x0976]!
0x00000078 00000000 ADDEQ R0,R0,R0

36 BEQ iss;
37 ;Check if char's are equal
38 CMP R3, R4 ;
39 ;if not equal, point R9 to start of string and reload R4
40 ADDNE R9, R2, #0;
41 LDRBNE R4, [R9],#1 ;
42 ;back to main iteration if not equal.
43 BNE ind;
44 ; if current chars are equal loop further.
45 BEQ ver
46 ;storing -1 if substring not found.
47 nf MOV R0, #-1;
48 BAL stop;
49 ; storing index of substring in R0 if substring found.
50 iss ADD R0, R5, #-1;
51 BAL stop;
52 stop BAL stop
53 ; input string
54 str_src DCB "reviver",0;
55 ; input substring
56 sub_str DCB "vive",0;
57 NOP
58 END
```

# Final Register Values:

Register	Value
Current	
R0	0x00000002
R1	0x0000006C
R2	0x00000074
R3	0x00000072
R4	0x00000000
R5	0x00000003
R6	0x00000000
R7	0x00000000
R8	0x0000006F
R9	0x00000079
R10	0x00000073
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000068
CPSR	0x600000D3
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC \$	0x00000068
Mode	Supervisor
States	103
Sec	0.00000172

**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;
    ;increment 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.

Memory 1

Address: 0x00000068

0x00000068: 72 65 76 69 76 65 72 00 61 62 63 00 02 00 00 00 00 00 00 00 A0 E1 68 00 00 00 70 00 00 00 8C 00 00 00 74 00 00 00  
0x000000A2: 00  
0x000000DC: 00  
0x00000116: 00

Call Stack + Locals Memory 1

Memory 2

Address: 0x00000070

0x00000070: 61 62 63 00 02 00 00 00 00 00 00 00 A0 E1 68 00 00 00 70 00 00 00 8C 00 00 00 74 00 00 00 00 00 00 00 00 00 00 00  
0x000000AA: 00

Command Memory 2

Memory 3

Address: 0x0000008C

0x000000C6: 00  
0x00000100: 00  
0x0000013A: 00  
0x00000174: 00

Real-Time Agent: Target Reset

## Index:

Memory 3

Address: 0x00000074

0x00000074: 02 00 00 00

## Setup:

D:\Studies\College\4th Sem\EC210 - Microprocessor's LAB\lab4\angrej - 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  
PC 0x00000000  
Status 0  
Sec 0x00000000

Disassembly

9: LDR R1, =str\_rev ;  
10: ;Pointer to the substring  
11: LDR R2, =sub\_str ;  
12: ;Pointer to result string  
13: LDR R3, =str\_rev ;  
14: ;Pointer to index  
15: LDR R4, =sub\_index ;  
16: ;loading index in R10  
17: LDR R10, [R1];  
18: ;storing the input string's pointer in R8  
19: ADD R8, R1, #0;  
20: ;storing the input substring's pointer in R9  
21: ADD R9, R2, #0;

Memory 1

Address: 0x00000068

0x00000068: 72 65 76 69 76 65 72 00 61 62 63 00 02 00 00 00 00 00 00 00 A0 E1 68 00 00 00 70 00 00 00 8C 00 00 00 74 00 00 00  
0x000000A2: 00  
0x000000DC: 00  
0x00000116: 00

Memory 2

Address: 0x00000070

0x00000070: 61 62 63 00 02 00 00 00 00 00 00 00 A0 E1 68 00 00 00 70 00 00 00 8C 00 00 00 74 00 00 00 00 00 00 00 00 00 00 00  
0x000000AA: 00  
0x0000013A: 00  
0x00000174: 00

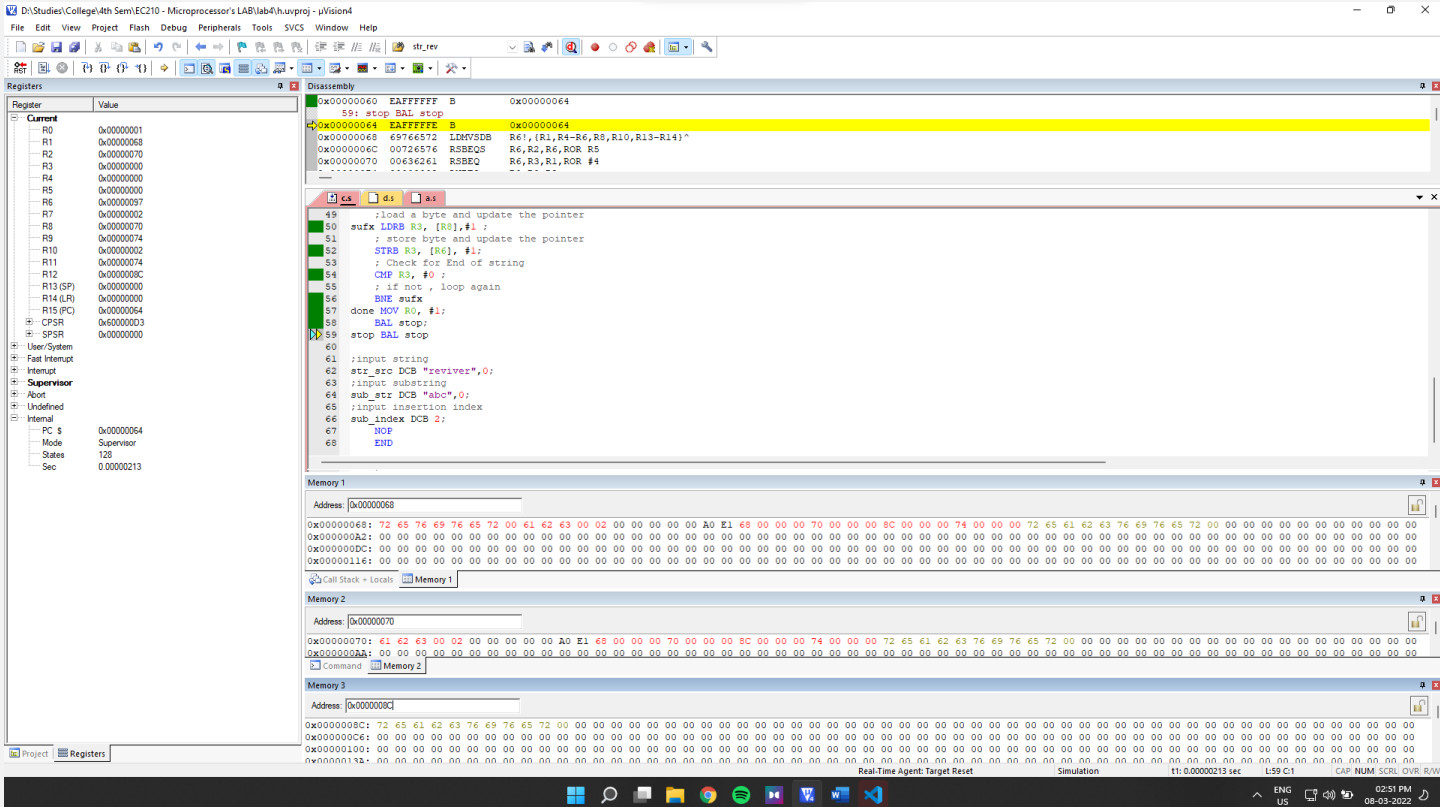
Memory 3

Address: 0x0000008C

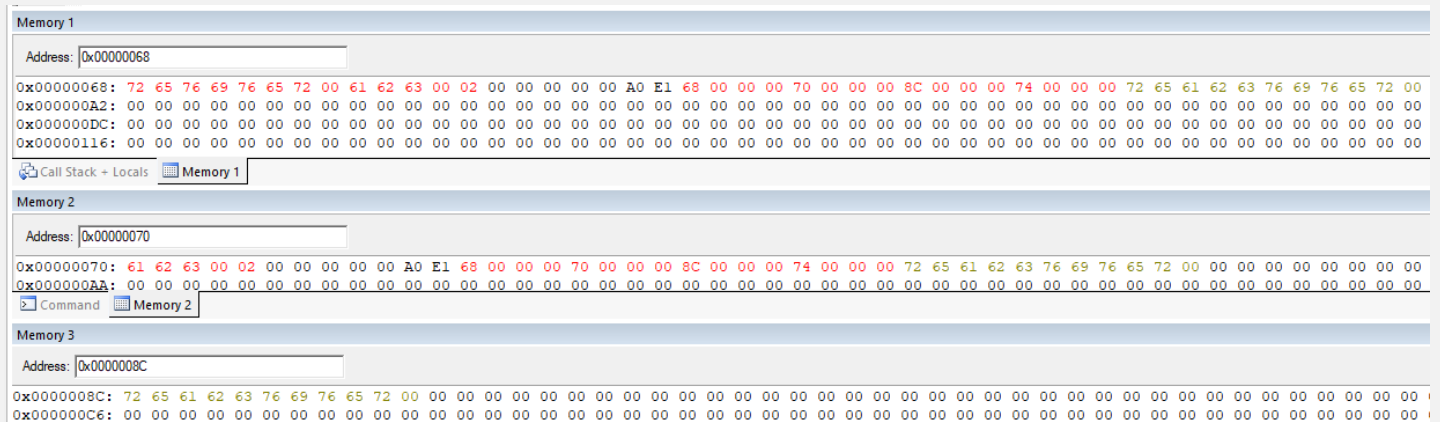
0x000000C6: 00  
0x00000100: 00  
0x0000013A: 00  
0x00000174: 00

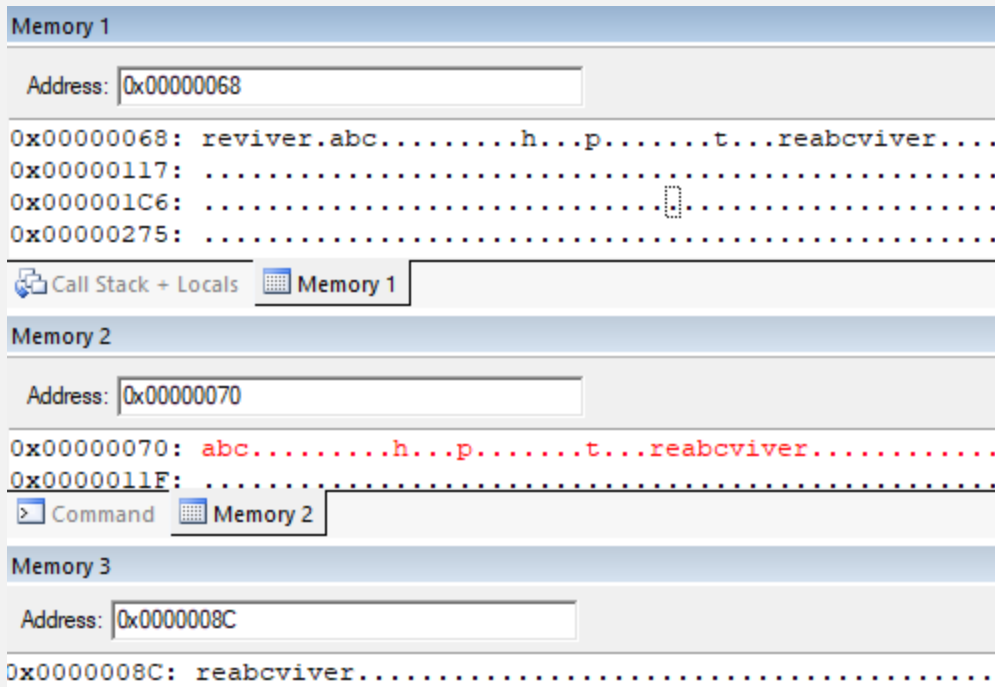
Real-Time Agent: Target Reset

## 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.

Memory 1																
Address:		0x00000070														
0x00000070:	54	68	69	73	20	69	73	20	61	20	74	65	78	74	00	00
0x000000AA:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x000000E4:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x0000011E:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Memory 1	
Address:	<input type="text" value="0x00000070"/>
0x00000070: This is a text.....	
0x0000011F: .....	
0x000001CE: .....	
0x0000027D: .....	

**Setup:**

Registers

Register	Value
Current	0x00000000
R0	0x00000000
R1	0x00000070
R2	0x00000054
R3	0x00000000
R4	0x0000007F
R5	0x00000063
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000000
CPSR	0x60000003
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC	0x00000020
Mode	Supervisor
States	141
Sec	0.00000235

Disassembly

```
0x0000001C 1AFFFFB BNE 0x00000010
26: MOV R4, #0;
27: ; Pointer to the string in R5
0x00000020 E3A04000 MOV R4,#0x00000000
28: MOV R5, R1;
29: ;loading character from string
...
```

Memory 1

Address	Value
0x00000070	This is a text.....
0x0000011F	.....
0x000001CE	.....
0x0000027D	.....

Command

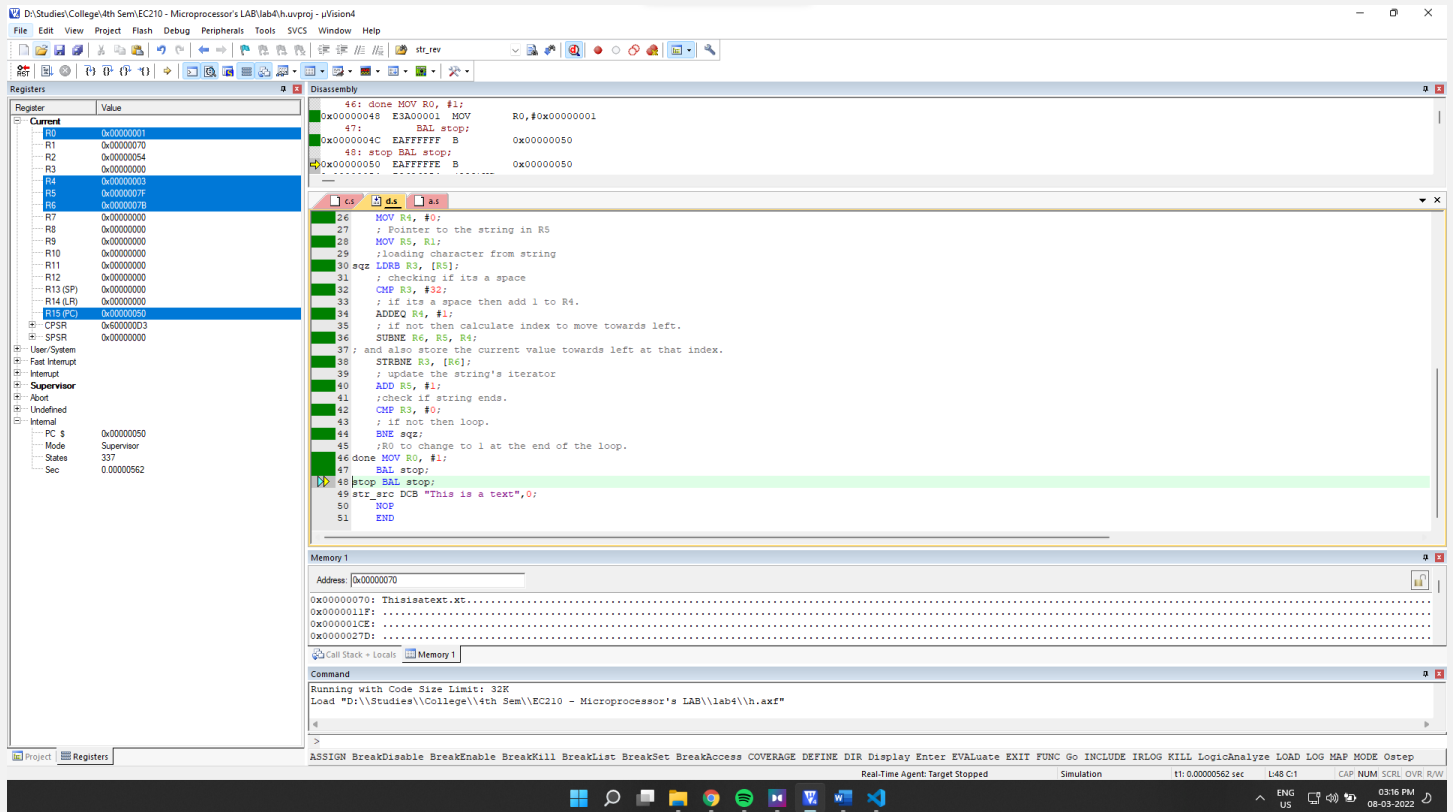
Running with Code Size Limit: 32K  
Load "D:\Studies\College\4th Sem\EC210 - Microprocessor's LAB\lab4\h.axf"

Project

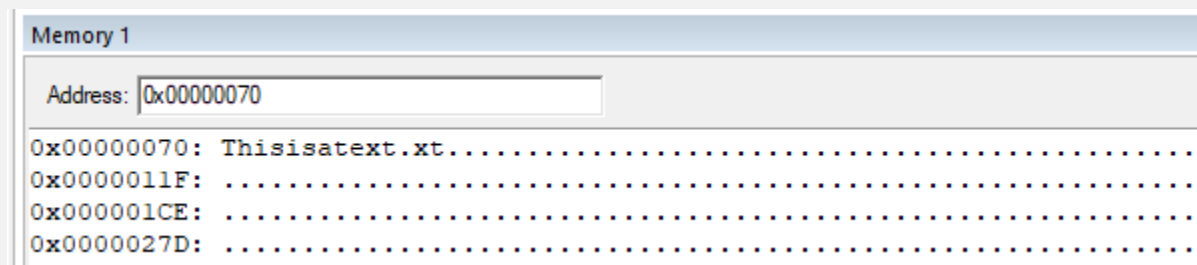
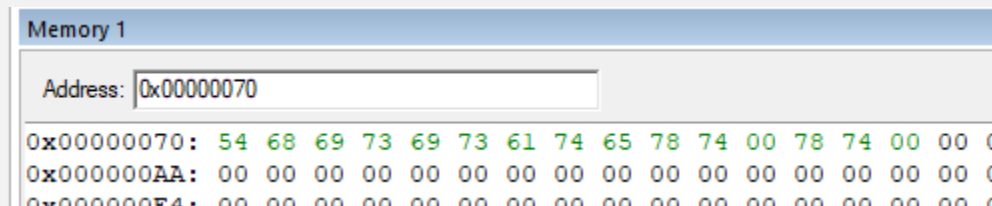
Registers

ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE DEFINE DIR Display Enter EVALuate EXIT FUNC Go INCLUDE IRLog KILL LogicAnalyze LOAD LOG MAP MODE Ostep  
Real-Time Agent: Target Stopped Simulation T1: 0.00000235 sec L26 C1 CAP. NUM. SCL. OVR. RAW  
ENG US 03:13 PM 08-03-2022

# Final Output:



# Final Memory:



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