First Semester 2022/2023

COMPUTER ORGANIZATION AND MICROPROCESSOR

ARM Assembly Project

Dr. Abualseoud Hanani

Section (2)

QOSSAY RIDA (1211553)

Contents:

Arm assembly code for this project:

```
; QOSSAY RIDA    1211553
    PRESERVE8
    THUMB
    AREA RESET, DATA, READONLY
    EXPORT __Vectors
__Vectors DCD 0x20001000 ; stack pointer value when stack is empty
          DCD Reset_Handler ; reset vector

;=====================================================================
;Declare pointer for each value in memory

TXT1Address  DCD Str1
TXT2Address  DCD Str2
TXT1AfterEditAddress  DCD TXT1AfterEdit
TXT2AfterEditAddress  DCD TXT2AfterEdit
NumberTXT1Address  DCD NumberTXT1
NumberTXT2Address  DCD NumberTXT2
TXT1AfterRemoveAddress DCD TXT1AfterRemove
COMMONAddress DCD  COMMON
ENCRYPT1Address DCD ENCRYPT1
ENCRYPT2Address DCD ENCRYPT2

;=====================================================================
;Declare the memory and what will contain

    AREA MYRAM, DATA, READONLY
Str1  DCB "Programming",0
Str2  DCB "Assembly",0

    AREA MYRAM1, DATA, READWRITE
TXT1AfterEdit  space 100
TXT2AfterEdit  space 100
NumberTXT1  DCD 0                ;Has number of letters converted for (Str1)
NumberTXT2  DCD 0                ;Has number of letters converted for (Str2)
TXT1AfterRemove  space 100       ;This Address will contain an array of unique characters that make the first string
COMMON DCD 0
ENCRYPT1 space 100
ENCRYPT2 space 100

;=====================================================================
```

```
41      AREA MYCODE, CODE, READONLY
42        ENTRY
43        EXPORT Reset_Handler
44
45  ;*****************************************
46  ; Main:
47
48  Reset_Handler
49
50        ;Call Convert Procedure to convert letters from upperCase to lowerCase for (Str1)
51        LDR R0,TXT1Address
52        MOV R2,#0
53        LDR R3 ,TXT1AfterEditAddress   ;R3 is pointer where store converted string
54        LDR R4,NumberTXT1Address       ;R4 is pointer to number of letters converted
55        BL Convert
56
57
58
59
60        ;Call Convert Procedure to convert letters from upperCase to lowerCase for (Str2)
61        LDR R0,TXT2Address
62        MOV R2,#0
63        LDR R3 ,TXT2AfterEditAddress   ;R3 is pointer where store converted string
64        LDR R4,NumberTXT2Address       ;R4 is pointer to number of letters converted
65        BL Convert
66
67        ;Call DeleteRepeated Procedure to delete all repeted character from (Str1)
68        ;And use the output to find repeted character between (the output) and (Str2) in nezt call
69        ;Example: input is "assemblyisveryeasy" the output will be "asemblyivr"
70        LDR R0, TXT1AfterEditAddress
71        LDR R1 , TXT1AfterRemoveAddress        ;R1 is pointer where store new String
72        MOV  R2, #0                            ;Is index has displacement for (R0)
73        BL DeleteRepeated
74
75        ;Call CountCOMMON Procedure to count the number of repeted letters between (Str1) and (Str2)
76        ;BY give this Procedure array of unique characters (TXT1AfterRemove) and (Str2)
77        LDR R0 , TXT1AfterRemoveAddress
78        LDR R1 , TXT2AfterEditAddress
79        MOV R2, #0
80        MOV R3, #0
81        MOV R4, R0
82        LDR R8, COMMONAddress
83        BL CountCOMMON
84
85        ;In fact, I think i deserve a full mark
86
87        ;IMPORTANT: The doctor didn't mention if the encryption was specific to the texts before or after Edit
88        ;Call Encrypt Procedure to encrypt (Str1)
89        LDR R0 , TXT1Address
90        LDR R1 , ENCRYPT1Address
91        BL Encrypt
```

```
92
93        ;Call Encrypt Procedure to encrypt (Str2)
94        LDR R0 , TXT2Address
95        LDR R1 , ENCRYPT2Address
96        BL Encrypt
97
98        ;Lode address for each value to Know the Address
99        LDR R0,TXT1Address
100       LDR R1,TXT2Address
101       LDR R2,TXT1AfterEditAddress
102       LDR R3,TXT2AfterEditAddress
103       LDR R4,NumberTXT1Address
104       LDR R5,NumberTXT2Address
105       LDR R6,TXT1AfterRemoveAddress
106       LDR R7,COMMONAddress
107       LDR R8,ENCRYPT1Address
108       LDR R9,ENCRYPT2Address
109
110   STOP
111       B STOP
112
113   ;*****************************************
114   ; Convert Procedure:
115   ; This procedure will use to convert letters from upperCase to lowerCase
116
117   Convert PROC
118       LDRB R1 ,[R0] , #1          ;Load new character from (Str1)
119       CMP R1,#0                   ;Compare character with zero
120       BGT Complete                ;If loaded character not equal 0 (null) branch to check character
121       STRB R1 , [R3]              ;If loaded character equal 0 (null) store 0 (null for end converted string)
122       STRB R2 , [R4]              ;Store the number of character we converted it
123       BX LR                       ;Get Back to main
124
125   Complete CMP R1 ,#0x5B          ;Compare character with 'Z'
126       BGT Completel               ;If loaded character grether than Z go to store it
127       CMP R1 ,#0x40               ;Compare character with 'A'
128       BLT Completel               ;If loaded character less than A go to store it
129       ADD R1,#0x20                ;If loaded character between 'A'-'Z' add 20 to it
130                                   ;because the value between upperCase and lowerCase equal 20
131       ADD R2,#1                   ;Increment the counter (this counter for number of character we converted it)
132   Completel STRB R1 , [R3] , #1   ;Store the character
133       B Convert                   ;Go to check the next character
134       ENDP
135
136   ;*****************************************
137   ; DeleteRepeated Procedure:
138   ; This procedure will use to find array of unique characters for input string
139
140   DeleteRepeated   PROC
141   loop1  ;load character from input string
```

```
142         LDRB R4, [R0, R2]       ;Load new character from (TXT1AfterEdit)
143         CMP R4, #0              ;Compare character with zero
144         BEQ endProc             ;If loaded character equal 0 (null) branch to endProc
145         CMP R4, #0x20           ;If loaded character equal 20 (space) branch to continue_input
146         BEQ continue_input
147         MOV R3, #0              ;If loaded character don't equal 0 (null) and don't equal 20 (space) lode 0 to R3
148                                 ;R3 is index has displacement for output (TXT1AfterRemove)
149 loop2
150         LDRB R5, [R1, R3]       ;Load character from output(TXT1AfterRemove)
151         CMP R5, #0              ;Compare character from output with zero
152         BEQ add_to_output       ;If loaded character from output equal 0 go to store it
153         CMP R4, R5              ;Compare character from output(R5) with loaded character from input(R4)
154         BEQ continue_input      ;If loaded character from output(R5) equal loaded character from input(R4)
155                                 ;Don't store it and go to fetch next input
156         ADD R3, #1              ;If loaded character from output(R5) don't equal loaded character from input(R4)
157                                 ;Increment (R3) to compaer the loaded character from input(R4) with all character in output
158         B loop2                 ;Continue to compare
159 add_to_output
160         STRB R4, [R1, R3]       ;Store loaded character from input(R4) to the output
161 continue_input
162         ADD R2, #1              ;Increment the displacement to fetch next character from input
163         B loop1                 ;Go to check character
164 endProc
165         BX LR                   ;Get Back to main
166         ENDP
167
168 ;*****************************************
169 ; CountCOMMON Procedure :
170 ; This procedure will compear between (Str2) and array has unique characters for (Str1)
171
172 CountCOMMON PROC
173 loop_str1
174     LDRB R5, [R0] , #1      ;Load new character from (TXT1AfterRemove)
175     CMP R5, #0              ;Compare character with zero
176     BEQ end_str1            ;If loaded character equal 0 (null) branch to end_str1
177     MOV R6, R1              ;Save address for (TXT2AfterEdit) at R6
178 loop_str2
179     LDRB R7, [R1]           ;Load new character from (TXT2AfterEdit)
180     CMP R7, #0              ;Compare character with zero
181     BEQ str2_done           ;If loaded character equal 0 (null) branch to str2_done
182     CMP R5, R7              ;Compare character from (TXT1AfterRemove) with character from (TXT2AfterEdit)
183     BEQ increment           ;If this two character are equal branch increment
184     ADD R1,#1               ;If If this two character aren't equal
185                             ;Increment R1 to fetch next character from (TXT2AfterEdit)
186     B loop_str2             ;Continue to compare
187 increment
188     ADD R2,#1               ;Increment because we find repeat character :)
189 str2_done
190     MOV R1, R6              ;Return the address for (TXT2AfterEdit) to R1
191     B loop_str1             ;Continue to compare
```

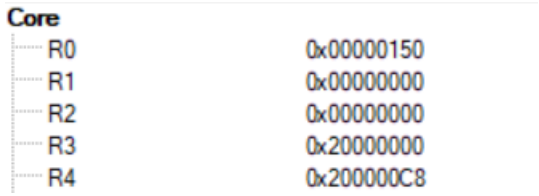
```
191     B loop_str1             ;Continue to compare
192 end_str1
193     STRB R2,[R8]            ;Store the number of repeat character
194     BX LR                   ;Get Back to main
195     ENDP
196
197 ;*****************************************
198 ; Encrypt Procedure :
199 ; This procedure will return encryption String
200
201 Encrypt PROC
202     LDRB R2 ,[R0] , #1      ;Load new character from (Str1)
203     CMP R2,#0               ;Compare character with zero
204     BEQ EndProc             ;If loaded character equal 0 (null) branch to EndProc
205     MVN R2 , R2             ;If loaded character not equal 0 (null) save NOT R2 in R2
206     STRB R2 , [R1] , #1     ;Store the character
207     B Encrypt               ;Go to check the next character
208 EndProc
209     BX LR                   ;Get Back to main
210     ENDP
211
212
213     END
214
215 ;===================================================================
```

Simulation for first operation:

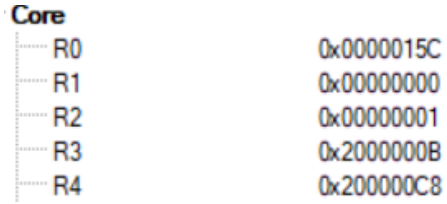
>> Code for convert (Str1)

```
;Call Convert Procedure to convert letters from upperCase to lowerCase for (Str1)
LDR R0,TXT1Address
MOV R2,#0
LDR R3 ,TXT1AfterEditAddress    ;R3 is pointer where store converted string
LDR R4,NumberTXT1Address        ;R4 is pointer to number of letters converted
BL Convert
```

>> Value of Register before (BL Convert)

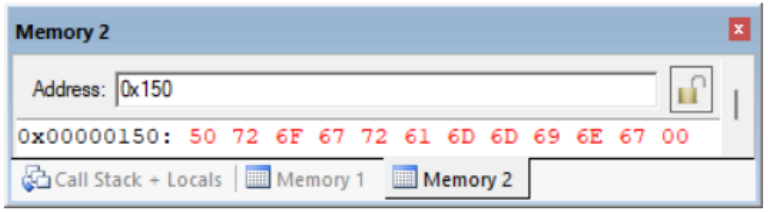


>> Value of Register after (BL Convert)

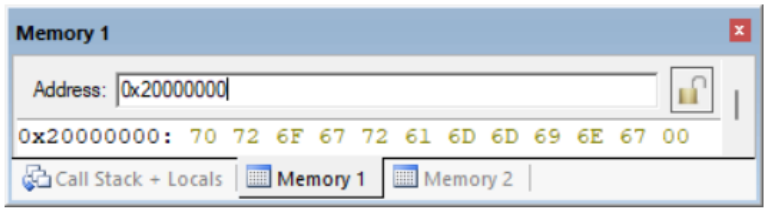


>> Value of memory after (BL Convert)

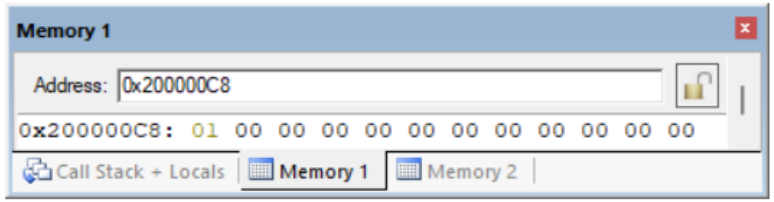
The value stored in
(TXT1Address)



The value stored in
(TXT1AfterEditAddress)



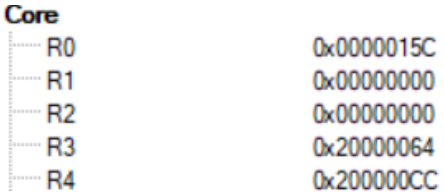
The value stored in
(NumberTXT1Address)



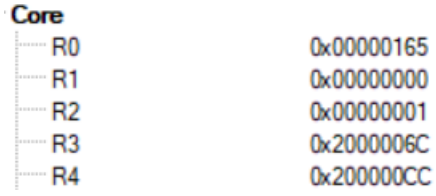
>> Code for convert (Str2)

```
;Call Convert Procedure to convert letters from upperCase to lowerCase for (Str2)
LDR R0,TXT2Address
MOV R2,#0
LDR R3 ,TXT2AfterEditAddress    ;R3 is pointer where store converted string
LDR R4,NumberTXT2Address        ;R4 is pointer to number of letters converted
BL Convert
```

>> Value of Register before (BL Convert)

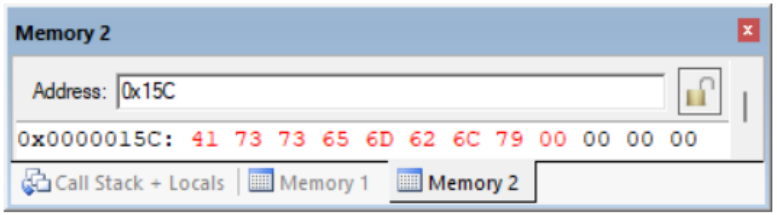


>> Value of Register after (BL Convert)

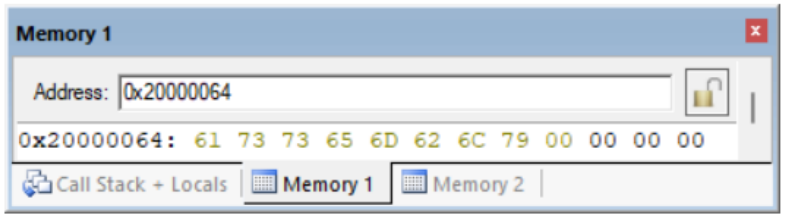


>> Value of memory after (BL Convert)

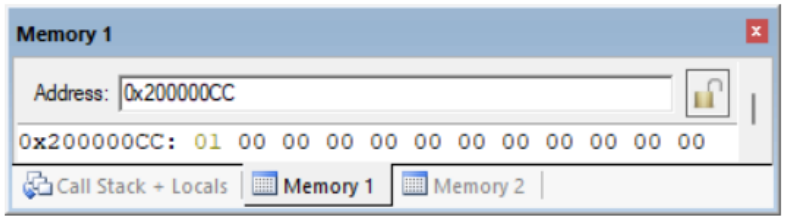
The value stored in
(TXT2Address)



The value stored in
(TXT2AfterEditAddress)



The value stored in
(NumberTXT2Address)



Simulation for second operation:

>> Explain the algorithms for the solution

We want to find the number of characters common to two different strings (Str1)(Str2), First we want to convert all letters to lowercase hence we use (TXT1AfterEdit)(TXT2AfterEdit), Then we create a procedure that creates an array containing the characters that make up the first string (TXT1AfterEdit) and the pointer for first index in this array (TXT1AfterRemove), for example if (TXT1AfterEdit) equal “programming” hence (TXT1AfterRemove) equal “progamin”, Now we compare the (TXT1AfterRemove) with (TXT2AfterEdit), Since there is a first loop that passes through the elements of the (TXT1AfterRemove) and another loop that passes through the elements of the (TXT2AfterEdit), so if an element of the (TXT1AfterRemove) matches an element of (TXT2AfterEdit), the first loop brings the next element and compares it with the elements of (TXT2AfterEdit)

(TXT1AfterEdit)= "programming" --> “progamin”

(TXT2AfterEdit)= "assembly"

“progamin”

"assembly"

COMMON =2

>> Create unique array from the first text

```
;Call DeleteRepeated Procedure to delete all repeted character from (Str1)
;And use the output to find repeted character between (the output) and (Str2) in nezt call
;Example: input is "assemblyisveryeasy" the output will be "asemblyivr"
LDR R0, TXT1AfterEditAddress
LDR R1 , TXT1AfterRemoveAddress        ;R1 is pointer where store new String
MOV  R2, #0                            ;Is index has displacement for (R0)
BL DeleteRepeated
```

>> Value of Register before (BL DeleteRepeated)

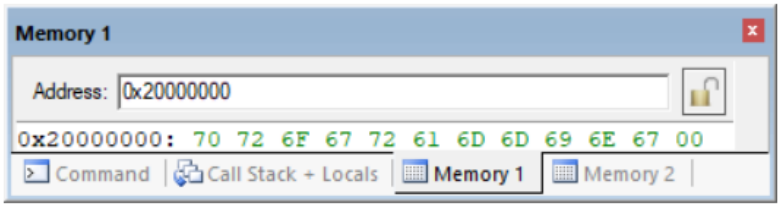
Core	
R0	0x20000000
R1	0x200000D0
R2	0x00000000

>> Value of Register after (BL DeleteRepeated)

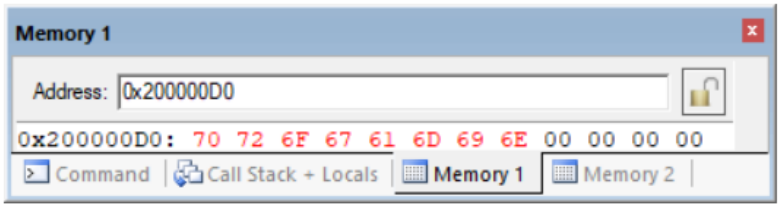
Core	
R0	0x20000000
R1	0x200000D0
R2	0x0000000B
R3	0x00000003
R4	0x00000000
R5	0x00000067

>> Value of memory after (BL DeleteRepeated)

The value stored in
(TXT1AfterEditAddress)



The value stored in
(TXT1AfterRemoveAddress)



>> Find COMMON

```
;Call CountCOMMON Procedure to count the number of repeted letters between (Str1) and (Str2)
;BY give this Procedure array of unique characters (TXT1AfterRemove) and (Str2)
LDR R0 , TXT1AfterRemoveAddress
LDR R1 , TXT2AfterEditAddress
MOV R2, #0
MOV R3, #0
MOV R4, R0
LDR R8, COMMONAddress
BL CountCOMMON
```

>> Value of Register before (BL CountCOMMON)

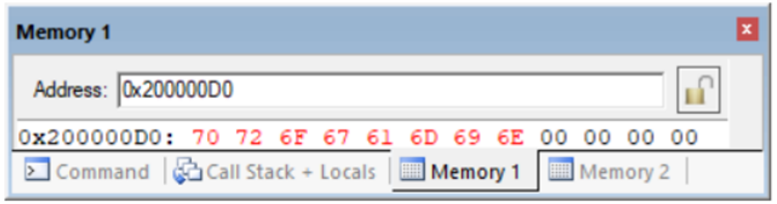
Core	
R0	0x200000D0
R1	0x20000064
R2	0x00000000
R3	0x00000000
R4	0x200000D0
R5	0x00000067
R6	0x00000000
R7	0x00000000
R8	0x20000134

>> Value of Register after (BL CountCOMMON)

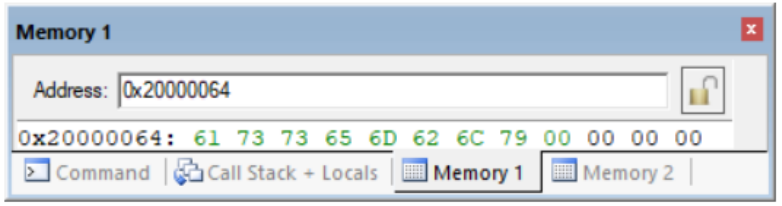
Core	
R0	0x200000D9
R1	0x20000064
R2	0x00000002
R3	0x00000000
R4	0x200000D0
R5	0x00000000
R6	0x20000064
R7	0x00000000
R8	0x20000134

>> Value of memory after (BL CountCOMMON)

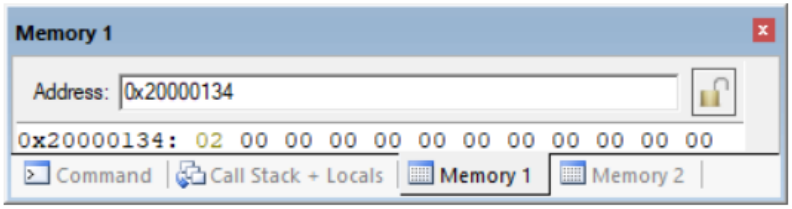
The value stored in
(TXT1AfterRemoveAddress)



The value stored in
(TXT2AfterEditAddress)



The value stored in
(COMMONAddress)



Simulation for third operation

>> Code for encryption (Str1)

```
;IMPORTANT: The doctor didn't mention if the encryption was specific to the texts before or after Edit
;Call Encrypt Procedure to encrypt (Str1)
LDR R0 , TXT1Address
LDR R1 , ENCRYPT1Address
BL Encrypt
```

>> Value of Register before (BL Encrypt)

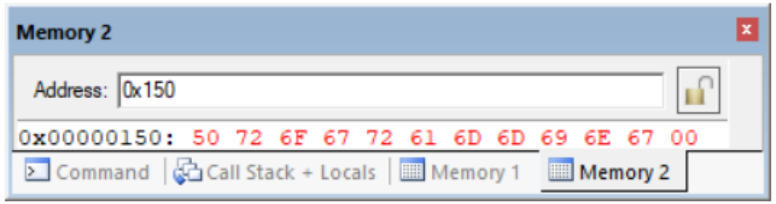
Core	
R0	0x00000150
R1	0x20000138

>> Value of Register after (BL Encrypt)

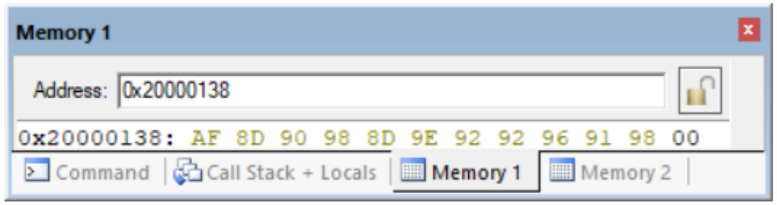
Core	
R0	0x0000015C
R1	0x20000143

>> Value of memory after (BL Encrypt)

The value stored in
(TXT1Address)



The value stored in
(ENCRYPT1Address)



>> Code for encryption (Str2)

```
;Call Encrypt Procedure to encrypt (Str2)
LDR R0 , TXT2Address
LDR R1 , ENCRYPT2Address
BL Encrypt
```

>> Value of Register before (BL Encrypt)

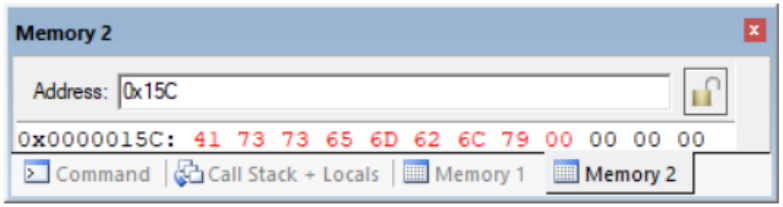
Core	
R0	0x0000015C
R1	0x2000019C

>> Value of Register after (BL Encrypt)

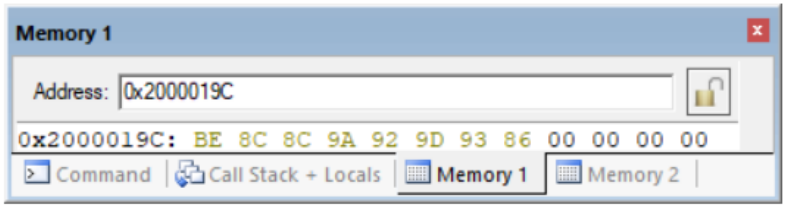
Core	
R0	0x00000165
R1	0x200001A4

>> Value of memory after (BL Encrypt)

The value stored in (TXT2Address)



The value stored in (ENCRYPT2Address)



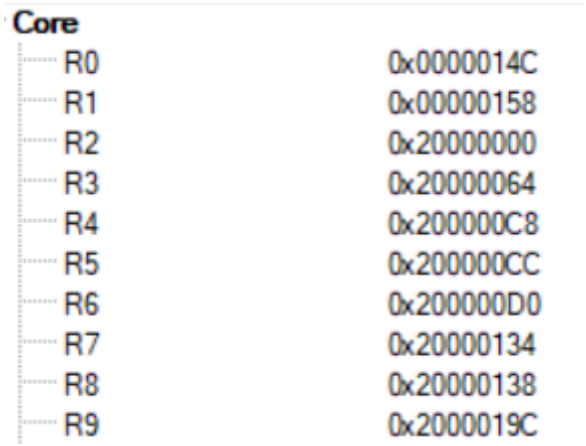
The final value for memory:

Loaded all address to registers then show the value for each address:

>> Code to load all address

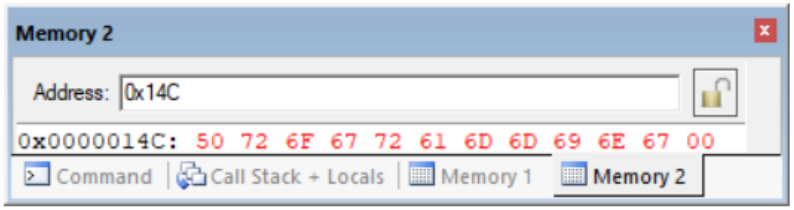
```
;Lode address for each value to Know the Address
LDR R0,TXT1Address
LDR R1,TXT2Address
LDR R2,TXT1AfterEditAddress
LDR R3,TXT2AfterEditAddress
LDR R4,NumberTXT1Address
LDR R5,NumberTXT2Address
LDR R6,TXT1AfterRemoveAddress
LDR R7,COMMONAddress
LDR R8,ENCRYPT1Address
LDR R9,ENCRYPT2Address
```

>> Value of Register after excite this instruction

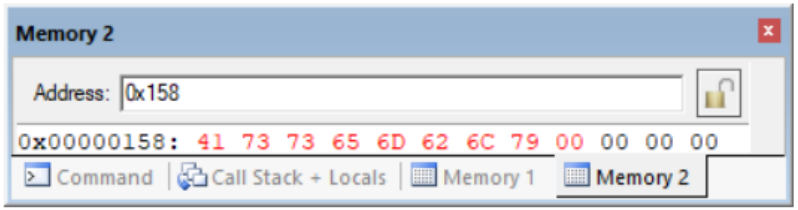


>> Value of Memory after excite this instruction

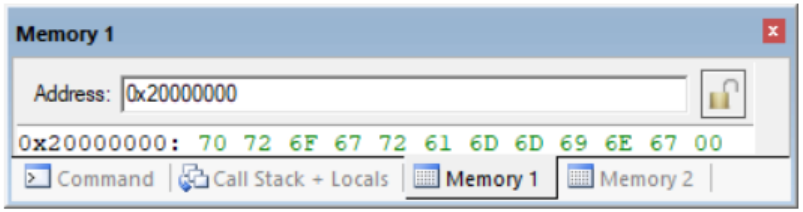
The value stored in
(TXT1Address)



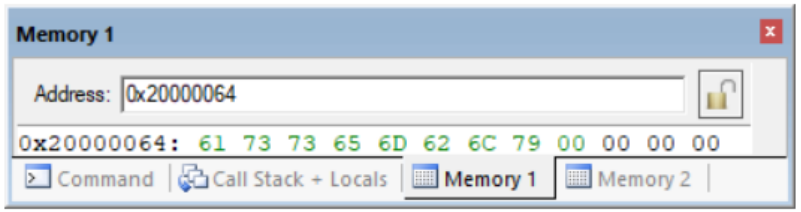
The value stored in
(TXT2Address)



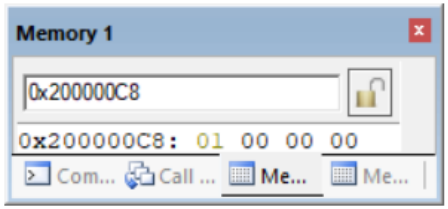
The value stored in
(TXT1AfterEditAddress)



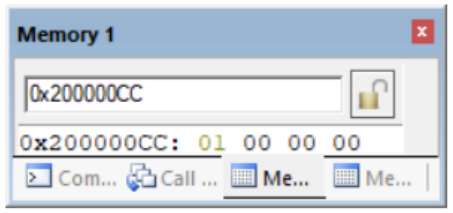
The value stored in
(TXT2AfterEditAddress)



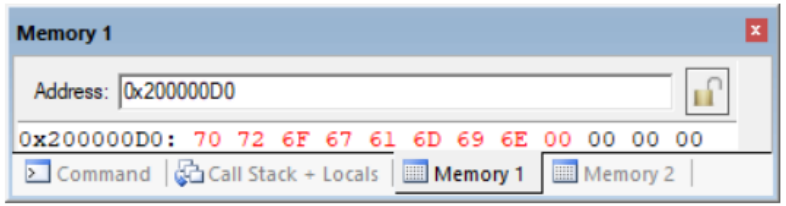
The value stored in
(NumberTXT1Address)



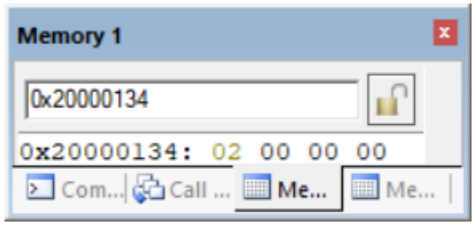
The value stored in
(NumberTXT2Address)



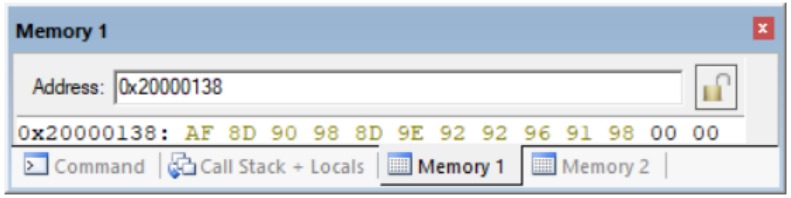
The value stored in
(TXT1AfterRemoveAddress)



The value stored in
(COMMONAddress)



The value stored in
(ENCRYPT1Address)



The value stored in
(ENCRYPT2Address)