EC-210

MICROPROCESSORS LAB

LAB-3

UTKARSH MAHAJAN 201EC164

ARNAV RAJ 201EC109

Objective: To demonstrate the use of load and Store instructions and their addressing.

Exercise:

3.2] Write an assembly program to add two numbers: C = a+ b where

(a) a and b are both unsigned 32 bit numbers

(b) a and b are both signed 32 bit numbers

(c) a and b are both unsigned 64 bit numbers

(d) a and b are both signed 64 bit numbers

Load a and b from memory and store the results including carry into new location.

->

(a) a and b are both unsigned 32 bit numbers

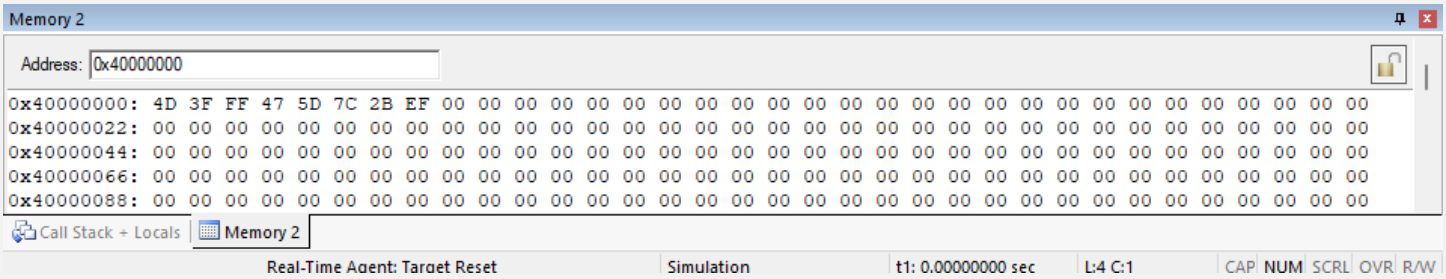
We will take values from the memory and store it in R2 and R3. Then store the Calculated result in R0 and R4. This result is later stored in memory positioned after the input value memory locations.

Source Code:

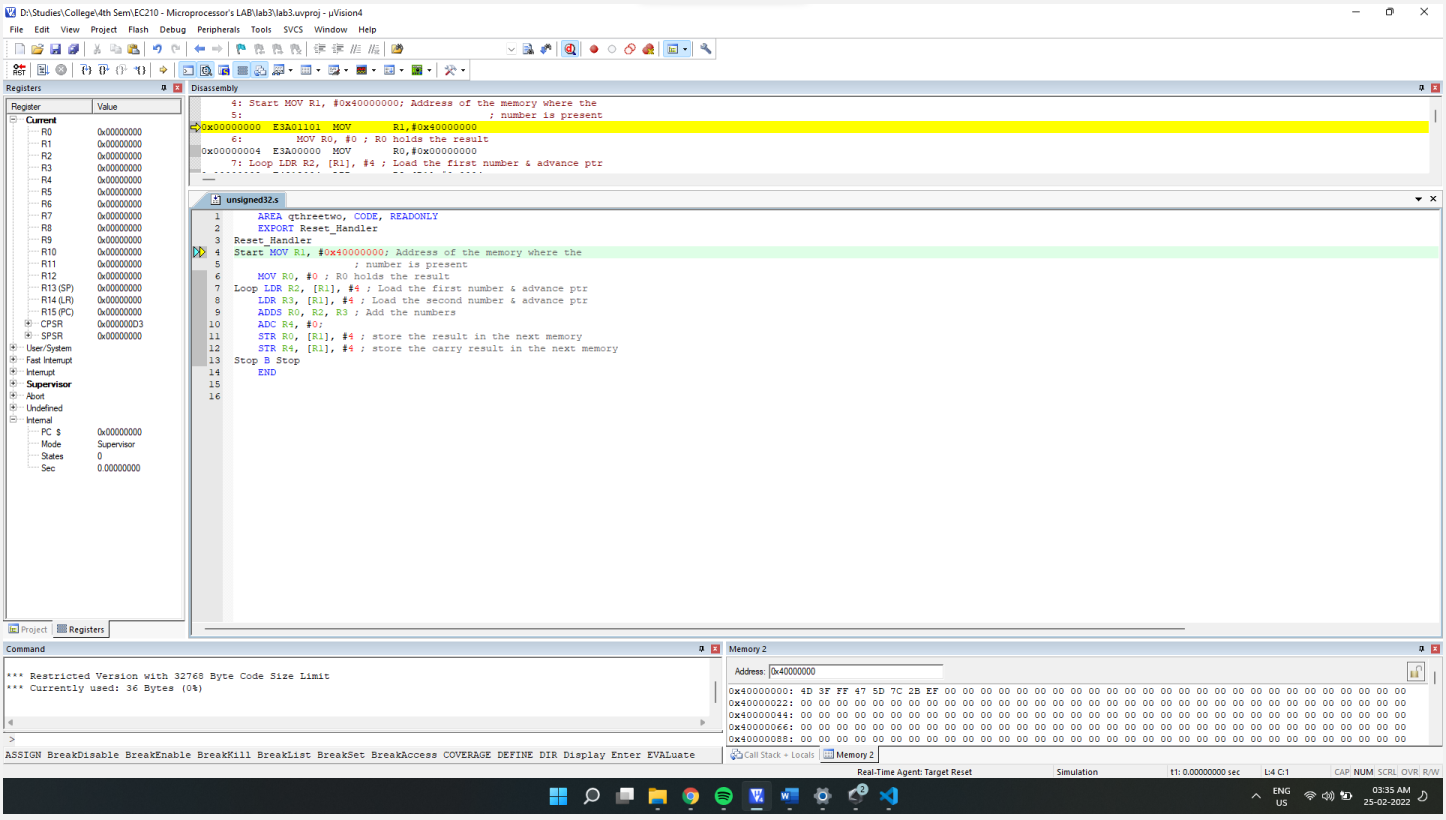
```
    AREA qthreetwo, CODE, READONLY
    EXPORT Reset_Handler
Reset_Handler
Start MOV R1, #0x40000000; Address of the memory where the
                        ; number is present
    MOV R0, #0 ; R0 holds the result
Loop LDR R2, [R1], #4 ; Load the first number & advance ptr
    LDR R3, [R1], #4 ; Load the second number & advance ptr
    ADDS R0, R2, R3 ; Add the numbers
    ADC R4, #0;
    STR R0, [R1], #4 ; store the result in the next memory
    STRB R4, [R1], #4 ; store the carry result in the next memory
Stop B Stop
    END
```

Debugging:

Initial Memory:

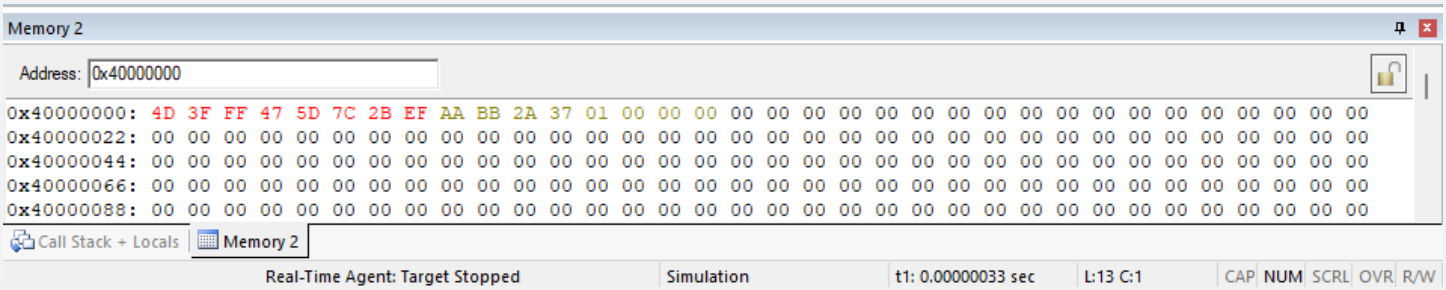


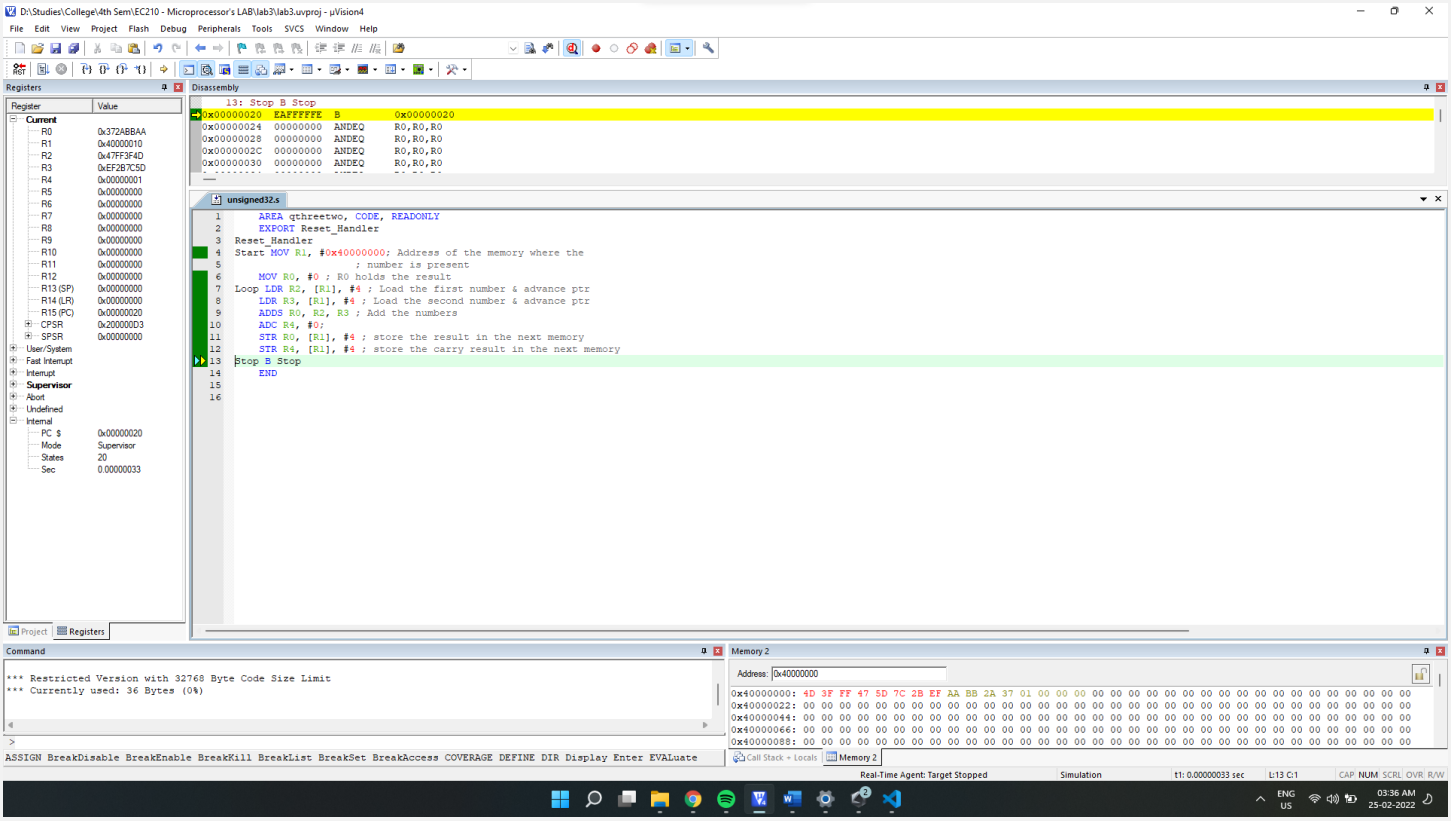
Setup:



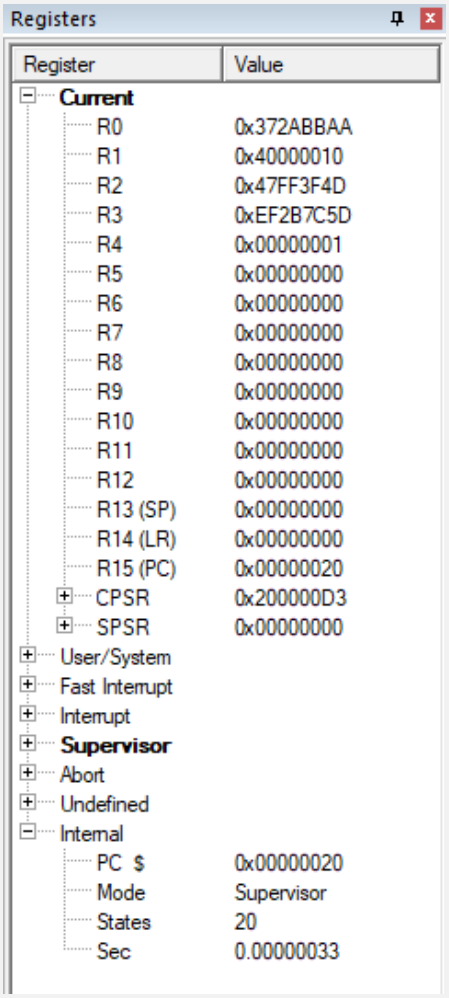
After Execution:

Final Output:





Final Register Values:



(b) a and b are both signed 32 bit numbers

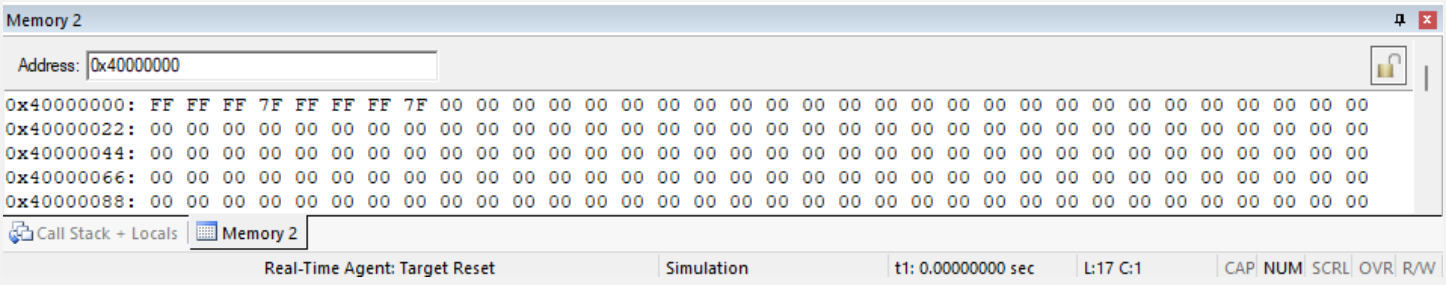
We will take values from the memory and store it in R2 and R3. While calculating addition in signed numbers, we know that we need to consider N(Negative) as the additional bit if overflow V=0 otherwise C(carry) for the **additional** bit. We will handle this by appending **conditional execution** code **VC**. Then store the Calculated result in R0 and R4. This result is later **stored in memory** locations positioned after the input value memory locations.

Source code

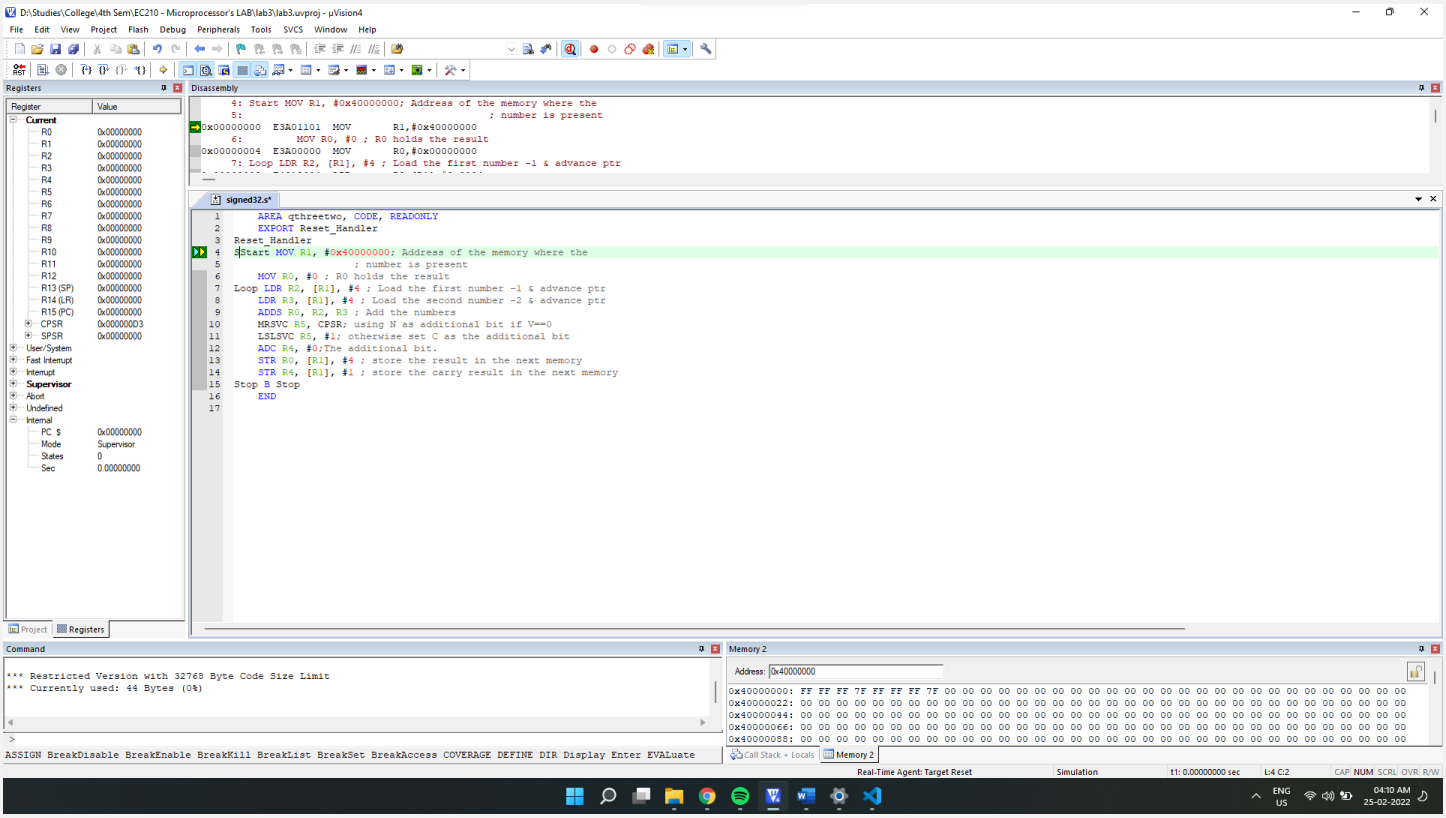
```
    AREA qthreetwo, CODE, READONLY
    EXPORT Reset_Handler
Reset_Handler
Start MOV R1, #0x40000000; Address of the memory where the
                        ; number is present
    MOV R0, #0 ; R0 holds the result
Loop LDR R2, [R1], #4 ; Load the first number -1 & advance ptr
    LDR R3, [R1], #4 ; Load the second number -2 & advance ptr
    ADDS R0, R2, R3 ; Add the numbers
    MRSVC R5, CPSR; using N as additional bit if V==0
    LSLSVC R5, #1; otherwise set C as the additional bit
    ADC R4, #0;The additional bit.
    STR R0, [R1], #4 ; store the result in the next memory
    STR R4, [R1], #1 ; store the carry result in the next memory
Stop B Stop
    END
```

Debugging:

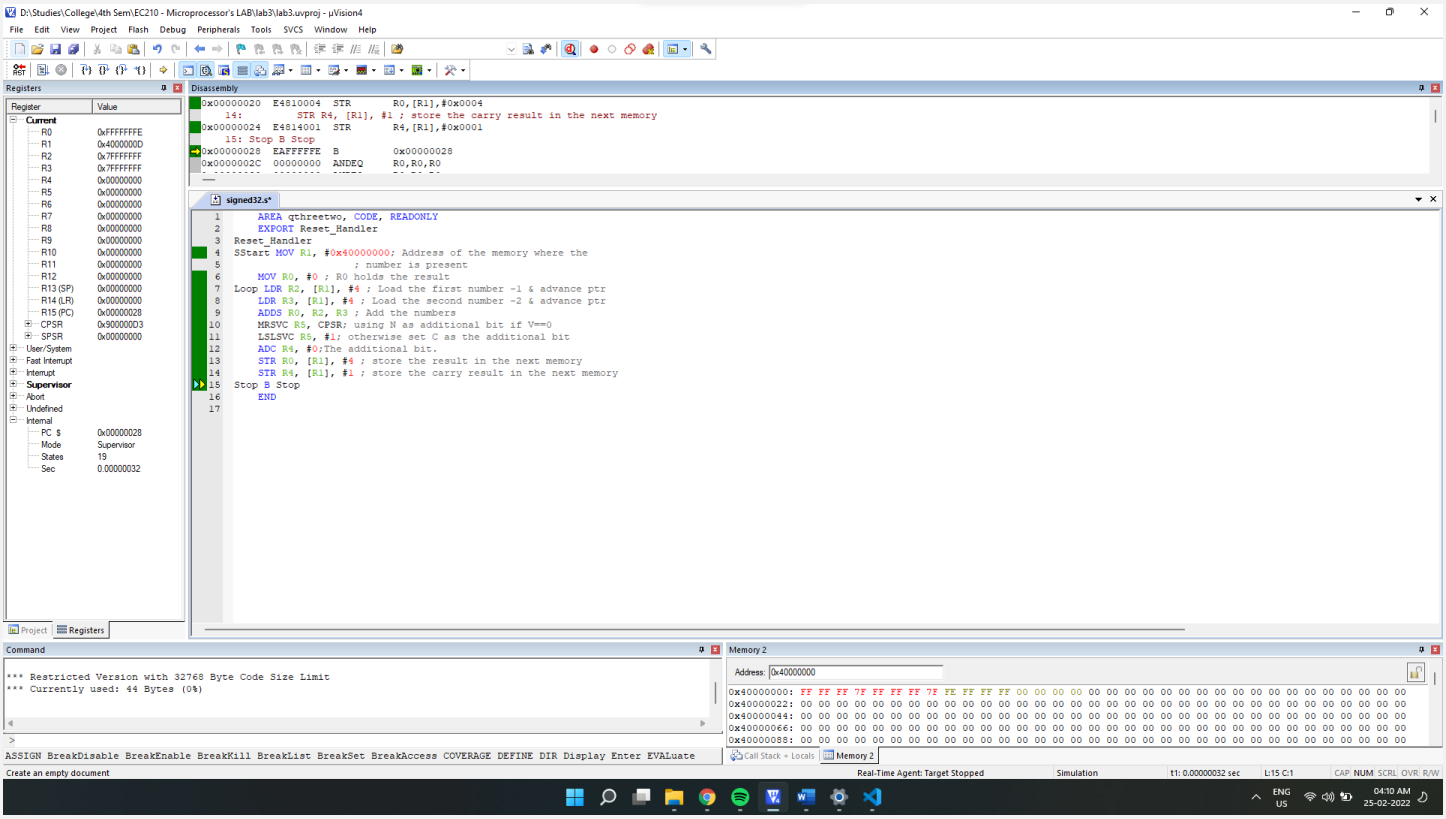
Initial Memory:



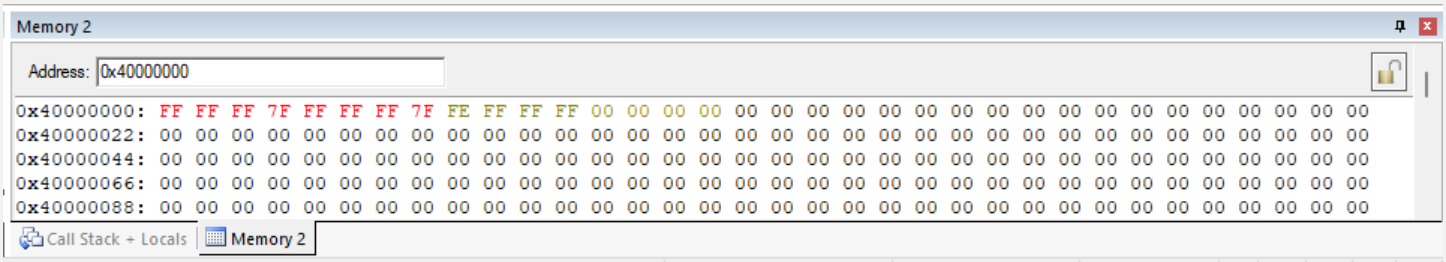
Setup:



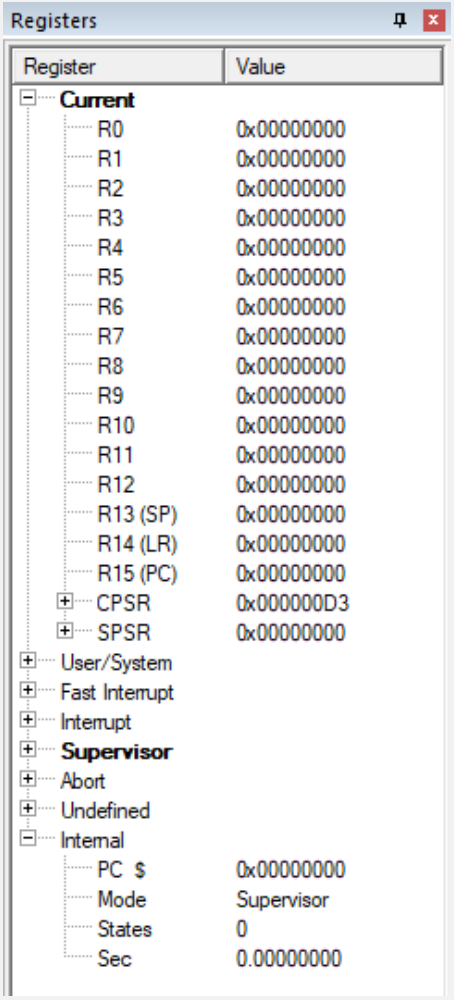
After Execution:



Final memory Output:



Final Register Values:



(c) a and b are both unsigned 64 bit numbers

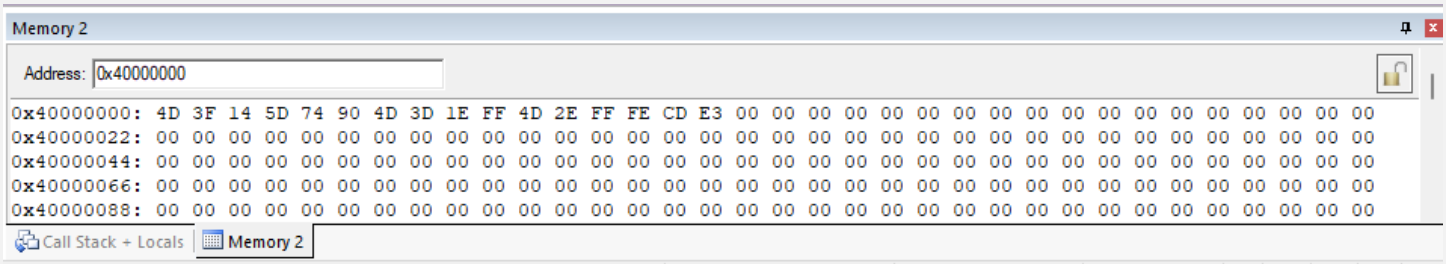
We will take values from the memory and store it in R2, R3, R4 and R3. Then store the Calculated result in R6, R7 and R8. This result is later stored in memory positioned after the input value memory locations.

Source code:

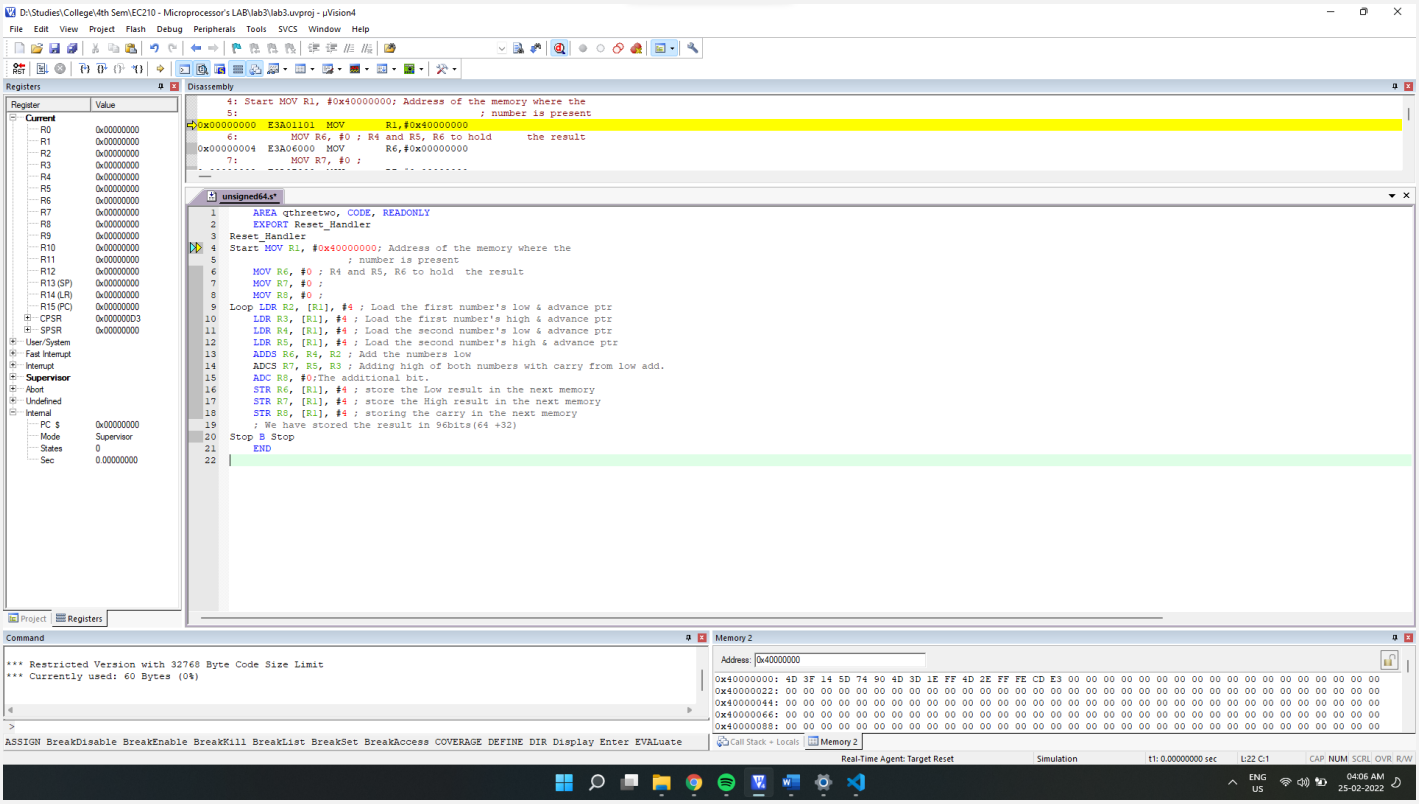
```
    AREA qthreetwo, CODE, READONLY
    EXPORT Reset_Handler
Reset_Handler
Start MOV R1, #0x40000000; Address of the memory where the
                        ; number is present
    MOV R6, #0 ; R4 and R5, R6 to hold  the result
    MOV R7, #0 ;
    MOV R8, #0 ;
Loop LDR R2, [R1], #4 ; Load the first number's low & advance ptr
    LDR R3, [R1], #4 ; Load the first number's high & advance ptr
    LDR R4, [R1], #4 ; Load the second number's low & advance ptr
    LDR R5, [R1], #4 ; Load the second number's high & advance ptr
    ADDS R6, R4, R2 ; Add the numbers low
    ADCS R7, R5, R3 ; Adding high of both numbers with carry from low add.
    ADC R8, #0;The additional bit.
    STR R6, [R1], #4 ; store the Low result in the next memory
    STR R7, [R1], #4 ; store the High result in the next memory
    STR R8, [R1], #4 ; storing the carry in the next memory
    ; We have stored the result in 96bits(64 +32)
Stop B Stop
    END
```

Debugging:

Initial Memory:

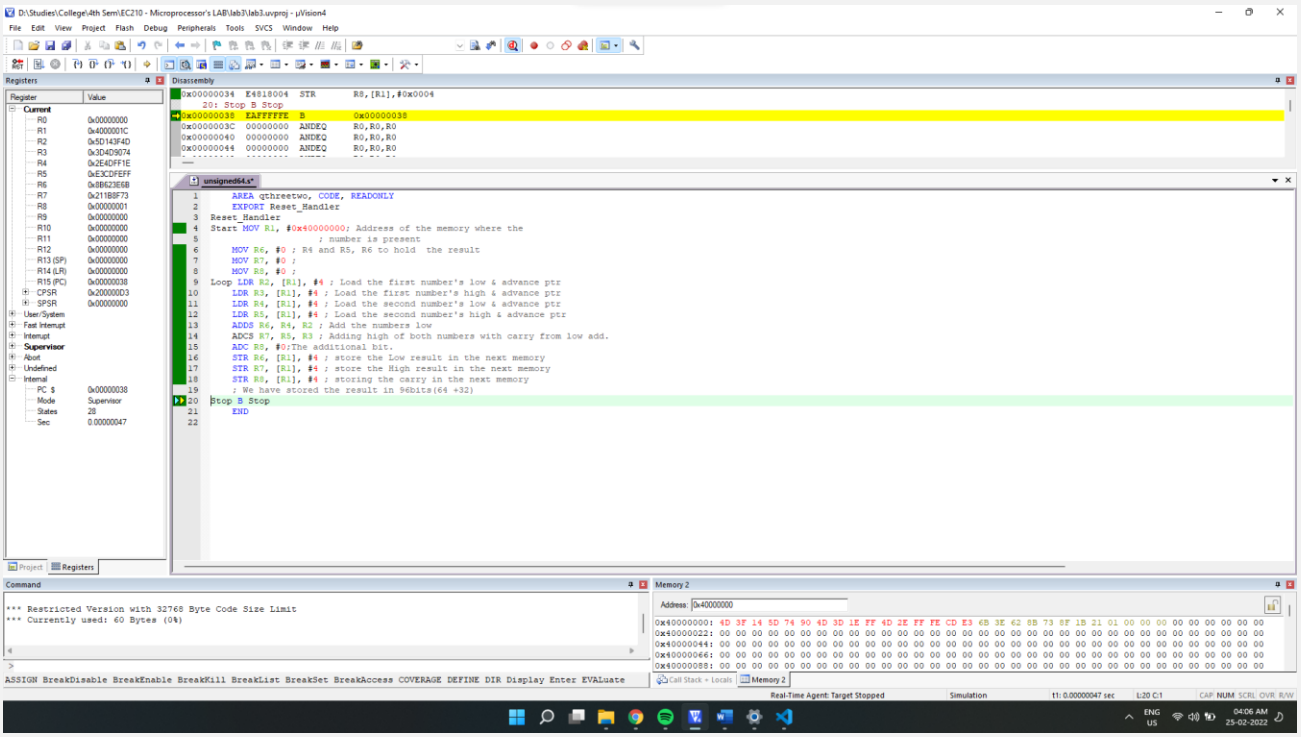


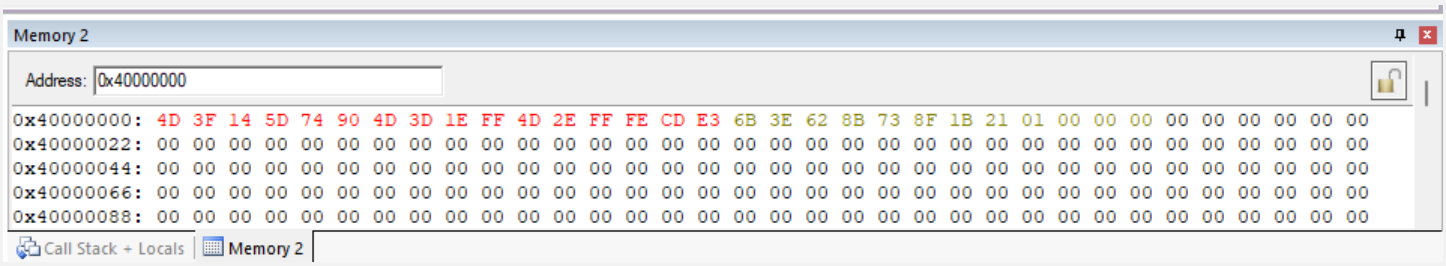
Setup:



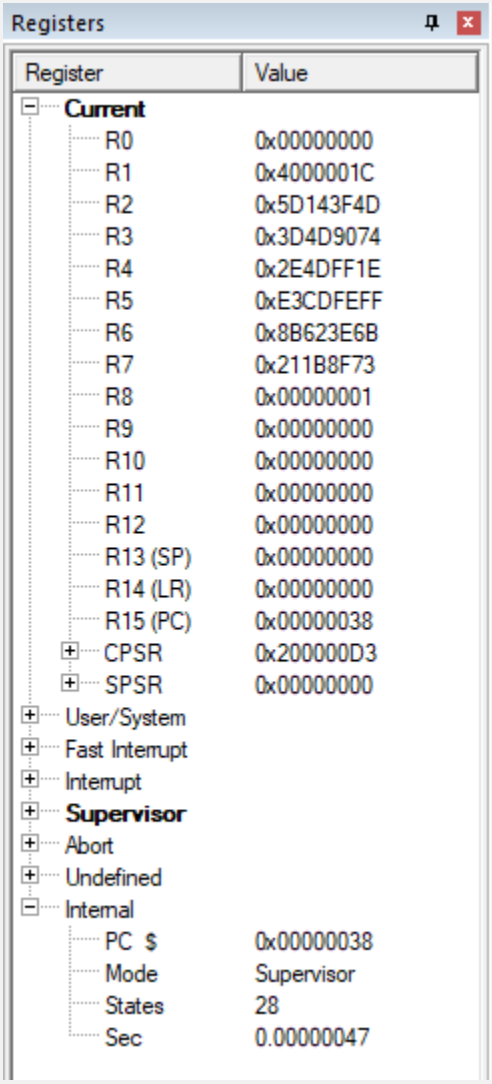
After Execution:

Final Output:





Final Register Values:



(d) a and b are both signed 64 bit numbers

We will take values from the memory and store it in R2 and R3. While calculating signed numbers, we know that we need to consider N(Negative) as the additional bit if overflow V=0 otherwise C(carry) for the **additional** bit. We will handle this by appending **conditional execution** code **VC**. Then store the Calculated result in R6, R7 and R8. This result is later **stored in memory**

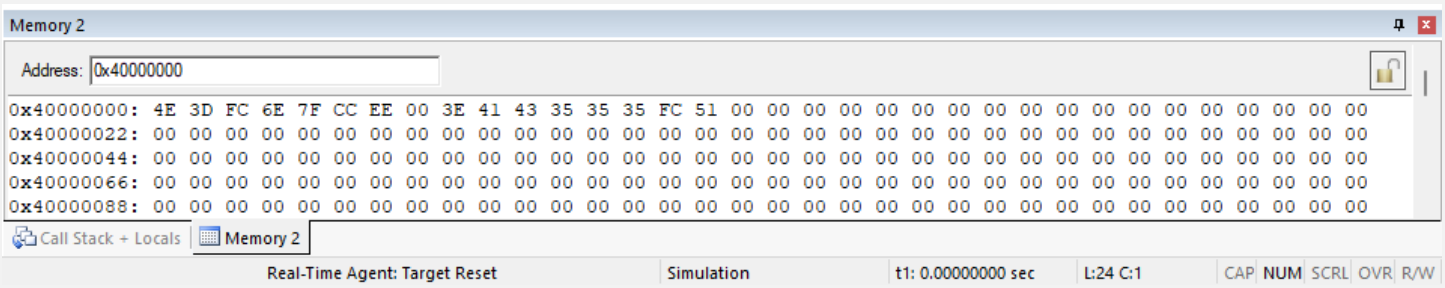
locations positioned after the input value memory locations. **65th bit will be sign**

Source code

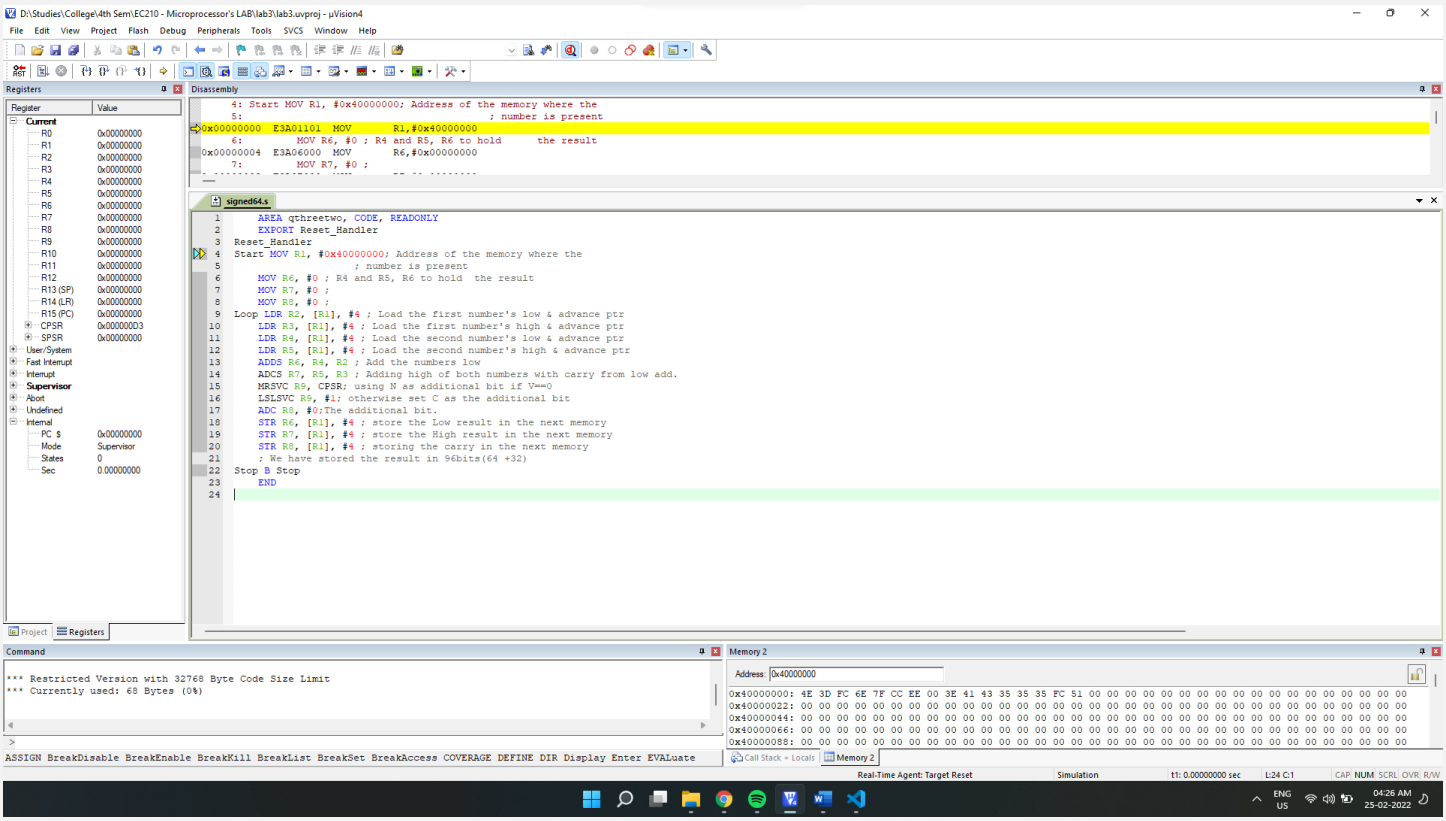
```
    AREA qthreetwo, CODE, READONLY
    EXPORT Reset_Handler
Reset_Handler
Start MOV R1, #0x40000000; Address of the memory where the
                        ; number is present
    MOV R6, #0 ; R4 and R5, R6 to hold  the result
    MOV R7, #0 ;
    MOV R8, #0 ;
Loop LDR R2, [R1], #4 ; Load the first number's low & advance ptr
    LDR R3, [R1], #4 ; Load the first number's high & advance ptr
    LDR R4, [R1], #4 ; Load the second number's low & advance ptr
    LDR R5, [R1], #4 ; Load the second number's high & advance ptr
    ADDS R6, R4, R2 ; Add the numbers low
    ADCS R7, R5, R3 ; Adding high of both numbers with carry from low add.
    MRSVC R9, CPSR; using N as additional bit if V==0
    LSLSVC R9, #1; otherwise set C as the additional bit
    ADC R8, #0;The additional bit.
    STR R6, [R1], #4 ; store the Low result in the next memory
    STR R7, [R1], #4 ; store the High result in the next memory
    STR R8, [R1], #4 ; storing the carry in the next memory
    ; We have stored the 65 bit result in 96bits(64 +32).
; 65th bit will be sign.
Stop B Stop
    END
```

Debugging:

Initial Memory:

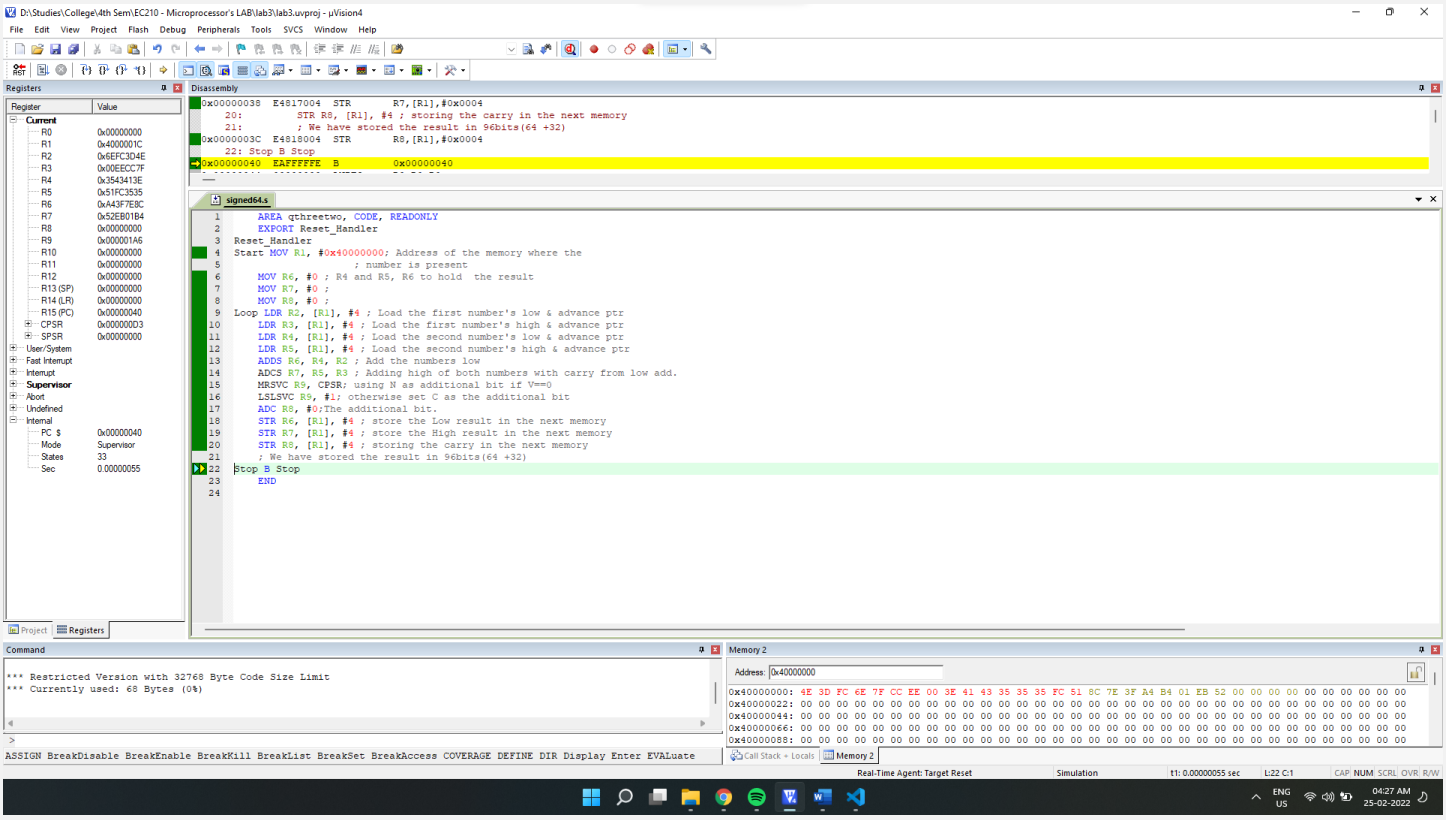


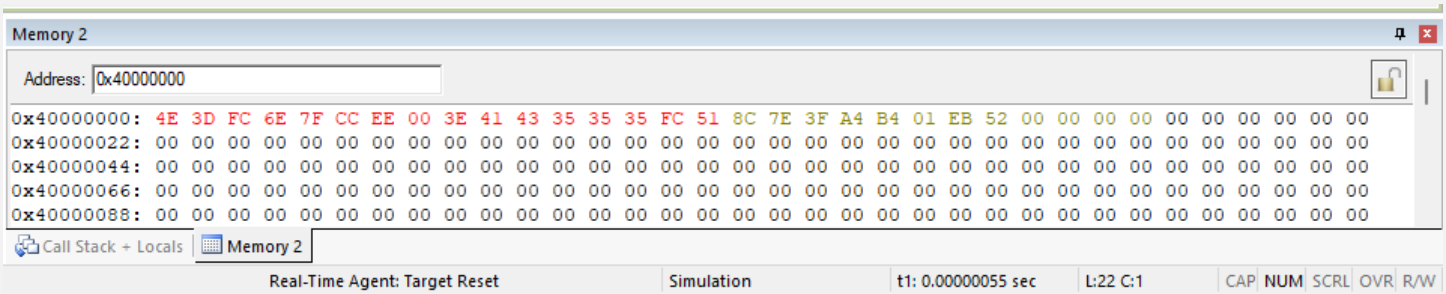
Setup:



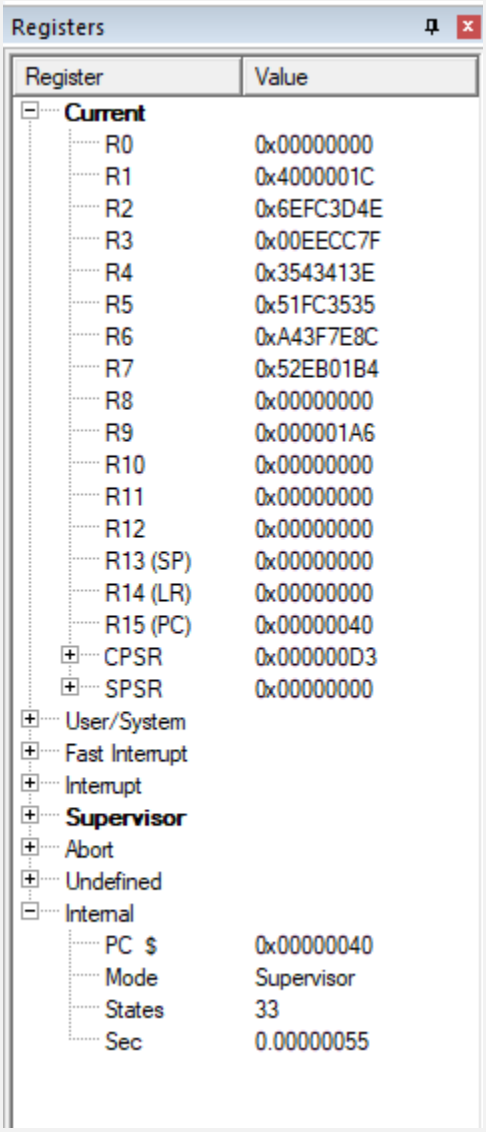
After Execution:

Final Output:





Final Register Values:



3.3] Write an assembly program to perform C = a – b where

(a) a and b are both unsigned 32 bit numbers

(b) a and b are both signed 32 bit numbers

(c) a and b are both unsigned 64 bit numbers

(d) a and b are both signed 64 bit numbers

->

(a) a and b are both unsigned 32 bit numbers

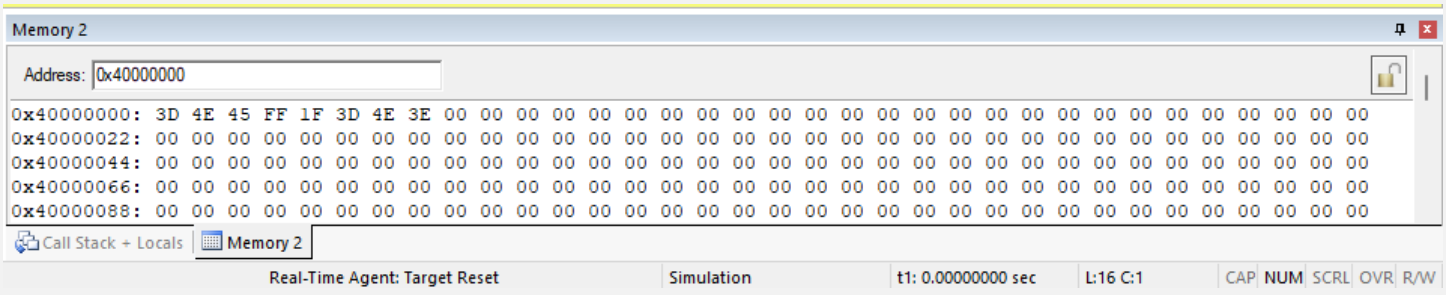
We will take values from the memory and store it in R2 and R3. Then store the Calculated result in R0 and R4. This result is later stored in memory positioned after the input value memory locations.

Source code:

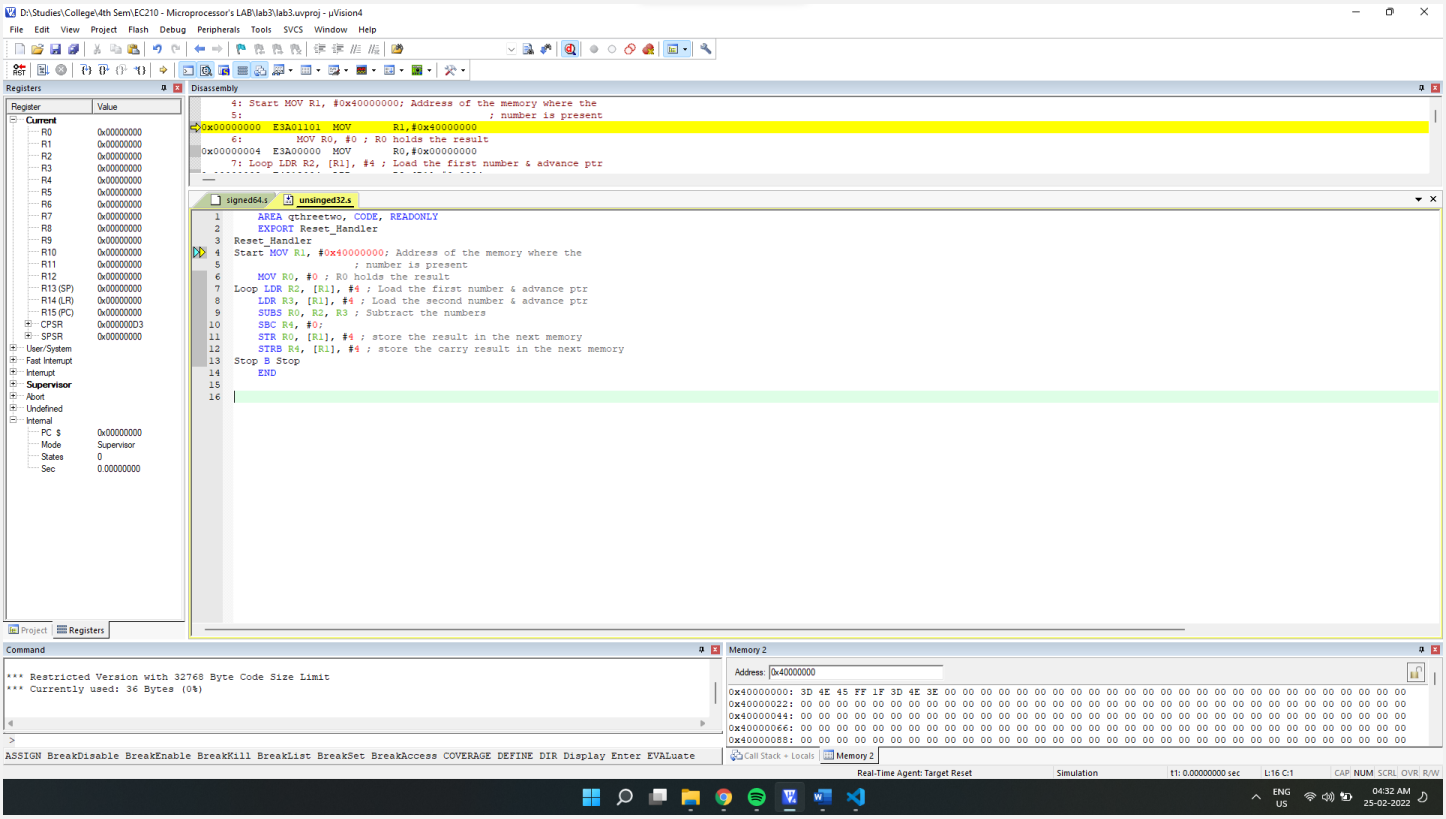
```
    AREA qthreetwo, CODE, READONLY
    EXPORT Reset_Handler
Reset_Handler
Start MOV R1, #0x40000000; Address of the memory where the
                        ; number is present
    MOV R0, #0 ; R0 holds the result
Loop LDR R2, [R1], #4 ; Load the first number & advance ptr
    LDR R3, [R1], #4 ; Load the second number & advance ptr
    SUBS R0, R2, R3 ; Subtract the numbers
    SBC R4, #0;
    STR R0, [R1], #4 ; store the result in the next memory
    STRB R4, [R1], #4 ; store the carry result in the next memory
Stop B Stop
    END
```

<u>Debugging:</u>

Initial Memory:

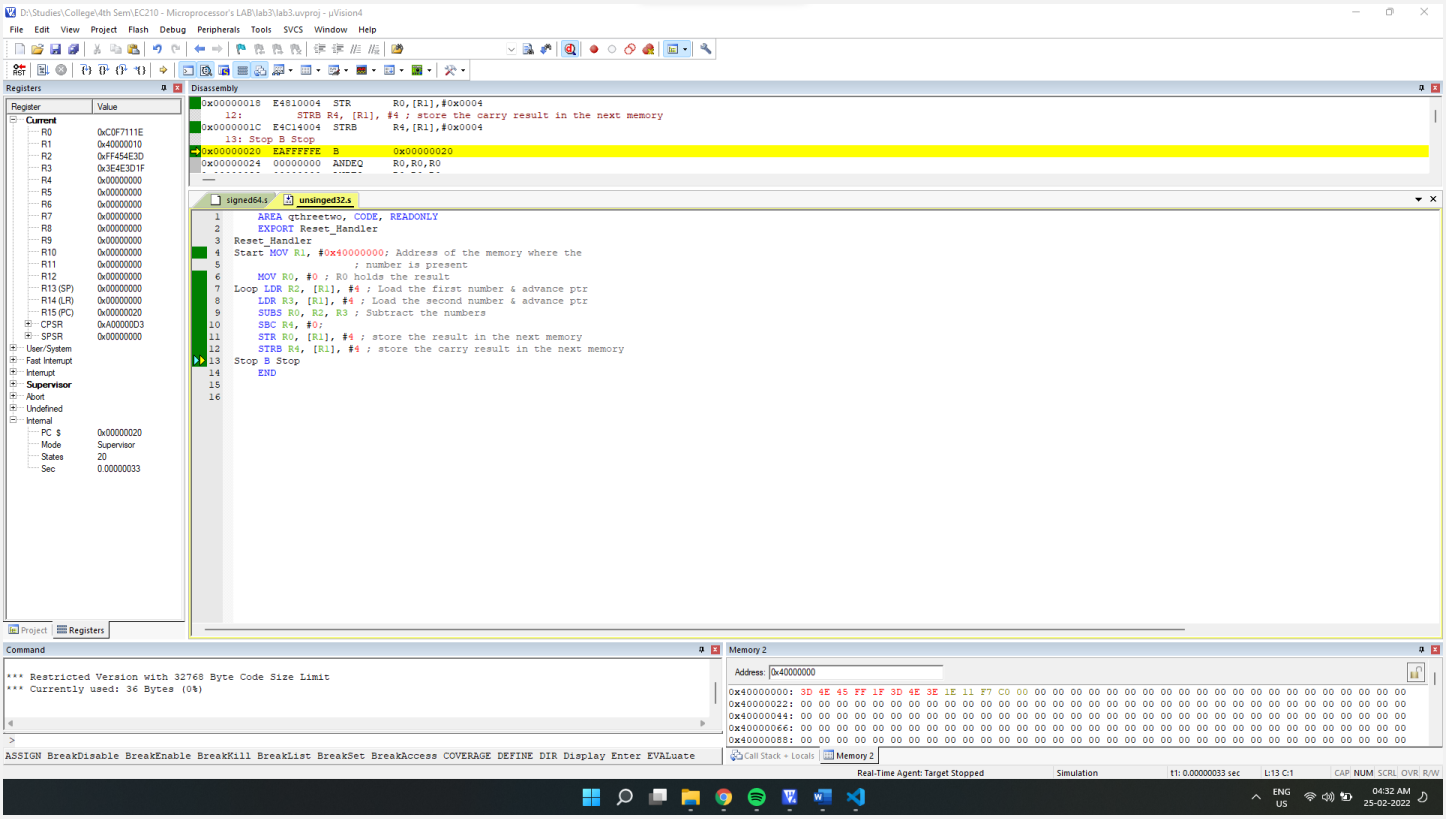


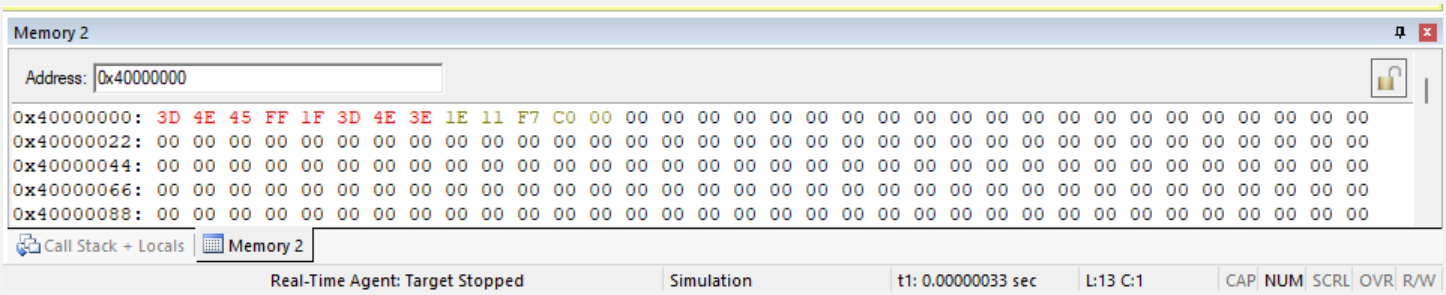
Setup:



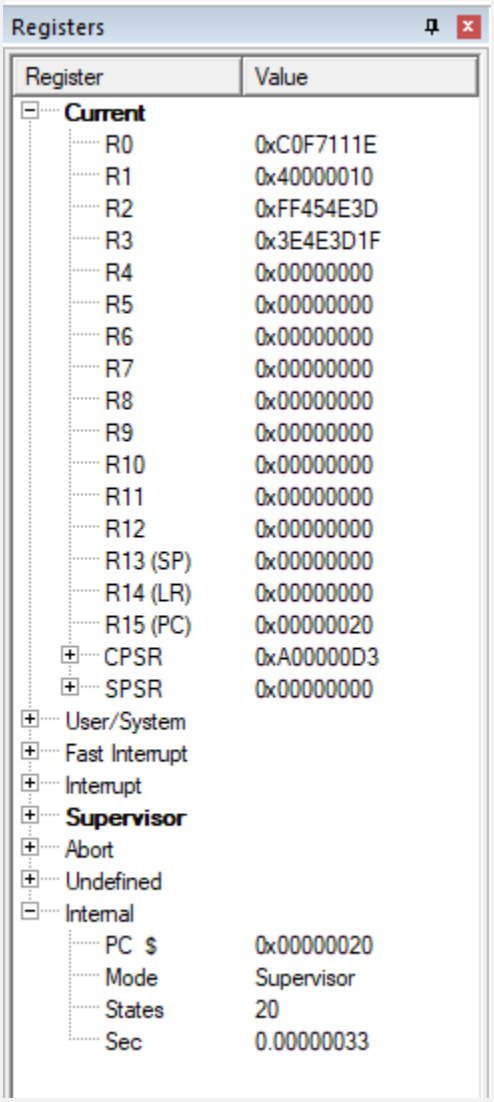
After Execution:

Final Output:





Final Register Values:



(b) a and b are both signed 32 bit numbers

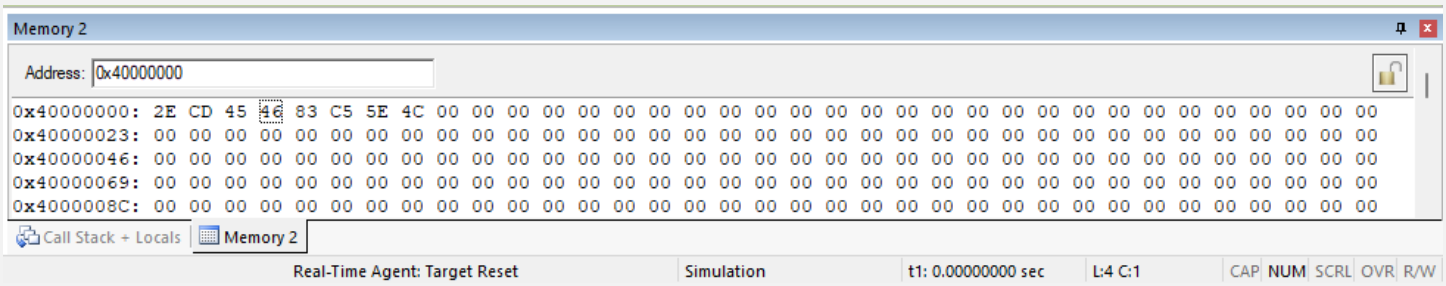
We will take values from the memory and store it in R2 and R3. While calculating subtraction in signed numbers, we know that we need to consider N(Negative) as the additional bit if overflow V=0 otherwise C(carry) for the **additional** bit. We will handle this by appending **conditional execution** code **VC**. Then store the Calculated result in R0 and R4. This result is later **stored in memory** locations positioned after the input value memory locations.

Source code:

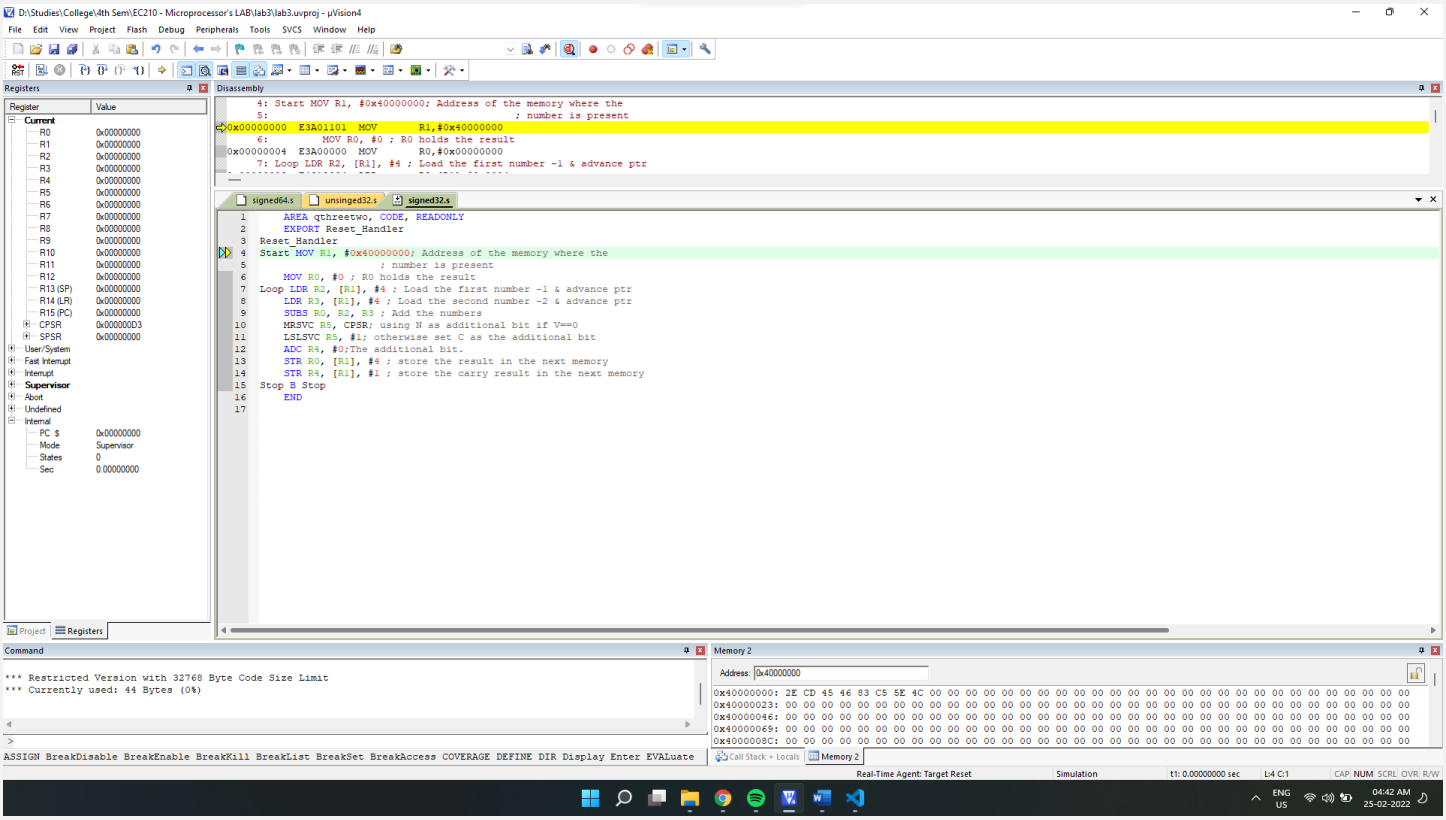
```
    AREA qthreetwo, CODE, READONLY
    EXPORT Reset_Handler
Reset_Handler
Start MOV R1, #0x40000000; Address of the memory where the
                         ; number is present
    MOV R0, #0 ; R0 holds the result
Loop LDR R2, [R1], #4 ; Load the first number -1 & advance ptr
    LDR R3, [R1], #4 ; Load the second number -2 & advance ptr
    SUBS R0, R2, R3 ; Add the numbers
    MRSVC R5, CPSR; using N as additional bit if V==0
    LSLSVC R5, #1; otherwise set C as the additional bit
    ADC R4, #0;The additional bit.
    STR R0, [R1], #4 ; store the result in the next memory
    STR R4, [R1], #1 ; store the carry result in the next memory
Stop B Stop
    END
```

Debugging:

Initial Memory:

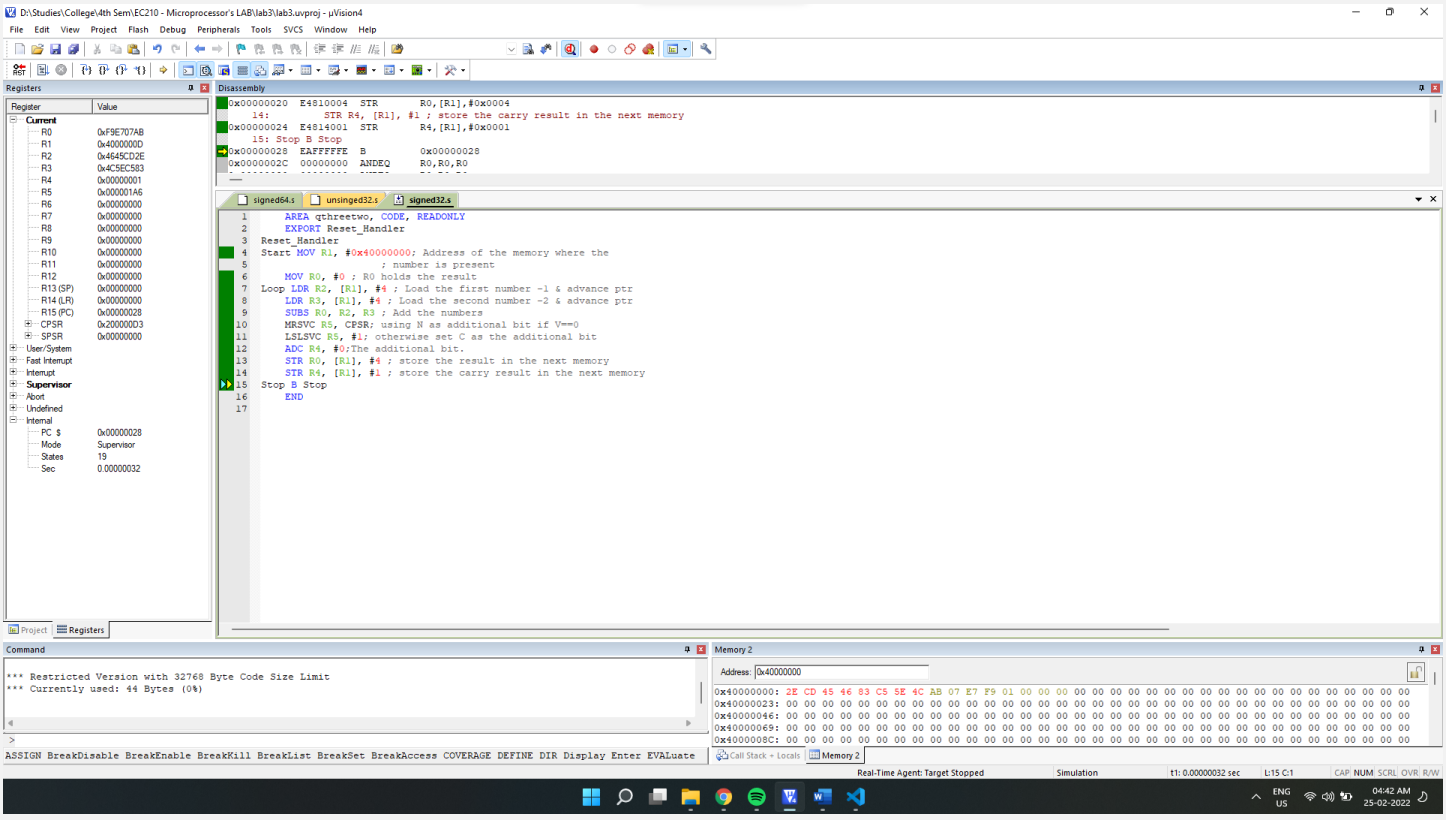


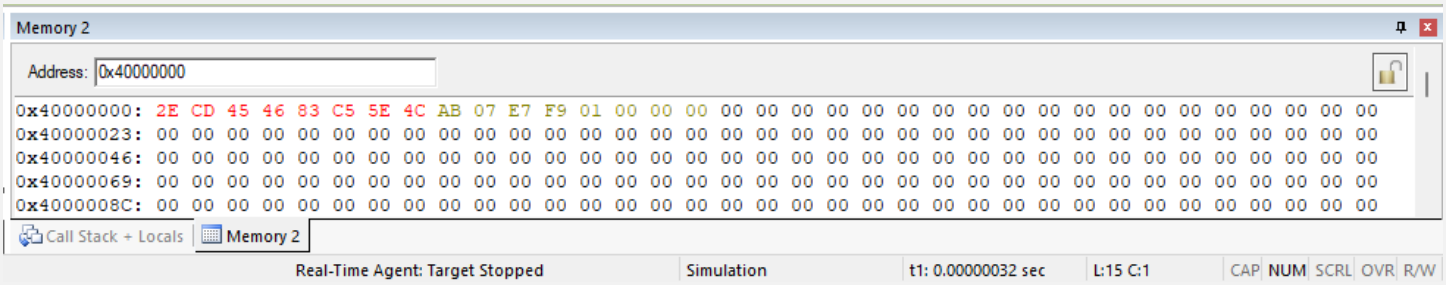
Setup:



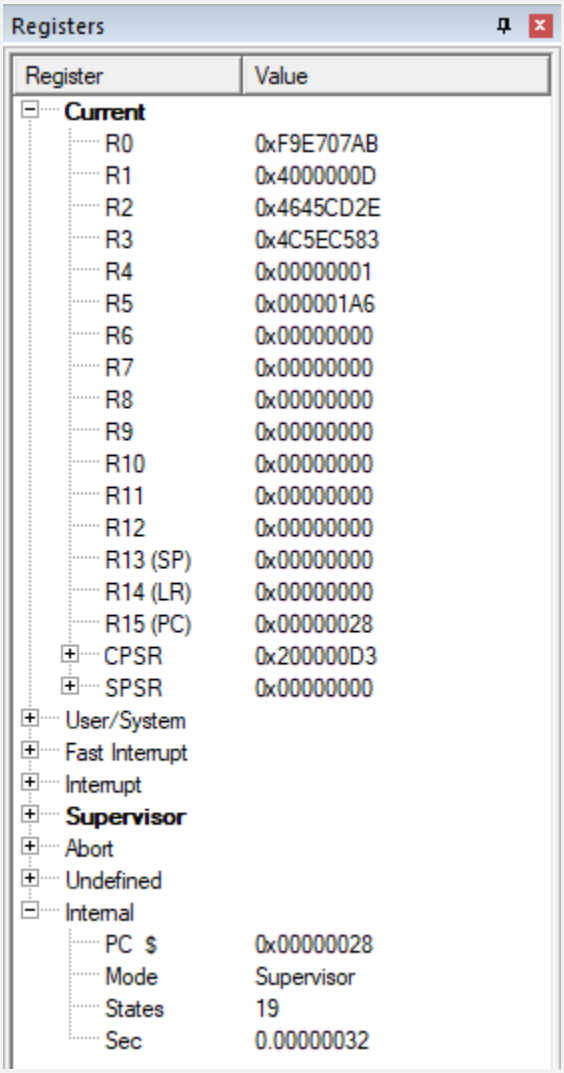
After Execution:

Final Output:





Final Register Values:



(c) a and b are both unsigned 64 bit numbers

We will take values from the memory and store it in R2 and R3. While calculating subtraction of signed numbers, we know that we need to consider N(Negative) as the additional bit if overflow V=0 otherwise C(carry) for the **additional** bit. We will handle this by appending **conditional execution** code **VC**. Then store the Calculated result in R6, R7 and R8. This result is later

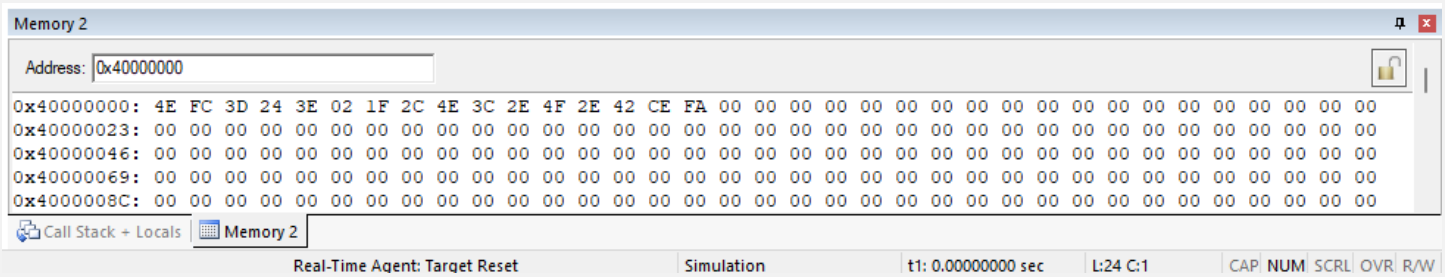
stored in memory locations positioned after the input value memory locations. **65th bit will be sign**

Source code:

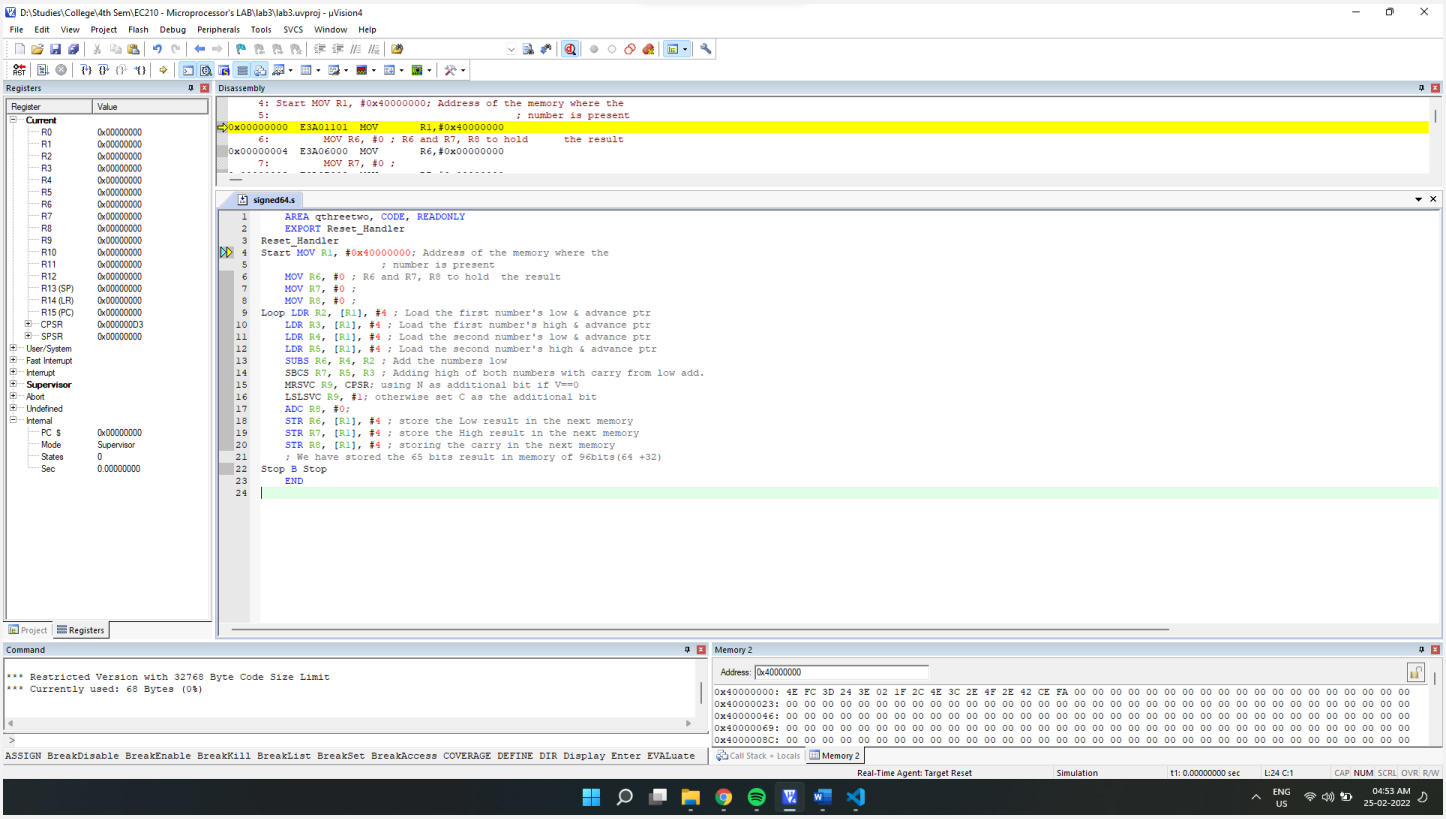
```
    AREA qthreetwo, CODE, READONLY
    EXPORT Reset_Handler
Reset_Handler
Start MOV R1, #0x40000000; Address of the memory where the
                        ; number is present
    MOV R6, #0 ; R6 and R7, R8 to hold  the result
    MOV R7, #0 ;
    MOV R8, #0 ;
Loop LDR R2, [R1], #4 ; Load the first number's low & advance ptr
    LDR R3, [R1], #4 ; Load the first number's high & advance ptr
    LDR R4, [R1], #4 ; Load the second number's low & advance ptr
    LDR R5, [R1], #4 ; Load the second number's high & advance ptr
    SUBS R6, R4, R2 ; Add the numbers low
    SBCS R7, R5, R3 ; Adding high of both numbers with carry from low add.
    MRSVC R9, CPSR; using N as additional bit if V==0
    LSLSVC R9, #1; otherwise set C as the additional bit
    ADC R8, #0;
    STR R6, [R1], #4 ; store the Low result in the next memory
    STR R7, [R1], #4 ; store the High result in the next memory
    STR R8, [R1], #4 ; storing the carry in the next memory
    ; We have stored the 65 bits result in memory of 96bits(64 +32)
Stop B Stop
    END
```

Debugging:

Initial Memory:

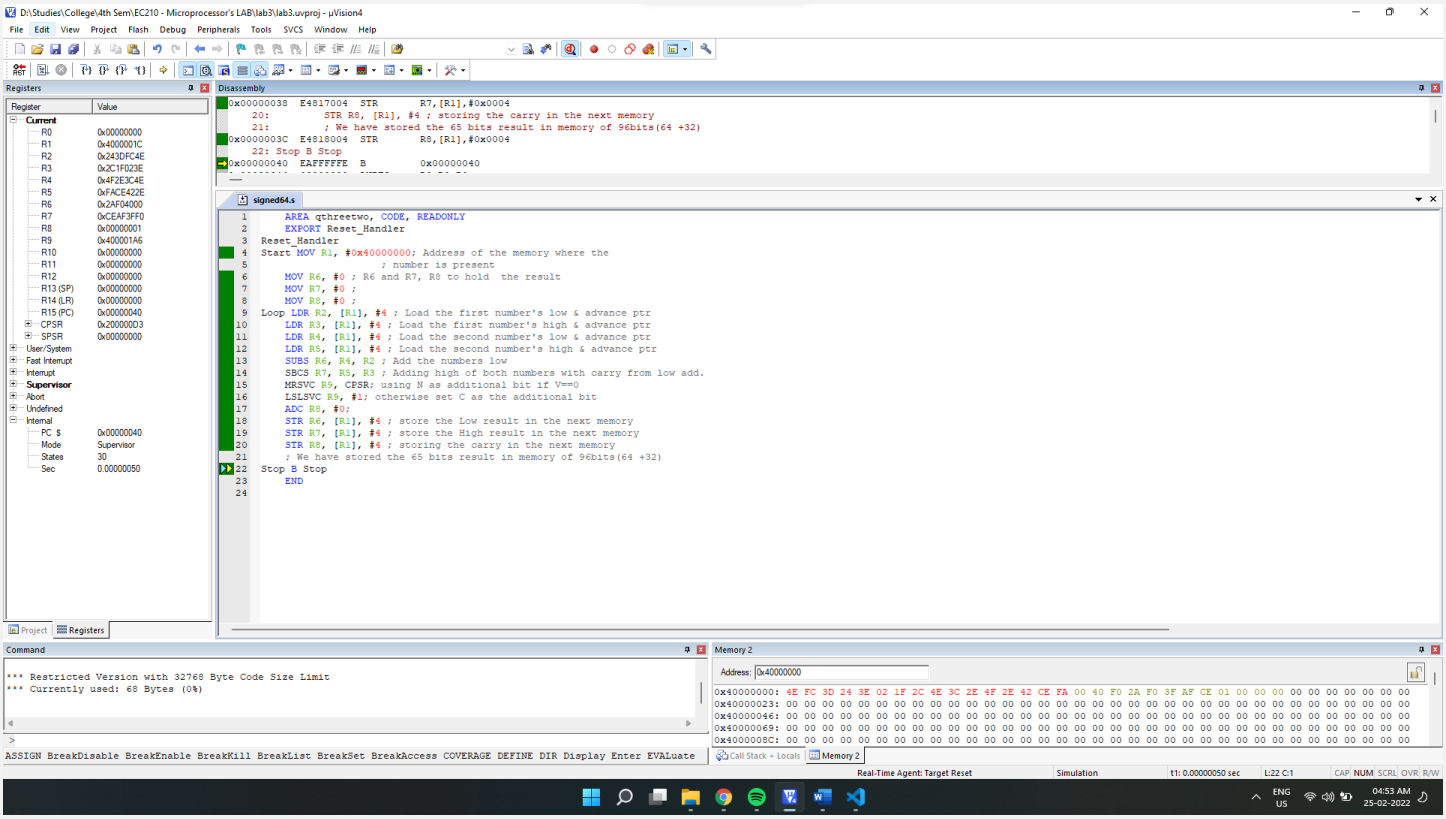


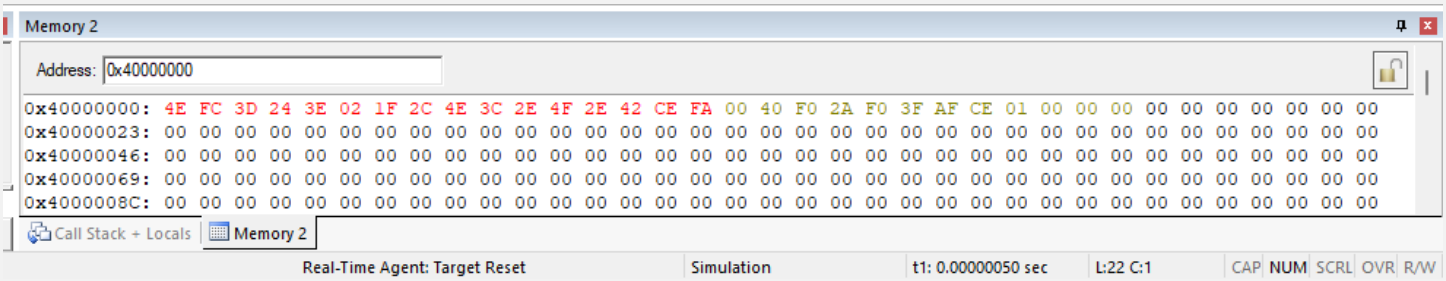
Setup:



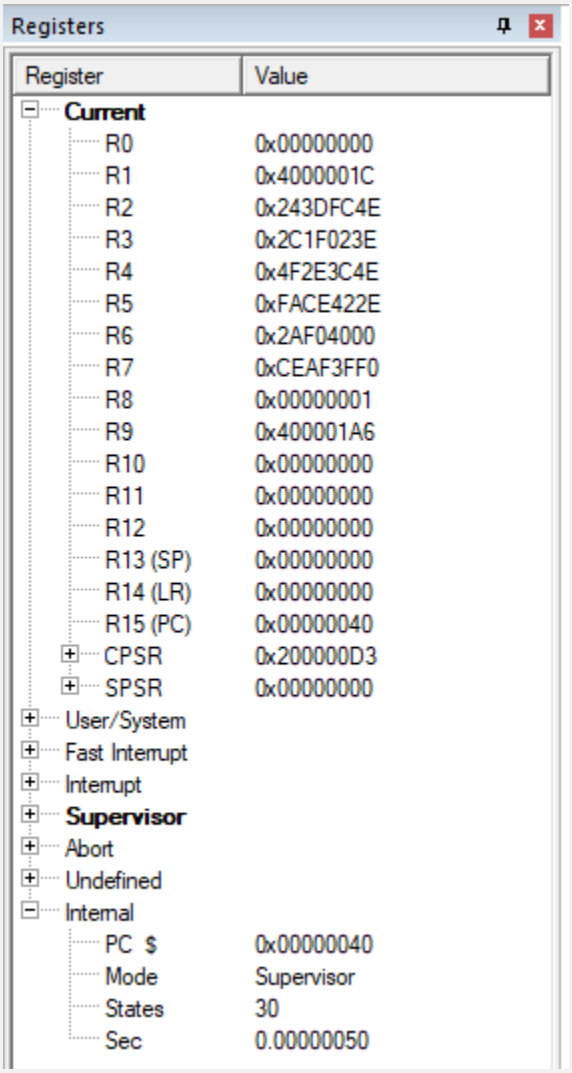
After Execution:

Final Output:





Final Register Values:



3.4] If a and b are two 32 bit numbers stored in consecutive memory location, swap the register content using EOR instruction and store back the results.

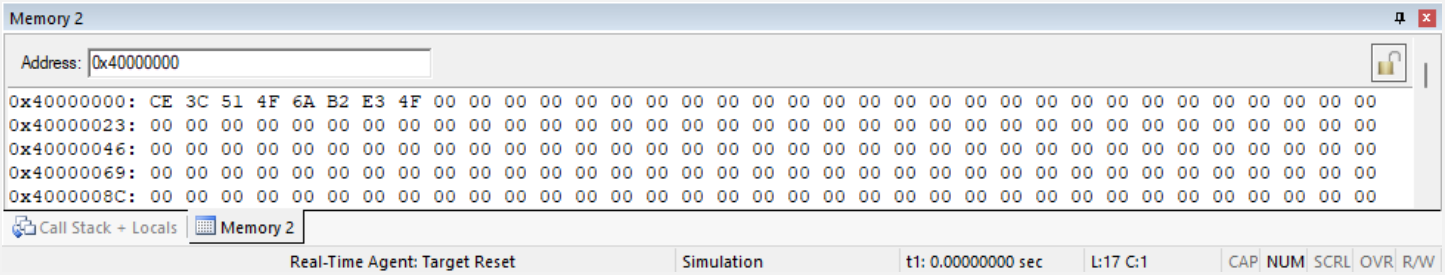
The EOR does bitwise XOR, this can be used to swap two variables. The XOR of two numbers x and y returns a number that has all the bits as 1 wherever bits of x and y differ.

Source code:

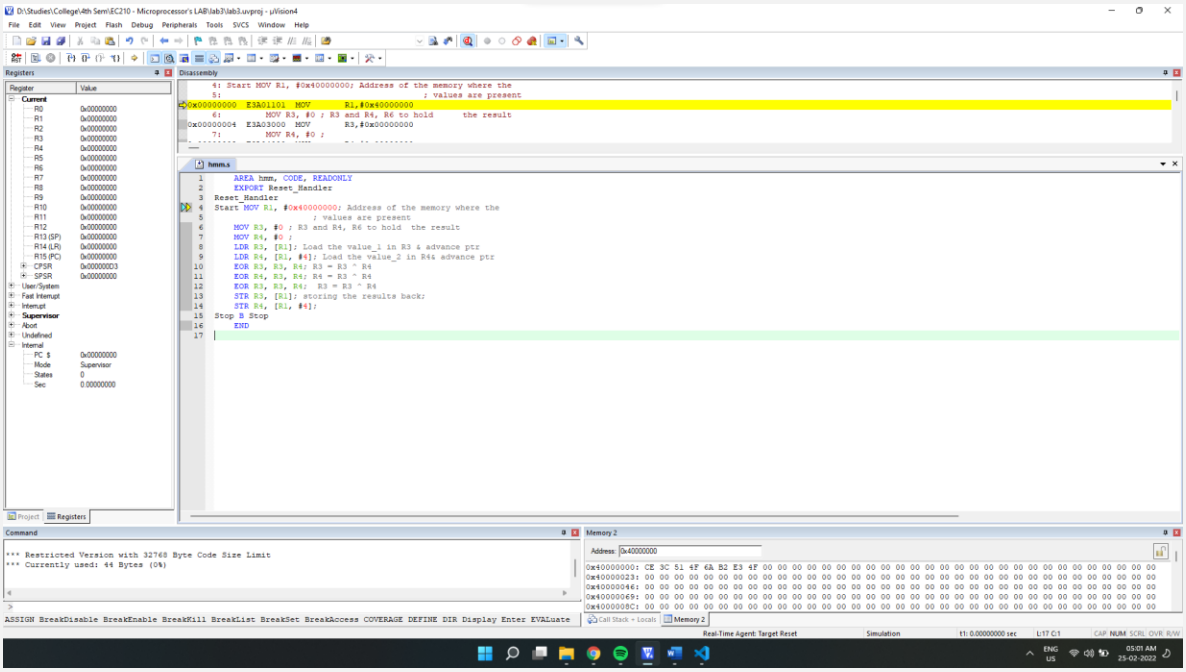
```
    AREA hmm, CODE, READONLY
    EXPORT Reset_Handler
Reset_Handler
Start MOV R1, #0x40000000; Address of the memory where the
                        ; values are present
    MOV R3, #0 ; R3 and R4, R6 to hold  the result
    MOV R4, #0 ;
    LDR R3, [R1]; Load the value_1 in R3 & advance ptr
    LDR R4, [R1, #4]; Load the value_2 in R4& advance ptr
    EOR R3, R3, R4; R3 = R3 ^ R4
    EOR R4, R3, R4; R4 = R3 ^ R4
    EOR R3, R3, R4;  R3 = R3 ^ R4
    STR R3, [R1]; storing the results back;
    STR R4, [R1, #4];
Stop B Stop
    END
```

Debugging:

Initial Memory:

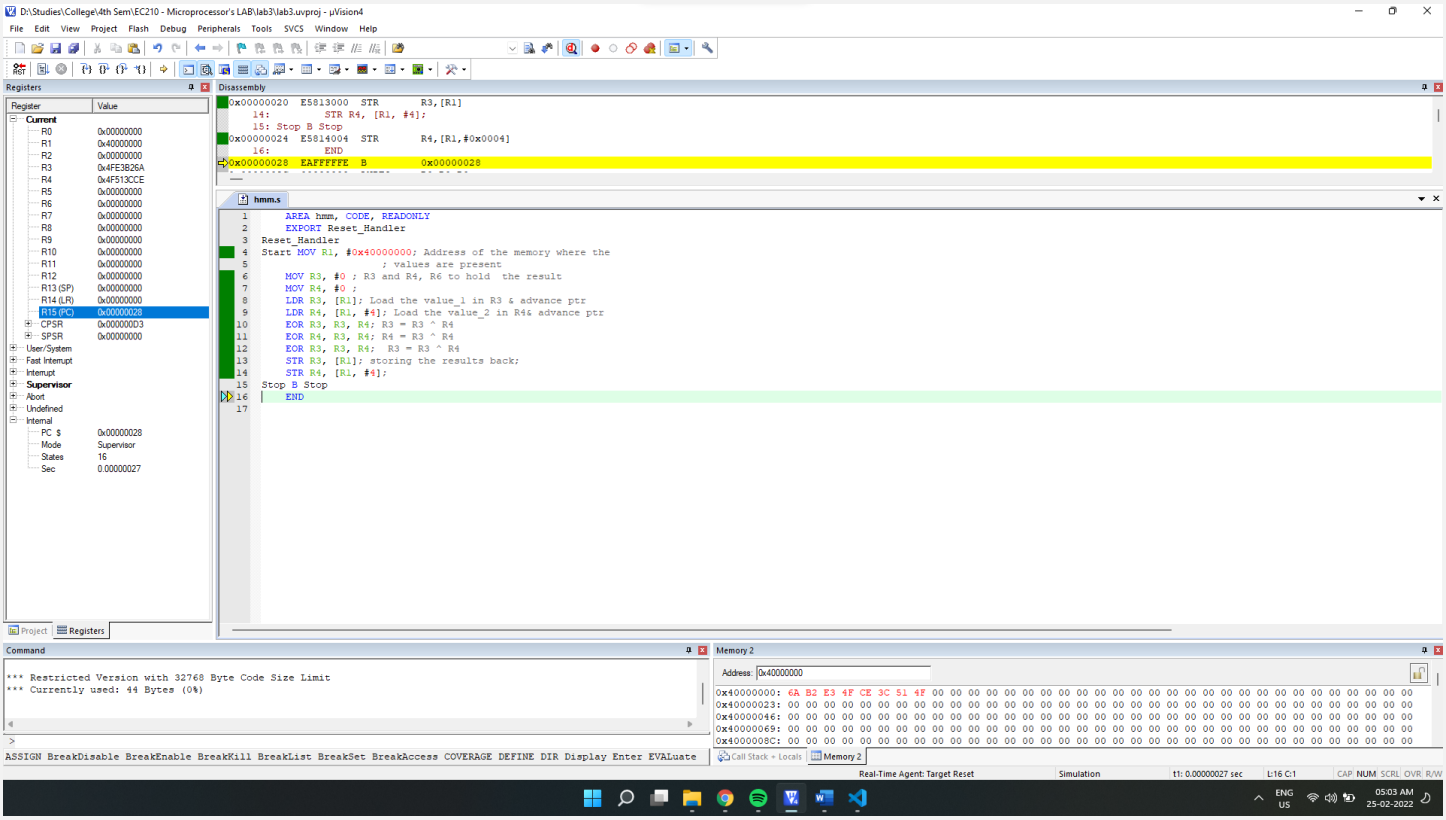


Setup:

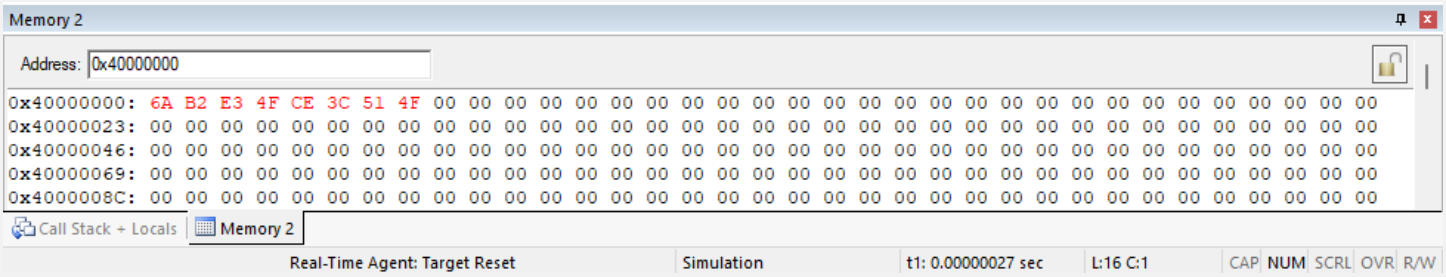


After Execution:

Final Output:



Final Memory:



Observation: we can see that the memory values are swapped for addresses 0x40000000 and 0x40000004.

3.5] SWP to add a constant value 0xFF to a block of 10 words stored in the consecutive memory locations using LDM and STM instructions.

->

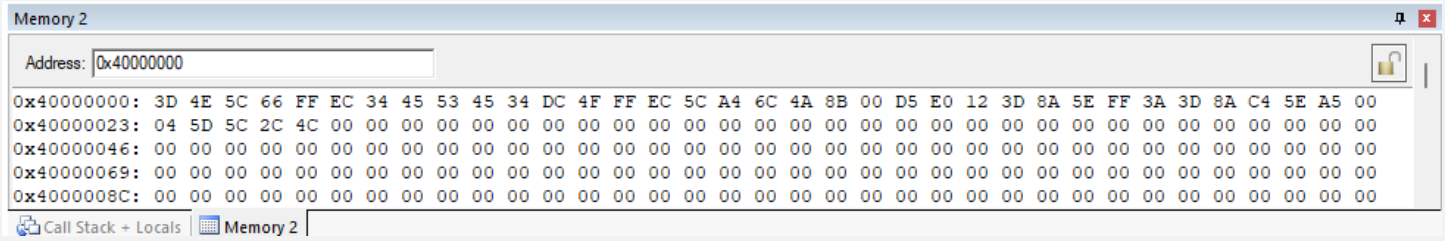
We will make use of LDMIA to load multiple registers and then use ADD instruction to add the required constant 0xFF. Then store all those values with STMIA.

Source code:

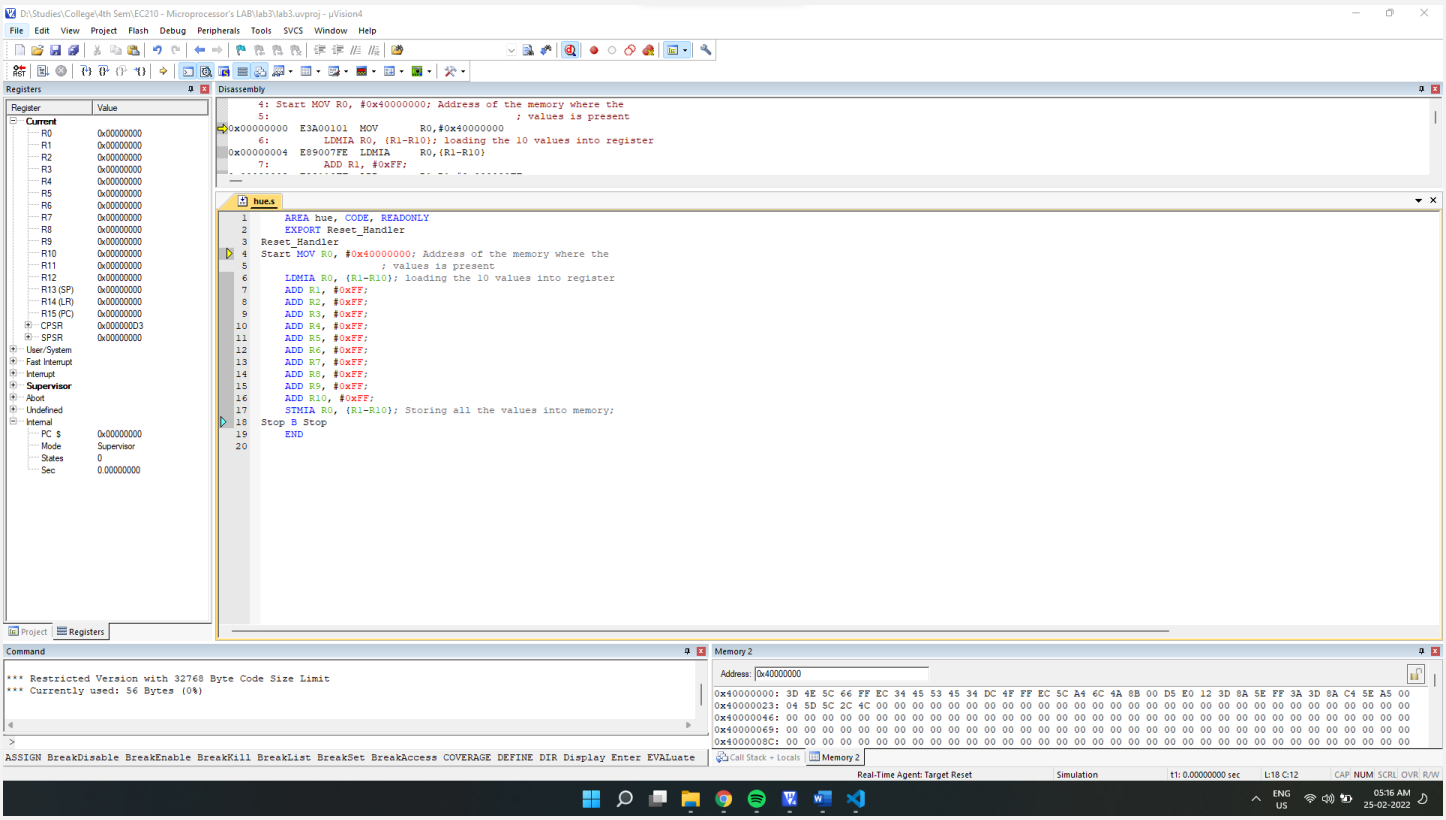
```
    AREA hue, CODE, READONLY
    EXPORT Reset_Handler
Reset_Handler
Start MOV R0, #0x40000000; Address of the memory where the
                        ; values is present
    LDMIA R0, {R1-R10}; loading the 10 values into register
    ADD R1, #0xFF;
    ADD R2, #0xFF;
    ADD R3, #0xFF;
    ADD R4, #0xFF;
    ADD R5, #0xFF;
    ADD R6, #0xFF;
    ADD R7, #0xFF;
    ADD R8, #0xFF;
    ADD R9, #0xFF;
    ADD R10, #0xFF;
    STMIA R0, {R1-R10}; Storing all the values into memory;
Stop B Stop
    END
```

Debugging:

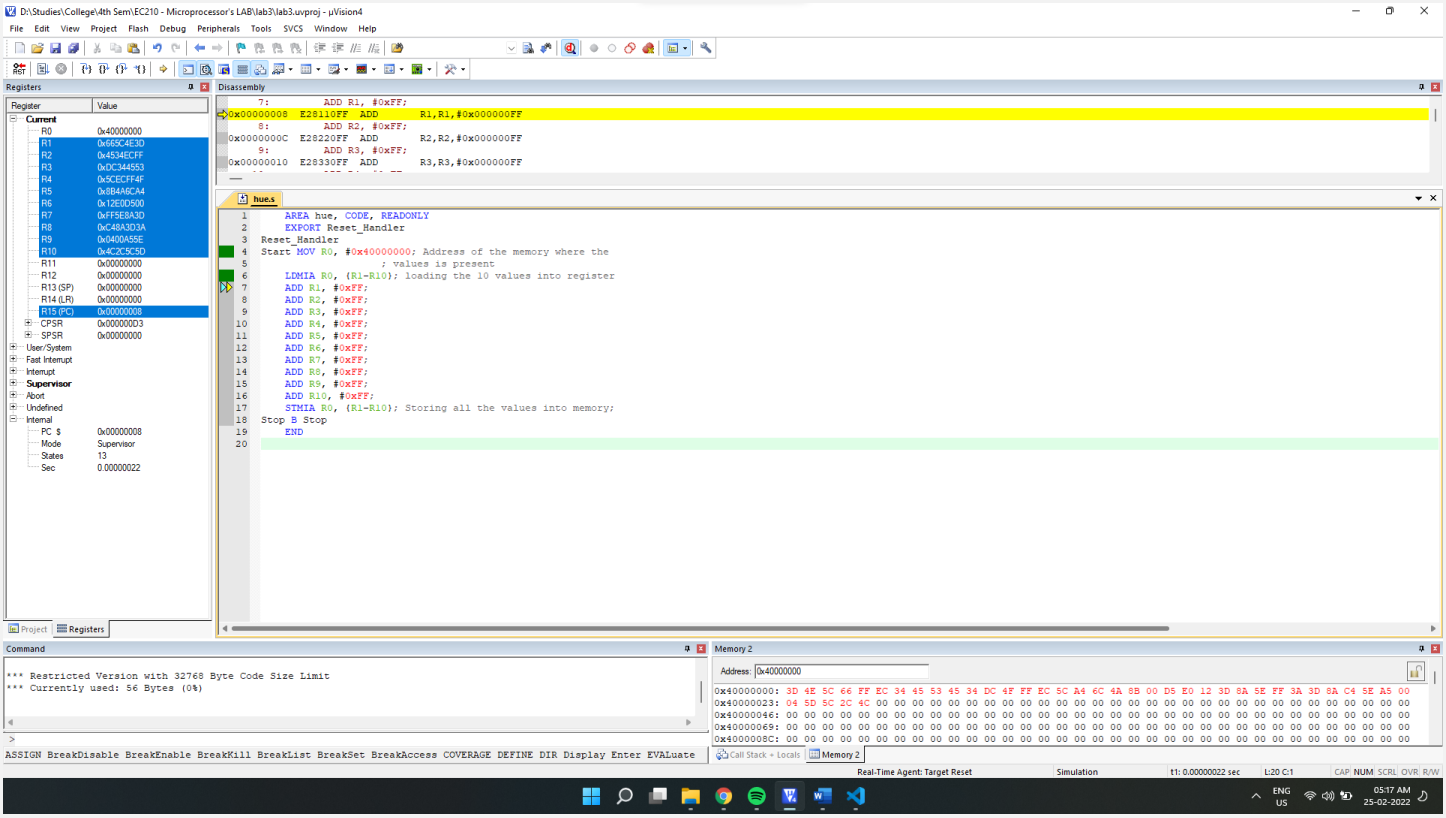
Initial Memory:

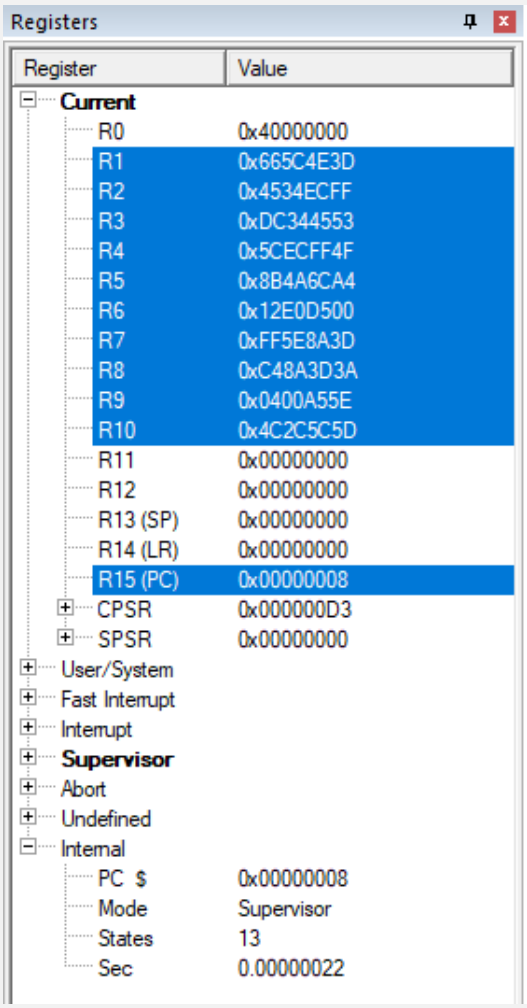


Setup:

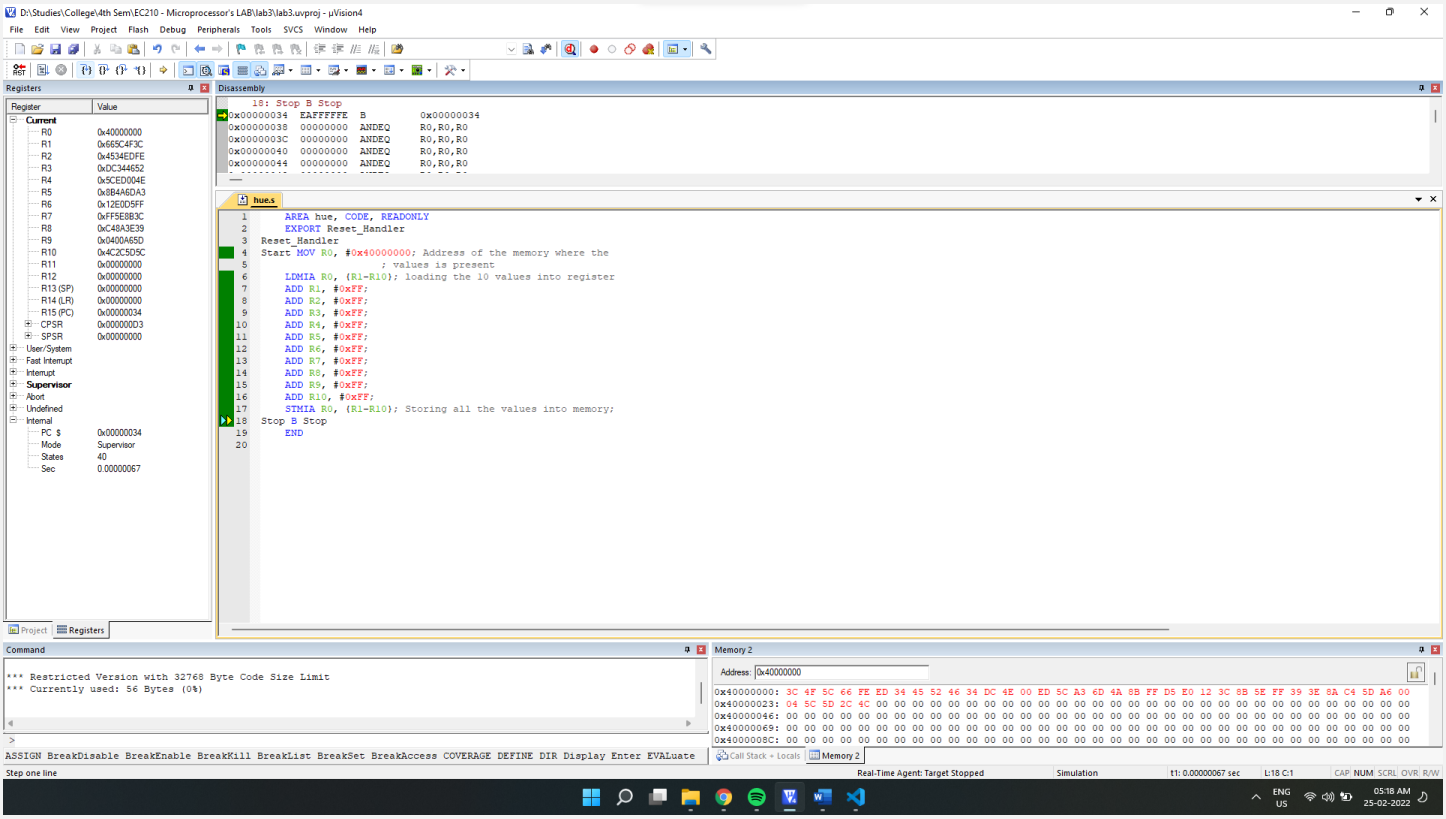


After Registers get loaded:

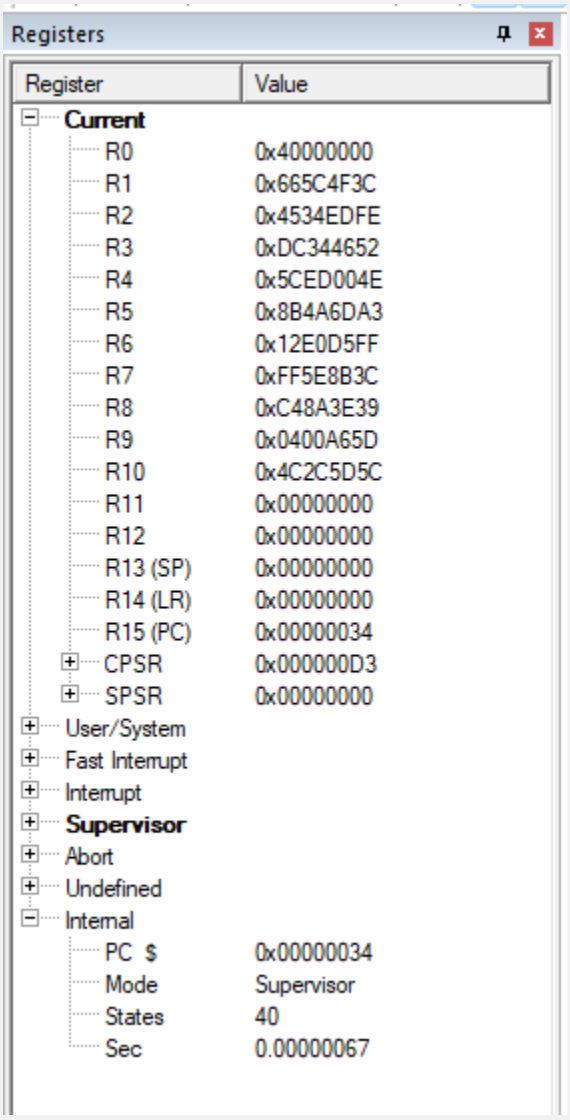


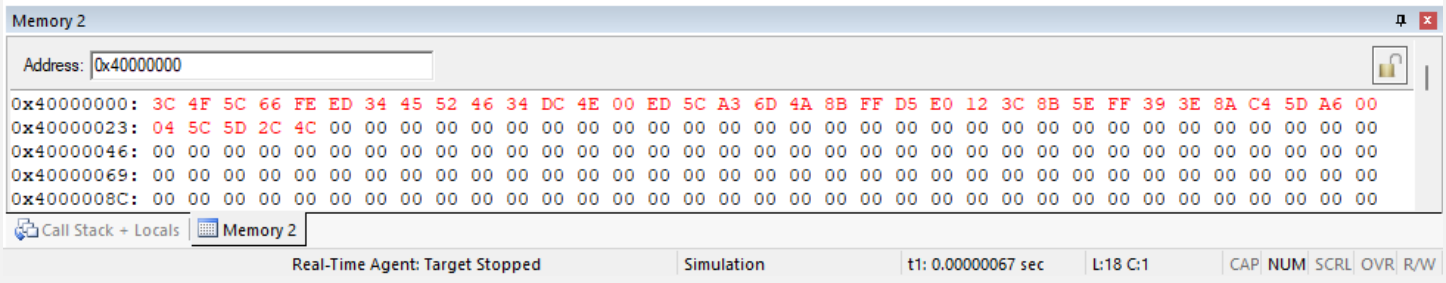


After complete Execution:



Final Output:





3.6] Exchange a block of 32 bit data stored in the processor registers (R0- R7) with that in the memory starting at address 0x40000000.

We will first load values into registers r0-r7 (from different memory location 0x50000000) that are to be swapped from the memory at location starting from **0x40000000** using LDMIA. Then we will make use of **SWP** instruction to swap values between register and memory. For repeating the instruction for different registers, we will use **ADD** instruction to increment the register storing the address.

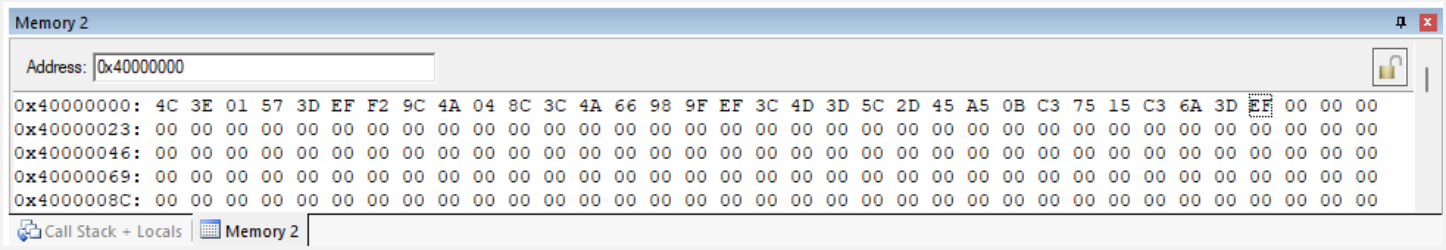
Source code:

```
    AREA huihui, CODE, READONLY
    EXPORT Reset_Handler
Reset_Handler
Start MOV R9, #0x50000000; for loading values into register
    LDMIA R9, {R0-R7}; Loading values into register that are to be swapped.
    MOV R8, #0x40000000; Address of the memory of which
; The values are to be swapped with the registers.
    SWP R0, R0, [R8];
    ADD R8, #4;
    SWP R1, R1, [R8];
    ADD R8, #4;
    SWP R2, R2, [R8];
    ADD R8, #4;
    SWP R3, R3, [R8];
    ADD R8, #4;
    SWP R4, R4, [R8];
    ADD R8, #4;
    SWP R5, R5, [R8];
    ADD R8, #4;
    SWP R6, R6, [R8];
    ADD R8, #4;
    SWP R7, R7, [R8];
Stop B Stop
    END
```

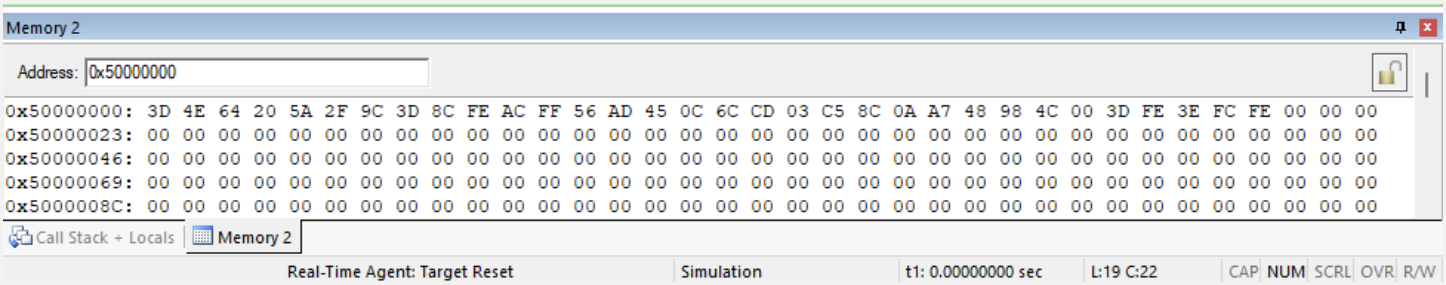
Debugging:

Initial Memory:

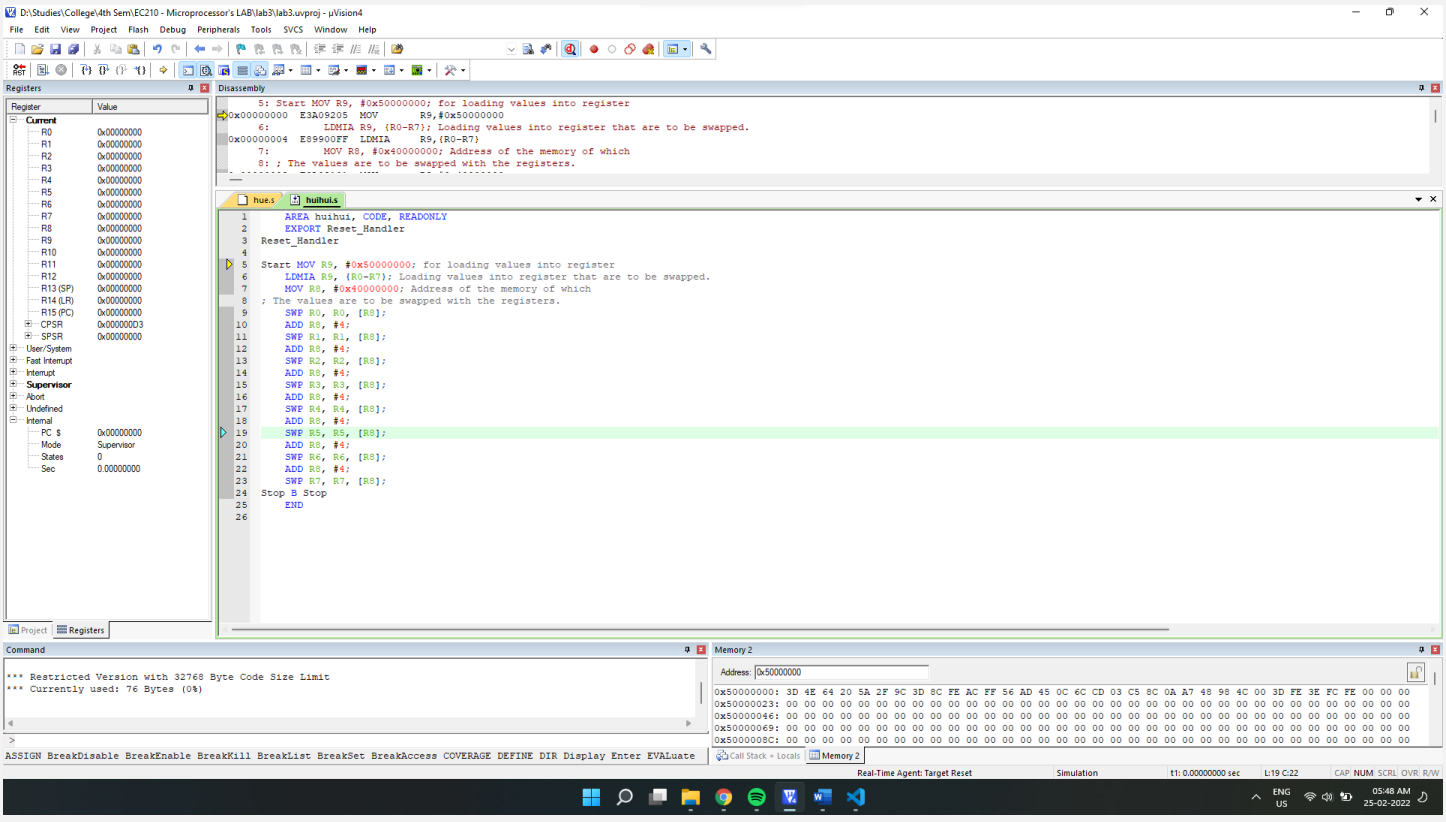
From 0x40000000



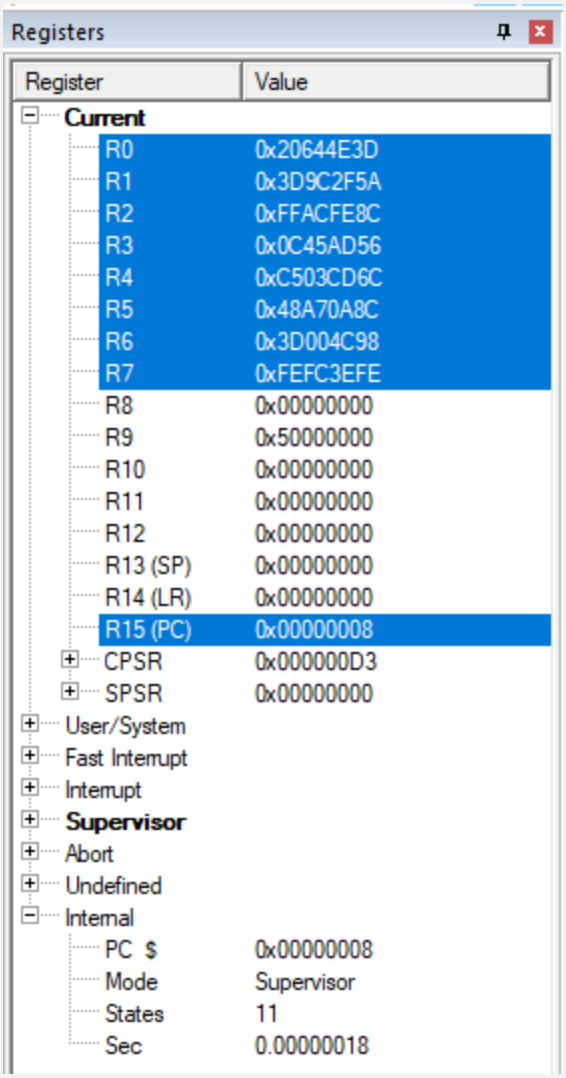
From 0x50000000 (for initializing register)

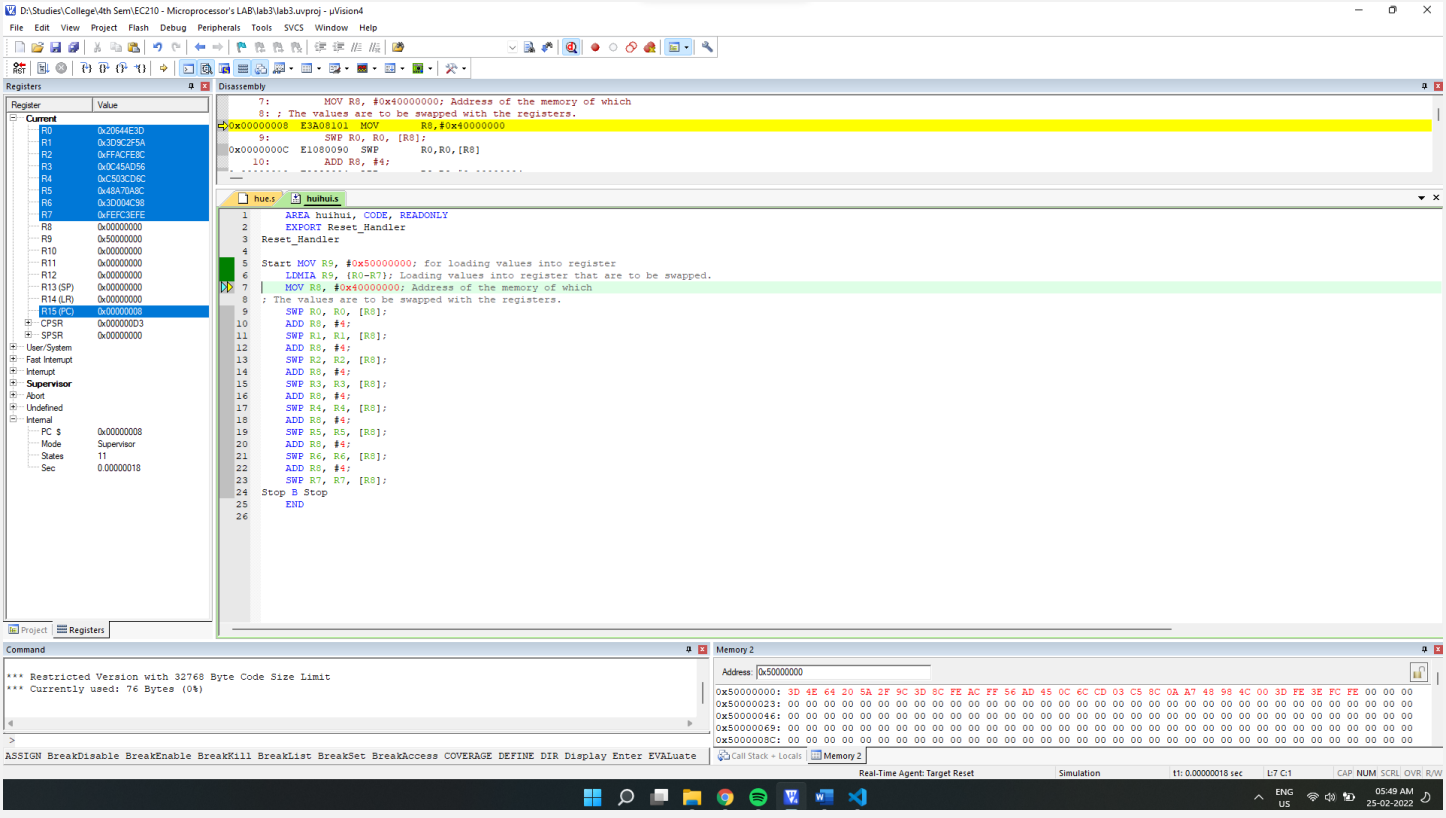


Setup:



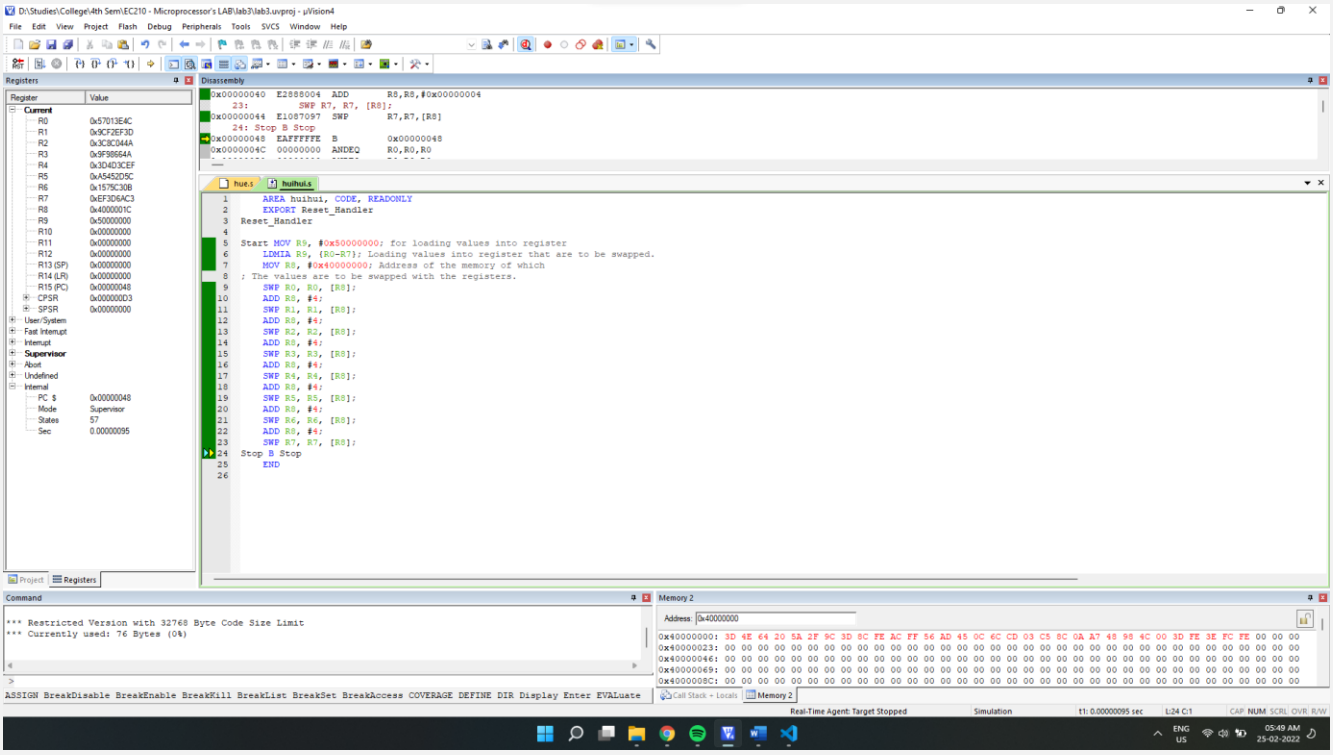
After registers get loaded:


Registers
Register
Value
Current
R0 0x20644E3D
R1 0x3D9C2F5A
R2 0xFFACFE8C
R3 0x0C45AD56
R4 0xC503CD6C
R5 0x48A70A8C
R6 0x3D004C98
R7 0xFEFC3EFE
R8 0x00000000
R9 0x50000000
R10 0x00000000
R11 0x00000000
R12 0x00000000
R13 (SP) 0x00000000
R14 (LR) 0x00000000
R15 (PC) 0x00000008
CPSR 0x000000D3
SPSR 0x00000000
User/System
Fast Interrupt
Interrupt
Supervisor
Abort
Undefined
Internal
PC $ 0x00000008
Mode Supervisor
States 11
Sec 0.00000018

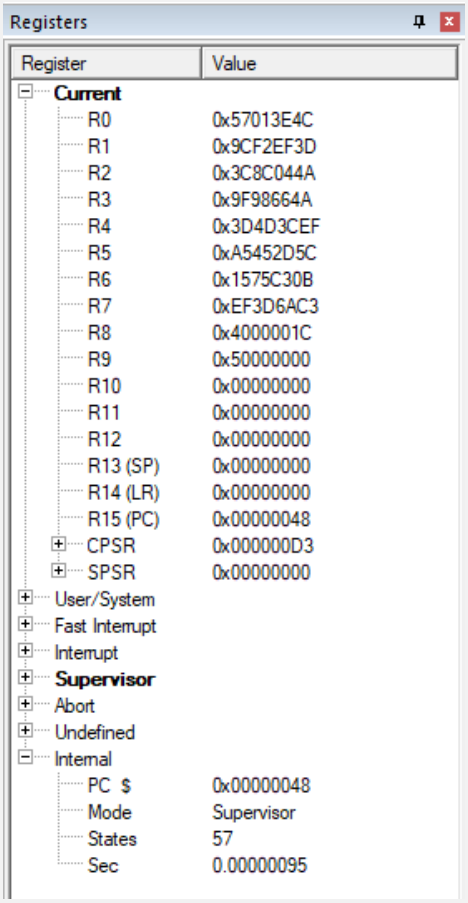

D:\Studies\College\4th Sem\EC210 - Microprocessor's LAB\lab3\lab3.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Disassembly
7: MOV R8, #0x40000000; Address of the memory of which
8: ; The values are to be swapped with the registers.
0x00000008 E3A08101 MOV R8,#0x40000000
9: SWP R0, R0, [R8];
0x0000000C E1080090 SWP R0,R0,[R8]
10: ADD R8, #4;
hue.s
huihui.s
1 AREA huihui, CODE, READONLY
2 EXPORT Reset_Handler
3 Reset_Handler
4
5 Start MOV R9, #0x50000000; for loading values into register
6 LDMIA R9, {R0-R7}; Loading values into register that are to be swapped.
7 MOV R8, #0x40000000; Address of the memory of which
8 ; The values are to be swapped with the registers.
9 SWP R0, R0, [R8];
10 ADD R8, #4;
11 SWP R1, R1, [R8];
12 ADD R8, #4;
13 SWP R2, R2, [R8];
14 ADD R8, #4;
15 SWP R3, R3, [R8];
16 ADD R8, #4;
17 SWP R4, R4, [R8];
18 ADD R8, #4;
19 SWP R5, R5, [R8];
20 ADD R8, #4;
21 SWP R6, R6, [R8];
22 ADD R8, #4;
23 SWP R7, R7, [R8];
24 Stop B Stop
25 END
26
Project
Registers
Command
*** Restricted Version with 32768 Byte Code Size Limit
*** Currently used: 76 Bytes (0%)
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE DEFINE DIR Display Enter EVALuate
Memory 2
Address: 0x50000000
0x50000000: 3D 4E 64 20 5A 2F 9C 3D 8C FE AC FF 56 AD 45 0C 6C CD 03 C5 8C 0A A7 48 98 4C 00 3D FE 3E FC FE 00 00 00
0x50000023: 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
0x50000046: 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
0x50000069: 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
0x5000008C: 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 2
Real-Time Agent: Target Reset
Simulation
t1: 0.00000018 sec
L:7 C:1
CAP NUM SCRL OVR R/W
ENG US
05:49 AM
25-02-2022

After Execution:

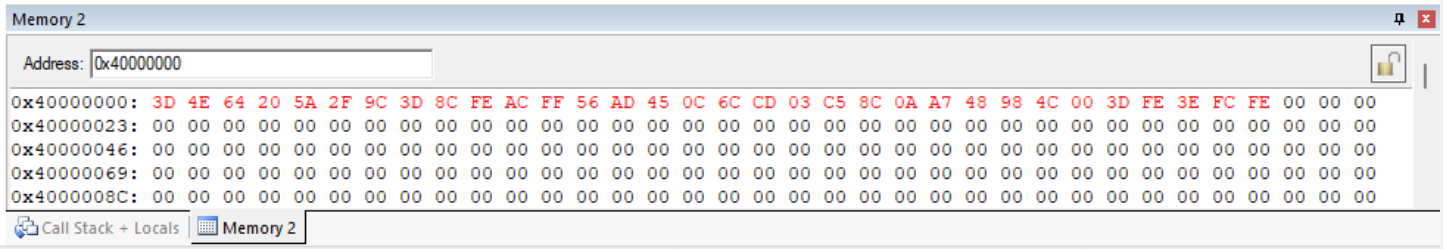
Final Output:



Final Values:



From 0x40000000



Observation:

We can see that values are swapped.

3.7] Take two Nos from the memory, No1 and No2 as given below
No1= 0xBDF85433 and No2 =0x8FFF5FF3

No1= 0x967ff500 and No2 =0x50000000

No1 =0x74FFF000 and No2 =0x6800AFFF

No1 =0x80000000 and No2 =0xFFFF0000

(A) For each case perform 32 bit signed comparison and set the register R3 to 0xFF is No1 is greater than No2. Else set R3 to 0x00 using conditional execution of instructions.

(B) For each case perform 32 bit unsigned comparison and set the register R3 to 0xFF is No1 is greater than No2. Else set R3 to 0x00 using conditional execution of instructions.

->

a) We will use LDR for first loading both the values into register, then compare them with CMP and then make use of condition execution code **GT(for signed numbers)** as suffix with **ADD** instruction made to add 0xFF in R3 for required condition(greater than).

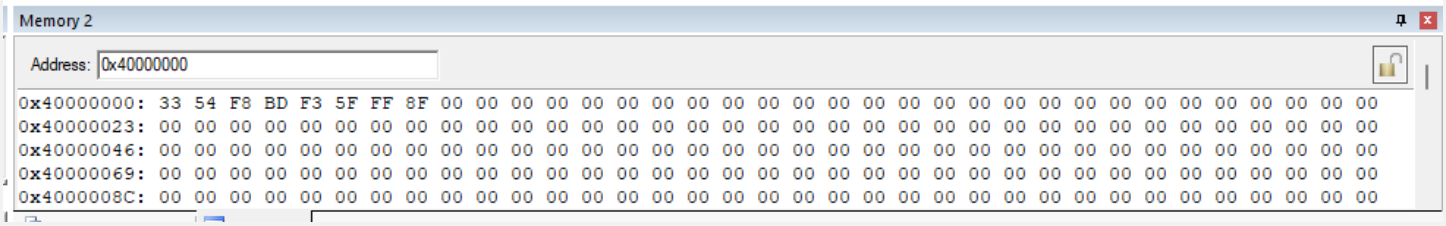
Source Code:

```
    AREA qthreetwo, CODE, READONLY
    EXPORT Reset_Handler
Reset_Handler
Start MOV R1, #0x40000000; Address of the memory where the
                        ; number is present
    MOV R3, #0 ; R3 holds the result
    LDR R0, [R1], #4 ; Load the first number & advance ptr
    LDR R2, [R1], #4 ; Load the second number  & advance ptr
    CMP R0, R2; compare
    ADDGT R3, #0xFF; add if greater than(signed)
Stop B Stop
    END
```

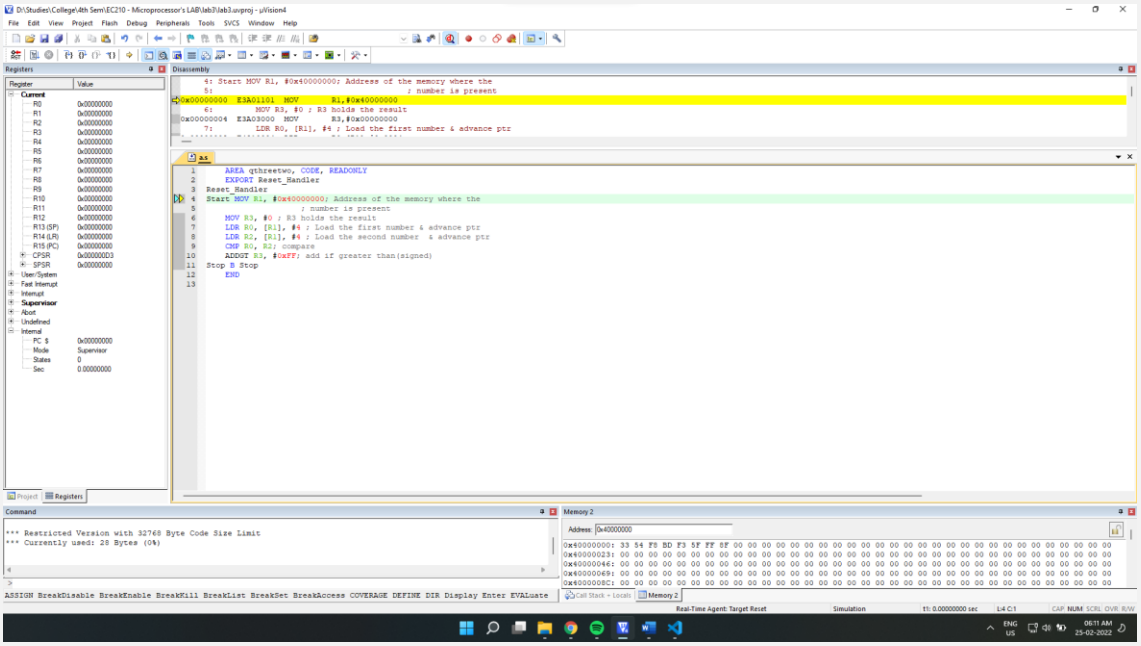
No1= 0xBDF85433 and No2 =0x8FFF5FF3

Debugging:

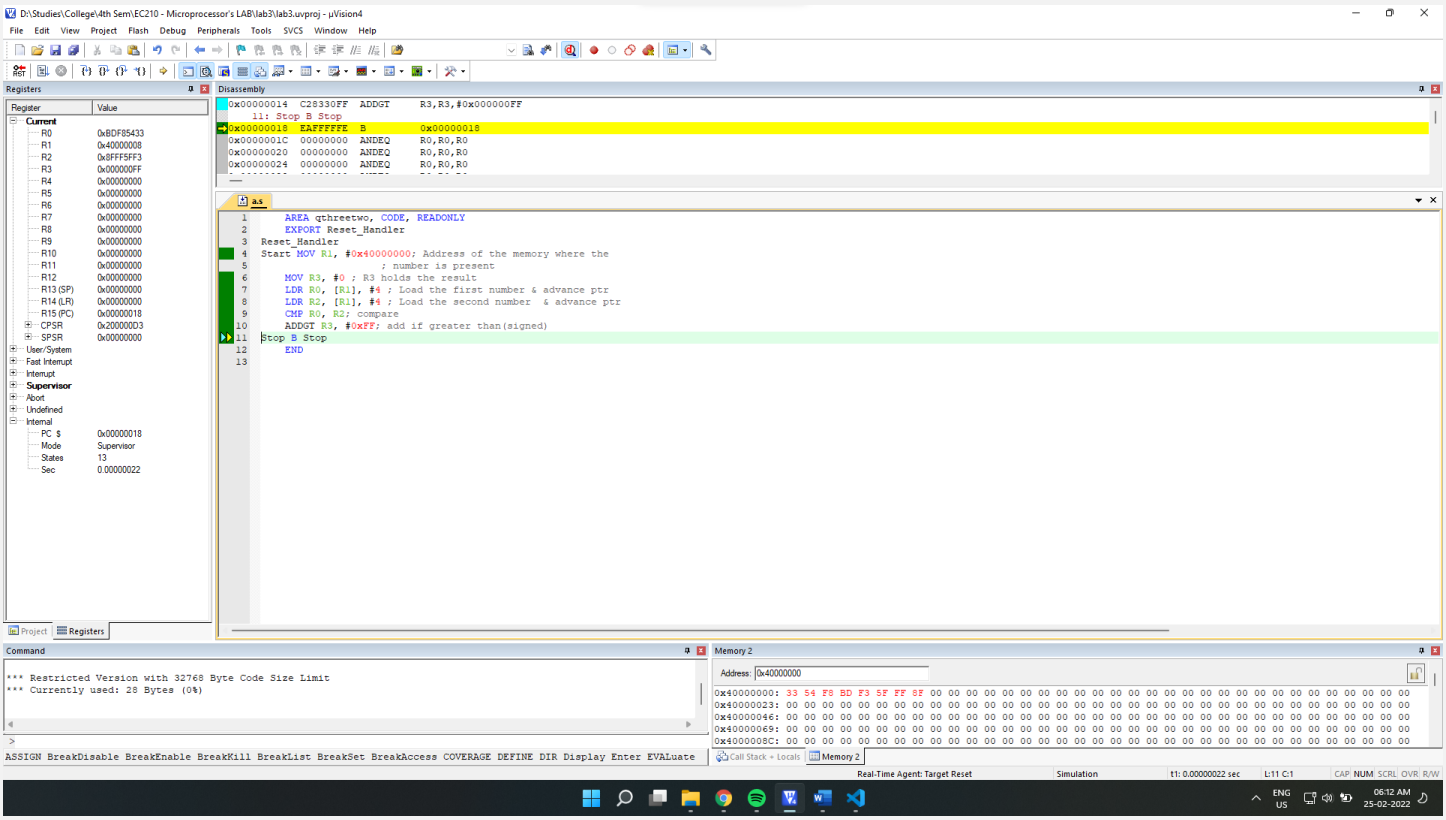
Initial Memory:

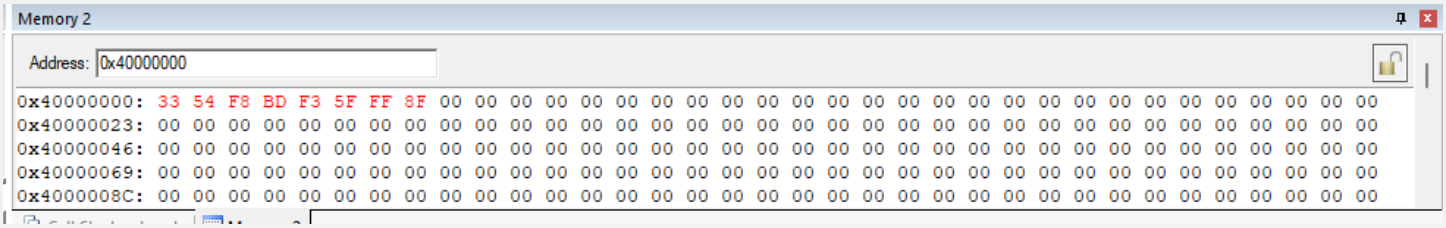


Setup:

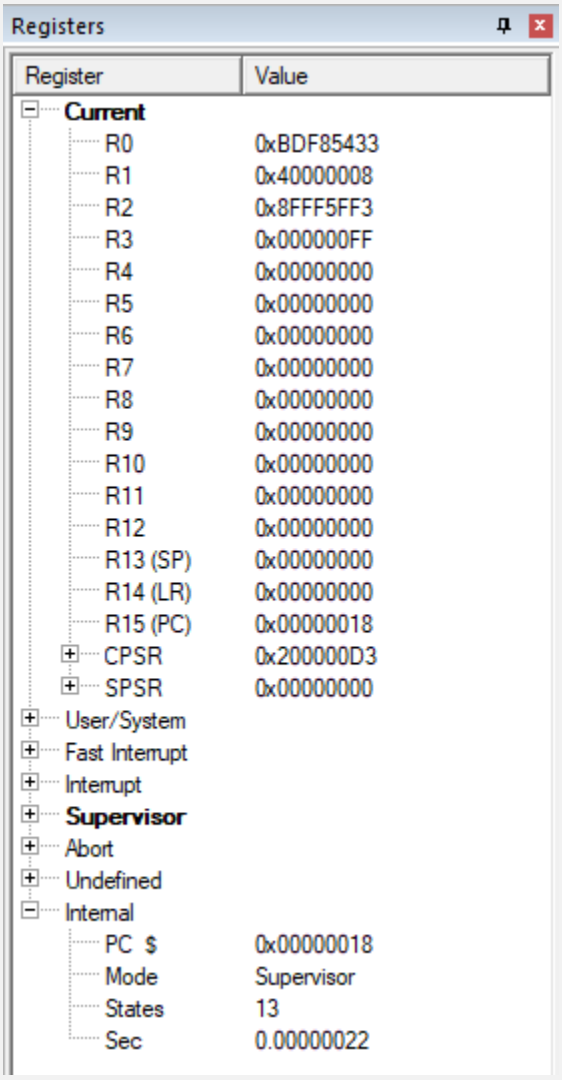


Final Output:





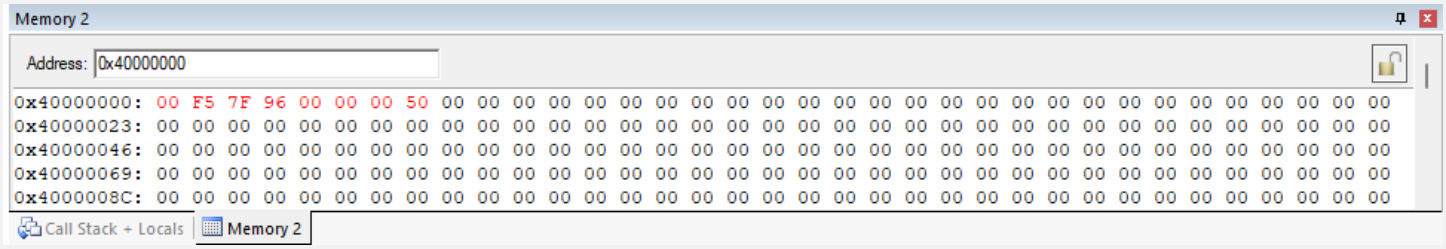
Final Register Values:



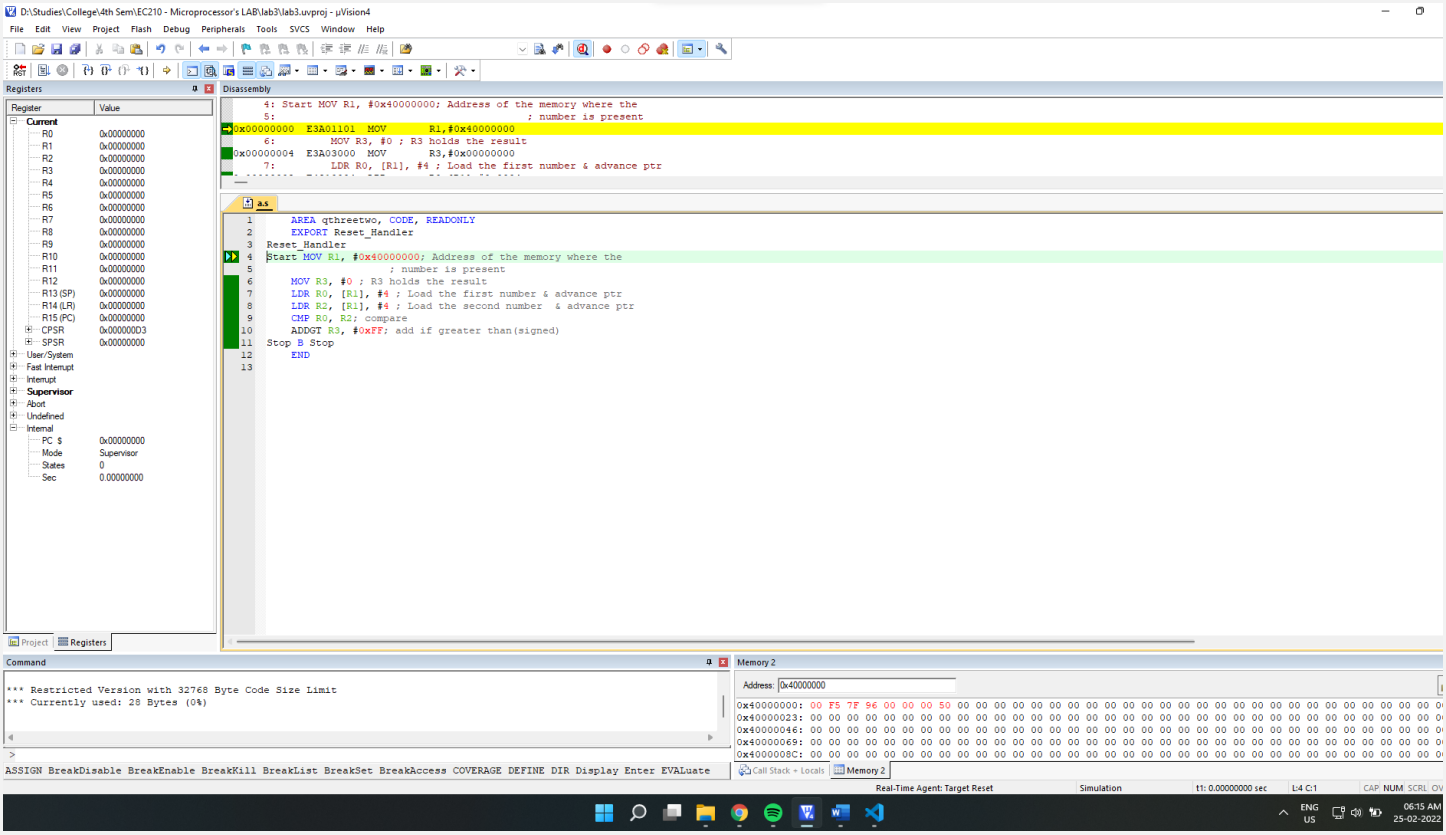
No1= 0x967ff500 and No2 =0x50000000

Debugging:

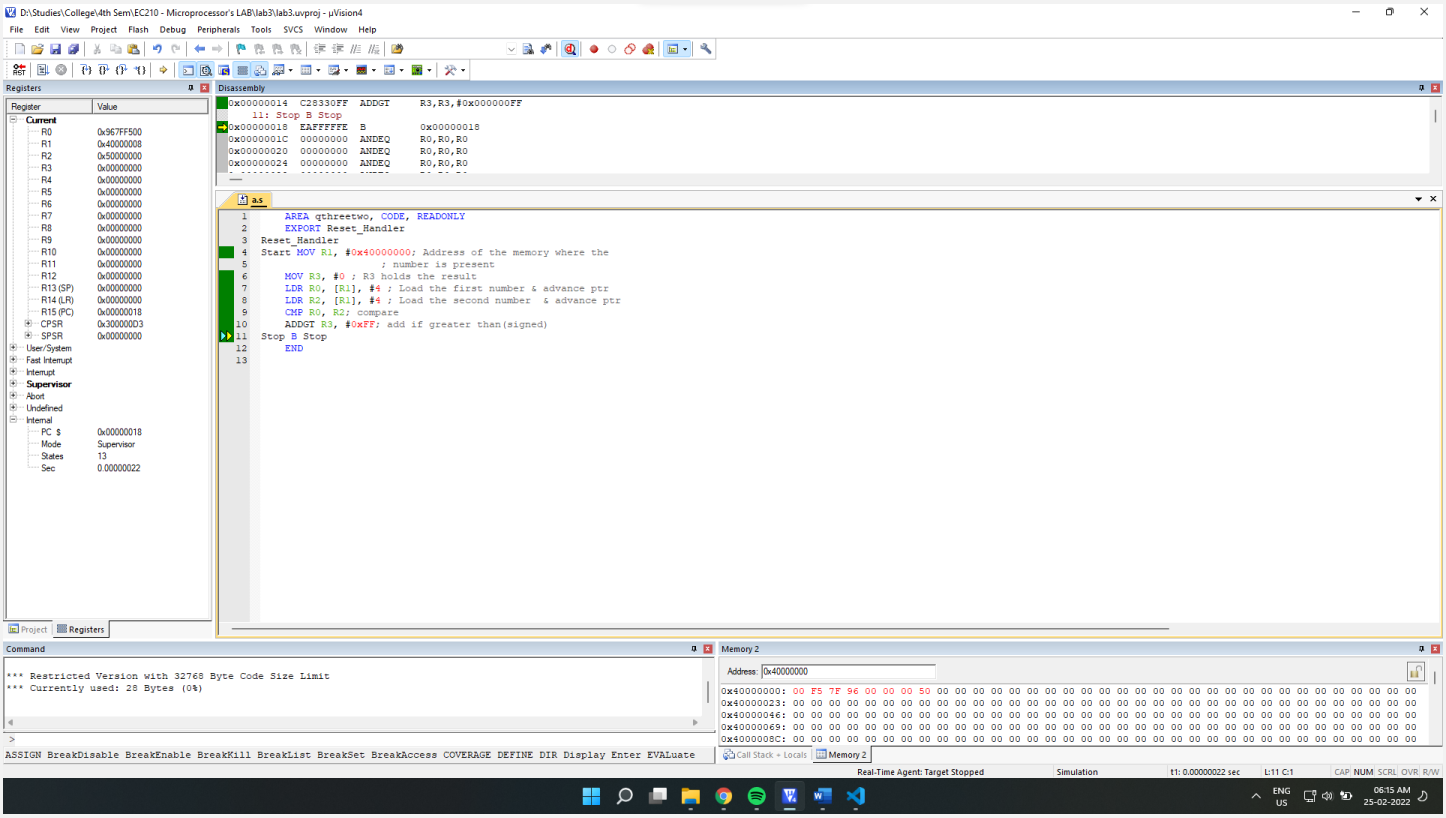
Initial Memory:



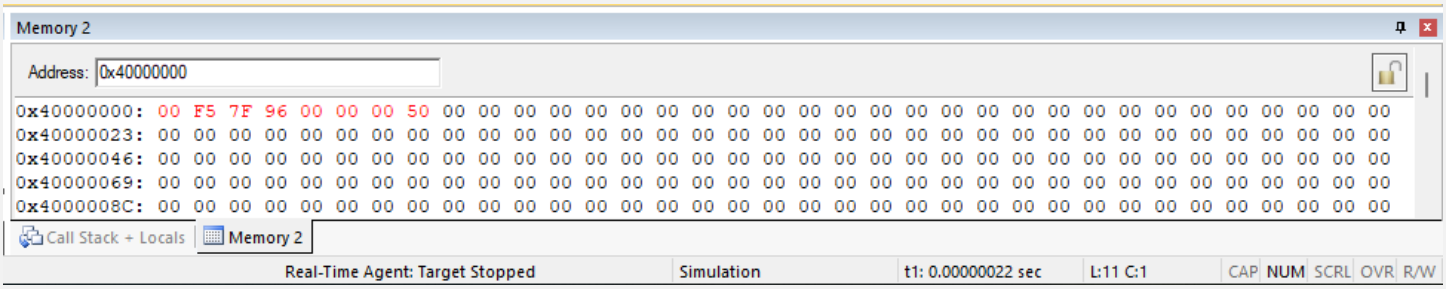
Setup:



After Execution:



Final Output:



Final Register Values:

Registers

Register	Value
Current	
R0	0x967FF500
R1	0x40000008
R2	0x50000000
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)	0x00000018
CPSR	0x300000D3
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC $	0x00000018
Mode	Supervisor
States	13
Sec	0.00000022

No1 =0x74FFF000 and No2 =0x6800AFFF

Debugging:

Initial Memory:

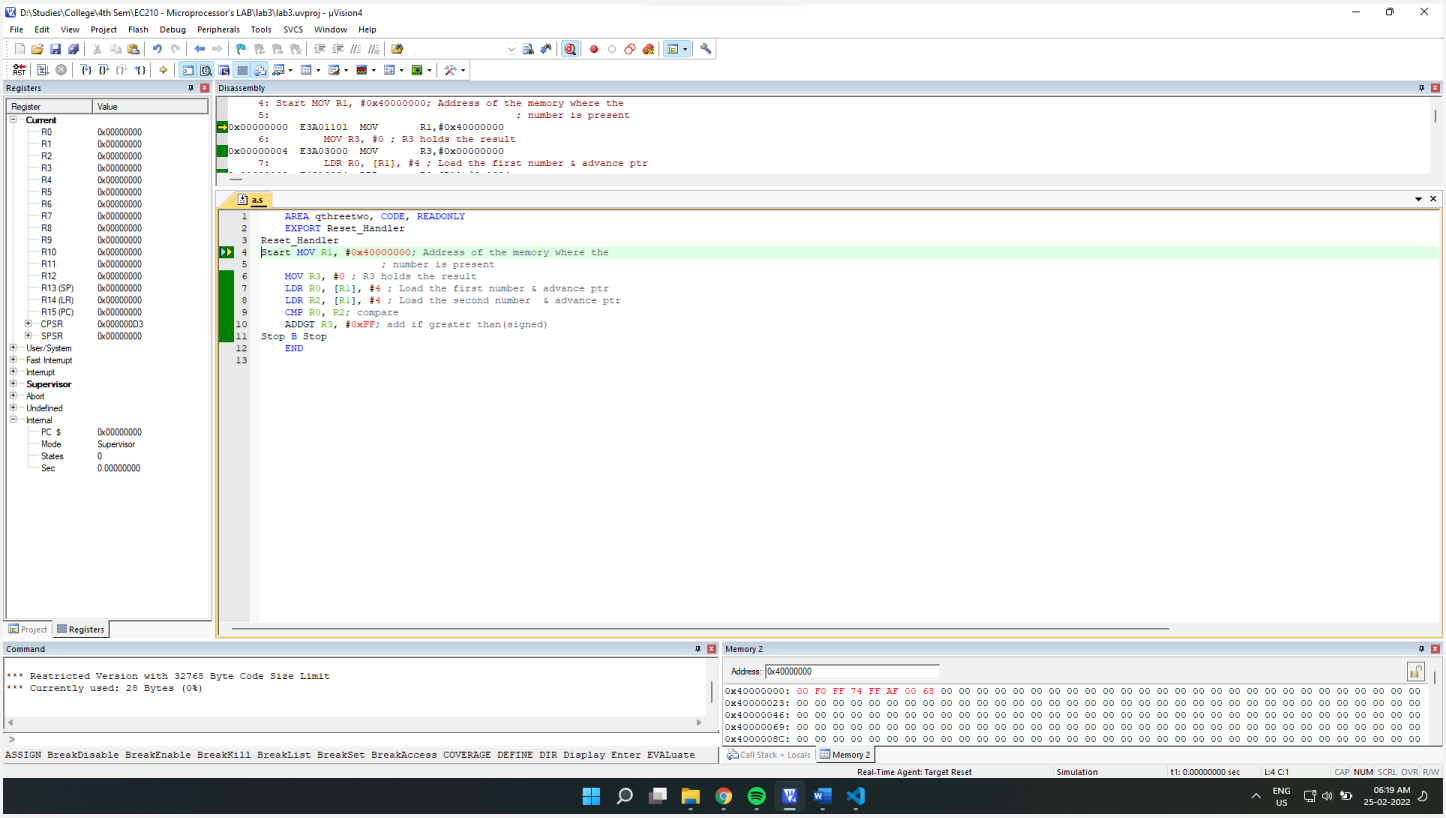
Memory 2

Address: 0x40000000

```
0x40000000: 00 F0 FF 74 FF AF 00 68 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000023: 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
0x40000046: 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
0x40000069: 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
0x4000008C: 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
```

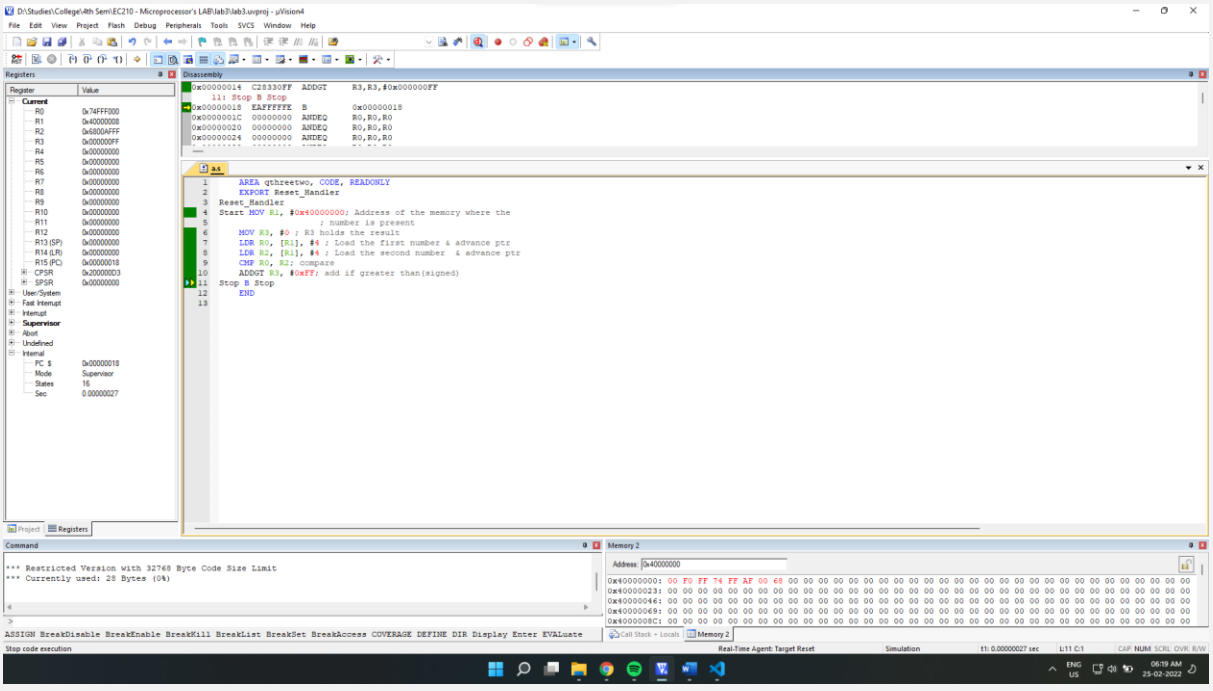
Call Stack + Locals | Memory 2

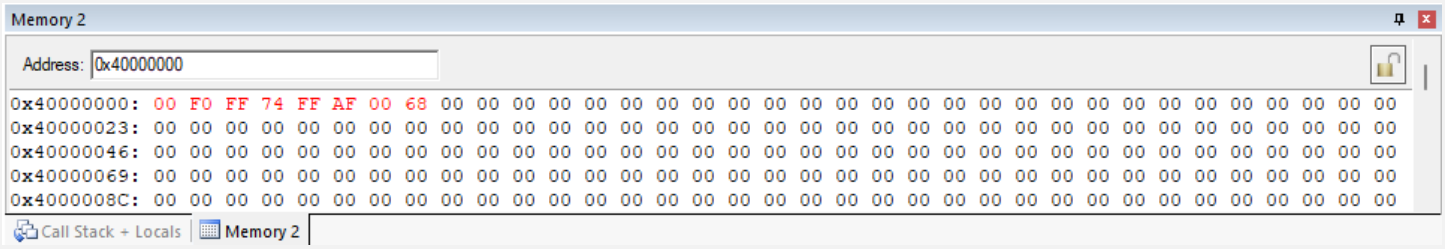
Setup:



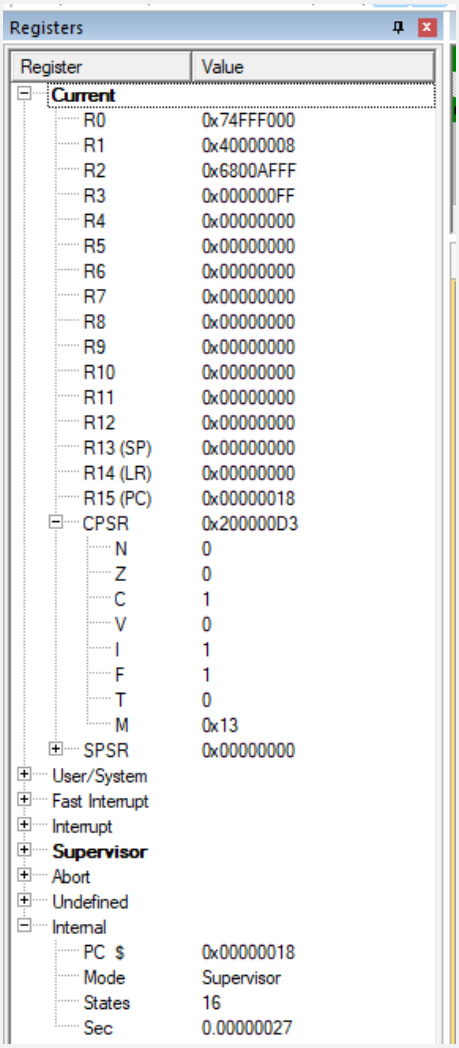
After Execution:

Final Output:





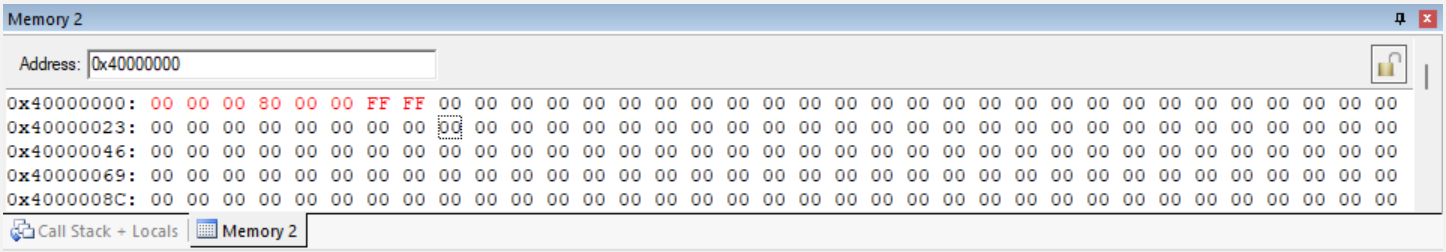
Final Register Values:



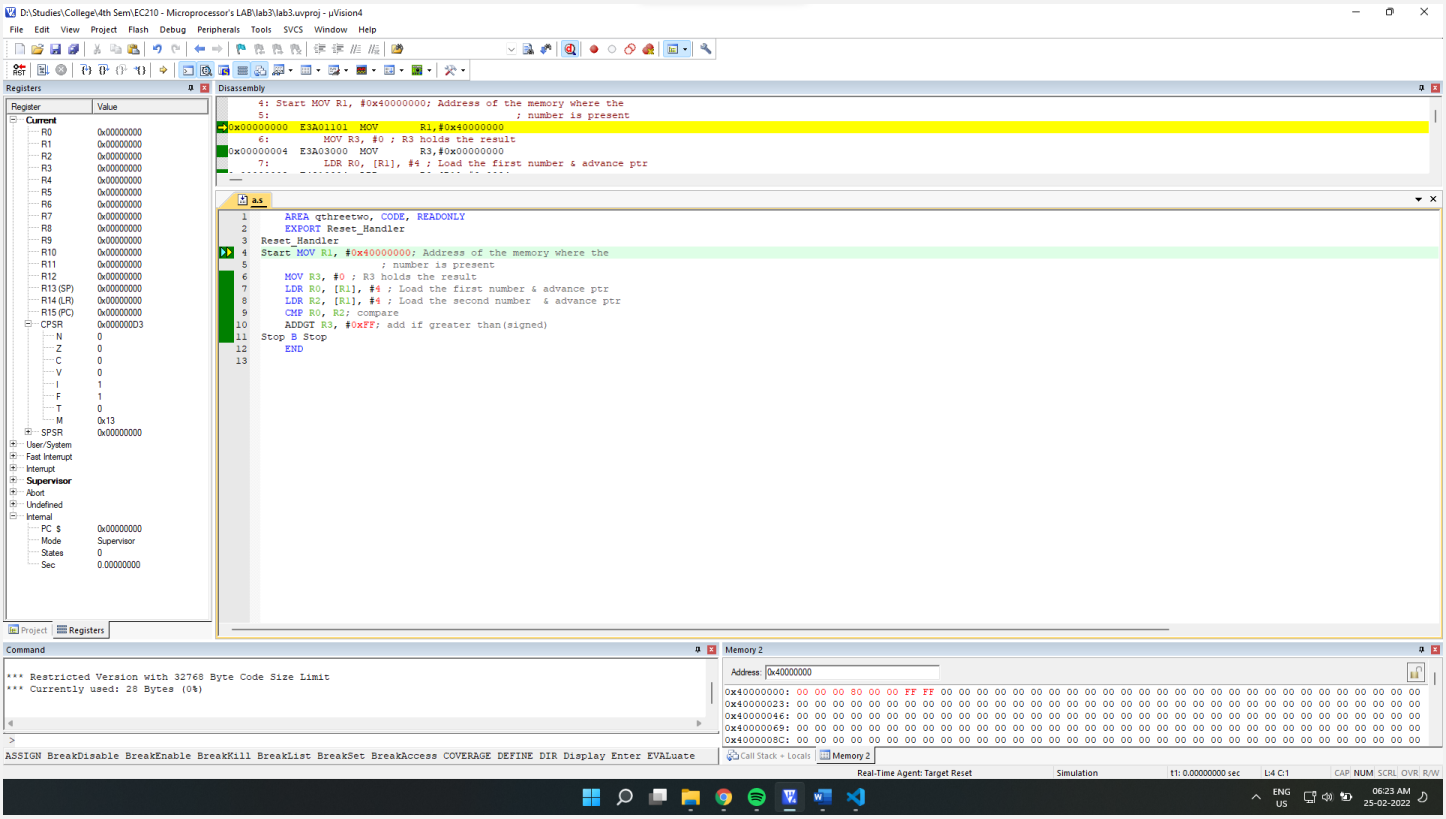
No1 =0x80000000 and No2 =0xFFFF0000

Debugging:

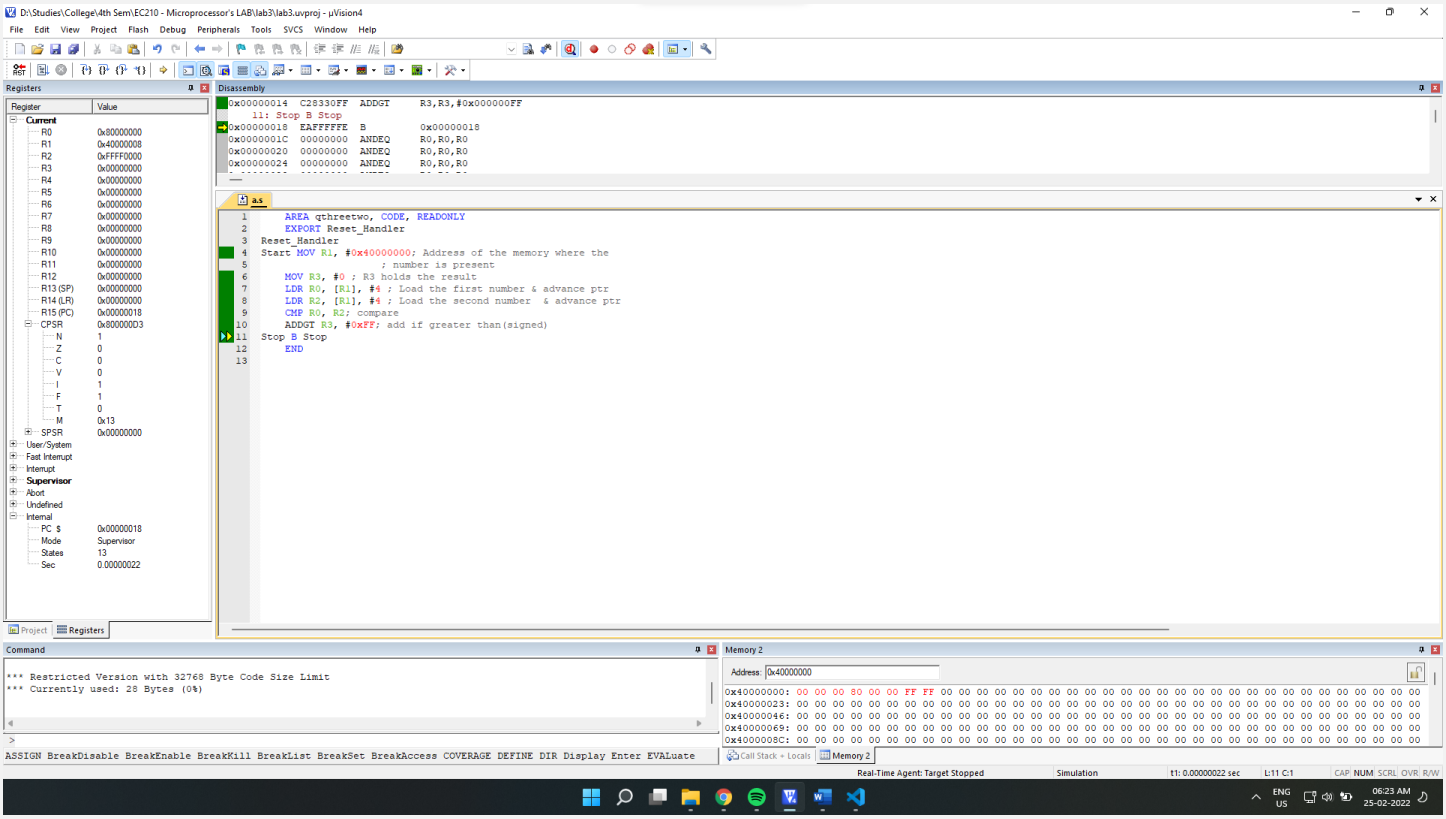
Initial Memory:



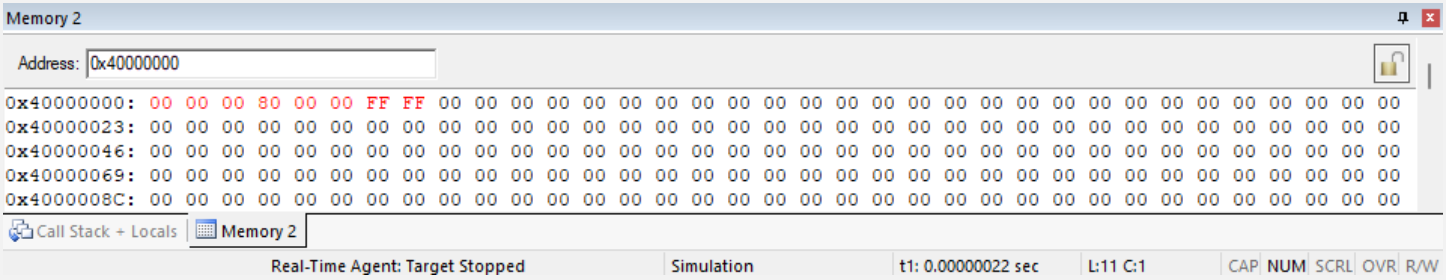
Setup:



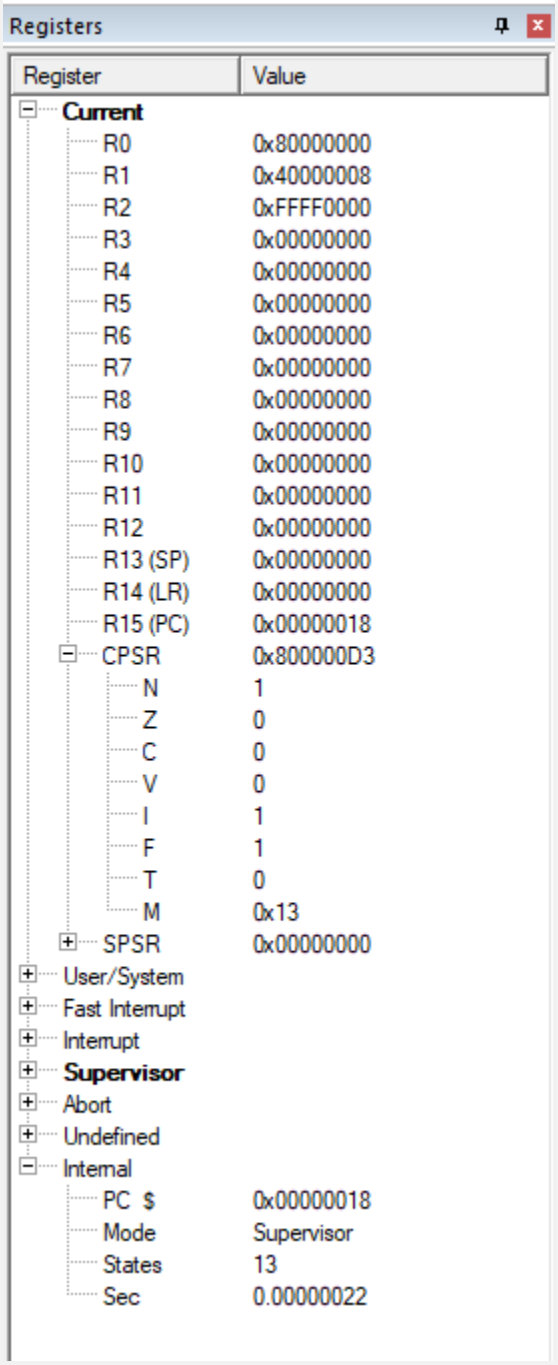
After Execution:



Final Output:



Final Register Values:



b) We will use LDR for first loading both the values into register, then compare them with CMP and then make use of condition execution code **HI(for unsigned numbers)** as suffix with **ADD** instruction made to add 0xFF in R3 for required condition(higher than/ greater than).

->

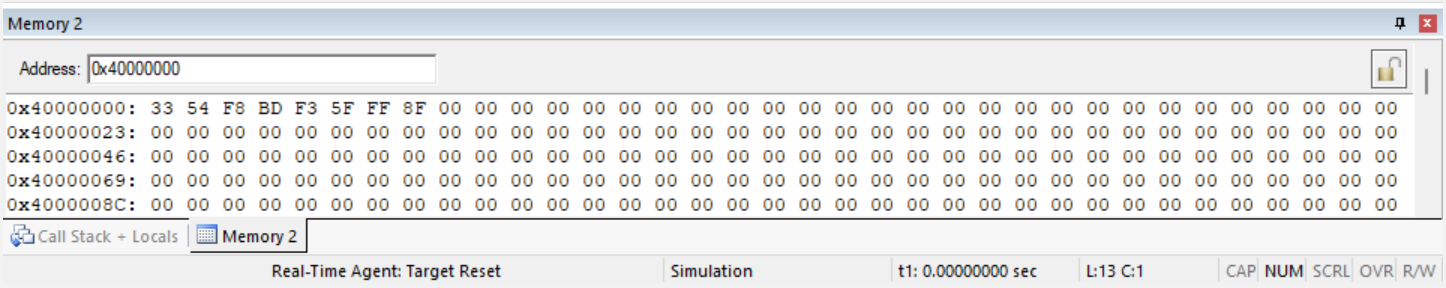
Source Code:

```
    AREA qthreetwo, CODE, READONLY
    EXPORT Reset_Handler
Reset_Handler
 Start MOV R1, #0x40000000; Address of the memory where the
                         ; number is present
    MOV R3, #0 ; R3 holds the result
    LDR R0, [R1], #4 ; Load the first number & advance ptr
    LDR R2, [R1], #4 ; Load the second number & advance ptr
    CMP R0, R2; compare
    ADDHI R3, #0xFF; add if higher/greater than(unsigned)
Stop B Stop
    END
```

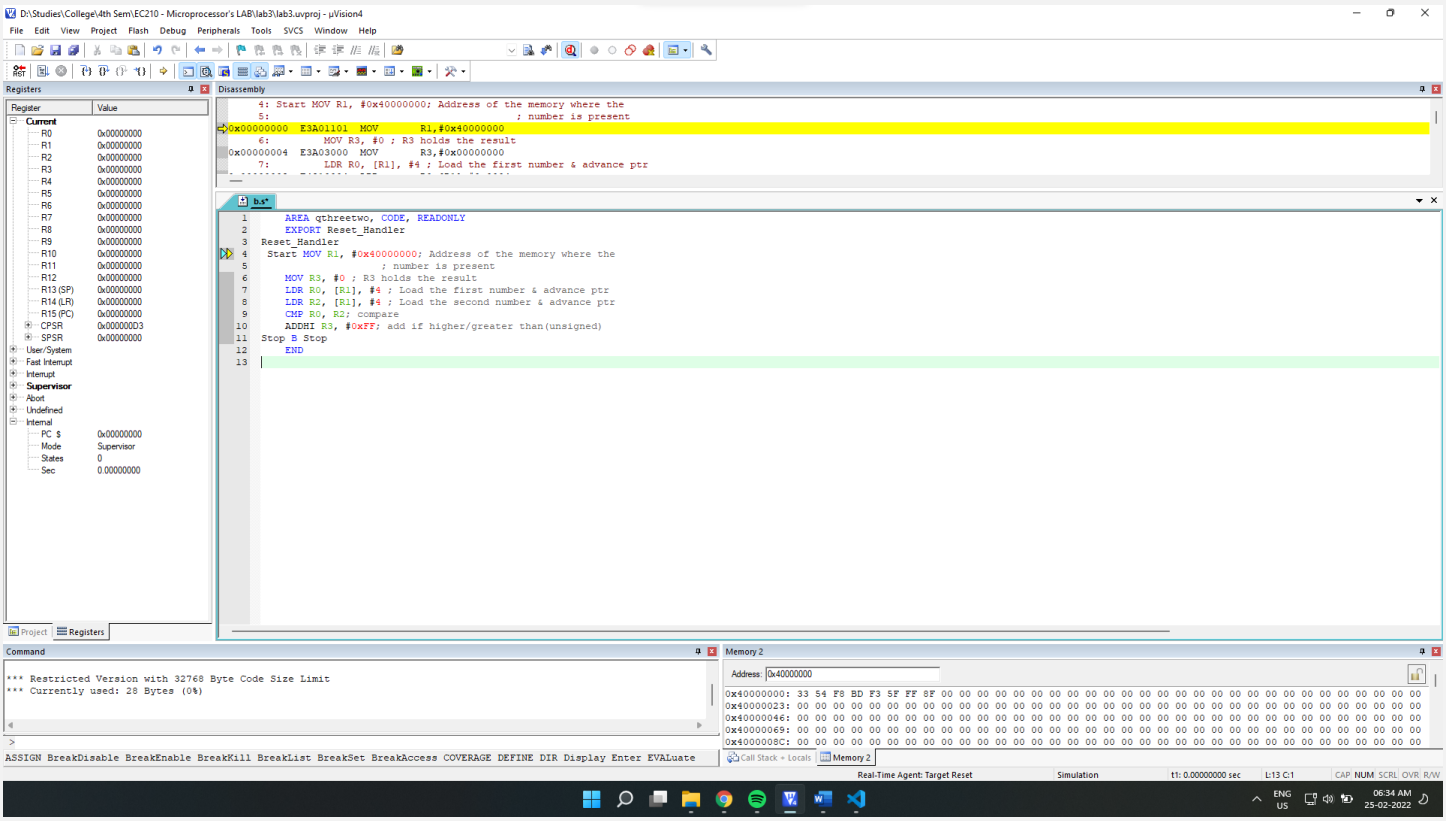
No1= 0xBDF85433 and No2 =0x8FFF5FF3

Debugging:

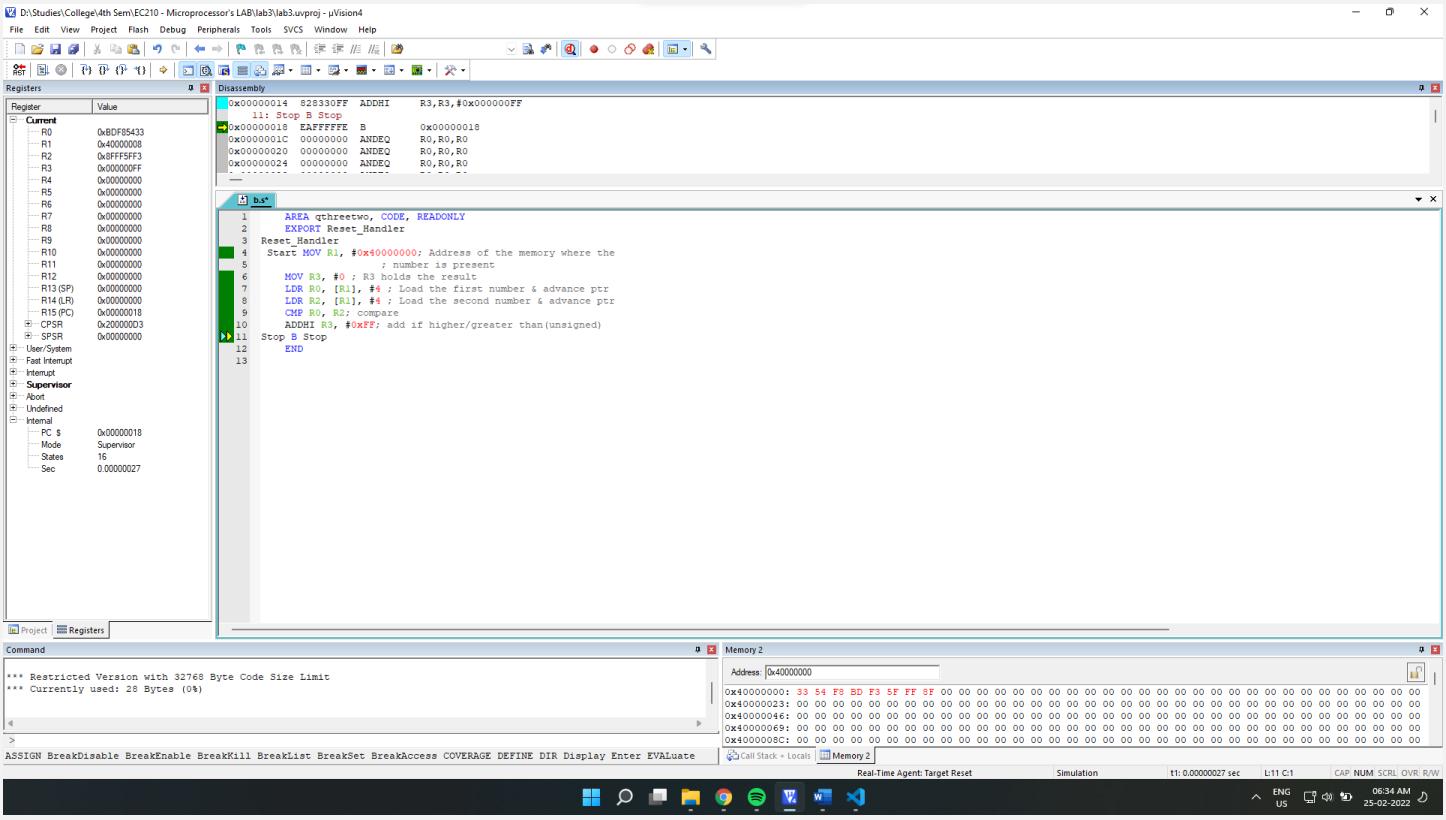
Initial Memory:



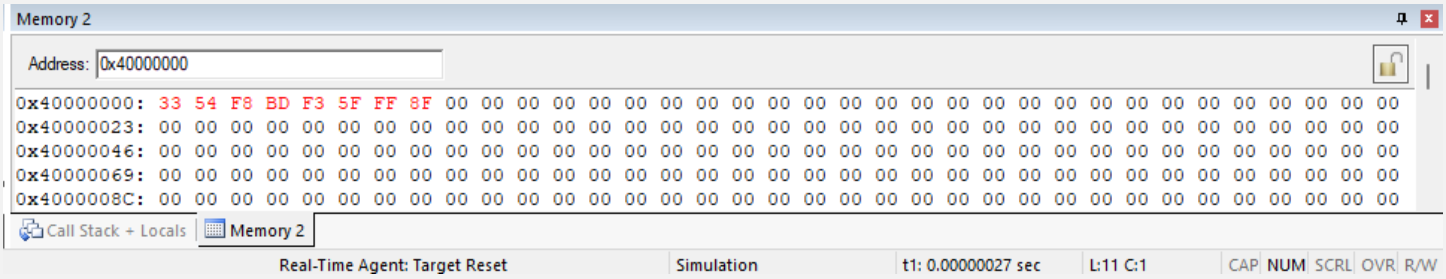
Setup:



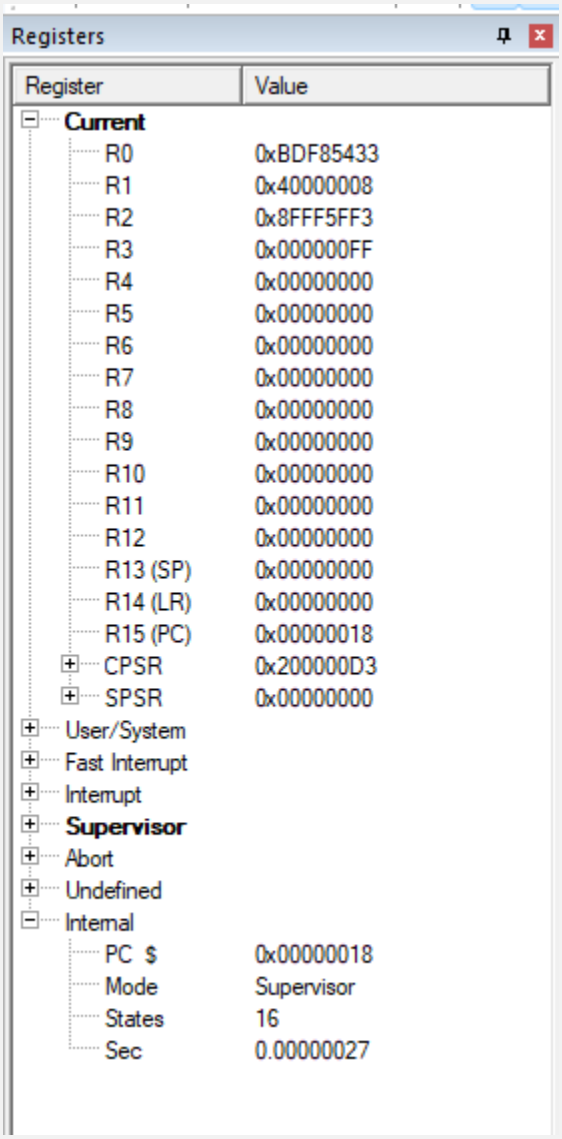
After Execution:



Final Output:



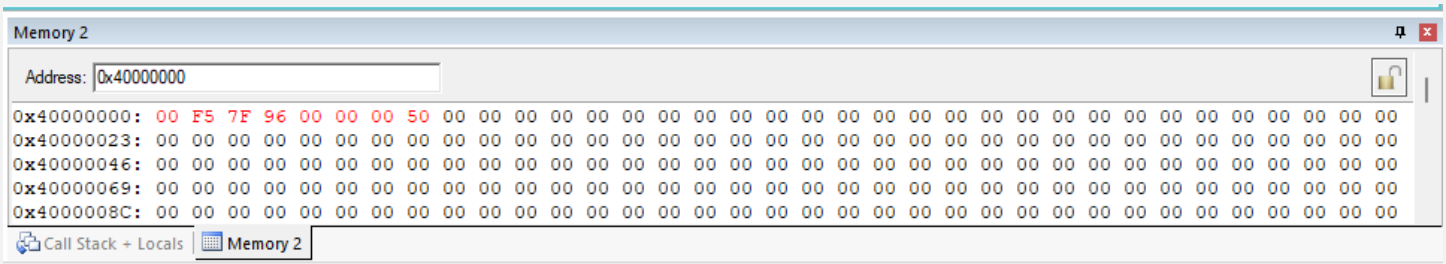
Final Register Values:



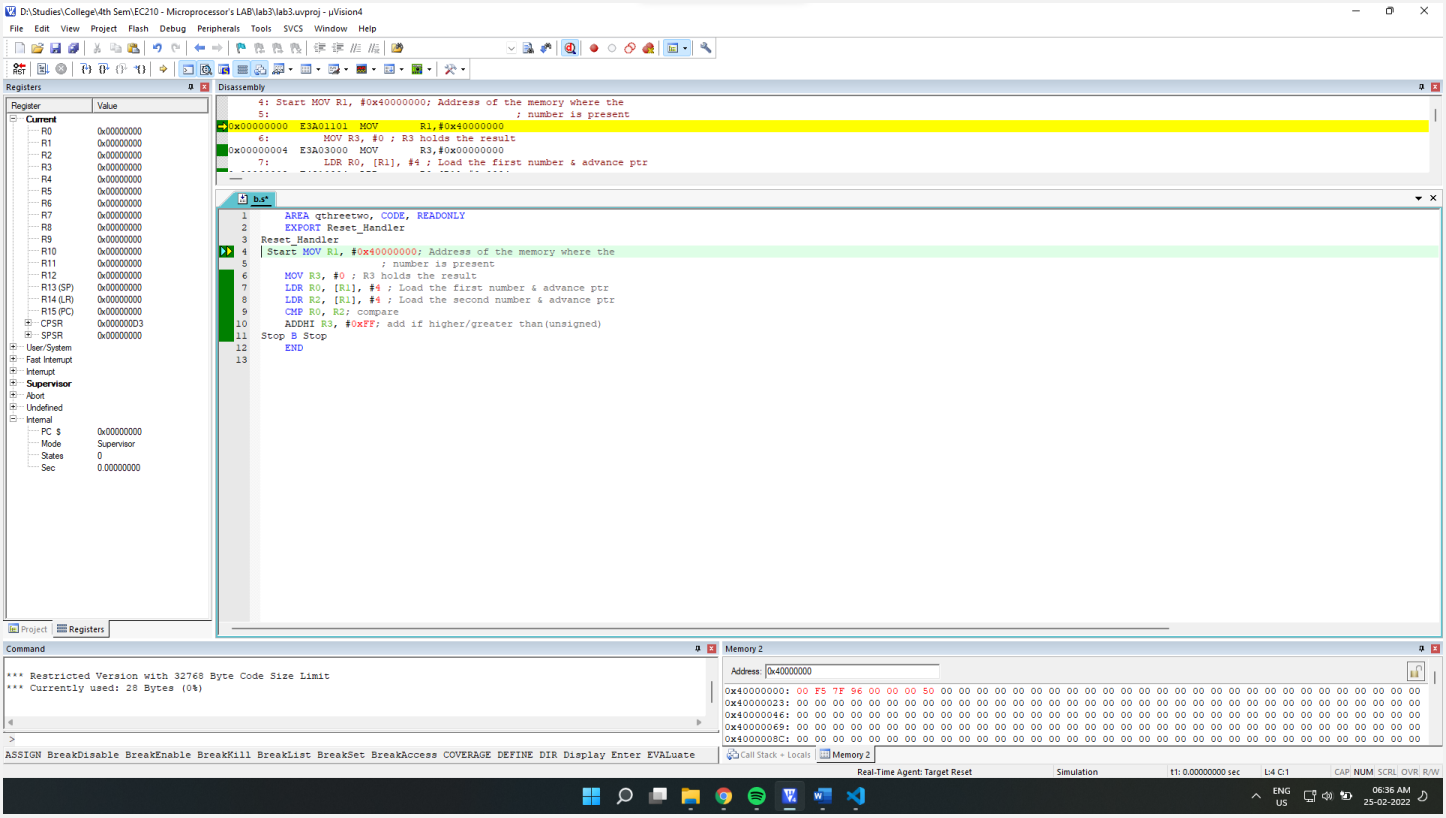
No1= 0x967ff500 and No2 =0x50000000

Debugging:

Initial Memory:

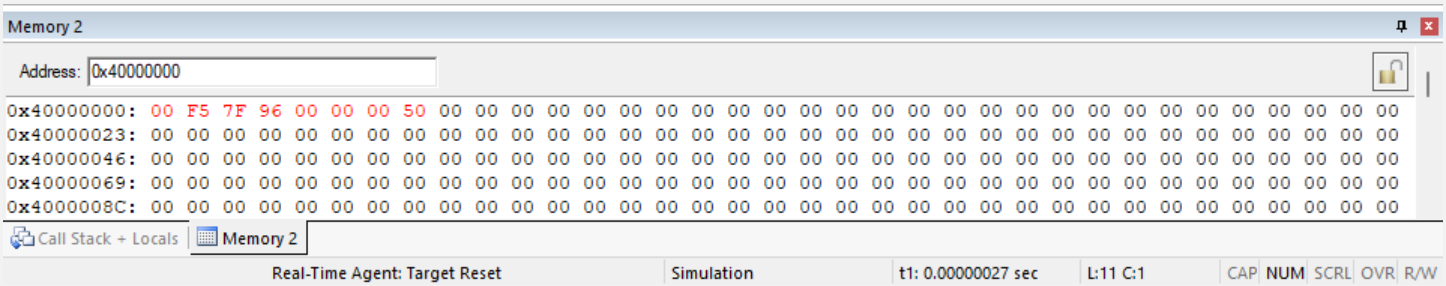


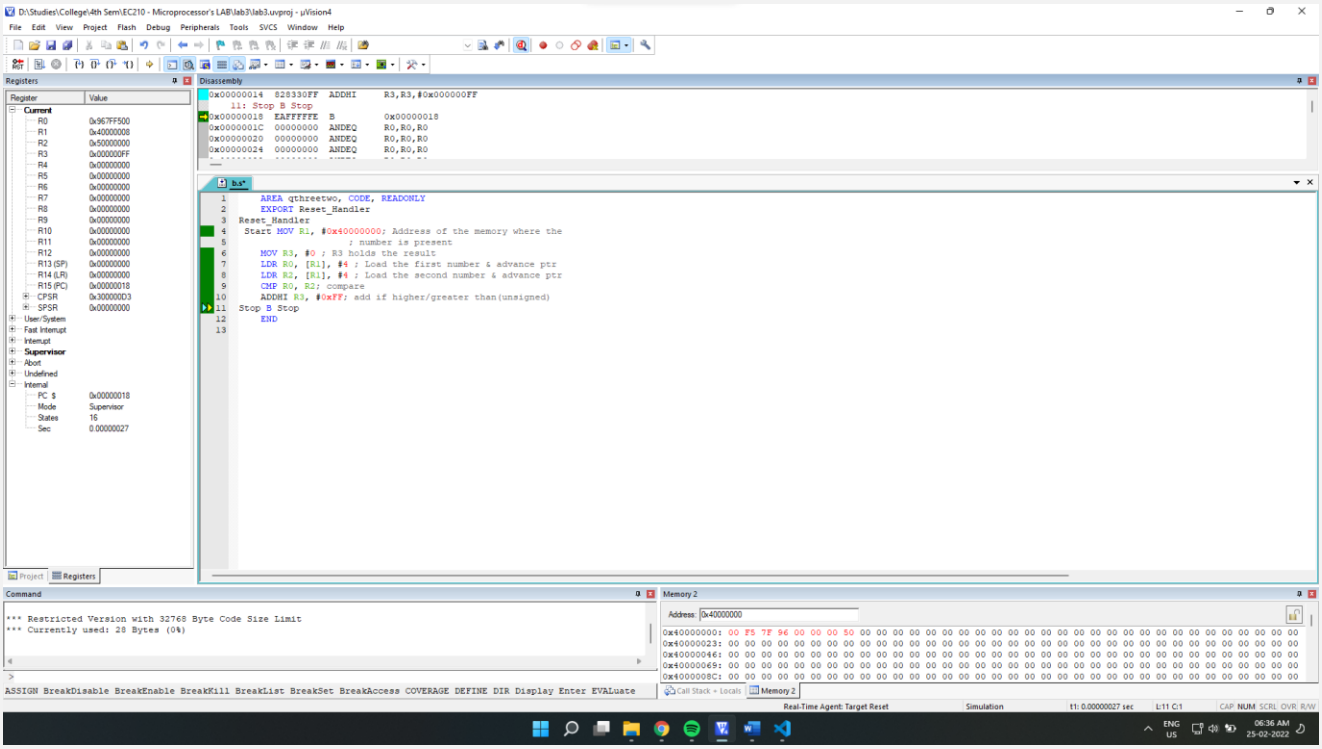
Setup:



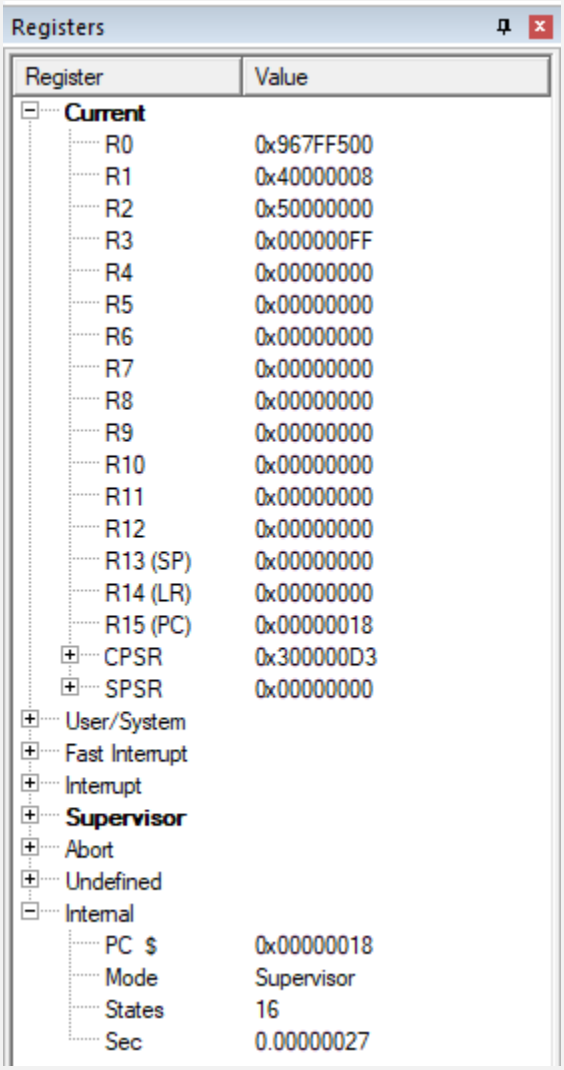
After Execution:

Final Output:





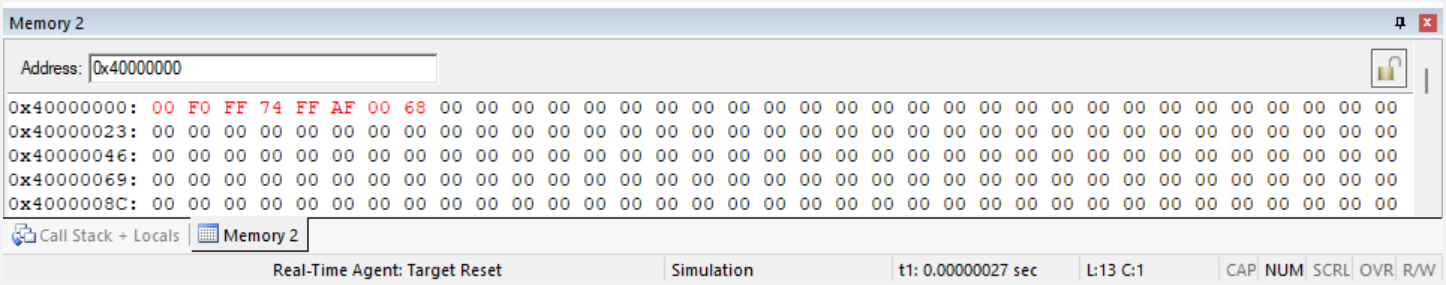
Final Register Values:



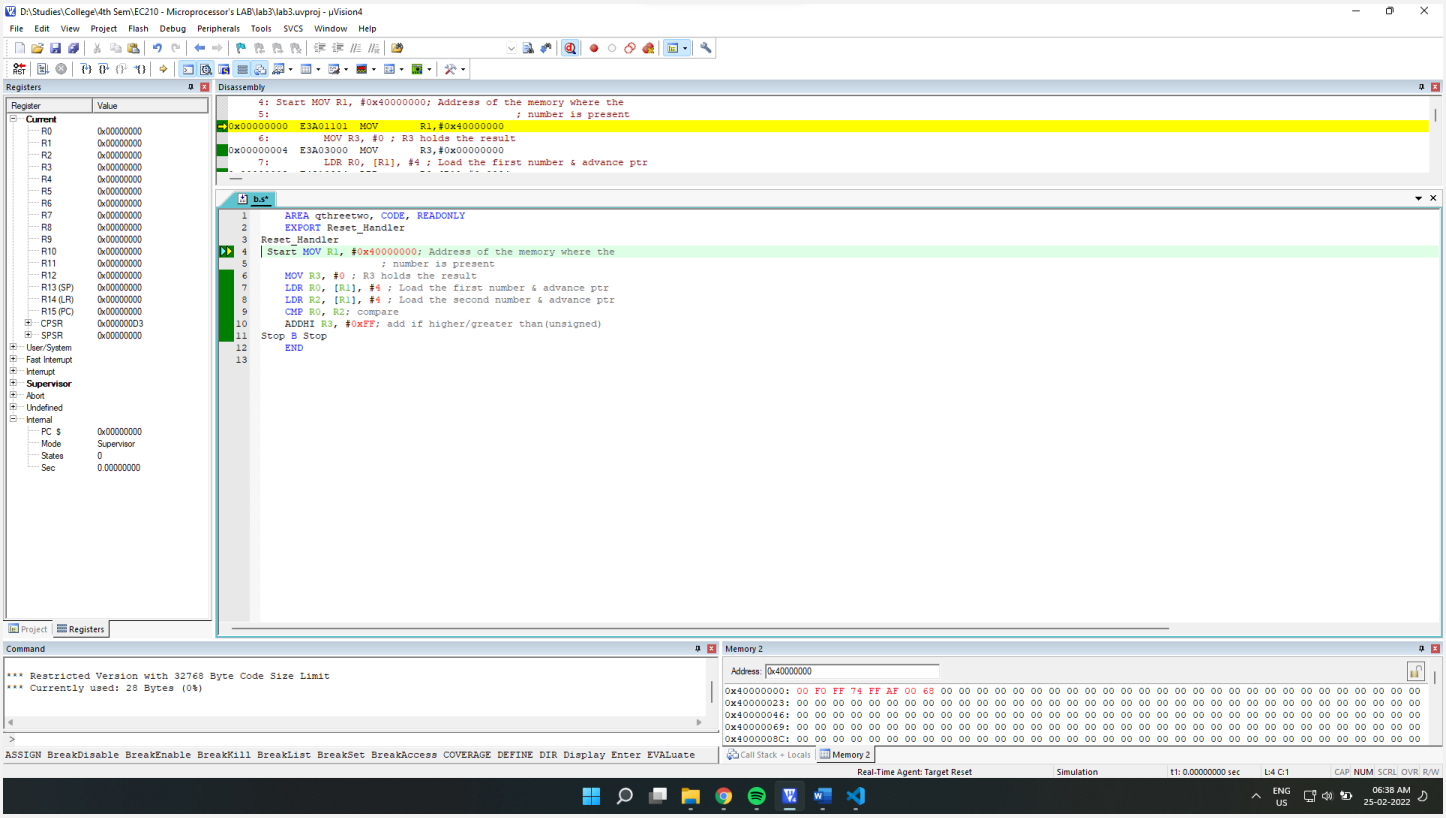
No1 =0x74FFF000 and No2 =0x6800AFFF

Debugging:

Initial Memory:

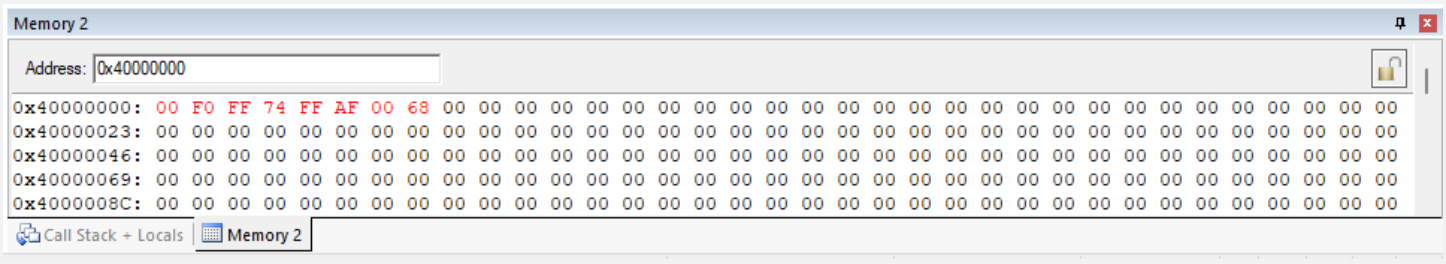


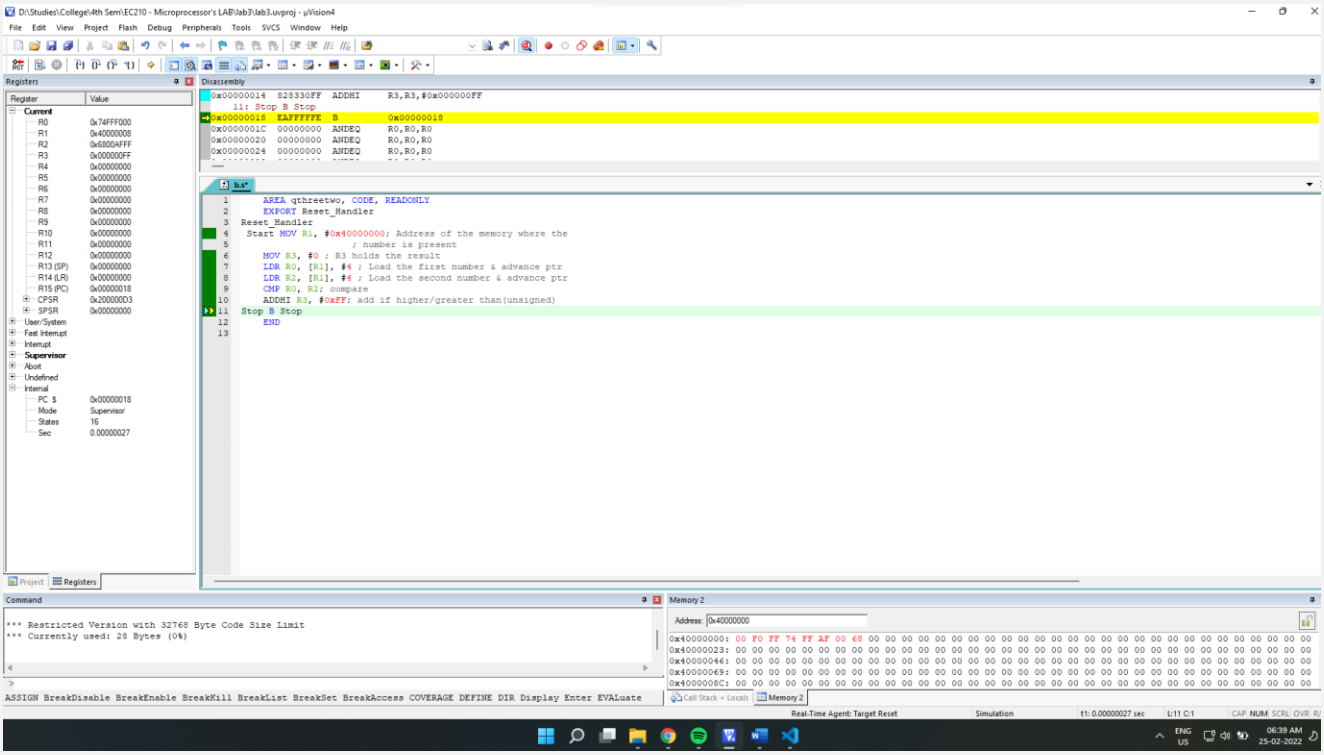
Setup:



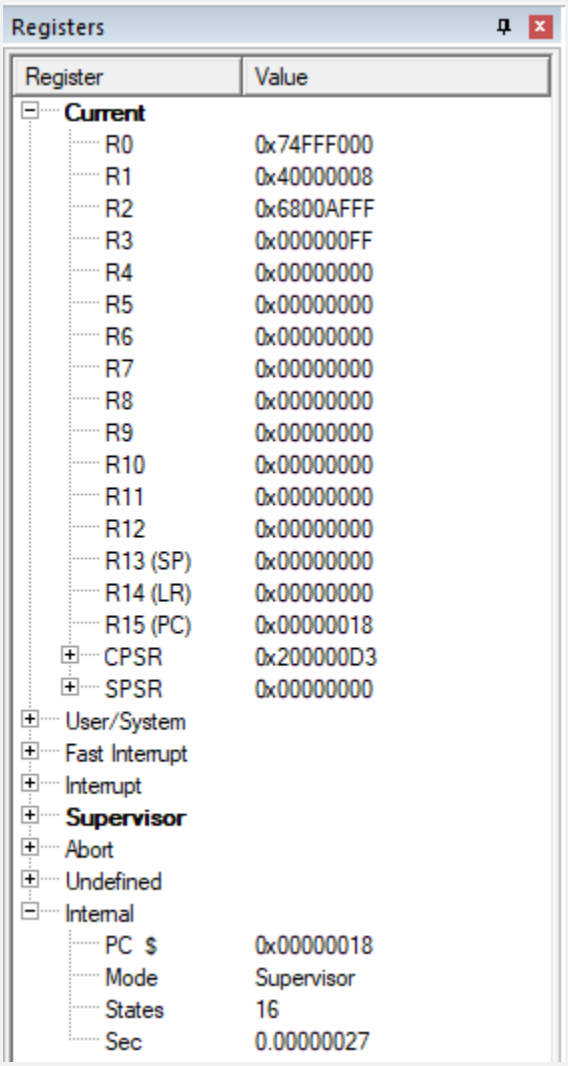
After Execution:

Final Output:





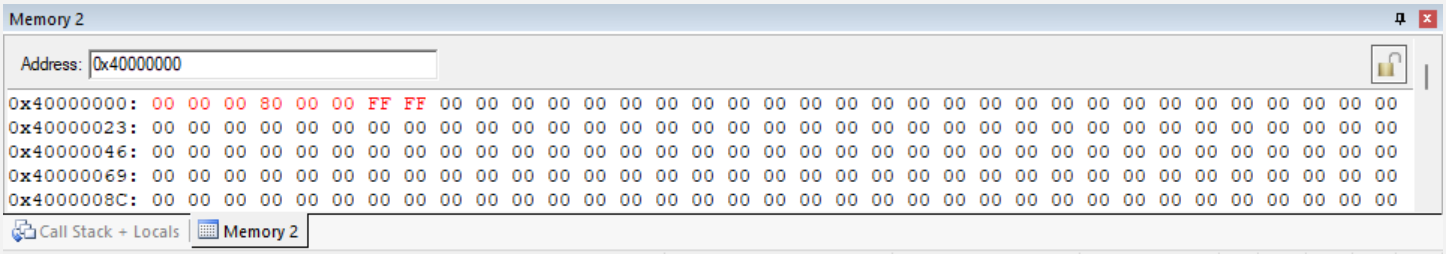
Final Register Values:



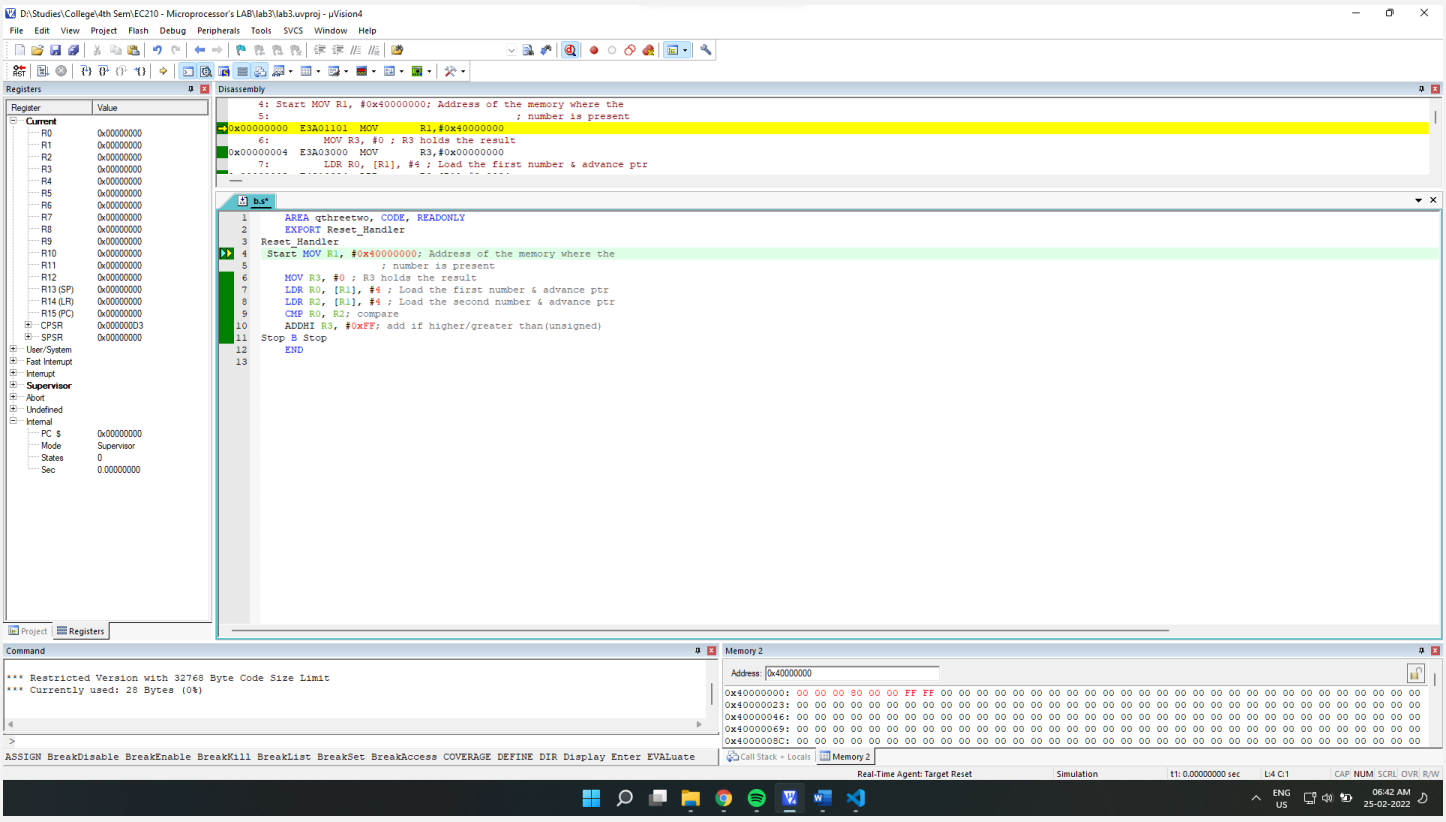
No1 =0x80000000 and No2 =0xFFFF0000

Debugging:

Initial Memory:

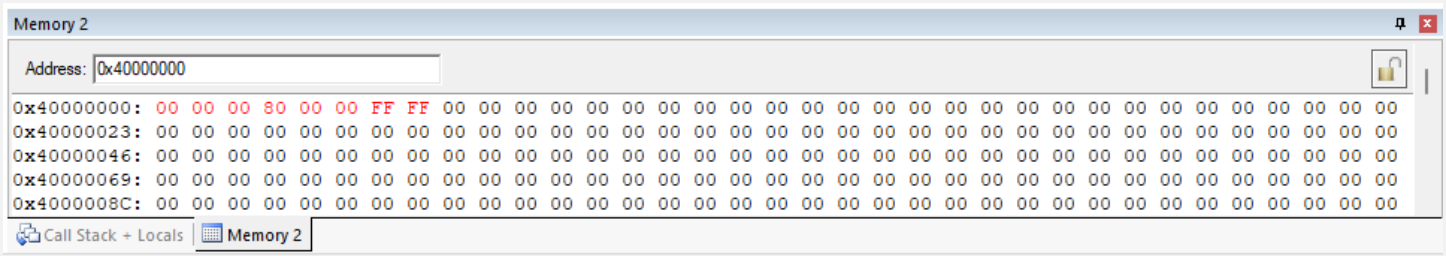


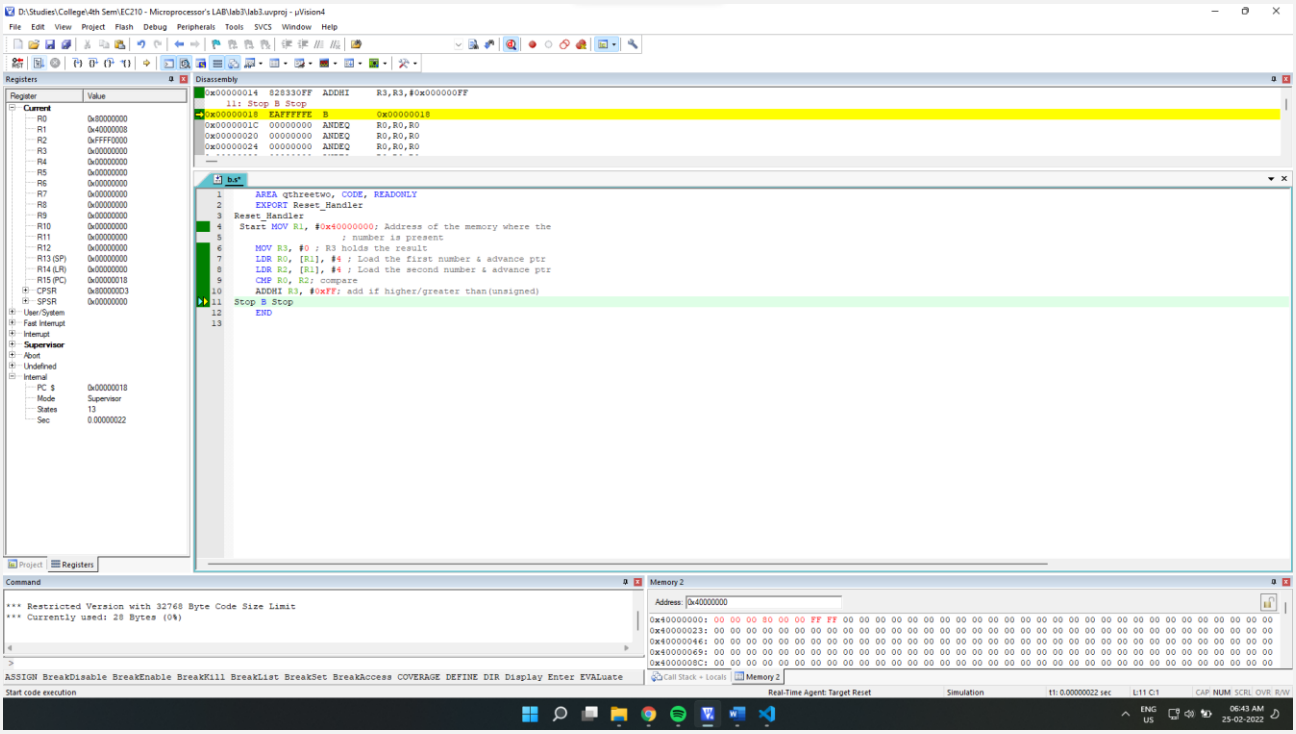
Setup:



After Execution:

Final Output:





Final Register Values:

Registers

Register	Value
Current	
R0	0x80000000
R1	0x40000008
R2	0xFFFF0000
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)	0x00000018
CPSR	0x800000D3
N	1
Z	0
C	0
V	0
I	1
F	1
T	0
M	0x13
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC $	0x00000018
Mode	Supervisor
States	13
Sec	0.00000022