**EC-210**

**Microprocessors Lab**

**LAB-7**



Utkarsh Mahajan 201EC164

ARNAV RAJ 201EC109

Objective: To understand division and code conversion operations.

**Exercise:**

**7.1]** Write a program to divide

(a) 32 bit number by 16 bit number.

->

Repeated subtraction:

Source Code:

    AREA AllocSpace, DATA,NOINIT,READWRITE

*;Space for storing result*

*;Utkarsh && Arnav Lab6*

c SPACE 8*;*

*;Actual Code*

    AREA hmm, CODE, READWRITE

    EXPORT Reset\_Handler

Reset\_Handler

*;address to the input number*

    LDR R1, =a*;*

    LDR R2, =bi*;*

*;address to the result*

    LDR R3, =c *;*

    LDR R4, [R1]*; load a*

    LDR R5, [R2]*; load b*

    MOV R6, #0*;*

div CMP R4, R5*; ? a < b*

    BCC done*; J if C clear (a<b)*

    SUB R4, R4, R5*; a=a-b;*

    ADD R6, R6, #1*; Q = Q+1;*

    B   div*; J to div for loop*

done STR R6, [R3]*; storing quotient*

    STR R4, [R3, #4]*; storing remainder*

stop BAL stop

*; input\_number*

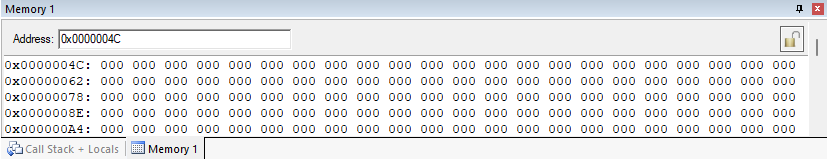
a   DCD 0xFA124025*;*

bi  DCD 0xCF85*;*

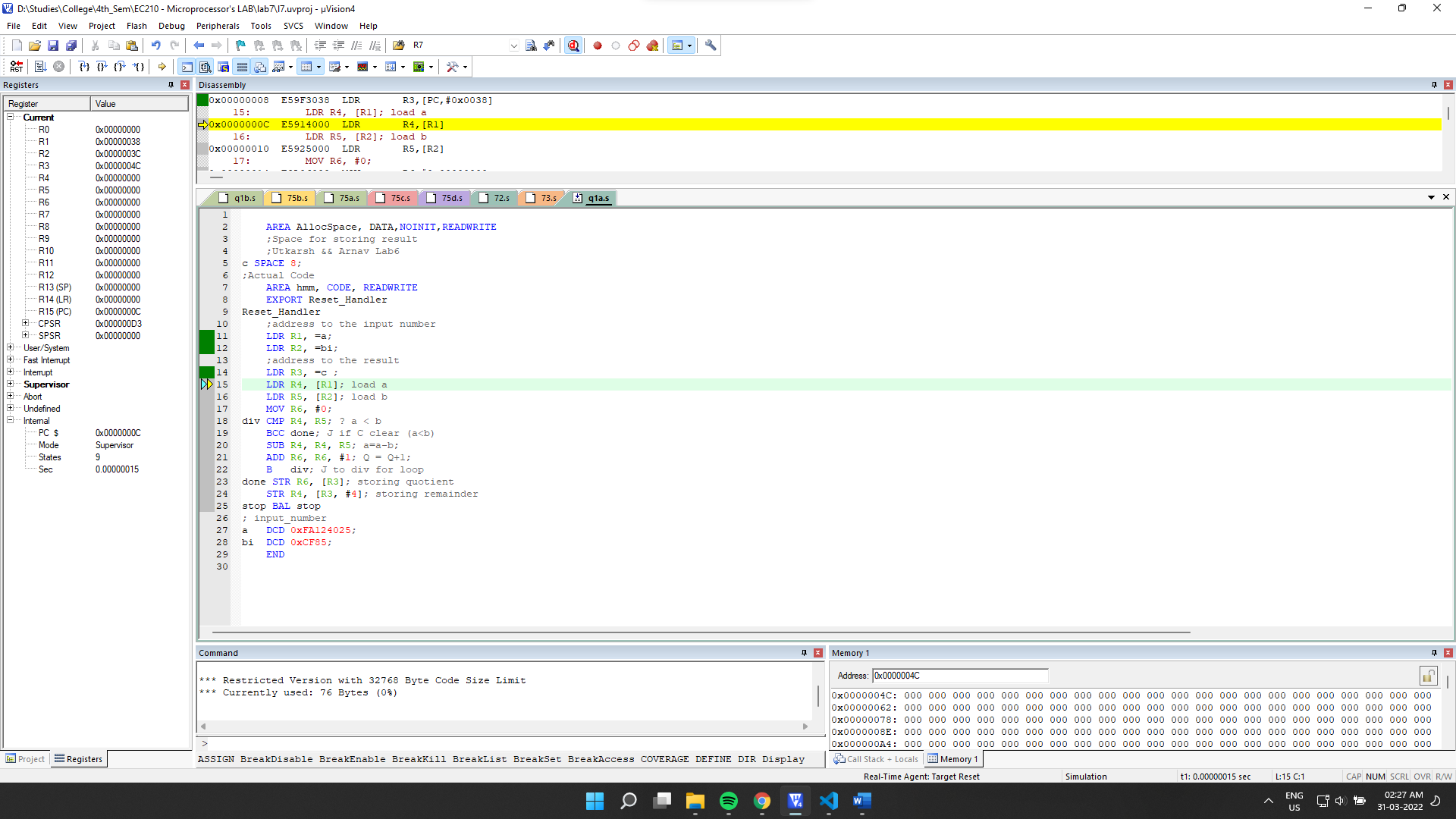
    END

Debugging:

Initial Memory: (after getting the address through register)



Setup:

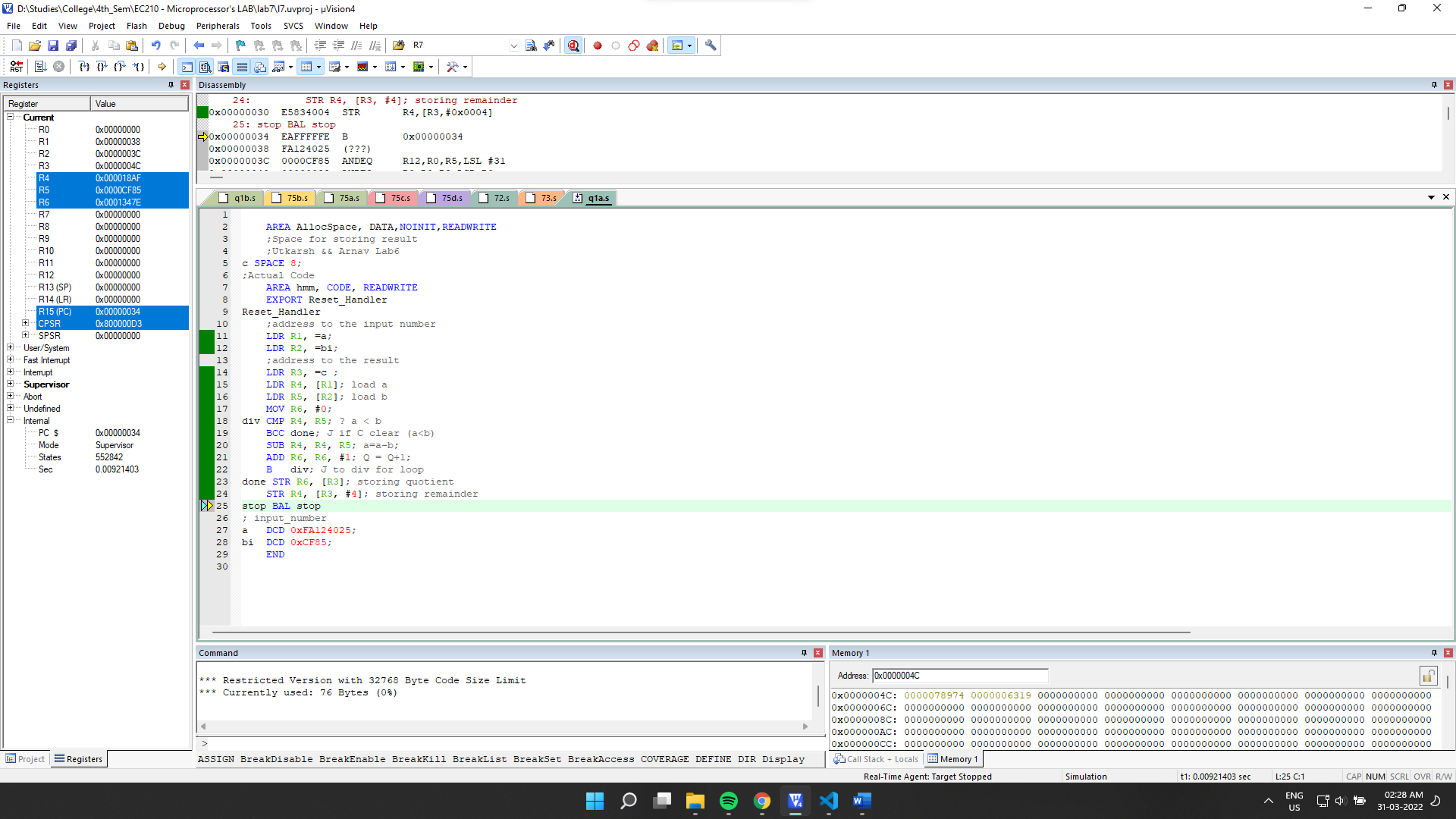


Inputs: a= 0xFA124025(4195500069), b=0xCF85(53125)

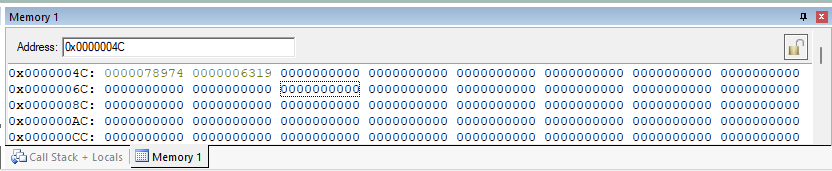
Expected results: 78974.1189459

Which is Q= 78974, R=6319

Final Output:



Final Memory: (in decimal)



Comparing the final memory with expected result, we can see that our code is correct.

**shift & subtract Method:**

Source Code:

    AREA AllocSpace, DATA,NOINIT,READWRITE

*;Space for storing result*

*;Utkarsh && Arnav Lab6*

c SPACE 8*;*

*;Actual Code*

    AREA hmm, CODE, READWRITE

    EXPORT Reset\_Handler

Reset\_Handler

*;address to the input number*

    LDR R1, =a*;*

    LDR R2, =bi*;*

*;address to the result*

    LDR R3, =c *;*

    LDR R4, [R1]*; load a*

    LDR R5, [R2]*; load b*

    MOV R6, #1*;*

div CMP R5, #0x80000000*;*

    CMPCC R5, R4*;*

    LSLCC R5, R5, #1*;*

    LSLCC R6, R6, #1*;*

    BCC div*;*

    MOV R7, #0*;*

div2 CMP R4, R5*;*

    SUBCS R4, R4, R5*;*

    ADDCS R7, R7, R6*;*

    LSRS R6, R6, #1*;*

    LSRNE R5, R5, #1*;*

    BNE div2*;*

    STR R7, [R3],#4*;*

    STR R5, [R3]*;*

stop BAL stop

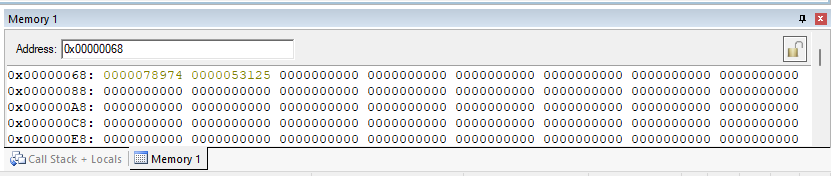
*; input\_number*

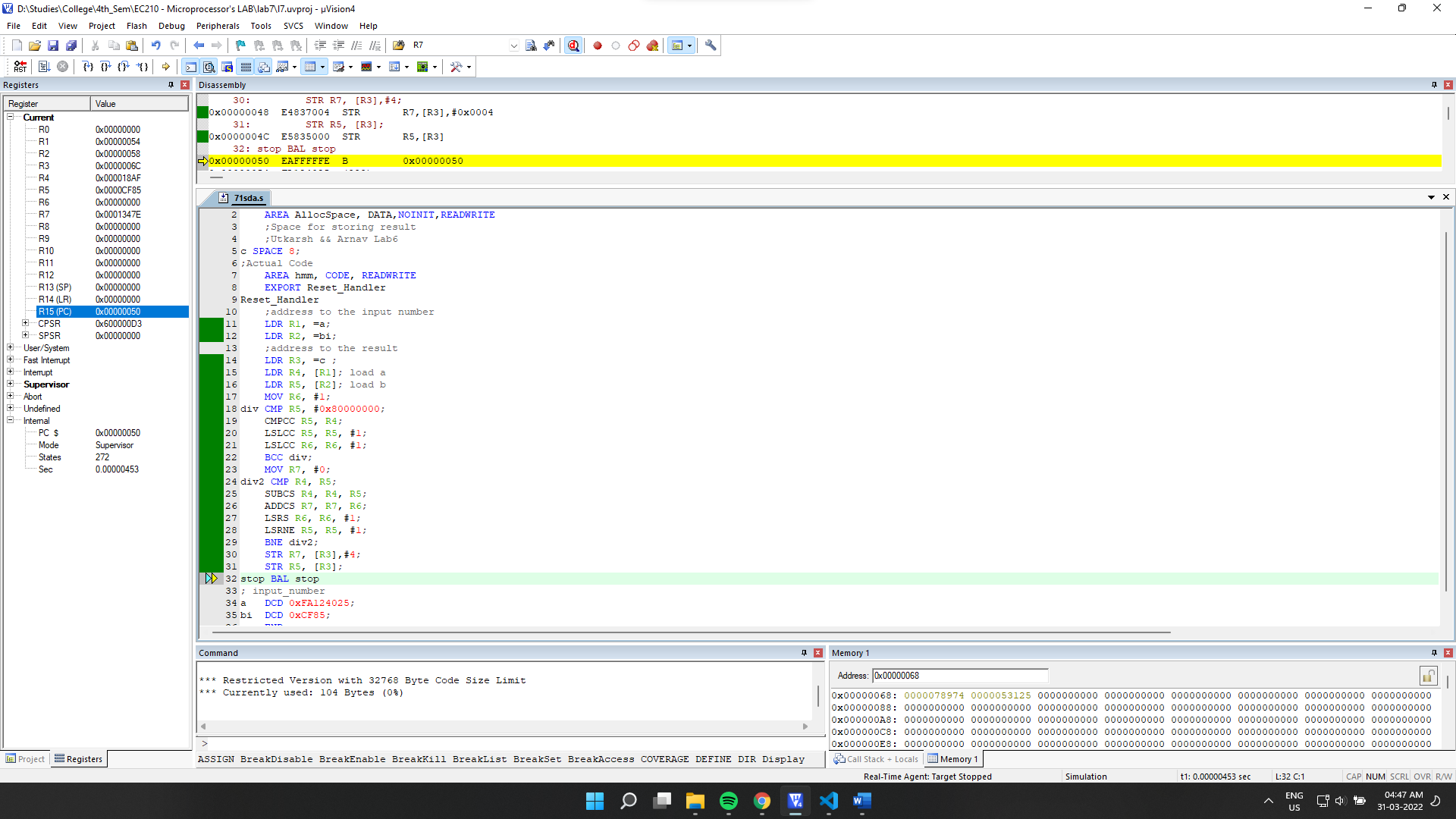
a   DCD 0xFA124025*;*

bi  DCD 0xCF85*;*

    END

Final Output:





It matches the final output.

**Execution Time:**

Subtraction method: 0.00000015sec

Shift and divide: 0.00000453sec

For smaller values, shift and divide seems to take more time.

(b) 64 bit number by 32 bit number

->

Repeated subtraction:

Source Code:

    AREA AllocSpace, DATA,NOINIT,READWRITE

*;Space for storing result*

*;Utkarsh && Arnav Lab6*

c SPACE 8*;*

*;Actual Code*

    AREA hmm, CODE, READWRITE

    EXPORT Reset\_Handler

Reset\_Handler

*;address to the input number*

    LDR R1, =a*;*

    LDR R2, =bi*;*

*;address to the result*

    LDR R3, =c *;*

    LDR R4, [R1]*; load a\_low*

    LDR R5, [R1, #4]*; load a\_high*

    LDR R6, [R2]*; load b*

    MOV R7, #0*; Q=0;*

div CMP R5, #0*;div process while a\_high>0*

    BEQ dlow*;*

    SUBS R4, R4, R6*; R4=R4-R6*

    SUBCC R5, R5, #1*; Sub borrow(if) from a\_high.*

    ADD R7, R7, #1*; Q = Q+1;*

    B   div

dlow CMP R4, R6*; Now Normal Subtraction process*

    BCC done*; for division like before*

    SUB R4, R4, R6*;  R4=R4-R6*

    ADD R7, R7, #1*; Q=Q+1*

    B   dlow*;*

done STR R7, [R3]*; storing quotient*

    STR R4, [R3, #4]*; storing remainder*

stop BAL stop

*; input\_number a=0x703D4444FA124025*

*; b= 0xFEFF4ED6*

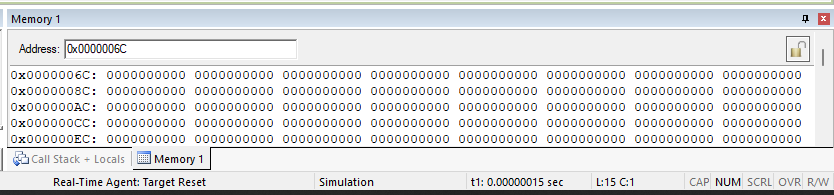
a   DCD 0xFA124025, 0x703D4444*;*

bi  DCD 0xFEFF4ED6*;*

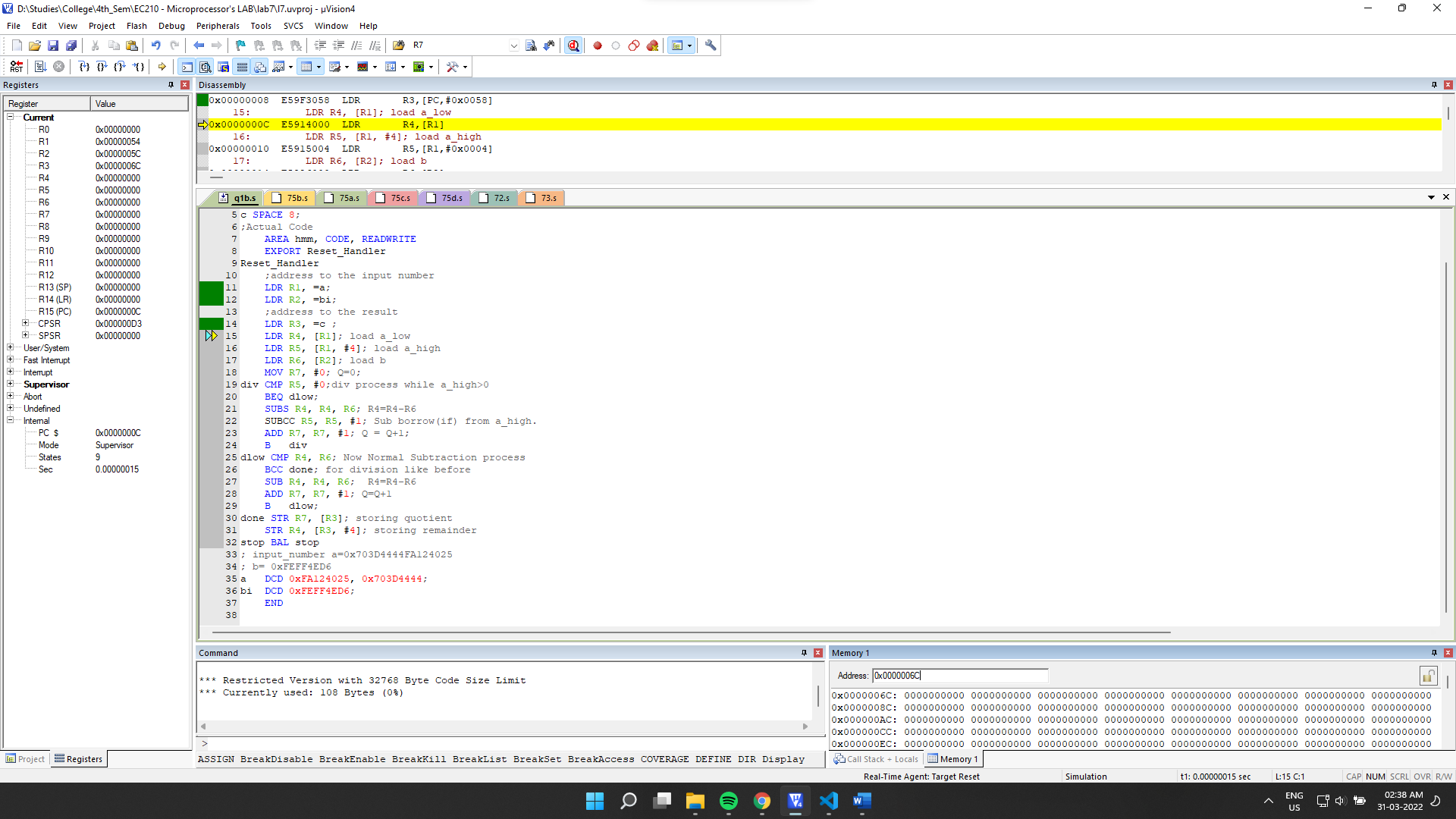
    END

Debugging:

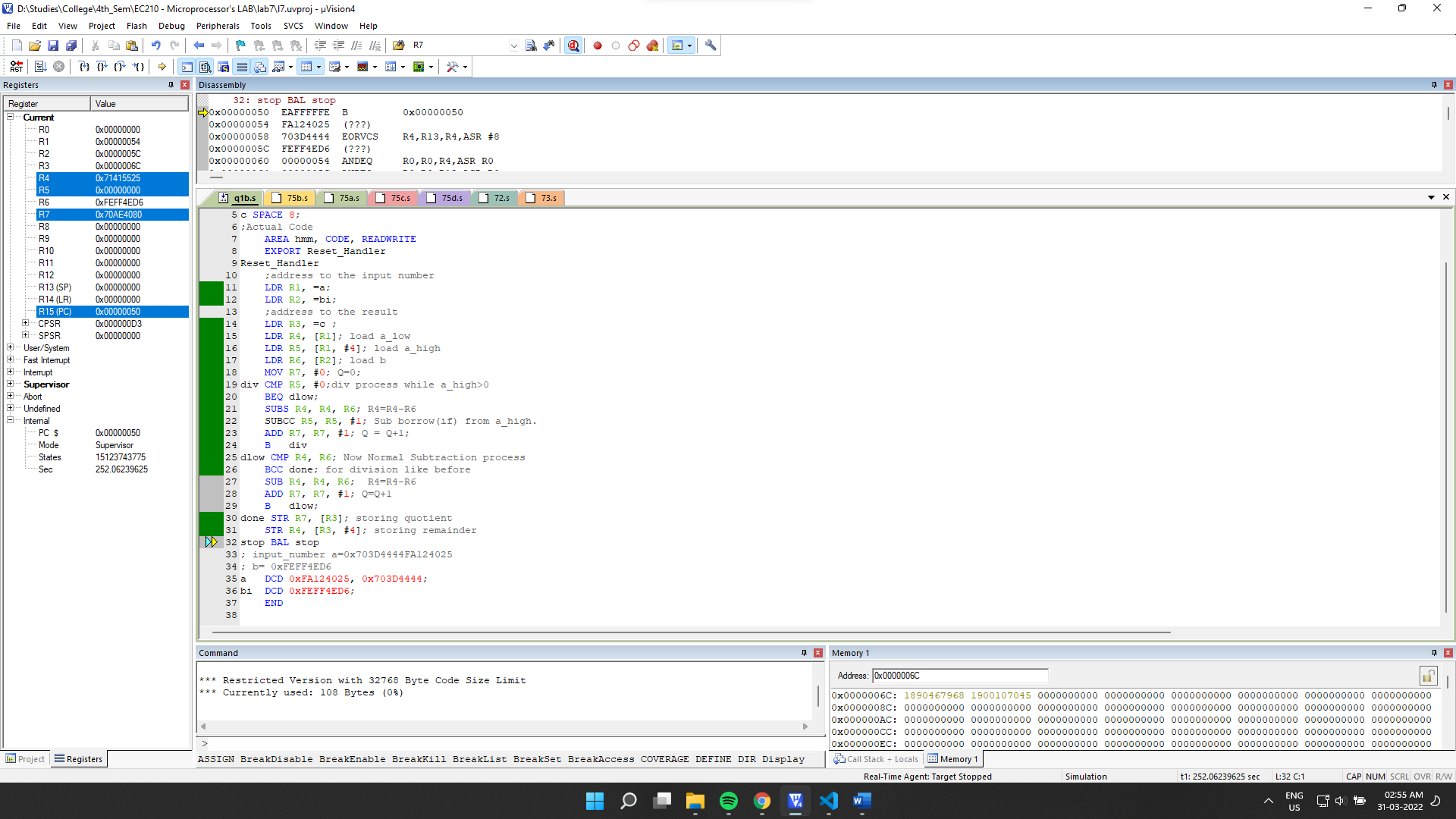
Initial Memory: (after getting the address through register)



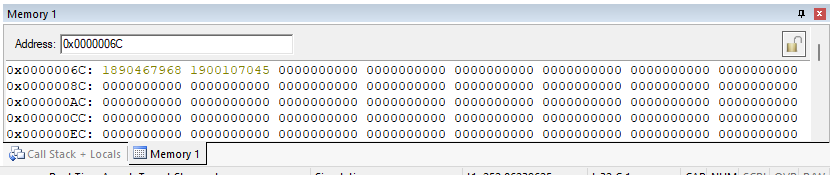
Setup:



Final Output:

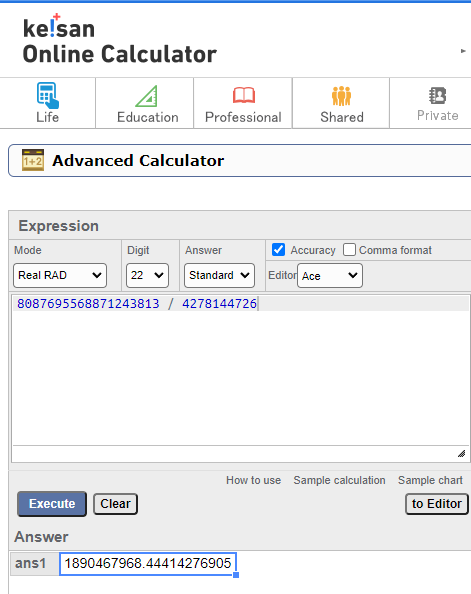


Final Memory:



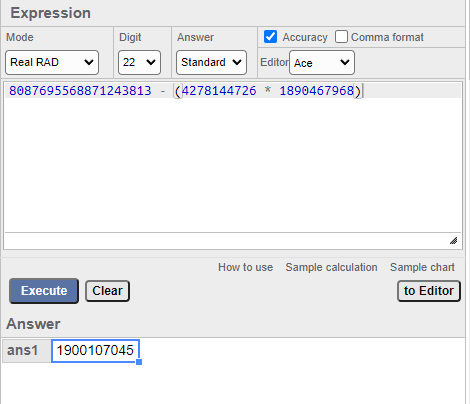
0x703D4444FA124025/ 0xFEFF4ED6 = 8087695568871243813/ 4278144726

Comparing the results with an online precise calculator: (<https://keisan.casio.com/calculator>)



We can see that our quotient matches.

Now for the remainder:



We can see that even it matches. Hence our result is correct.

**7.2]** Write a program that takes a 16 bit Hex number and converts it into its BCD equivalent.

->

Source Code:

    AREA AllocSpace, DATA,NOINIT,READWRITE

*;Space for result*

array SPACE 1024*;*

*;Actual Code*

    AREA hmm, CODE, READWRITE

    EXPORT Reset\_Handler

Reset\_Handler

*;address to the input number*

    LDR R1, =inputnumber *;*

*;address for result*

    LDR R2, =array*;*

    MOV R12, R2*;*

*; load the  input number from memory*

    LDR R4, [R1]*;*

*; Getting the last bit*

    MOV R5, R4*; R5 = R4*

    MOV R6, #0*;*

    MOV R7, #0*; for storing no of nos.*

*;dividing and finding nth digit.*

*;division by the use of subtracting*

*; until a remainder is found.*

*; where remainder < 10*

div SUB R5, R5, #10*; subtracting by 10*

    CMP R5, #10*; checking if remainder < 10*

*;incrementing iterator to store quotient*

    ADD R6, R6, #1*;*

*; if remainder >=10 then loop to sub further.*

    BPL div

    ADD R7, #1*; incrementing nos.*

    STRB R5, [R2], #1*; storing digits*

*;comparing quotient and 10.*

    CMP R6, #10*;*

*;if quotient >10 then store quotient in remainder*

    MOVPL R5, R6*;*

*; && reset quotient*

    MOVPL R6, #0*;*

*; && loop again*

    BPL div*;*

*; check if quotient is zero*

    CMP R6, #0*;*

*; increment no of digits counter if no*

    ADDNE R7, #1*;*

*; &&  store it in R2 if no.*

    STRBNE R6, [R2]*; storing last digit*

*;*

stop BAL stop

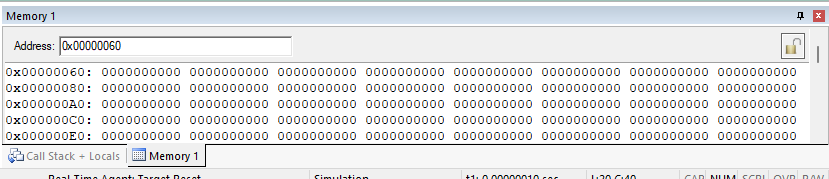
*; input\_number*

inputnumber DCD 0xFABC*;64188*

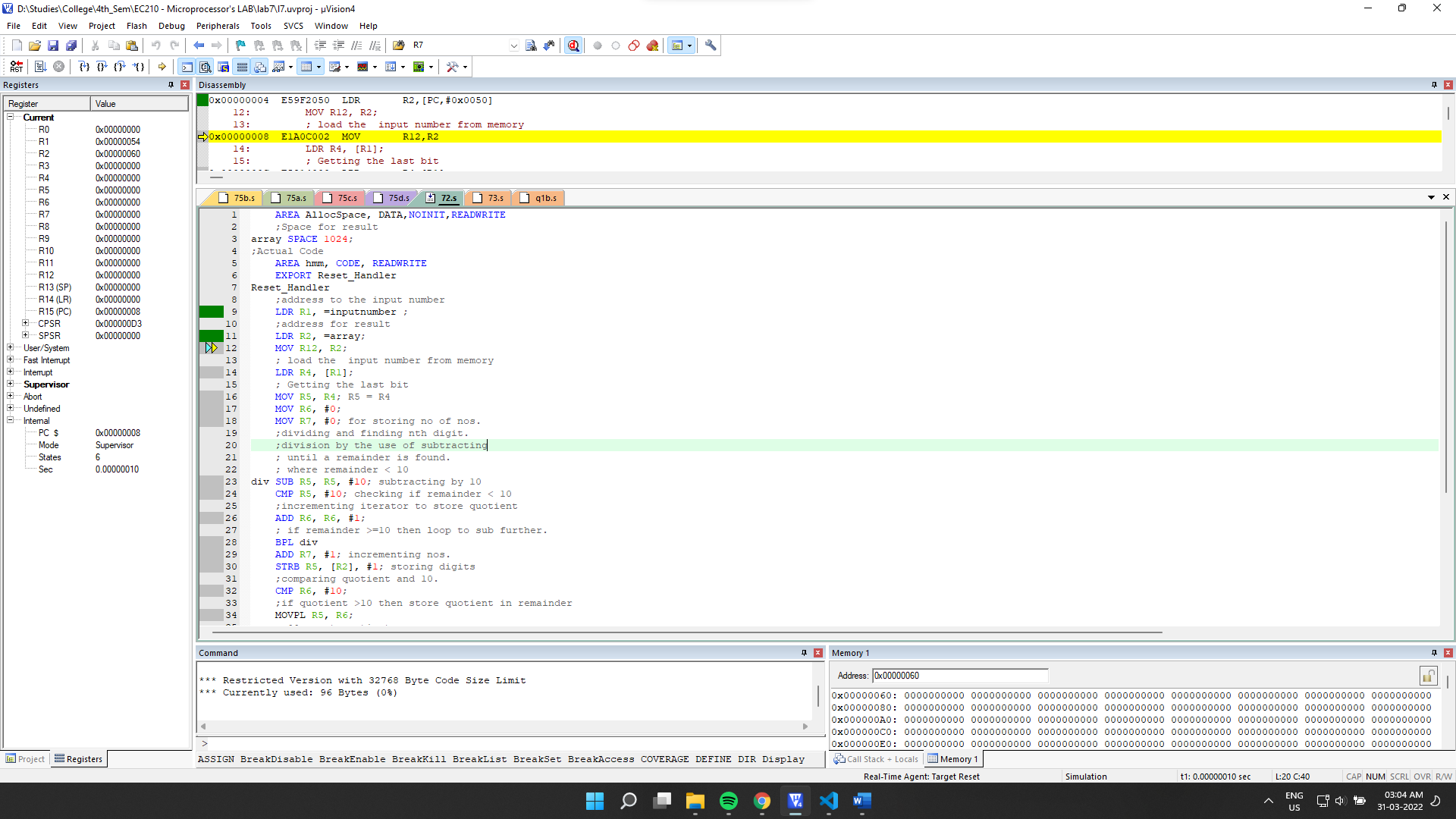
    END

Debugging:

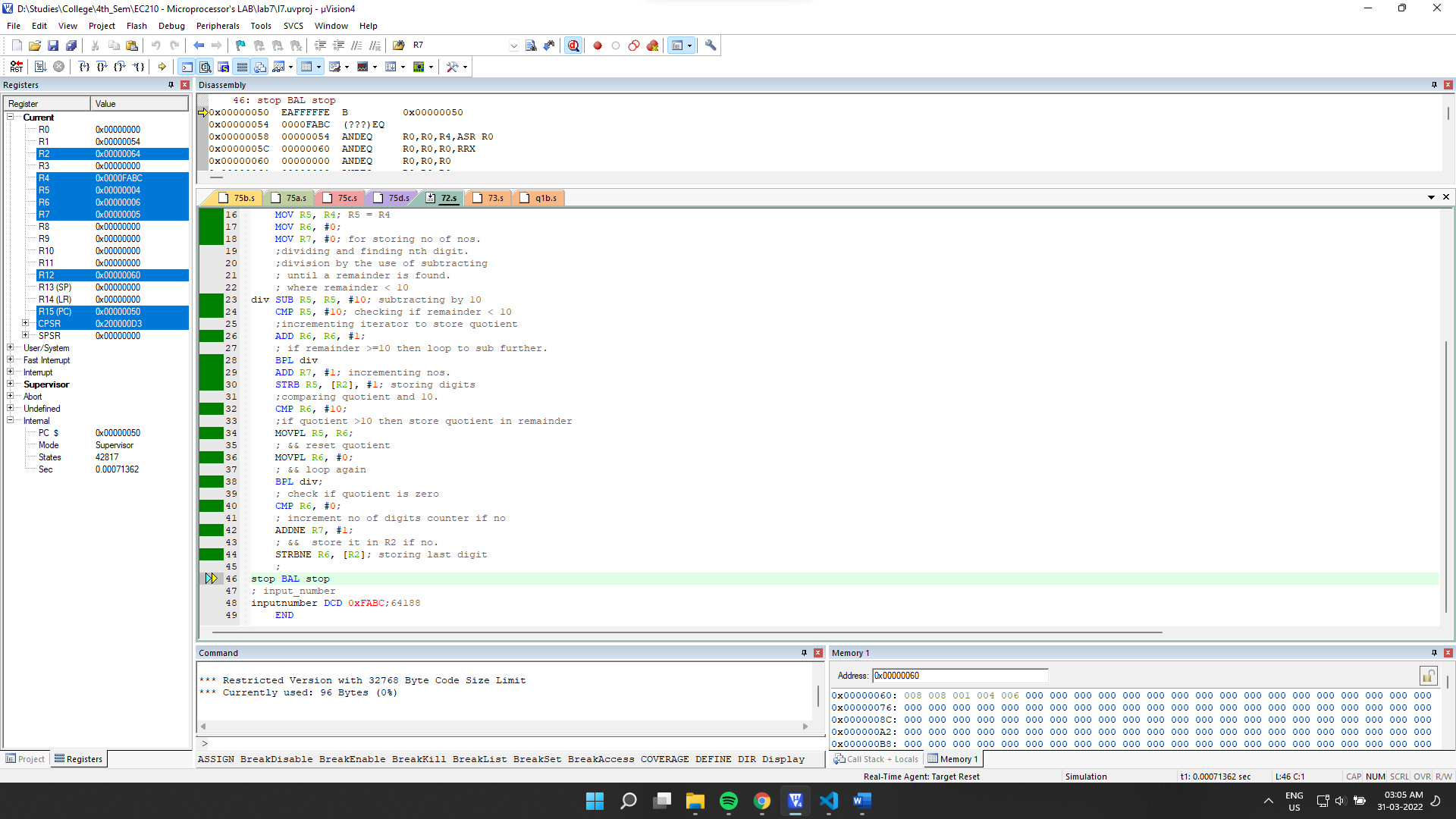
Initial Memory: (after getting the address through register)



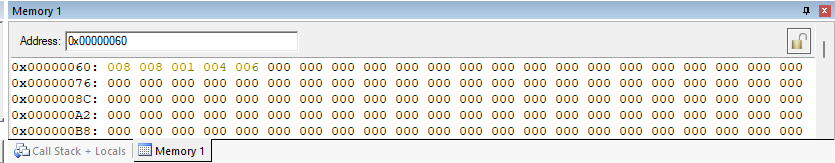
Setup:



Final Output:



Final Memory:



We can see that for our input 0xFABC (64188 in decimal) the BCD values stored matches.

7.3] Write a program that takes an 4 digit BCD number and converts it into is Hex equivalent.

->

Source Code:

    AREA AllocSpace, DATA,NOINIT,READWRITE

*;Space for storing result*

*;Utkarsh && Arnav Lab6*

bcd SPACE 4*;*

*;Actual Code*

    AREA hmm, CODE, READWRITE

    EXPORT Reset\_Handler

Reset\_Handler

*;address to the input number*

    LDR R1, =asc*;*

*;address to the result*

    LDR R3, =bcd *;*

    MOV R8, #1*; multiplier for nth digit*

    MOV R7, #0*;*

    MOV R10, #10*;*

    LDRB R5, [R1, #3]*; load digit*

    ADD R7, R7, R5*;*

    MUL R9, R8, R10*;*

    LDRB R5, [R1, #2]*; load digit*

    MUL R7, R10, R7*;*

    ADD R7, R7, R5*;*

    LDRB R5, [R1, #1]*; load digit*

    MUL R7, R10, R7*;*

    ADD R7, R7, R5*;*

    LDRB R5, [R1]*; load digit*

    MUL R7, R10, R7*;*

    ADD R7, R7, R5*;*

    STR R7, [R3]*; storing quotient*

stop BAL stop

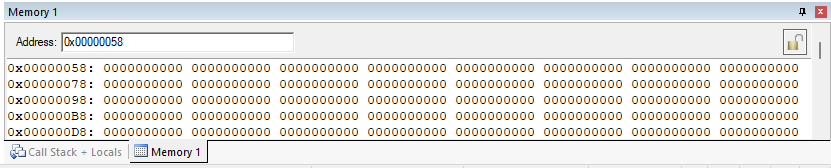
*; input\_number*

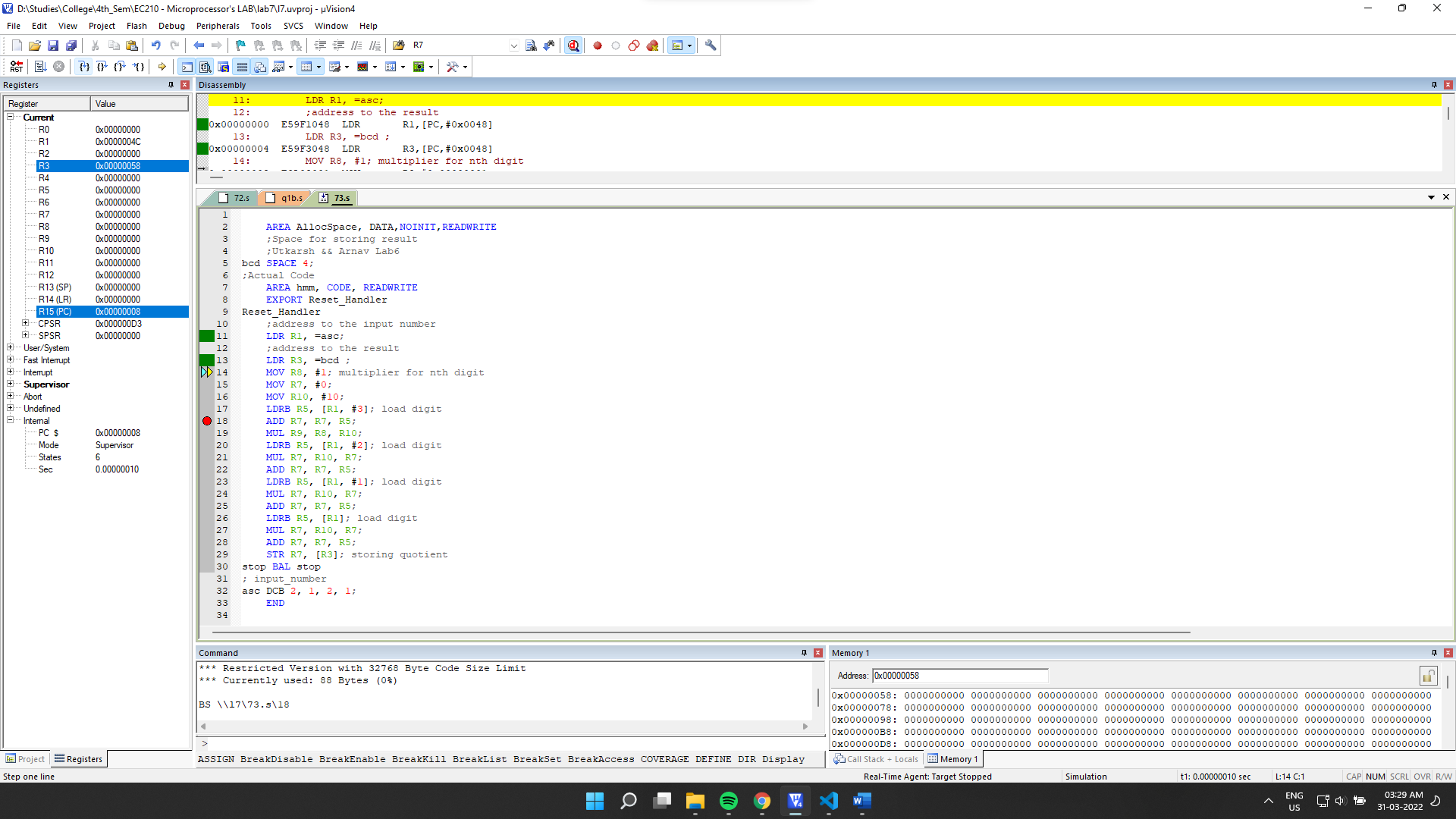
asc DCB 2, 1, 2, 1*;*

    END

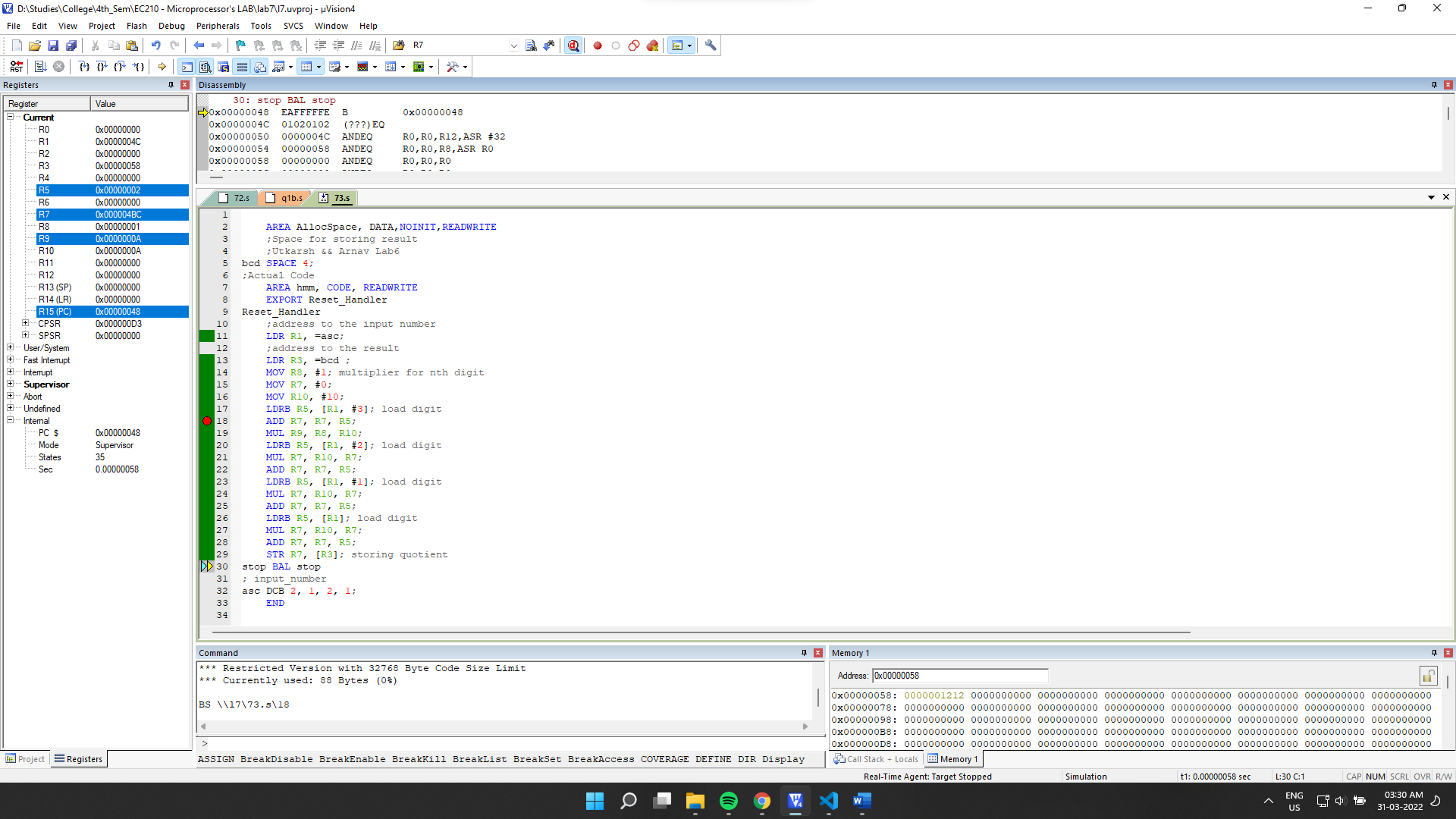
Debugging:

Initial Memory: (after getting the address through register)

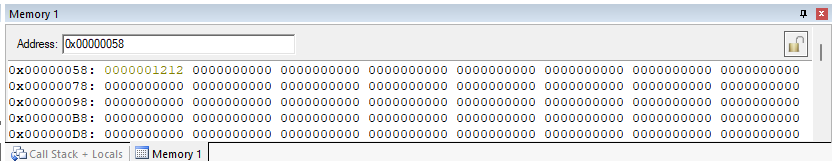
Setup:



Final Output:



Final Memory:



We can see that the output matches our input 1212 which was in BCD format.

7.4] Perform addition of two 8 digit BCD numbers and give the result in BCD.

->

Source Code:

    AREA AllocSpace, DATA,NOINIT,READWRITE

*;Utkarsh && Arnav Lab6*

*;Space for storing result*

z SPACE 9*;*

*;Actual Code*

    AREA hmm, CODE, READWRITE

    EXPORT Reset\_Handler

Reset\_Handler

*;address to the input number*

    LDR R1, =x*;*

    LDR R2, =y*;*

*;address to the result*

    LDR R3, =z *;*

    MOV R0, #0*;*

    MOV R7, #0*; carry\_bit xD*

loop CMP R0, #8*;*

    BEQ done*; jump if all bits done*

    LDRB R4, [R1], #1*; load bit, x[i]*

    LDRB R5, [R2], #1*; load bit, y[i]*

    ADD R6, R4, R5*; z[i] =x[i] + y[i]*

    ADD R6, R6, R7*;  z[i] += c[i-1]*

    MOV R7, #0*; c[i]=0*

    CMP R6, #9*; ? z[i]>9*

    SUBHI R6, R6, #10*; r6-10:-*

    MOVHI R7, #1*;   r7=1:-*

    STRB R6, [R3], #1*; z[i] -> mem[i]*

    ADD R0, R0, #1*; i=i+1;*

    B loop*; loop*

done STRB R7, [R3]*; storing carry*

stop BAL stop

*; input\_number*

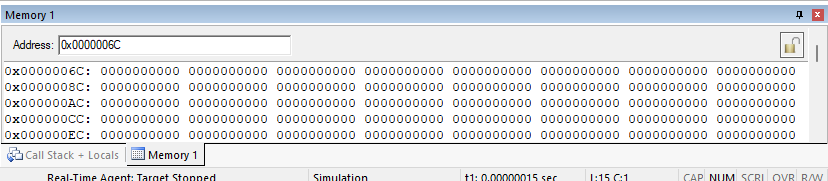
x   DCB 1, 8, 0, 1, 2, 3, 4, 2*;*

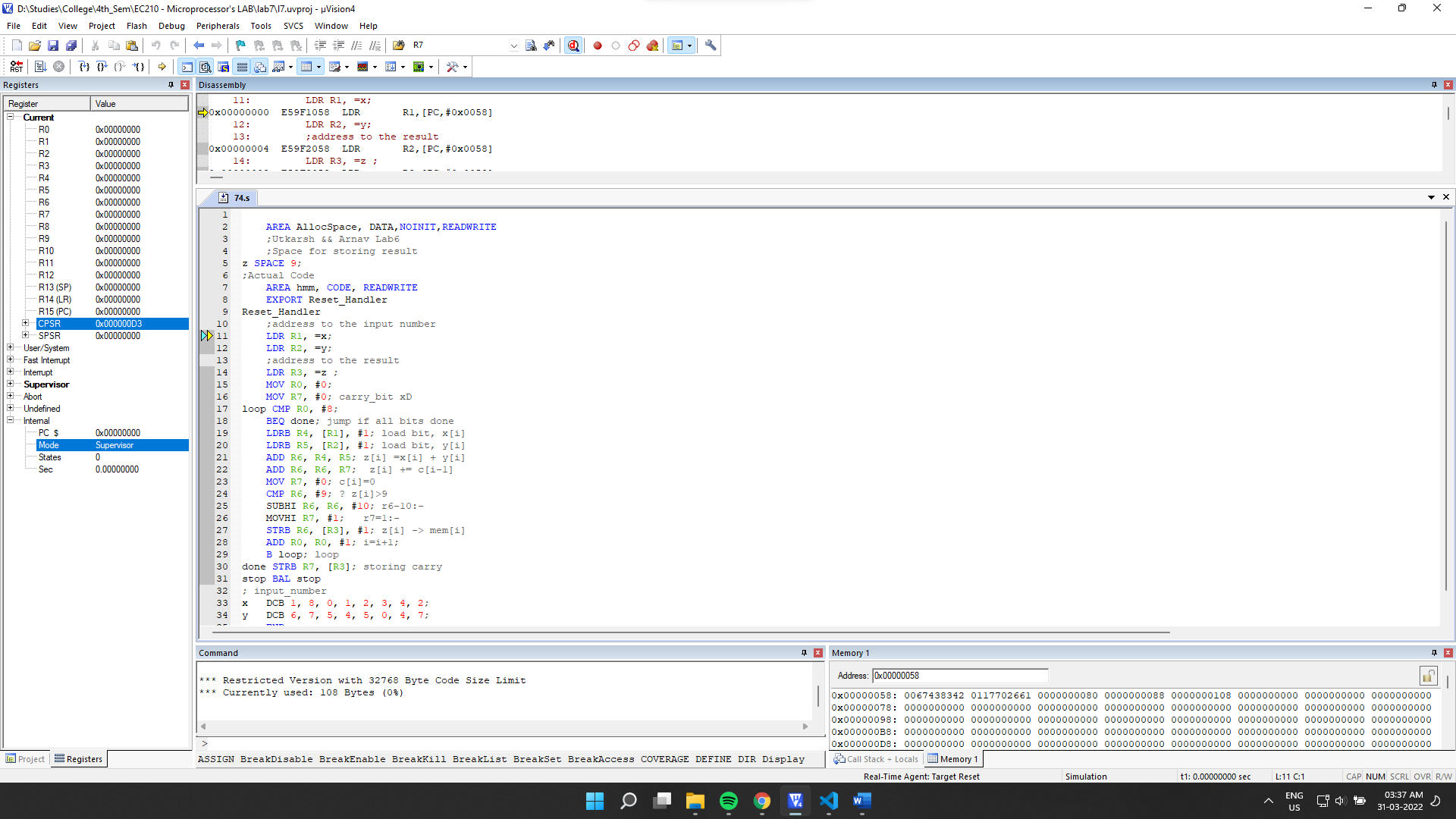
y   DCB 6, 7, 5, 4, 5, 0, 4, 7*;*

    END

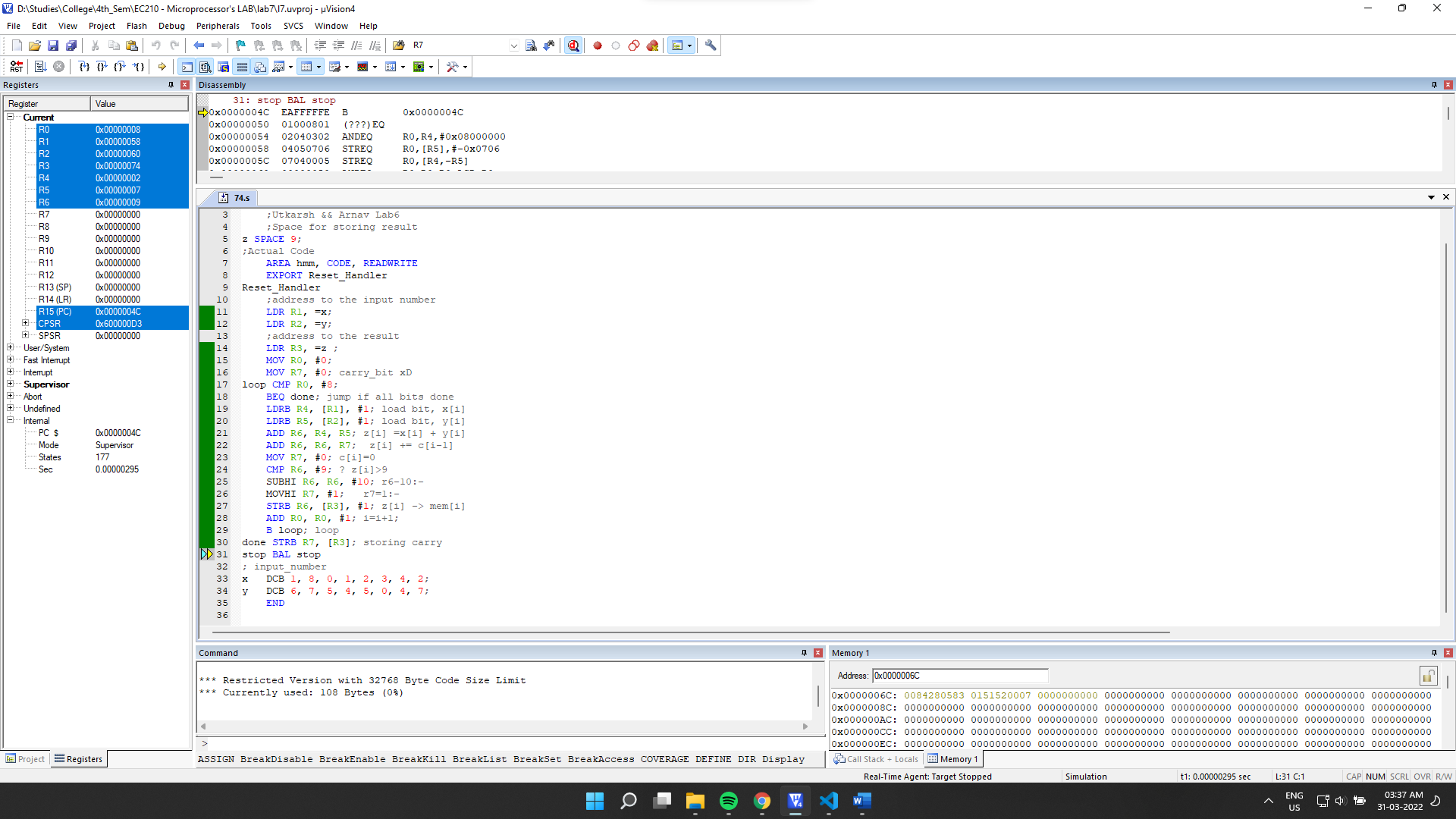
Debugging:

Initial Memory: (after getting the address through register)

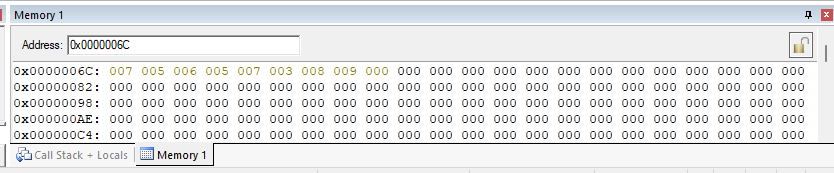


Setup: 

Final Output:



Final Memory:



We can see that for our input 24321081 + 74054576 (in BCD)

=98375657, which matches our output.

7.5] Write a program to convert a given (Note: Consider BCD and HX to 4 digits no)

(a) ASCII to BCD

->

Source Code:

    AREA AllocSpace, DATA,NOINIT,READWRITE

*;Space for storing result*

*;Utkarsh && Arnav Lab6*

bcd SPACE 4*;*

*;Actual Code*

    AREA hmm, CODE, READWRITE

    EXPORT Reset\_Handler

Reset\_Handler

*;address to the input number*

    LDR R1, =asc*;*

    LDR R2, =cf*;*

*;address to the result*

    LDR R3, =bcd *;*

    LDR R4, [R1]*; load ascii val*

    LDR R5, [R2]*; load conv factor*

    SUB R5, R4, R5*;*

    STR R5, [R3]*; storing quotient*

stop BAL stop

*; input\_number*

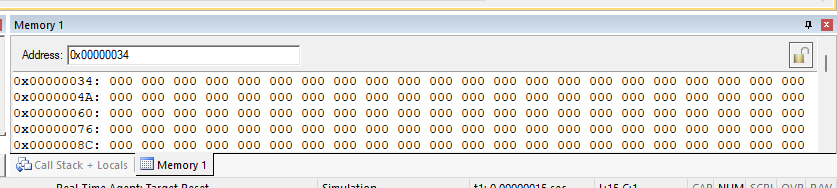
asc DCB "1212"*;*

cf  DCD 0x30303030*;*

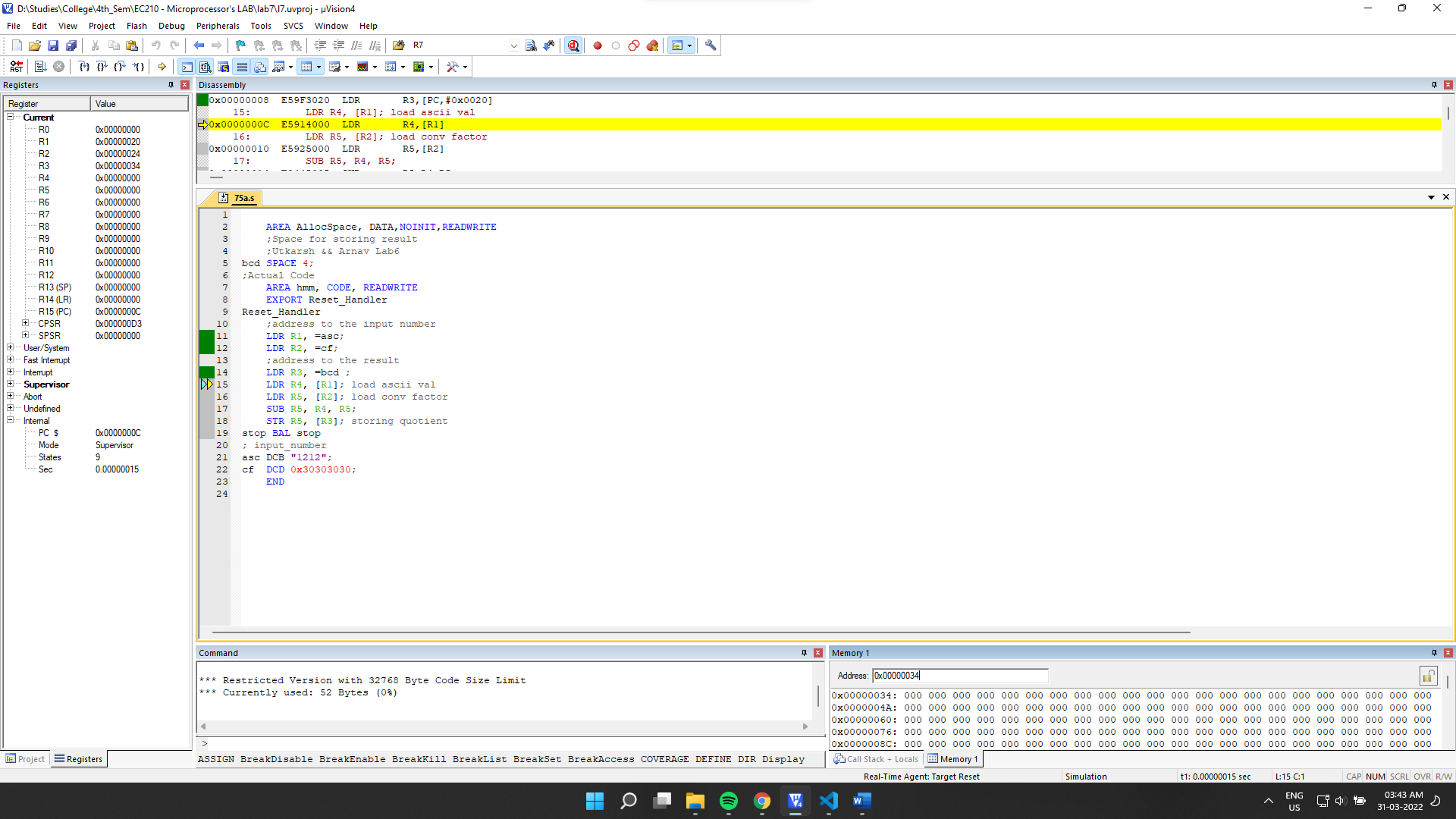
    END

Debugging:

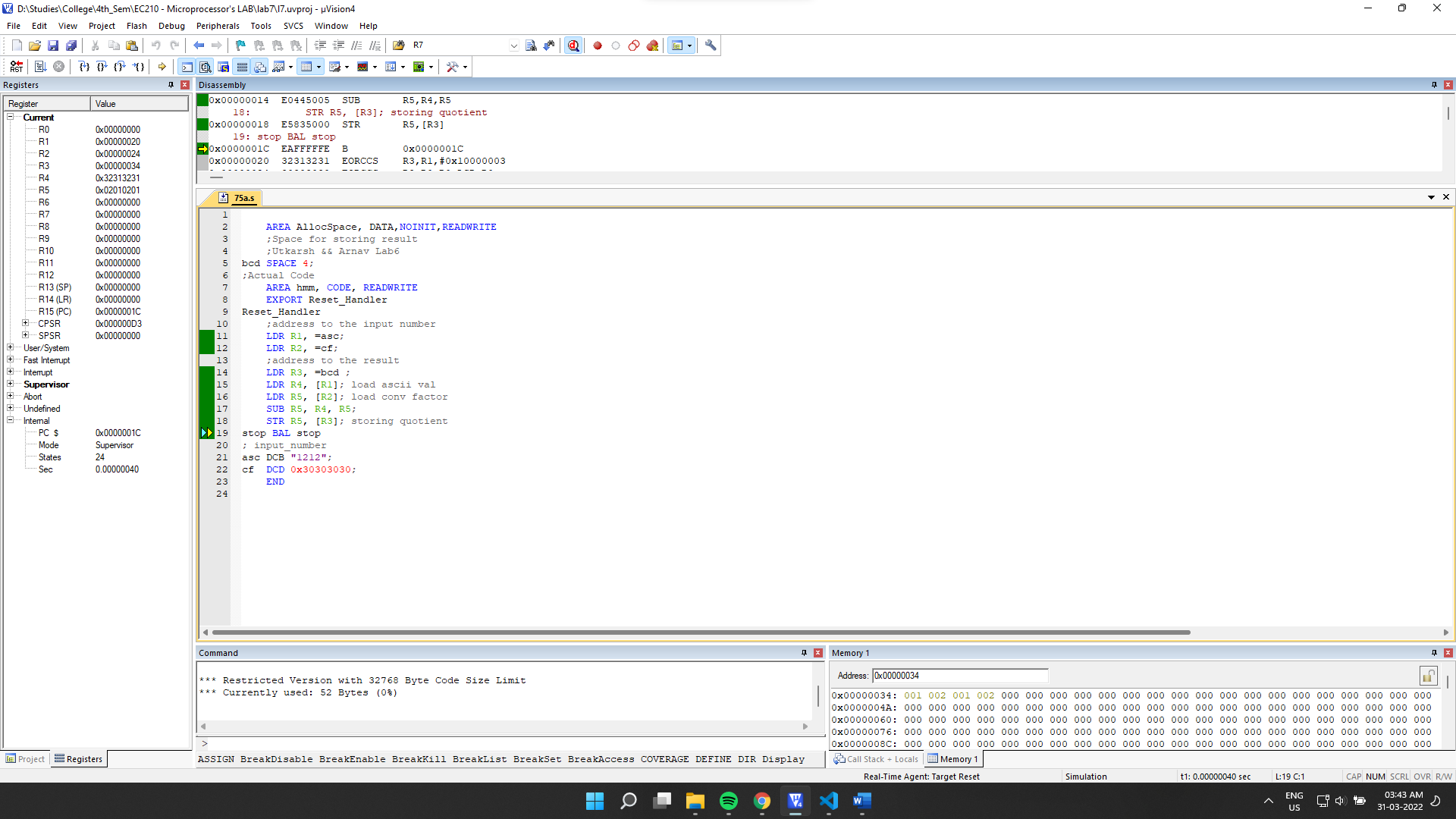
Initial Memory: (after getting the address through register)



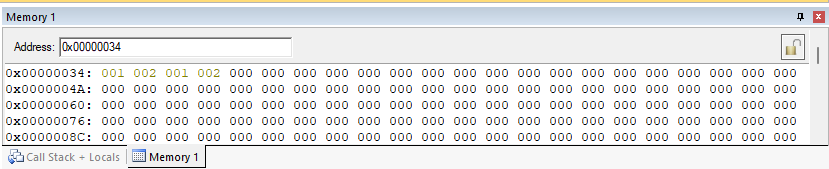
Setup:



Final Output:



Final Memory:



We can see that output in hex(decimal) matches our input (ASCII).

(b) ASCII to HX

-> Source Code:

    AREA AllocSpace, DATA,NOINIT,READWRITE

*;Space for storing result*

*;Utkarsh && Arnav Lab6*

bcd SPACE 4*;*

*;Actual Code*

    AREA hmm, CODE, READWRITE

    EXPORT Reset\_Handler

Reset\_Handler

*;address to the input number*

    LDR R1, =asc*;*

    LDR R2, =cf*;*

*;address to the result*

    LDR R3, =bcd *;*

    MOV R8, #1*;*

    MOV R7, #0*;*

    MOV R10, #10*;*

    LDRB R4, [R1, #3]*; load ascii val*

    SUB R5, R4, #0x30*; bcd = ascii-conv*

    ADD R7, R7, R5*;*

    MUL R9, R8, R10*;*

    LDRB R4, [R1, #2]*; load ascii val*

    SUB R5, R4, #0x30*; bcd = ascii-conv*

    MUL R5, R9, R5*;*

    ADD R7, R7, R5*;*

    MUL R8, R9, R10*;*

    LDRB R4, [R1, #1]*; load ascii val*

    SUB R5, R4, #0x30*; bcd = ascii-conv*

    MUL R5, R8, R5*;*

    ADD R7, R7, R5*;*

    MUL R9, R8, R10*;*

    LDRB R4, [R1]*; load ascii val*

    SUB R5, R4, #0x30*; bcd = ascii-conv*

    MUL R5, R9, R5*;*

    ADD R7, R7, R5*;*

    STR R7, [R3]*; storing quotient*

stop BAL stop

*; input\_number*

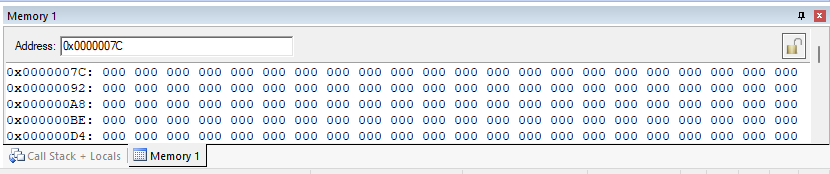
asc DCB "1212"*;*

cf  DCD 0x30303030*;*

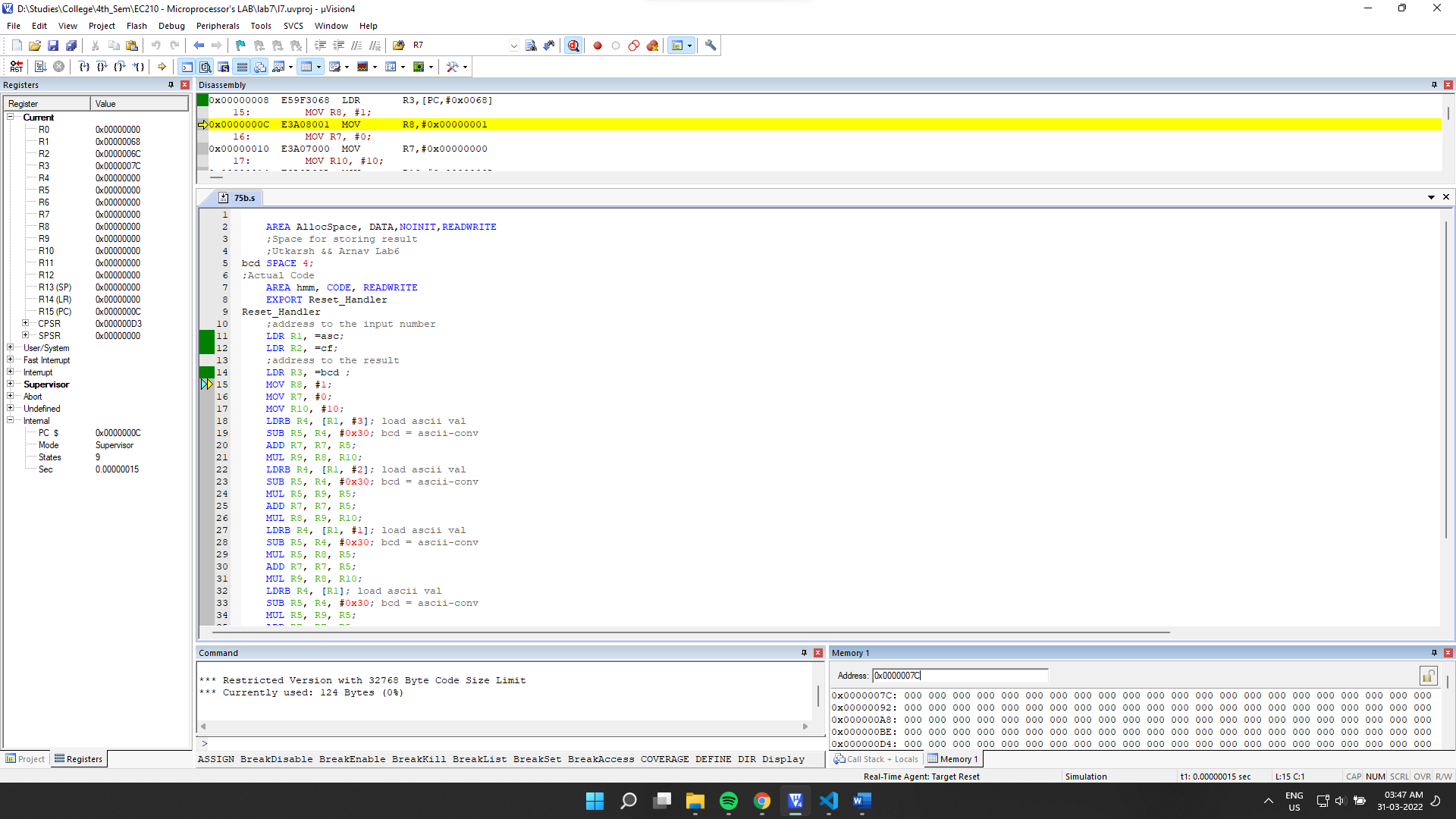
    END

Debugging:

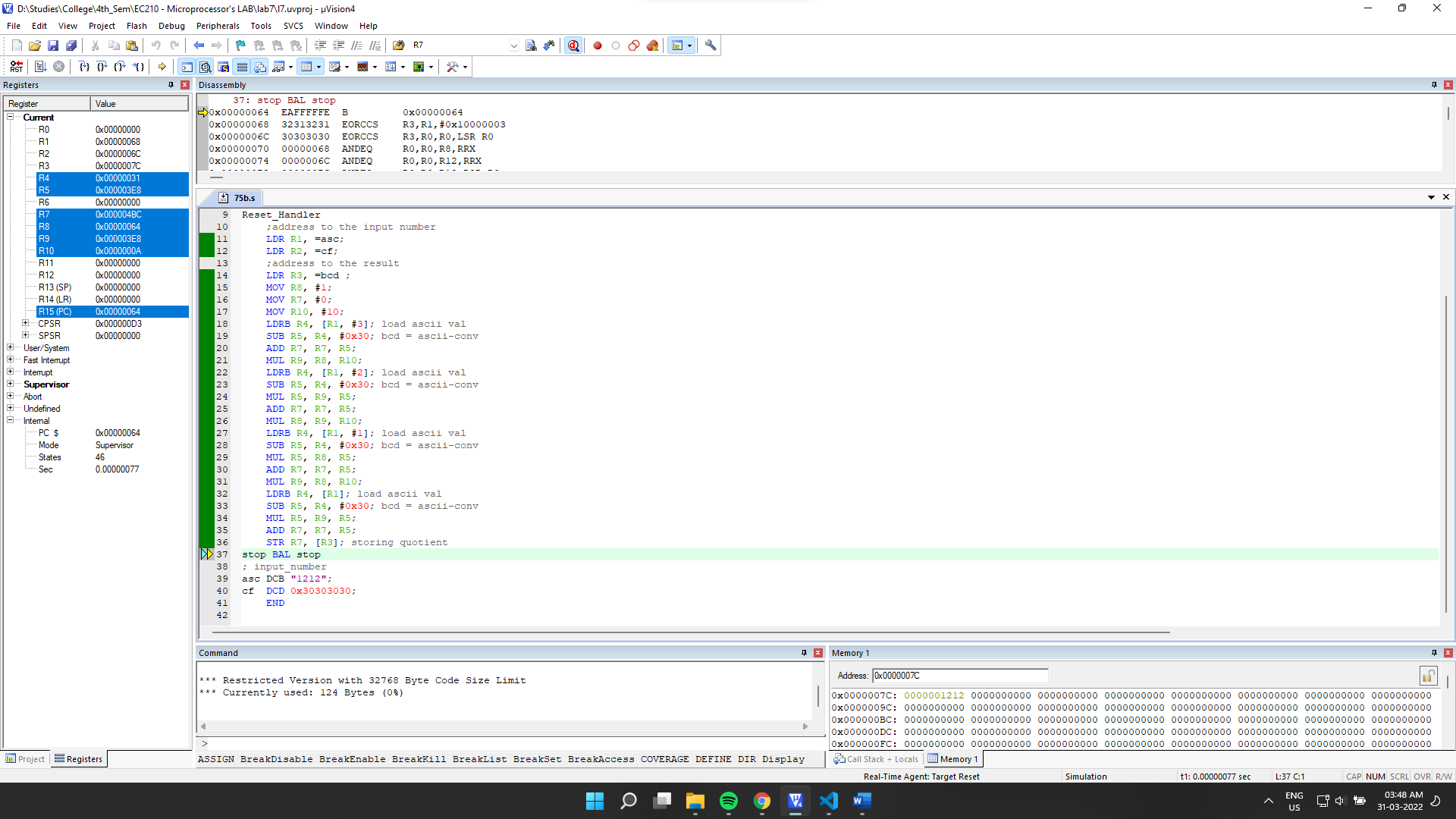
Initial Memory: (after getting the address through register)



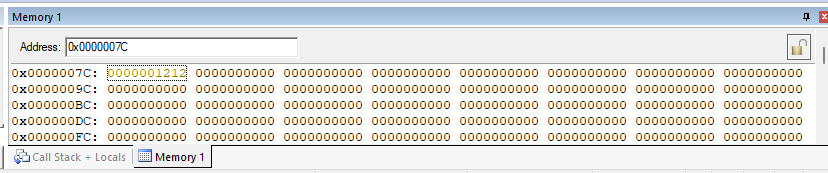
Setup:



Final Output:



Final Memory:



We can see that for the given input “1212” we got the proper hex value in the output.

(c) HX to ASCII

->

Source Code:

    AREA AllocSpace, DATA,NOINIT,READWRITE

*;Space for result*

array SPACE 1024*;*

*;Actual Code*

    AREA hmm, CODE, READWRITE

    EXPORT Reset\_Handler

Reset\_Handler

*;address to the input number*

    LDR R1, =inputnumber *;*

*;address for space used in operation*

    LDR R2, =array*;*

    MOV R12, R2*;*

*; load the  input number from memory*

    LDR R4, [R1]*;*

    LDRB R9, [R2]*;*

    ADD R2 ,#3*;*

*; Getting the last bit*

    MOV R5, R4*; R5 = R4*

    MOV R6, #0*;*

    MOV R7, #0*; for storing no of nos.*

*;dividing and finding nth digit.*

*;division by the use of subtracting*

*; until a remainder is found.*

*; where remainder < 10*

div SUB R5, R5, #10*; subtracting by 10*

    CMP R5, #10*; checking if remainder < 10*

*;incrementing iterator to store quotient*

    ADD R6, R6, #1*;*

*; if remainder >=10 then loop to sub further.*

    BPL div

    ADD R7, #1*; incrementing nos.*

    ADD R5, R5, #0x30*;*

    STRB R5, [R2], #-1*; storing digits*

*;comparing quotient and 10.*

    CMP R6, #10*;*

*;if quotient >10 then store quotient in remainder*

    MOVPL R5, R6*;*

*; && reset quotient*

    MOVPL R6, #0*;*

*; && loop again*

    BPL div*;*

*; check if quotient is zero*

    CMP R6, #0*;*

*; increment no of digits counter if no*

    ADDNE R7, #1*;*

*; &&  store it in R2 if no.*

    ADD R6, R6, #0x30*;*

    STRBNE R6, [R2]*; storing last digit*

*;*

stop BAL stop

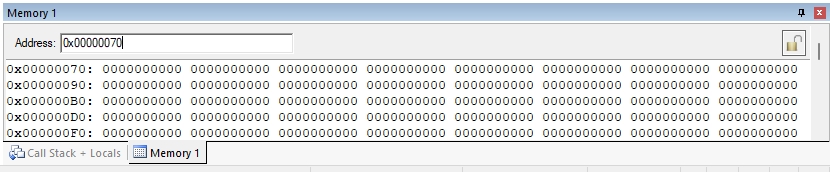
*; input\_number*

inputnumber DCD 1425*;*

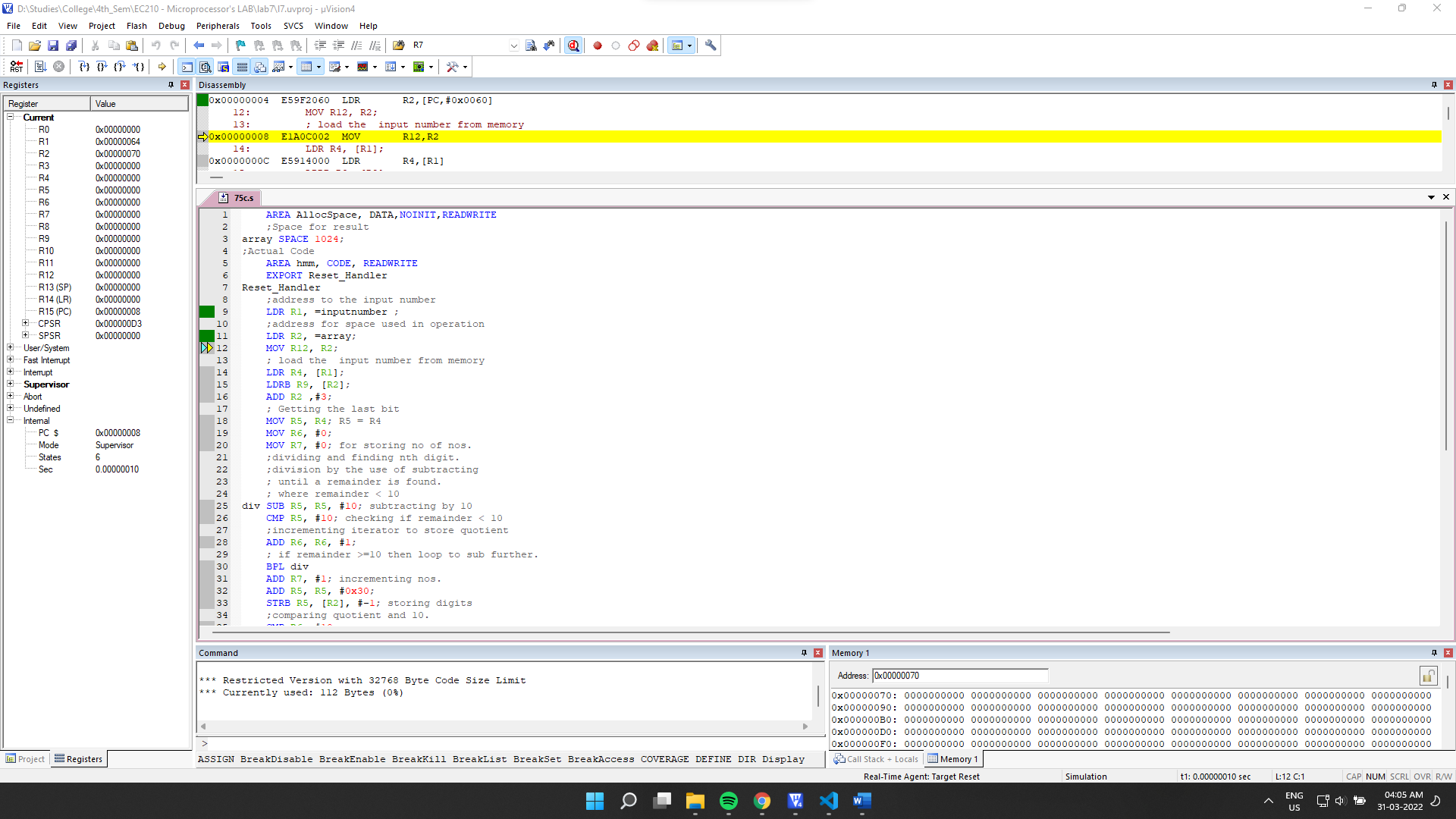
    END

Debugging:

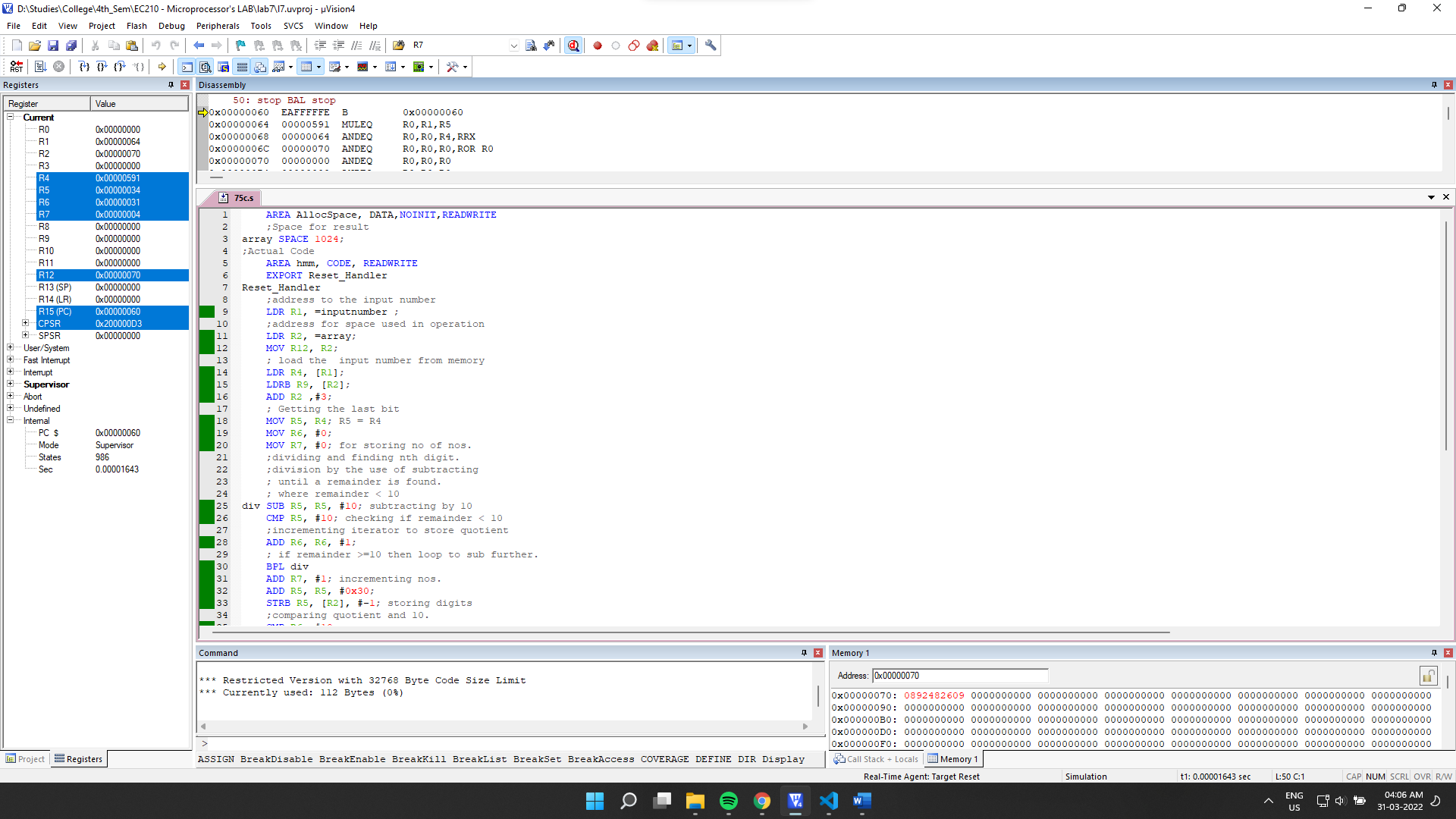
Initial Memory: (after getting the address through register)



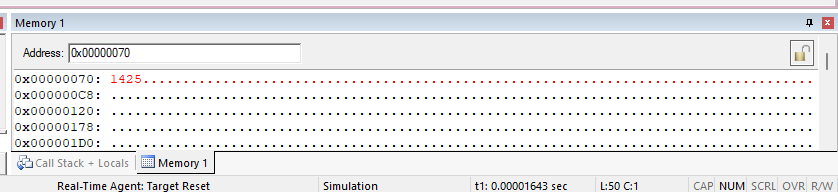
Setup:



Final Output:



Final Memory: (in ascii mode)



We can see that our output matches the expected result for the input 1425.

(d) BCD to ASCII

->

Source Code:

    AREA AllocSpace, DATA,NOINIT,READWRITE

*;Space for storing result*

*;Utkarsh && Arnav Lab6*

asc SPACE 4*;*

*;Actual Code*

    AREA hmm, CODE, READWRITE

    EXPORT Reset\_Handler

Reset\_Handler

*;address to the input number*

    LDR R1, =bcd*;*

    LDR R2, =cf*;*

*;address to the result*

    LDR R3, =asc *;*

    LDR R4, [R1]*; load ascii val*

    LDR R5, [R2]*; load conv factor*

    ADD R5, R4, R5*;*

    AND R6, R5, #0xFF*;*

    AND R7, R5, #0xFF00*;*

    AND R8, R5, #0xFF0000*;*

    AND R9, R5, #0xFF000000*;*

    LSL R6, #24*; shifting towards other side as*

    LSL R7, #8*; required for ascii string*

    LSR R8, #8*;*

    LSR R9, #24*;*

    MOV R5, R6*;*

    ADD R5, R7*;*

    ADD R5, R8*;*

    ADD R5, R9*;*

    STR R5, [R3]*; storing quotient*

stop BAL stop

*; input\_number*

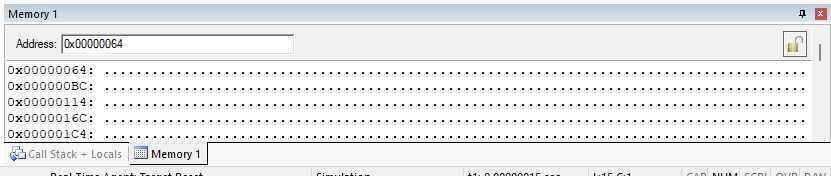
bcd DCB 2, 1, 2, 1*;*

cf  DCD 0x30303030*;*

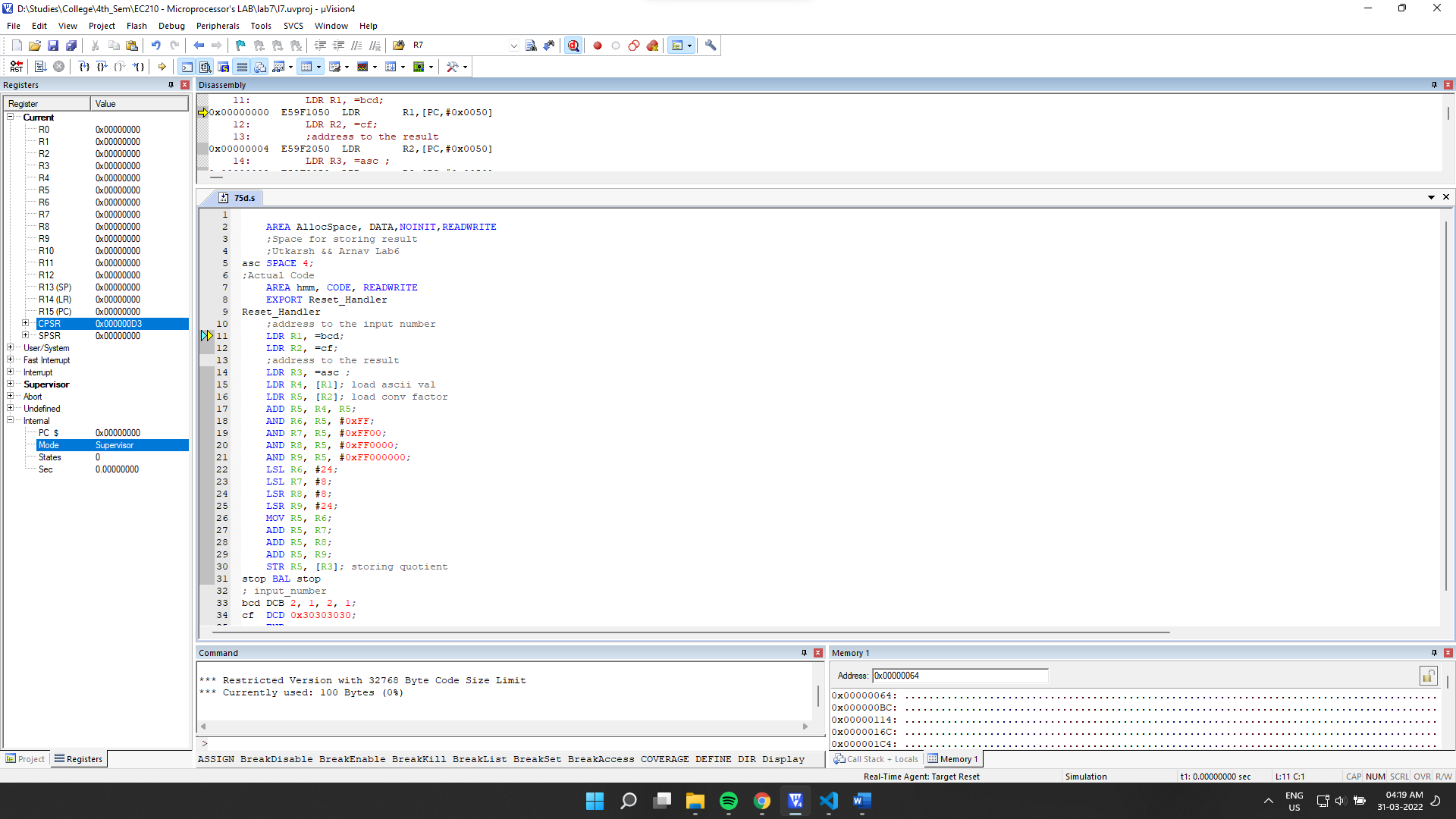
    END

Debugging:

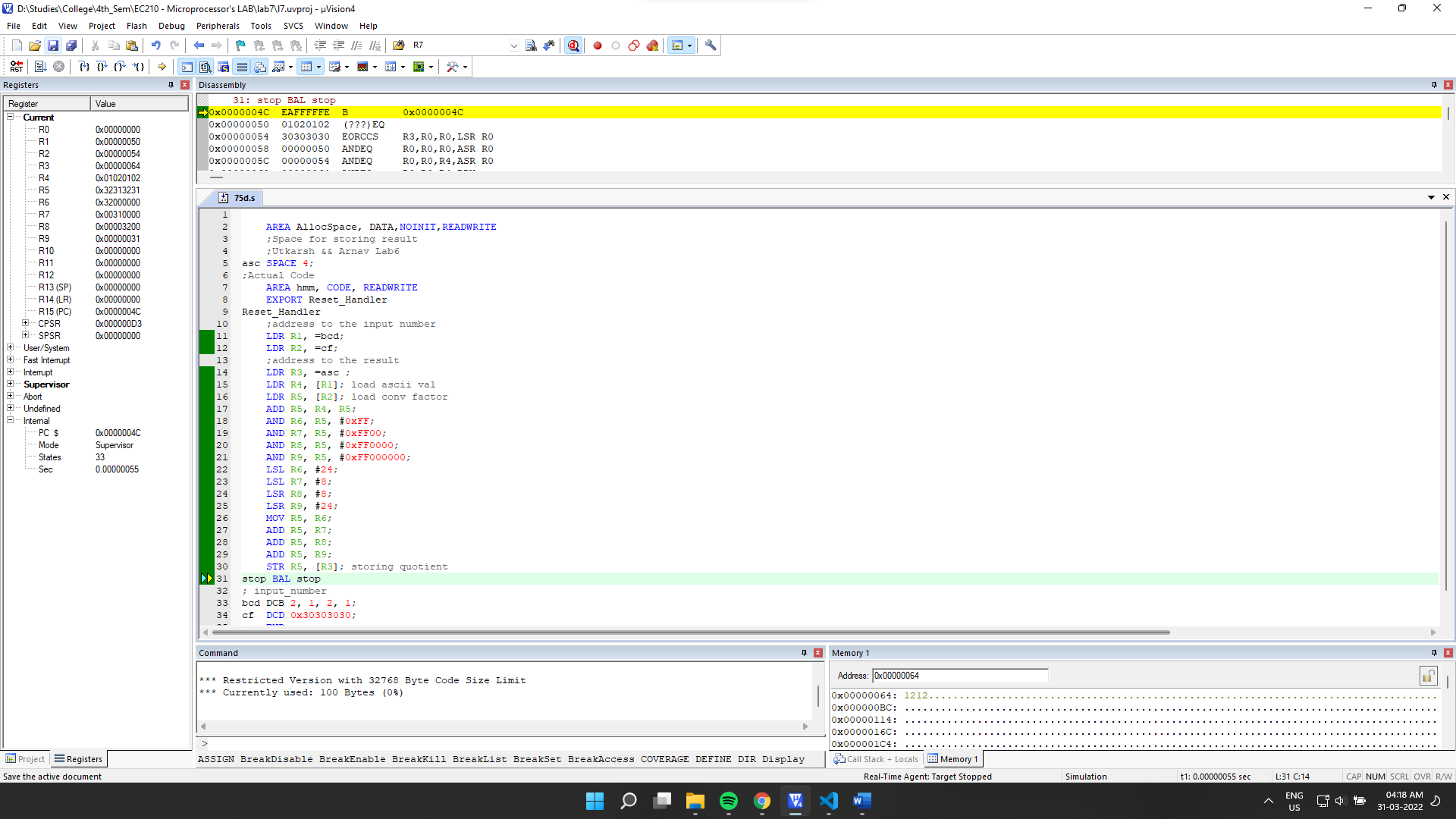
Initial Memory: (after getting the address through register)



Setup:



Final Output:



Final Memory:

