Marmara University
Faculty of Engineering

CSE 4219
Principles of Embedded System Design

Assignment # 1

Instructor: Sanem ARSLAN YILMAZ **Due:** 03.11.2024

	Department	Student Id Number	Name & Surname
1	CSE	150120066	Zeynep YILMAZ
2	CSE	150120055	Muhammed Talha KARAGÜL
3	CSE	150120070	Semih BAĞ

1) Question 1

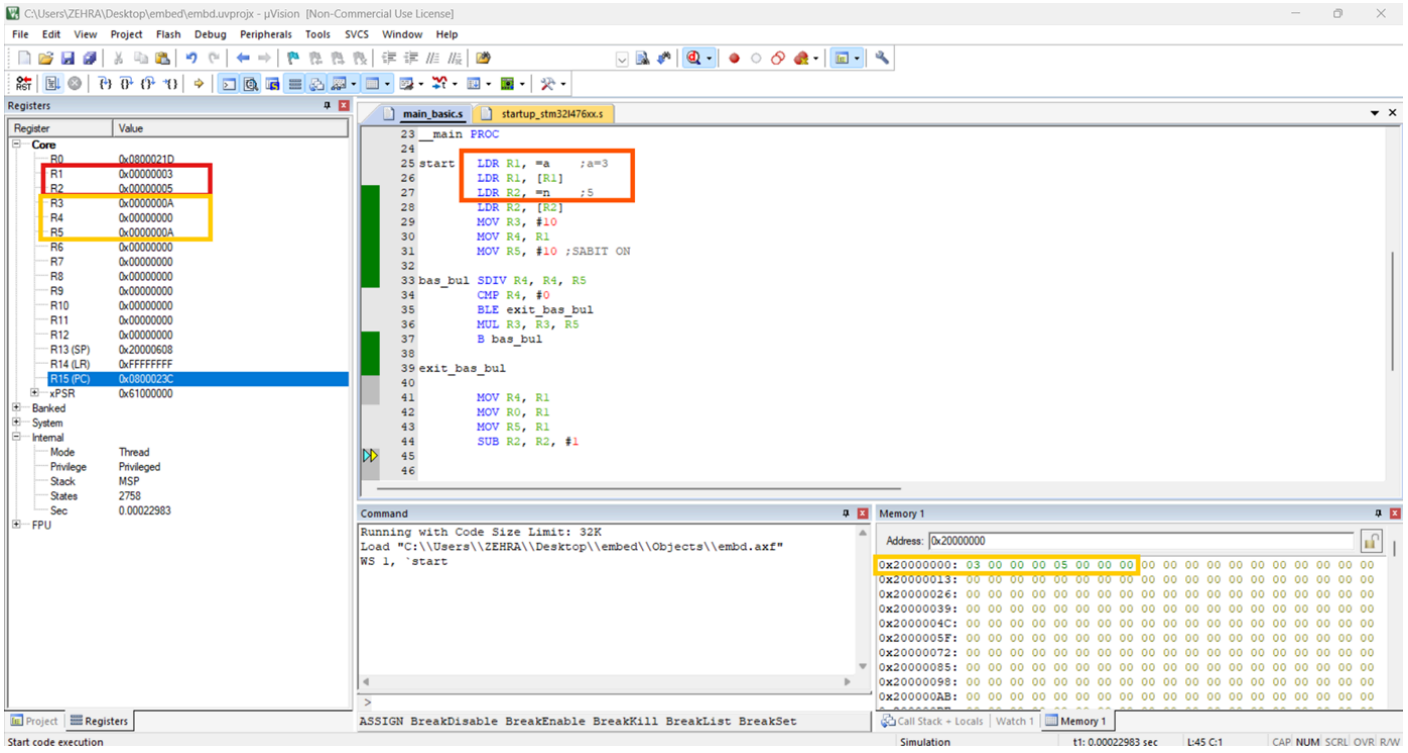
INFORMATION

There is no error in the code in the question, but when debugging, it works correctly with "f11" and when you press "f5", it does not show the register values correctly, and it writes values to the register we do not use.

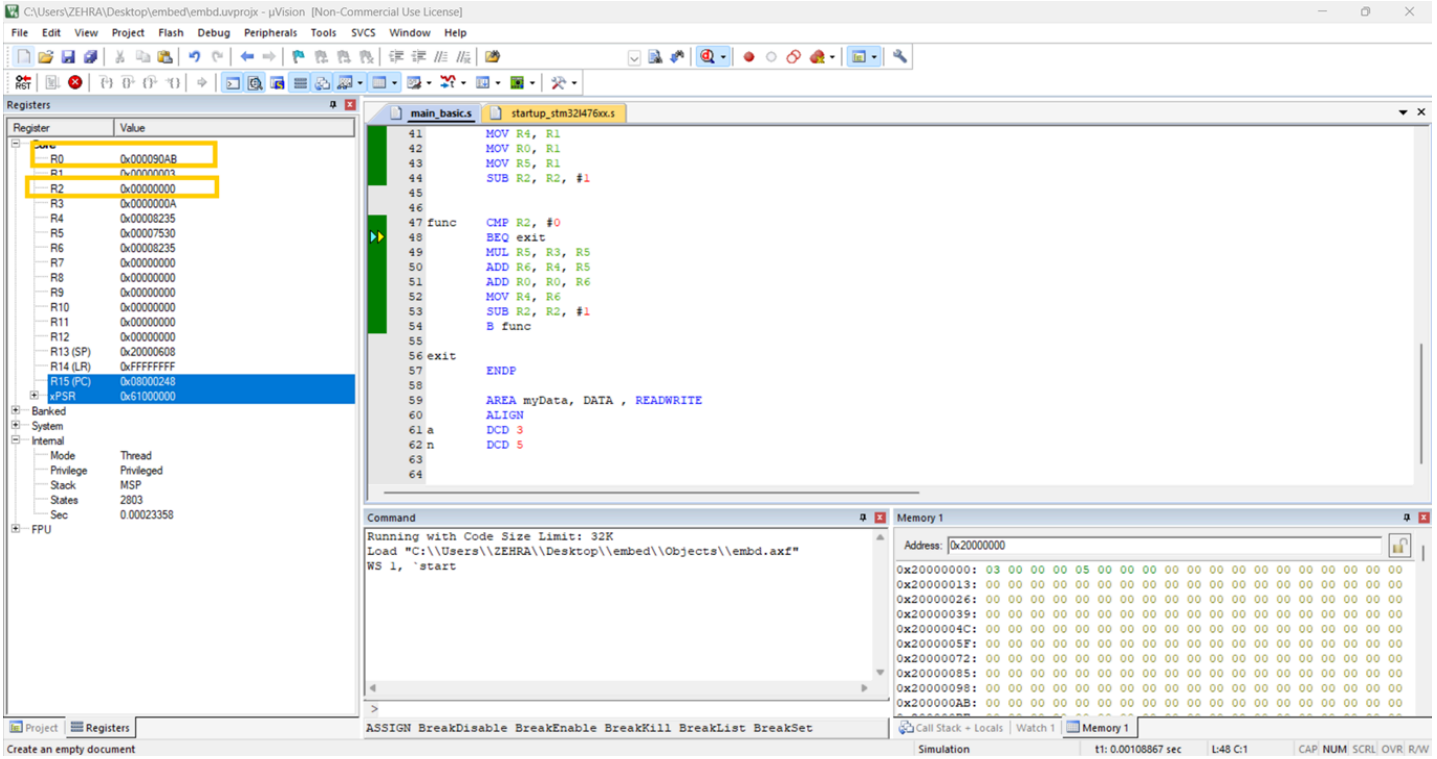
Algorithm:

A. First, assign the values of a and n to the R1/R2 registers
B. We assign the number to be multiplied to the R4 register and perform the operation on R4
C. The primary goal is to find the number of digits and understand how many times we will multiply the number while multiplying (exp: if it is 45, multiply it by 100 to make 4500+45=4545)
D. We enter the bas_bul function, we apply /10 to the R4 value to find the number of digits
E. The loop rotates until it reaches the value of 0, and at the same time we multiply the multiplier by 10 to finalize it
F. R3 becomes our multiplier after this loop
G. We come to our main function, I will define a few registers here
    a. R0 is the register where we perform the cumulative addition
    b. R4 is the register where we hold the repeated number from the previous loop
    c. R5 is the register that is continuously multiplied by the multiplier (it holds the number that goes as 45-4500-450000)
    d. R6 is the register where we temporarily hold the repeated number hold
    e. R0 added register (45-4545-454545)
    f. R2 is the register that holds the "i" value required for the loop
H. In the "func" part of the code, the part where the loop is entered as much as the entered n value and the cumulatively desired addition is made
I. First of all, we approached the question as if there were 2 cumulative additions here, the first will be our result value and will add all the loop results, the second will be the value to be added in that loop will also include the number from the previous value
J. Therefore, first we calculate the number in that loop and add the number from the previous loop to it to calculate the number to be added to the result
K. Then, by adding this to the result, we prepare the repeated number of the number to be added to the instant result in the next loop (MOV R4, R6)
L. The value we finally reach becomes our cumulative total value

In the example you gave, a=3, n=5, when writing the value 3 side by side, it needs to be multiplied by 10, so R3 was separated as 10 from the bas_bul function and written to the register values.



After the cumulative total was completed, it was saved in R0 and returned to R2, that is, n, and exited the function.



2) Question 2

Algorithm:

A. First, we assign the values to be given to us (array elements, matrix values, column numbers to be changed) to our registers.

B. Since the aim of the problem is to change 2 columns in the matrix, it should enter the loop as many times as the number of rows and continuously adjust the position in the array. We aimed to reduce the number of instructions in the loop by calculating these distances in advance. Therefore, we defined the position of the 1st column as R3 (Array start address + 4 * desired 1st column) and the position of the 2nd column as R4 (Array start address + 4 * desired 2nd column) outside the loop.

C. At the same time, we defined R1 to move to the next row in each loop and wrote the number of rows into R1. We did this in order to move the column registers to the next row in each loop without moving them.

D. We set R6, which we will use for the number of loops, to zero and send it to the loop where it will return as many times as the number of rows.

E. Here we will define a few registers

a. R7/R8- Registers that will temporarily store the values in the columns to be changed
b. R3- Register that holds the address of the values in the first column to be changed, will point to a lower row in each loop
c. R4- Register that holds the address of the values in the second column to be changed, will point to a lower row in each loop
d. R1- Register that holds the number of rows to move the R3 and R4 registers to the next row
e. R2- Number of rows that determines how many times the loop will be
f. R6- Value of i that ensures the loop continues as many times as the number of rows

F. In the "loop" section of the code, there is our loop that is necessary for the values in the columns to be changed

G. First of all, we wrote the values in the columns to our temporary registers (R7/R8)

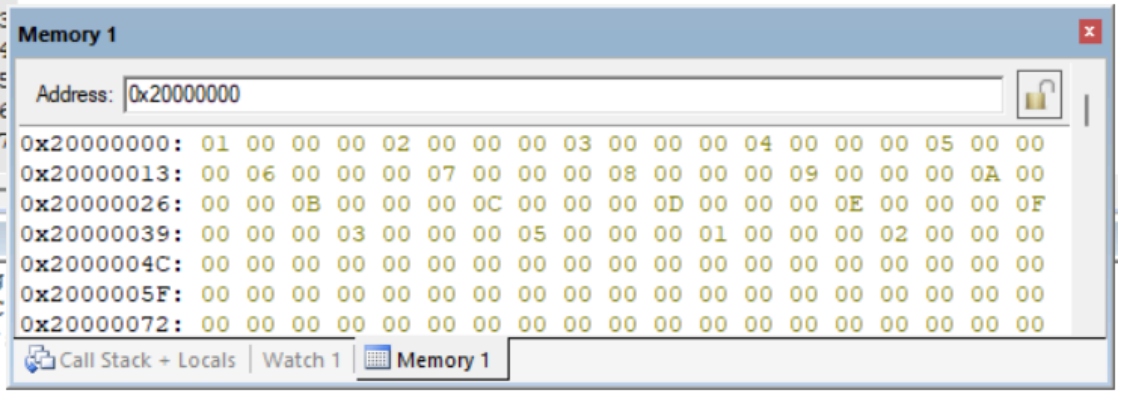
H. Then we wrote them to each other's addresses by crossing them.

I. Then, we added the R1 value that we defined before the loop to the two column values to move to the next row and moved our addresses that point to the column to the next row.

J. Then, we increased our loop count by 1 and compared it with the number of rows to exit the loop/repeat the loop. We have completed the start process.

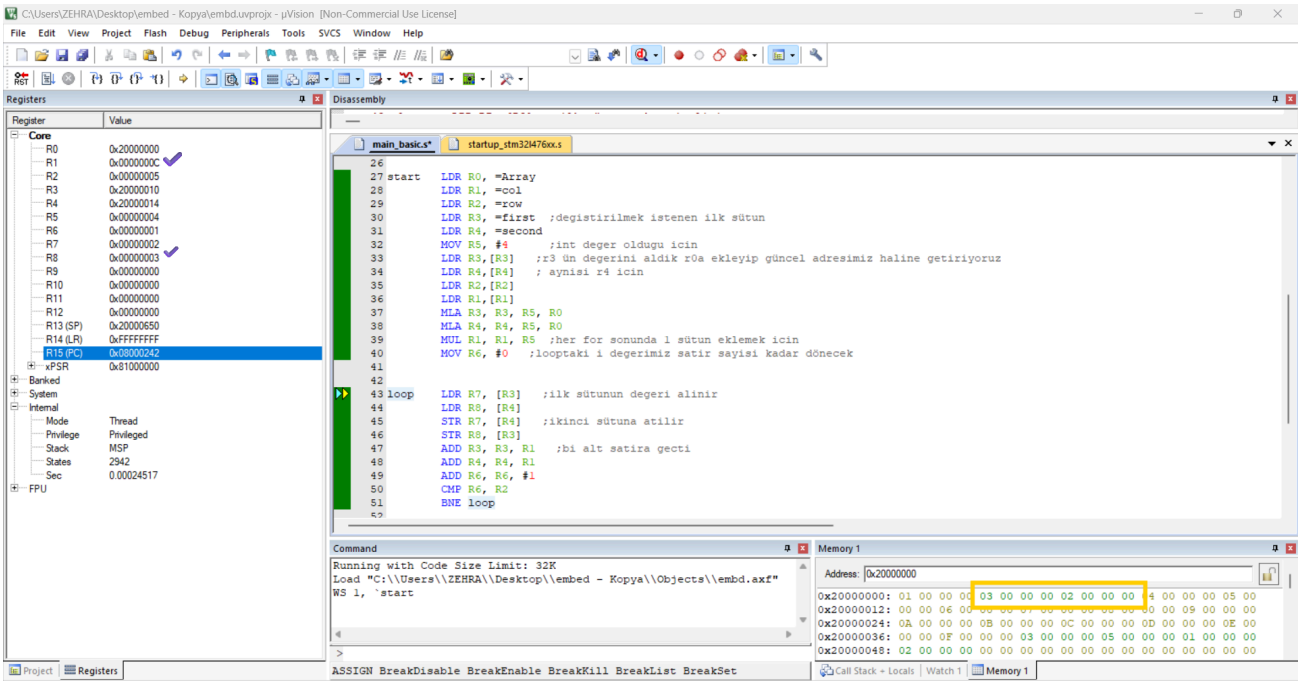
All the variables we defined in the data area section are written to memory:

```
55
56          AREA myData, DATA , READWRITE
57          ALIGN
58 Array    DCD 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
59 col      DCD 3
60 row      DCD 5
61 first    DCD 1
62 second   DCD 2
```

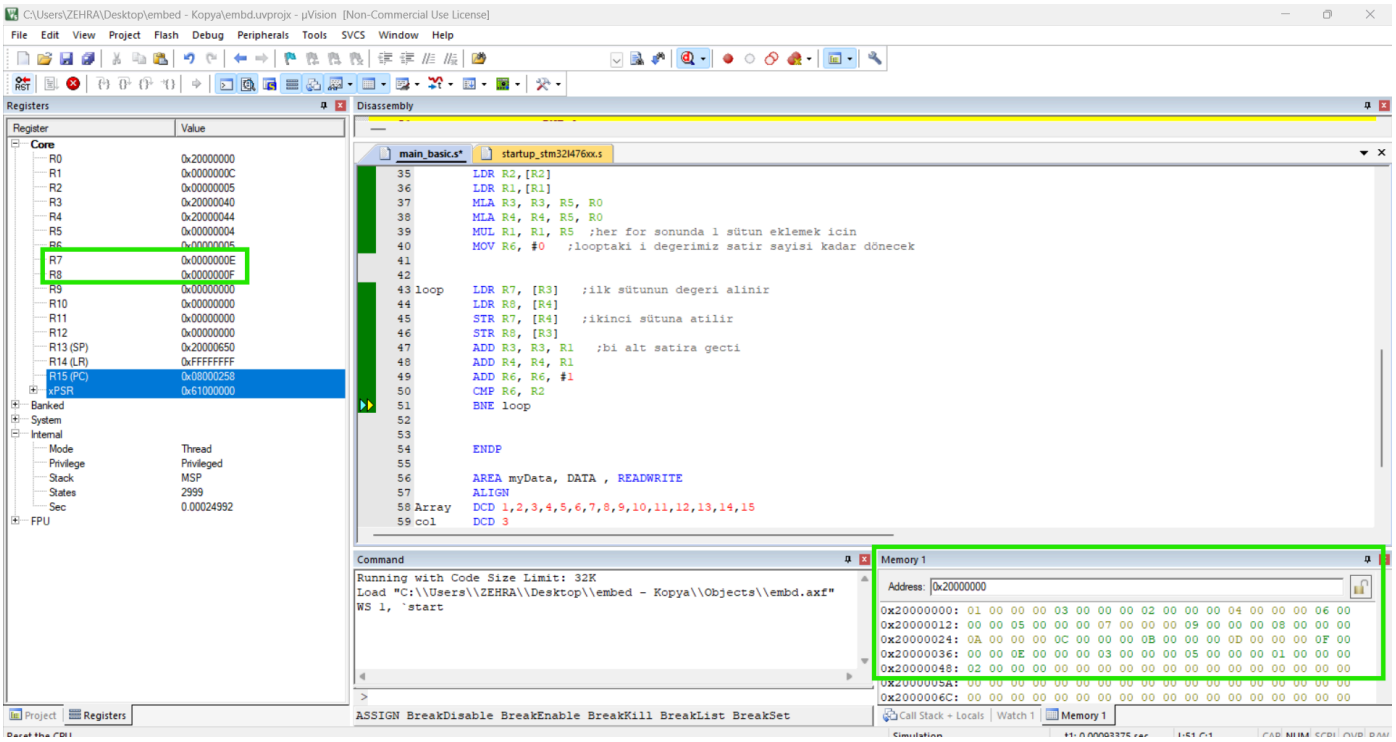


After completing the first round in the loop:

- ➔ R1 value is the row that needs to be added (4*12= 0xC)
- ➔ R7 value is the first row first column value =2
- ➔ R8 value is the first row second column value =3
- ➔ R6 has completed the first round in the loop = 1
- ➔ Finally, the R0 value does not change at all and the array start is 0x20000000



When the loop is completed, the hex values of R7/R8, E and F, and the values we want in the array have changed.



3) Question3

INFORMATION

We saved the result value to R8!!

Algoritma:

A. First, we extended the 8-bit data given to us and turned it into 13-bit data. While extending, we split the 8-bit data in our hand to put p0-p1-p2-p4-p8 in between (we added the necessary bits to the desired position by shifting

B. Then, we sent the information of the added bits to our function that undertook the main task

C. Let's explain the information of the added bits and the necessary registers as follows

i. R5- Since the added bits will take value according to the "1" repetition of certain bits in the data (0 if it is even, 1 if it is odd), we collected the number of times it was repeated on R5

ii. R1- An array that we took as input, this array holds the location of the data that the p values should look at (For example, for p8, positions 9,10,11,12 are looked at, for this, we keep the values of 512,1024,2048,5096, which are decimal values, as an array in R1 and when necessary, we take only that bit with the and operation.) We assigned this to R1 and this is a function that returns as much as the length of the array in R1 We put it in.

iii. R2- The register we use to go through the values in the R1 array one by one

iv.

v. R0- The register that contains the 13-bit data we extended

vi. R4- The 13-bit data and the result of the bits in the locations to be looked at with the and

vii. R5- A register we use to exit the loop

D. The function that performs the operation is:

E. First, we write the value in the location array to be looked at to R2

F. Then, since we added the value "0" to the end of all arrays for the "end of the array", we ensure that it returns until the "0" element is reached with the CMP instruction

G. If it is equal, we go down to the return section in the function and and the value in R5 with #1 according to the value in its last bit, and we understand whether the total number of 1 is even or odd

H. Then we return to the value in the 'link register'

I. If it is not equal, we continue the loop and write the hex value of the value in R2, the bit we will look at, and R0, our 13-bit data, to R4 with the and instruction

J. Then, we compare R4 with #0 and We check if it contains '1'

K. If it contains '1', we increase our R5 register by 1 with the addne conditional addition

L. If there is no 1 value in the location being looked at, all of our bits become 0 and R5 does not change

M. Then we go to the next value in the R1 register that holds the location array to be looked at and we re-enter the loop

N. When we repeat this until it is finished, we return to main with the link register and after shifting our R5 value below the required bit, we add it to our R0 value so that we write the actual values to our p8-p4..etc values in the 13-bit data

As a result, the hex version of the 13-bit data is kept in the r0 register. It holds data starting from 0x2000000 in memory.