

**CSE 4219 Principles of Embedded System Design**

**Project-1 Report**

Ahmet Abdullah Gültekin 150121025

Ayşe Gülsüm Eren 150120005

Enes Haksoy Öztürk 150120024

**Q1)**

The code starts by setting initial values for a, n, and sum. Then, it enters a loop that repeats n times. In each iteration, the code first shifts `sum` to make space for a new digit, then adds the current i value as a new digit to sum. After that, i is incremented by one. Once i reaches n, the loop ends, and the final step multiplies sum by the initial value of a to get the final result.

For instance, let f(a, n) = f(3, 5) like in the given example.

f(3, 5) = 3 + 33 + 333 + 3333 + 3333 = 37035

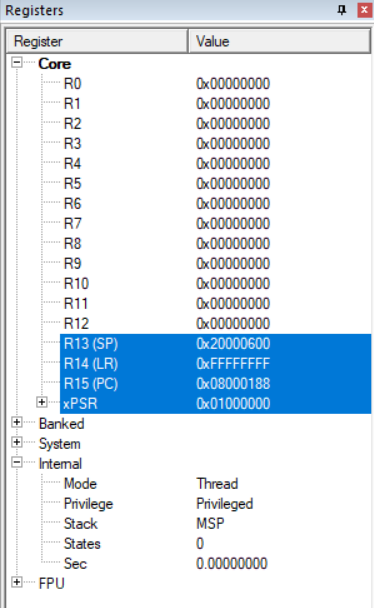
Actually it is equal to the (12345 \* 3) and this means that the formula should be

f(a, n) = 1234…. (until ‘n’) multiplied by a. In more detail,

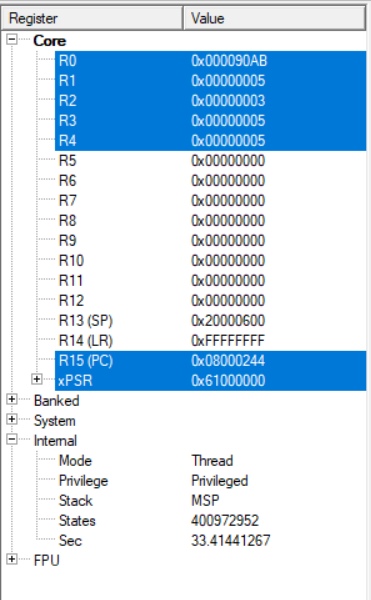
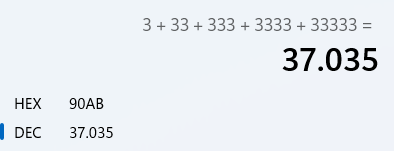
f(3, 5) = (10000\*1 + 1000\*2 + 100\*3 + 10\*4 + 1\*5) \* 3 = 12345\*3 = 37035

I store the result in R0 at the end of this iteration (loop).

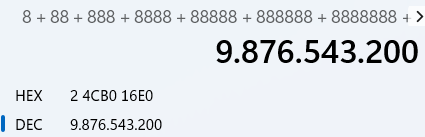
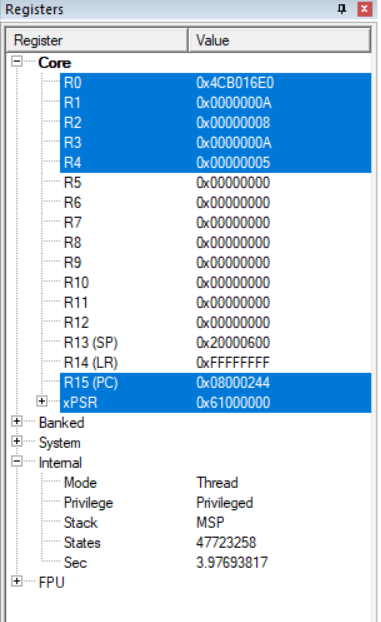
Registers at initial state



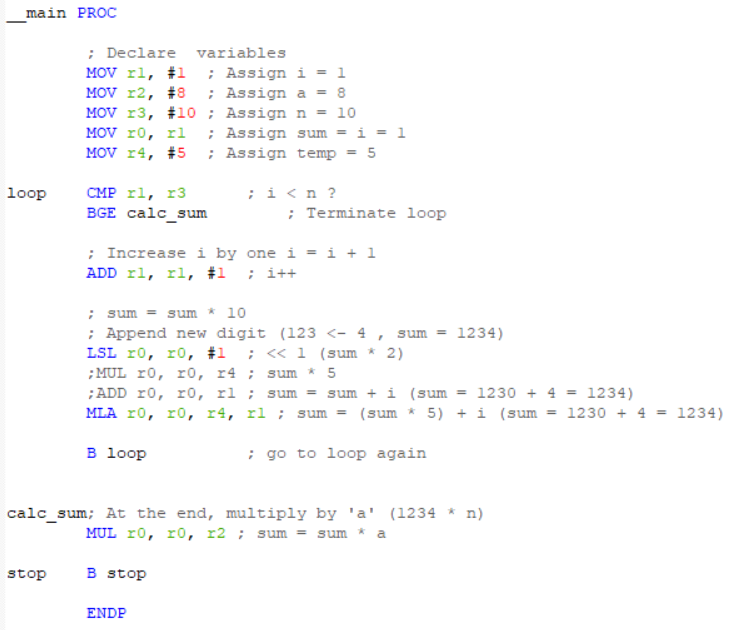
For a=3, n=5:

For a=8, n=10:



Source Code



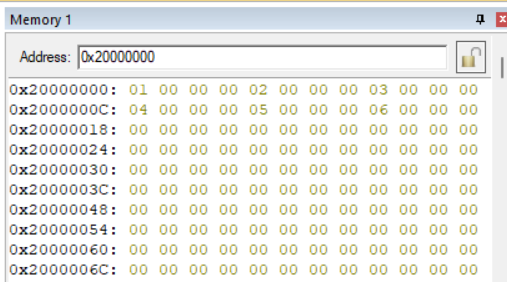
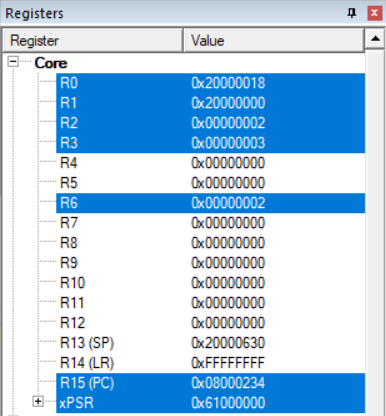
**Q2)**

The code starts by setting initial values for matrix row count, column count, number of columns to swap, i and j for loop operations. r0 and r1 includes Matrix addresses. fill\_loop controls the copy operation using i , according to column number it calls relevant label. r10 shows matrix1’s column address using i. r11 shows matrix2’s column address using i, it is different for all labels so r11 calculated 3 times in fill\_loop. If i not equal to swap columns, it calls copy\_column which copies the column to same column, otherwise it calls mov\_1to2 or mov\_2to1 which copies the column to special column.

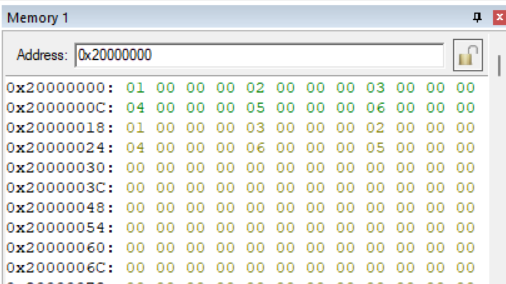
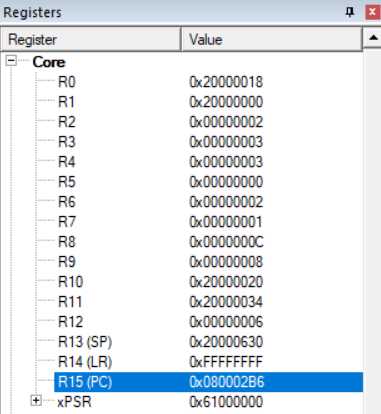
For [2][3] matrix 1, 2, 3, 4, 5, 6 .Swap columns 1 and 2.

0x20000000 first matrix’s address. 0x20000018 new matrix’s address

Registers and memory at the beginning

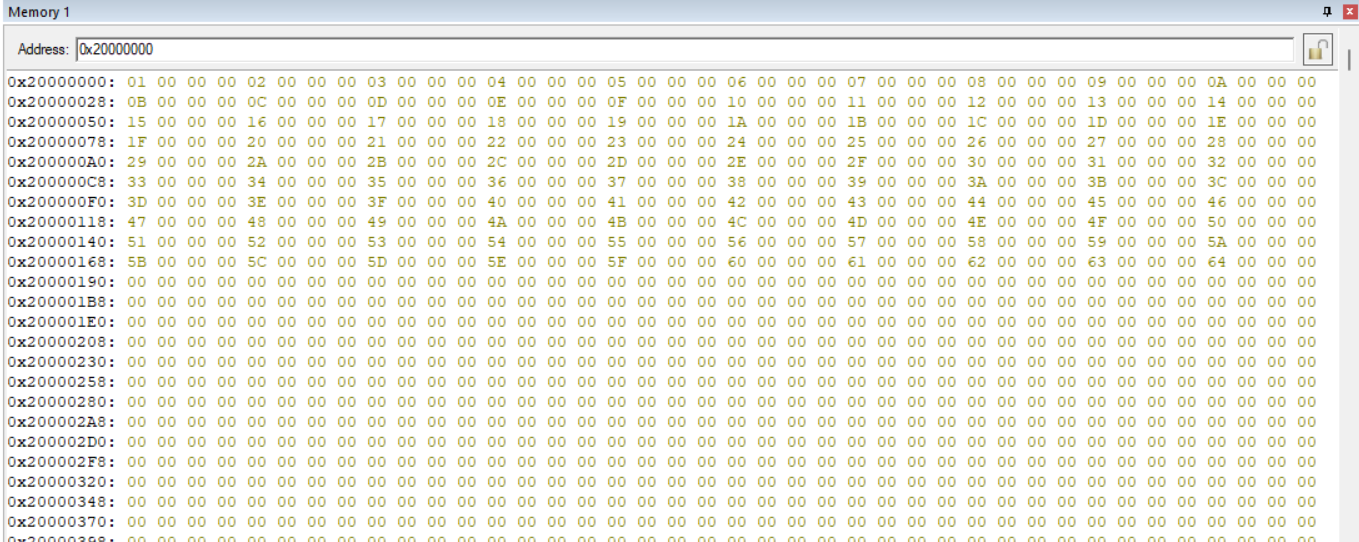


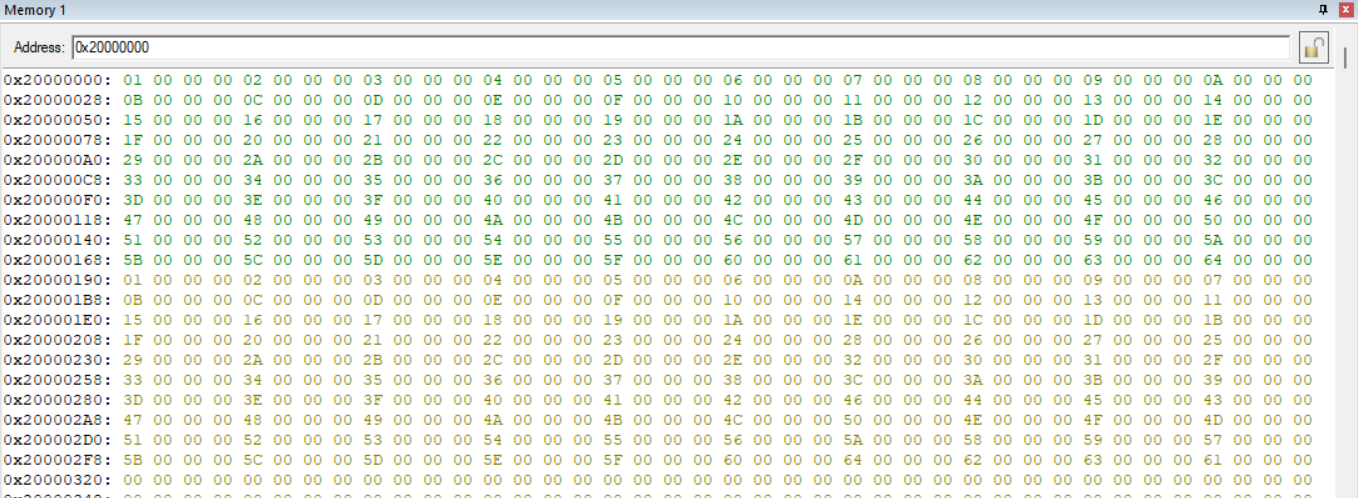
Registers and memory at the end



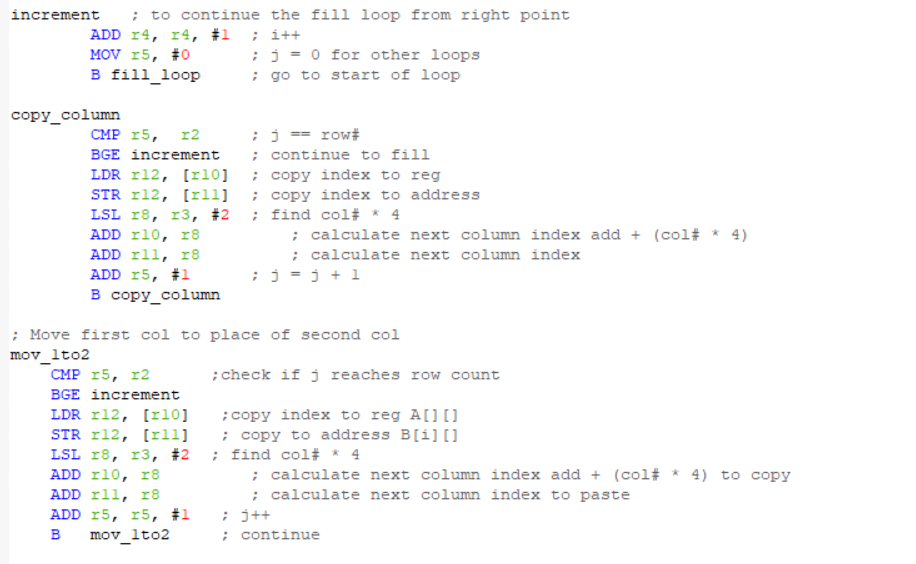
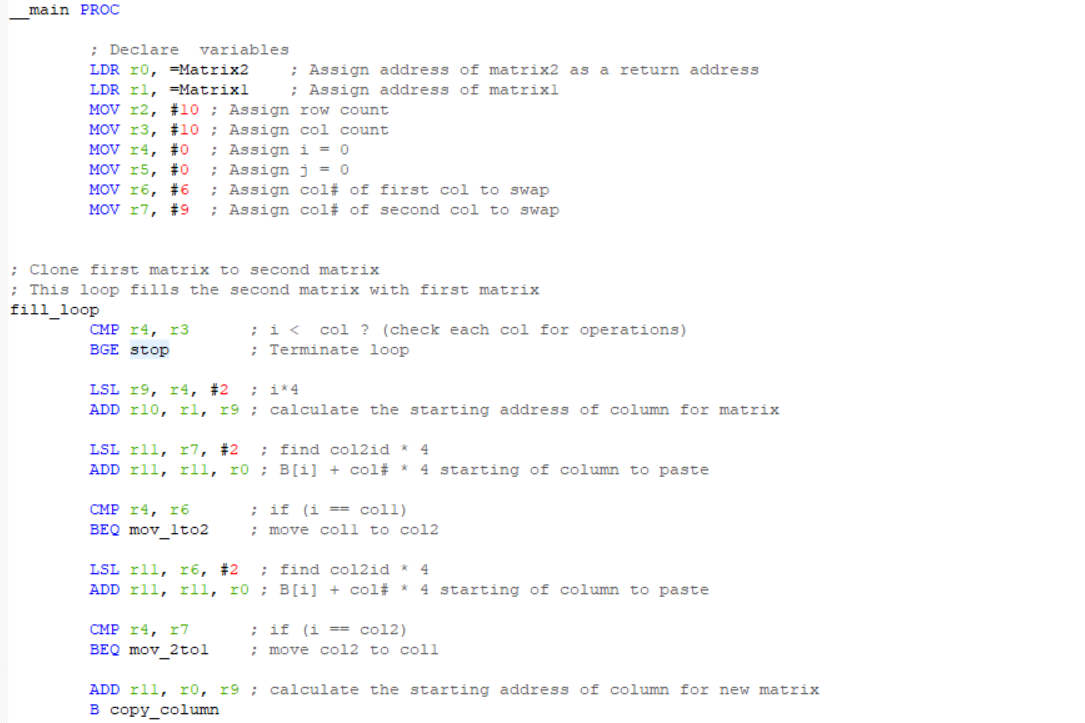
For [10][10] matrix .Swap columns 6 and 9.

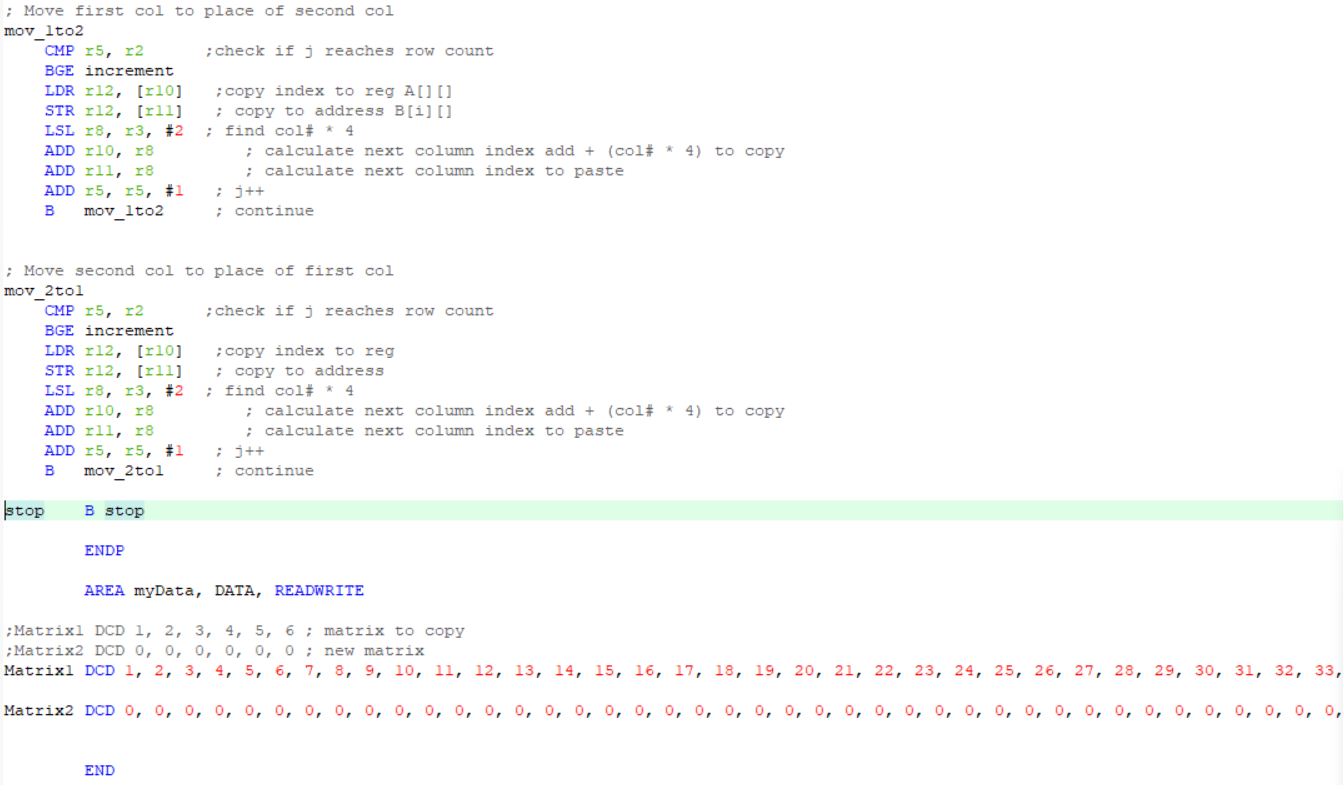
0x20000000 first matrix’s address. 0x20000190 new matrix’s address





Source Code





**Q3)**

In this question, firstly we give a constant 8-bit data in memory and 13-bit memory section with full of zeros. We give the 8-bit data to r0 with LDR operation. We shifted the r0 to access the bits of the locations and give them to registers such as r1=first bit,r2=second bit etc. For doing that, we used MOVS operation with barrel shifter, and we iterated the wanted bit into carry. After that we used ADC with the cleaned registers. After that we cleared the register for the wanted p0, p1, p2, p4, p8 bits. Then we perform the given rules and find the values of the parity bits before the mode operation. For doing the mode operation, we created a loop and checked the given value, if it is less than two, we give the mode operation’s result to the p1, p2, p4, p8 parity bits. After that, we added all the bits and performed mode operation for the p0 parity bit. Then we perform Logical Shift Left operation for the moving proper places to our bits. And after, we added the results of the Logical Shift Left operation and stored in r0. Finally, we LDR the 13-bit data location to r1 and we stored the final result r0 in there with STR operation.

metin, ekran görüntüsü, sayı, numara, paralel içeren bir resim

Açıklama otomatik olarak oluşturuldu

We used the value given in assignment pdf 0xB3.r8 assigned to the eighth bit, r7 assigned to seventh bit etc. Before finding the parity bits our memory looked like this.

metin, ekran görüntüsü, sayı, numara, paralel içeren bir resim

Açıklama otomatik olarak oluşturuldu

Before the mode operation, our parity bits (r9=p1, r10=p2, r11=p4, r12=p8) looked like this.

metin, ekran görüntüsü, sayı, numara, paralel içeren bir resim

Açıklama otomatik olarak oluşturuldu

After the mode operation our parity bits looked like this. Still, we did not calculate the p0 bit. Now we calculate the p0 parity bit.

metin, ekran görüntüsü, sayı, numara, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

After calculating the parity bit p0, we found it 7.we stored it in the r0 because we used all other registers to hold the 12 bit data bit by bit and we are not using the address of the 8-bit data anymore.

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

After all the calculations, we found the parity bits and held them in the registers. After this step of the project, we added properly located bits to the r0 which holds the p0 bit. We did not perform any shift operation in r0, and because of this we added all other values to the r0.

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

In this photo, we showed that our result is in the r0 and the allocated memory address for the output of the program is in r1. After that we stored the r0 into that address.

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

In this photo, our memory is shown after our program stopped. As we see, our compiler added 2 bytes of padding to the memory because of the performance. And we see that our result is written as 2B-17. It is because the lower memory address holds the lower bits of the data. Here is the program we wrote.

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, çizgi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu