**CompE-271**

* I declare that all material in this assignment is my own work except where there is clear reference to the work of others.
* I have read, understood and agree to the SDSU Policy on Plagiarism and Cheating on the university website at <http://go.sdsu.edu/student_affairs/srr/cheating-plagiarism.aspx> , the syllabus and the student-teacher contract for the consequences of plagiarism, including both academic and punitive sanctions.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Remark\*. By submitting this assignment report electronically, you are deemed to have signed the declaration above.*

10/27/2019

Homework 7

[Hw7]

Ckick below to enter/change your Name and RedID

Ethan Nagelvoort, 821234668

**Content**

(\* - Mandatory)

1\*. Description of the problem/method

I first had to find the length of the string by using a while loop that would end if r2 would ever contain zero. I used ldrb to load r2 with the first character in the string and over the course of the while loop, it would hold every other character until it would hit 0. I would also have another register, r1, that would act as a counter and would iterate every time the loop would restart, this would hence give the length. The function then goes into a for loop that uses r3=0 and r1 in cmp. Once r3 is equal or greater than r1, it will branch out to end. If r3 is less than r1, the function will branch to inside. In inside, the parentheses portion of the magic number equation occurs. I use ldrb to access the strings characters and add and mult for the rest of the equation, the final value is stored in r4. R3 is also iterated and then the function branches back to the for loop. In the end branch, I mod r4 by 1001. To do this I get one register to equal 1001 and then use udiv, mult, and sub to simulate the operation of a mod, the function then stores the final value in r2. R2 is then returned.

2. Pseudocode (if required. Mandatory for the Lab assignments, starting from #5 and Projects)

computeMagicNumber:

save stack pointer in register r12

reserve 32 bytes for local variables

push link register onto stack

r1 = 0, this is counter variable

r0 holds parameter string

inside while branch

load register r2 with whatever is in mem location of r0+r1

branch to zero if r1 = 0

branch to notZero if r1 != 0

inside notZero branch

r1 = r1 + 1

loop back to while branch

inside zero branch

r2 = 0

r3 = 0

branch to For branch

inside For branch

branch to end if r3>=r1

branch to inside if r3<r1

inside inside branch

load register r4 with whatever is in mem location r0+r3

r3 = r3 + 1

r4 = r4 \* r3

r2 = r2 + r4

loop back to For branch

inside end branch

r3 = 100

r4 = 10

r3= r3\*r4

r3= r3+1

r1 = r2/r3

r1 = r1\*r3

r2=r2-r1

return r2

pop link register from stack

restore the stack pointer

3\*. C-code

/\*

\* AssemblyFunctions.S

\*

\*/

.global computeMagicNumber

.text

computeMagicNumber:

mov r12, r13 // saves stack pointer in register r12

sub sp, #32 // reserve 32 bytes for local variables

push {r2} // push link register onto stack

mov r1, #0 // r1 = 0, this is counter variable

// r0 holds parameter string

while: // inside while branch

ldrb r2, [r0,r1] //load register r2 with whatever is in mem location of r0+r1

cmp r2,#0

beq zero // branch to zero if r1 = 0

bne notZero // branch to notZero if r1 != 0

notZero: //inside notZero branch

add r1, r1, #1 //r1 = r1 + 1

b while //loop back to while branch

zero: //inside zero branch

mov r2, #0 //r2 = 0

mov r3, #0 //r3 = 0

b For //branch to For branch

For: //inside For branch

cmp r3, r1

bge end //branch to end if r3>=r1

blt inside //branch to inside if r3<r1

inside: //inside inside branch

ldrb r4, [r0,r3] //load register r4 with whatever is in mem location r0+r3

add r3, r3, #1 //r3 = r3 + 1

mul r4, r4, r3 //r4 = r4 \* r3

add r2, r2, r4 //r2 = r2 + r4

b For //loop back to For branch

end: //inside end branch

mov r3, #100 //r3 = 100

mov r4, #10 //r4 = 10

mul r3, r3, r4 //r3= r3\*r4

add r3, r3, #1 //r3= r3+1

udiv r1, r2, r3 //r1 = r2/r3

mul r1, r1, r3 //r1 = r1\*r3

sub r2, r2, r1 //r2=r2-r1

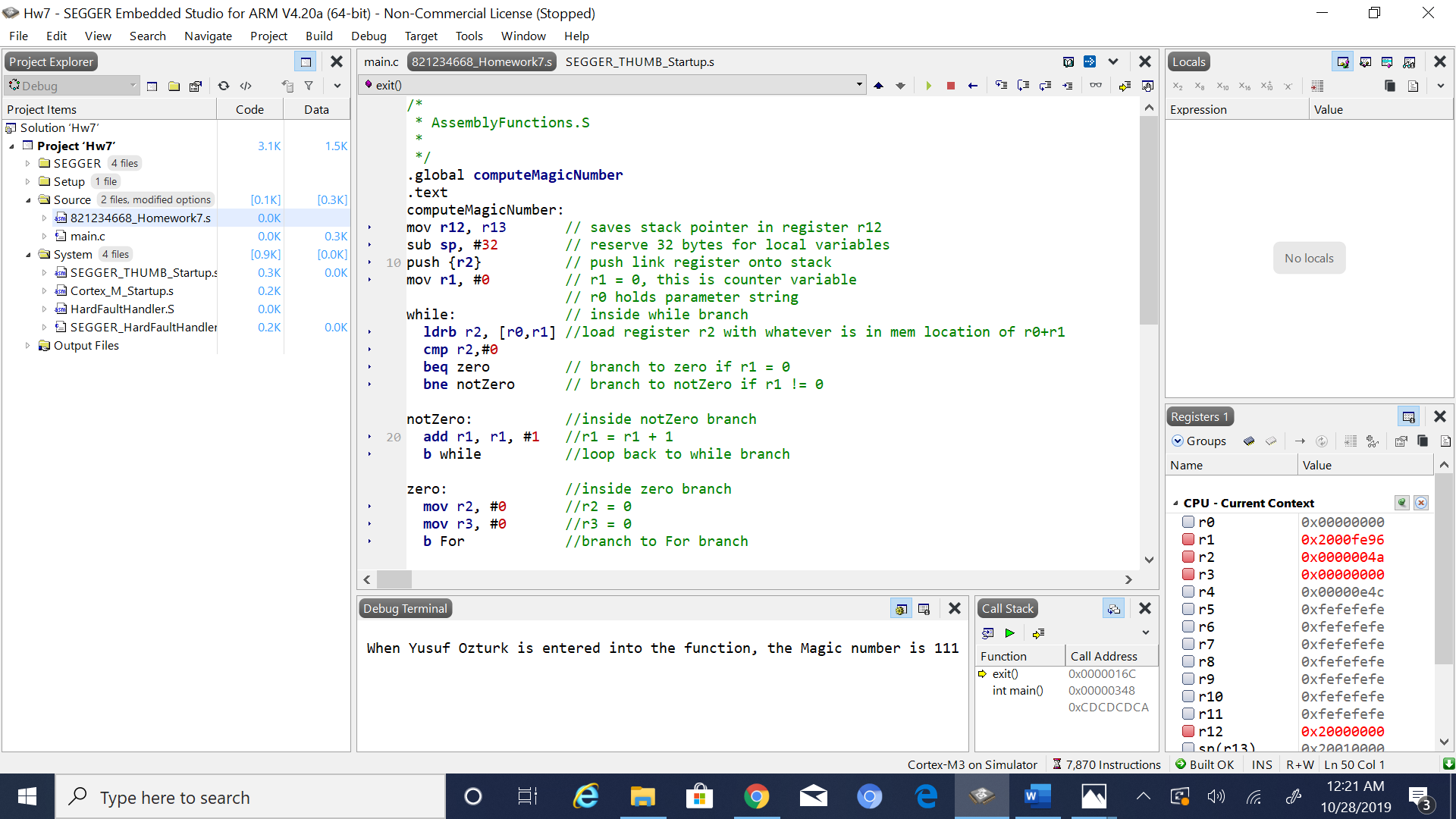
mov r0, r2 //return r2

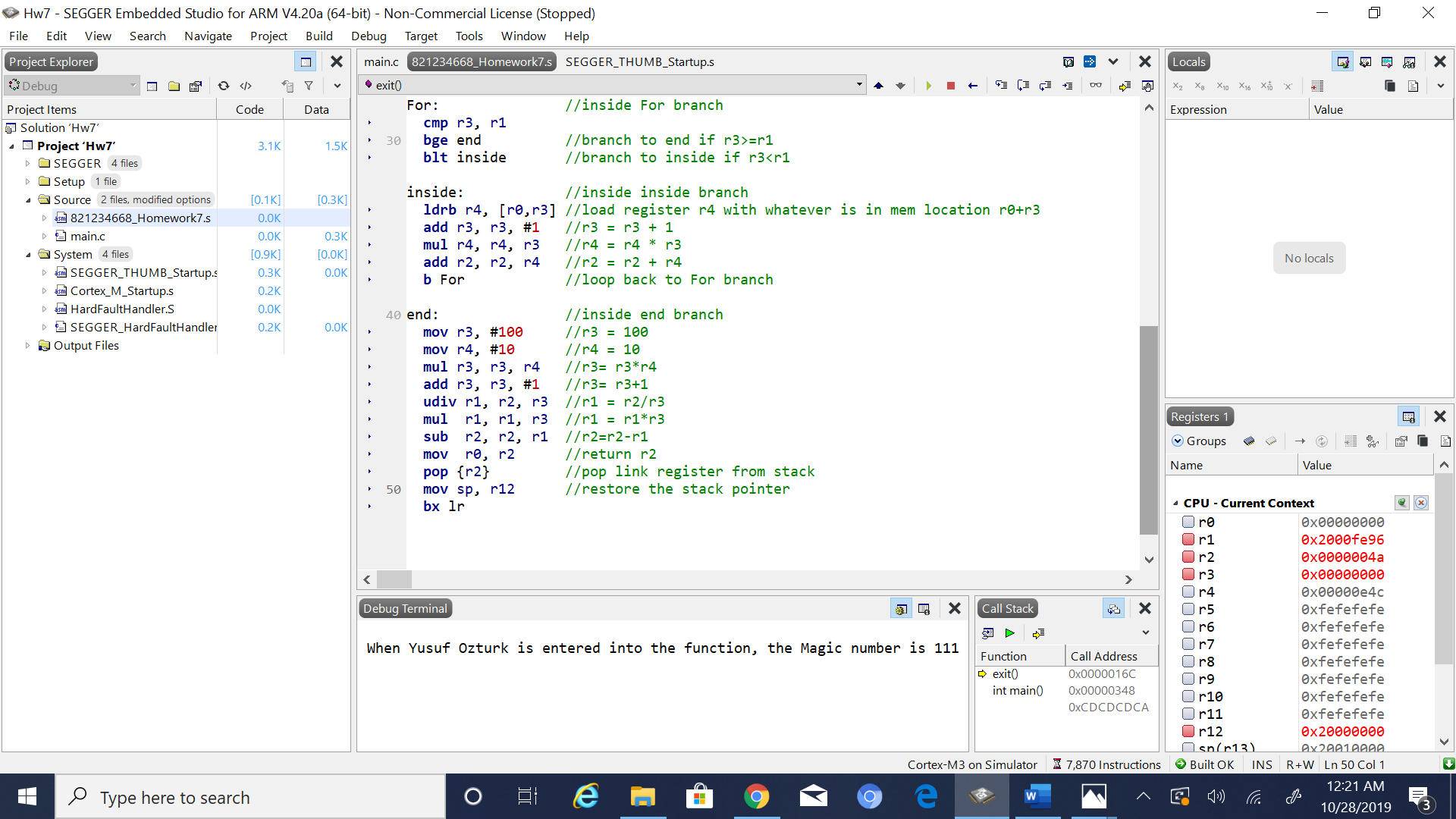
pop {r2} //pop link register from stack

mov sp, r12 //restore the stack pointer

bx lr

4\*. Screen capture of the code and the resulting display(s)





5. Conclusion (if applicable)

6\*. References.

Ken Arnolds slides on assembly coding