**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.*

10/21/2019

[Homework 6]

[Hw6]

Ckick below to enter/change your Name and RedID

Ethan Nagelvoort, 821234668

**Content**

(\* - Mandatory)

1\*. Description of the problem/method

For the first function, isNumber, I used ldrb to store r0’s bits into r1. Then I checked if this is greater than or equal to 0x30(1 on the ascii table). If it was then it would branch to if1, if not then it would branch to exit, where 0 would be returned. In the if1 branch I checked if r1 was less than or equal to 0x39(9 on the ascii table). If it was, then it would branch to if2, where 1 would be returned. If it wasn’t, then the function would branch to exit.

For the compare function, I used cmp on the two registers that contained the a and the b values. The function will then split into three branches, end if a>b, end2 if a<b and end3 if a=b. In end, 1 is returned, in end2, -1 is returned, and in end3, 0 is returned.

For the countOnes function, I place the contents of r0 into r1 and set r2 to 0, this will be used a counter variable. The function then goes into a while branch where it branches to loop if r1 != 0 and branches to done if r1=0. In the loop branch, I set r3 = r1 & 1. I then add r3 to r2 and use lsr r1, r1, #1 to go to the next bit in number. The branch then branches back to while. In the done branch, the function returns r2.

For the returnHammingDistance function, I set firstparameter in r2 and set r1 = r2 ^ r1. This sets r1 to have 1’s in the places where firstparameter and secondparameter have differences in their binary values. I then set r3 to 0 and this will be a counter variable. I then have a branch called while2 which branches to loop2 if r1 != 0 and branches to finish if r1=0. In loop2, I set r4 = r1 & 1 and add this value to r3. I then do lsr r1, r1, #1 to go to the next bit in r1, the function then branches back to while2. In the finish branch, r3 is returned.

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

  isNumber:

  save stack pointer into register r12

  reserve 32 bytes of space for local variables

 push link register onto stack -- make sure you pop it out before you return

  load character bit into r1

  Use ascii table to see if character is greater than 0x30, use cmp

  if is goto branch if1

 if not branch to exit

  inside branch if1

  use ascii table to see if character is less than 0x39, use cmp

  if is goto branch if2

  if not branch to exit

  If inside branch if2

  character is a digit, so have r0 contain #1

  branch to exit2

  inside branch exit

  character is not a digit, so have r0 contain #0

  branch to exit2

  inside branch exit2

  pop link register from stack

  restore the stack pointer -- Please note stack pointer should be equal to the

        value it had when you entered the function .

 return from the function by copying link register into  program counter

  compare:

save stack pointer into register r12

 reserve 32 bytes of space for local variables

  push link register onto stack -- make sure you pop it out before you return

   r0 = a, r1=b

  put a in r2

  cmp a, b

  branch to end if a>b

  branch to end2 if a<b

  branch to end3 if a=b

 inside branch end

  return 1

  branch to end4

 inside branch end2

  r3 = 0

  r3 = r3 + (-1)

  return -1

  branch to end4

  inside branch end3

  return 0

  branch to end4

 inside end4

 pop link register from stack

 restore the stack pointer -- Please note stack pointer should be equal to the

      value it had when you entered the function .

  return from the function by copying link register into  program counter

  countOnes:

  save stack pointer into register r12

  reserve 32 bytes of space for local variables

 push link register onto stack -- make sure you pop it out before you return

                        r0 = int number

  r1 = r0/int number

  r2 = 0, r2 is counter variable

  inside while branch

  check if (r1 = 0)

  branch to done if r1==0

  branch to loop if r1!=0

  inside loop branch

  r3 = 1

  r3 = r0 & r3 or number = number & 1

  r2 = r2 + r3, adds 1 if r3 is 1 and 0 if r3 is 0

  r1 = r1 >> 1, done in order to loop through bits of integer

  branches back to while branch

  inside done branch

  return r2, the counter varialbe

  pop link register from stack

  restore the stack pointer -- Please note stack pointer should be equal to the

  value it had when you entered the function .

  return from the function by copying link register into  program counter

  returnHammingDistance:

  save stack pointer into register r12

  reserve 32 bytes of space for local variables

  push link register onto stack -- make sure you pop it out before you return

    r0 = firstparameter, r1 = secondparameter

  r2 = r0, firstparameter

  r1 = r2 ^ r1

  r3 = 0, this will hold the result

  inside while2 branch

  check if(r1 = 0)

  branch to finish if r1 = 0

  branch to loop2 if r1 != 0

  inside loop2 branch

 r4 = 1

  r4 = r1 & r4/1

  r3 = r3 + r4

  r1= r1 >> 1

  loop back to while2 branch

  inside finish branch

  r0 = r3, return result

  pop link register from stack

  restore the stack pointer -- Please note stack pointer should be equal to the

value it had when you entered the function .

 return from the function by copying link register into  program counter

3\*. C-code

/\*

\* AFunctions.s

\*

\* Please rename this file as Homework6\_yourReadID.s and submit only this file

\*/

.global isNumber

.data

// declare any global variables here

.text

isNumber:

mov r12,r13 // save stack pointer into register r12

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

push {lr} // push link register onto stack -- make sure you pop it out before you return

ldrb r1,[r0] //load character bit into r1

cmp r1,#0x30 //Use ascii table to see if character is greater than 0x30, use cmp

BGE if1 //if is goto branch if1

b exit //if not branch to exit

if1: //inside branch if1

cmp r1,#0x39 //use ascii table to see if character is less than 0x39, use cmp

BLE if2 //if is goto branch if2

b exit //if not branch to exit

if2: //inside branch if2

mov r0, #1 //character is a digit, so have r0 contain #1

b exit2 //branch to exit2

exit: //inside branch exit

mov r0, #0 //character is not a digit, so have r0 contain #0

b exit2 //branch to exit2

exit2: //inside branch exit2

pop {r1} // pop link register from stack

mov sp,r12 // restore the stack pointer -- Please note stack pointer should be equal to the

// value it had when you entered the function .

mov pc,lr // return from the function by copying link register into program counter

.global compare

.data

// declare any global variables here

.text

compare:

mov r12,r13 // save stack pointer into register r12

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

push {lr} // push link register onto stack -- make sure you pop it out before you return

//r0 = a, r1=b

mov r2, r0 //put a in r2

cmp r2, r1 //cmp a, b

BGT end //branch to end if a>b

BLT end2 //branch to end2 if a<b

BEQ end3 //branch to end3 if a=b

end: //inside branch end

mov r0,#1 //return 1

b end4 //branch to end4

end2: //inside branch end2

mov r3,#0 //r3 = 0

add r3,r3,#-1 //r3 = r3 + (-1)

mov r0, r3 //return -1

b end4 //branch to end4

end3: //inside branch end3

mov r0,#0 //return 0

b end4 //branch to end4

end4: //inside end4

pop {r1} // pop link register from stack

mov sp,r12 // restore the stack pointer -- Please note stack pointer should be equal to the

// value it had when you entered the function .

mov pc,lr // return from the function by copying link register into program counter

.global countOnes

.data

// declare any global variables here

countOnes:

mov r12,r13 // save stack pointer into register r12

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

push {lr} // push link register onto stack -- make sure you pop it out before you return

// r0 = int number

mov r1, r0 // r1 = r0/int number

mov r2, #0 // r2 = 0, r2 is counter variable

while: //inside while branch

cmp r1, #0 //check if (r1 = 0)

beq done //branch to done if r1==0

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

loop: //inside loop branch

mov r3, #1 //r3 = 1

and r3, r1, r3 //r3 = r0 & r3 or number = number & 1

add r2, r2, r3 //r2 = r2 + r3, adds 1 if r3 is 1 and 0 if r3 is 0

lsr r1, r1, #1 // r1 = r1 >> 1, done in order to loop through bits of integer

b while // branches back to while branch

done: //inside done branch

mov r0, r2 // return r2, the counter varialbe

pop {r2} // pop link register from stack

mov sp,r12 // restore the stack pointer -- Please note stack pointer should be equal to the

// value it had when you entered the function .

mov pc,lr // return from the function by copying link register into program counter

.global returnHammingDistance

.data

// declare any global variables here

returnHammingDistance:

mov r12,r13 // save stack pointer into register r12

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

push {lr} // push link register onto stack -- make sure you pop it out before you return

// r0 = firstparameter, r1 = secondparameter

mov r2, r0 // r2 = r0, firstparameter

eor r1, r2, r1 // r1 = r2 ^ r1, helps determine which bits are different between the two

mov r3, #0 // r3 = 0, this will hold the result

while2: //inside while2 branch

cmp r1, #0 //check if(r1 = 0)

beq finish //branch to finish if r1 = 0

bne loop2 //branch to loop2 if r1 != 0

loop2: //inside loop2 branch

mov r4, #1 //r4 = 1

and r4, r1, r4 //r4 = r1 & r4/1

add r3, r3, r4 //r3 = r3 + r4

lsr r1, r1, #1 //r1= r1 >> 1

b while2 //loop back to while2 branch

finish: //inside finish branch

mov r0, r3 //r0 = r3, return result

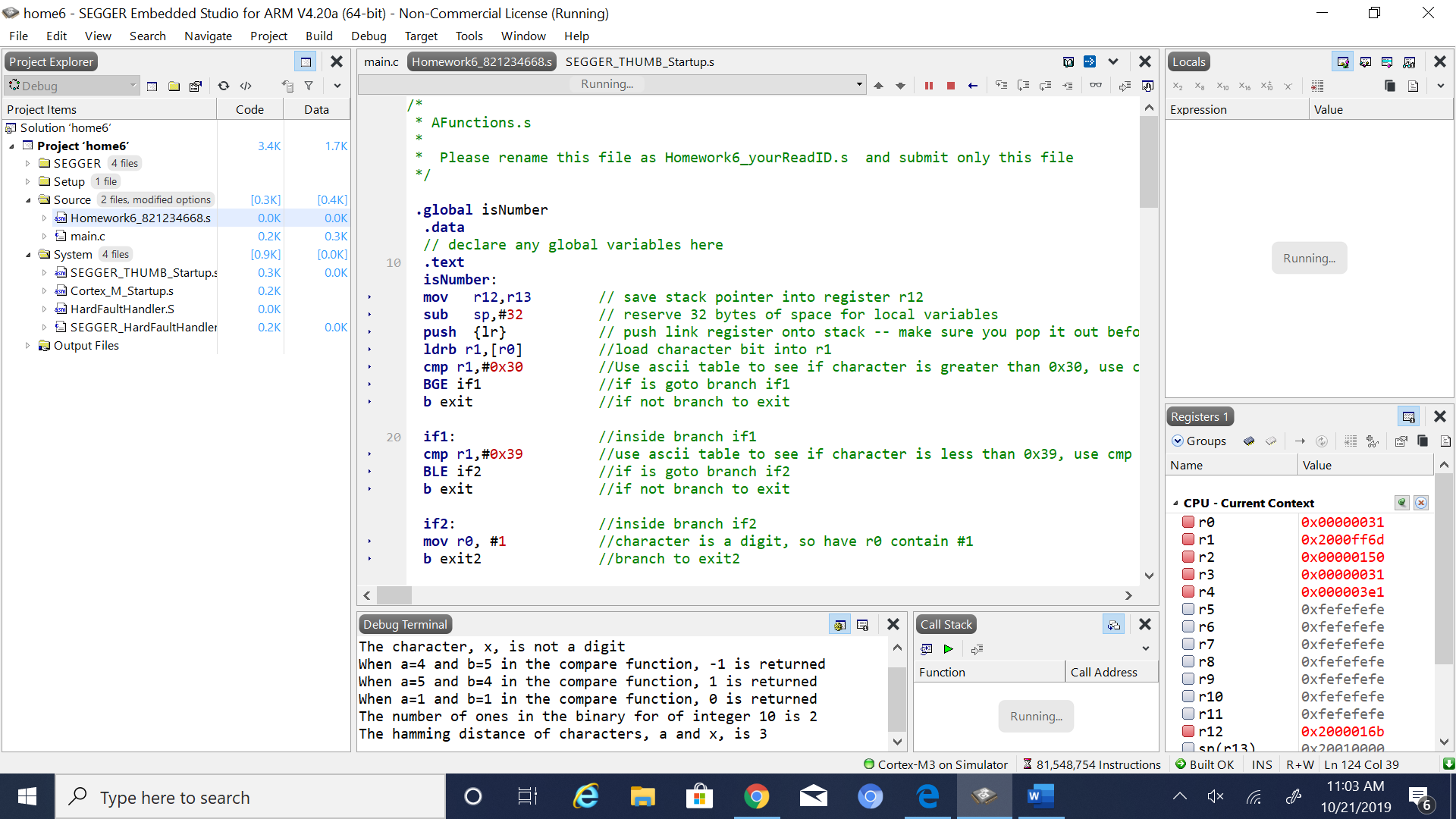
pop {r4} // pop link register from stack

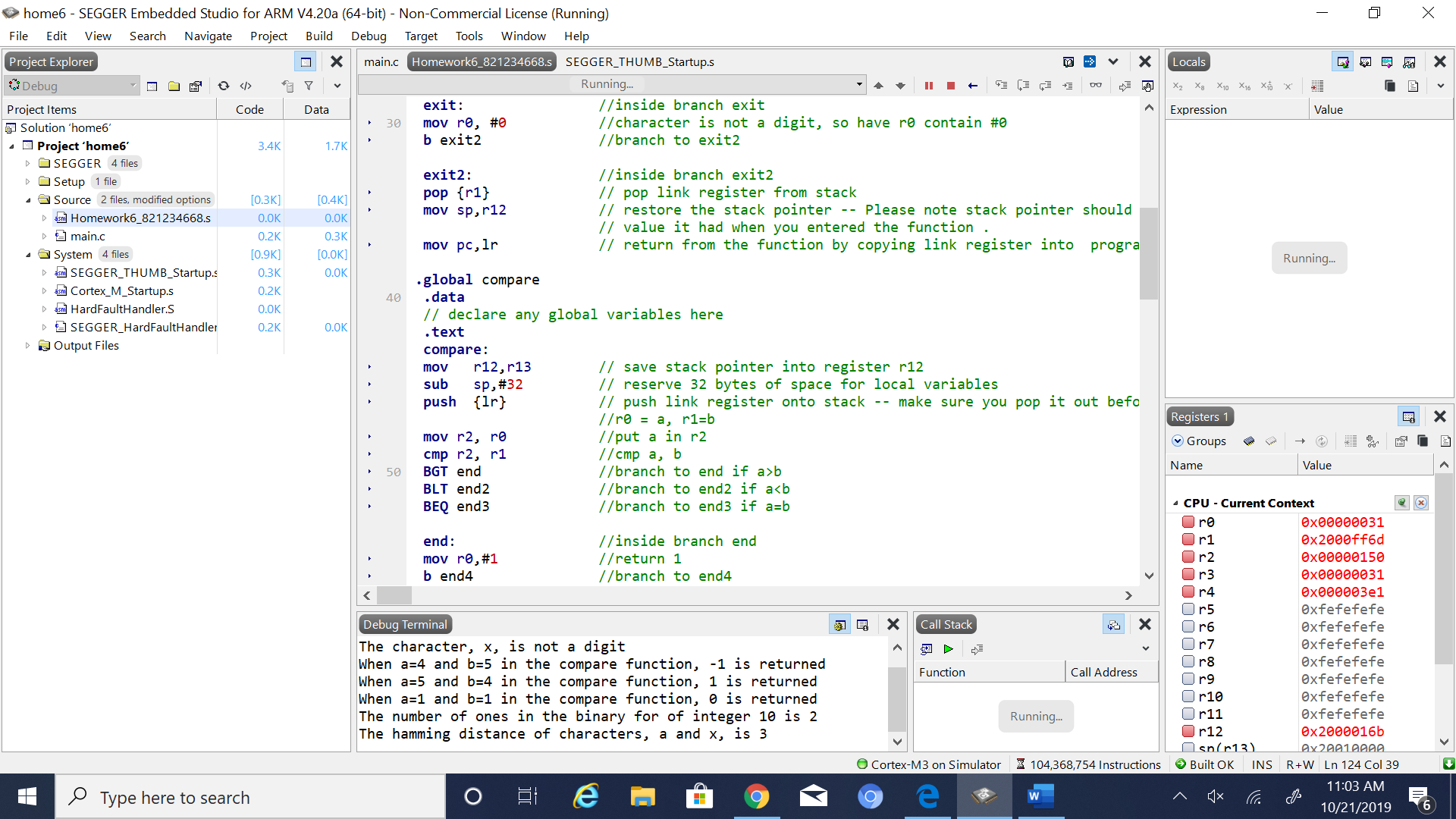
mov sp,r12 // restore the stack pointer -- Please note stack pointer should be equal to the

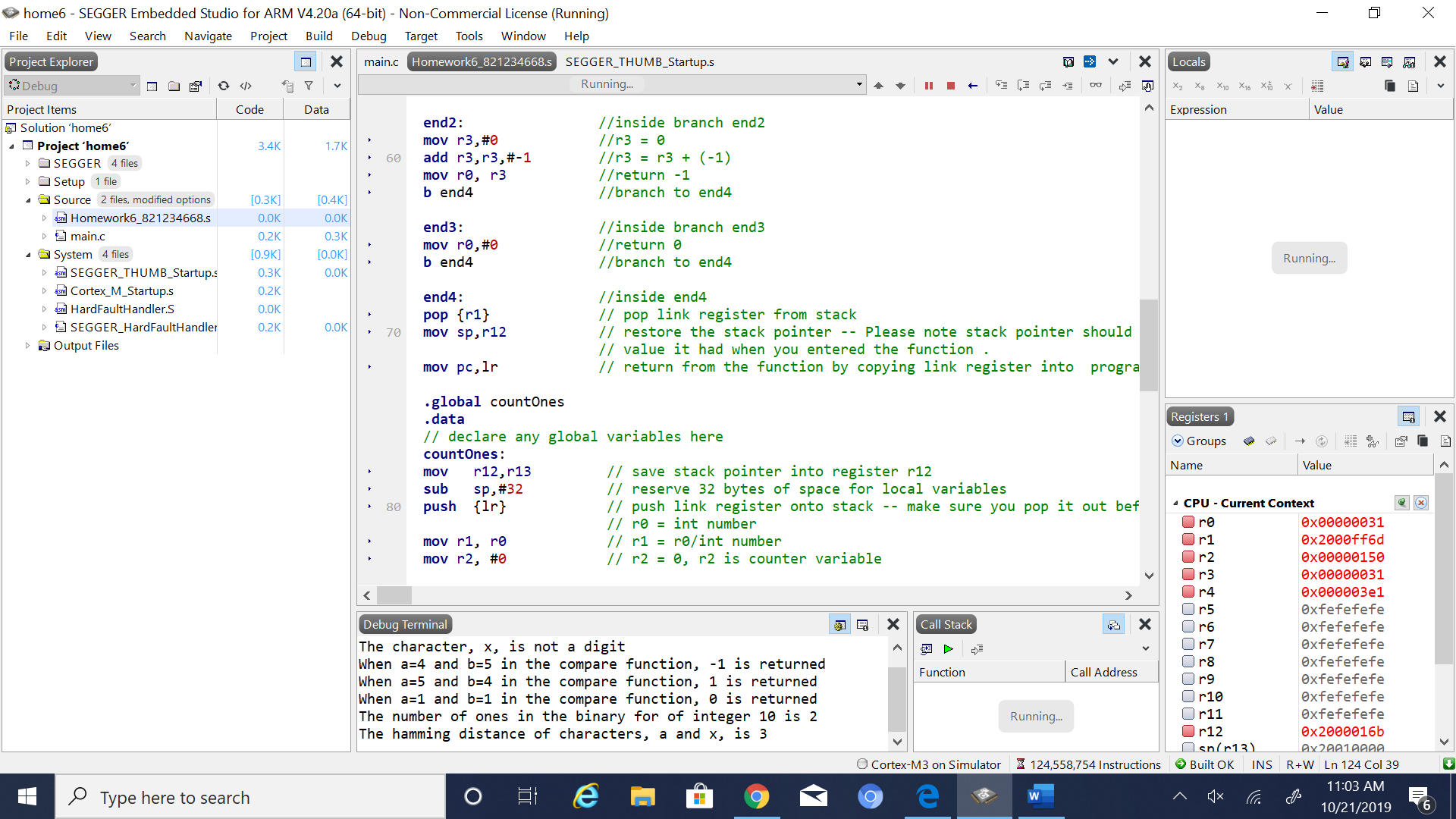
// value it had when you entered the function .

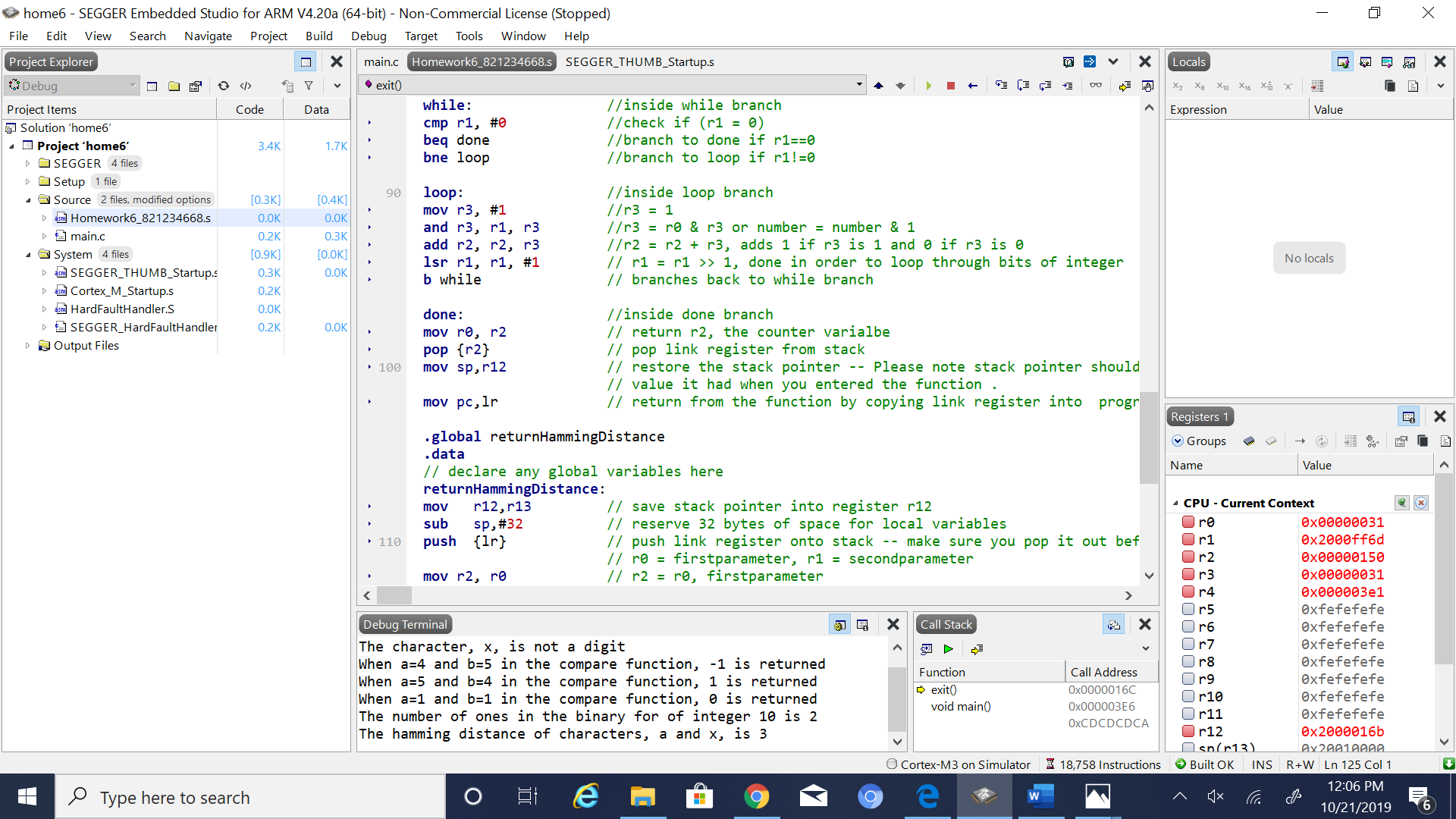
mov pc,lr // return from the function by copying link register into program counter

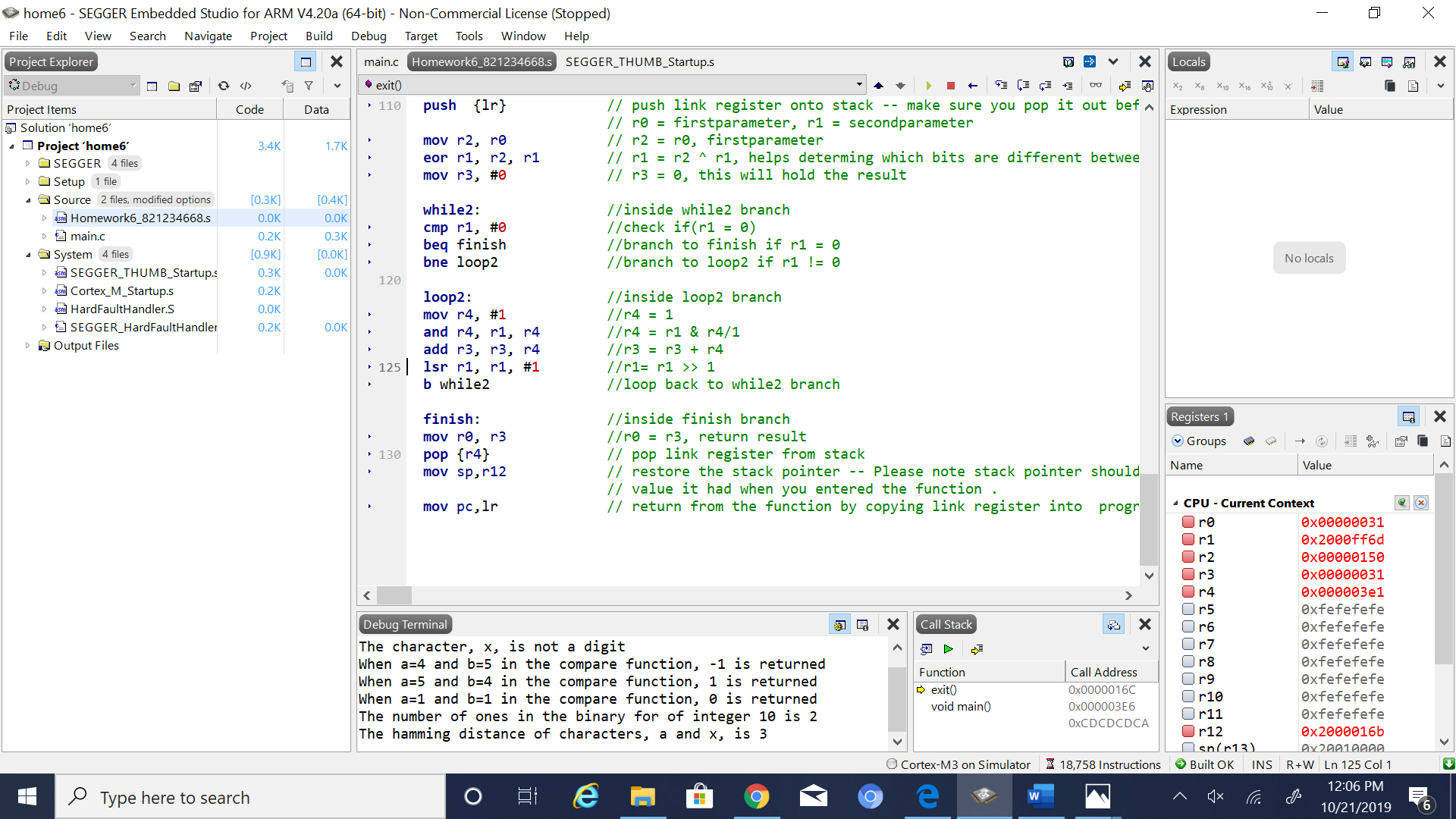
4\*. Screen capture of the code and the resulting display(s)











5. Conclusion (if applicable)

6\*. References.

Ken Arnolds PowerPoint slides on arm coding