**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.*

11/8/2019

Homework 8

hw8

Ckick below to enter/change your Name and RedID

Ethan Nagelvoort, 821234668

**Content**

(\* - Mandatory)

1\*. Description of the problem/method

For the post fix function, I have a while branch that checks if r0 equals any of the ascii values of (,), or any of the operators. If it is not equal to any of this, I store r0 into r1 and increment r0 and r1. R0 will now point to the next character. If r0 equals ( , then I increment r0. If r0 equals ), then another while branch occurs that ends until r0 equals ( which then it will branch back to the first while branch. In this while branch, I pop the stack onto r2, then increment r1 and r0. If r0 is an operator then it branches to operators, which then checks if r0 is less than r2. If so then it branches to order, if not then it branches to while. In c2, I pop stack onto r2, have r1=r2, increment r1, and branch back to operators. In the c1 branch, I check if r2 is equal to +, if so then branch to while. In c1, I also check if r2 does not equal -, if not then branch to finish. In the first while branch, I also check if r0 is equal to 0, which checks if r0 is at the end of the string. If so then it branches to finish, which leads to the end of the function.

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

// declare any global variables here

toPostFix:

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

Inside while branch:

brach to finish if r0 is at end of char array

branch to rightP if r0 equals ascii value of (

branch to leftP if r0 equals ascii vlaue of )

branch to operators if r0 equal ascii value of \*

branch to operators if r0 equal ascii value of +

branch to operators if r0 equal ascii value of -

branch to operator if r0 equal ascii value of /

branch to NoOperator if r0 is not equal to any of the operators

Inside NoOperator branch:

store whats in r0 into r1

Increment r0

Increment r1

branch back to while

iniside leftP branch:

push r0 onto the stack

increment r0

branch back to while

inside rightP branch:

pop stack into r2

if r2 is equal to ascii value of ( then branch to while

store whats in r0 into r1

increment r0

increment r1

branch back to rightP

inside operators branch:

pop stack onto r2

cmp r2,r0

branch to order if r2<r0

branch to while if r2>r0

inside order branch:

branch to c1 if r2 = \* value

branch to c2 if r2 != / value

inside c1 branch:

branch to while if r2 = + vlaue

branch to finish if r2 != -

Inside c2 branch:

pop from stack into r2

move contents of r2 into r1

increment r1

branch back to operators

Inside finish branch

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

.global toPostFix

.data

// declare any global variables here

.text

toPostFix:

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

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

push {r2} // push link register onto stack -- make sure you pop it out before you return

while: //Inside while branch:

cmp r0, #0x00

beq finish //brach to finish if r0 is at end of char array

cmp r0, #0x28

beq leftP //branch to ( if r0 equals ascii value of (

cmp r0, #0x29

beq rightP // branch to ) if r0 equals ascii vlaue of )

cmp r0, #0x2A

beq operators //branch to operators if r0 equal ascii value of \*

cmp r0, #0x2B

beq operators //branch to operators if r0 equal ascii value of +

cmp r0, #0x2D

beq operators //branch to operators if r0 equal ascii value of -

cmp r0, #0x2F

beq operators //branch to operator if r0 equal ascii value of /

bne NoOperator //branch to NoOperator if r0 is not equal to any of the operators

NoOperator: //Inside NoOperator branch:

str r0,[r1] //store whats in r0 into r1

add r0,r0,#1 //Increment r0

add r1,r1,#1 //Increment r1

b while //branch back to while

leftP: //iniside leftP branch:

push{r0} //push r0 onto the stack

add r0,r0,#1 //increment r0

b while //branch back to while

rightP: //inside rightP branch:

pop{r2} //pop stack into r2

cmp r2,#0x28

beq while //if r2 is equal to ascii value of ( then branch to while

str r0,[r1] //store whats in r0 into r1

add r0,r0,#1 //increment r0

add r1,r1,#1 //increment r1

b rightP //branch back to rightP

operators: //inside operators branch:

pop{r2} //pop stack onto r2

cmp r2,r0

blt order //branch to order if r2<r0

bgt while //branch to while if r2>r0

order: //inside order branch:

cmp r2, #0x2A

beq c1 //branch to c1 if r2 = \* value

cmp r2, #0x2F

bne c2 //branch to c2 if r2 != / value

c1: //inside c1 branch:

cmp r2, #0x2B

beq goBack //branch to goBack if r2 = + vlaue

cmp r2, #0x2D

bne finish //branch to finish if r2 != -

c2: //Inside c2 branch:

pop{r2} //pop from stack into r2

mov r1,r2 //move contents of r2 into r1

add r1,r1, #1 //increment r1

b operators //branch back to operators

finish: //Inside finish branch

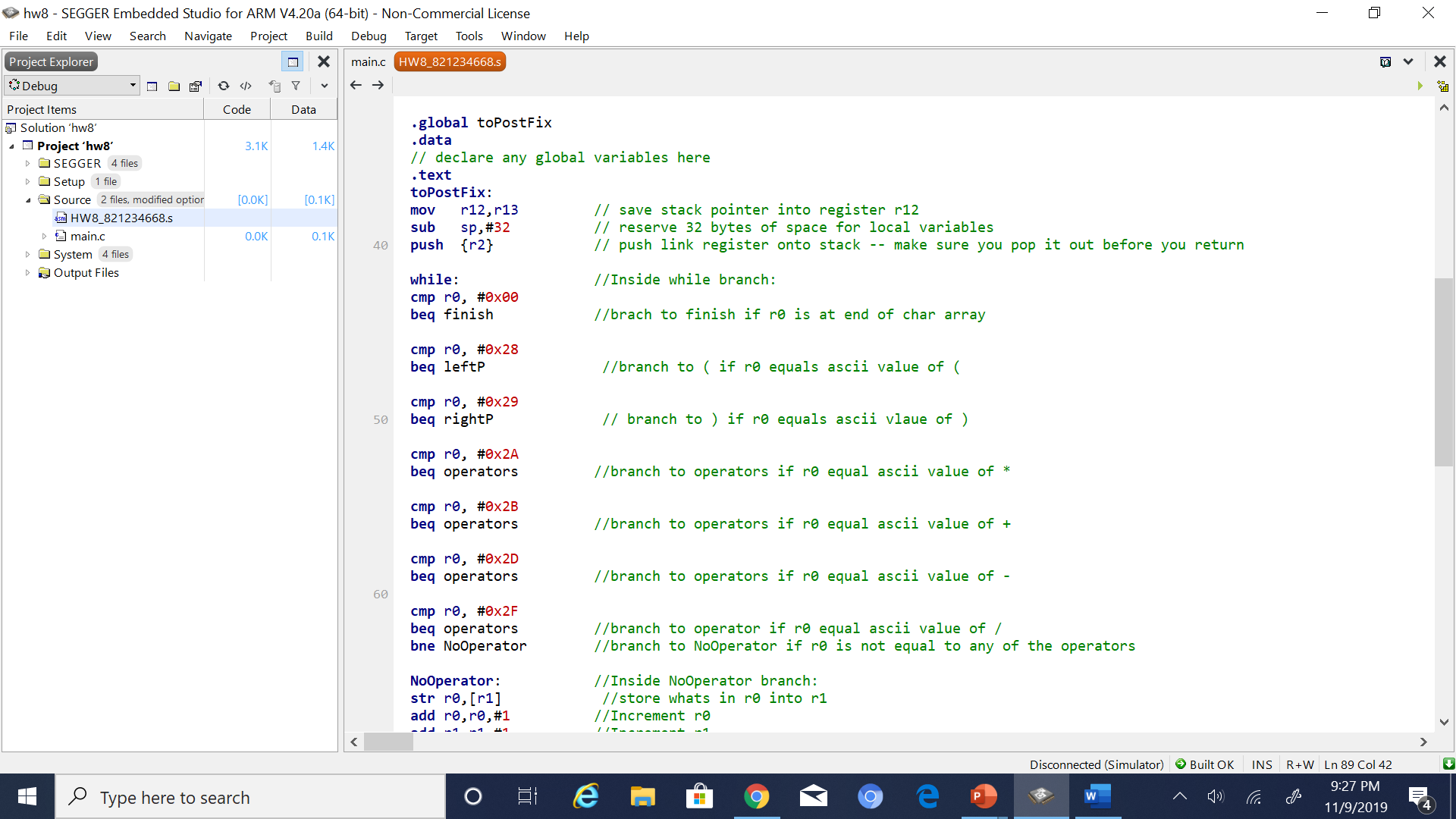
pop {r2} // pop link register from stack

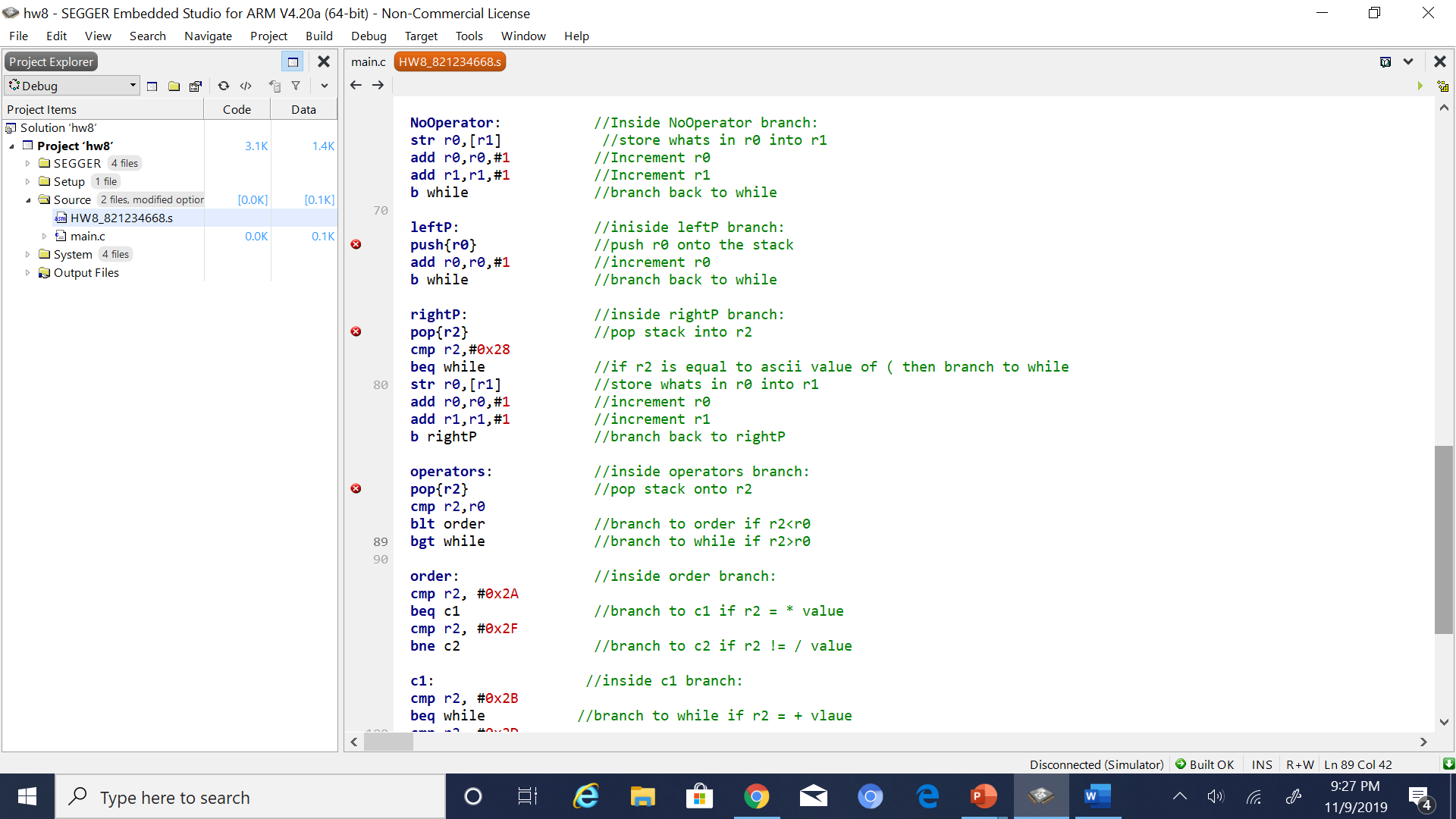
mov sp,r12 // restore the stack pointer -- Please note stack pointer should be equal to the

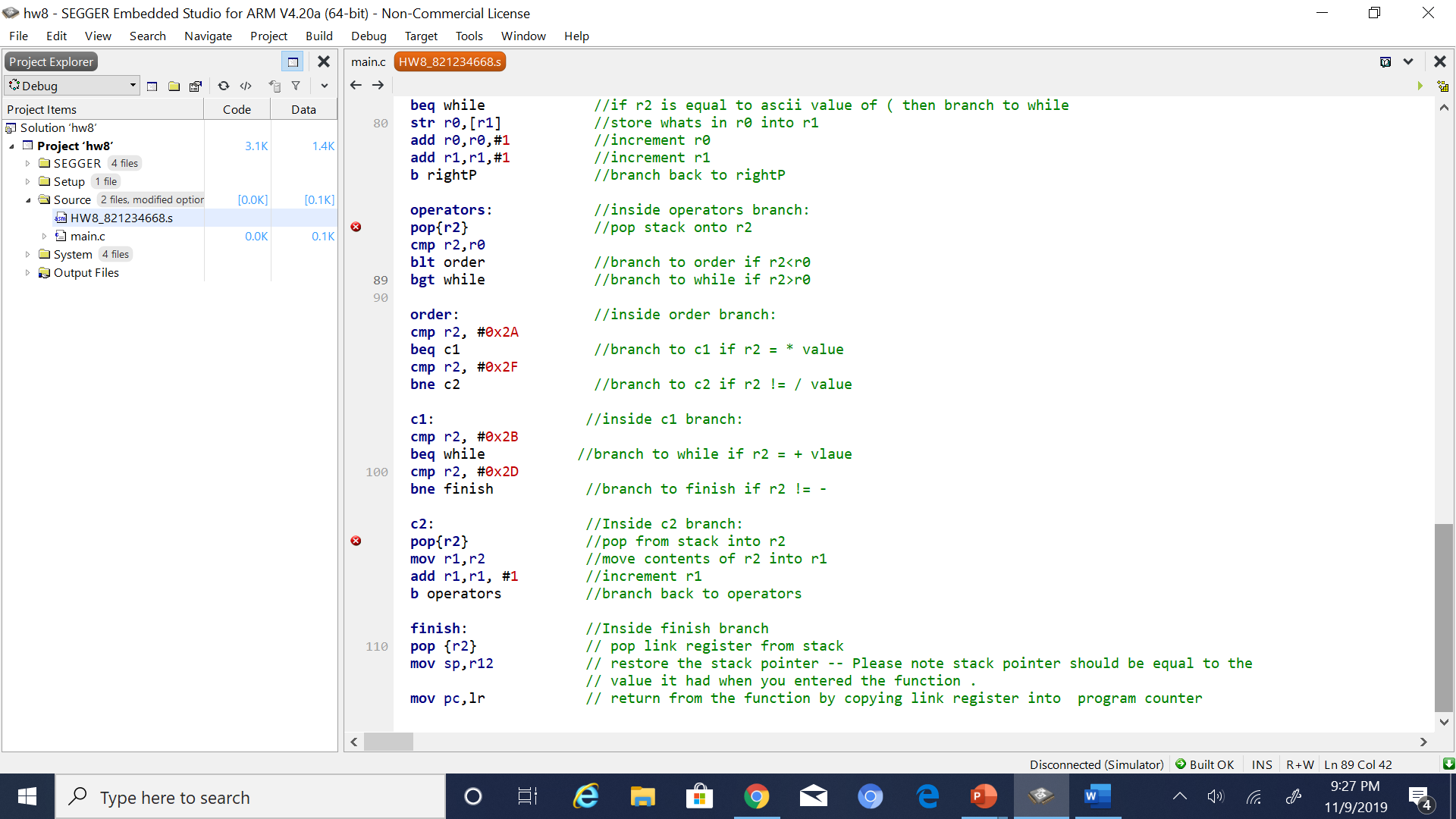
// value it had when you entered the function .

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

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







5. Conclusion (if applicable)

6\*. References.

Ken arnolds slides on arm code