**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.*

9/28/2019

[Homework #4]

[Hw4]

Ckick below to enter/change your Name and RedID

Ethan Nagelvoort, 821234668

**Content**

(\* - Mandatory)

1\*. Description of the problem/method

I started making my function class by creating a union, called fpr, that represents the structure of a float. In the union I had a float f that represents the float that is used in the functions and a structure, with all the bit fields that are used to represent a float: the sign bit, the exponent, and the mantissa. I put the struct in the order of mantissa, exponent, and sign because my computer is in little endian. After I created my union, I moved onto the first function, bitwisedFloatCompare(float number1,float number2). This function takes in two floats and determines which one is greater. It returns an int; 1 if number1 is bigger than number2, -1 if number2 is bigger than number1, and 0 if the two floats are equal. I started by creating two fprs named f1 and f2 and set the f’s of f1 to number1 and f2’s f to number2. I also declared an int compare equal to 0. If the function does not go through any of the following code, then it would show that the two floats are equal. I then checked the sign bits of the two floats. I first checked if f1’s sign bit was bigger than f2’s and if it was compare would equal -1 because then f1 would be negative and f2 would be positive. Then I checked the reverse and then I would set compare to equal 1. I did these through an if and an if else statement. Then in the else part of the statement, I checked the exponent part of the floats. I did this through a for statement that loops through the 8 bits of the exponent part. I used i as the int in the for statement. I then used an if statement to check if f1 exponent bits shifted i times is bigger than f2’s exponent bits shifted i times. This checks to see where there is a difference in the two exponent bits. If there was, compare would equal 1 and then there would be a break statement to end the loop. I then did the reverse in an else if statement and had compare equal to -1 since f2 would be bigger here and another break statement would be placed. Then in the else statement I checked the mantissa portion of the two floating point numbers. I did this the same way as the exponent part but instead of looping through 8 bits, I looped through 24. At the end of the function I returned compare.

I then moved on a created the second function, printFloatRepresentation(float number) which returns void and prints out a float in a specific form. For example, if -71 was passed through the function, (1) 10000101 (1)00011100000000000000000 would be printed out. The first part of what printed is the sign bit, then the exponent, then the assumed bit with the significand/mantissa. I did this by first creating fpr f1 and setting f1’s f to number and then I printed out the sign bit of f1. I then declared an int i to 8 and then created a for loop statement that looks like this: for(int i--; i>=0; i--). This for statement is used to accurately go through the floats 8 bits. In the loop, I used an if statement to check if f1’s exponent bits shifted i times are equal to 1 by using & 1. If this is true , then 1 would be printed out. Then in the else statement, I had 0 printed out. Then, after the loop, I declared another int called ii which is equal to 23. I then created a for loop with ii, the same way as the for loop before. This loop works the same way as the exponent loop, except this one loops 23 times and checks the mantissa/significant/fraction part of the float. Then when this loop is finished, the bits of the floating point number should have been all printed out.

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

3\*. C-code

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//

// FloatCompare

//

// Description:Accepts two float numbers and compares them bitwise based

// on floating point representations. This function will have

// to convert the given numbers into IEEE floating

// representation and then do bitwise comparison

// Preconditions:two input arguments are passed

// Postconditions:Returns 1,-1 or 0 based on the comparison

// 1 if number1>number2

// -1 if number2>number1

// 0 if equal

// Calls: N/A

// Called by: main

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Struct is used in order to create bit fields for the mantissa, exponent, and sign

//This allows us to use bit operators in the functions

//Union is used to combine a float and the struct together

typedef union{

float f;

struct

{

unsigned int m : 23;//mantissa

unsigned int e : 8; //exponent

unsigned int s : 1; //sign

} sem;

}fpr;

int bitwisedFloatCompare(float number1,float number2)

{

//Write the function to compare and return the corresponding value

int compare = 0;//compare will stay 0 if the floats are equal

fpr f1;

fpr f2;

f1.f = number1;

f2.f = number2;

/\*compares the sign bits of the two floats,

the float with the smallest sign bit is larger\*/

if(f1.sem.s > f2.sem.s)

{

compare = -1;

}

else if(f2.sem.s > f1.sem.s)

{

compare= 1;

}

else

{

/\*compares the exponent parts of the two floats,

the float with the bigger exponent is larger\*/

for(int i = 0; i<8; i++)

{

if((f1.sem.e << i) > (f2.sem.e << i))

{

compare = 1;

break;

}

else if((f2.sem.e << i) > (f1.sem.e << i))

{

compare = -1;

break;

}

else

{

/\*compares the mantissa/fraction parts of the two floats,

the float with the bigger exponent is larger\*/

for(int i=0; i<23; i++)

{

if((f1.sem.m << i) > (f2.sem.m << i))

{

compare = 1;

break;

}

else if((f2.sem.m << i) > (f1.sem.m << i))

{

compare = -1;

break;

}

}

}

}

}

return compare;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//

// FloatRepresentation

//

// Description:Accepts one float number and prints out the bit form of that

// floating point number. It is printed out as the sign bit

// first and then the exponent portion and then the assumed

// bit, combined with the significant.

// Preconditions:one input arguemnt is passed

// Postconditions:Prints out the Floating point number in its bit form.

// Prints the sign bit in parenthisis first and then

// the exponent bit and then the assumed bit in

// parenthisis with the significant.

//

// Calls: N/A

// Called by: main

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void printFloatRepresentation(float number)

{

fpr f1;

f1.f = number;

printf("(%d)\t",f1.sem.s);//prints sign bit

int i = 8;//number of bits in exponent

//prints exponent bit

for(i--; i>=0; i--)

{

int ex = f1.sem.e;

if( (ex >> i) & 1)

printf("1");

else

printf("0");

}

printf("\t(1)");

int ii = 23;//number of bits in mantissa

//prints significant/mantissa/fraction bits

for(ii--;ii>=0;ii--)

{

int man = f1.sem.m;

if((man >> ii) & 1)

printf("1");

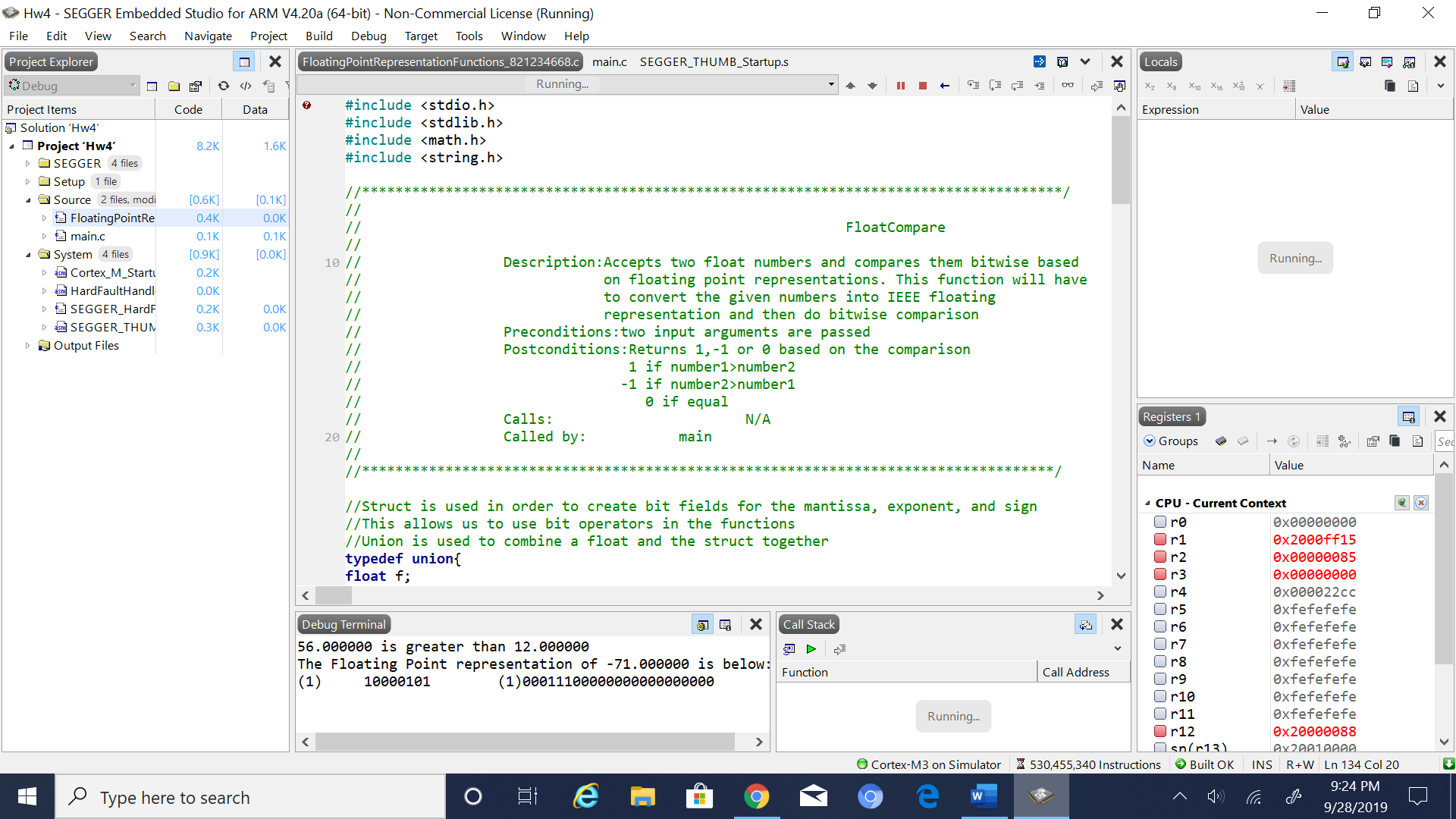
else

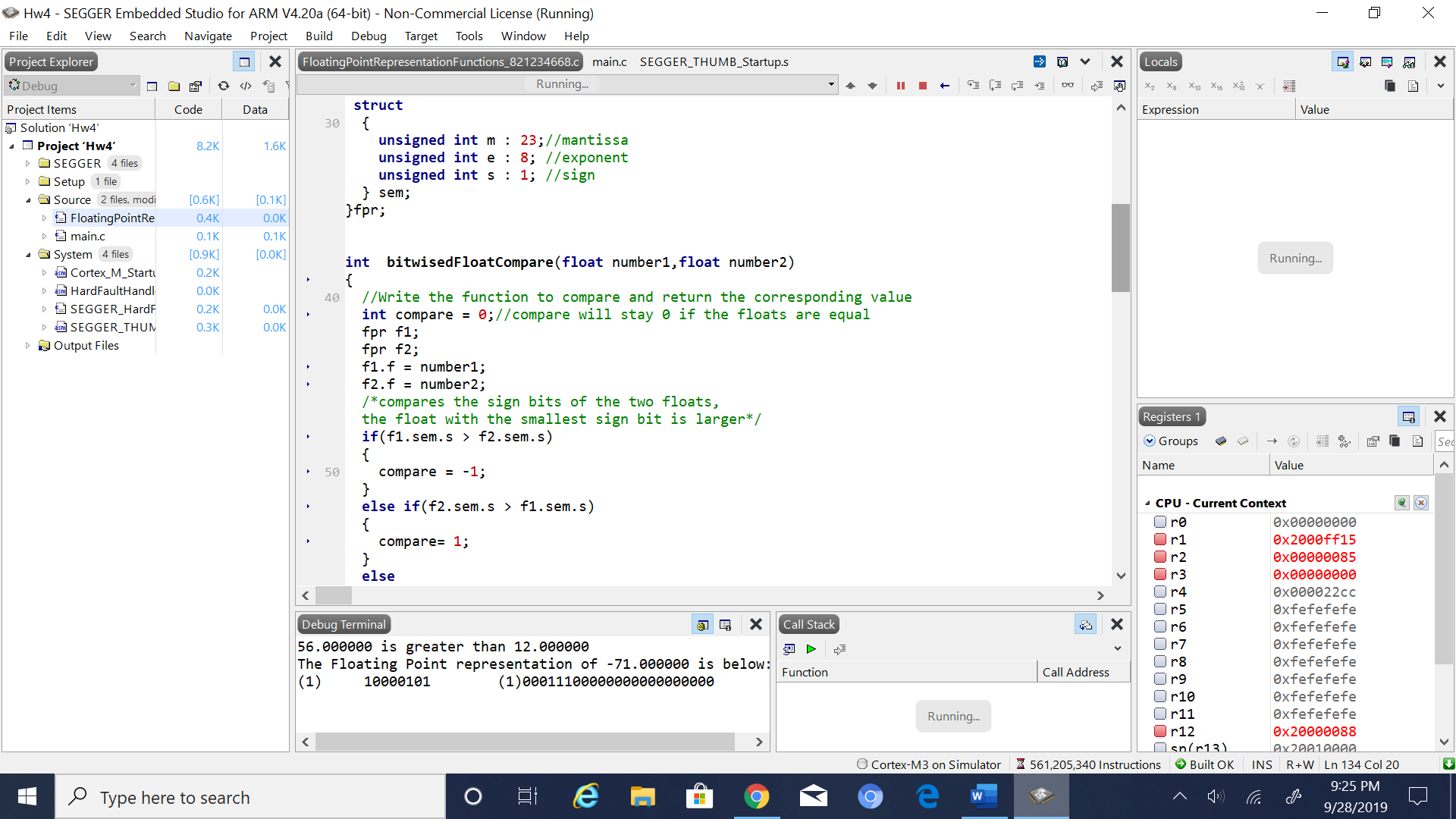
printf("0");

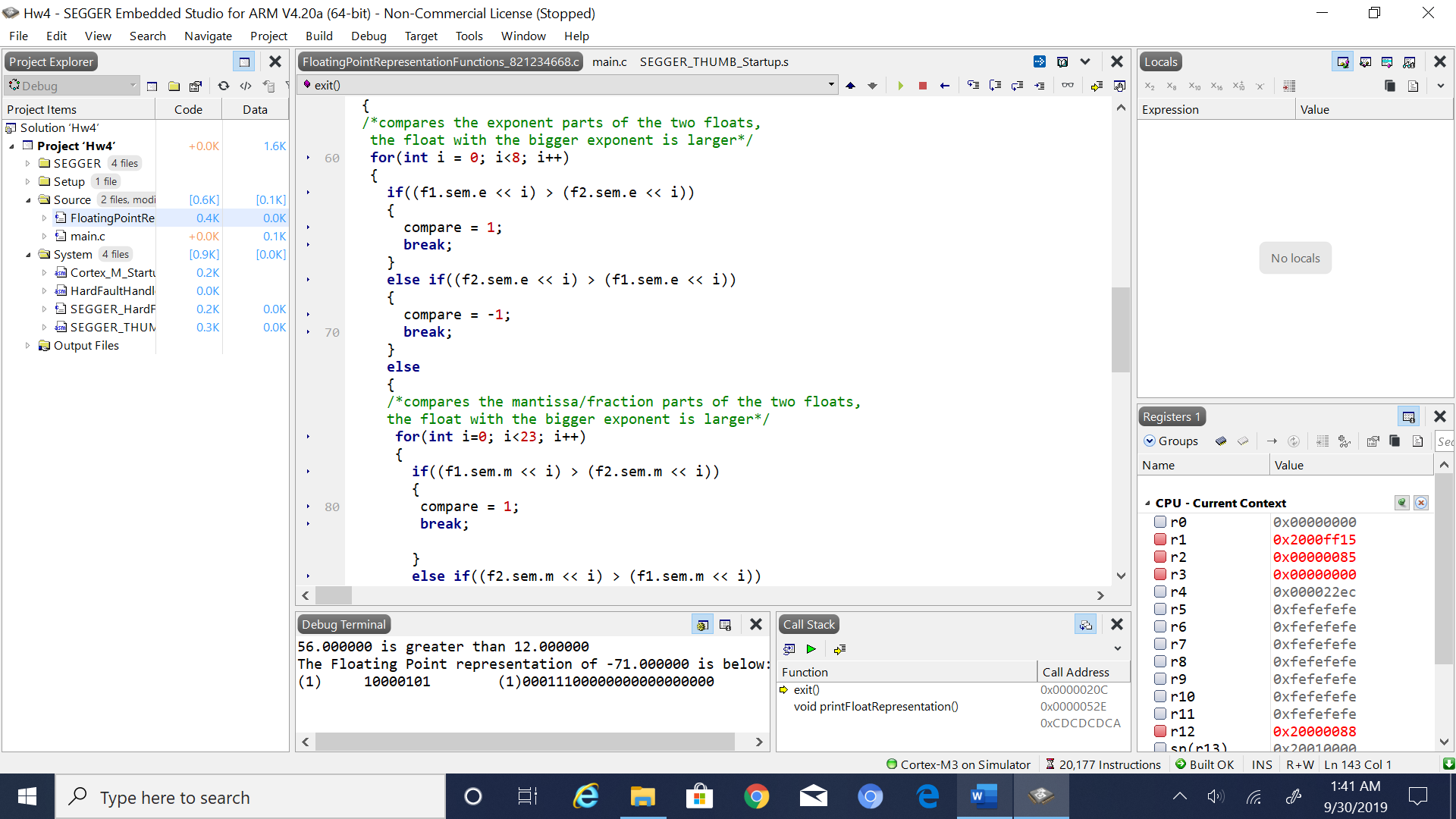
}

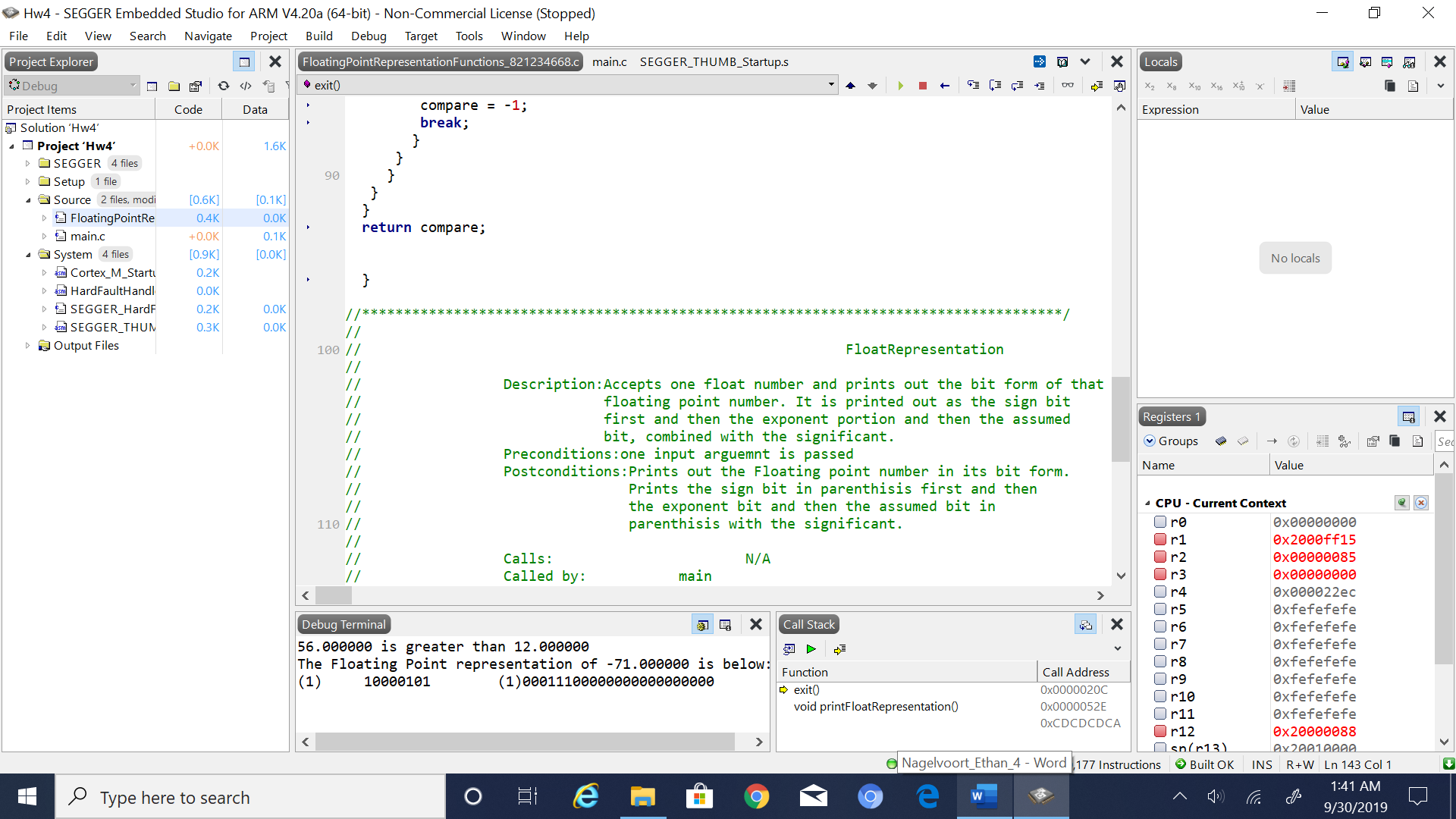
}

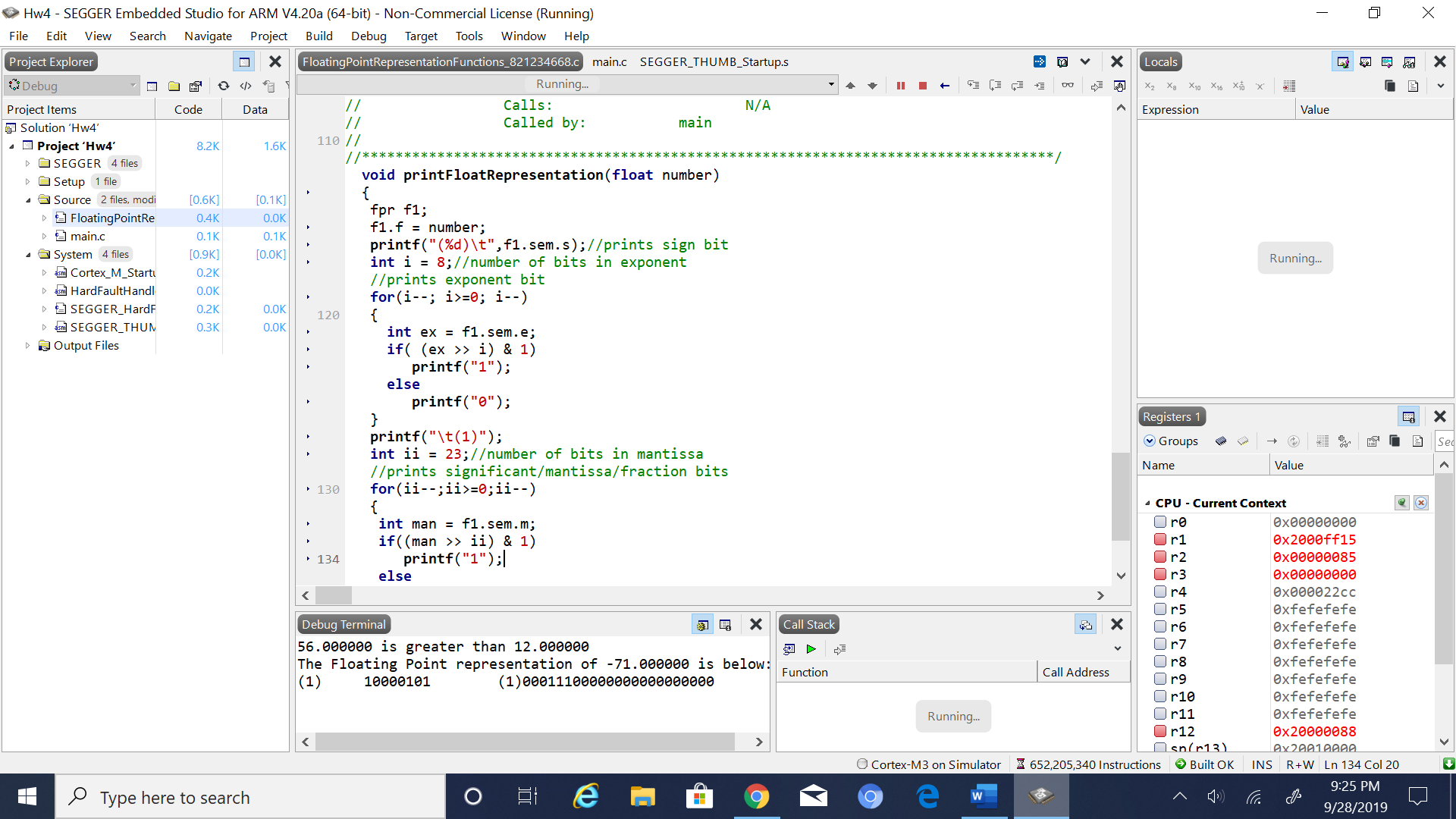
4\*. Screen capture of the code and the resulting display(s)

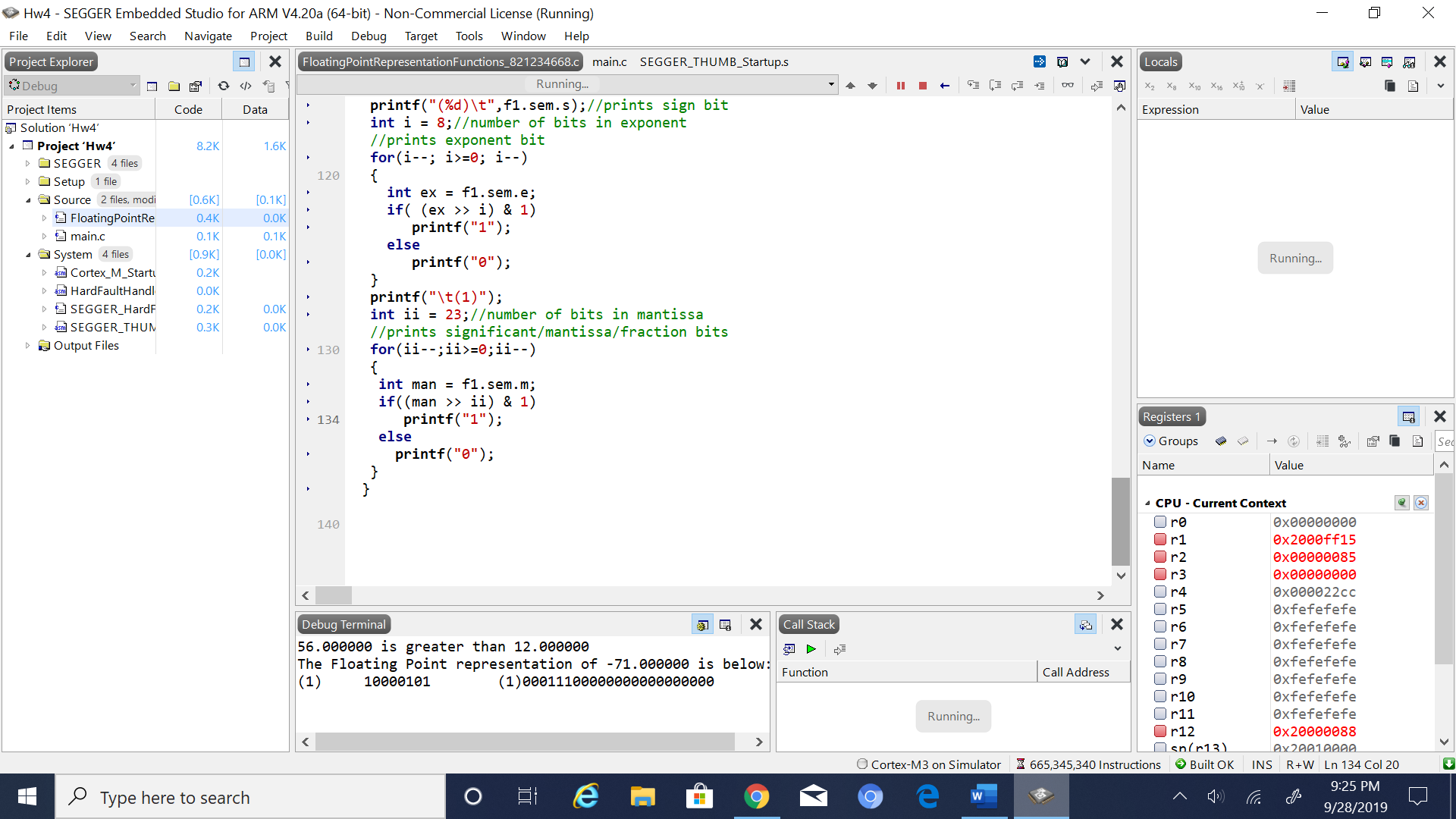


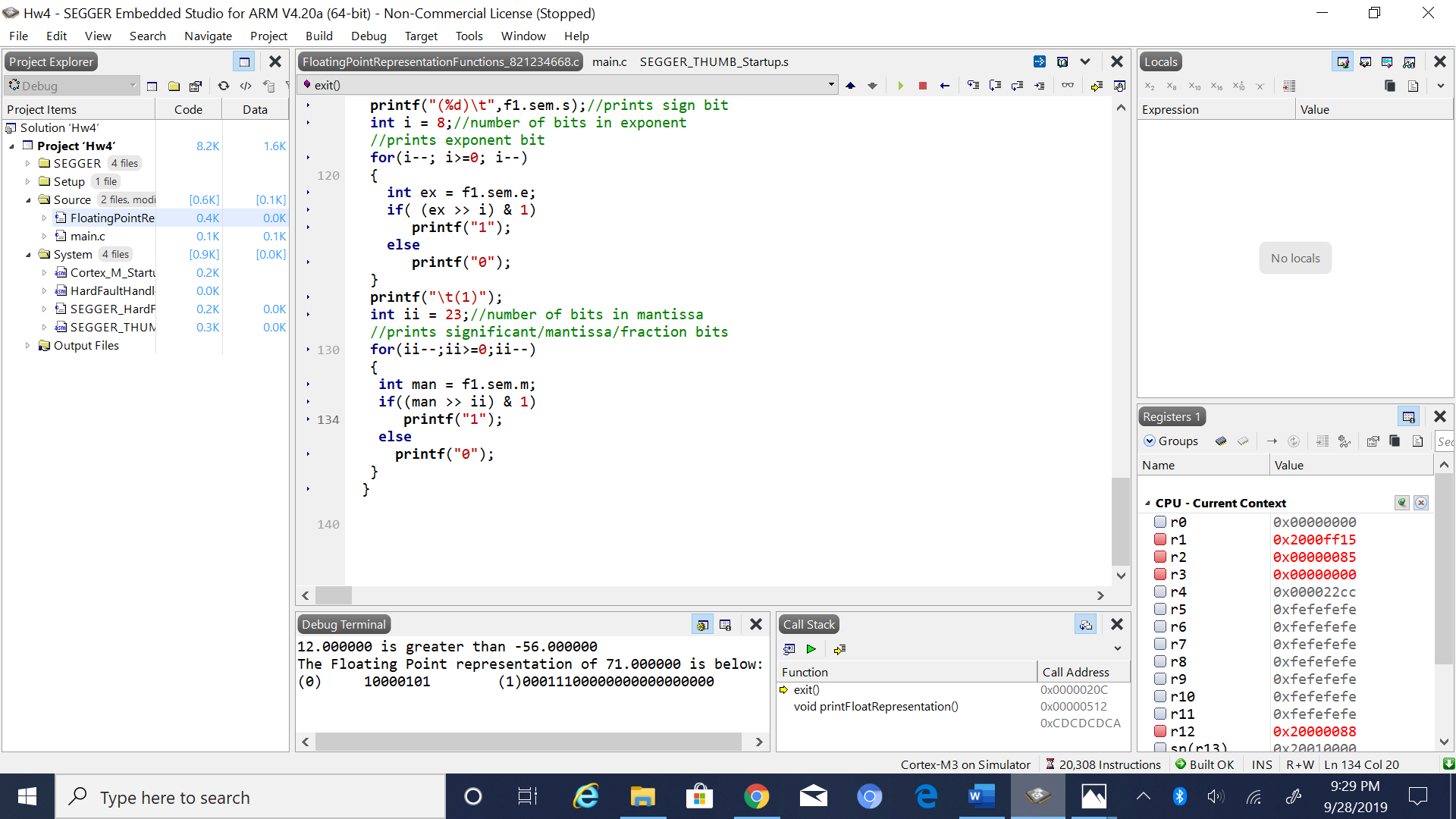


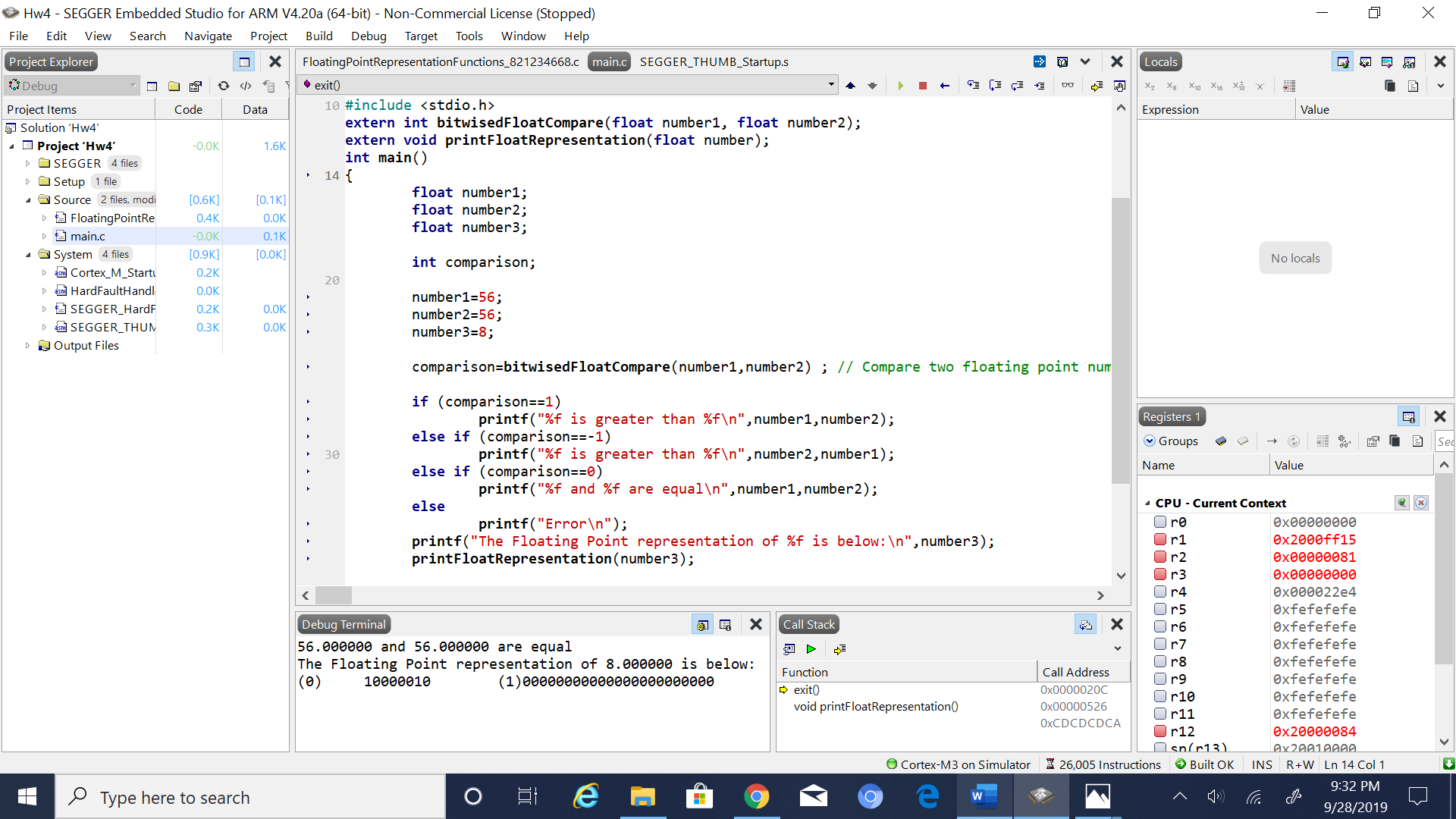












5. Conclusion (if applicable)

6\*. References.

Ken Arnolds slides on Floating point numbers