**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.*

9/11/2019

[Homework #2]

[Hw2]

Ckick below to enter/change your Name and RedID

Ethan Nagelvoort, 821234668

**Content**

(\* - Mandatory)

1\*. Description of the problem/method

Complete the three functions given in the template that involve bit manipulation. The first function to fulfill was countNumberofOnes, which takes in a pointer (called \*intData) to a uint32\_t, which is an unsigned int described in the header file, and returns and int. This function takes in an integer and counts how many ones are in the integers binary form. For example, if 8 is passed in, one will be returned. To do this I declared an int count and set it to 0 and an int tmp which equals \*intData. Then I used a for loop to increment count if tmp has a 1 currently. I did this by using the & operator, so if tmp & 1 was true, count would increment. At the end of the while loop I would shift tmp one bit to the left to get to the next bit in the integer. After the loop, I would return count.

The next function was setBit which takes in an uint32\_t \*intData, uint32\_t bitPosition, and uint32\_t value, and returns void. This function is meant to go to the bit position described with bitPosition in the integer \*intData and change that bit to value. I did this by declaring int tmp and making it equal value bit shifted to the left bitPosition-1 times. Then I make \*intData equal itself xored with tmp. This will not change the bits that do not need to be changed because all tmp has is a bunch of zeros, so the bits that have ones in \*intData will still retain their place in the new \*intData. However, the place that needs to change in the given bit position will change either from 1 to 0 or from 0 to 1 due to the xor.

The last function was hammingDistance which takes in a uint32\_t inData1 and a uint32\_t inData2, and returns an int. This function is meant to take in two integers and return the number of times the bits in the integers differ. For example if 8 and 9 were passed in, 1 would be returned because 8’s binary is 1000 and 9’s is 1001 and thus there is only one difference. To fulfill this function I set a count equal to 0 and and int tmp set to inData1^inData2. Then I have a while loop that stops if tmp ever equals 0. In the loop I increment count and set tmp to equal tmp & (tmp-1). Finding the this tmp until it turns into 0 will return the number of differences in the two given ints. After the loop, count is returned.

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

3\*. C-code

/\*

\* BitFunctions.c

\*

\* Created on: September 9, 2019

\* Author: Ethan Nagelvoort

\*/

#include "BitManipulation.h"

#include <stdio.h>

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

//

// countNumberofOnes

//

// Description: Counts the number of 1s in an integer passed as argument

// Preconditions:input argument is passed as a pointer

// Postconditions:the number of 1s returned

//

// Calls: N/A

// Called by: main

//

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

int countNumberofOnes(uint32\_t \*intData)

{

// Write a function that counts number of 1s in an integer passed

int count = 0;

int tmp = \*intData;

for(int i=0; tmp>0; i++)

{

count += (tmp & 1);

tmp >>= 1;

}

return count;

}

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

//\*

//\* setBit

//\*

//\* Description: The function sets the bit in the specified bit position in an to the specifid value.

//\* Preconditions: Value can be a 1 or 0. bitPosition will be between 0 and 31 (for integer size argument)

//\* Postconditions: The bit of \*inData at position biPosition will be set to value

//\*

//\* Calls: N/A

//\* Called by: main

//\*

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

void setBit(uint32\_t \*intData, uint32\_t bitPosition,uint32\_t value)

{

// Please do not treat the integer as arrays , this question is about bit manipulations

// You will need to use bitwise operations to solve this question

int tmp = value<<(bitPosition-1);

\*intData=\*intData^tmp;

}

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

//\* hammingDistance

//\* Description: Function hammingDistance calculates total number of bits

//\* that need to be inverted in order to change inData1 into inData2 or vice versa.

//\* Preconditions: The function accepts two unsigned integers as input

//\* Postconditions: The function returns the hamming distance

//\*

// Calls: N/A

// Called by: main

//\*

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

int hammingDistance(uint32\_t inData1, uint32\_t inData2)

{

int count = 0;

int tmp = inData1^inData2;

while (tmp != 0)

{

count++;

tmp = tmp & (tmp - 1);

}

return count;

}

/\*

\* BitManipulations.c

\*

\* Created on: September 9, 2019

\* Author: Ethan Nagelvoort

\*/

#include <stdio.h>

#include "BitManipulation.h"

int main(int \*argc, char \*\*argv)

{

uint32\_t Number;

uint32\_t bitPosition;

uint32\_t value;

int numOnes;

uint32\_t input1 = 152; //binary is 10011000

uint32\_t input2 = 130; //binary is 10000010

uint32\_t hDist;

value = 1;

bitPosition = 21;

Number = 15345;

int num2 = Number;

numOnes = countNumberofOnes(&Number);

setBit(&Number, bitPosition,value); // set bit in bitposition to value

hDist = hammingDistance(input1, input2); // Calculates hamming distance

printf("\nHamming Distance between %d and %d is %d\n", input1, input2, hDist);

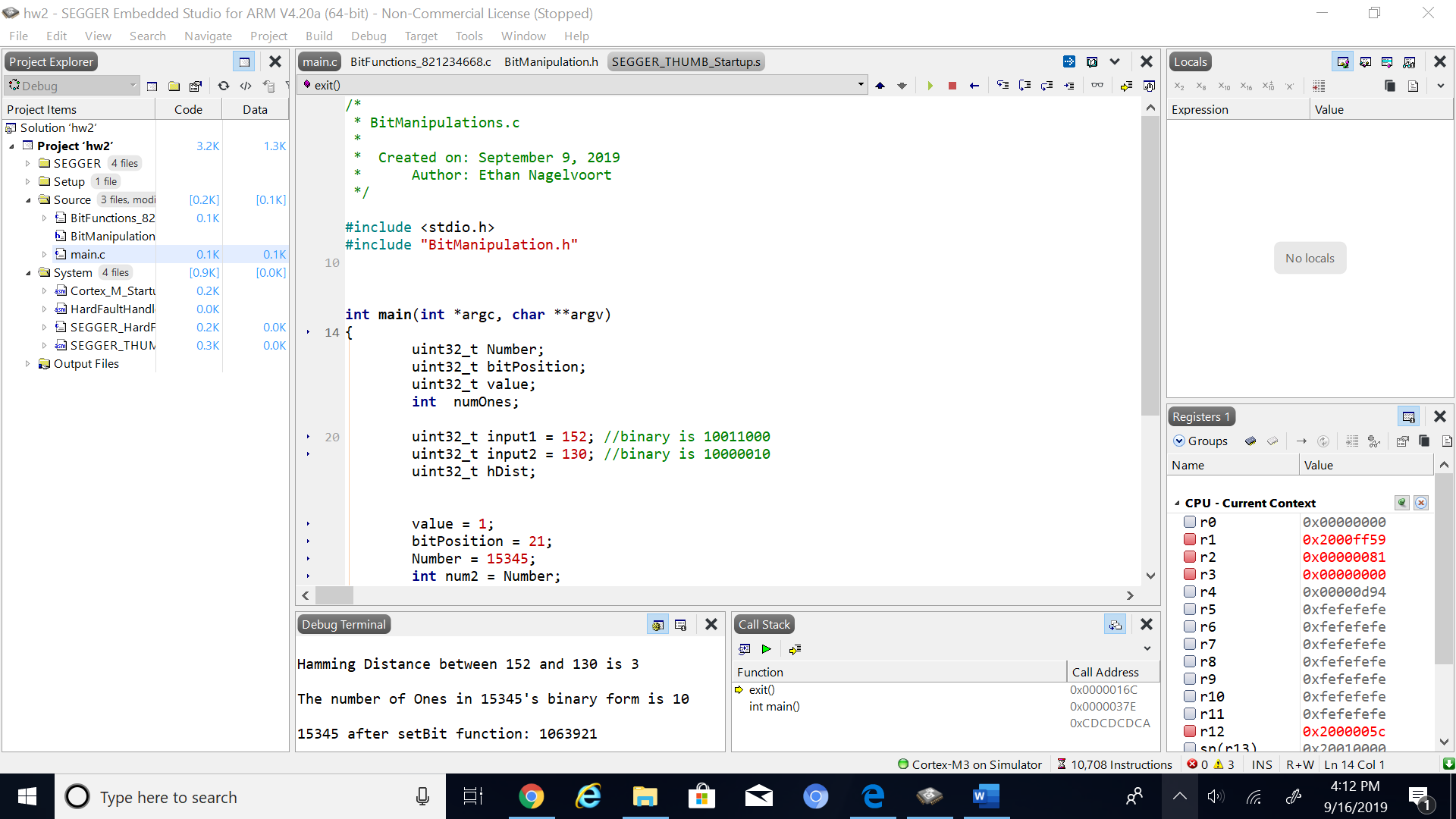
printf("\nThe number of Ones in %d's binary form is %d\n", num2, numOnes);

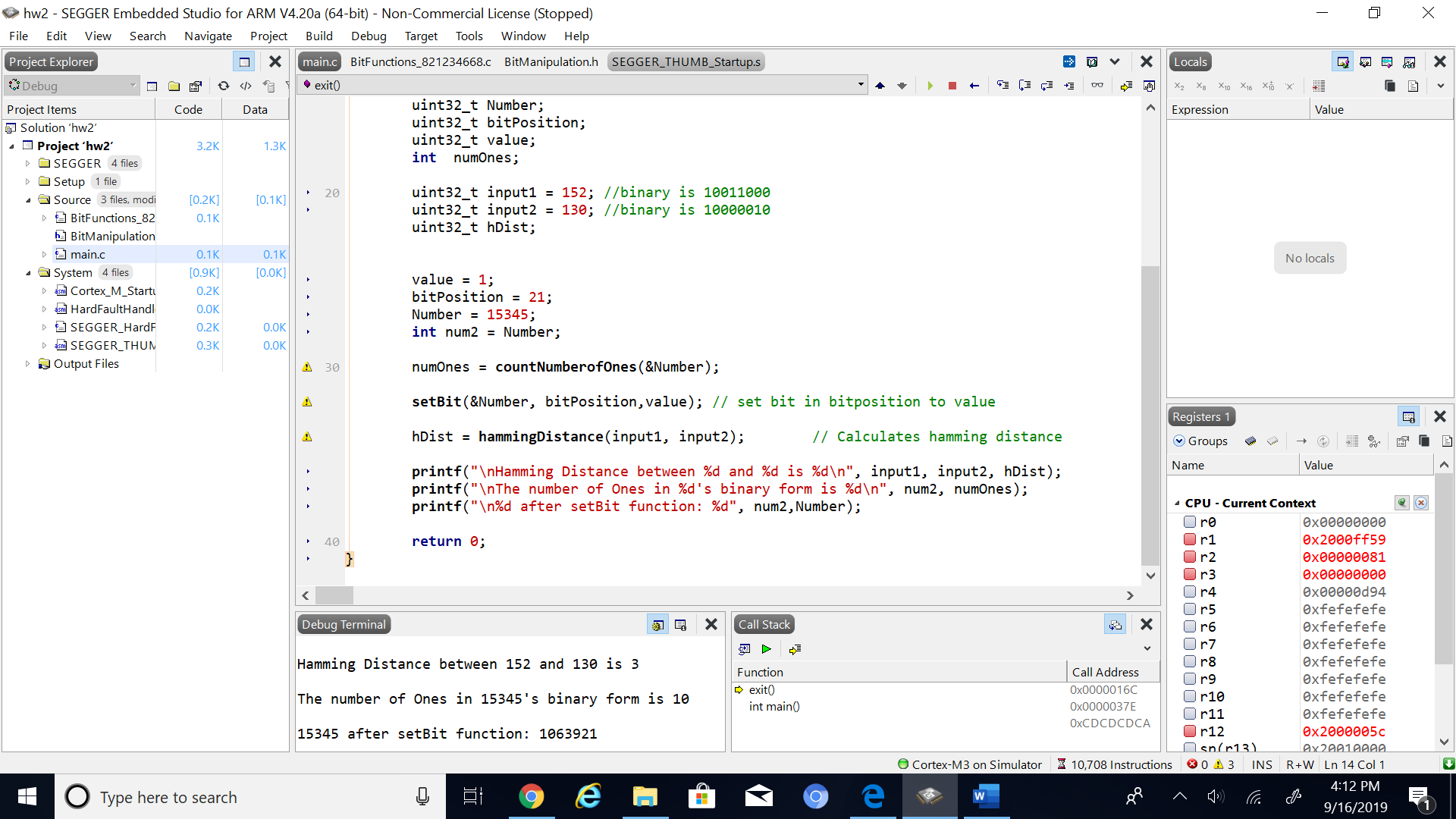
printf("\n%d after setBit function: %d", num2,Number);

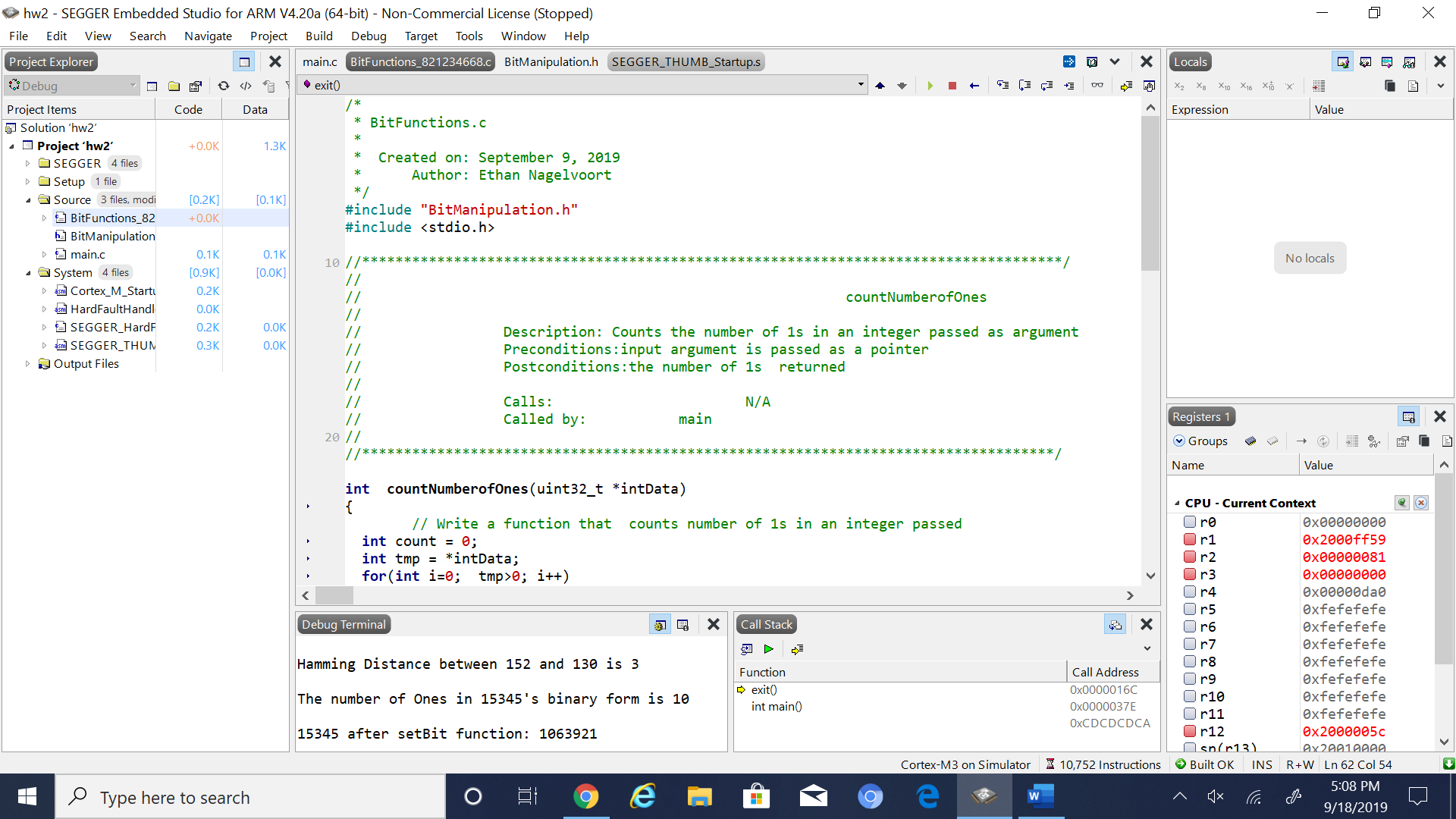
return 0;

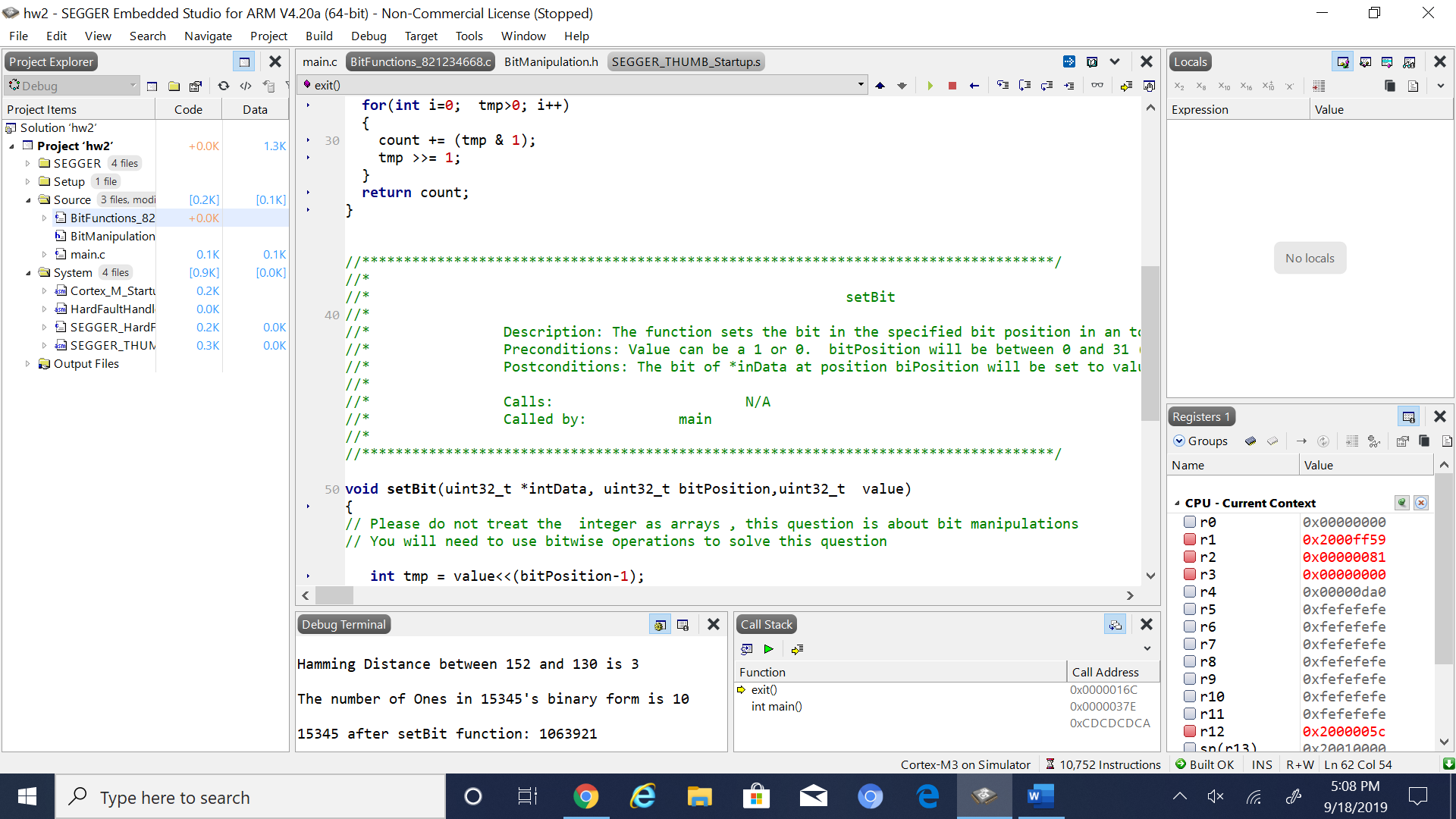
}

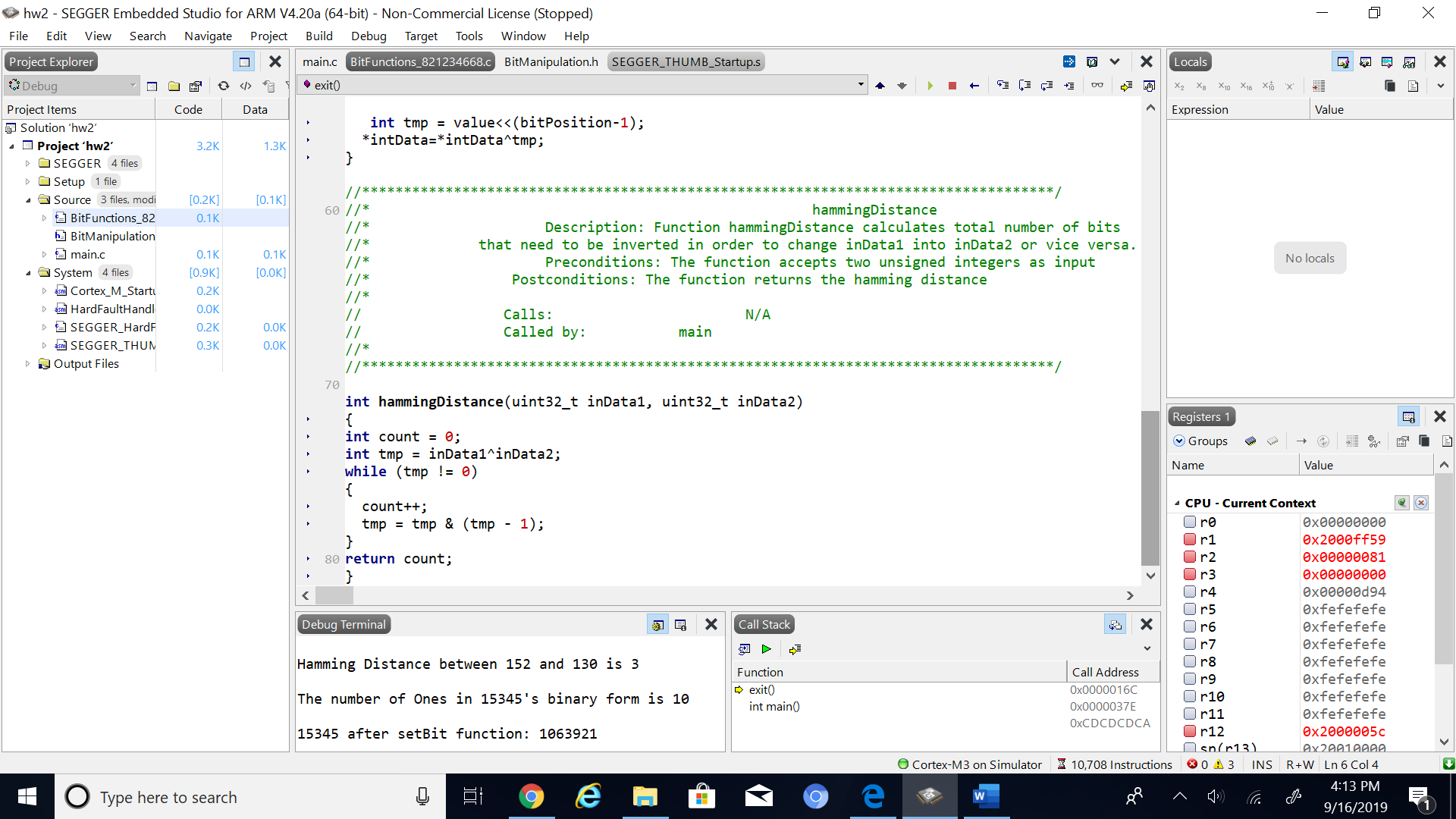
4\*. Screen capture of the code and the resulting display(s)











5. Conclusion (if applicable)

6\*. References.

Professors Arnolds given PowerPoint slides on blackboard on bit operations.