CS 342000 | CS343000
Instructor: Professor Izidor Gertner
Spring 2022
Azwad Shameem, 4/10/2022
Take Home Test 1

Table of Contents

Objective

The objective of the take home test is to understand and compare and contrast the various architectures. From MIPS instruction to Intel x86 ISA to Intel x86 64-bit ISA all have their respective differences and similarities and we need to take note of these. To compare the similarities, we use the MARS simulator for MIPS and Visual Studio's debugger for Intel x86 ISA 32 bit and Linux GCC and GDB for Linux 64-bit. From using these tools, we can observe the similarities in the differences in runtime in how the data is handled and where the data is handled. Using the information on how the different platforms and compilers handle the task we will be able to understand their differences and similarities and understand them.

Solutions

MIPS

2-2_1.asm

Edit Execute

Shameem_2-2_1.asm

```
1   .data
2   a: .word 1
3   b: .word 2
4   c: .word 3
5   d: .word 4
6   e: .word 5
7
8   .text
9   lw $s0, a
10  lw $s1, b
11  lw $s2, c
12  lw $s3, d
13  lw $s4, e
14
15  # a = b + c
16  add $s0, $s1, $s2
17  sw $s0, a
18
19  # d = a - e
20  sub $s3, $s0, $s4
21  sw $s3, d
```

Figure 1: Shameem_2-2_1.asm code in MARS

The code written is written in 2-2_1.asm in MIPS and is shown in figure 1, which is the image above. The code entails five static variables, a, b, c, d, e and these five variables are used to perform arithmetic which is stored in memory afterwards. The arithmetic performed is the addition of b and c, whose result is stored in a. The other arithmetic is performed with the subtraction of a and e, whose result is stored in d.

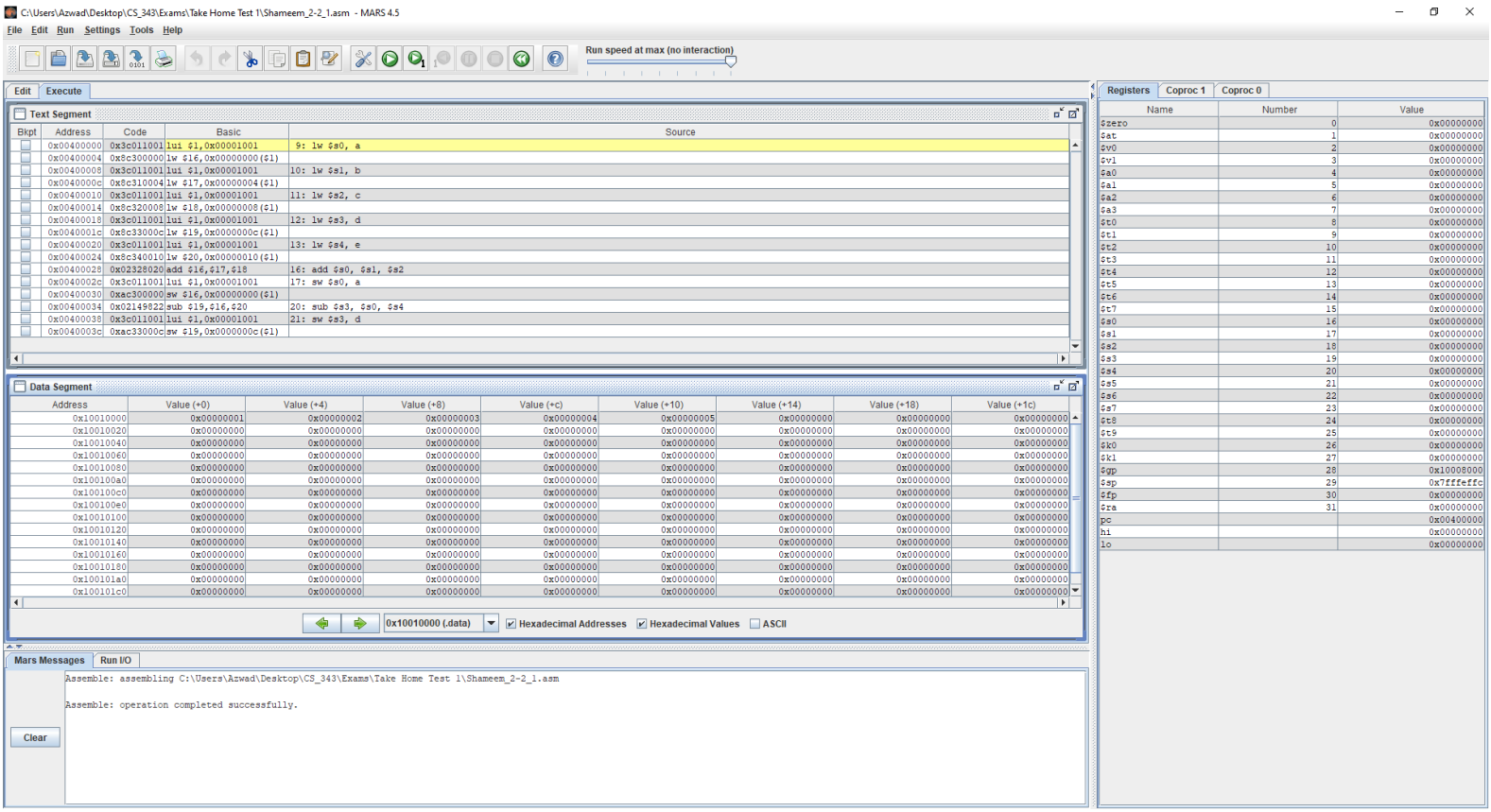


Figure 2: Shameem_2-2_1.asm code executed in MARS

Text Segment: This window shows that the code starts from line 9 because the prior lines don't require a register.

Registers: This window shows the value of the stack pointer which is 0x7FFFEFFC.

Data Segment: This window shows the value of the static variables stored into the addresses.

Variable a is stored in address: 0x10010000 containing the value: 0x00000001.
Variable b is stored in address: 0x10010004 containing the value: 0x00000002.
Variable c is stored in address: 0x10010008 containing the value: 0x00000003.
Variable d is stored in address: 0x1001000c containing the value: 0x00000004.
Variable e is stored in address: 0x10010010 containing the value: 0x00000005.

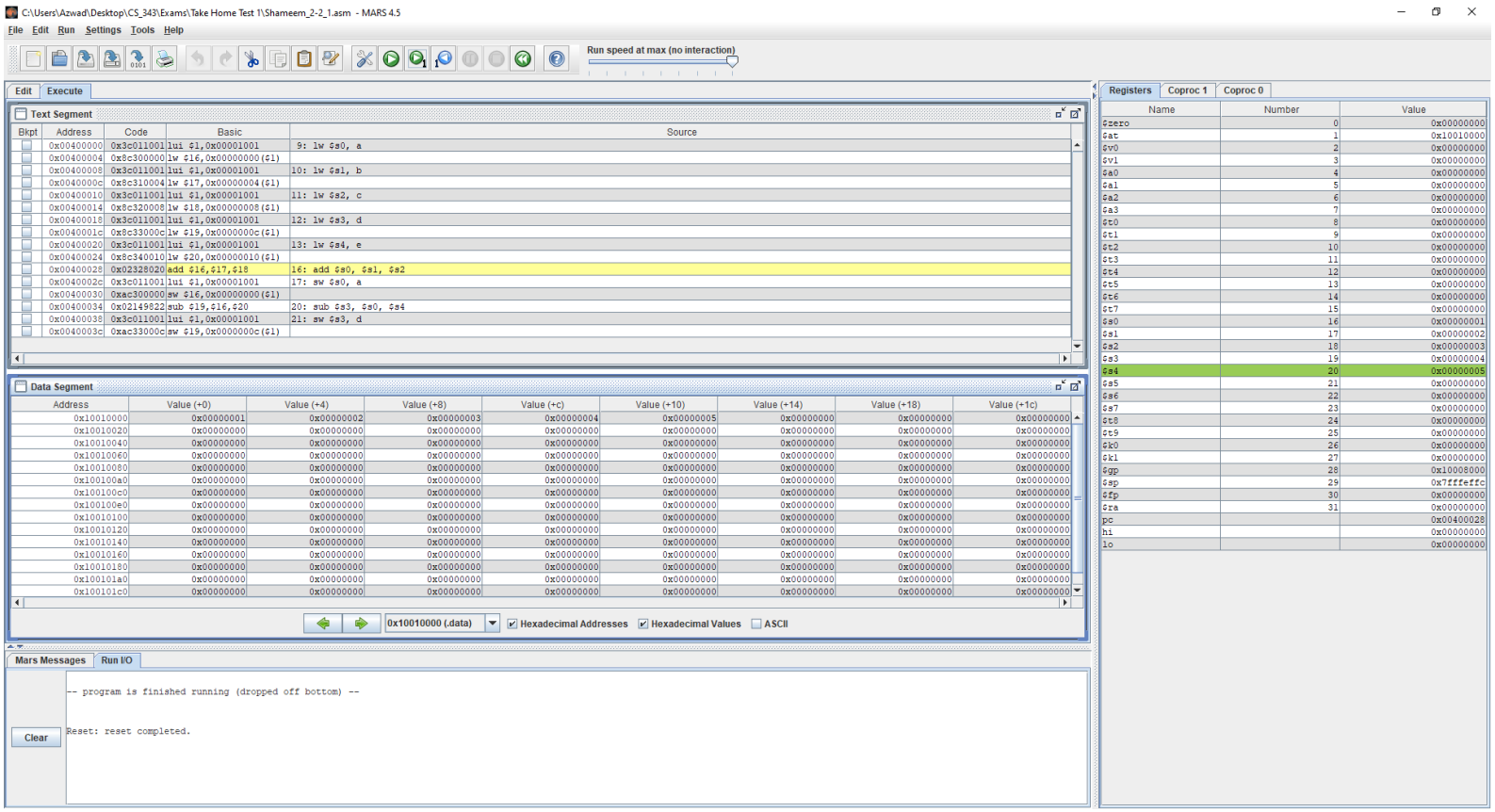


Figure 3: Shameem_2-2_1.asm code executed in MARS (Lines 9-16)

Text Segment: This window shows that the code ran lines 9 to 16 and that the static variables

have been loaded into the registers during after running lines 9 to 16.

Registers: This window shows the values

Value of a is stored into $s0,

Value of b is stored into $s1,

Value of c is stored into $s2,

Value of d is stored into $s3,

Value of e is stored into $s4.

Data Segment: This window didn't change from figure 2 because lines 9 to 16 doesn't store any values.

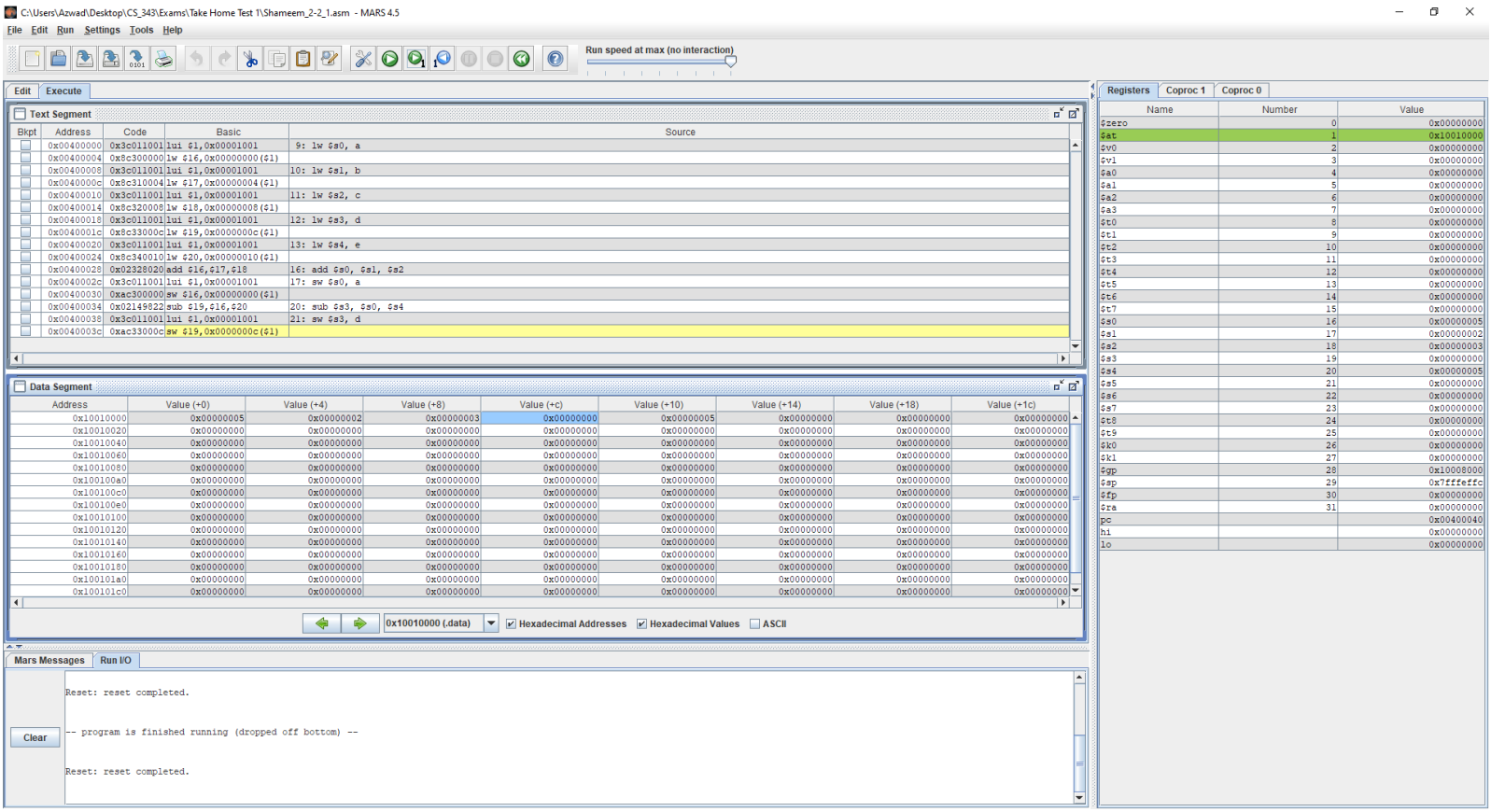


Figure 4: Shameem_2-2_1.asm code executed in MARS (Lines 16-19)

Text Segment: This window shows the code running lines 16 to 19 and the effects on the

registers and data segments during said run.

Registers: This window shows that the value of $s1 and $s2 are added, and the result is stored into $s0. The window also shows that the value of $s0 and $s4 are subtracted, and the result is stored into $s3.

Data Segment: This window shows that the addition of $s1 and $s2, and it's $s0 is stored into the adr0x1001000 and allotted to the variable a, whose value is now 0x00000005.

This window also shows the subtraction of $s0 and $s4, and it's result $s3 is stored into the address 0x1001000C and allotted to the variable d, which contains the value 0x00000000.

2-2_2.asm

Edit Execute

Shameem_2-2_2.asm

```
1  .data
2  f: .word 0
3  g: .word 50
4  h: .word 40
5  i: .word 30
6  j: .word 20
7  .text
8  lw $s0, f
9  lw $s1, g
10 lw $s2, h
11 lw $s3, i
12 lw $s4, j
13 # t0 = g+h
14 add $t0, $s1, $s2
15 # t1 = i+j
16 add $t1, $s3, $s4
17 # f = t0 - t1
18 sub $s0, $t0, $t1
19 sw $s0, f
```

Figure 5: Shameem_2-2_2.asm code in MARS

The code written in 2-2_2.asm shows five static variables f, g, h, I, j which will be used to do arithmetic and store the value in memory. At line 14, we add g and h and store the result into $t0. Then at line 16, we add i and j and store the result into $t1. Then, we add $t0 and $t1 and store it into f. Lastly, we save that result in $s0 in the data segment.

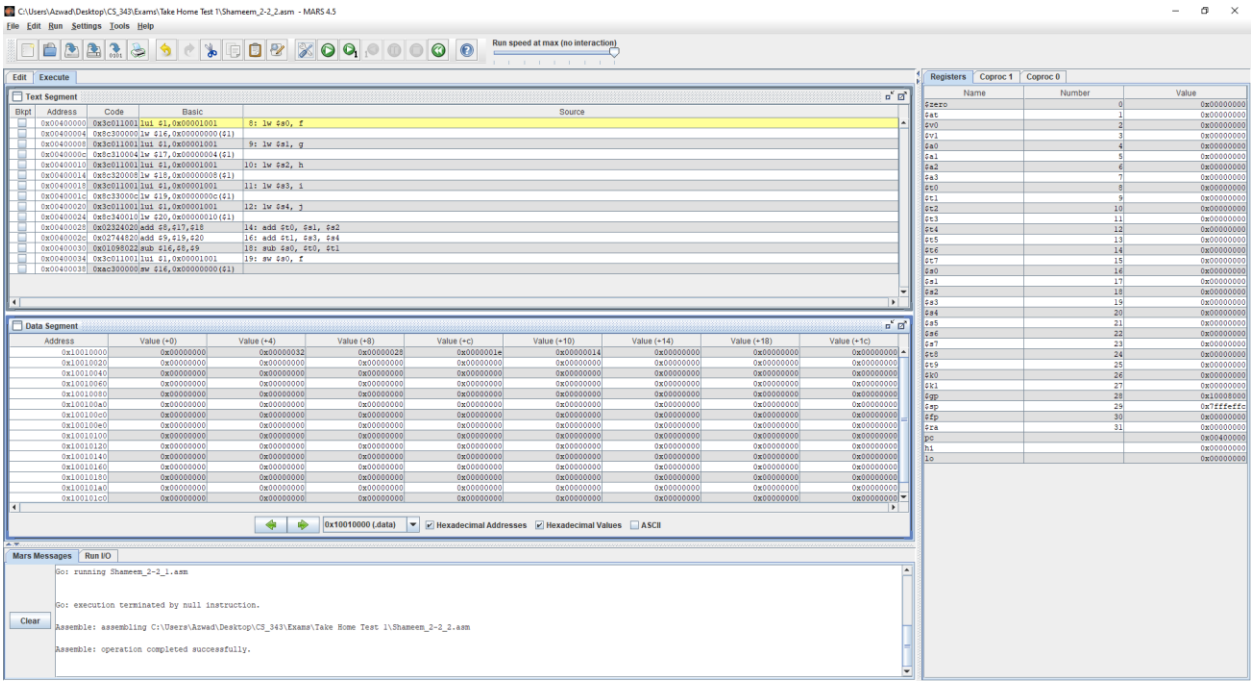


Figure 6: Shameem_2-2_2.asm code executed in MARS (lines 1-7)

Text Segment: This window shows that the code starts at line 8 because lines 1-7 do not require registers.

Registers: This window shows value of the stack pointer is 0x7FFFEFFC. Also, no registers have been stored because no operations have been performed at this stage.

Data Segment: This window shows the value of the static variables being stored into the addresses.

Variable f is stored in address: 0x10010000 containing the value: 0x00000000.
Variable g is stored in address: 0x10010004 containing the value: 0x00000032.
Variable h is stored in address: 0x10010008 containing the value: 0x00000028.
Variable i is stored in address: 0x1001000C containing the value: 0x0000001e.
Variable j is stored in address: 0x10010010 containing the value: 0x00000014.

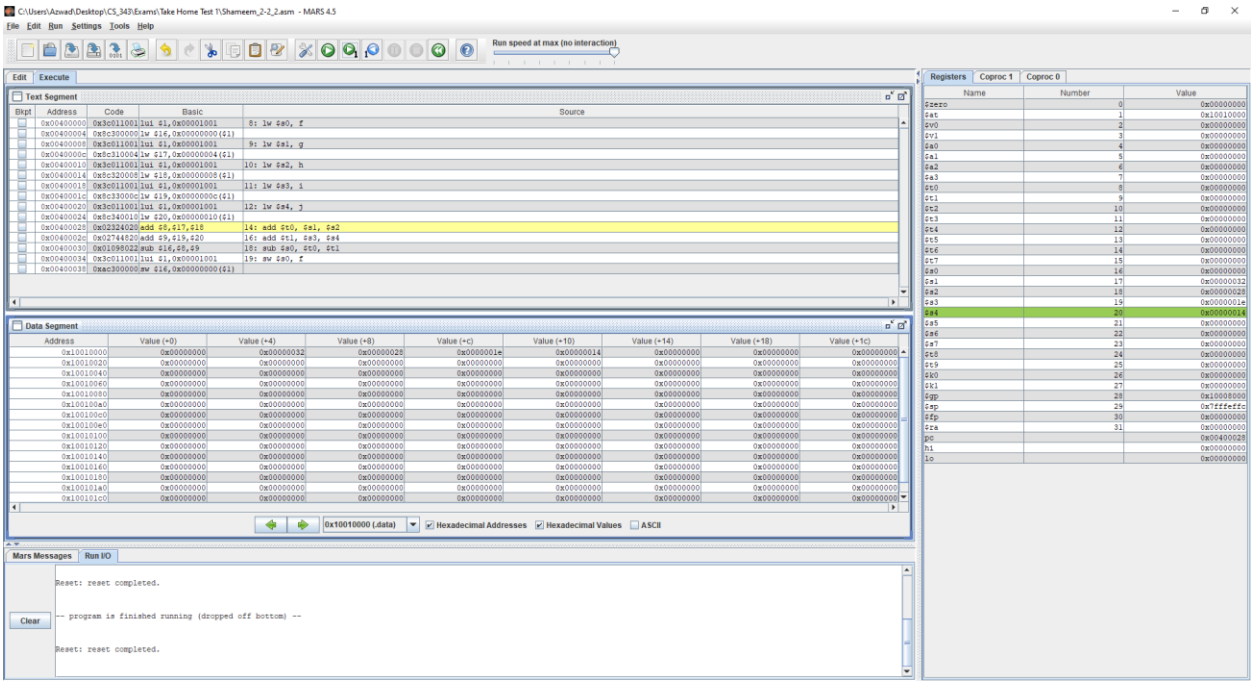


Figure 7: Shameem_2-2_2.asm code executed in MARS (lines 8-13)

Text Segment: This window shows that the lines 8-13 ran and the corresponding static variables are loaded into the registers after running lines 8-13.

Registers: This window shows that values of f, g, h, i and j is stored into $s0, $s1, $s2, $s3, $s4.

Data Segment: This window shows that even after running lines 8-13 the data segment remains the same because lines 8-13 doesn't store nor change values.

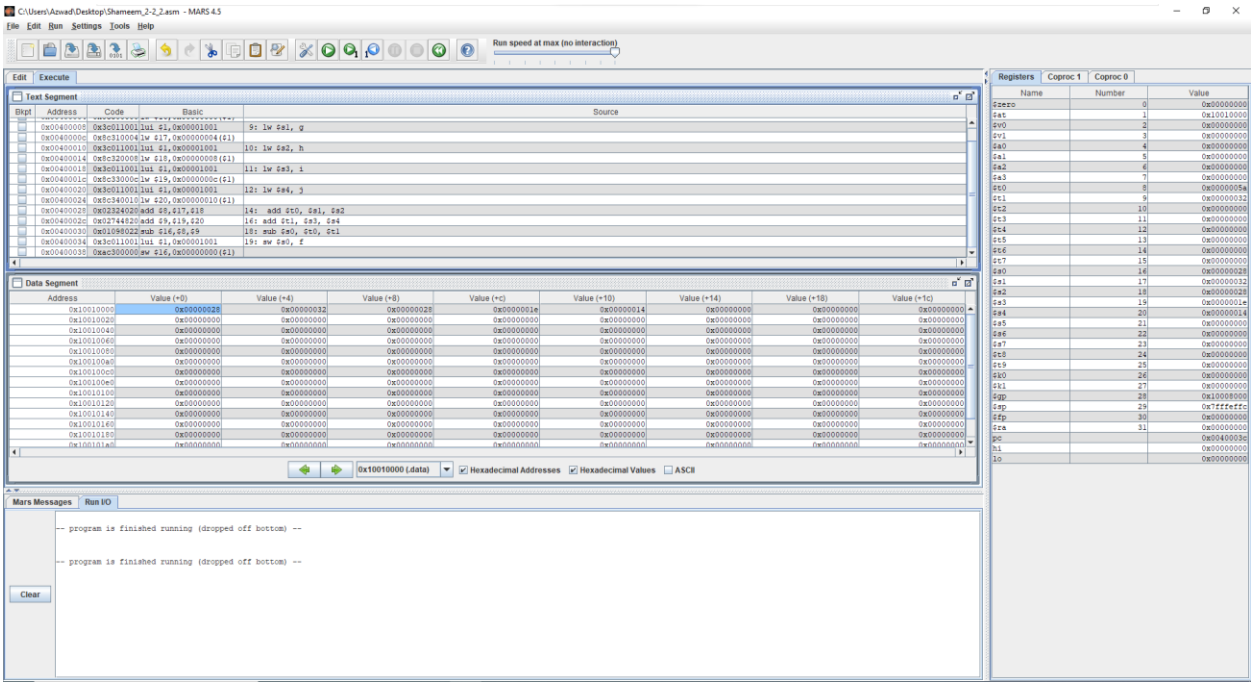


Figure 8: Shameem_2-2_2.asm code executed in MARS (lines 14-19)

Text Segment: This window shows that the code runs lines 14-19. The window also shows that lines 14-19 has addition and subtraction instructions whose values should now be reflected in the registers and data segment.

Registers: This window shows that the values of $s1 and $s2 and the result of their addition which is stored in $t0. Also, the window shows the values of $s3 and $s4 and the result of their addition is stored into $t1. Lastly, the window shows the values of $t0 and $t1 and the result of their subtraction which is stored into $s0.

Data Segment: This window shows that after the subtraction of $t0 and $t1, the result $s0 is stored into the address 0x10010000 and is allotted to the variable f which contains the value 0x00000028.

2-3_1.asm

Edit Execute

Shameem_2-3_1.asm

```
1  .data
2  g: .word 0
3  h: .word 22
4  A: .word 0-100
5  size: .word 100
6  .text
7  #just to set A[8] to 55
8  li $t1, 55
9  la $s3, A
10 sw $t1, 32($s3)
11 lw $s1, g
12 lw $s2, h
13 #loading the value of A[8] into t0
14 lw $t0, 32($s3)
15 add $s1, $s2, $t0
16 sw $s1, g
```

Figure 9: Shameem_2-3_1 code displayed in MARS

The code written for 2-3_1 has four static variables g, h, A, size which is used for addition and creating an array. In the code the value is stored into the 8th index of the array and is saved into $t0 plus the value of $s2, h, and $t0 is saved into $s1, g.

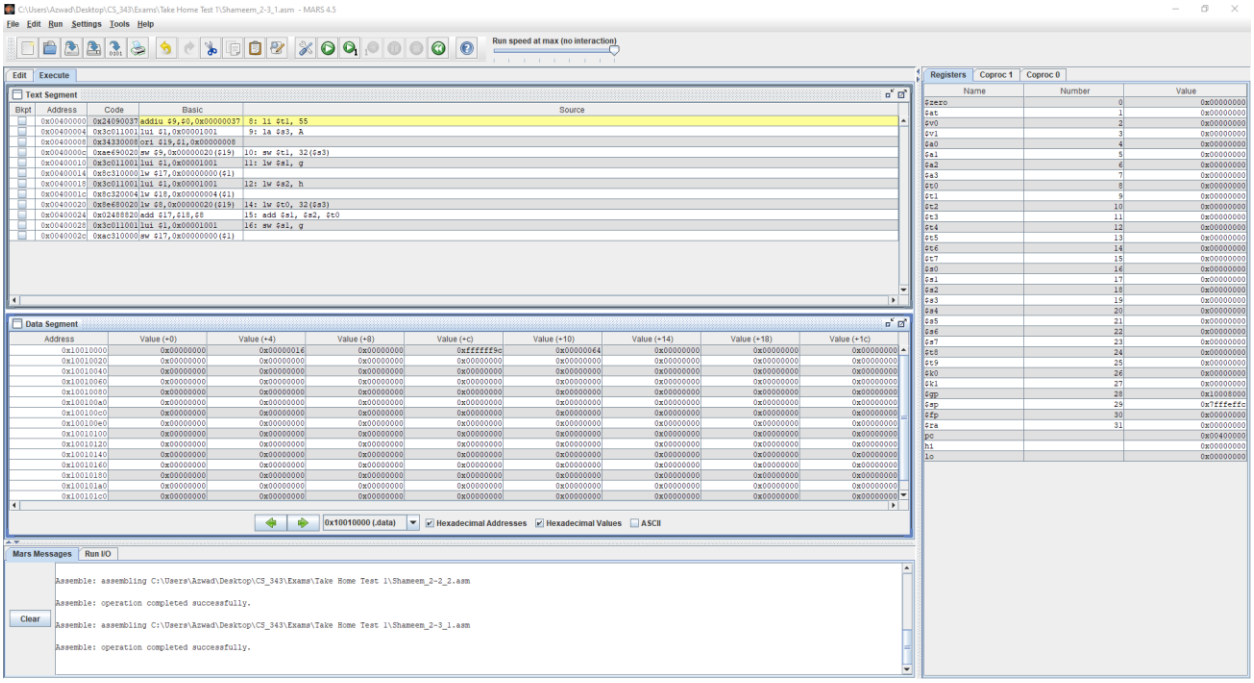


Figure 10: Shameem_2-3_1.asm code executed in MARS (lines 1-7)

Text Segment: This window shows that the code starts at line 8 because lines 1-7 don't use registers.

Registers: This window shows the value of $sp, which is 0x7FFFEFFC.

Data Segment: This window shows the value of the static variables being stored.

Variable g is stored in the address 0x10010000 which has the value 0x00000000.

Variable h is stored in address: 0x10010004 containing the value: 0x00000016.

Variable A is stored in addresses: 0x10010008 and containing the values: 0x00000000.

Also, variable A is stored in addresses 0x1001000C containing the values 0xFFFFFF9C.

Variable size is stored in address: 0x10010010 containing the value: 0x00000064.

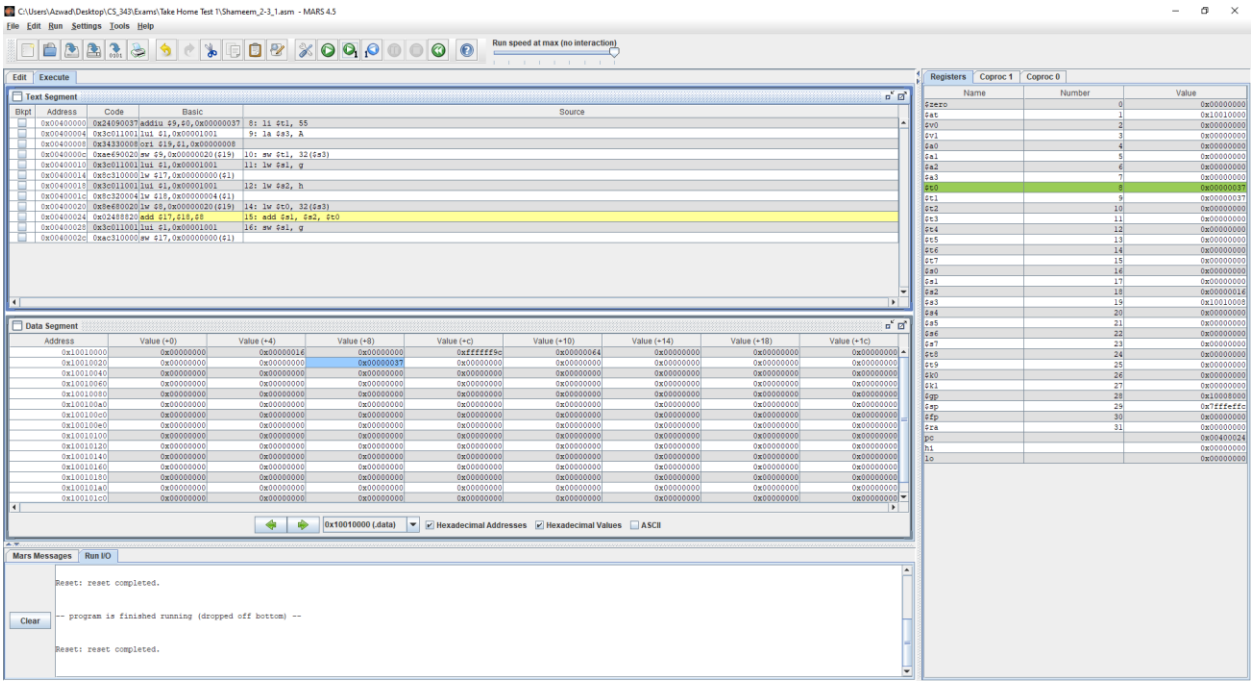


Figure 11: Shameem_2-3_1.asm code executed in MARS (lines 8-14)

Text Segment: This window shows that the code runs lines 8-14. Lines 8-14 include the static variables and a value being loaded into the registers. Also, the code stores the value of a register into the data segment which is highlighted in blue.

Registers: This window shows that the value 55 is stored into $t1, the address of A is stored into $s3, the value of g is stored into $s1, the value of h is stored into $s2 and the value of 8th index in the array A is stored into $t0.

Data Segment: This window shows that the value of $t1, which is 55, is stored in the 8th index of array A. The previous statement is shown in Address 0x10010028, which contains the value 0x00000037.

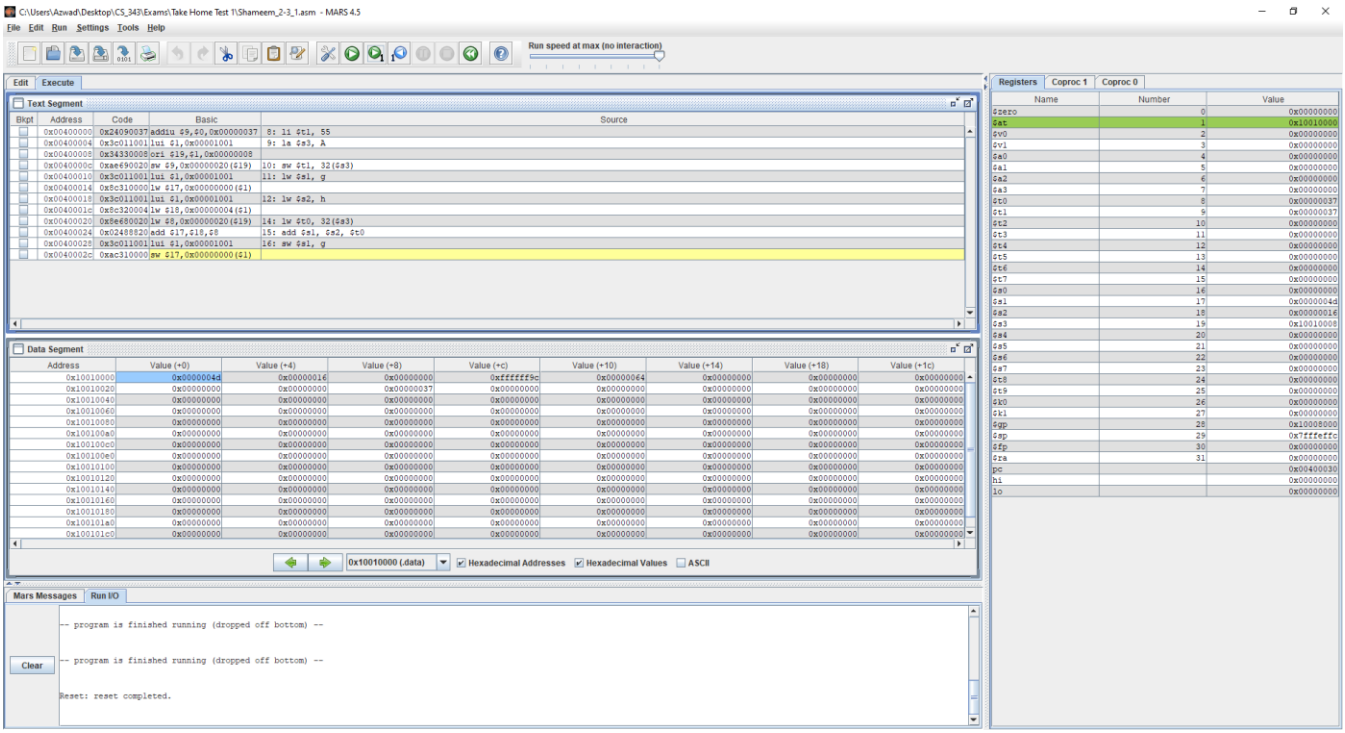


Fig 12: Shameem_2-3_1.asm code executed in MARS (lines 15-16)

Text Segment: This window shows that the code runs lines 15-16. The window also shows that in lines 15-16 we add $s2 and $t0 and store the result in $s1 and save the value of $s1 into g.

Registers: This window shows the added value of $s2 and $t0, which is 0x0000004D, stored into $s1.

Data Segment: This window shows that the result of the addition of $s2 and $t0, which is 0x0000004D is stored in $s1, which is saved in variable g which is highlighted at address 0x10010000 and has the value 0x0000004D.

2-3_2.asm

Edit Execute

Shameem_2-3_2.asm

```
1  .data
2  h: .word 25
3  A: .word 0-100
4  size: .word 100
5  .text
6  lw $s2, h
7  #initializing A[8] to 20
8  li $t1, 200
9  la $s3, A
10 sw $t1, 32($s3)
11 #A[12] = h + A[8]
12 lw $t0, 32($s3)
13 add $t0, $s2, $t0
14 sw $t0, 48($s3)
```

Figure 14: Shameem_2-3_2.asm code displayed in MARS

The code written for 2-3_2.asm uses three static variables h, A and size. The code uses these three static variables to create an array, compute addition. In the code the value 200 is stored in the array and is added to the static variable h. Then the result of the addition of 200 and h is stored into the index 12 of the array A.

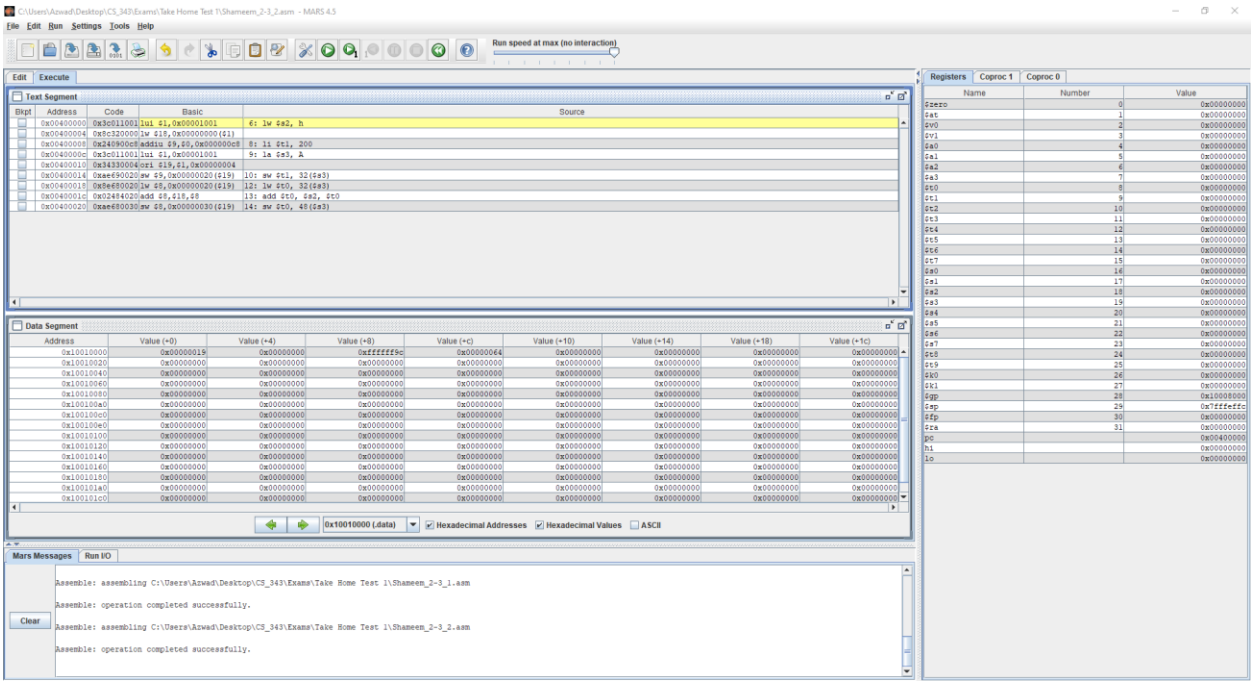


Figure 15: Shameem_2-3_2.asm code executed in MARS (lines 1-5)

Text Segment: This window shows that the code starts at line 6 because lines 1-5 don't need registers.

Registers: This window shows that the value of $sp, which is 0x7FFFEFFC.

Data Segment: This window shows the value of the static variables is stored at the addresses.

Variable h is stored in address 0x10010000 and has the value 0x00000019.
Variable A is stored in addresses 0x10010004 and has the values 0x00000000.
Variable A is stored in addresses 0x10000008 and has the value: 0xFFFFFF9c.
Variable size is stored in address 0x1001000C containing the value 0x00000064.

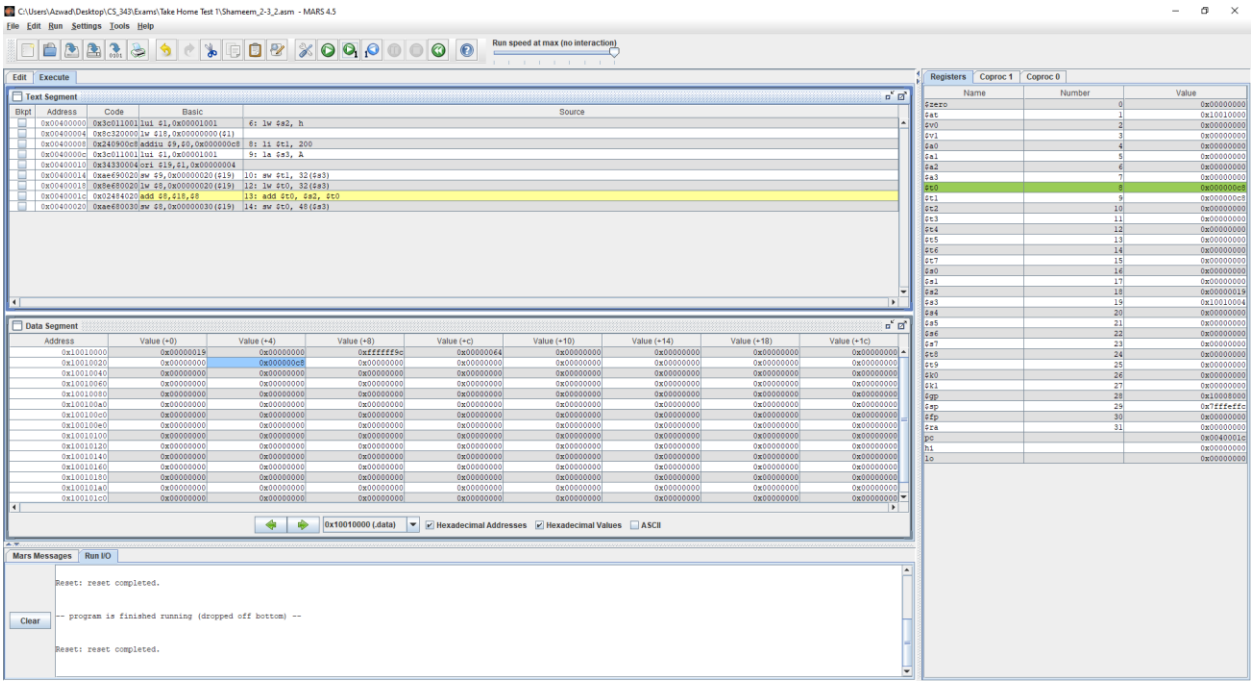


Figure 16: Shameem_2-3_2.asm code executed in MARS (lines 6-12)

Text Segment: This window shows that the code runs lines 6-12. In lines 6-12 the code loads the static variables and the value 200 to the registers.

Registers: This window shows that the values stored.

The value of h is stored into $s2.
The value of 200 is stored into $t1, which is shown as 0x000000C8.
The address of A is stored into $s3.
The value of 8th index of array A is stored into $t0.

Data Segment: This window shows that the value of $t1 is stored in the 8th index of array A which is located at the address 0x10010024 and has the value 0x000000C8 and is highlighted.

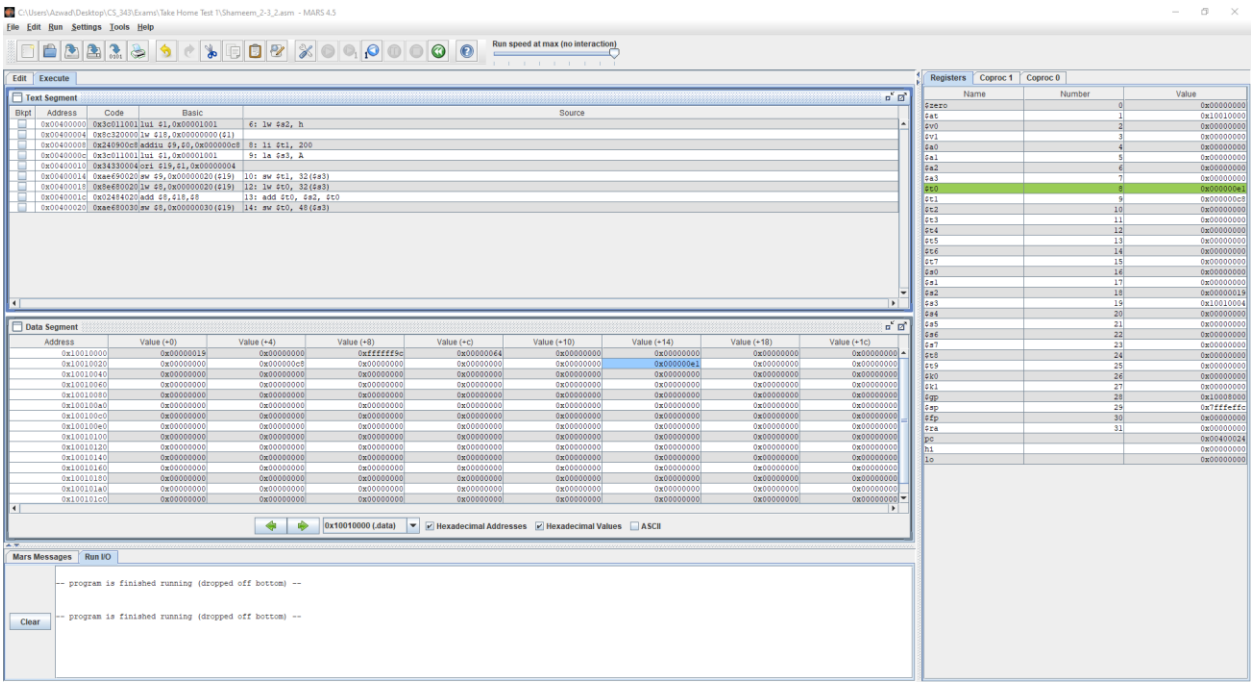


Figure 17: Shameem_2-3_2.asm code executed in MARS (lines 13-14)

Text Segment: This window shows that the code runs lines 13-14. In lines 13-14 the code adds $s2 and $t0 and stores it into $t0 which is then stored into the array.

Registers: This window shows that the value of $s2 and $t0 are added, and the result is stored into $t0 is highlighted.

Data Segment: This window shows that the value of $t1 is stored in the 8th index of array A which is located at the address 0x10010024 and has the value 0x000000C8 and is highlighted.

2-5_2.asm

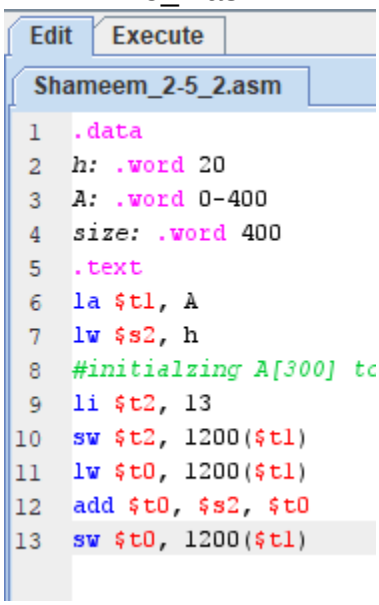


Figure 18: Shameem_2-5_2.asm code in MARS

The code written uses three static variables h, A and size. The code uses these three variables to initialize the array at 300th index. Then the code also performs addition using $s0 and $t0 and stores the result into $t0 and then stores that into another part of the array and data segment.

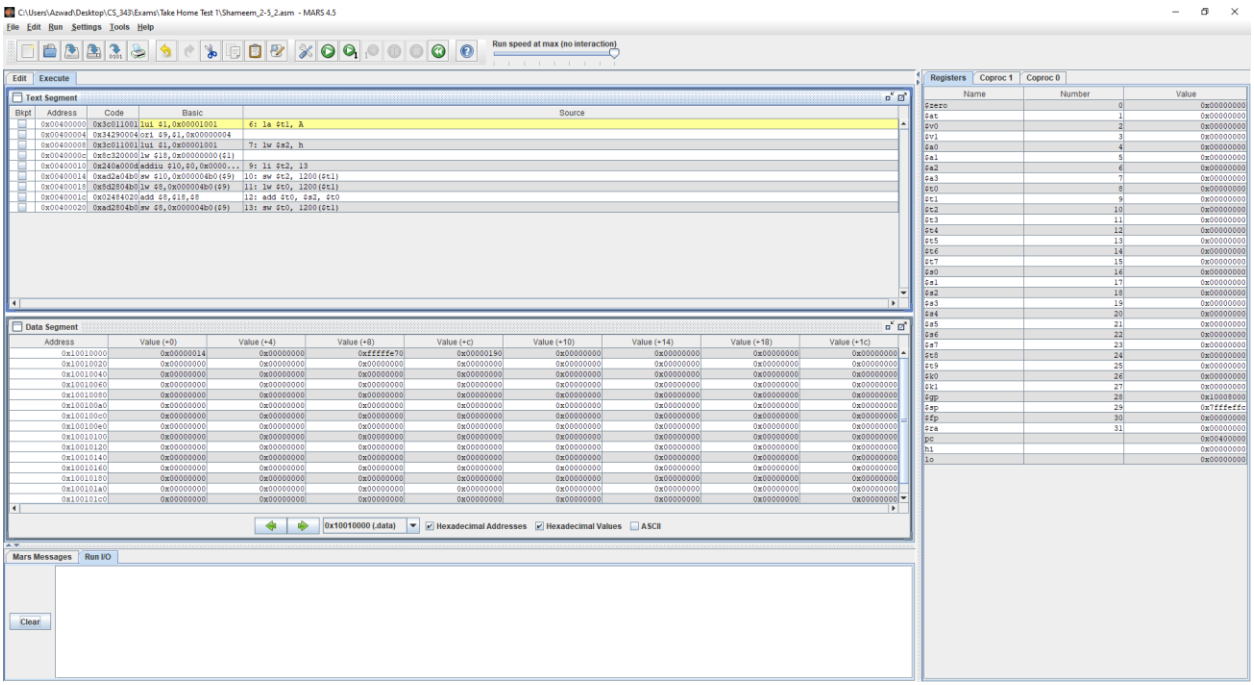


Figure 19: Shameem_2-5_2.asm code executed in MARS (lines 1-5)

Text Segment: This window shows that the code starts at line 6 because lines 1-5 don't need registers.

Registers: This window shows that the value of $sp, which is 0x7FFFEFFC.

Data Segment: This window shows the value of the static variables is stored at the addresses.

The variable h is stored at address 0x10010000 and has the value 0x00000014.

The variable A is stored at addresses 0x10010004 and has the values: 0x00000000.

The variable A is stored at addresses 0x10000008 and has the values: 0xFFFFFE70.

The variable size is stored at address 0x1001000C and has the value: 0x00000190.

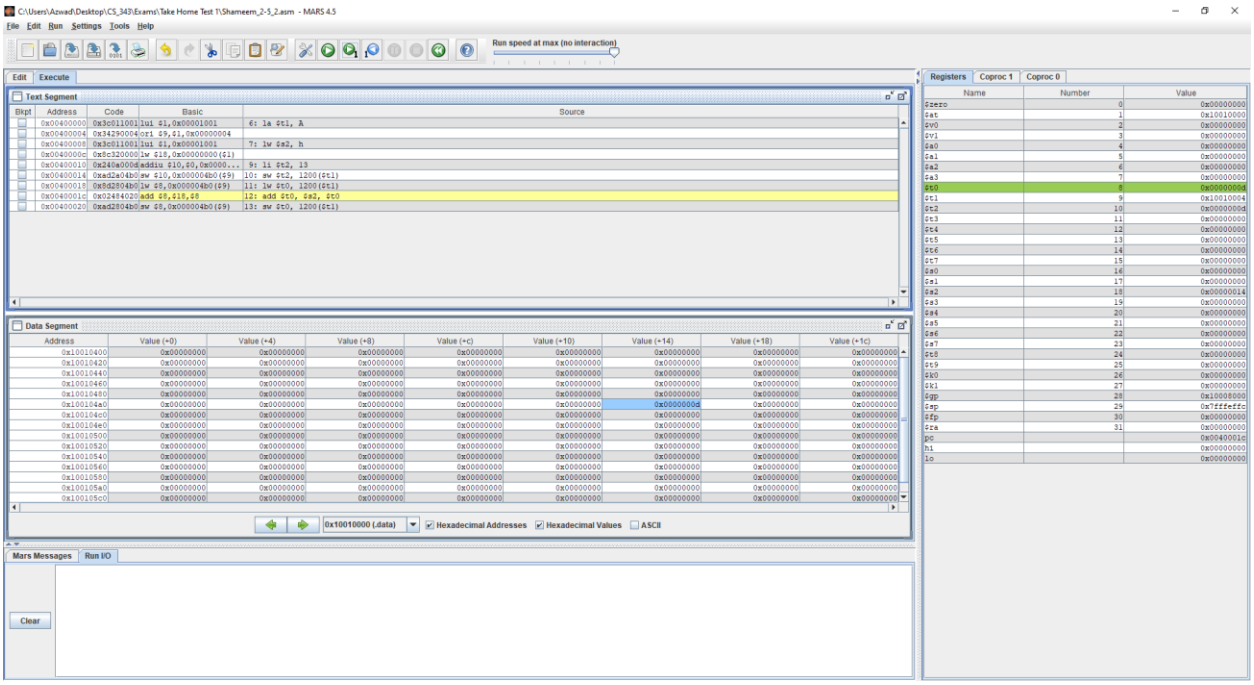


Figure 20: Shameem_2-5_2.asm code executed in MARS (lines 6-12)

Text Segment: This window shows that the code runs lines 6-11. In lines 6-11, the static variables and the value of $t0 is loaded into the registers and data segment which is highlighted.

Registers: This window shows that the values of the variables and where they are stored.

The value of h is stored at $s2.
The value of 13 is stored into $t2 as a hexadecimal which is 0x0000000D.
The address of A is stored at $t1.
The value of 300th index of array A is stored into $t0.

Data Segment: This window shows that value of $t2, which is 13, is stored in the 300th index of the array A which is at address 0x100104B4 and has the value of 0x0000000D.

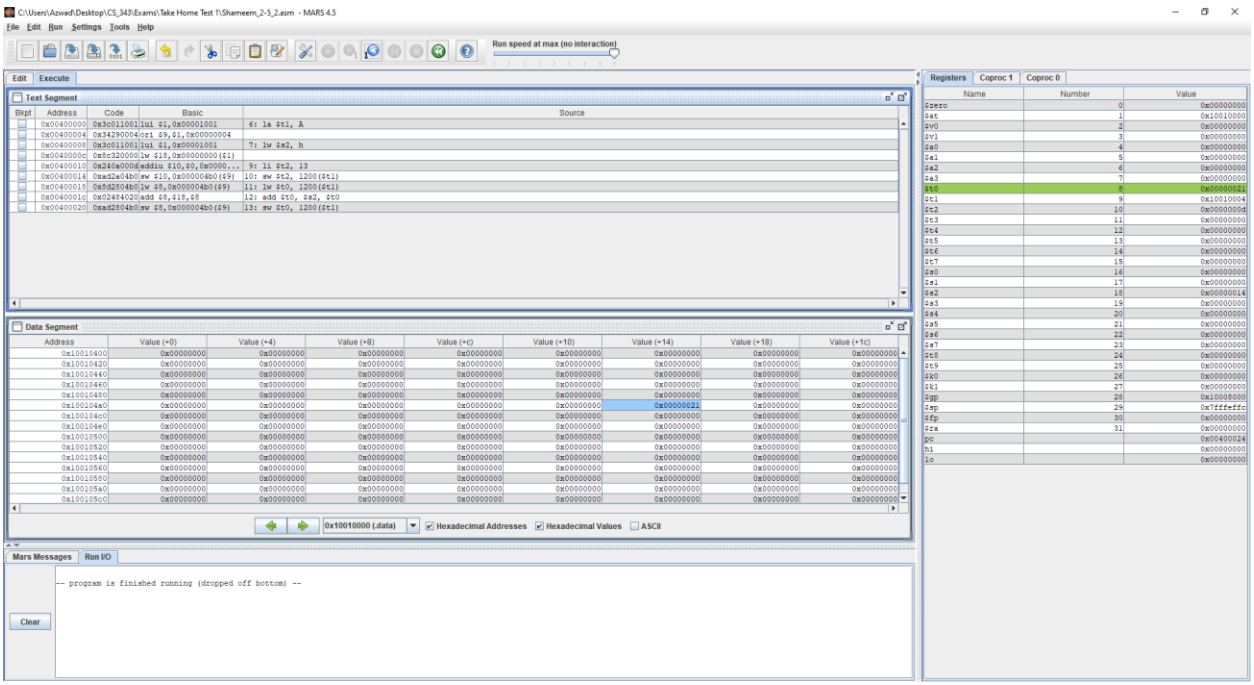


Figure 21: Shameem_2-5_2.asm code executed in MARS (lines 13)

Text Segment: This window shows that the code runs line 13. In line 13, the static variables and the new value of $t0 is loaded into the registers and data segment which is highlighted.

Registers: This window shows that the value of the addition of $s2 and $t0, whose result is stored in $t0.

Data Segment: This window shows that the addition of $s2 and $t0 is stored into the address 0x100104B4 and put into the 300th index of the array A, which now has the value 0x00000021.

2-6_1.asm

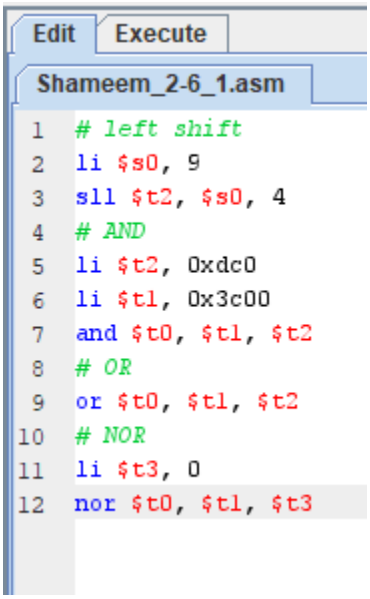


Figure 22: Shameem_2-6_1.asm code in MARS

The code written stores the values into the registers and then uses the values to perform three bitwise operations, Shift Left Logical, AND, OR and NOR.

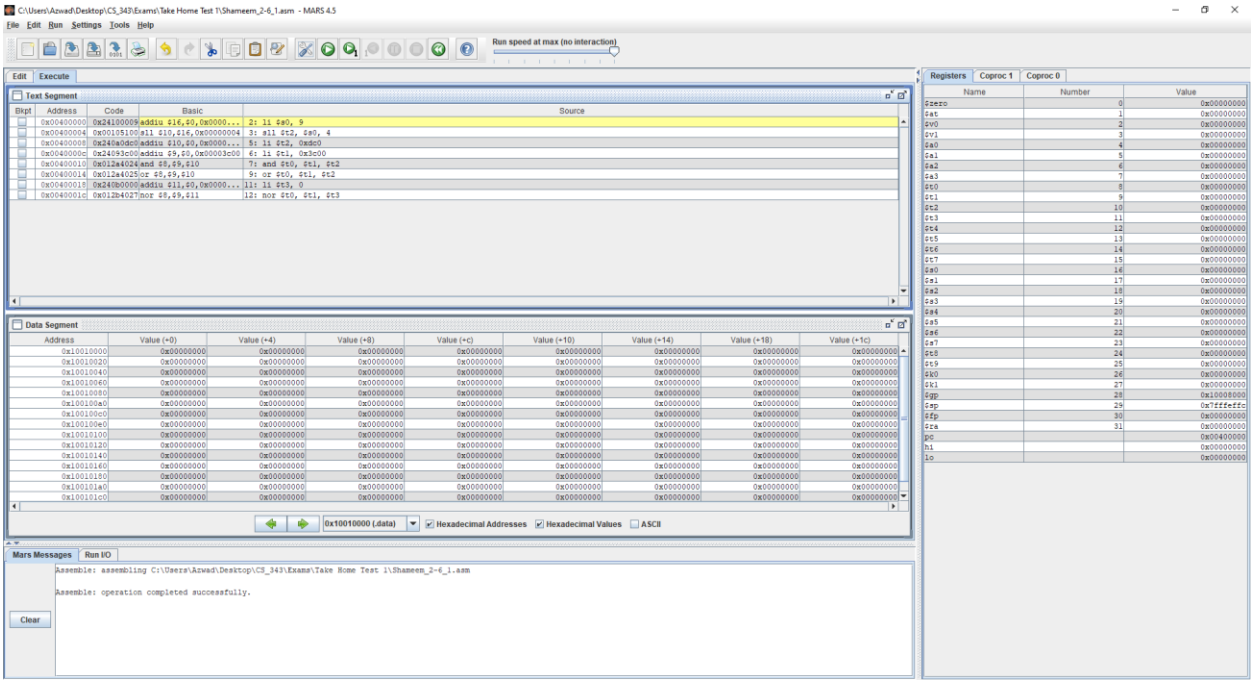


Figure 23: Shameem_2-6_1.asm code executed in MARS (lines 1-5)

Text Segment: This window shows that the code starts at line 2 because line 1 don't need registers.

Registers: This window shows that the value of $sp, which is 0x7FFFEFFC.

Data Segment: This window has not been updated at this stage.

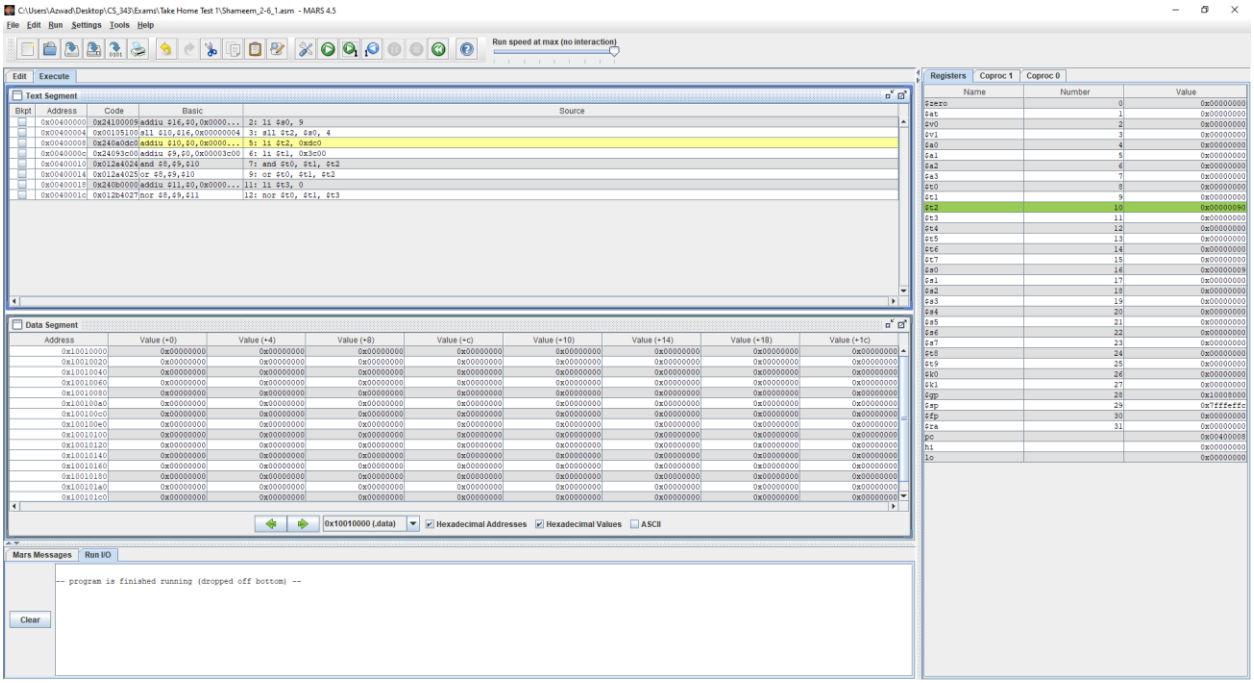


Figure 24: Shameem_2-6_1.asm code executed in MARS (lines 2-4)

Text Segment: This window shows that the code runs lines 2-4. At line 2 the static value is loaded into the register and at line 3 the result of Shift Left Logical has also been added to the register.

Registers: This window shows that the values have been updated.

> At the register $s0 the value of 9 is stored which is 0x00000009 in hexadecimal. Also, the Shift Left Logical is stored in $t2 and has the value of 0x00000090.

Data Segment: This window has not been updated at this stage.

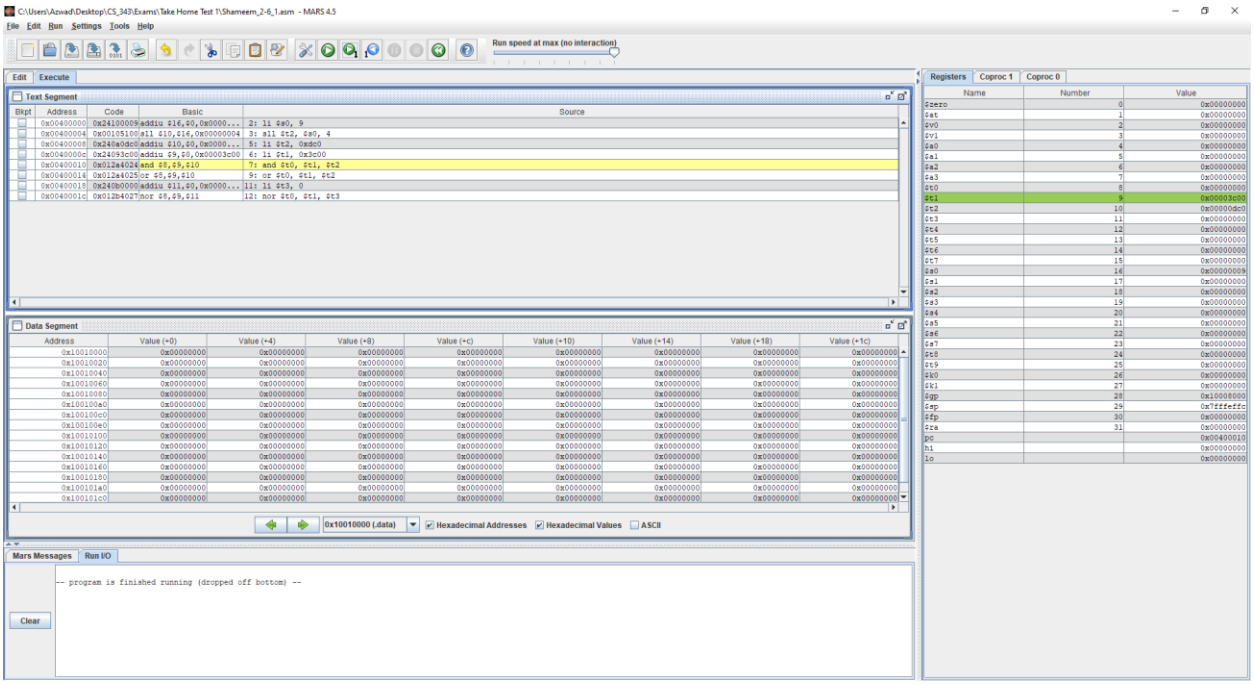


Figure 25: Shameem_2-6_1.asm code executed in MARS (lines 5-7)

Text Segment: This window shows that the code runs lines 5-7. At lines 5-6 the values are loaded into the registers $t1 and $t2 and at line 7 the values of $t1 and $t2 have been AND and stored into $t0 .

Registers: This window shows that the values have been updated.

At the register $t1 the value 0x3C00 is stored
Also, at the register $t2 the value 0xDC0 is stored.
Lastly, at the register $t0 the AND operator's result and $t1, $t2 is stored, which has the value of 0x00003C00

Data Segment: This window has not been updated at this stage.

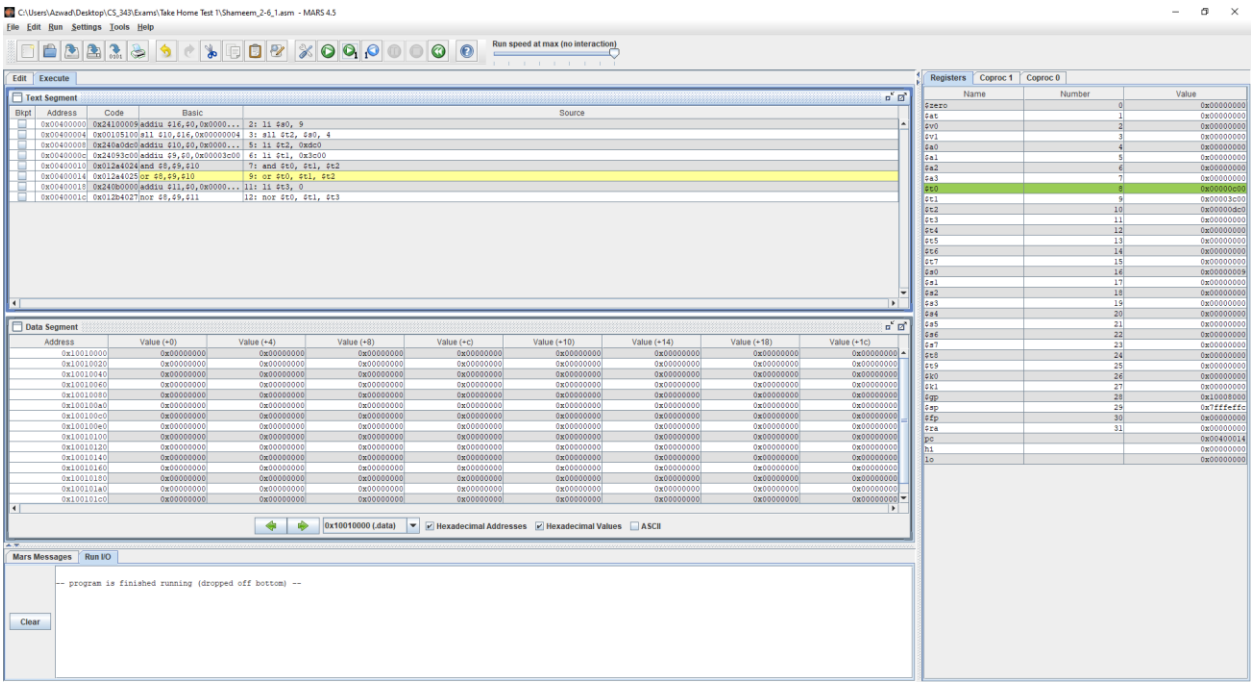


Figure 26: Shameem_2-6_1.asm code executed in MARS (line 9)

Text Segment: This window shows that the code runs lines 9. At line 9 the OR operation is performed with $t1 and $t2 and stored into $t0.

Registers: This window shows that the values at the register $t0 the value of the OR operation with $t1 and $t2, which is 0x00000C00 is stored.

Data Segment: This window has not been updated at this stage.

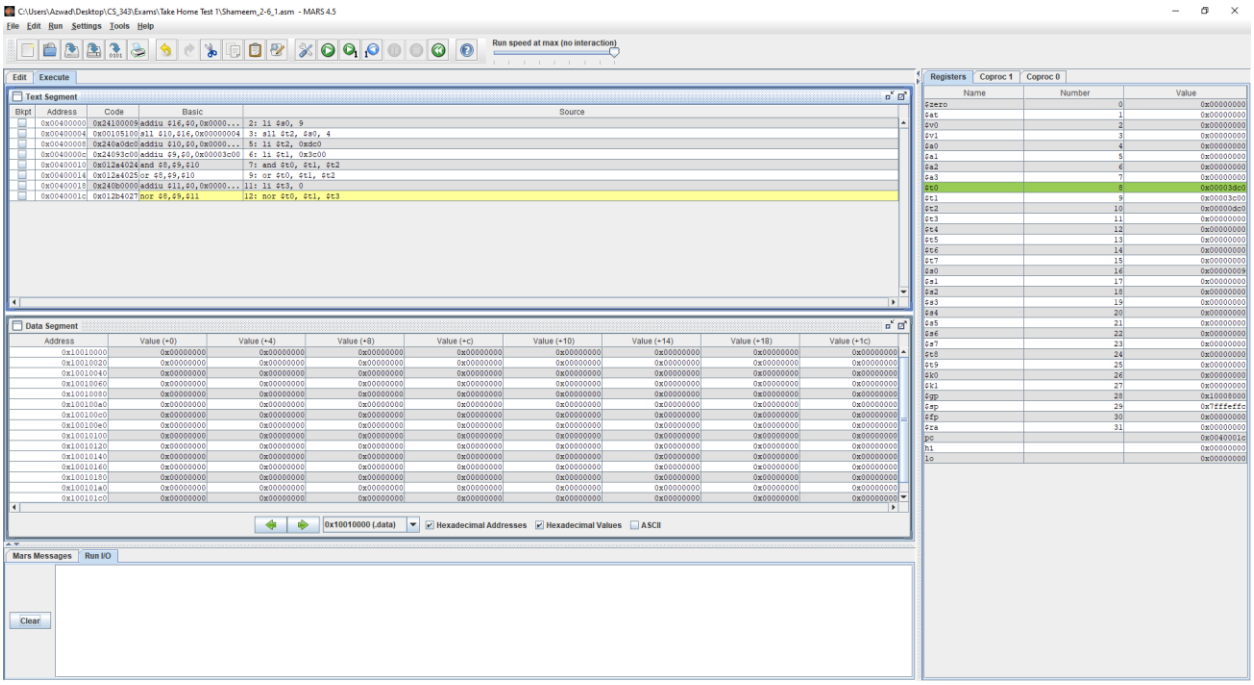


Figure 27: Shameem_2-6_1.asm code executed in MARS (line 11-12)

Text Segment: This window shows that the code runs lines 11-12. At line 11 the value of $t3 is loaded as 0 and at line 12 the NOR operation is used with $t1 and $t3 and stored at $t0.

Registers: This window shows that the values at the register $t0 the value of the NOR operation with $t1 and $t3, which is 0xFFFFCEFF is stored.

Data Segment: This window has not been updated at this stage.

2-7_1.asm

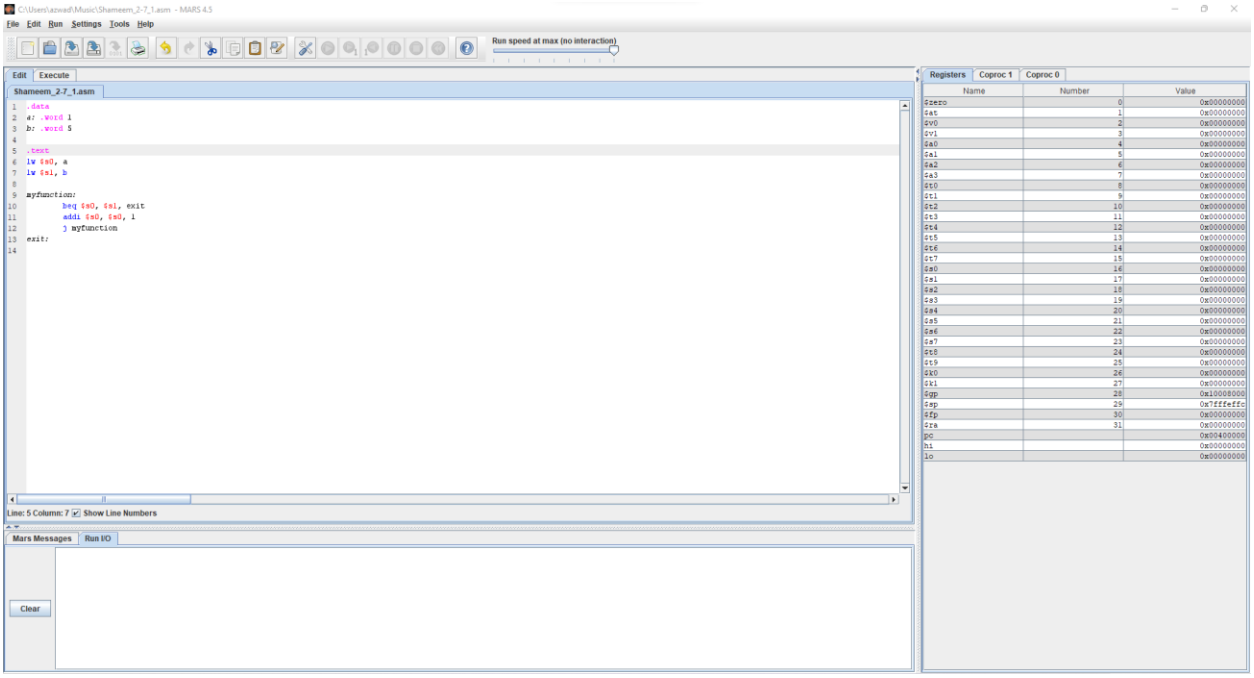


Figure 28: Shameem_2-7_1.asm code in MARS

The code written has two static variables a and b with the values 0 and 10 respectively. The first part of the code checks if the two values are equal and then they will exit otherwise if they are not equal then the value will increment.

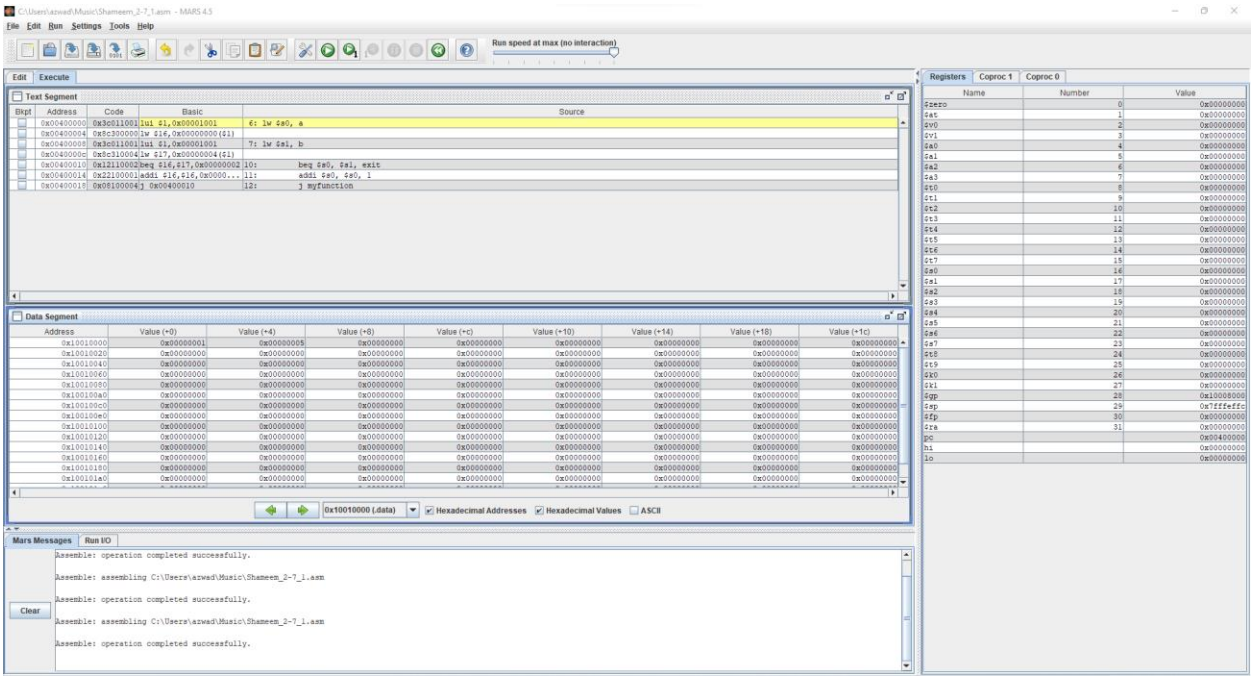


Figure 29: Shameem_2-7_1.asm code executed in MARS (lines 1-5)

Text Segment: This window shows that the code starts at line 6 because the previous lines don't use any registers.

Registers: This window shows that the value of $sp, which is 0x7FFFEFFC.

Data Segment: The window shows the values of the static variables.

At address 0x10010000 there is a value of 0x00000001.
At address 0x10010000 there is a value of 0x00000005.

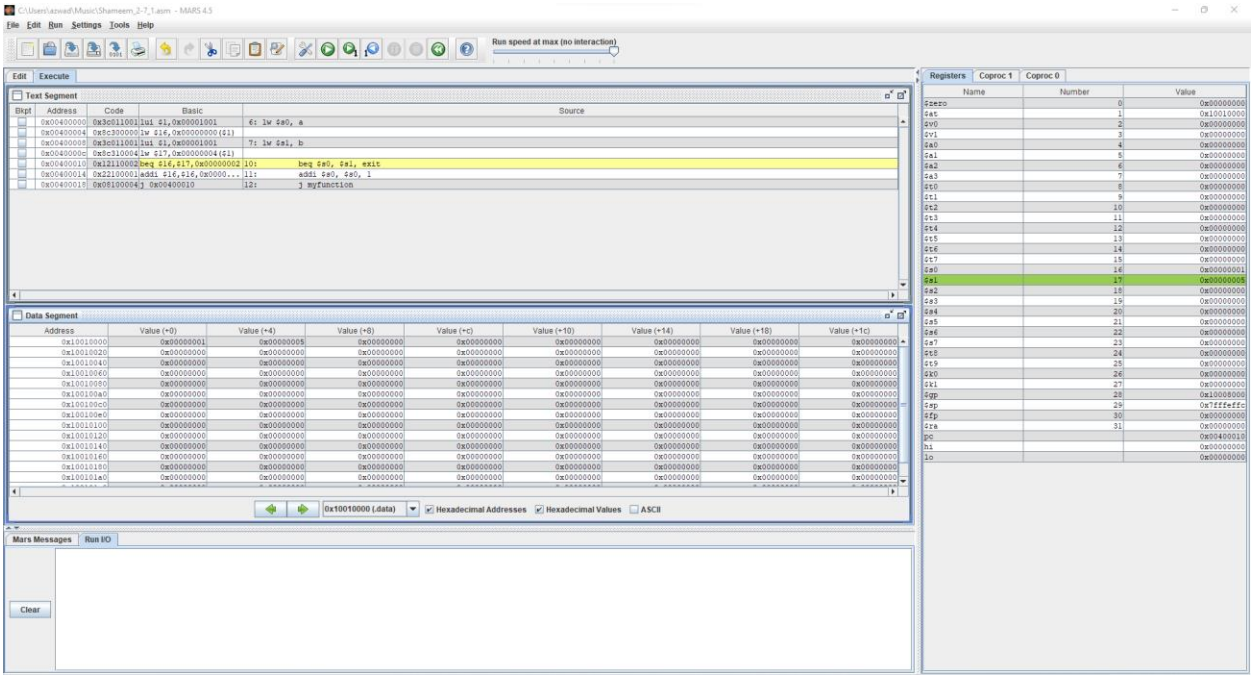


Figure 30: Shameem_2-7_1.asm code executed in MARS (lines 6-9)

Text Segment: This window shows that the code runs 6-9.

Registers: This window shows that $s0 is updated with the value of a which is 0x00000001 and $s1 is updated with the value of b which is 0x00000005.

Data Segment: The window shows the values of the static variables.

At address 0x10010000 there is a value of 0x00000001.
At address 0x10010000 there is a value of 0x00000005.

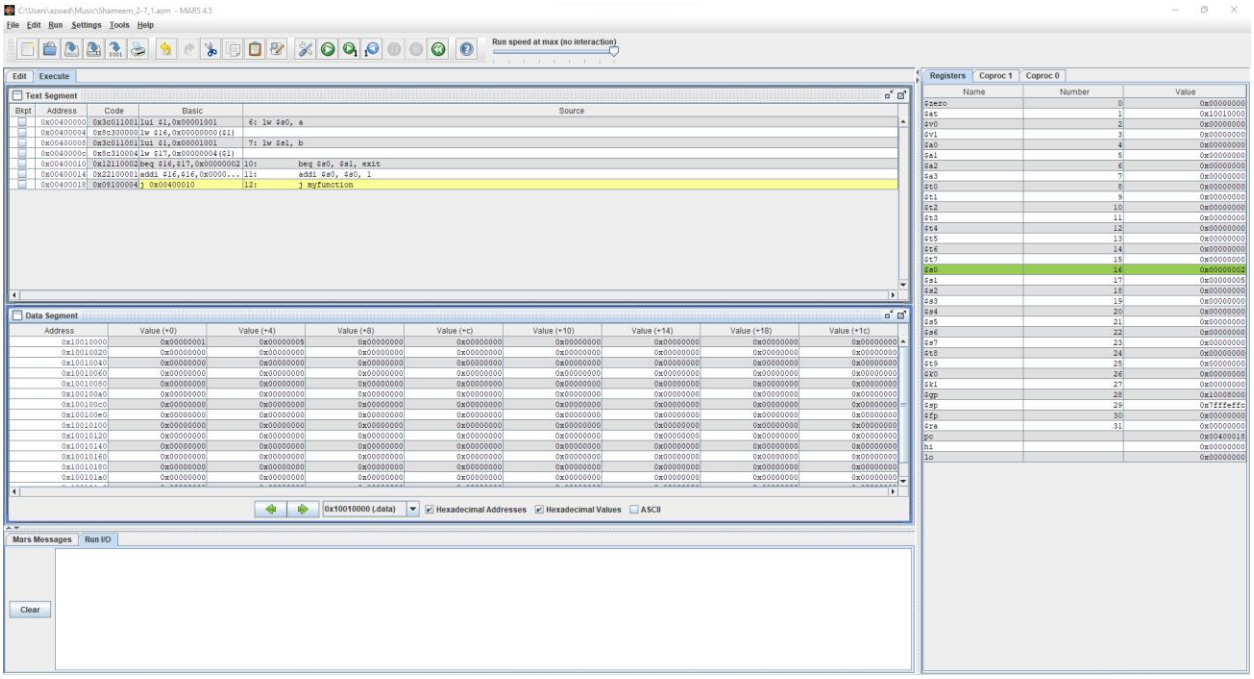


Figure 30: Shameem_2-7_1.asm code executed in MARS (lines 10-11)

Text Segment: This window shows that the code runs 10-11.

Registers: This window shows that $s0 is updated with the incremented value of a which is 0x00000002.

Data Segment: The window shows the values of the static variables.

At address 0x10010000 there is a value of 0x00000001.
At address 0x10010000 there is a value of 0x00000005.

This process will repeat until the value of a equals the value of b. When a equals b the program will exit.

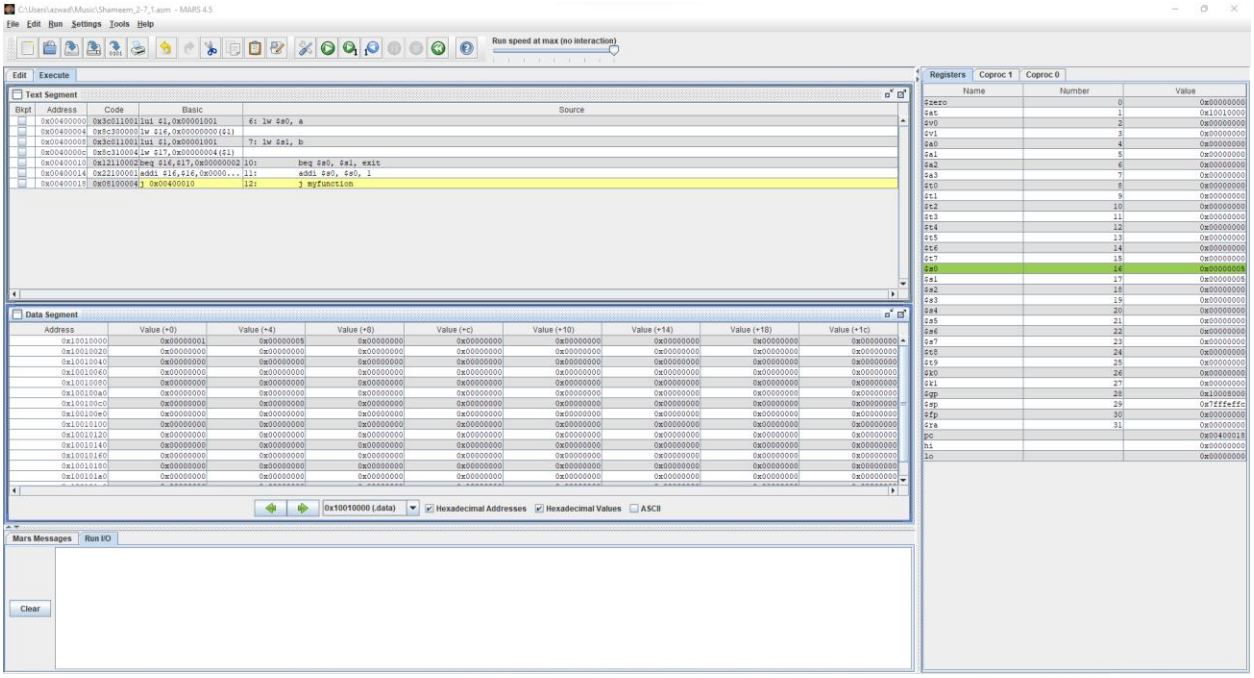


Figure 31: Shameem_2-7_1.asm code executed in MARS (lines 10-11)

Text Segment: This window shows that the code runs 10-11.

Registers: This window shows that $s0 is updated with the incremented value of a which is 0x00000005.

Data Segment: The window shows the values of the static variables.

At address 0x10010000 there is a value of 0x00000001.
At address 0x10010000 there is a value of 0x00000005.

Now a = b, so the next time when the program runs the program will exit because a=b.

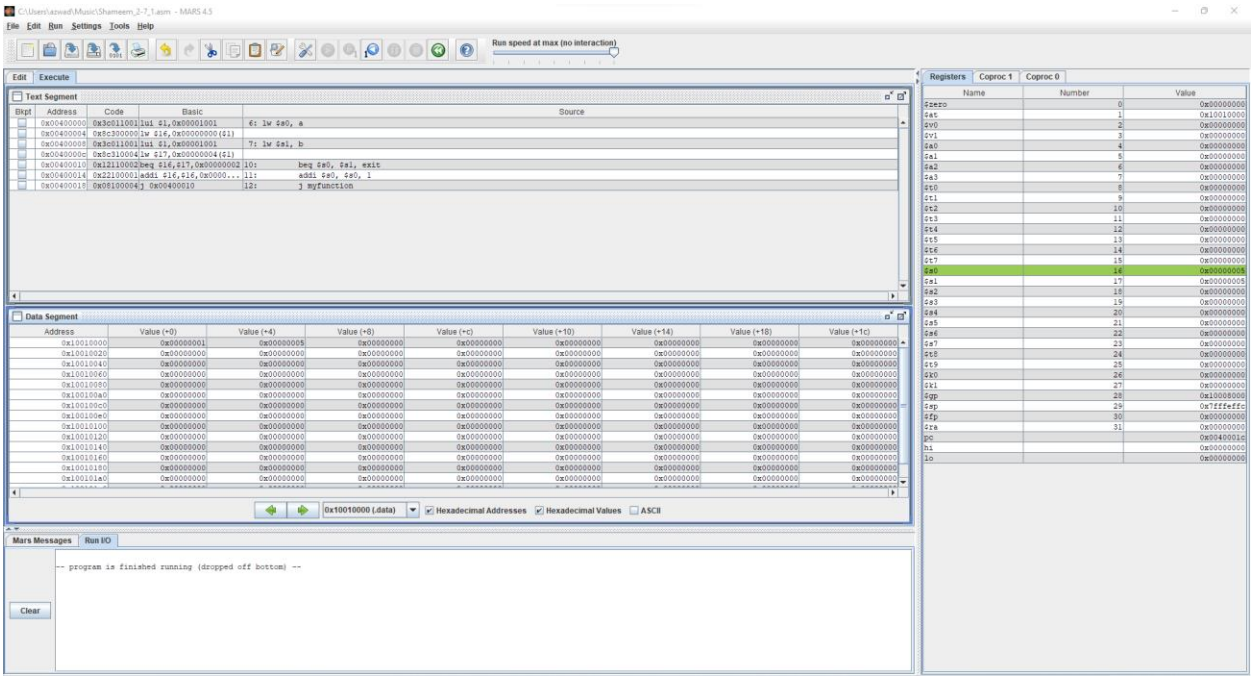


Figure 31: Shameem_2-7_1.asm code executed in MARS (lines 10)

Text Segment: This window shows that the code ran line 10.

Registers: This window does not change.

Data Segment: The window shows the values of the static variables.

At address 0x10010000 there is a value of 0x00000001.
At address 0x10010000 there is a value of 0x00000005.

The program has finished running because it has exited since the values of a = b which triggers the program to exit.

myadd.asm

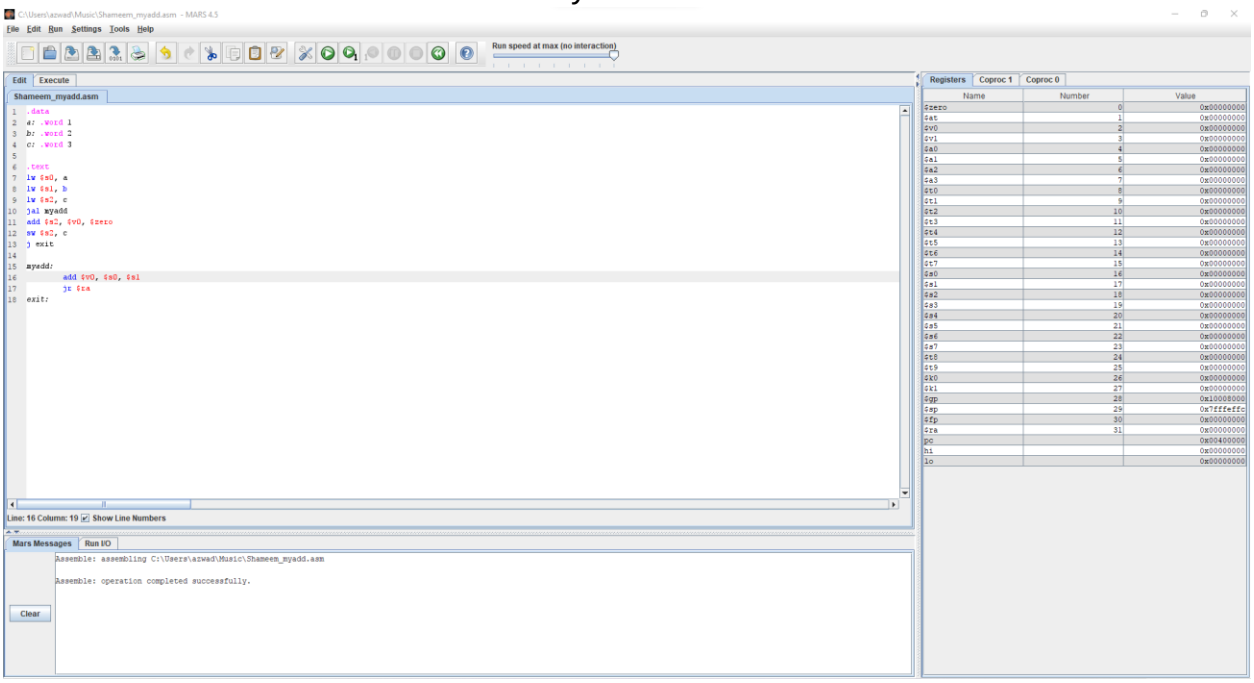


Figure 32: Shameem_myadd.asm code in MARS

The code has three static variables a, b and c and uses these static variables to load them into the registers. Then the code performs addition with $s0 and $s1 and stores the result into $v0. Lastly, the code then takes $v0 and $zero and adds them and stores the result into $s2 which is then saved into the variable c.

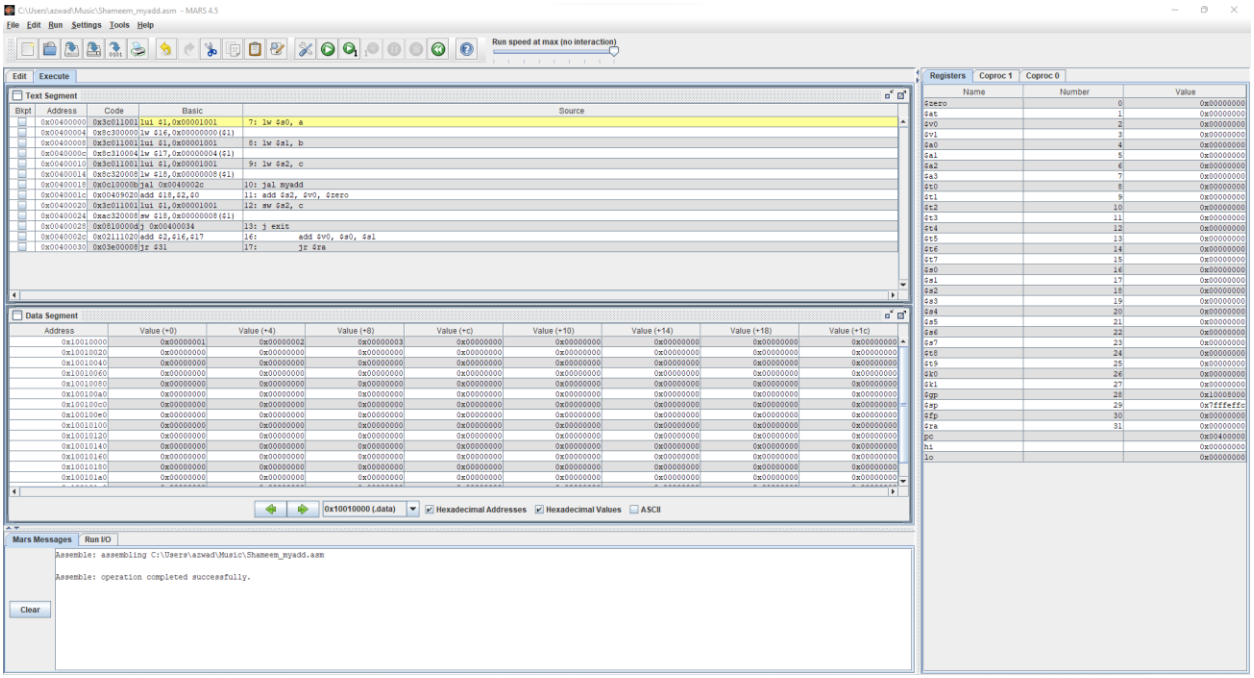


Figure 33: Shameem_myadd.asm code executed in MARS (lines 1-7)

Text Segment: This window shows that the code starts at line 7 because lines 1-6 don't need registers.

Registers: This window shows that the value of $sp, which is 0x7FFFEFFC.

Data Segment: This window shows the values of the static variables.

The static variable a is stored in address 0x10010000 and has the value 0x00000001.

The static variable b is stored in address 0x10010004 and has the value 0x00000002.

The static variable c is stored in address 0x10010008 and has the value 0x00000003.

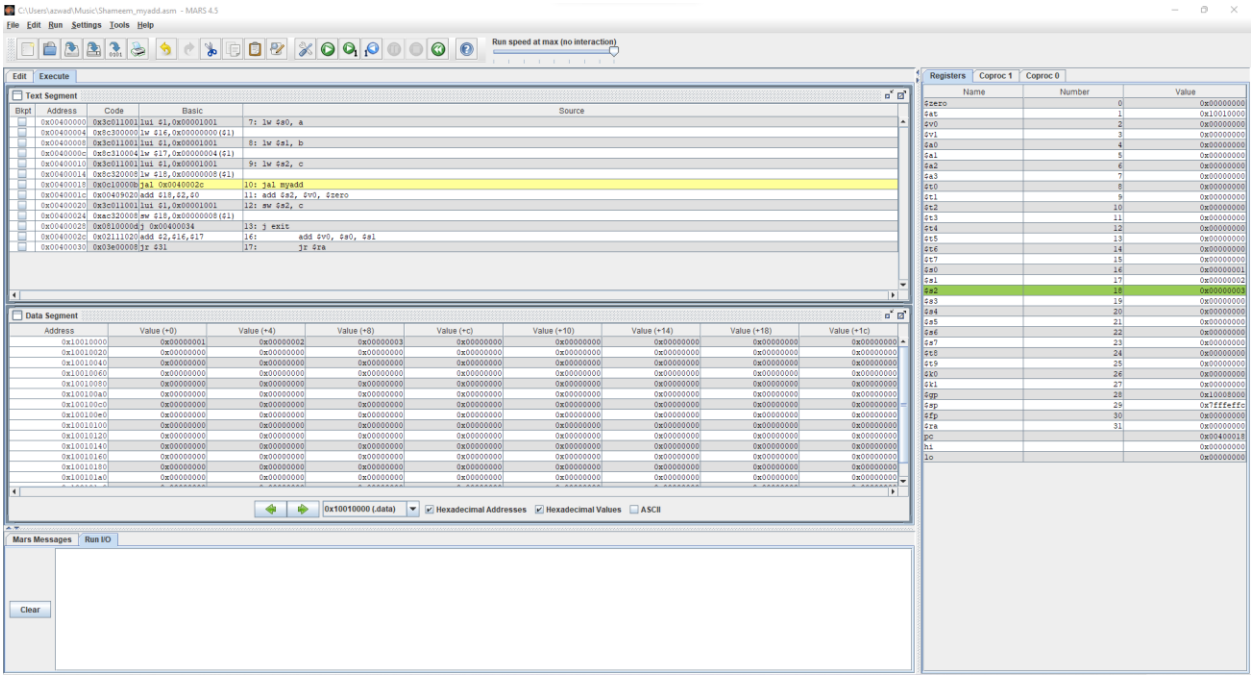


Figure 34: Shameem_myadd.asm code executed in MARS (lines 7-9)

Text Segment: This window shows that the code runs line 7-9 Lines 7-9 stores the static value into the registers.

Registers: This window shows that the values of the static variable in the registers.

The value of variable a which is 0x00000001 is stored at register $s0.
The value of variable b which is 0x00000002 is stored at register $s1.
The value of variable c which is 0x00000003 is stored at register $s2.

Data Segment: This window has not changed because lines 7-9 don't store anything.

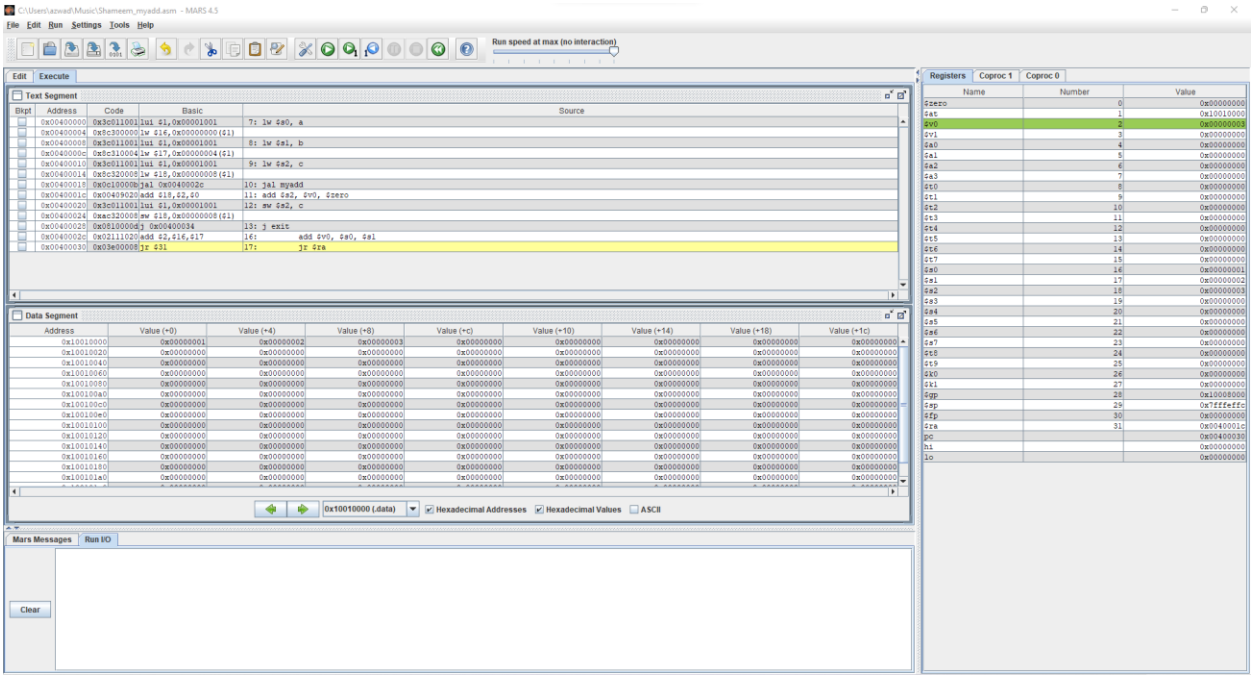


Figure 35: Shameem_myadd.asm code executed in MARS (lines 10 & 16)

Text Segment: This window shows that the code runs line 10 & 16. Line 10 goes to myadd and line 16 does addition of $s0 and $s1 and stores the result into $v0.

Registers: This window shows that the value of the addition of $s0 and $s1 is in the register $v0.

Data Segment: This window has not changed because lines 10 & 16 don't store anything.

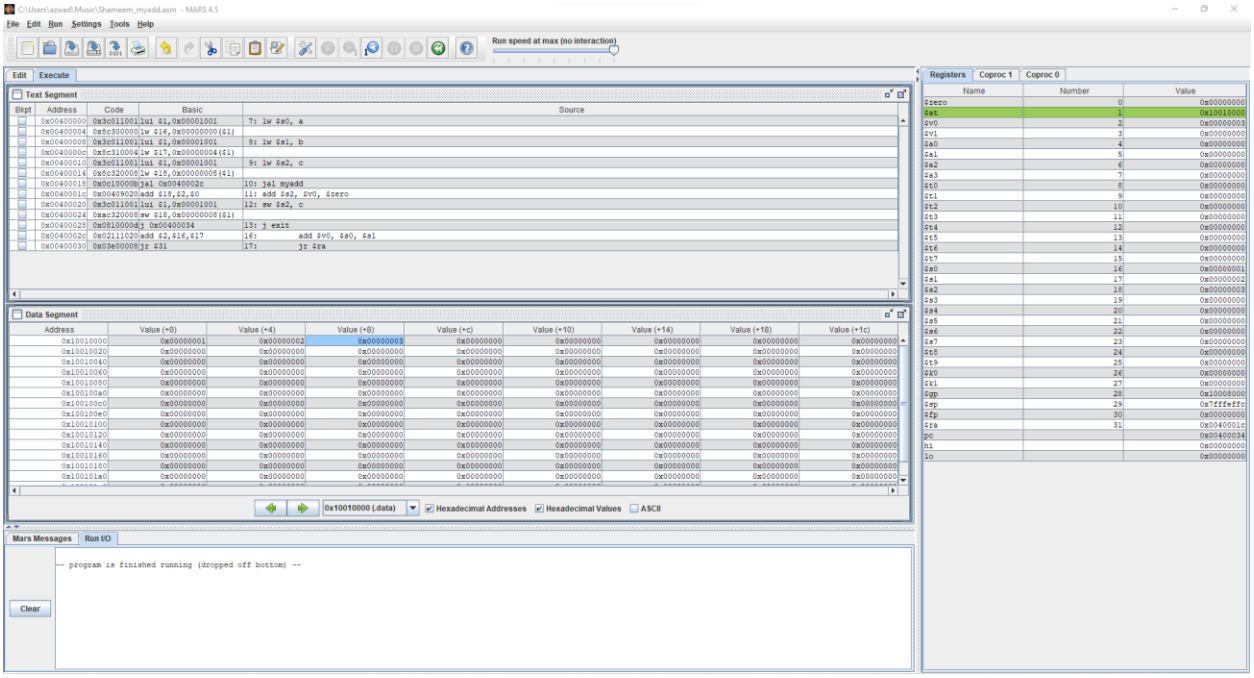


Figure 36: Shameem_myadd.asm code executed in MARS (lines 17 & 11-12)

Text Segment: This window shows that the code runs line 17 & 11-12. Line 17 jumps out of the function and lines 11-12 compute the addition of $v0 and $zero and store the result into $s2 and then load the result of $s2 into the variable c.

Registers: This window shows that the value of the addition of $v0 and $zero is in the register $s2.

Data Segment: This window shows the addition of $v0 and $zero which is stored in $s2 is stored into the variable c which is stored at the address 0x1001008 and has the value 0x00000003.

Intel X32 ISA Windows 32-bit

2-2_1.c

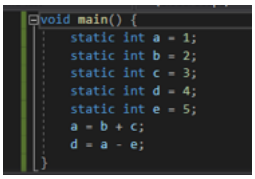


Figure 37: Shameem_2-2_1.c code in Visual Studio

In the code we have 5 static variables a, b, c, d and e with the values of 1, 2, 3, 4 and 5 respectively. The code then uses the variables b and c and adds them and puts the result in variable a. The code also subtracts a and e and puts that result in the variable d.

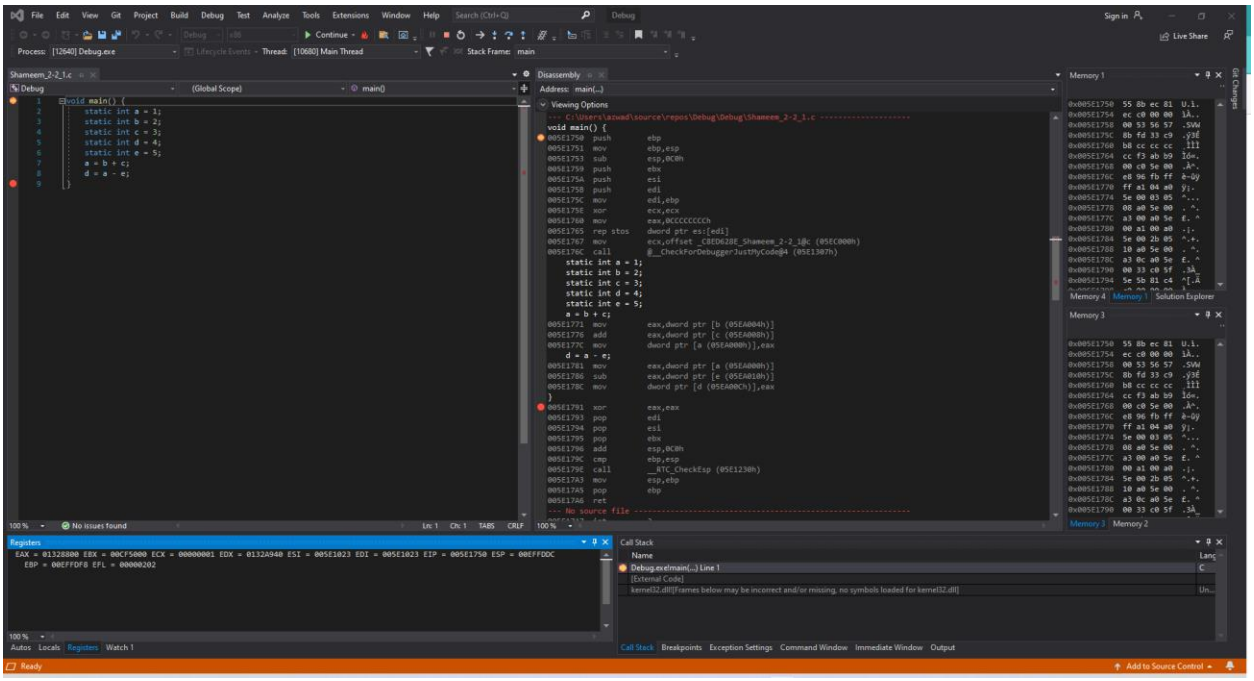


Figure 38: Shameem_2-2_1.c code in Debugger mode in Visual Studio

Disassembly: This window shows that the code starts at line 1.

Registers: This window shows that the value of ESP, the stack pointer, is 0x004FF778.

Memory: Memory window 1 shows the values around the stack pointer which is random. While Memory window 2 displays the value of the static variables stored into the addresses.
Variable a is stored in address 0x005EA000 and has the value 01 00 00 00.
Variable b is stored in address 0x005EA004 and has the value 02 00 00 00.
Variable c is stored in address 0x005EA008 and has the value 03 00 00 00.
Variable d is stored in address 0x005EA00C and has the value 04 00 00 00.
Variable e is stored in address 0x005EA010 and has the value 05 00 00 00.

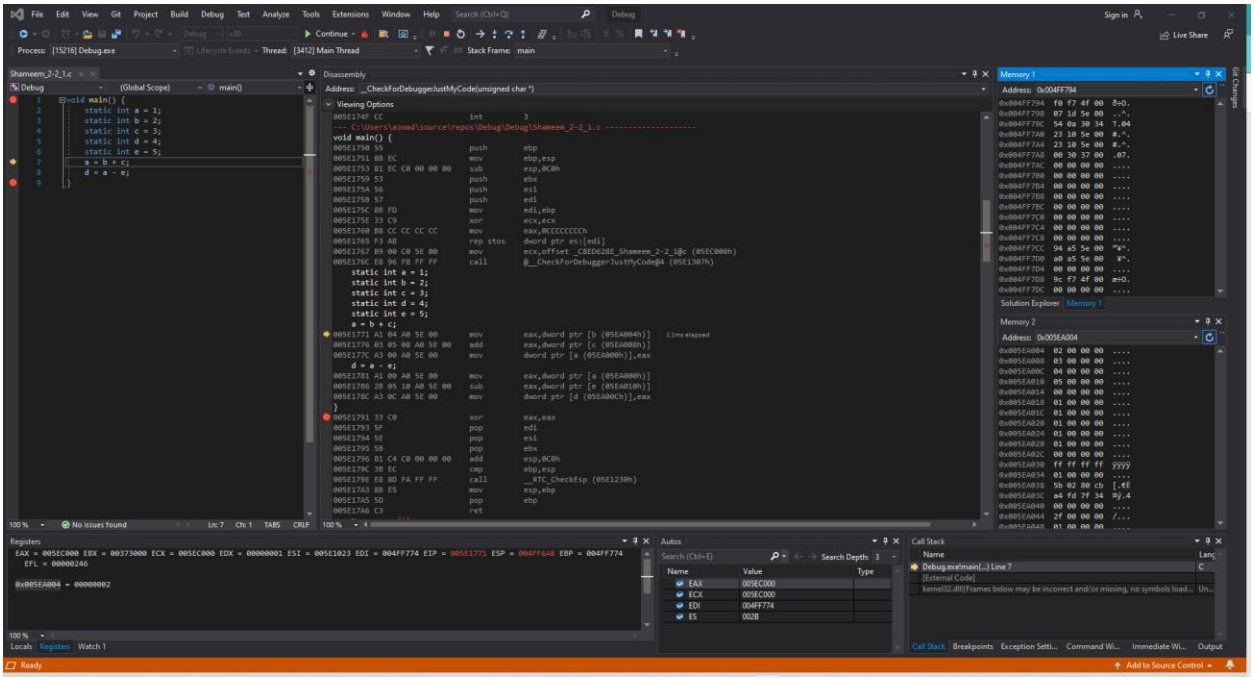


Figure 39: Shameem_2-2_1.c code in Debugger mode in Visual Studio (Lines 1-6)

Disassembly: This window shows that the code runs lines 1-6 and that the registers and memory is set to have the appropriate values to start the addition.

Registers: This window shows the value that stored in each register which has changed from before because it is setting up registers for the program to run

Memory: Memory window 1 shows the values around the stack pointer which only some are initialized now. While Memory window 2 displays the value of the static variables stored into the addresses, but since we don't have any more static variables to store it didn't change anything.

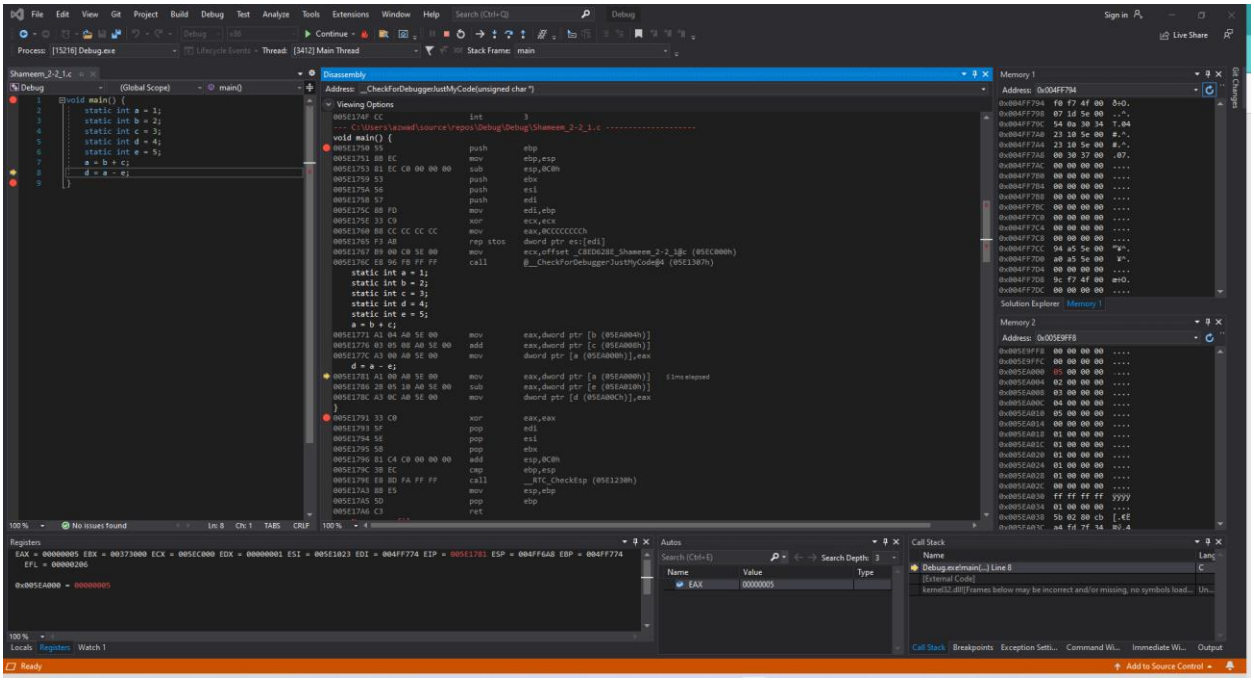


Figure 40: Shameem_2-2_1.c code in Debugger mode in Visual Studio (Lines 7)

Disassembly: This window shows that the code runs line 7 and that the registers and memory moved a value into the register so that it can be added to another value which the result is then saved into memory.

Registers: This window shows the value stored in each register.

The value of b is moved to the register.

The value in that register is moved into the memory and stored in the address of a which overwrites the previous value of a.

Memory: Memory window 1 shows that the stack pointer stays the same since line 7 didn't store anything. While memory window 2 displays the new value of the static variable a at address 0x005EA000 with the value of 05 00 00 00.

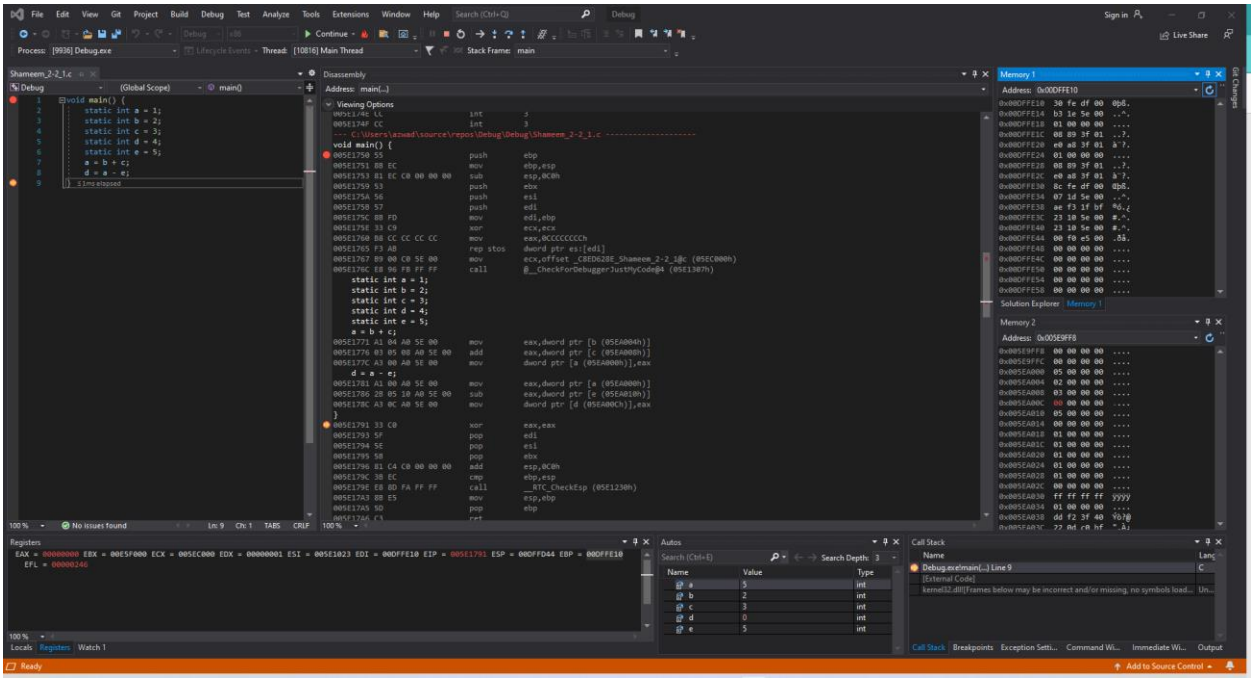


Figure 41: Shameem_2-2_1.c code in Debugger mode in Visual Studio (Lines 8-9)

Disassembly: This window shows that the code runs lines 8-9 and the value was moved into the register so it can be subtracted with another value which would lead the result to be saved in memory.

Registers: This window shows that the value of a that was moved to the register has been subtracted with the value of e and then moved into memory and stored in the address of d which overwrites the value at d.

Memory: Memory window 1 shows the values around the stack pointer which only some are initialized now. While Memory window 2 displays the value of the static variable d is stored in the address 0x005EA00C with the value of 00 00 00 00.

2-2_2.c

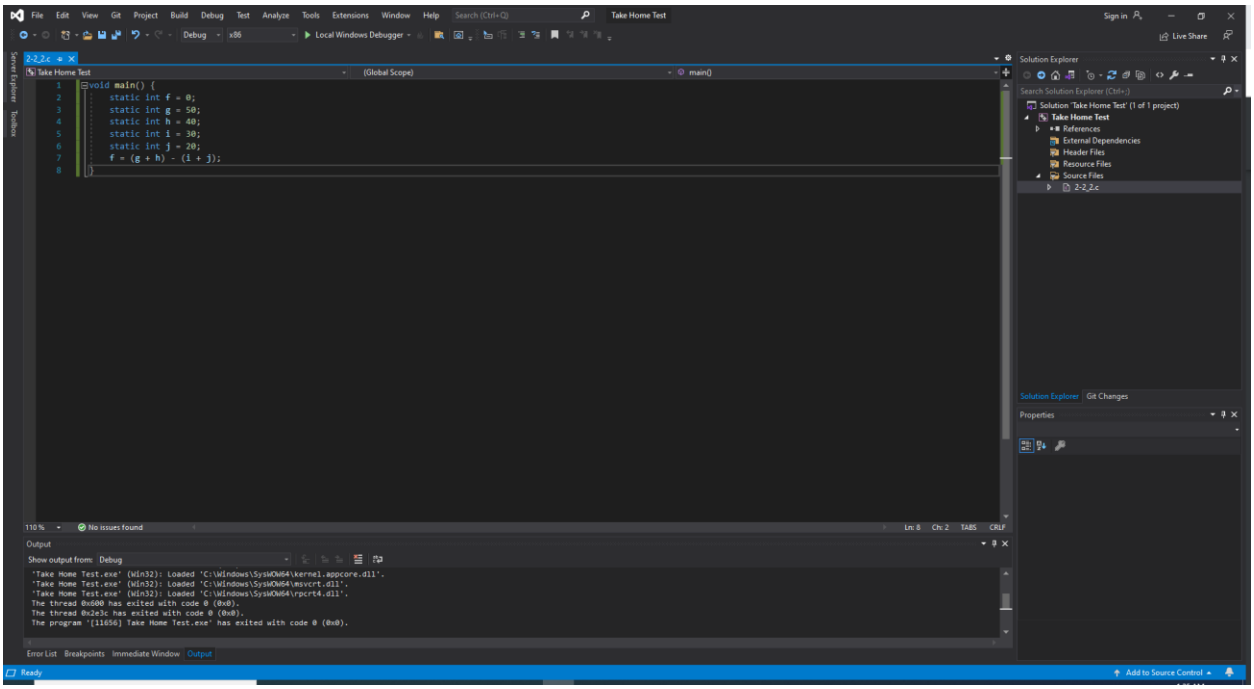


Figure 42: Shameem_2-2_2.c code in Visual Studio

In the code we have 5 static variables f, g, h, I, j with the corresponding values of 0, 50, 40, 30 and 20 respectively. The code then uses the values of the variables to perform two additions and subtractions. The additions entail g + h and i + j which the result of the two additions is subtracted and stored into the variable f.

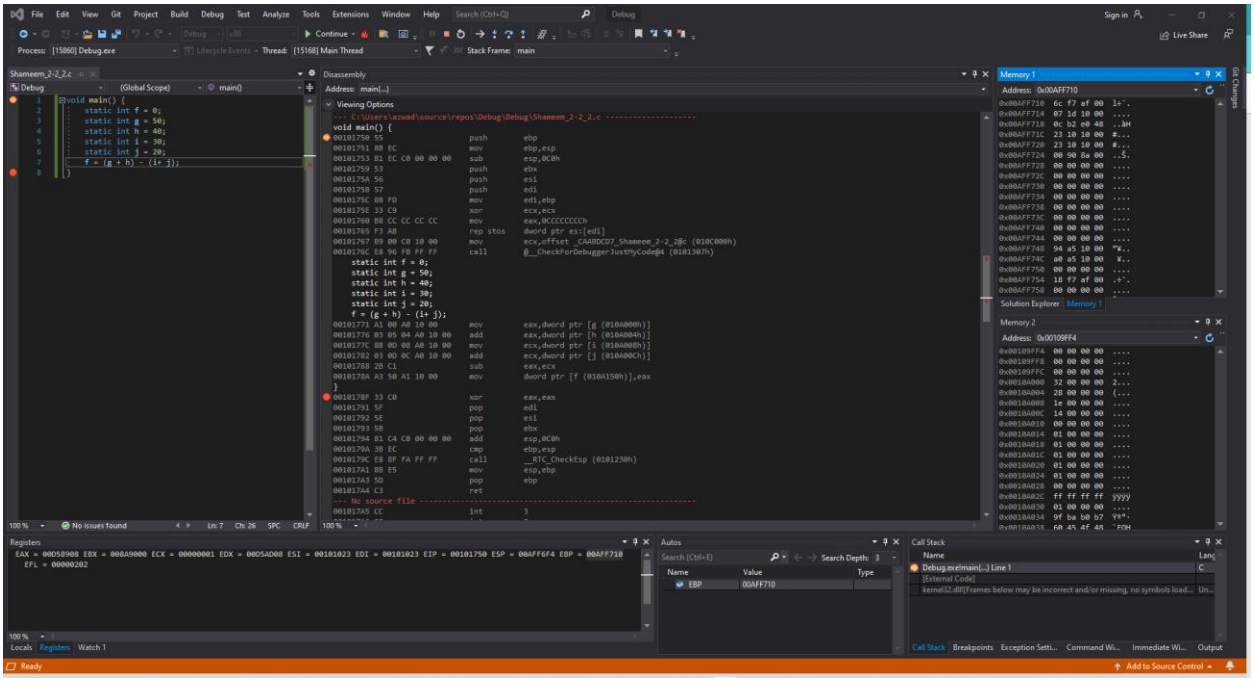


Figure 43: Shameem_2-2_2.c code in Debugger mode in Visual Studio

Disassembly: This window shows that the code starts at line 1.

Registers: This window shows that the value of ESP, the stack pointer, is 0x004FF778.

Memory: Memory window 1 shows the values around the stack pointer which is random. While Memory window 2 displays the value of the static variables stored into the addresses.

Variable f is stored in address 0x00109FFC and has the value 00 00 00 00.
Variable g is stored in address 0x0010A000 and has the value 32 00 00 00.
Variable h is stored in address 0x0010A004 and has the value 28 00 00 00.
Variable i is stored in address 0x0010A008 and has the value 1e 00 00 00.
Variable j is stored in address 0x0010A00C and has the value 14 00 00 00.

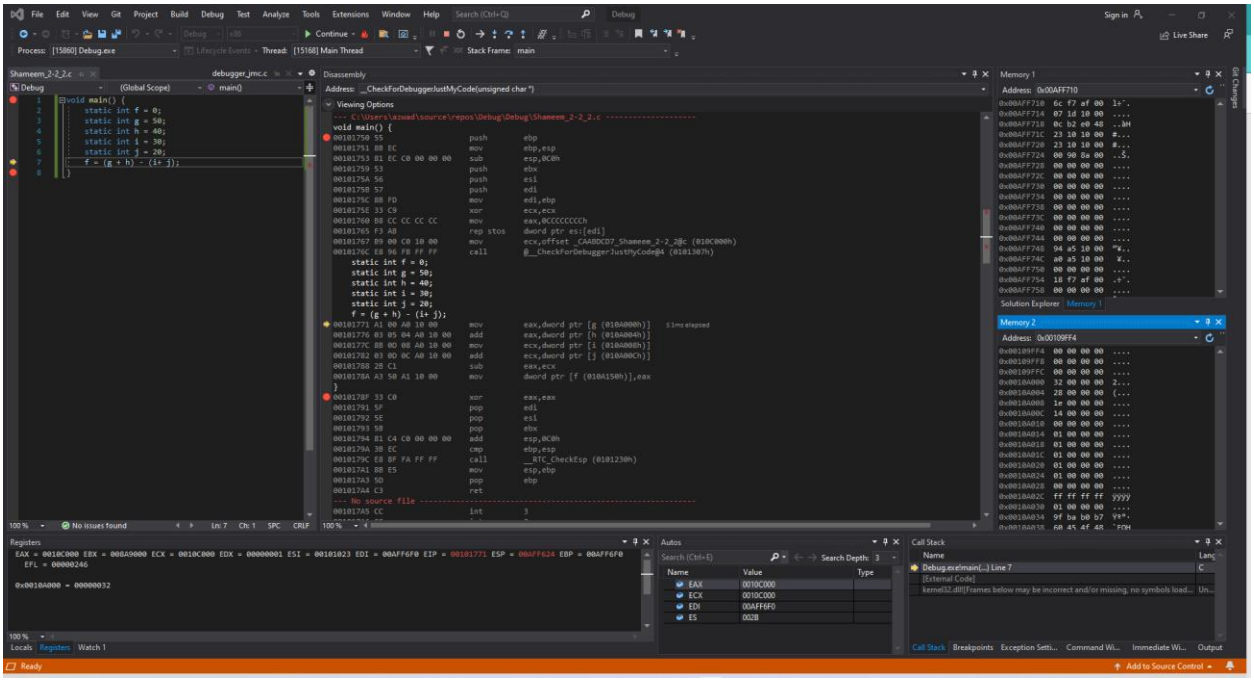


Figure 44: Shameem_2-2_2.c code in Debugger mode in Visual Studio

Disassembly: This window shows that the code ran lines 1-6.

Registers: This window shows that the values that are stored in each register has been slightly changed from before to set up registers for the program to run.

Memory: Memory window 1 shows the values around the stack pointer which some are initialized. While Memory window 2 shows no changes in its display because lines 1-6 do not store anything in address or change values.

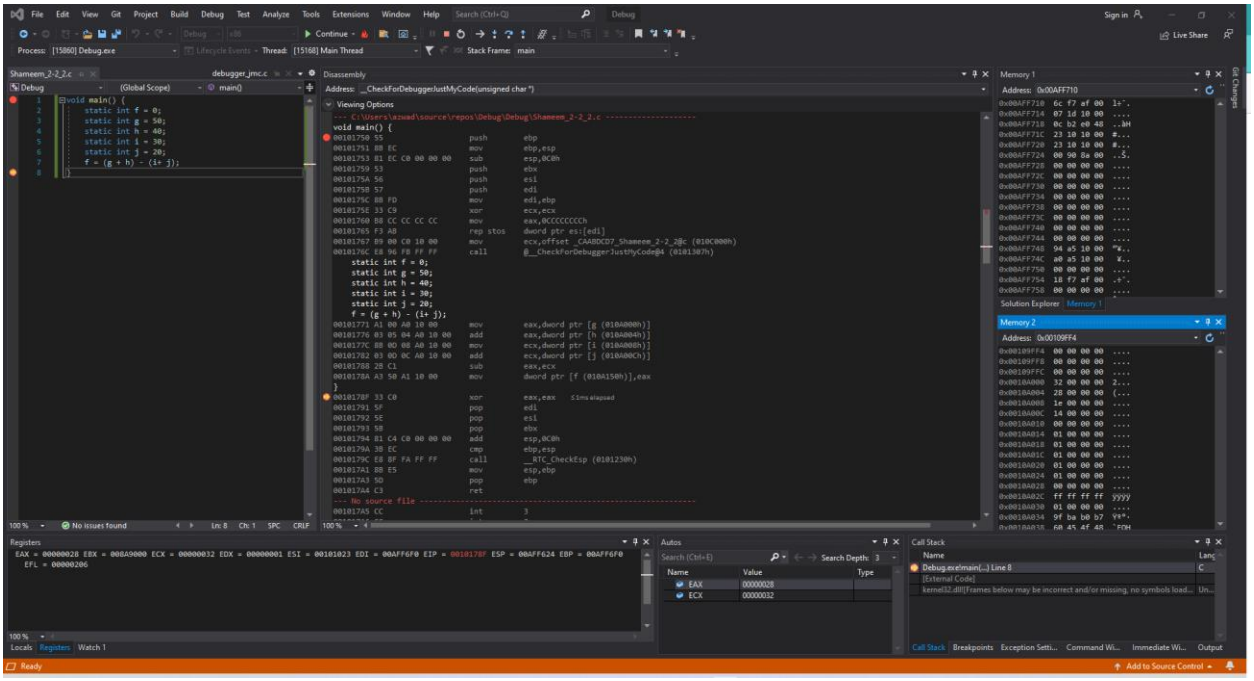


Figure 45: Shameem_2-2_2.c code in Debugger mode in Visual Studio

Disassembly: This window shows that the code ran lines 8.

Registers: This window shows that the values that are stored in each register.

For the value of g in memory, which is moved to the register with the value of the addition of that the value in register with value of h in memory.

For the value of I in memory, which is moved to the register with the value of the addition of that the value in register with value of j in memory.

The values in the two registers are subtracted which then is moved into the memory and stored in the address of f which overwrites the previous value at f.

Memory: Memory window 1 shows the values around the stack pointer remains the same. While Memory window 2 displays the variable f which is stored at address 0x00109FFC with the value 28 00 00 00.

2-3_1.c

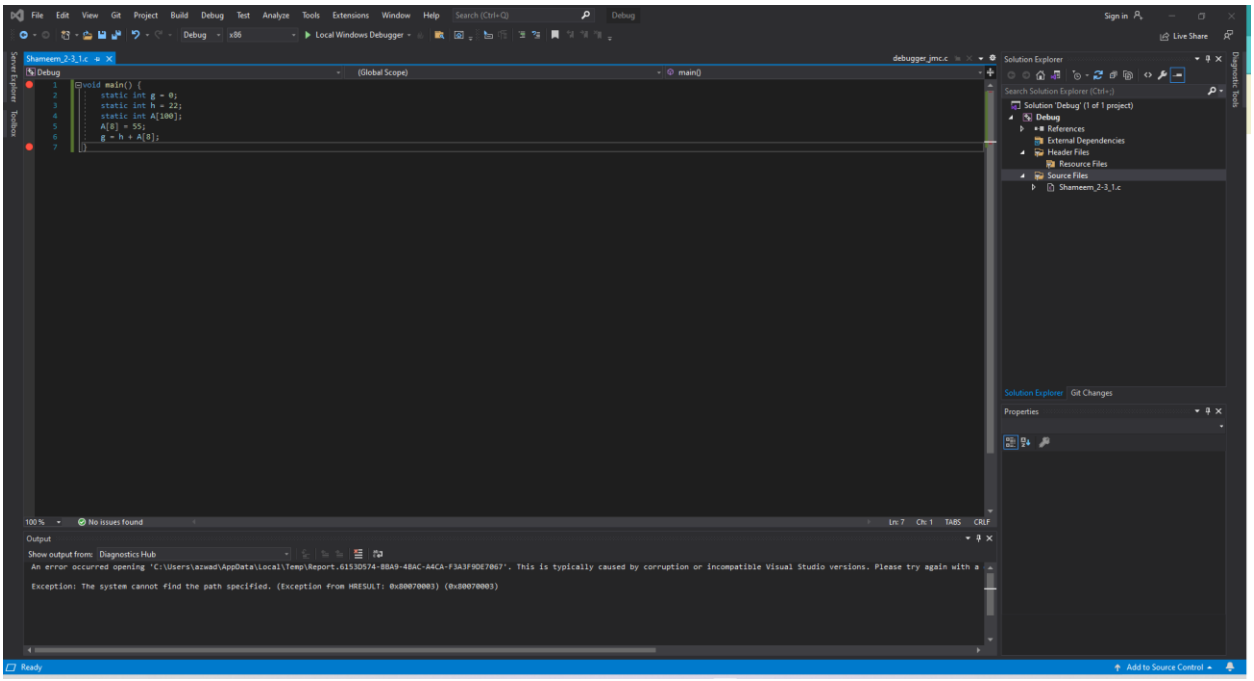


Figure 46: Shameem_2-3_1.c code in Visual Studio

The code has two static variables g and h plus the array A. The code uses the static variables to set the 8th index of the array A to the value 55 and then sets the value of g to the result of the addition of h and the 8th index of the array A.

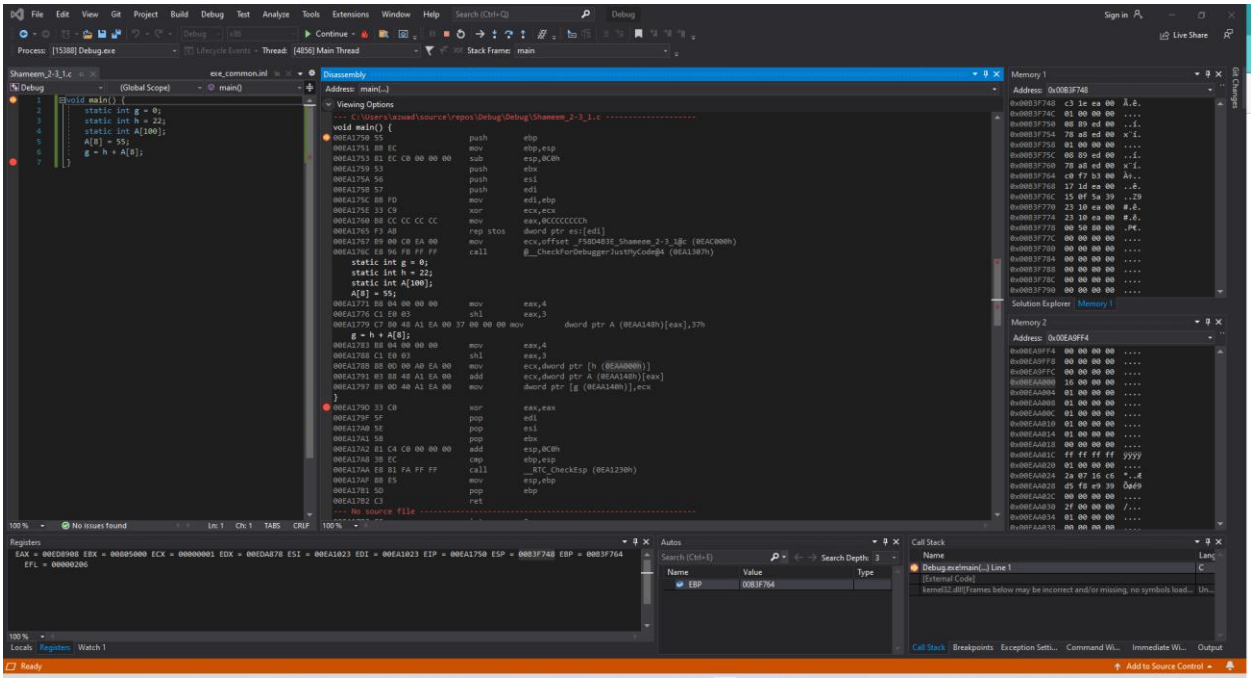


Figure 47: Shameem_2-3_1.c code ran in Visual Studio (Line 1)

Disassembly: This window shows that the code starts at line 1.

Registers: This window shows that the value of ESP, the stack pointer, is 0x00B3F748.

Memory: Memory window 1 shows the values around the stack pointer which is random. While Memory window 2 displays the value of the static variables stored into the addresses.

Variable g is stored at address 0x00EA9FFCand has the value 00 00 00 00.
Variable h is stored at address 0x00EAA000 and has the value 16 00 00 00.
Variable A is stored at address 0x00EAA168 and has the value 00 00 00 00.

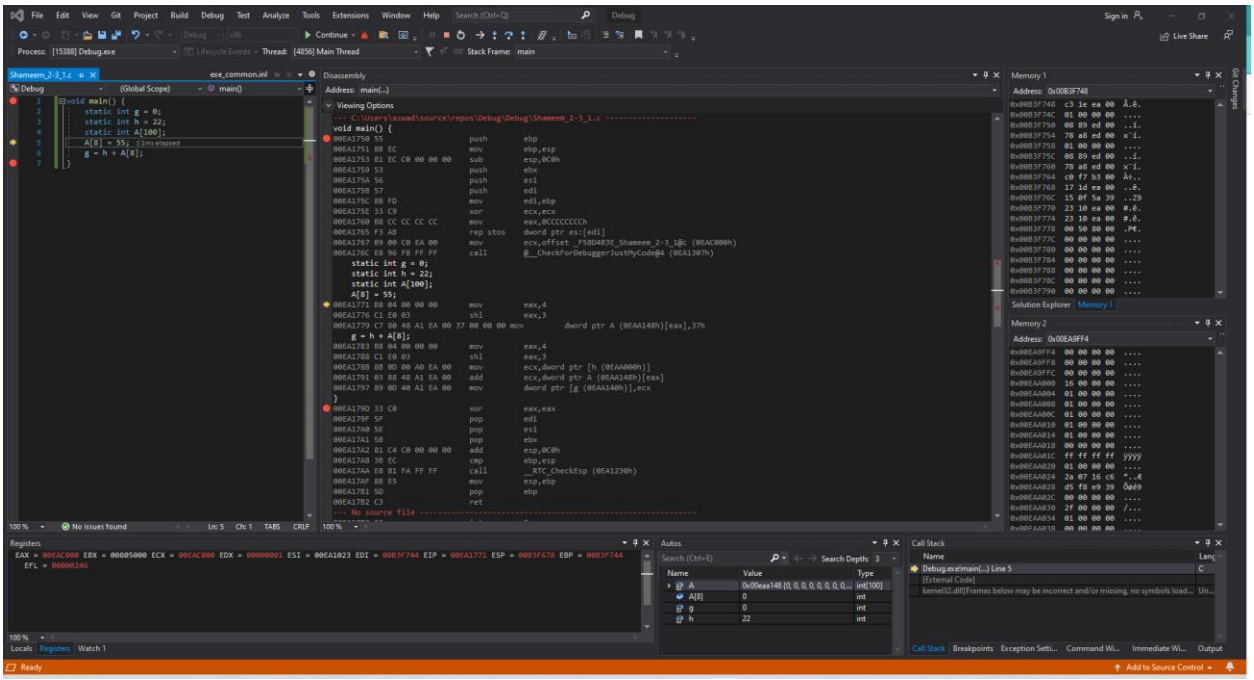


Figure 48: Shameem_2-3_1.c code ran in Visual Studio (Line 1-4)

Disassembly: This window shows that the code ran lines 1-4.

Registers: This window shows that the values that are stored in each register has been slightly changed from before to set up registers for the program to run.

Memory: Memory window 1 shows the values around the stack pointer which some are initialized. While Memory window 2 shows no changes in its display because lines 1-4 do not store anything in address or change values.

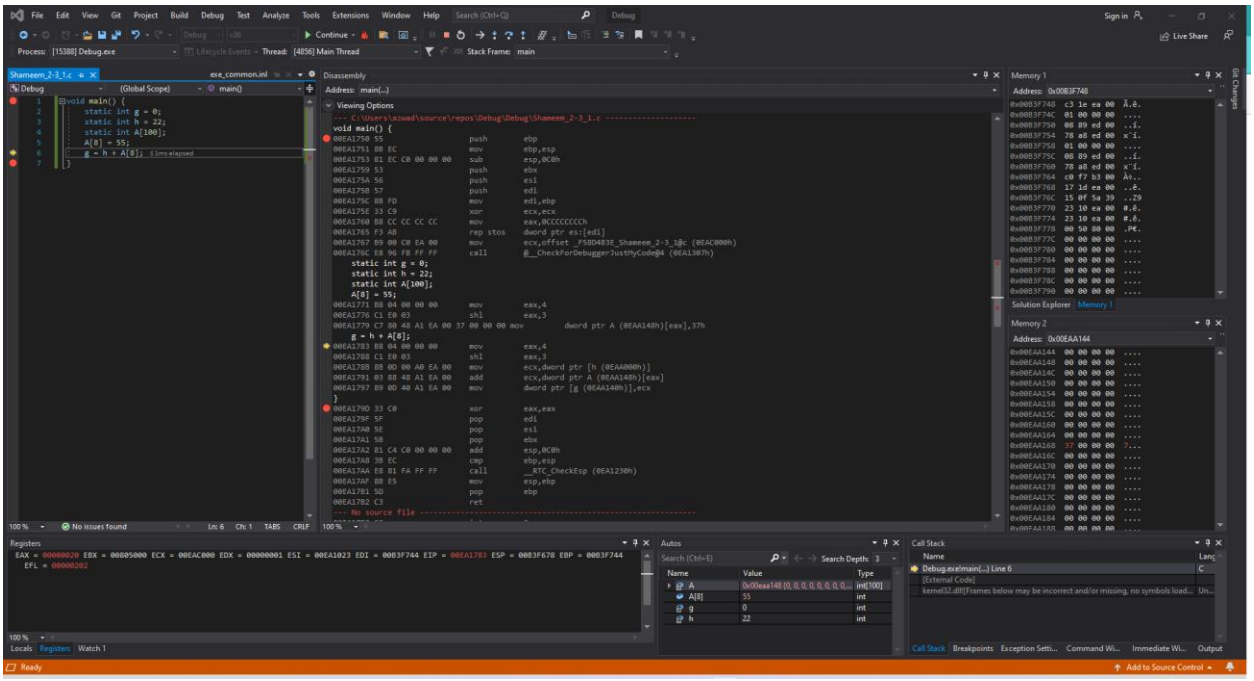


Figure 49: Shameem_2-2_2.c code ran in Debugger mode in Visual Studio (Line 5)

Disassembly: This window shows that the code ran lines 5. The 8th index of the array A has been inputted with the value of 55 into the memory.

Registers: This window shows that the values that the register EAX set to 00000004 and then shifted by 3 which makes it become 000000020 because 4 * 8 equals 20.

Memory: Memory window 1 shows the values around the stack pointer remains the same. While Memory window 2 displays the value stored in Array A at address 0x00EAA168 with a value of 37 00 00 00.

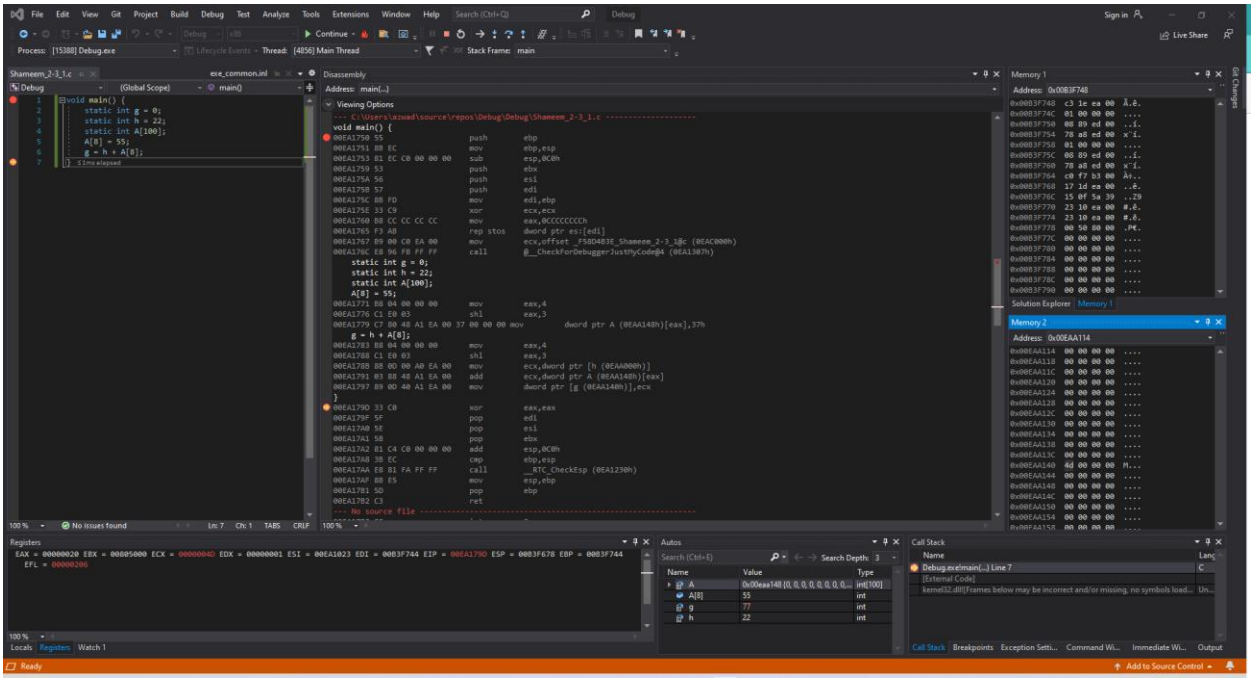


Figure 50: Shameem_2-2_2.c code ran in Debugger mode in Visual Studio (Line 6)

Disassembly: This window shows that the code ran lines 6. The value of g will be stored as the addition of the value of g and the value at the 8th index of the array A.

Registers: This window shows that the value for ECX is set to the value of h which is then added by the 8th index of the array.

Memory: Memory window 1 shows the values around the stack pointer remains the same. While Memory window 2 displays the value stored in Array A at address 0x00EAA168 with a value of 4d 00 00 00.

2-3_2.c

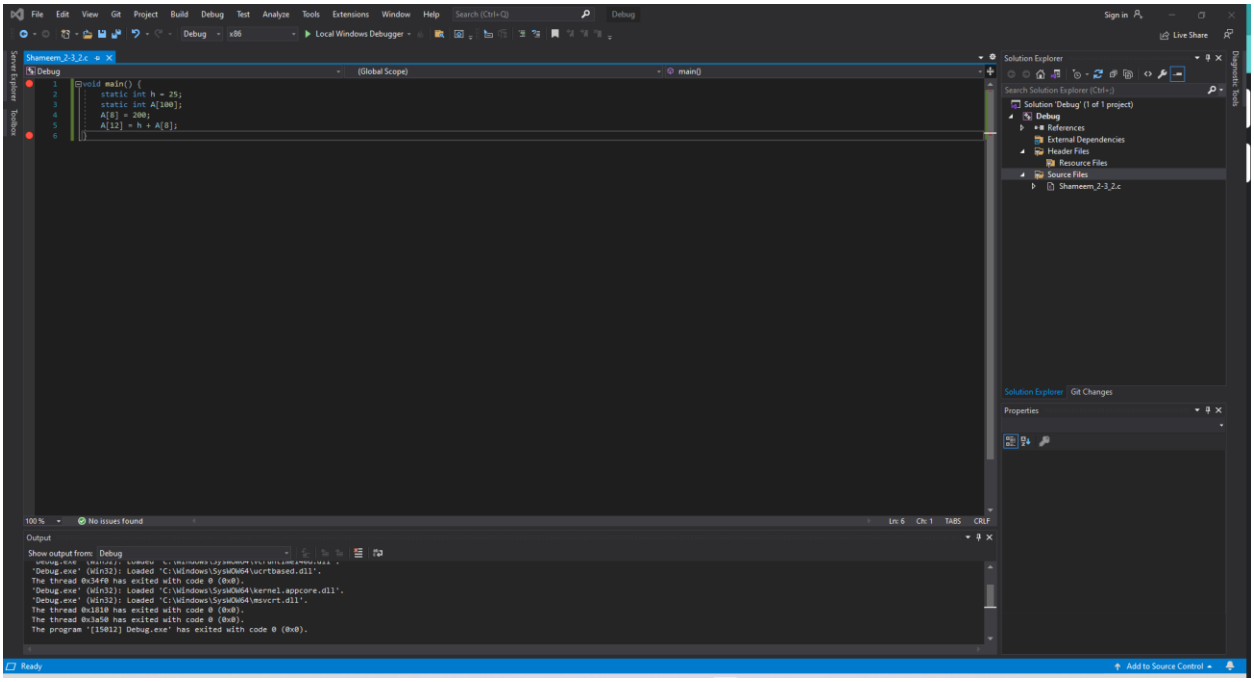


Figure 51: Shameem_2-3_2.c code in Visual Studio

The code has one static variable h and the array A. The code then sets the 8th index of the array to 200. Then the code sets the 12th index of the array to the value of h plus the 8th index of the array.

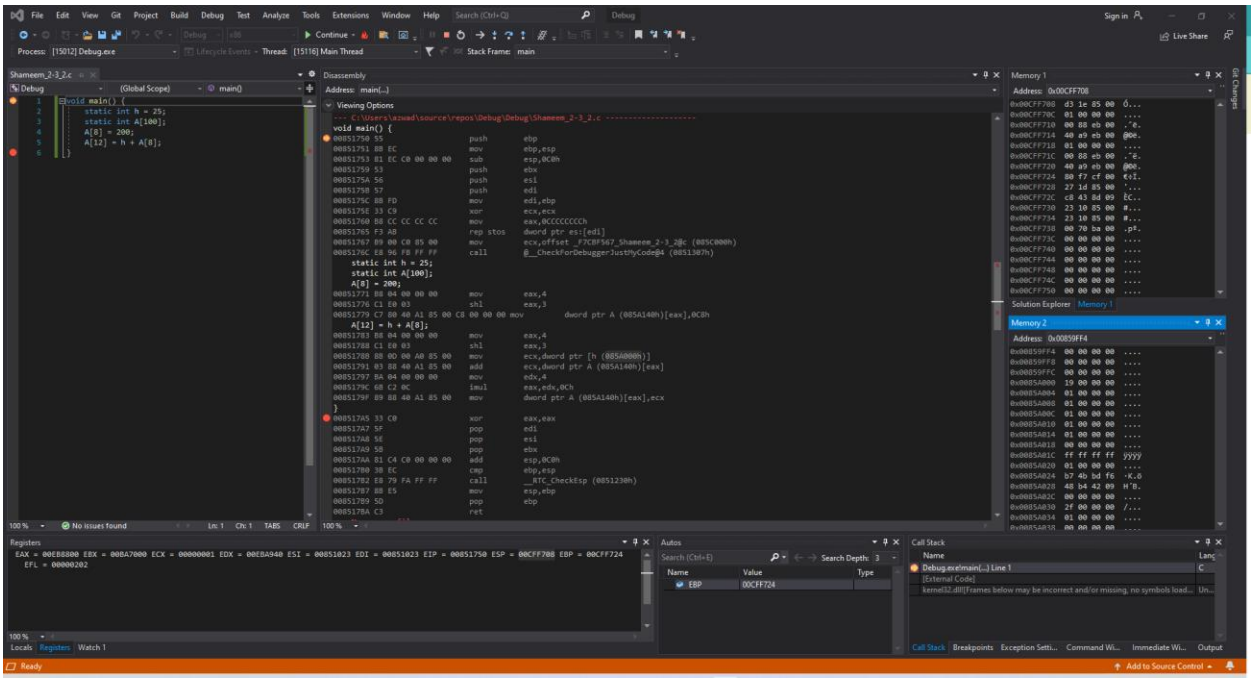


Figure 52: Shameem_2-3_1.c code ran in Visual Studio (Line 1)

Disassembly: This window shows that the code starts at line 1.

Registers: This window shows that the value of ESP, the stack pointer, is 0x00CFF708.

Memory: Memory window 1 shows the values around the stack pointer which is random. While Memory window 2 displays the value of the static variables stored into the addresses.

Variable h is stored at address 0x0085A00 and has the value 19 00 00 00.
Variable A is stored at address 0x0085140 and has the value 00 00 00 00.

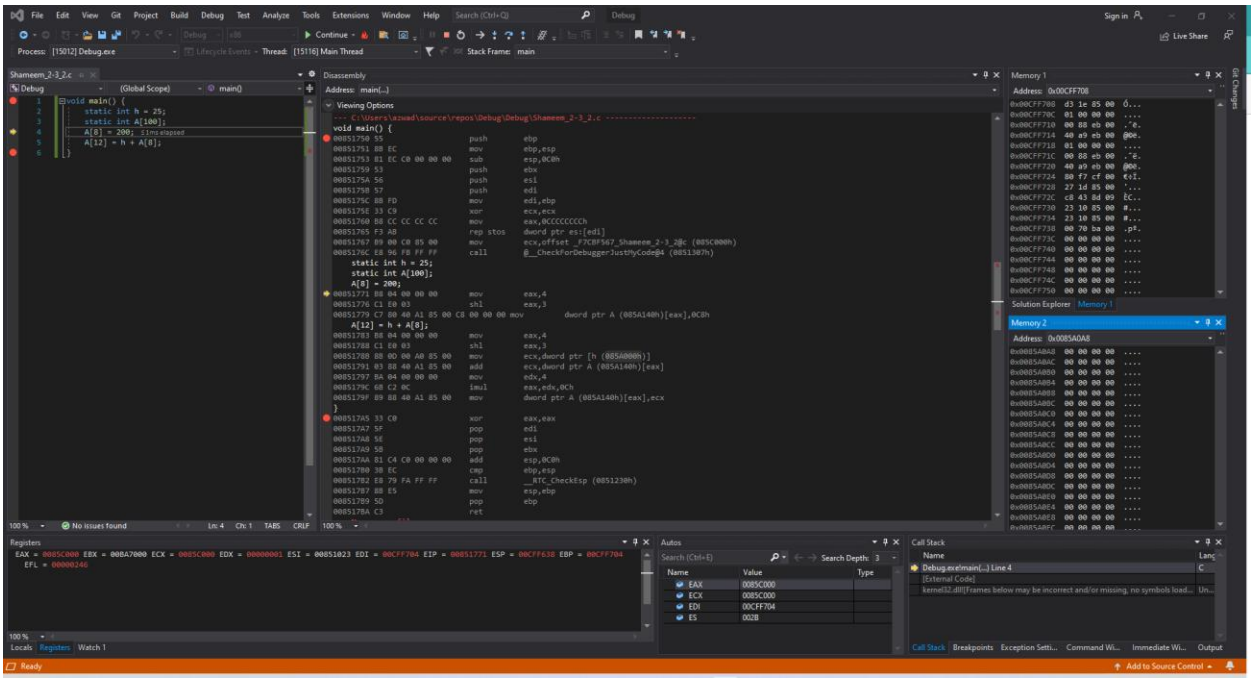


Figure 53: Shameem_2-3_1.c code ran in Visual Studio (Line 1-3)

Disassembly: This window shows that the code ran lines 1-3.

Registers: This window shows that the values that are stored in each register has been slightly changed from before to set up registers for the program to run.

Memory: Memory window 1 shows the values around the stack pointer which some are initialized. While Memory window 2 shows no changes in its display because lines 1-3 do not store anything in address or change values.

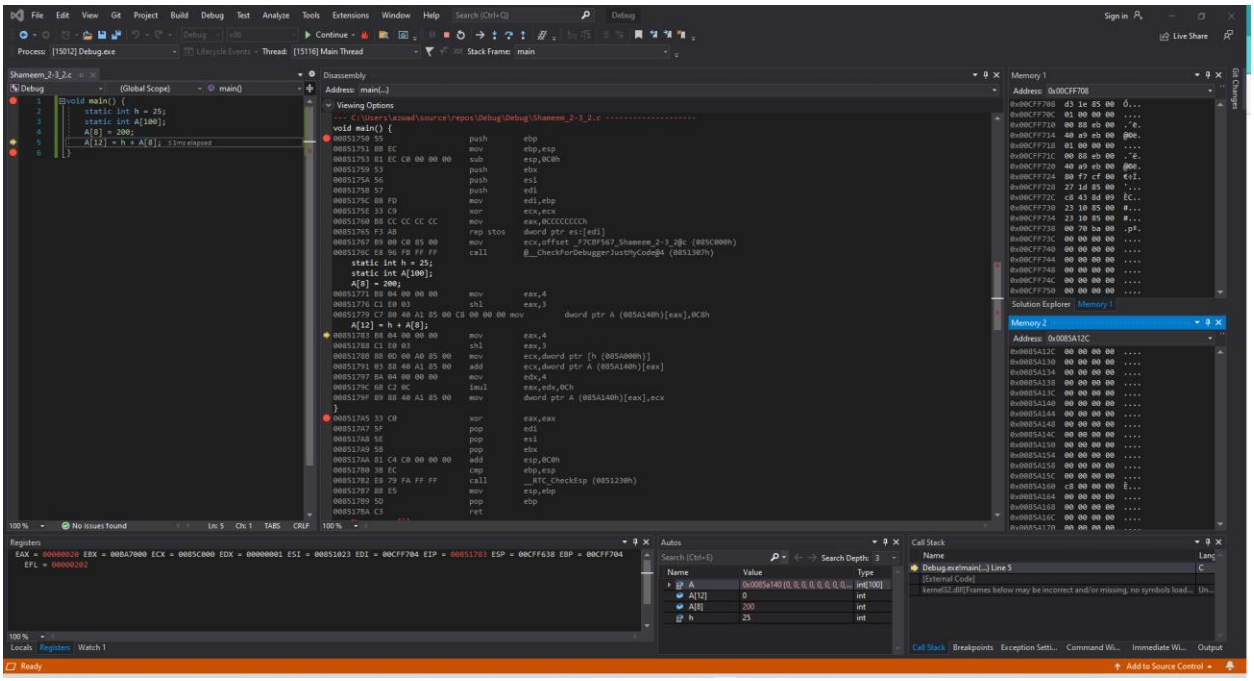


Figure 54: Shameem_2-3_1.c code ran in Visual Studio (Line 4)

Disassembly: This window shows that the code ran line 4.

Registers: This window shows that the values that are stored in each register has been slightly changed from before to set up registers for the program to run.

Memory: Memory window 1 shows the values around the stack pointer which some are initialized. While Memory window 2 shows the value c8 00 00 00 at the address 0x0085A160

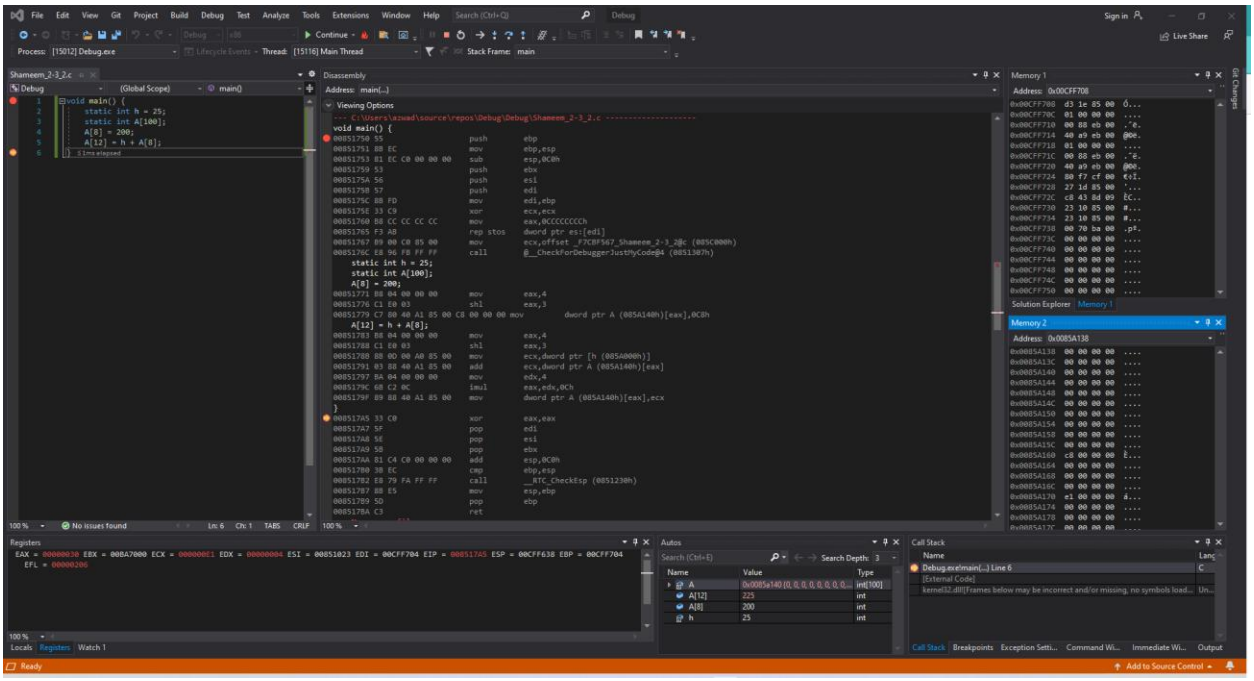


Figure 55: Shameem_2-2_2.c code ran in Debugger mode in Visual Studio (Line 6)

Disassembly: This window shows that the code ran lines 6. The value of h will be stored with the addition of the value of 8th index of the array A and at the 12th index of the array A.

Registers: This window shows that the value for ECX is set to the value of h which is then added by the 12th index of the array.

Memory: Memory window 1 shows the values around the stack pointer remains the same. While Memory window 2 displays the value stored in Array A at address 0x0085A170 with a value of e1 00 00 00.

2-5_1.c

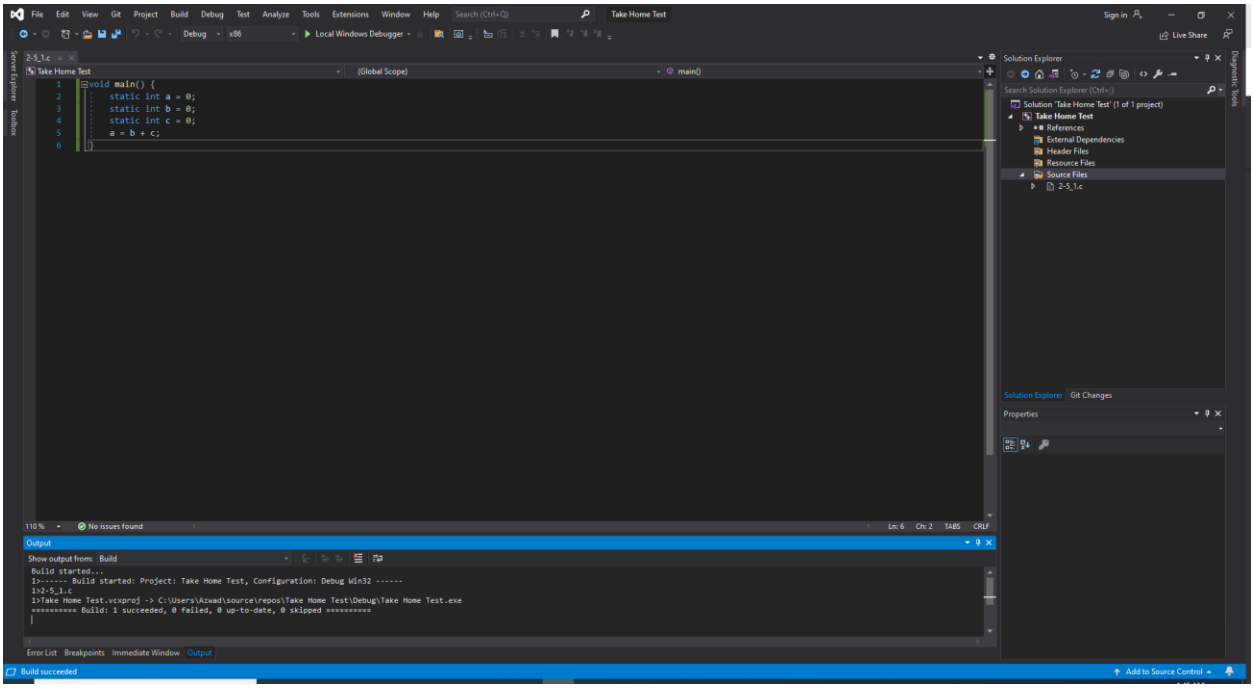


Figure 56: Shameem_2-5_1.c code in Visual Studio

The code has three static variables a, b, c and they all have the value of 0. Then the code adds b and c and stores that result in the variable a.

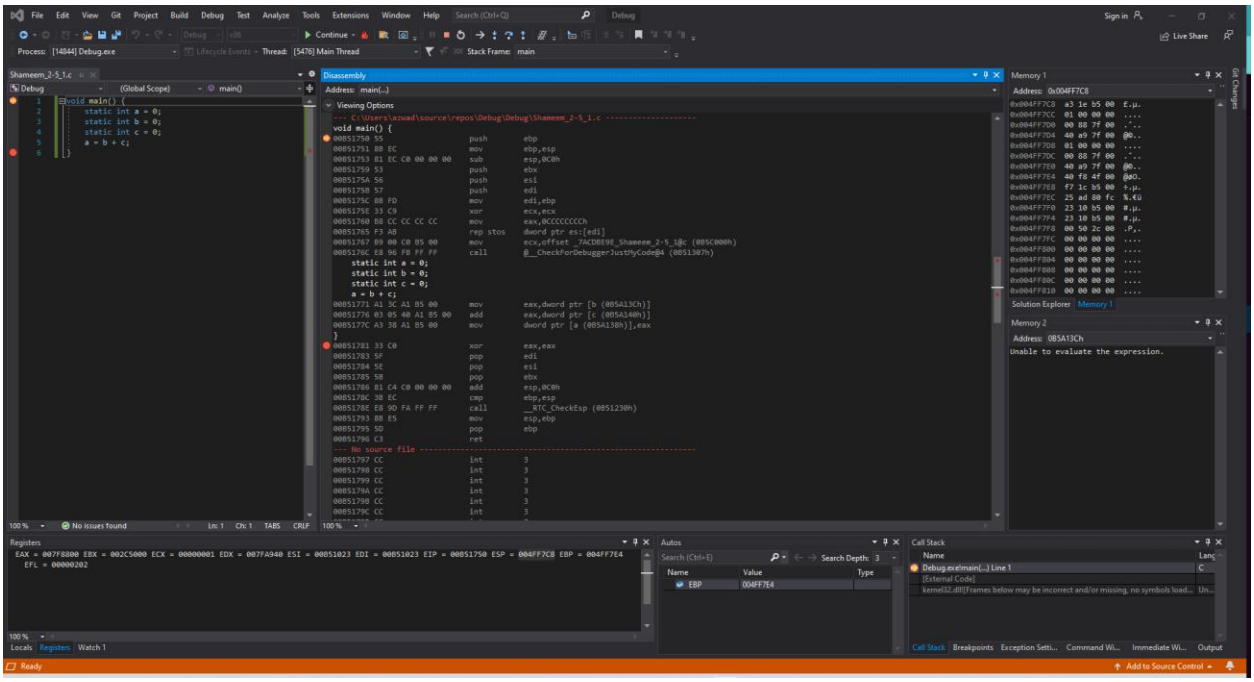


Figure 57: Shameem_2-5_1.c code ran in Visual Studio (Line 1)

Disassembly: This window shows that the code starts at line 1.

Registers: This window shows that the value of ESP, the stack pointer, is 0x00B51750.

Memory: Memory window 1 shows the values around the stack pointer which is random. While Memory window 2 is not able to display at this stage.

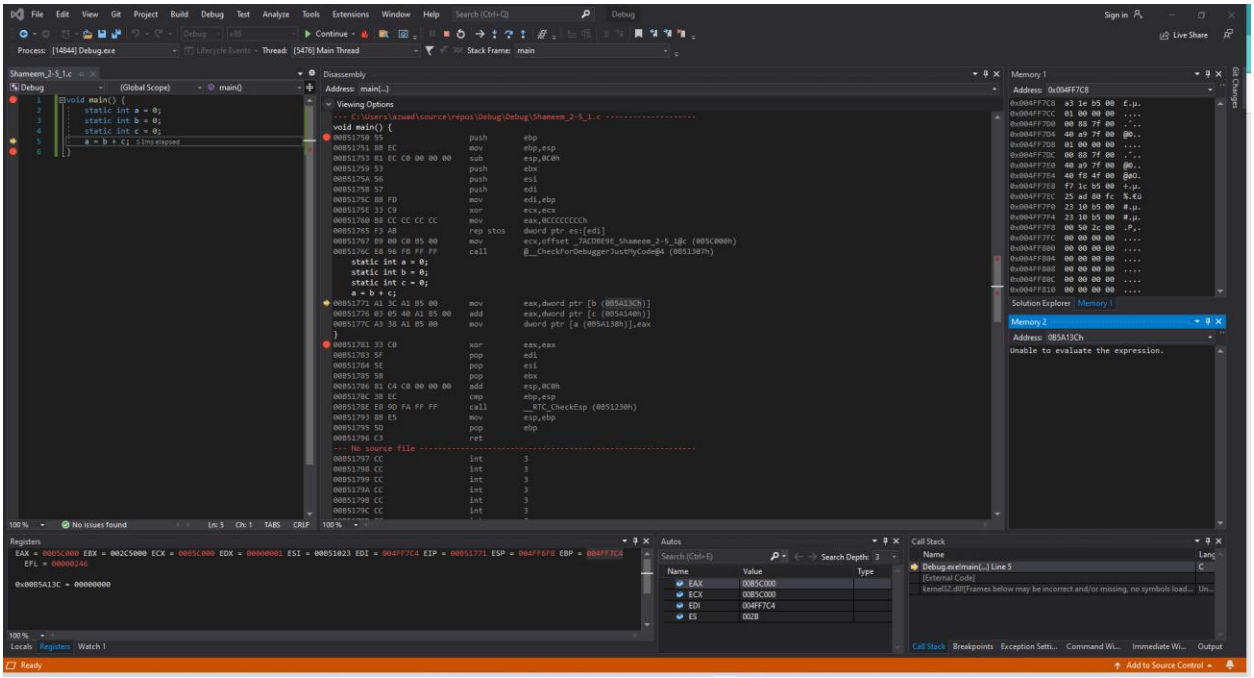


Figure 58: Shameem_2-5_1.c code ran in Visual Studio (Line 1-4)

Disassembly: This window shows that the code ran lines 1-4.

Registers: This window shows that the values that are stored in each register has been slightly changed from before to set up registers for the program to run.

Memory: Memory window 1 shows the values around the stack pointer which some are initialized. While Memory window 2 shows no changes in its display because lines 1-3 do not store anything in address or change values.

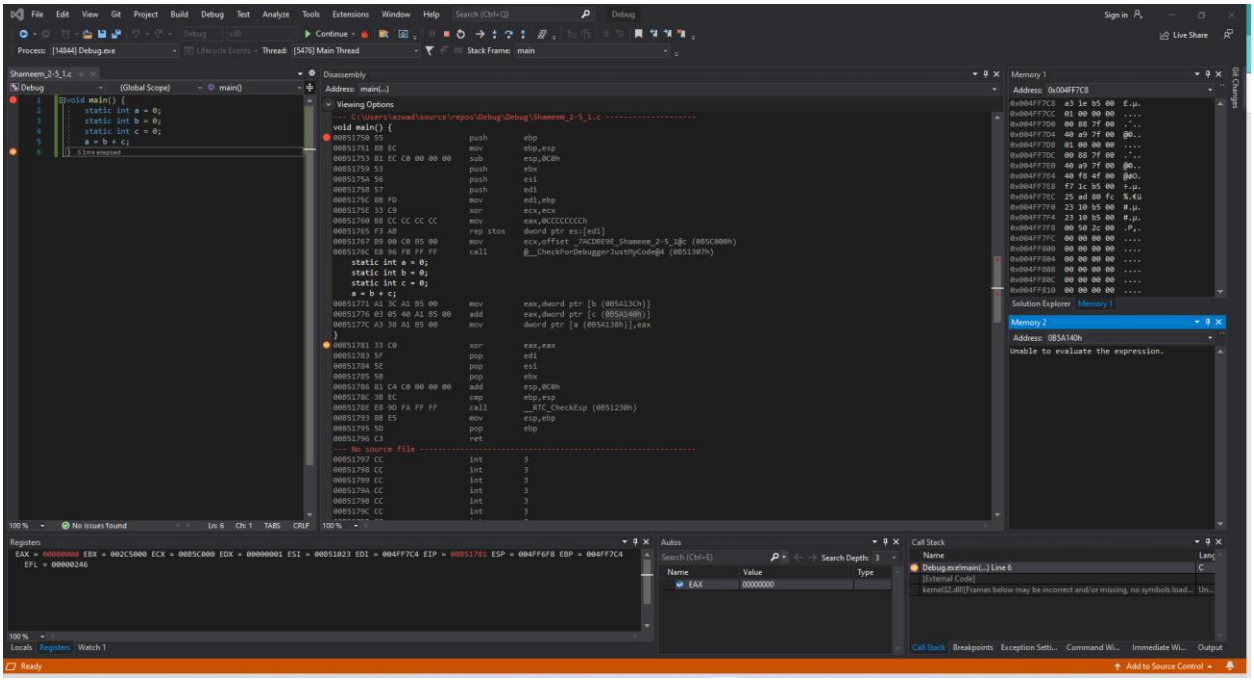


Figure 59: Shameem_2-5_1.c code ran in Debugger mode in Visual Studio (Line 5)

Disassembly: This window shows that the code ran lines 5. The value of a will be stored as the addition of the value of a and the value of c.

Registers: This window shows that the value for ECX is set 00000000 because the results total is 0.

Memory: Memory window 1 shows the values around the stack pointer remains the same. While Memory window 2 still cannot display and does not get highlighted because the value for everything is zero, which explains why nothing changes even after the whole program runs.

2-6_1.c

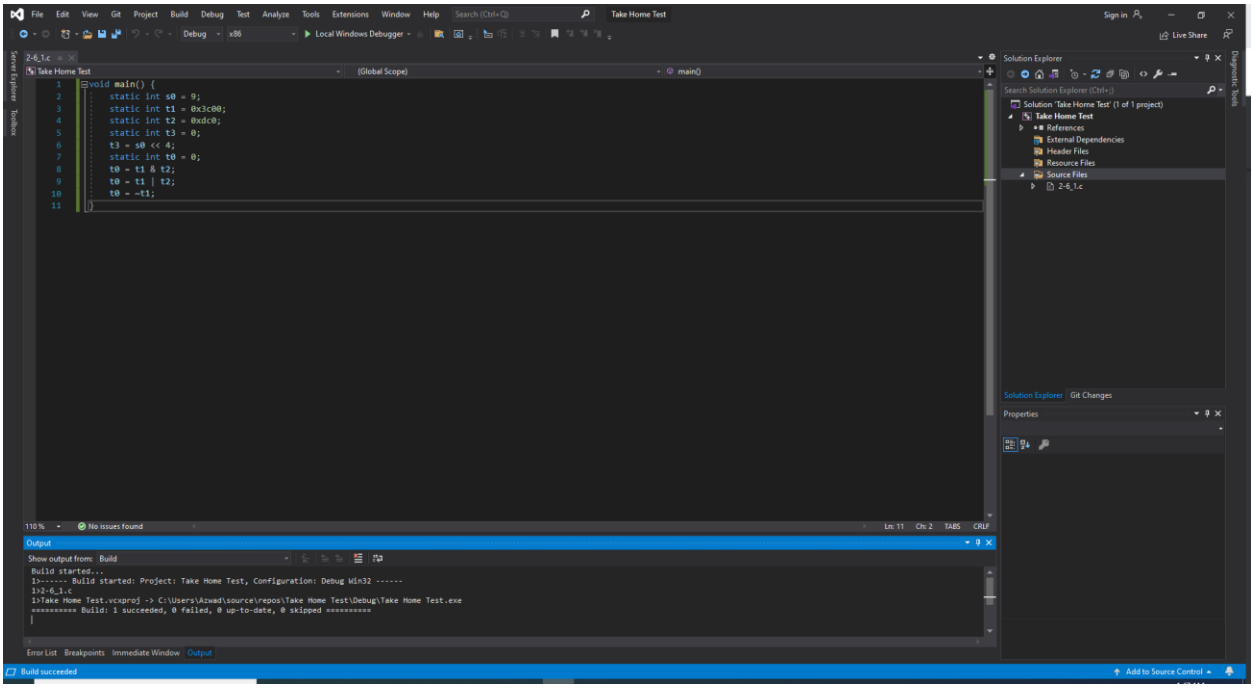


Figure 60: Shameem_2-6_1.c code in Visual Studio

The code has four static variables s0, t0, t1, t2, t3 with the values 9, 0, 0x3C00, 0xDC0, 0 respectively. The code then uses these variables to then bit shift s0 to the left by 4 and then save that value in variable t3. Then the code does AND with t1 and t2 and saves it in t0. The code then does OR with t1 and t2 and saves it in t2. Lastly, the code does NOT t1 and saves the result in the variable t0.

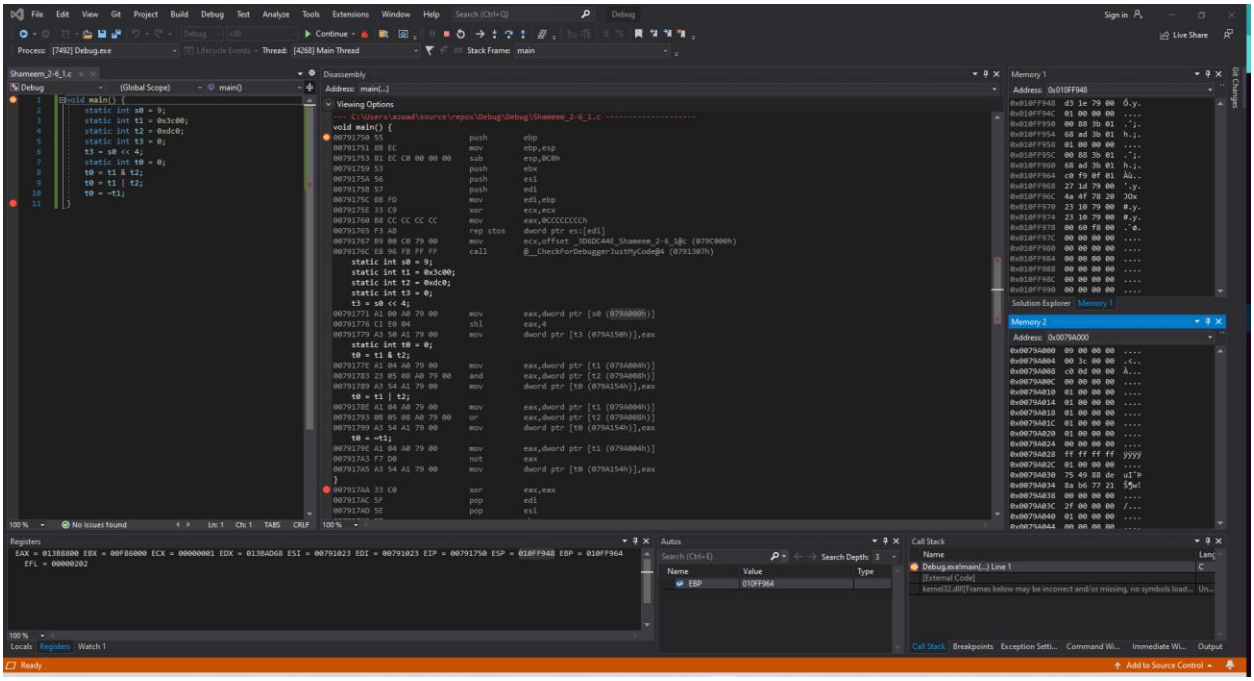


Figure 61: Shameem_2-6_1.c code ran in Visual Studio (Line 1)

Disassembly: This window shows that the code starts at line 1.

Registers: This window shows that the value of ESP, the stack pointer, is 0x010FF948.

Memory: Memory window 1 shows the values around the stack pointer which is random. While Memory window 2 displays the value of the static variables stored into the addresses.

Variable s0 is stored at address 0x0079A000 and has the value 09 00 00 00.
Variable t1 is stored at address 0x0079A004 and has the value 00 3c 00 00.
Variable t2 is stored at address 0x0079A008 and has the value c0 0d 00 00.
Variable t3 is stored at address 0x0079A150 and has the value 00 00 00 00.
Variable t0 is stored at address 0x0079A154 and has the value 00 00 00 00.

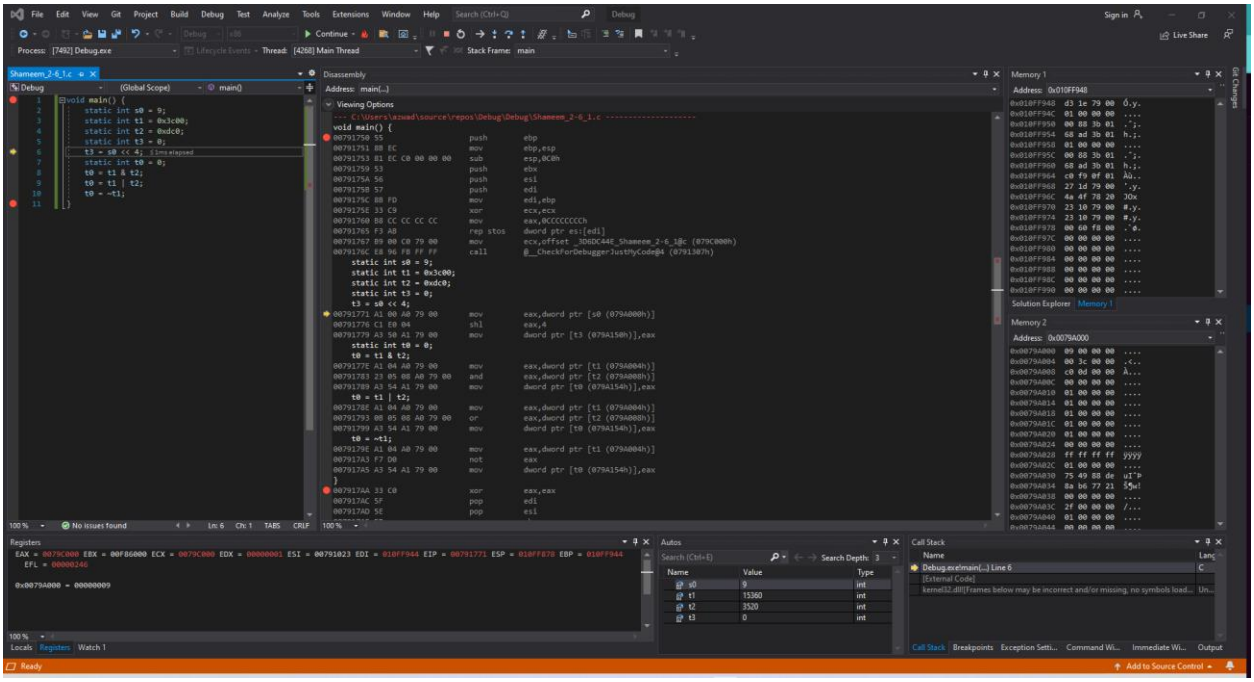


Figure 62: Shameem_2-6_1.c code ran in Visual Studio (Line 1-4)

Disassembly: This window shows that the code ran lines 1-4.

Registers: This window shows that the values that are stored in each register has been slightly changed from before to set up registers for the program to run.

Memory: Memory window 1 shows the values around the stack pointer which some are initialized. While Memory window 2 shows no changes in its display because lines 1-4 do not store anything in address or change values.

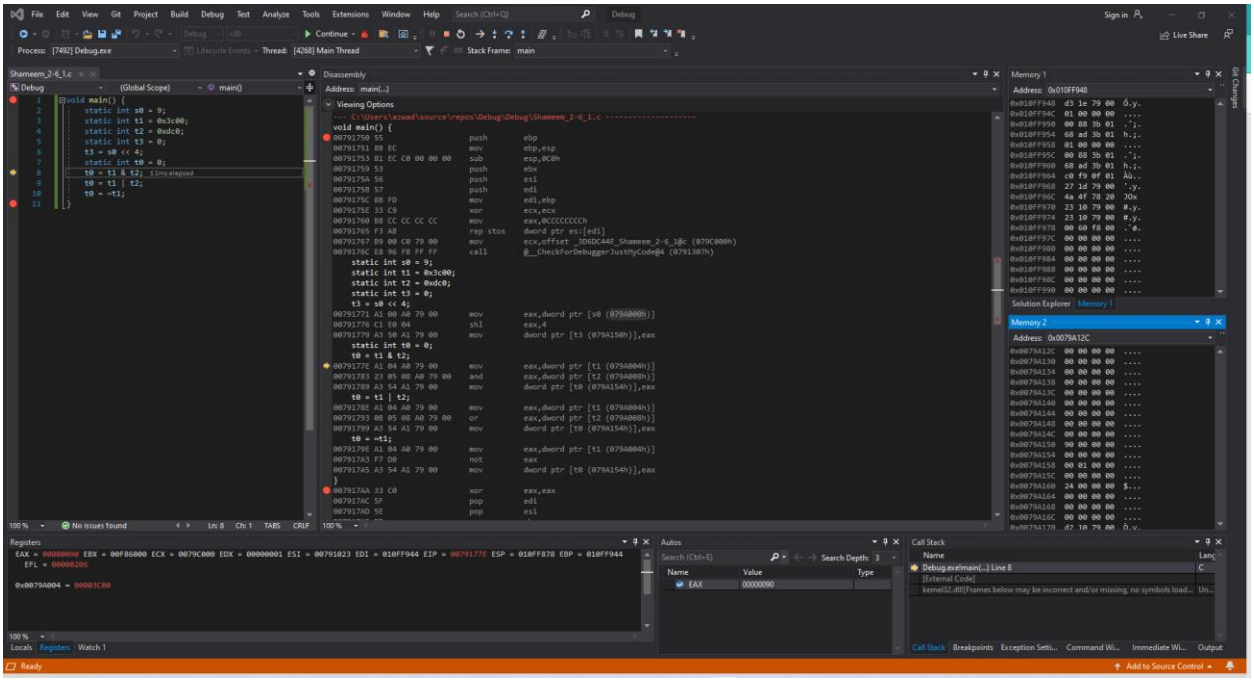


Figure 63: Shameem_2-6_1.c code ran in Visual Studio (Line 6)

Disassembly: This window shows that the code ran line 6.

Registers: This window shows that the values that are stored in each register has been slightly changed from before to set up registers for the program to run.

Memory: Memory window 1 shows the values around the stack pointer which some are initialized. While Memory window 2 shows the value 90 00 00 00 at the address 0x0079A150.

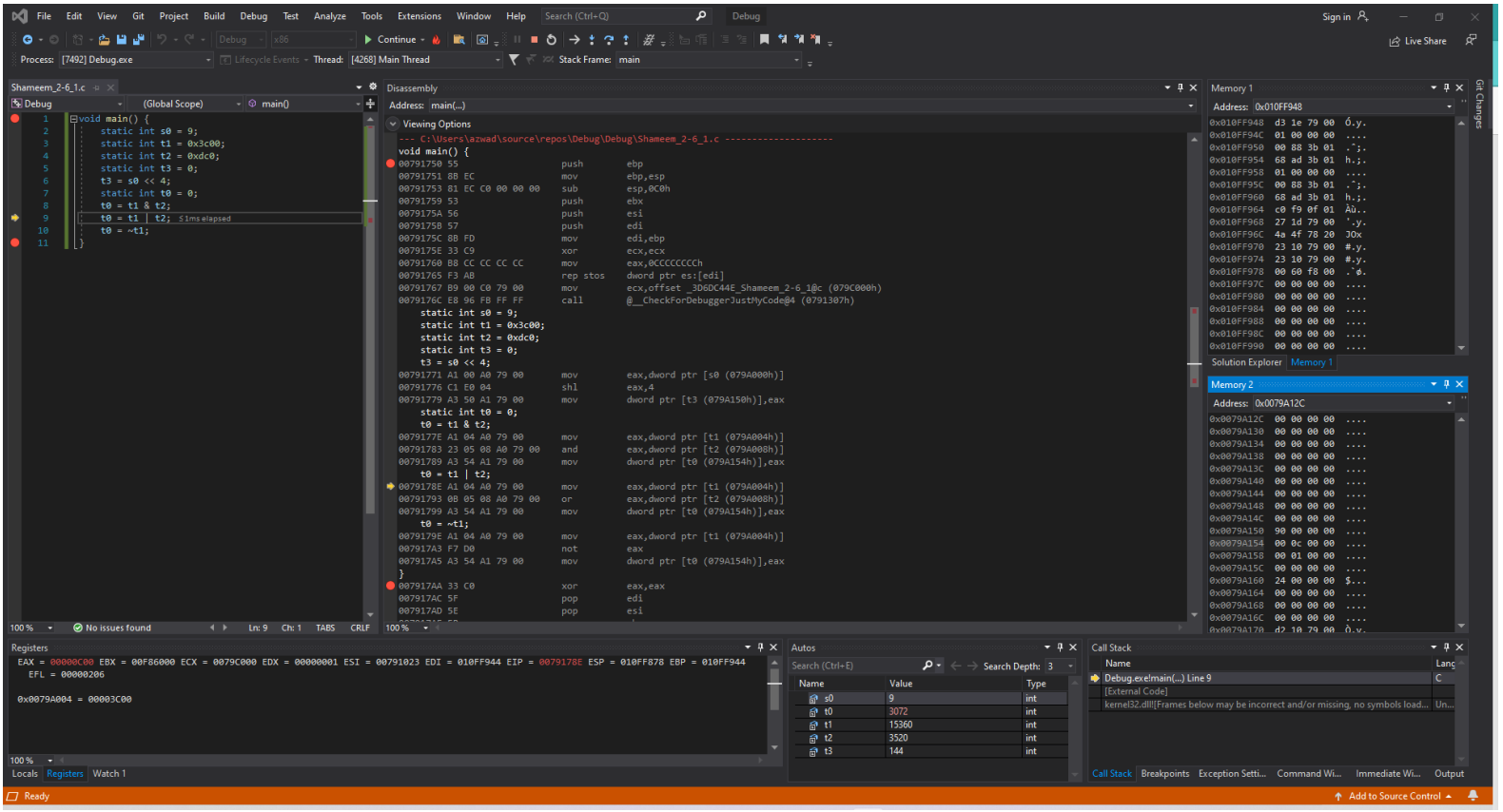


Figure 4: Shameem_2-6_1.c code ran in Visual Studio (Line 8)

Disassembly: This window shows that the code ran line 8.

Registers: This window shows that the values of t1 stored in memory is moved to the register and then that value is used to compute bitwise operation. Also, the value in that register is moved into the memory which overwrites the previous value stored at $t0.

Memory: Memory window 1 shows the values around the stack pointer which some are initialized. While Memory window 2 shows the value 00 0c 00 00 at the address 0x0079A154.

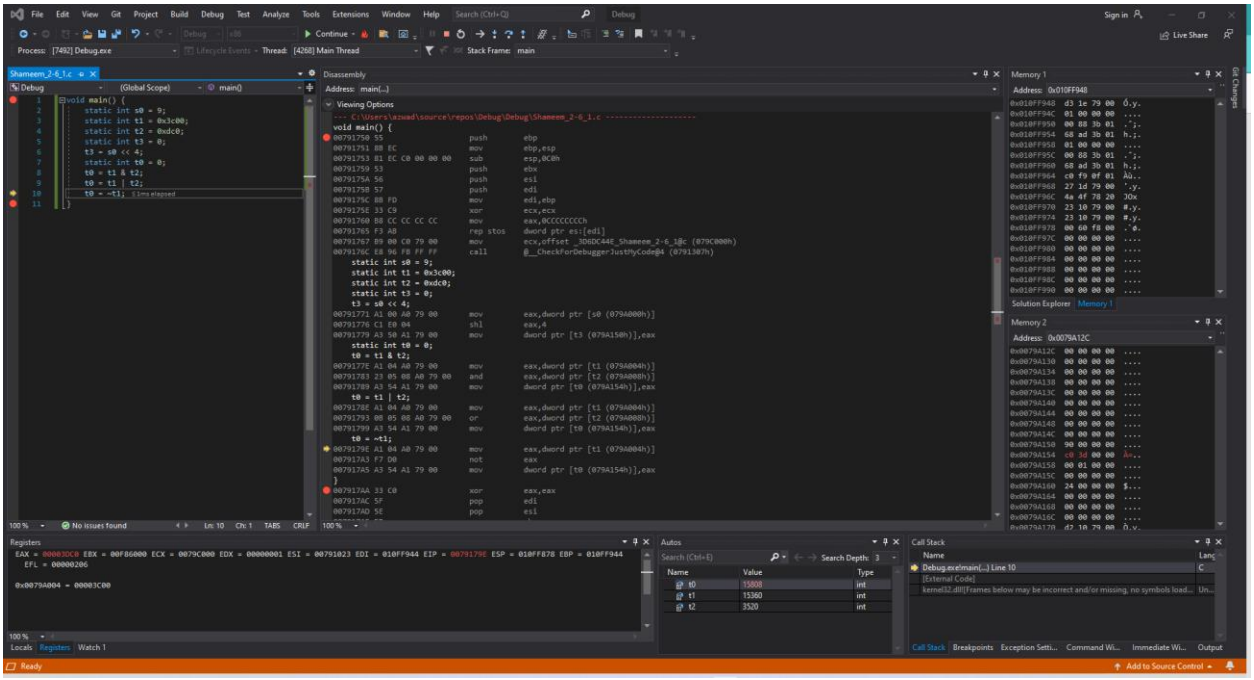


Figure 65: Shameem_2-6_1.c code ran in Visual Studio (Line 9)

Disassembly: This window shows that the code ran line 9.

Registers: This window shows that the value of t1 is moved to the register and then the value of that is compared with t2 for the bitwise operation which is stored in the register. Then that value in the register is moved into the memory and overwrites the previous value stored at t0.

Memory: Memory window 1 shows the values around the stack pointer which some are initialized. While Memory window 2 shows the value c0 3d 00 00 at the address 0x0079A154.

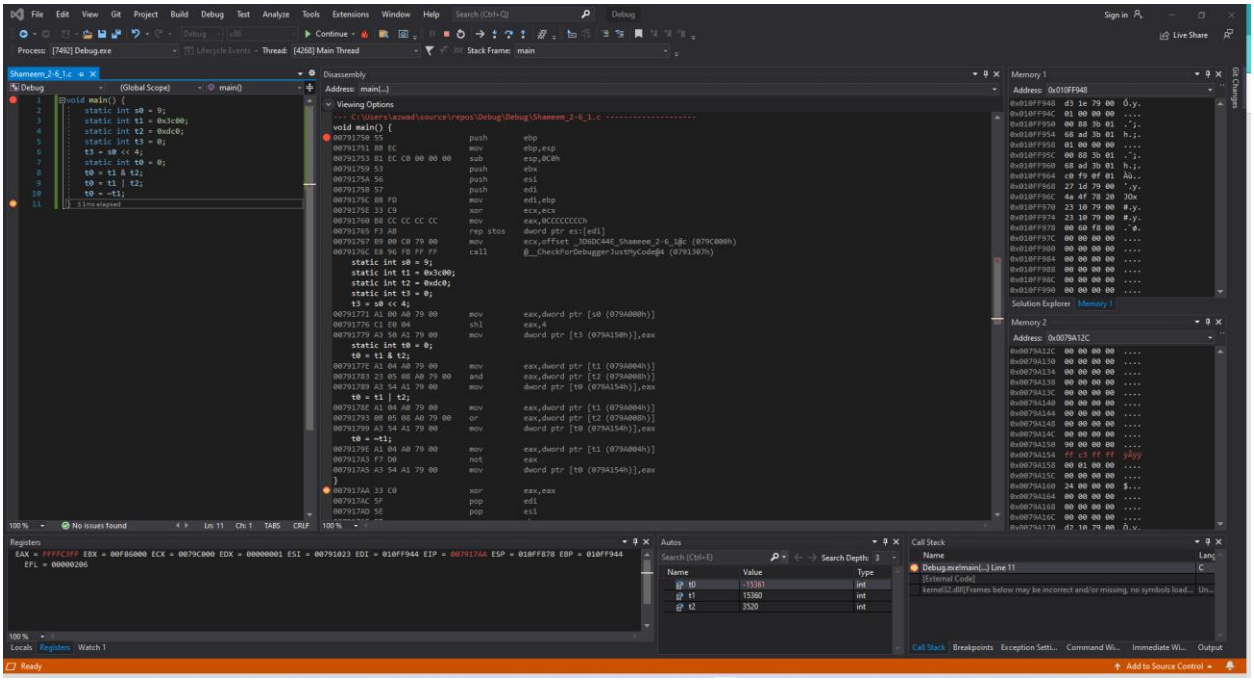


Figure 66: Shameem_2-6_1.c code ran in Visual Studio (Line 10)

Disassembly: This window shows that the code ran line 10.

Registers: This window shows that the value of t1 is moved to the register and then the value of that is compared with t2 for the bitwise operation which is stored in the register. Then that value in the register is moved into the memory and overwrites the previous value stored at t0.

Memory: Memory window 1 shows the values around the stack pointer which some are initialized. While Memory window 2 shows the value ff c3 ff ff at the address 0x0079A154.

2-7_1.c

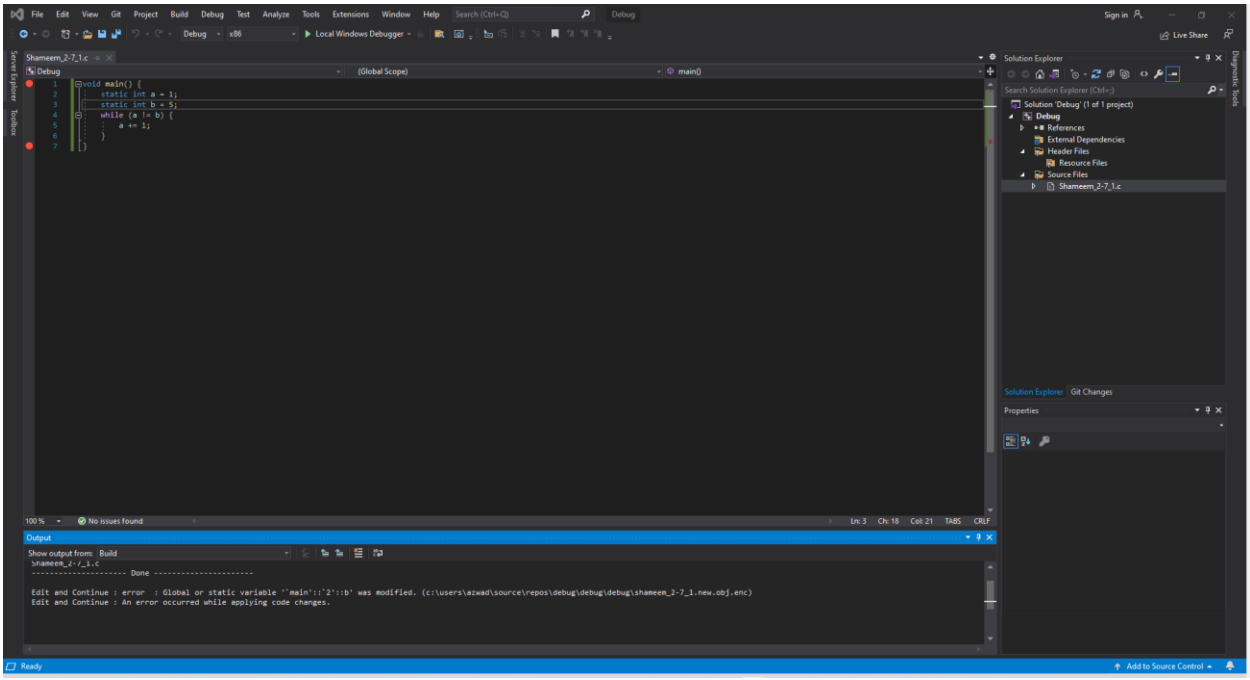


Figure 67: Shameem_2-7_1.c code in Visual Studio

The code has three static variables a and b with the respective values 1 and 5. The code then goes into a while loop where for each loop the variable a is incremented by 1 and continues that process until a equals b. Once variable a equals variable b the function then stops.

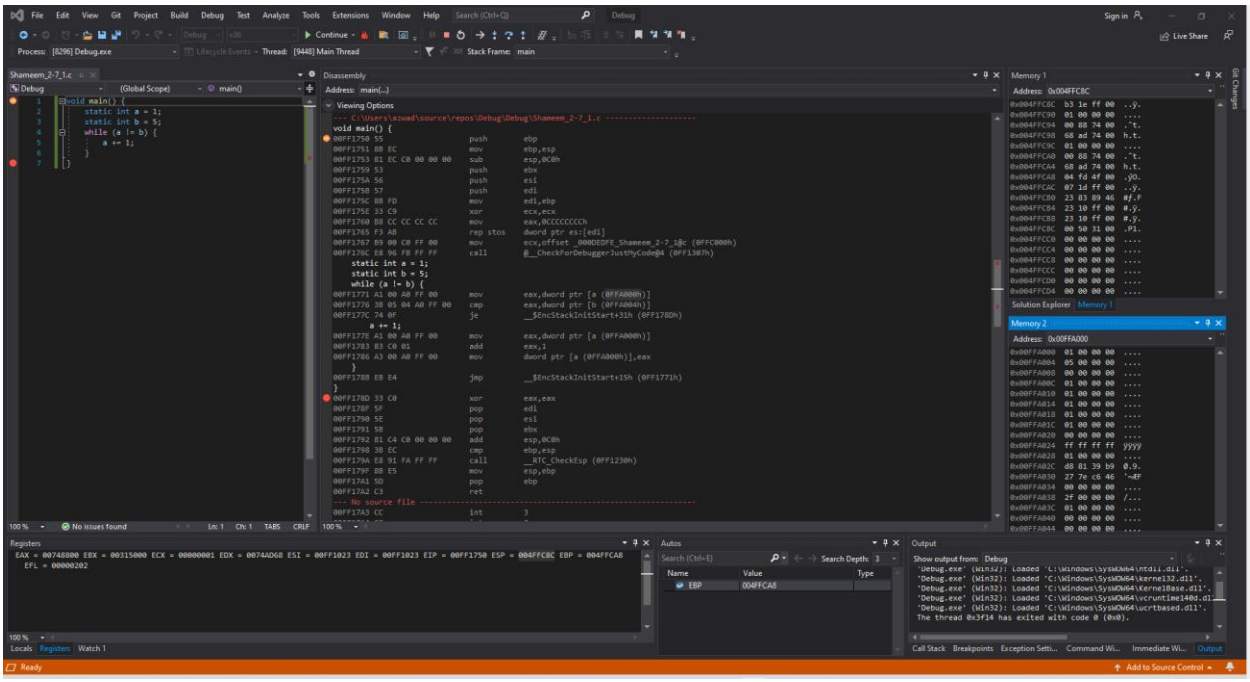


Figure 68: Shameem_2-7_1.c code ran in Visual Studio (before Line 1)

Disassembly: This window shows that the code starts at line 1.

Registers: This window shows that the value of ESP, the stack pointer, is 0x004FFC8C.

Memory: Memory window 1 shows the values around the stack pointer which is random. While Memory window 2 displays the value of the static variables stored into the addresses.

Variable a is stored at address 0x00FFA000 and has the value 01 00 00 00.
Variable b is stored at address 0x00FFA004 and has the value 05 00 00 00.

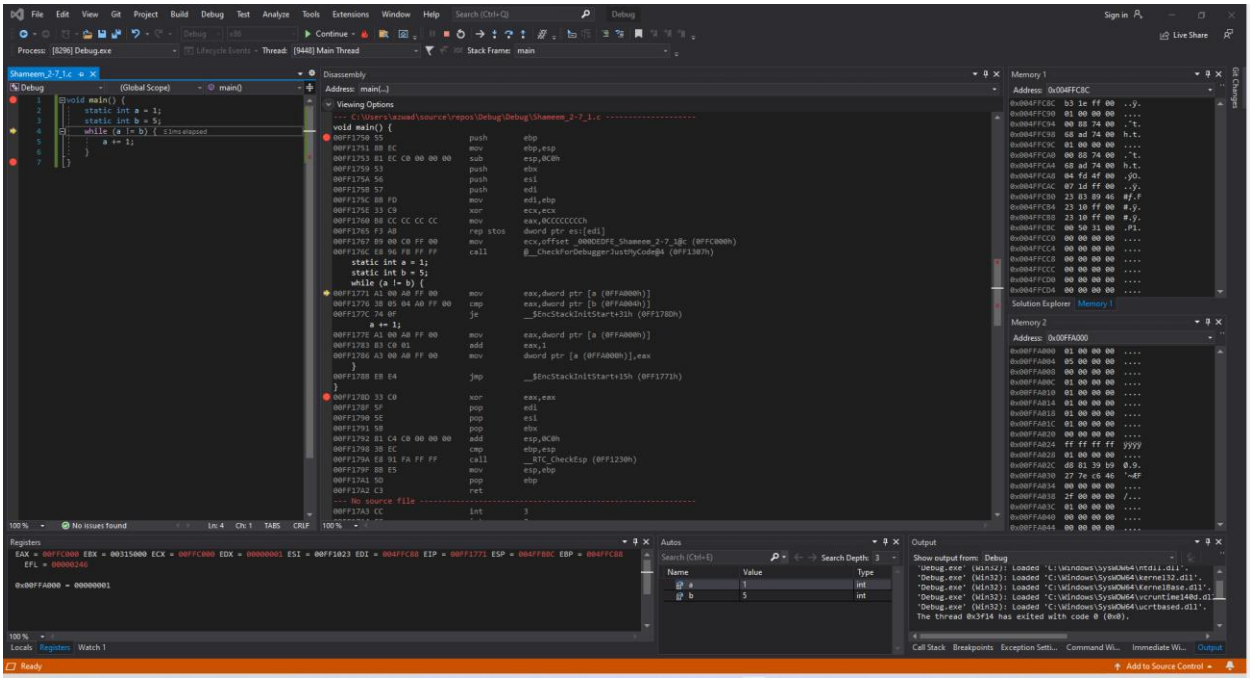


Figure 69: Shameem_2-7_1.c code ran in Visual Studio (Lines 1-3)

Disassembly: This window shows that the code runs lines 1-3.

Registers: This window shows that the value of a is loaded into the register at EDX.

Memory: Memory window 1 shows the values around the stack pointer which is somewhat initialized. While Memory window 2 displays the value of the static variables stored into the addresses.

Variable a is stored at address 0x00FFA000 and has the value 01 00 00 00.
Variable b is stored at address 0x00FFA004 and has the value 05 00 00 00.

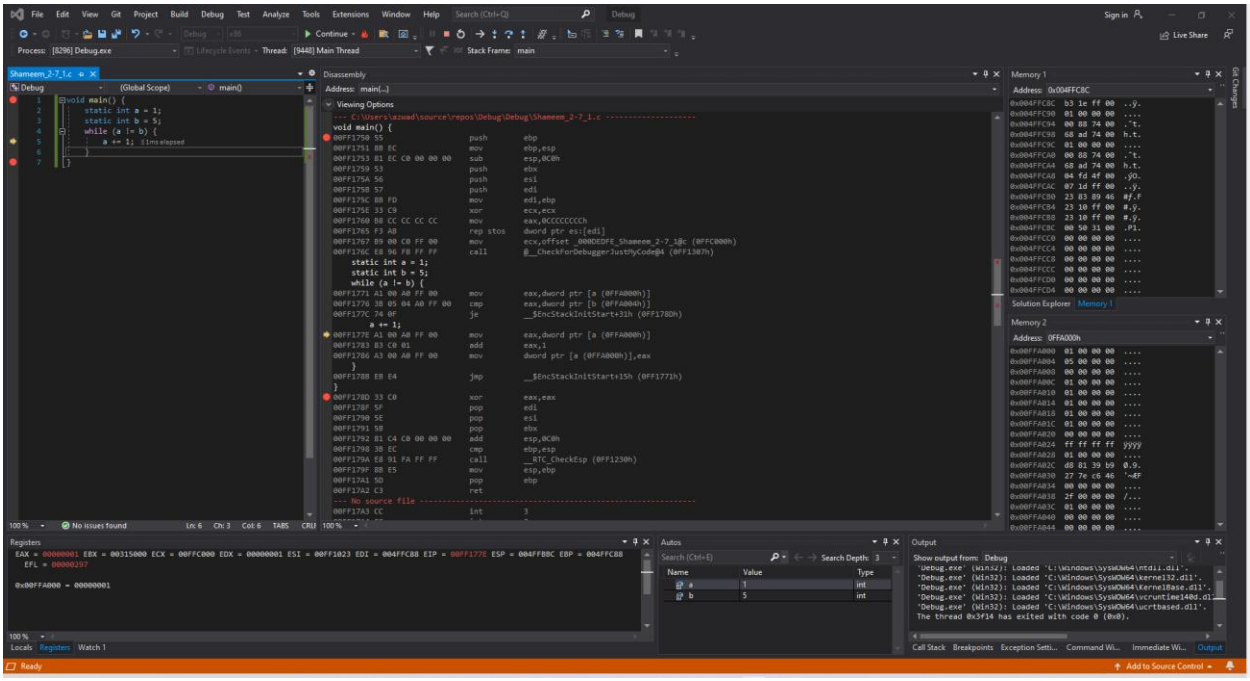


Figure 70: Shameem_2-7_1.c code ran in Visual Studio (Lines 1-3)

Disassembly: This window shows that the code runs lines 4.

Registers: This window shows that the value of a is moved into the EAX as well so that it can increment the value of variable a.

Memory: Memory window 1 shows the values around the stack pointer which is somewhat initialized. While Memory window 2 displays the value of the static variables stored into the addresses.

Variable a is stored at address 0x00FFA000 and has the value 01 00 00 00.
Variable b is stored at address 0x00FFA004 and has the value 05 00 00 00.

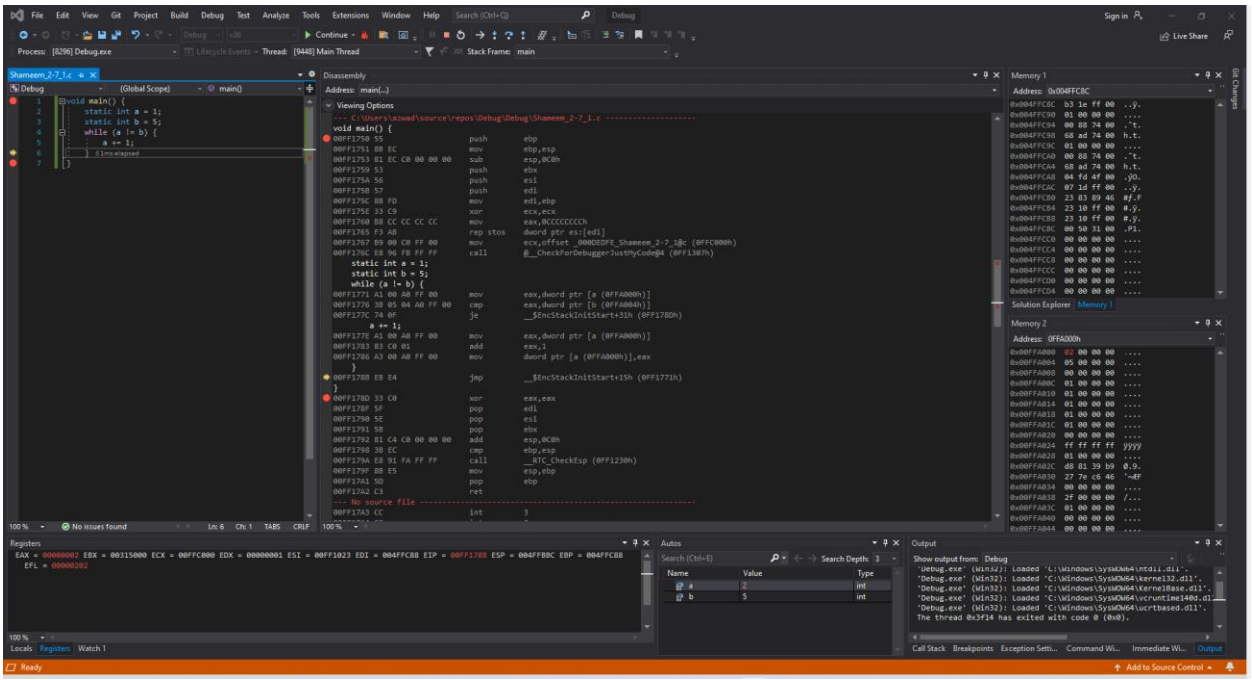


Figure 71: Shameem_2-7_1.c code ran in Visual Studio (Line 5)

Disassembly: This window shows that the code runs line 5.

Registers: This window shows that the EAX has become the value 00000002 which means that a has been incremented.

Memory: Memory window 1 shows the values around the stack pointer which is somewhat initialized. While Memory window 2 displays the value of the static variables stored into the addresses.
Variable a is highlighted and stored at address 0x00FFA000 and has the value 02 00 00 00.
Variable b is stored at address 0x00FFA004 and has the value 05 00 00 00.

This process of incrementation will continue until a is equal to b and when it does the program will exit.

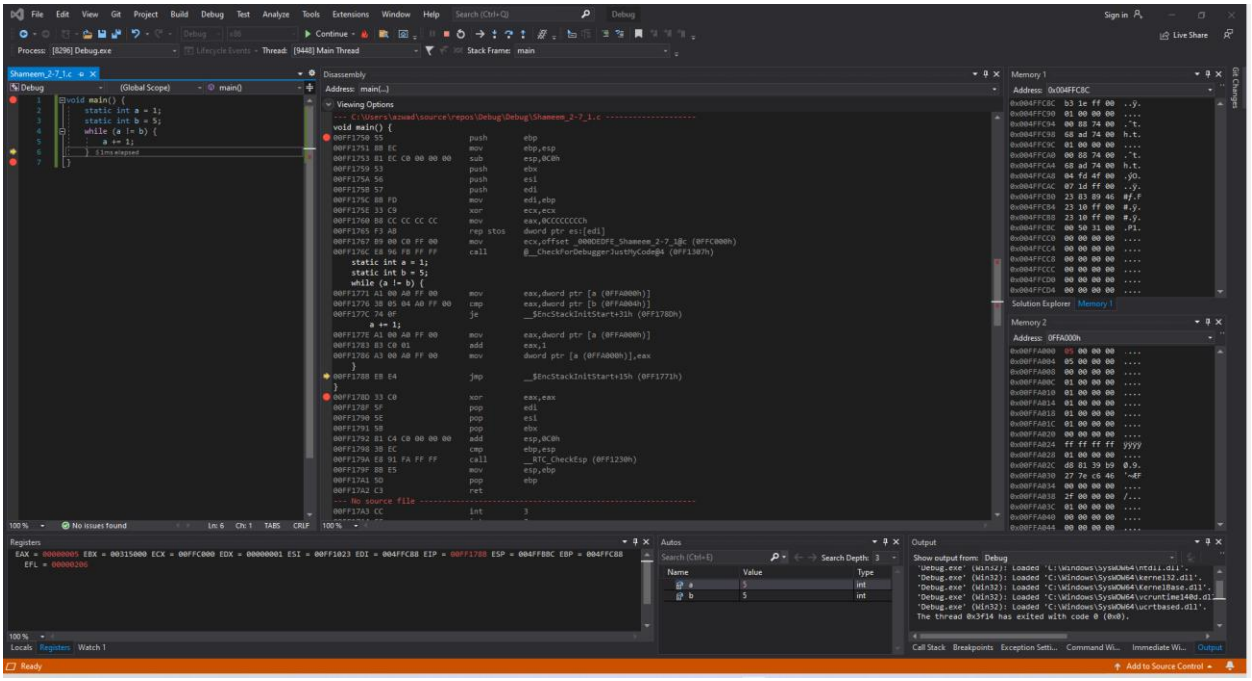


Figure 72: Shameem_2-7_1.c code ran in Visual Studio (Lines 4-5)

Disassembly: This window shows that the code runs lines 4-5.

Registers: This window shows that the EAX has become the value 00000005 which means that a has been incremented four times.

Memory: Memory window 1 shows the values around the stack pointer which is somewhat initialized. While Memory window 2 displays the value of the static variables stored into the addresses.
Variable a is highlighted and stored at address 0x00FFA000 and has the value 05 00 00 00.
Variable b is stored at address 0x00FFA004 and has the value 05 00 00 00.

Now that variable a and variable b are equal the program will not loop any longer and will exit.

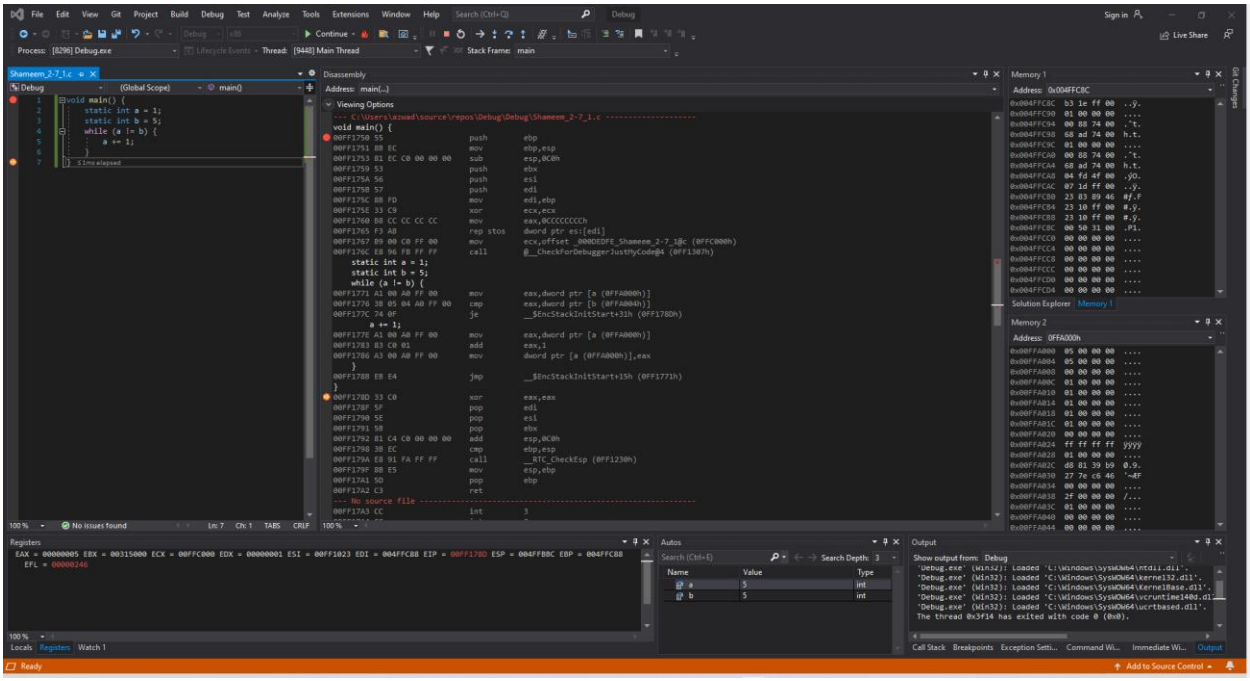


Figure 73: Shameem_2-7_1.c code ran in Visual Studio (Lines 6-7)

Disassembly: This window shows that the code runs lines 6-7.

Registers: This window shows that EIP and EFL have changed but nothing else because there is no change in values.

Memory: Memory window 1 shows the values around the stack pointer which is somewhat initialized. While Memory window 2 displays the value of the static variables stored into the addresses.

Variable a is stored at address 0x00FFA000 and has the value 05 00 00 00.
Variable b is stored at address 0x00FFA004 and has the value 05 00 00 00.

natural_generator.c

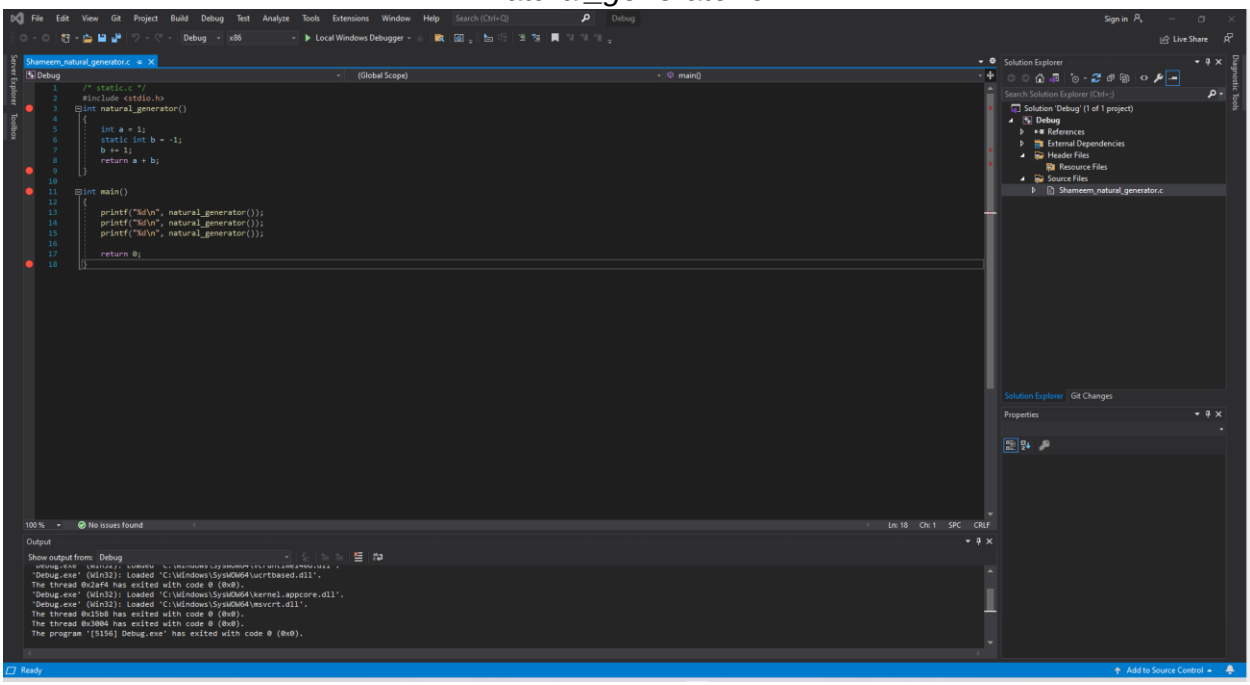


Figure 74: Shameem_natural_generator.c code in Visual Studio

The function natural_generator() has an variable integer a and a static integer b with values 1 and -1 respectively. The function natural_generator() increments the static variable b every time its called and then returns the value of the variable a plus the value of the static variable b.

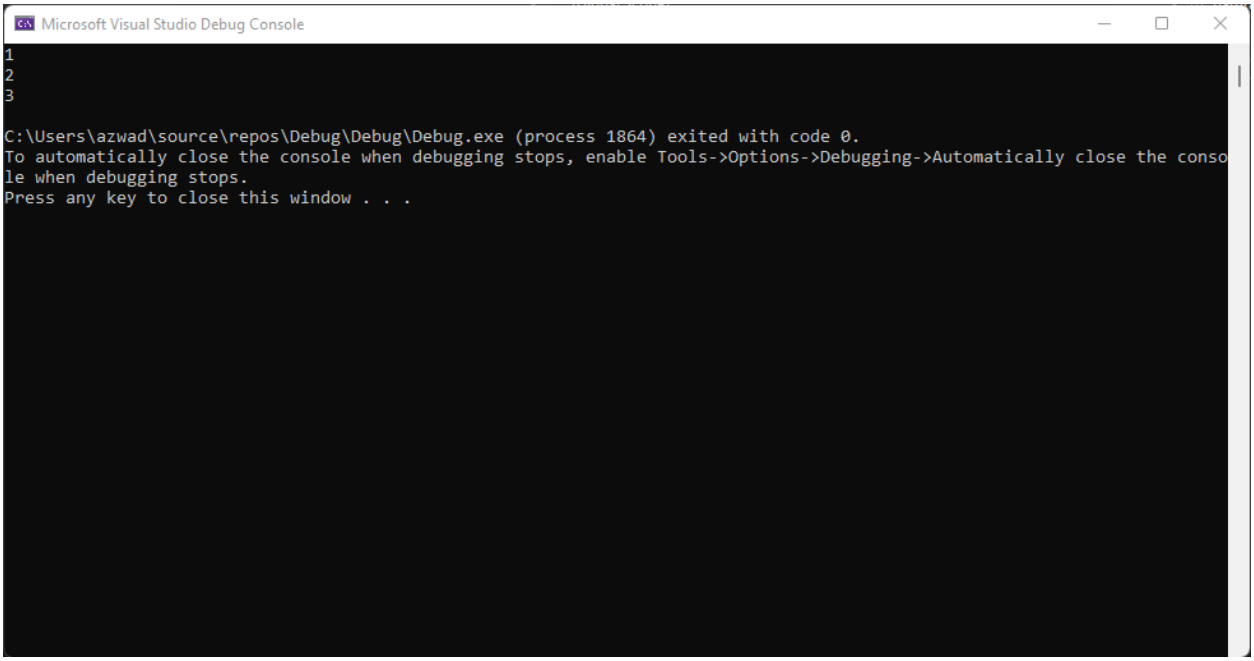


Figure 75: Shameem_natural_generator.c Figure of the command prompt output which shows the output 1, 2 and 3 which is the output of the print statement in the C code.

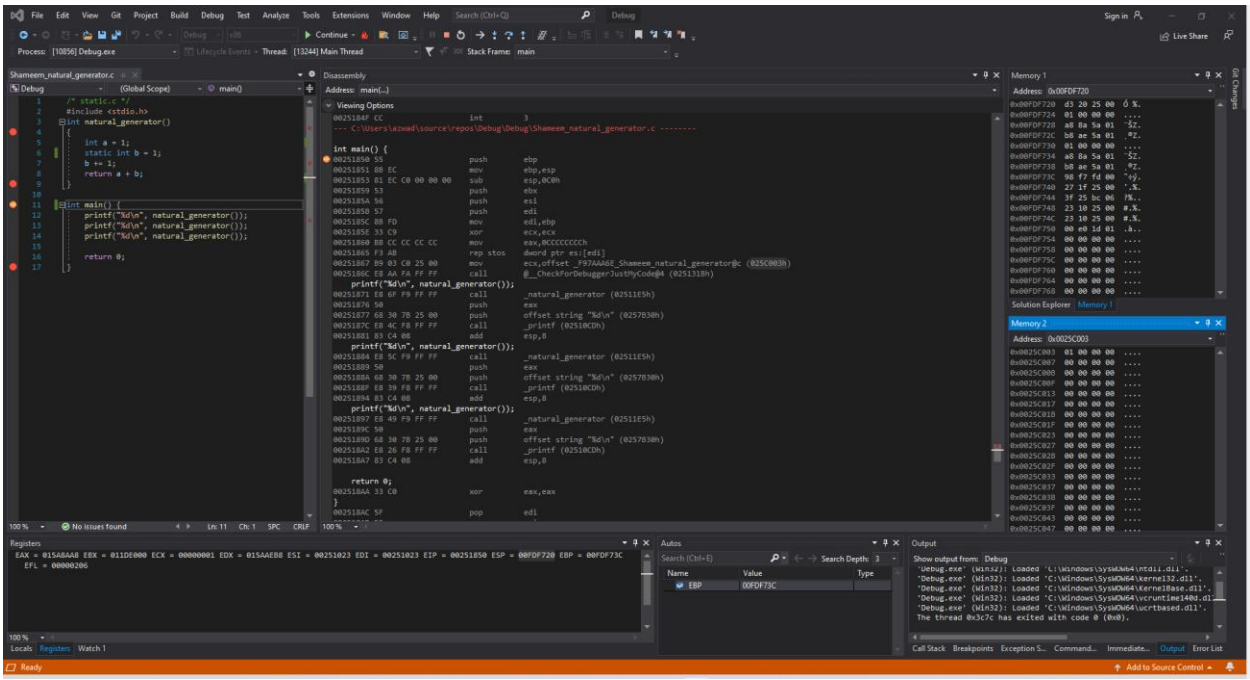


Figure 76: Shameem_natural_generator.c code ran in Visual Studio (before Line 1)

Disassembly: This window shows that the code starts at line 1.

Registers: This window shows that the value of ESP, the stack pointer, is 0x0021F930.

Memory: Memory window 1 shows the values around the stack pointer which is random. While Memory window 2 displays the value of the static variables stored into the addresses.

Variable b is stored at address 0x0025C003 and has the value 01 00 00 00.

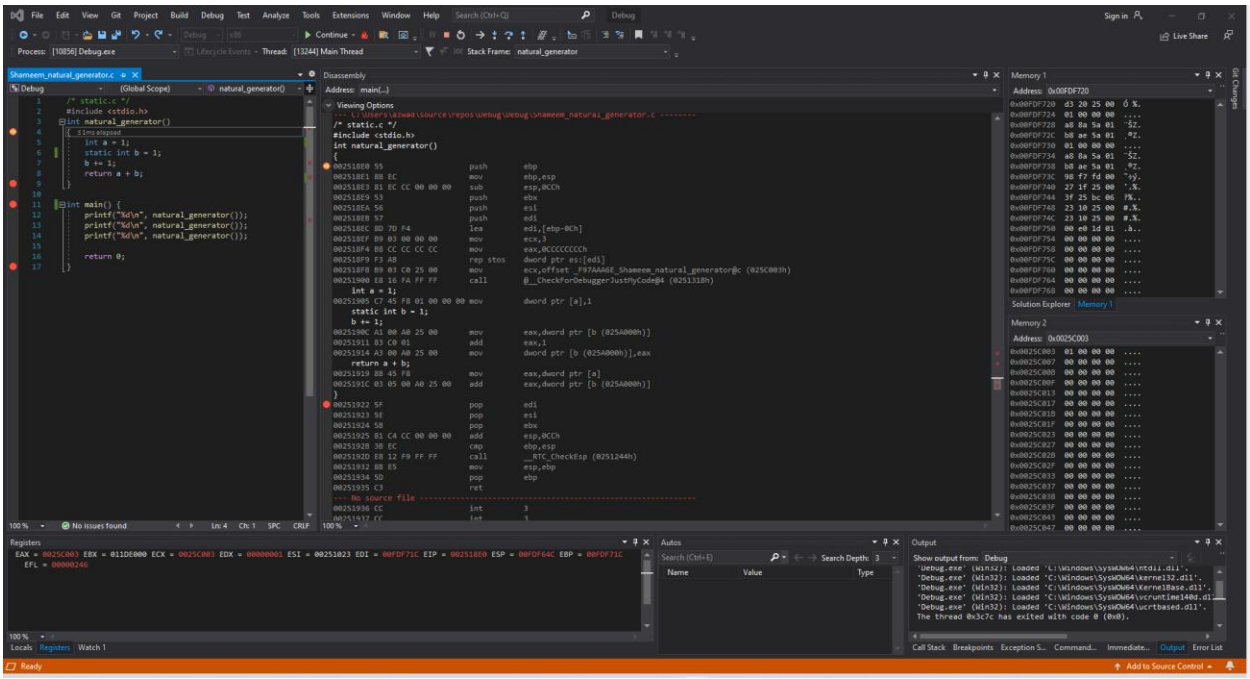


Figure 77: Shameem_natural_generator.c code ran in Visual Studio (11-12, 4)

Disassembly: This window shows that the code calls the natural_generator() function.

Registers: This window shows that the register values change because we are now in the natural generator function.

Memory: Memory window 1 shows the values around the stack pointer. While Memory window 2 displays the value of the static variables stored into the addresses.

Variable b is stored at address 0x0025C003 and has the value 01 00 00 00.

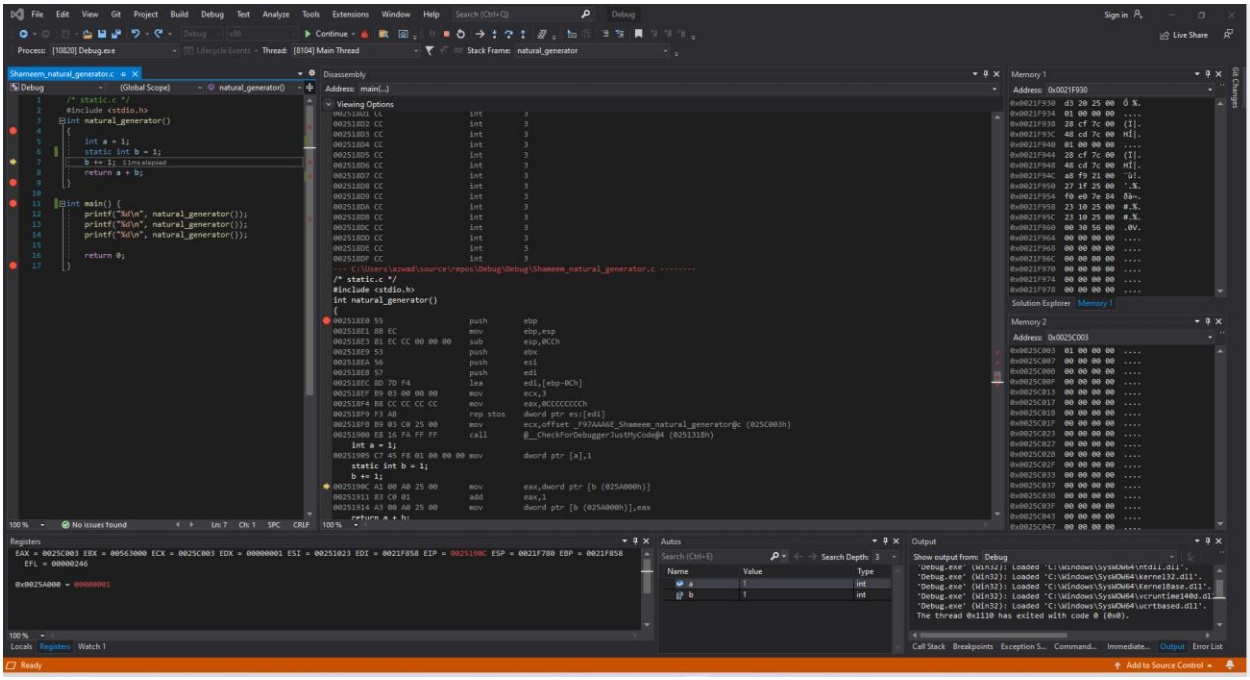


Figure 78: Shameem_natural_generator.c code ran in Visual Studio (5-6)

Disassembly: This window shows that the code runs lines 5-6 which has integer a and static int b.

Registers: This window shows that the value of b is pulled into the register so that b can be incremented.

Memory: Memory window 1 shows the values around the stack pointer. While Memory window 2 displays the value of the static variables stored into the addresses.

Variable b is stored at address 0x0025C003 and has the value 01 00 00 00.

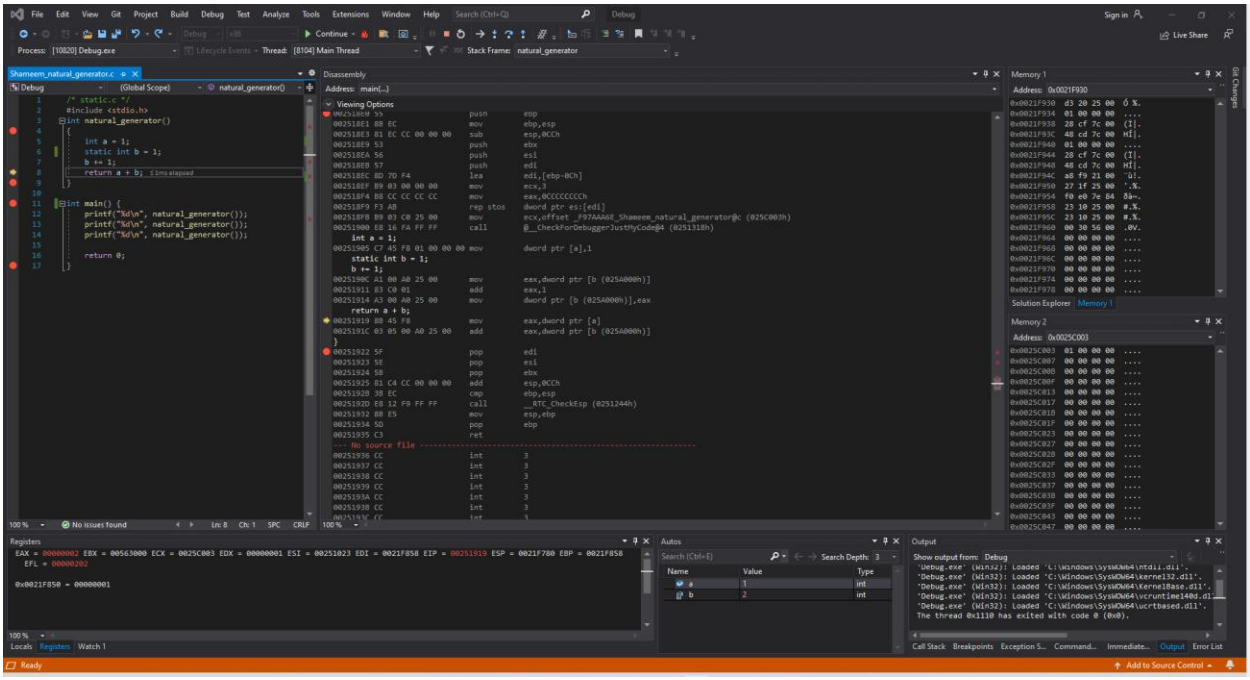


Figure 79: Shameem_natural_generator.c code ran in Visual Studio (Line 7)

Disassembly: This window shows that the code runs lines 5-6 which has integer a and static int b.

Registers: This window shows that the value of EAX is now 00000002 because b has been incremented.

Memory: Memory window 1 shows the values around the stack pointer. While Memory window 2 displays the value of the static variables stored into the addresses.

Variable b is stored at address 0x0025C003 and has the value 01 00 00 00.

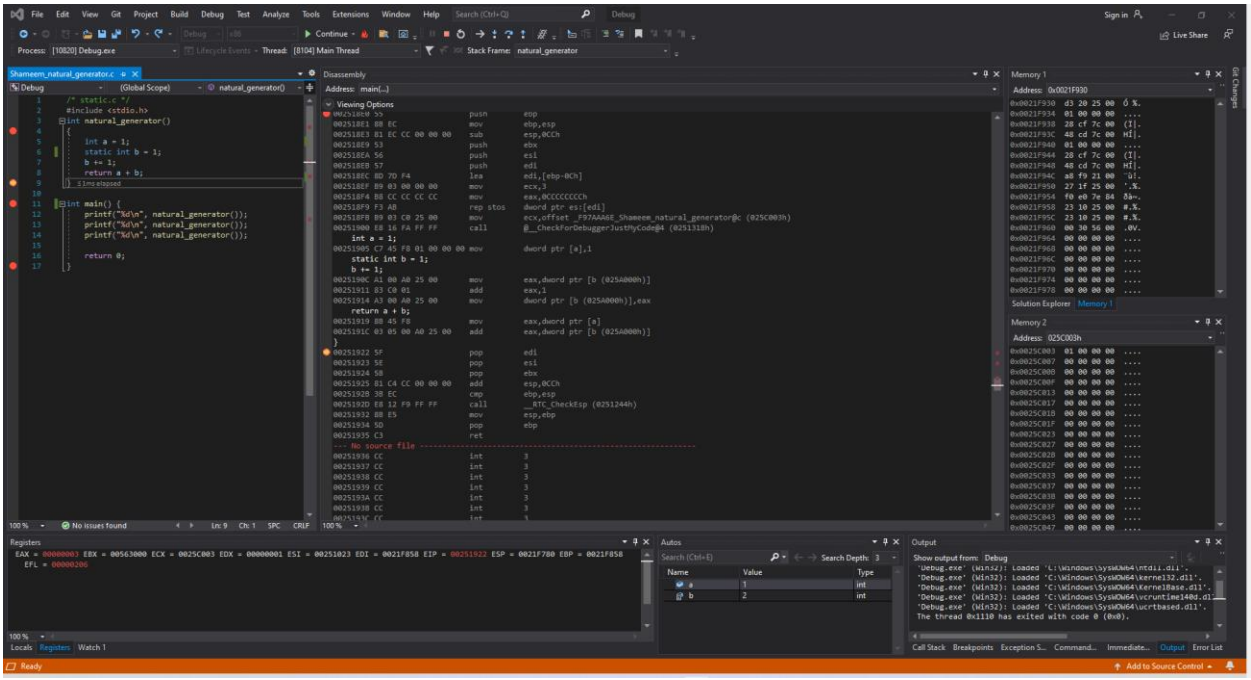


Figure 80: Shameem_natural_generator.c code ran in Visual Studio (Line 8)

Disassembly: This window shows that the code runs lines 5-6 which has integer a and static int b.

Registers: This window shows that the value of EAX is now 00000003 because the function now returns a + b which is 1 + 2.

Memory: Memory window 1 shows the values around the stack pointer. While Memory window 2 displays the value of the static variables stored into the addresses.

Variable b is stored at address 0x0025C003 and has the value 01 00 00 00.

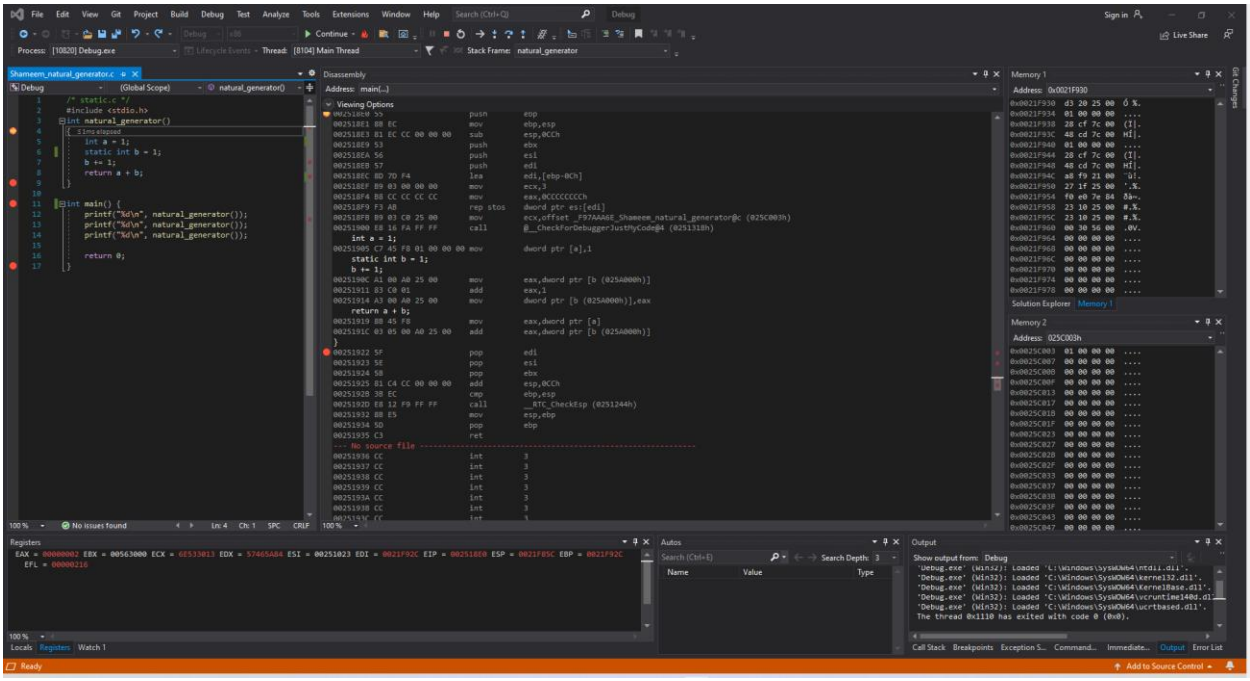


Figure 81: Shameem_natural_generator.c code ran in Visual Studio (Lines 13, 3-8)

Disassembly: This window shows that the code runs lines 3-8.

Registers: This window shows that the value of EAX is now 00000002.

Memory: Memory window 1 shows the values around the stack pointer. While Memory window 2 displays the value of the static variables stored into the addresses.

Variable b is stored at address 0x0025C003 and has the value 01 00 00 00.

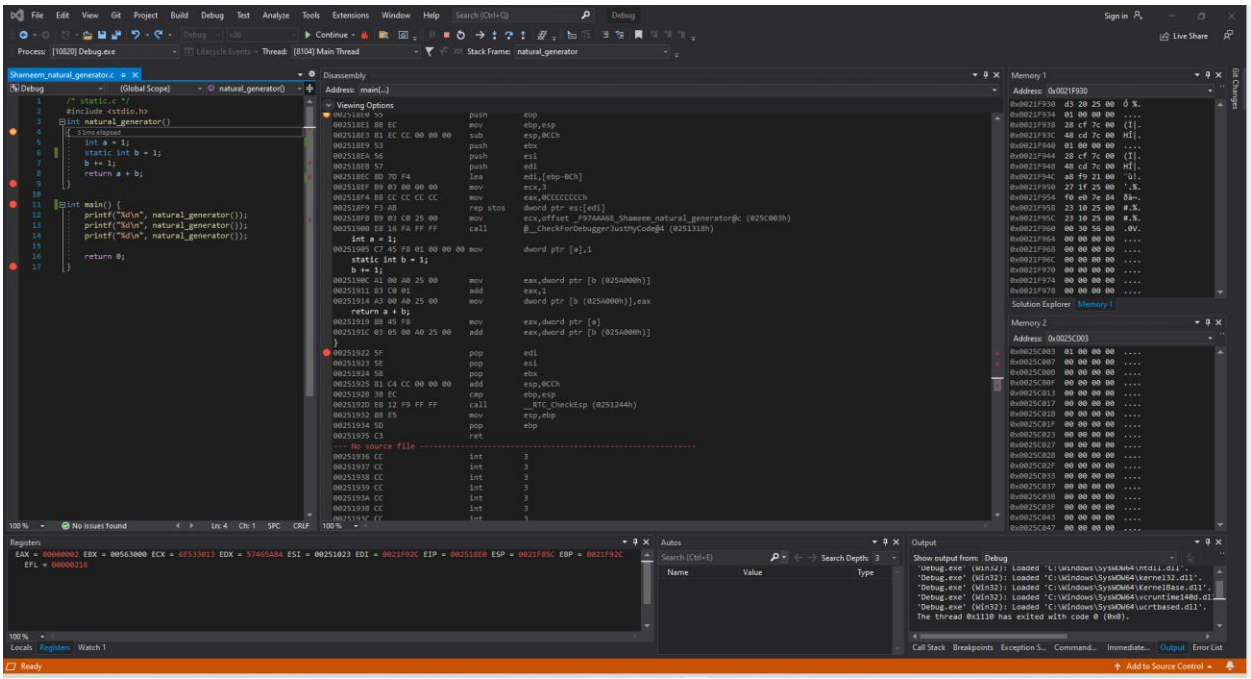


Figure 82: Shameem_natural_generator.c code ran in Visual Studio (Lines 14, 3-8)

Disassembly: This window shows that the code runs lines 3-8 which has integer a and static int b.

Registers: This window shows that the value of EAX is now 00000002.

Memory: Memory window 1 shows the values around the stack pointer. While Memory window 2 displays the value of the static variables stored into the addresses.

Variable b is stored at address 0x0025C003 and has the value 01 00 00 00.

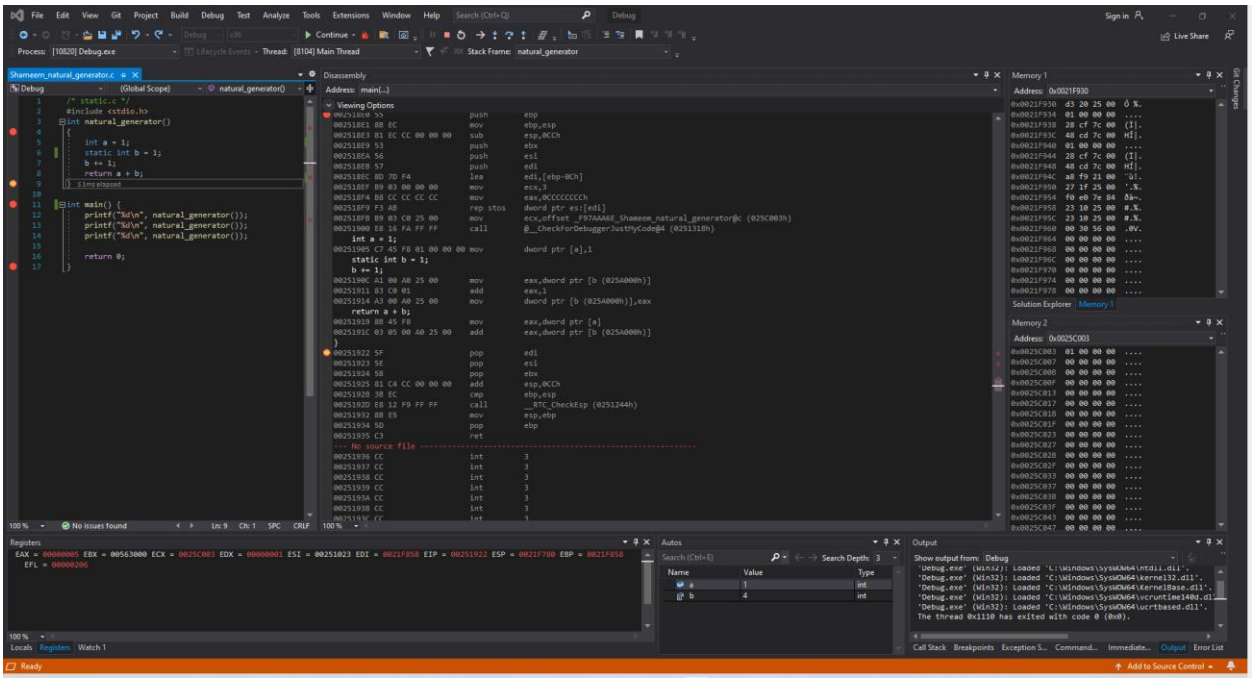


Figure 82: Shameem_natural_generator.c code ran in Visual Studio (Lines 14, 3-8)

Disassembly: This window shows that the code runs lines 3-8 which has integer a and static int b.

Registers: This window shows that the value of EAX is now 00000005 and EDX is 00000001.

Memory: Memory window 1 shows the values around the stack pointer. While Memory window 2 displays the value of the static variables stored into the addresses.

Variable b is stored at address 0x0025C003 and has the value 01 00 00 00.

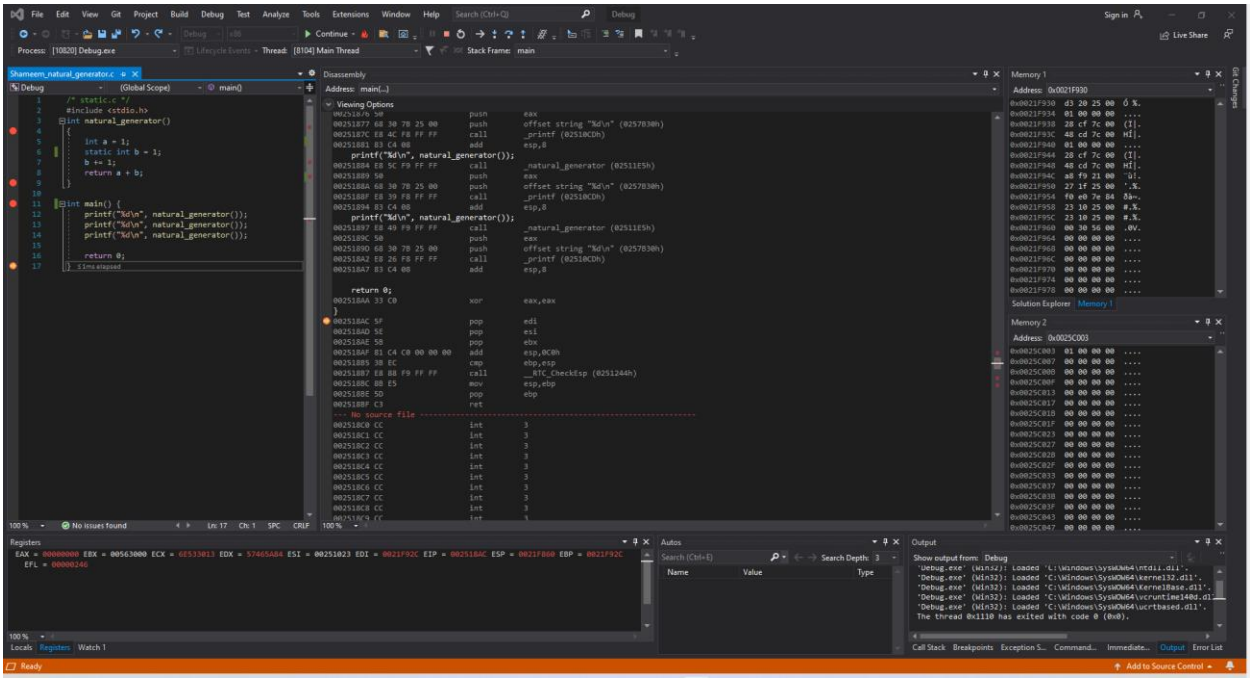


Figure 83: Shameem_natural_generator.c code ran in Visual Studio (Line 16)

Disassembly: This window shows that the code runs line 16 which returns.

Registers: This window shows that the value of EAX is now 00000001 because int main() returns 0.

Memory: Memory window 1 shows the values around the stack pointer. While Memory window 2 displays the value of the static variables stored into the addresses.

Variable b is stored at address 0x0025C003 and has the value 01 00 00 00.

myadd.c

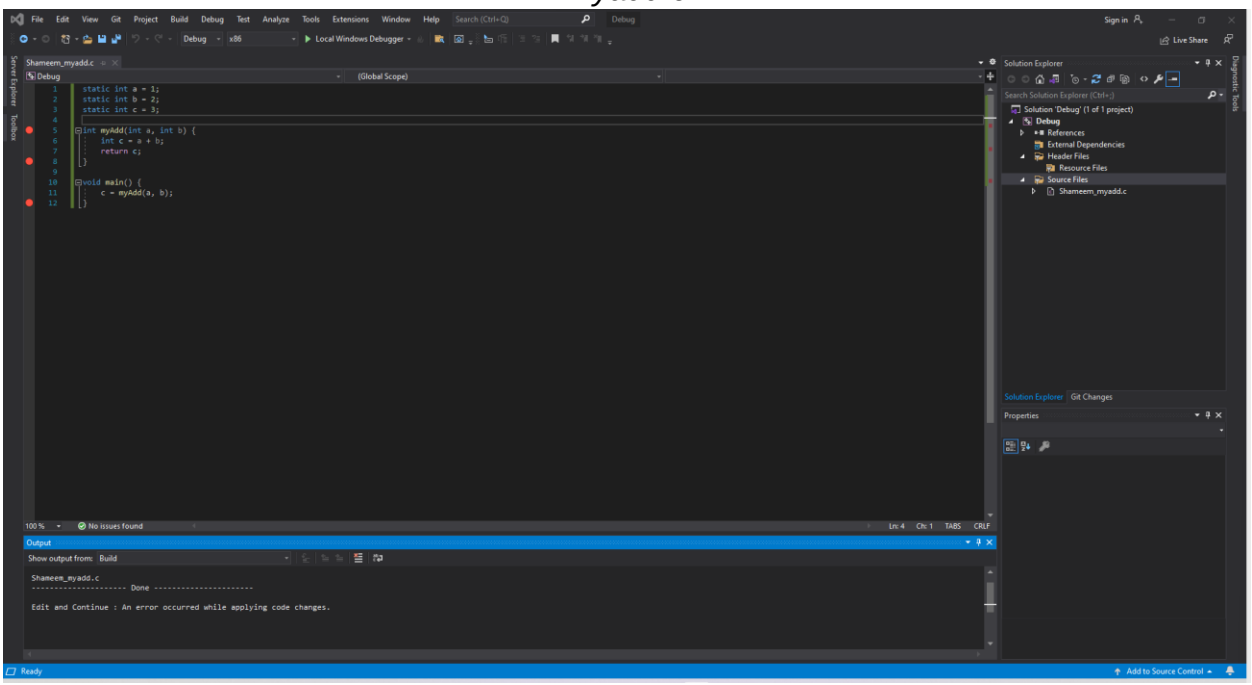


Figure 84: Shameem_natural_generator.c code in Visual Studio

The code has three static variables a, b and c whose values are 1, 2, 3 respectively. The function myAdd() takes variables a and b and adds them together then stores that result into the variable c and returns. The void main() next sets c equal to the value of the function myAdd() and then exits the program.

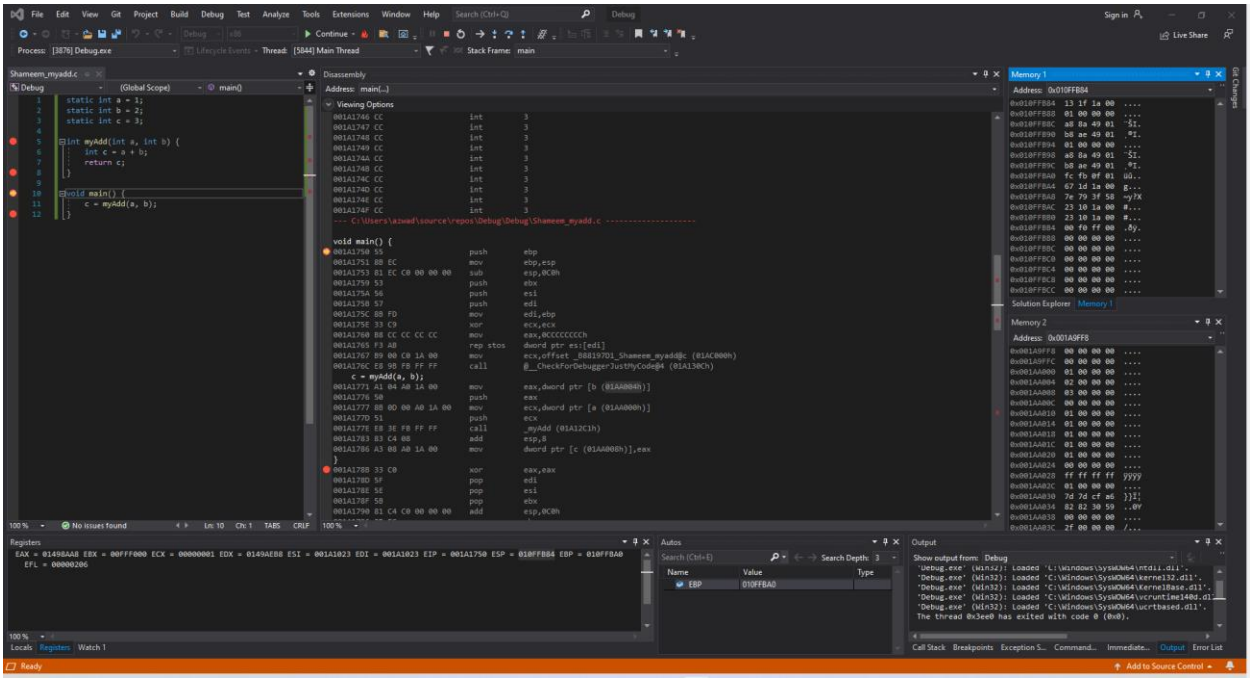


Figure 85: Shameem_myadd.c code ran in Visual Studio (before Line 10)

Disassembly: This window shows that it starts debugging at line 10.

Registers: This window shows that the value of ESP, the stack pointer, is 010FFB84.

Memory: Memory window 1 shows the values around the stack pointer which is random. While Memory window 2 displays the value of the static variables stored into the addresses.

Variable a is stored at address 0x001A9FFCand has the value 01 00 00 00.
Variable b is stored at address 0x001AA000and has the value 02 00 00 00.
Variable c is stored at address 0x001AA000and has the value 03 00 00 00.

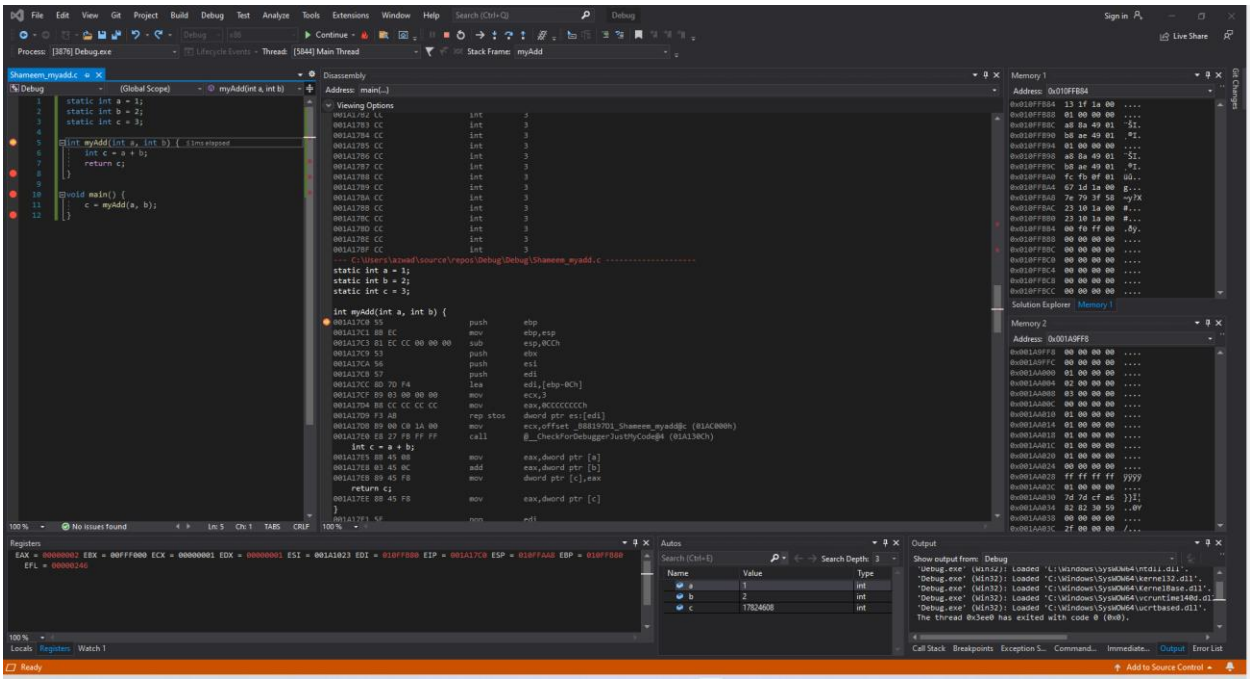


Figure 86: Shameem_myadd.c code ran in Visual Studio (before Line 5)

Disassembly: This window shows that the code is right now on the myAdd() function and is about to run the function at line 5.

Registers: This window shows that the values have changed and that the EAX is now 00000002, which means that the value of variable b is now in the register.

Memory: Memory window 1 shows the values around the stack pointer which is somewhat initialized. While Memory window 2 displays the value of the static variables stored into the addresses.

Variable a is stored at address 0x001A9FFCand has the value 01 00 00 00.
Variable b is stored at address 0x001AA000and has the value 02 00 00 00.
Variable c is stored at address 0x001AA000and has the value 03 00 00 00.

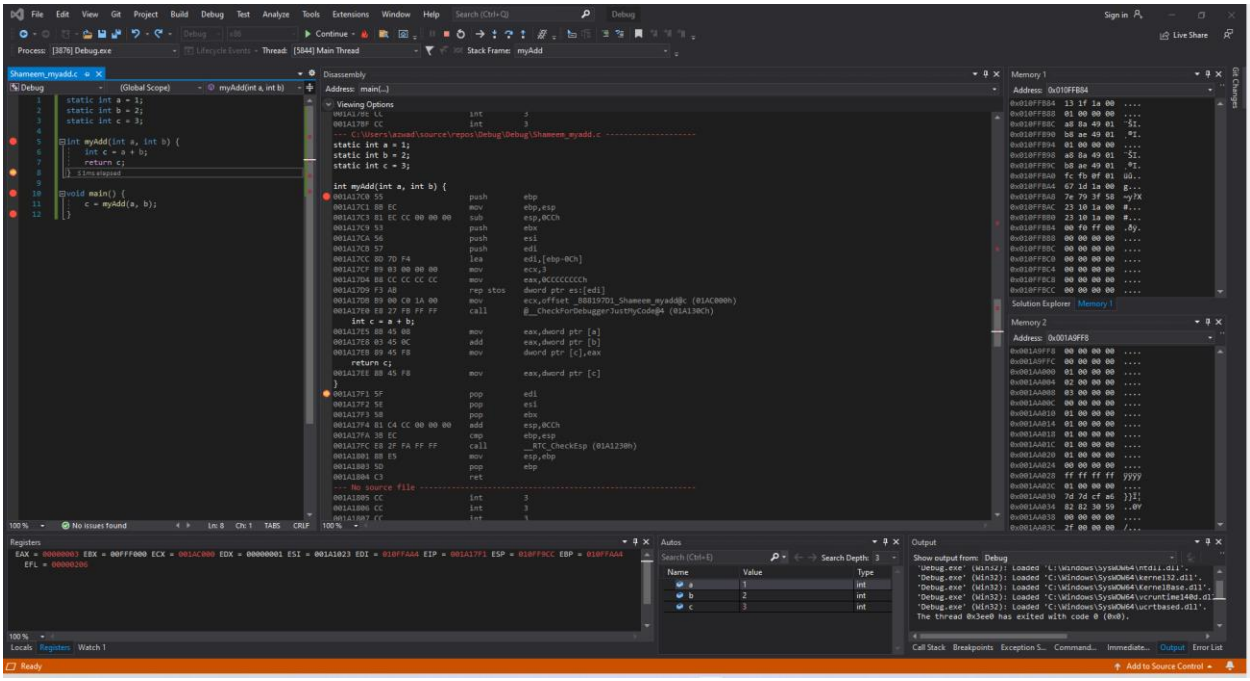


Figure 87: Shameem_myadd.c code ran in Visual Studio (Line 5-7)

Disassembly: This window shows that the code ran and finished running the myAdd() function.

Registers: This window shows that the values have changed and that the EAX is now 00000003, this is the value of c which is the value of the addition of a and b.

Memory: Memory window 1 shows the values around the stack pointer. While Memory window 2 displays the value of the static variables stored into the addresses.

Variable a is stored at address 0x001A9FFCand has the value 01 00 00 00.
Variable b is stored at address 0x001AA000and has the value 02 00 00 00.
Variable c is stored at address 0x001AA000and has the value 03 00 00 00.

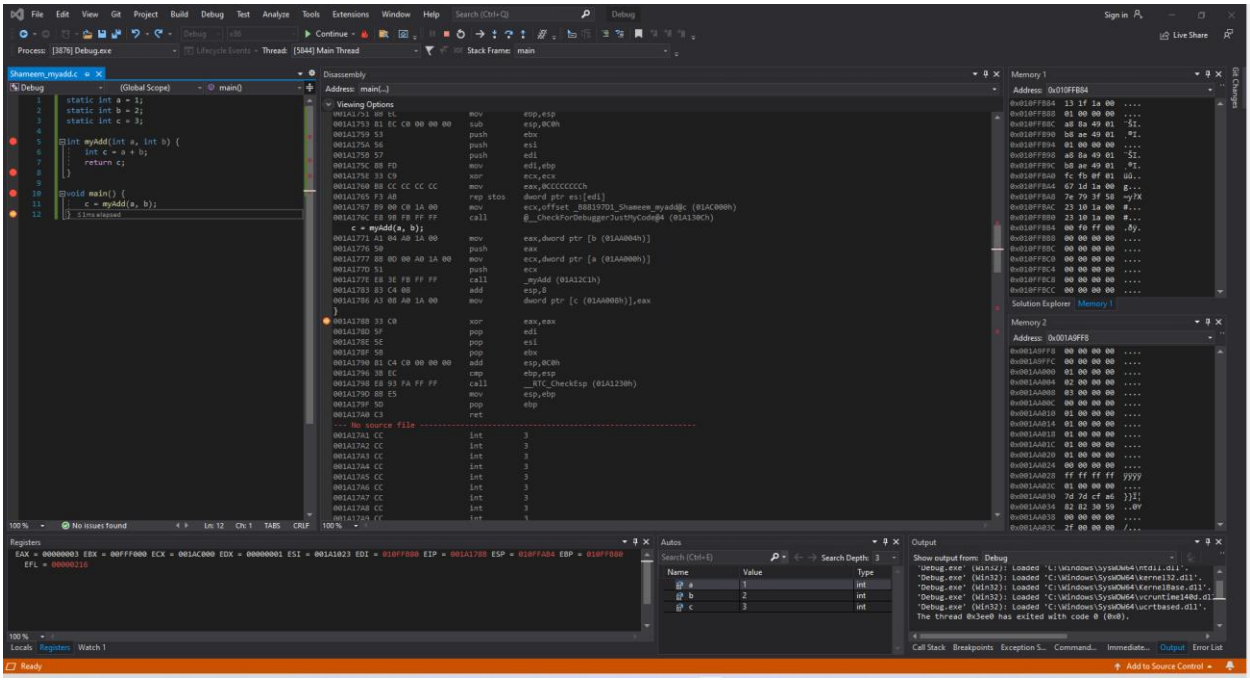


Figure 88: Shameem_myadd.c code ran in Visual Studio (Before Line 12)

Disassembly: This window shows that the code ran and is about to exit the void main().

Registers: This window shows that the values have changed because the code is back in void main() currently.

Memory: Memory window 1 shows the values around the stack pointer. While Memory window 2 displays the value of the static variables stored into the addresses.

Variable a is stored at address 0x001A9FFCand has the value 01 00 00 00.
Variable b is stored at address 0x001AA000and has the value 02 00 00 00.
Variable c is stored at address 0x001AA000and has the value 03 00 00 00.

X86_64 ISA Linux 64-bit

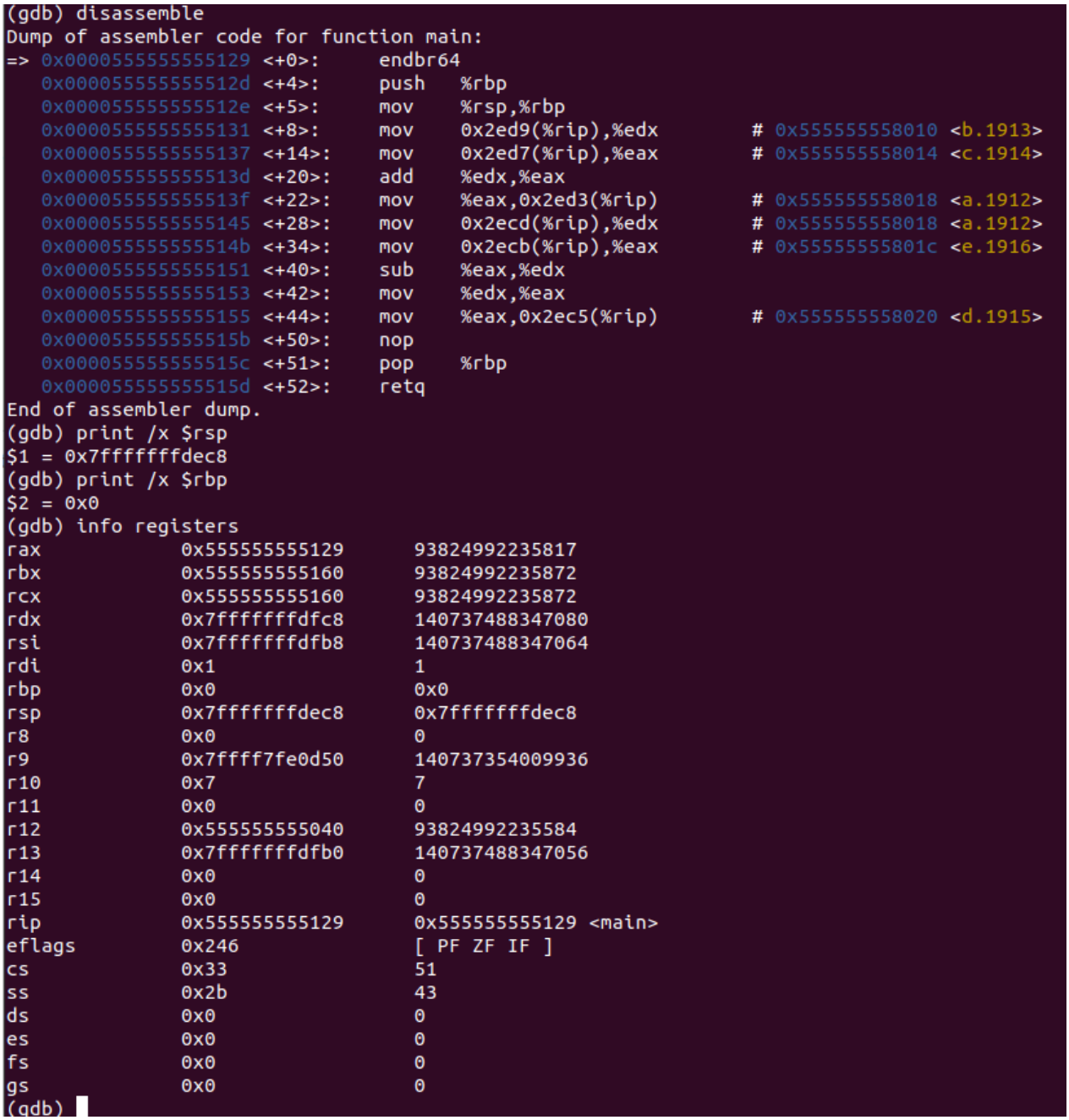
2-2_1.c

```
(gdb) list
1       void main() {
2               static int a = 1;
3               static int b = 2;
4               static int c = 3;
5               static int d = 4;
6               static int e = 5;
7               a = b + c;
8               d = a - e;
9       }
```

Figure 89: Shameem_2-2_1.c code in gdb

The code written is written in 2-2_1.c and is shown in figure 1, which is the image above. The code entails five static variables, a, b, c, d, e and these five variables are used to perform arithmetic which is stored in memory afterwards. The arithmetic performed is the addition of b and c, whose result is stored in a. The other arithmetic is performed with the subtraction of a and e, whose result is stored in d.

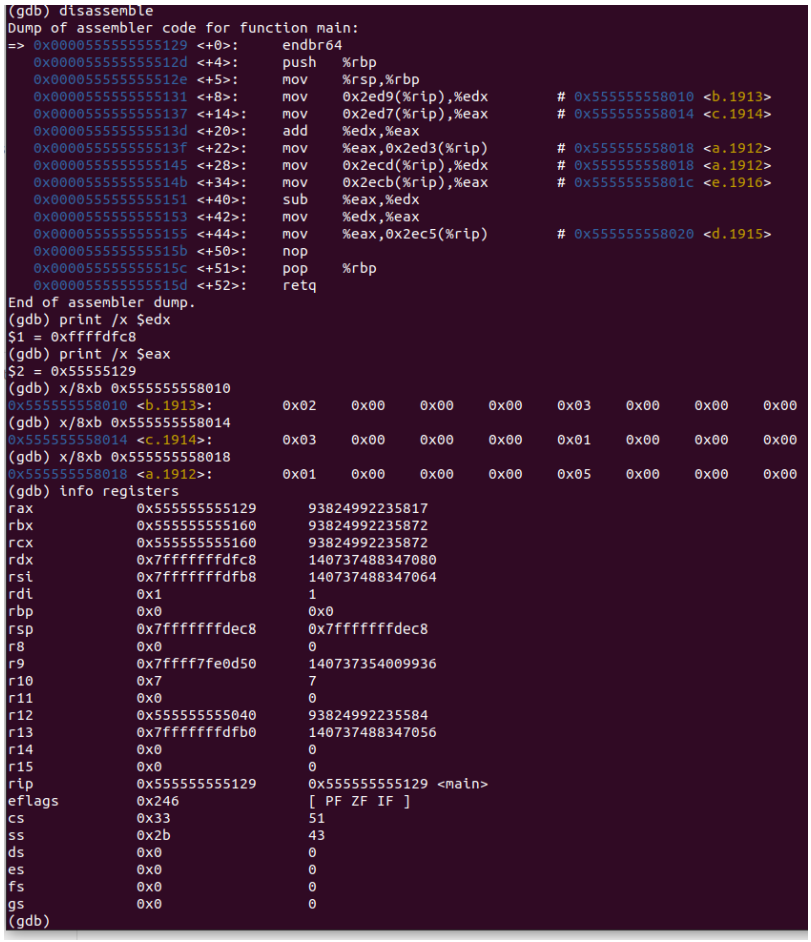
Figure 90: Shameem_2-2_1.c code in gdb Line 1



Disassembly: This window shows that the code runs line 1 and sets stack and base pointer

Registers: This window shows the value that's stored in the stack pointer which is 0x7fffffffdf78 and the value that's stored in the base pointer which is at 0x0.

Figure 91: Shameem_2-2_1.c code in gdb Lines 1-8



Disassembly: This shows that the code runs lines 1-8. In lines 1-8 the values are moved into
registers.

Registers: This shows the value that's stored in each register.

The value of b in memory is moved to edx.
The value of c in memory is moved to eax.
The addition is performed on the two registers and result is stored in eax.
The value in addition value in eax is moved into the memory and stored in the address of a which overwrites the value of a.

Memory: This shows the values stored in the addresses.

0x555555558010 contains 0x02 0x00 0x00 0x00 0x03 0x00 0x00 0x00
0x555555558014 contains 0x03 0x00 0x00 0x00 0x01 0x00 0x00 0x00
0x555555558018 contains 0x01 0x00 0x00 0x00 0x05 0x00 0x00 0x00

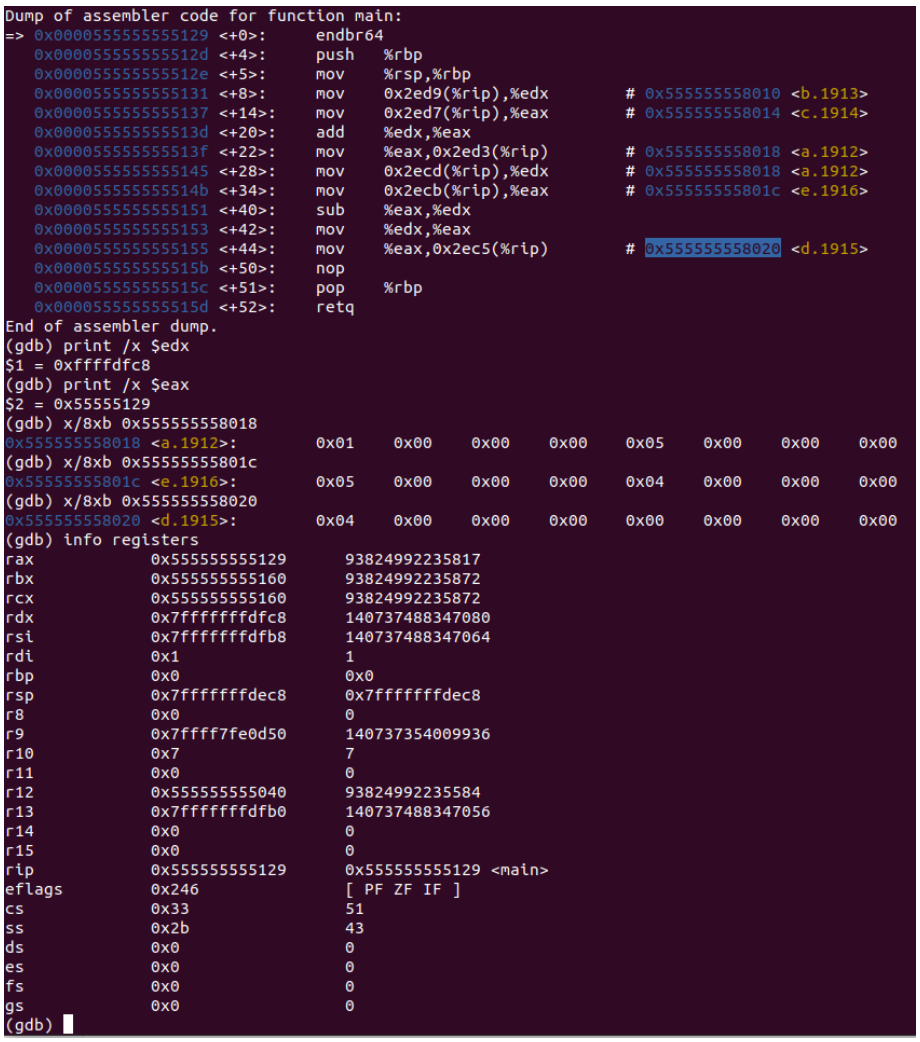


Figure 92: Shameem_2-2_1.c code in gdb Lines 8-9

Disassembly: This shows that the code runs lines 8-9.
Registers: This shows the values that are stored in each register.

The value of a in memory is moved to edx.
The value of e in memory is moved to eax.
Then subtraction is performed on the two registers and result is stored in edx.
Then eax is set to the same value as edx.
The value of eax is then moved into the memory and stored in the address of f which overwrites the previous value at f.

Memory: This shows the values stored in the addresses.
The address 0x555555558018 has value 0x01 0x00 0x00 0x00 0x04 0x00 0x00 0x00.
The address 0x55555555801c has value 0x05 0x00 0x00 0x00 0x04 0x00 0x00 0x00.
The address 0x555555558020 has value 0x04 0x00 0x00 0x00 0x00 0x00 0x00 0x00.

2-2_2.c

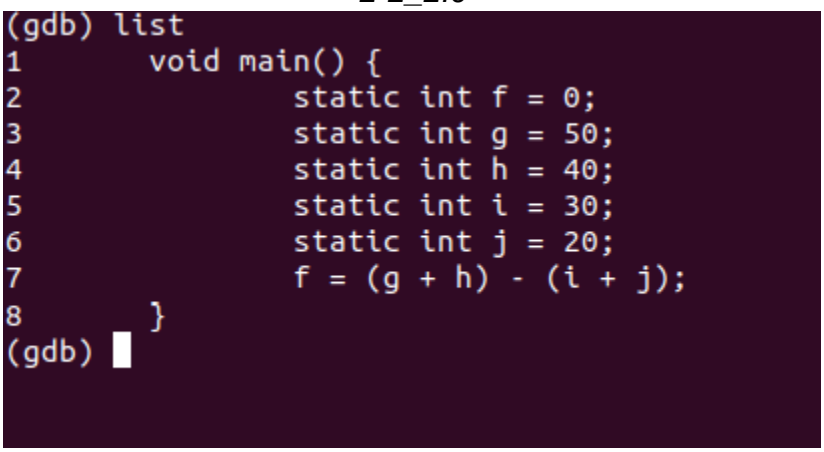


Figure 93: Shameem_2-2_2.c code in gdb

The code written for 2-2_2.c uses five static variables f, g, h, I, j with the values 0, 50, 40, 30, 20 respectively. The code then uses the values of the variables to perform two additions and subtractions. The additions entail g + h and i + j which the result of the two additions is subtracted and stored into the variable f.

```
(gdb) disassemble
Dump of assembler code for function main:
=> 0x0000555555555129 <+0>:     endbr64
   0x000055555555512d <+4>:     push   %rbp
   0x000055555555512e <+5>:     mov    %rsp,%rbp
   0x0000555555555131 <+8>:     mov    0x2ed9(%rip),%edx        # 0x555555558010 <g.1913>
   0x0000555555555137 <+14>:    mov    0x2ed7(%rip),%eax        # 0x555555558014 <h.1914>
   0x000055555555513d <+20>:    lea    (%rdx,%rax,1),%ecx
   0x0000555555555140 <+23>:    mov    0x2ed2(%rip),%edx        # 0x555555558018 <i.1915>
   0x0000555555555146 <+29>:    mov    0x2ed0(%rip),%eax        # 0x55555555801c <j.1916>
   0x000055555555514c <+35>:    add    %edx,%eax
   0x000055555555514e <+37>:    sub    %eax,%ecx
   0x0000555555555150 <+39>:    mov    %ecx,%eax
   0x0000555555555152 <+41>:    mov    %eax,0x2ecc(%rip)        # 0x555555558024 <f.1912>
   0x0000555555555158 <+47>:    nop
   0x0000555555555159 <+48>:    pop    %rbp
   0x000055555555515a <+49>:    retq
End of assembler dump.
(gdb) print /x $rsp
$1 = 0x7fffffffdec8
(gdb) print /x $rbp
$2 = 0x0
(gdb) info registers
rax            0x555555555129      93824992235817
rbx            0x555555555160      93824992235872
rcx            0x555555555160      93824992235872
rdx            0x7fffffffdfc8      140737488347080
rsi            0x7fffffffdfb8      140737488347064
rdi            0x1                 1
rbp            0x0                 0x0
rsp            0x7fffffffdec8      0x7fffffffdec8
r8             0x0                 0
r9             0x7ffff7fe0d50      140737354009936
r10            0x7                 7
r11            0x0                 0
r12            0x555555555040      93824992235584
r13            0x7fffffffdfb0      140737488347056
r14            0x0                 0
r15            0x0                 0
rip            0x555555555129      0x555555555129 <main>
eflags         0x246               [ PF ZF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb)
```

Figure 94: Shameem_2-2_1.c code in gdb before line 1

Disassembly: This shows that the code runs line 1 and that stack pointer and base pointer are set.

Registers: This shows the value that's stored in each register which is the stack pointer at 0x7fffffffdf78 and the base pointer at 0x0.

```
(gdb) disassemble
Dump of assembler code for function main:
=> 0x0000555555555129 <+0>:     endbr64
   0x000055555555512d <+4>:     push   %rbp
   0x000055555555512e <+5>:     mov    %rsp,%rbp
   0x0000555555555131 <+8>:     mov    0x2ed9(%rip),%edx        # 0x555555558010 <g.1913>
   0x0000555555555137 <+14>:    mov    0x2ed7(%rip),%eax        # 0x555555558014 <h.1914>
   0x000055555555513d <+20>:    lea    (%rdx,%rax,1),%ecx
   0x0000555555555140 <+23>:    mov    0x2ed2(%rip),%edx        # 0x555555558018 <i.1915>
   0x0000555555555146 <+29>:    mov    0x2ed0(%rip),%eax        # 0x55555555801c <j.1916>
   0x000055555555514c <+35>:    add    %edx,%eax
   0x000055555555514e <+37>:    sub    %eax,%ecx
   0x0000555555555150 <+39>:    mov    %ecx,%eax
   0x0000555555555152 <+41>:    mov    %eax,0x2ecc(%rip)        # 0x555555558024 <f.1912>
   0x0000555555555158 <+47>:    nop
   0x0000555555555159 <+48>:    pop    %rbp
   0x000055555555515a <+49>:    retq
End of assembler dump.
(gdb) print /x $edx
$9 = 0xffffdfc8
(gdb) print /x $eax
$10 = 0x55555129
(gdb) print /x $ecx
$11 = 0x55555160
(gdb) x/8xb 0x555555558010
0x555555558010 <g.1913>:        0x32    0x00    0x00    0x00    0x28    0x00    0x00    0x00
(gdb) x/8xb 0x555555558014
0x555555558014 <h.1914>:        0x28    0x00    0x00    0x00    0x1e    0x00    0x00    0x00
(gdb) x/8xb 0x555555558018
0x555555558018 <i.1915>:        0x1e    0x00    0x00    0x00    0x14    0x00    0x00    0x00
(gdb) x/8xb 0x55555555801c
0x55555555801c <j.1916>:        0x14    0x00    0x00    0x00    0x00    0x00    0x00    0x00
(gdb) x/8xb 0x555555558024
0x555555558024 <f.1912>:        0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
(gdb) info registers
rax            0x555555555129      93824992235817
rbx            0x555555555160      93824992235872
rcx            0x555555555160      93824992235872
rdx            0x7fffffffdfc8      140737488347080
rsi            0x7fffffffdfb8      140737488347064
rdi            0x1                 1
rbp            0x0                 0x0
rsp            0x7fffffffdec8      0x7fffffffdec8
r8             0x0                 0
r9             0x7ffff7fe0d50      140737354009936
r10            0x7                 7
r11            0x0                 0
r12            0x555555555040      93824992235584
r13            0x7fffffffdfb0      140737488347056
r14            0x0                 0
r15            0x0                 0
rip            0x555555555129      0x555555555129 <main>
eflags         0x246               [ PF ZF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb)
```

Figure 95: Shameem_2-2_1.c code in gdb before line 1-8

Disassembly: This shows that the code runs line 1 and that stack pointer and base pointer are set.

Registers: This shows the value that's stored in the registers.

The value of g in memory is moved to edx.
The value of h in memory is moved to eax.
Then the operation is performed with the two registers and then the result is put in ecx. The value of i in memory is moved to edx.
The value of j in memory is moved to eax.
Then the operation is performed with the two registers and result is put in eax.
The addition is performed with two registers and result is stored in eax.
The subtraction is performed with eax and ecx and that result is put into ecx.
The value in the register is then into the memory which is stored in the address of f which then overwrites the previous value of f.

Memory: This shows the values stored in an address.

0x555555558010 contains 0x32 0x00 0x00 0x00 0x28 0x00 0x00 0x00
0x555555558014 contains 0x28 0x00 0x00 0x00 0x1e 0x00 0x00 0x00
0x555555558018 contains 0x1e 0x00 0x00 0x00 0x14 0x00 0x00 0x00
0x55555555801c contains 0x14 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x555555558020 contains 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

2-3_1.c

```
(gdb) list
1       void main() {
2               static int g = 0;
3               static int h = 22;
4               static int A[100];
5               A[8] = 55;
6               g = h + A[8];
7       }
(gdb)
```

Figure 96: Shameem_2-3_1.c code in gdb

The code has two static variables g and h plus the array A. The code uses the static variables to set the 8th index of the array A to the value 55 and then sets the value of g to the result of the addition of h and the 8th index of the array A.

```
(gdb) disassemble
Dump of assembler code for function main:
=> 0x0000555555555129 <+0>:     endbr64
   0x000055555555512d <+4>:     push   %rbp
   0x000055555555512e <+5>:     mov    %rsp,%rbp
   0x0000555555555131 <+8>:     movl   $0x37,0x2f25(%rip)        # 0x555555558060 <A.1914+32>
   0x000055555555513b <+18>:    mov    0x2f1f(%rip),%edx        # 0x555555558060 <A.1914+32>
   0x0000555555555141 <+24>:    mov    0x2ec9(%rip),%eax        # 0x555555558010 <h.1913>
   0x0000555555555147 <+30>:    add    %edx,%eax
   0x0000555555555149 <+32>:    mov    %eax,0x3081(%rip)        # 0x5555555581d0 <g.1912>
   0x000055555555514f <+38>:    nop
   0x0000555555555150 <+39>:    pop    %rbp
   0x0000555555555151 <+40>:    retq
End of assembler dump.
(gdb) print /x $rsp
$1 = 0x7fffffffdec8
(gdb) print /x $rbp
$2 = 0x0
(gdb) info registers
rax            0x555555555129      93824992235817
rbx            0x555555555160      93824992235872
rcx            0x555555555160      93824992235872
rdx            0x7fffffffdfc8      140737488347080
rsi            0x7fffffffdfb8      140737488347064
rdi            0x1                 1
rbp            0x0                 0x0
rsp            0x7fffffffdec8      0x7fffffffdec8
r8             0x0                 0
r9             0x7ffff7fe0d50      140737354009936
r10            0x7                 7
r11            0x0                 0
r12            0x555555555040      93824992235584
r13            0x7fffffffdfb0      140737488347056
r14            0x0                 0
r15            0x0                 0
rip            0x555555555129      0x555555555129 <main>
eflags         0x246               [ PF ZF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb)
```

Figure 97: Shameem_2-3_1.c code in gdb before line 1

Disassembly: This shows that the code runs line 1 and that stack pointer and base pointer are set.

Registers: This shows the value that's stored in each register which is the stack pointer at 0x7fffffffdf78 and the base pointer at 0x0.

```
(gdb) disassemble
Dump of assembler code for function main:
=> 0x0000555555555129 <+0>:     endbr64
   0x000055555555512d <+4>:     push   %rbp
   0x000055555555512e <+5>:     mov    %rsp,%rbp
   0x0000555555555131 <+8>:     movl   $0x37,0x2f25(%rip)        # 0x555555558060 <A.1914+32>
   0x000055555555513b <+18>:    mov    0x2f1f(%rip),%edx         # 0x555555558060 <A.1914+32>
   0x0000555555555141 <+24>:    mov    0x2ec9(%rip),%eax         # 0x555555558010 <h.1913>
   0x0000555555555147 <+30>:    add    %edx,%eax
   0x0000555555555149 <+32>:    mov    %eax,0x3081(%rip)         # 0x5555555581d0 <g.1912>
   0x000055555555514f <+38>:    nop
   0x0000555555555150 <+39>:    pop    %rbp
   0x0000555555555151 <+40>:    retq
End of assembler dump.
(gdb) print /x $edx
$3 = 0xffffdfc8
(gdb) print /x $eax
$4 = 0x55555129
(gdb) x/8xb 0x555555558060
0x555555558060 <A.1914+32>:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
(gdb) info registers
rax            0x555555555129      93824992235817
rbx            0x555555555160      93824992235872
rcx            0x555555555160      93824992235872
rdx            0x7fffffffdfc8      140737488347080
rsi            0x7fffffffdfb8      140737488347064
rdi            0x1                 1
rbp            0x0                 0x0
rsp            0x7fffffffdec8      0x7fffffffdec8
r8             0x0                 0
r9             0x7ffff7fe0d50      140737354009936
r10            0x7                 7
r11            0x0                 0
r12            0x555555555040      93824992235584
r13            0x7fffffffdfb0      140737488347056
r14            0x0                 0
r15            0x0                 0
rip            0x555555555129      0x555555555129 <main>
eflags         0x246               [ PF ZF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb)
```

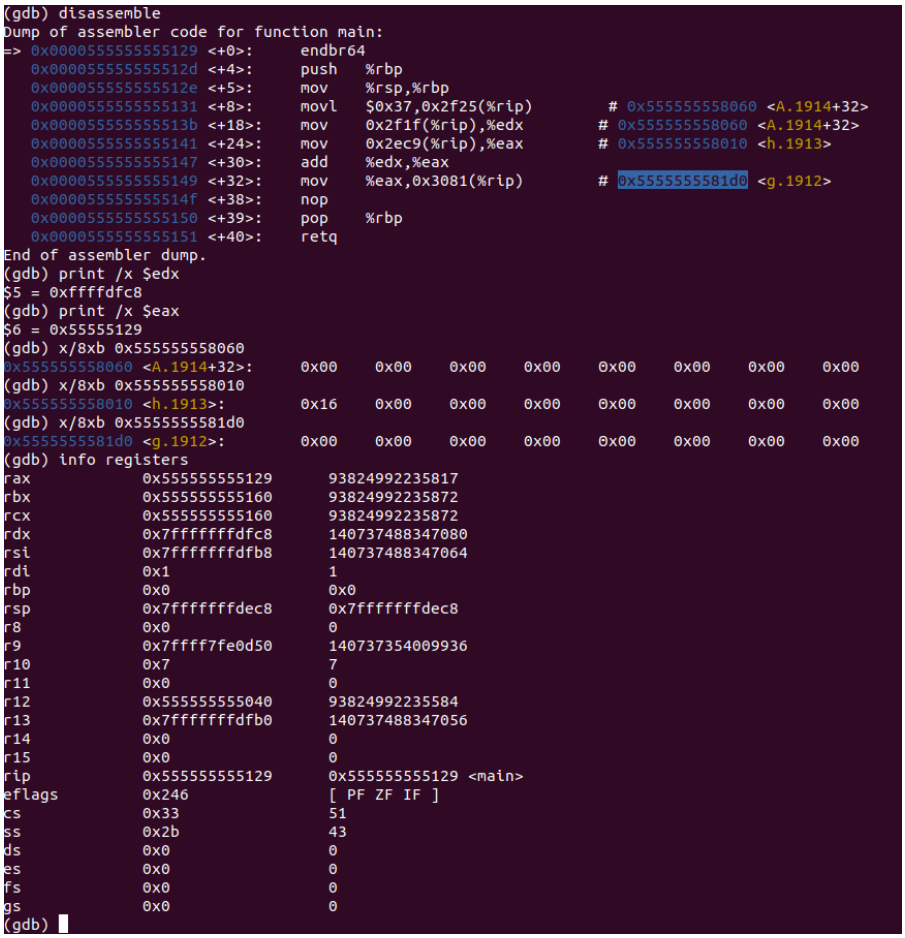
Figure 98: Shameem_2-3_1.c code in gdb line 1-5

Disassembly: This shows that the code runs lines 1-5 and the 8th index of the array A is set to 55.

Registers: This shows that the values don't change in lines 1-5

Memory: This shows the address 0x555555558060 has the value 0x32 0x00 0x00 0x00 0x28 0x00 0x00 0x00.

Figure 99: Shameem_2-3_1.c code in gdb lines 6-7



Disassembly: This shows that the code runs lines 6-7 and that the values are moved into registers for addition and the result of the addition is stored in memory.

Registers: This shows that the values stored in the registers.

The value of the 8th index of array A is stored in edx.
The value of h in memory is moved to eax.
The addition is performed with the 8th index of array A and h and is stored in eax.
The value result of the addition is then moved into the memory and stored in the address of f which overwrites the previous value of f.

Memory: This shows the addresses and the values they contain.
The address 0x555555558060 has the value 0x37 0x00 0x00 0x00 0x00 0x00 0x00 0x00.
The address 0x555555558010 has the value 0x16 0x00 0x00 0x00 0x00 0x00 0x00 0x00.
The address 0x5555555581d0 has the value 0x4d 0x00 0x00 0x00 0x00 0x00 0x00 0x00

2-3_2.c

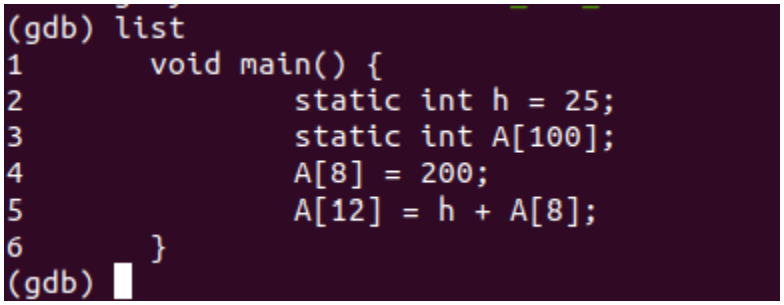


Figure 100: Shameem_2-3_2.c code in gdb

The code has one static variable h and the array A. The code then sets the 8th index of the array to 200. Then the code sets the 12th index of the array to the value of h plus the 8th index of the array.

```
Dump of assembler code for function main:
=> 0x0000555555555129 <+0>:     endbr64
   0x000055555555512d <+4>:     push   %rbp
   0x000055555555512e <+5>:     mov    %rsp,%rbp
   0x0000555555555131 <+8>:     movl   $0xc8,0x2f25(%rip)        # 0x555555558060 <A.1913+32>
   0x000055555555513b <+18>:    mov    0x2f1f(%rip),%edx        # 0x555555558060 <A.1913+32>
   0x0000555555555141 <+24>:    mov    0x2ec9(%rip),%eax        # 0x555555558010 <h.1912>
   0x0000555555555147 <+30>:    add    %edx,%eax
   0x0000555555555149 <+32>:    mov    %eax,0x2f21(%rip)        # 0x555555558070 <A.1913+48>
   0x000055555555514f <+38>:    nop
   0x0000555555555150 <+39>:    pop    %rbp
   0x0000555555555151 <+40>:    retq
End of assembler dump.
(gdb) print /x $rsp
$1 = 0x7fffffffdec8
(gdb) print /x $rbp
$2 = 0x0
(gdb) info registers
rax            0x555555555129      93824992235817
rbx            0x555555555160      93824992235872
rcx            0x555555555160      93824992235872
rdx            0x7fffffffdfc8      140737488347080
rsi            0x7fffffffdfb8      140737488347064
rdi            0x1                 1
rbp            0x0                 0x0
rsp            0x7fffffffdec8      0x7fffffffdec8
r8             0x0                 0
r9             0x7ffff7fe0d50      140737354009936
r10            0x7                 7
r11            0x0                 0
r12            0x555555555040      93824992235584
r13            0x7fffffffdfb0      140737488347056
r14            0x0                 0
r15            0x0                 0
rip            0x555555555129      0x555555555129 <main>
eflags         0x246               [ PF ZF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb)
```

Figure 101: Shameem_2-3_2.c code in gdb before line 1

Disassembly: This shows that the code runs line 1 and that stack pointer and base pointer are set.

Registers: This shows the value that's stored in each register which is the stack pointer at 0x7fffffffdf78 and the base pointer at 0x0.

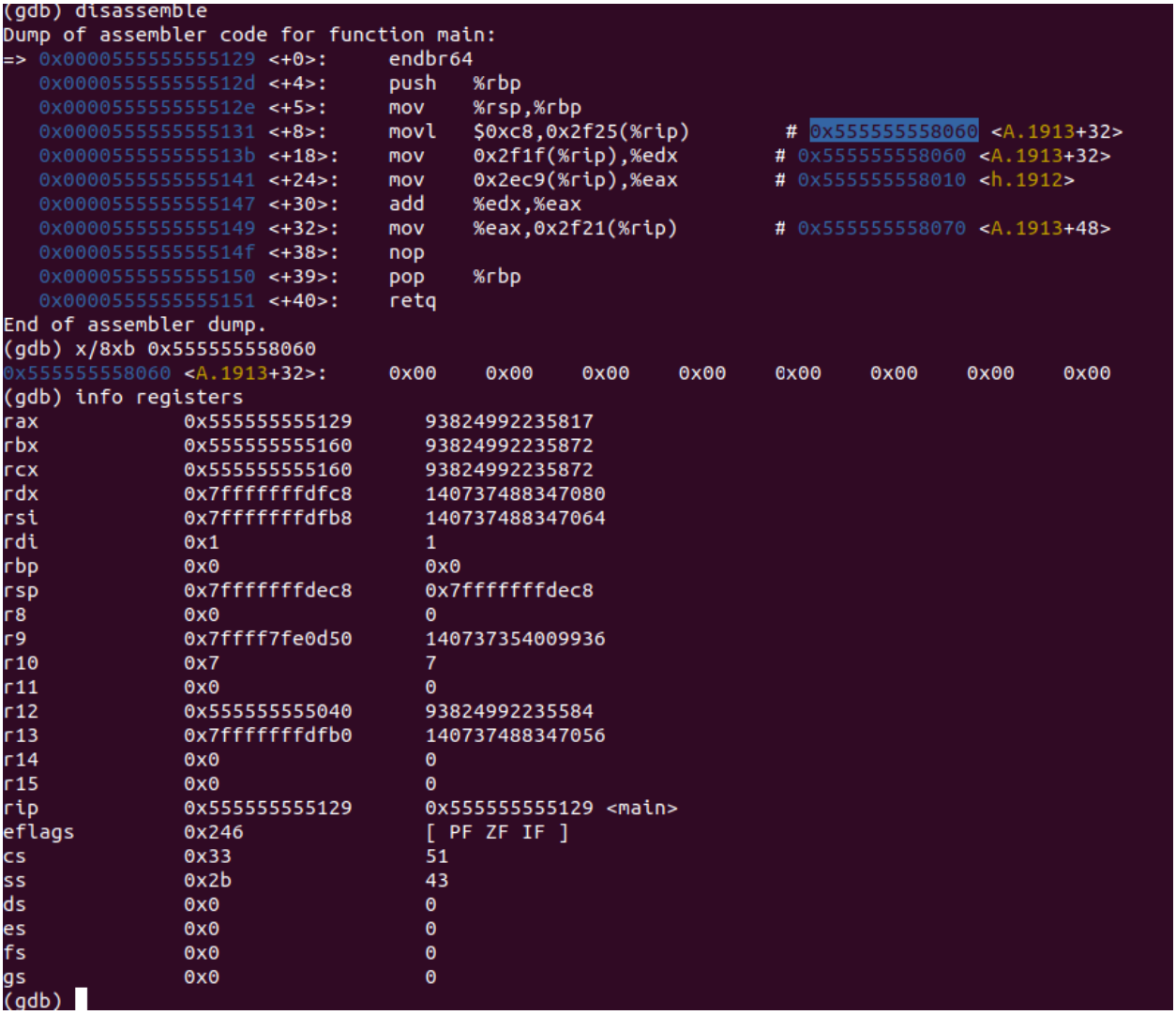


Figure 102: Shameem_2-3_2.c code in gdb lines 1-4

Disassembly: This shows that the code runs lines 1-4 and that the 8th index of the array A is set to the value 200.

Registers: This shows the value stays the same because lines 1-4 don't change values or store anything.

Memory: This shows the address 0x555555558060 which has the value 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00.

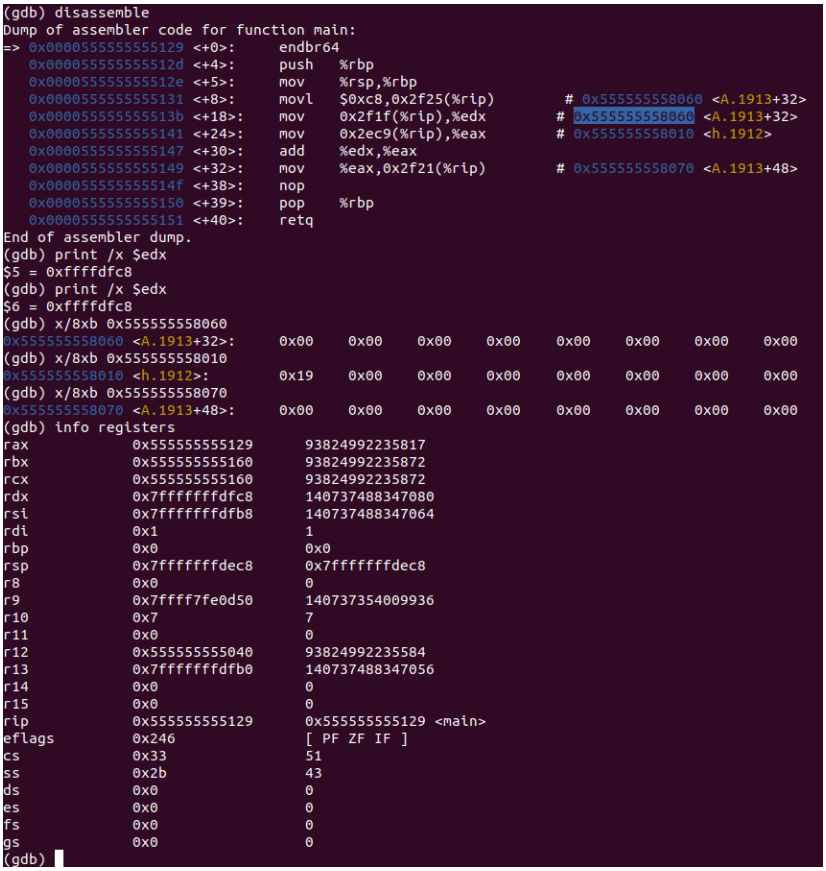


Figure 103: Shameem_2-3_2.c code in gdb before lines 5-6

Disassembly: This shows that the code runs lines 5-6 and that the values are moved into the registers so that addition can be performed which then the result of that addition is stored in memory.

Registers: This shows the value in each register.

The value of the 8th index of array A is stored in ex.

The value of h in memory is moved to eax.

The value in the register from addition is moved into the memory and stored into the address of the 12th index of the array A.

Memory: This shows the address and the values.

The address 0x555555558060 has the values 0xc8 0x00 0x00 0x00 0x00 0x00 0x00 0x00.

The address 0x555555558010 has the values 0x19 0x00 0x00 0x00 0x00 0x00 0x00 0x00.

The address 0x5555555581d0 has the values 0xe1 0x00 0x00 0x00 0x00 0x00 0x00 0x00

2-5_1.c

```
(gdb) list
1       void main() {
2               static int a = 0;
3               static int b = 0;
4               static int c = 0;
5               a = b + c;
6       }
```

Figure 104: Shameem_2-5_1.c code in gdb

The code has three static variables a, b, c and they all have the value of 0. Then the code adds b and c and stores that result in the variable a.

```
(gdb) disassemble
Dump of assembler code for function main:
=> 0x0000555555555129 <+0>:     endbr64
   0x000055555555512d <+4>:     push   %rbp
   0x000055555555512e <+5>:     mov    %rsp,%rbp
   0x0000555555555131 <+8>:     mov    0x2edd(%rip),%edx        # 0x555555558014 <b.1913>
   0x0000555555555137 <+14>:    mov    0x2edb(%rip),%eax        # 0x555555558018 <c.1914>
   0x000055555555513d <+20>:    add    %edx,%eax
   0x000055555555513f <+22>:    mov    %eax,0x2ed7(%rip)        # 0x55555555801c <a.1912>
   0x0000555555555145 <+28>:    nop
   0x0000555555555146 <+29>:    pop    %rbp
   0x0000555555555147 <+30>:    retq
End of assembler dump.
(gdb) print /x $rsp
$1 = 0x7fffffffdec8
(gdb) print /x $rbp
$2 = 0x0
(gdb) info registers
rax            0x555555555129      93824992235817
rbx            0x555555555150      93824992235856
rcx            0x555555555150      93824992235856
rdx            0x7fffffffdfc8      140737488347080
rsi            0x7fffffffdfb8      140737488347064
rdi            0x1                 1
rbp            0x0                 0x0
rsp            0x7fffffffdec8      0x7fffffffdec8
r8             0x0                 0
r9             0x7ffff7fe0d50      140737354009936
r10            0x7                 7
r11            0x0                 0
r12            0x555555555040      93824992235584
r13            0x7fffffffdfb0      140737488347056
r14            0x0                 0
r15            0x0                 0
rip            0x555555555129      0x555555555129 <main>
eflags         0x246               [ PF ZF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb)
```

Figure 105: Shameem_2-5_1.c code in gdb before line 1

Disassembly: This shows that the code runs line 1 and that stack pointer and base pointer are set.

Registers: This shows the value that's stored in each register which is the stack pointer at 0x7fffffffdf78 and the base pointer at 0x0.

```
(gdb) disassemble
Dump of assembler code for function main:
=> 0x0000555555555129 <+0>:     endbr64
   0x000055555555512d <+4>:     push   %rbp
   0x000055555555512e <+5>:     mov    %rsp,%rbp
   0x0000555555555131 <+8>:     mov    0x2edd(%rip),%edx        # 0x555555558014 <b.1913>
   0x0000555555555137 <+14>:    mov    0x2edb(%rip),%eax        # 0x555555558018 <c.1914>
   0x000055555555513d <+20>:    add    %edx,%eax
   0x000055555555513f <+22>:    mov    %eax,0x2ed7(%rip)        # 0x55555555801c <a.1912>
   0x0000555555555145 <+28>:    nop
   0x0000555555555146 <+29>:    pop    %rbp
   0x0000555555555147 <+30>:    retq
End of assembler dump.
(gdb) print /x $edx
$3 = 0xffffdfc8
(gdb) print /x $eax
$4 = 0x55555129
(gdb) x/8xb 0x555555558014
0x555555558014 <b.1913>:        0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
(gdb) x/8xb 0x555555558018
0x555555558018 <c.1914>:        0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
(gdb) x/8xb 0x55555555801c
0x55555555801c <a.1912>:        0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
(gdb) info registers
rax            0x555555555129      93824992235817
rbx            0x555555555150      93824992235856
rcx            0x555555555150      93824992235856
rdx            0x7fffffffdfc8      140737488347080
rsi            0x7fffffffdfb8      140737488347064
rdi            0x1                 1
rbp            0x0                 0x0
rsp            0x7fffffffdec8      0x7fffffffdec8
r8             0x0                 0
r9             0x7ffff7fe0d50      140737354009936
r10            0x7                 7
r11            0x0                 0
r12            0x555555555040      93824992235584
r13            0x7fffffffdfb0      140737488347056
r14            0x0                 0
r15            0x0                 0
rip            0x555555555129      0x555555555129 <main>
eflags         0x246               [ PF ZF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb)
```

Figure 106: Shameem_2-5_1.c code in gdb lines 1-5

Disassembly: This shows that the code runs lines 1-4.

Registers: This shows that a is stored in edx and b is stored in eax which allows addition to be performed and then stored in eax.

Memory: This window shows the address and its storeds values.
The address 0x555555558014 has the value 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00.
The address 0x555555558018 has the value 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00.
The address 0x55555555801C has the value 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00.

2-6_1.c

```
(gdb) list
1       void main() {
2               static int s0 = 9;
3               static int t1 = 0x3c00;
4               static int t2 = 0xdc0;
5               static int t3 = 0;
6               t3 = s0 << 4;
7               static int t0 = 0;
8               t0 = t1 & t2;
9               t0 = t1 | t2;
10              t0 = ~t1;
(gdb)
```

Figure 107: Shameem_2-6_1.c code in gdb

The code has four static variables s0, t0, t1, t2, t3 with the values 9, 0, 0x3C00, 0xDC0, 0 respectively. The code then uses these variables to then bit shift s0 to the left by 4 and then save that value in variable t3. Then the code does AND with t1 and t2 and saves it in t0. The code then does OR with t1 and t2 and saves it in t2. Lastly, the code does NOT t1 and saves the result in the variable t0.

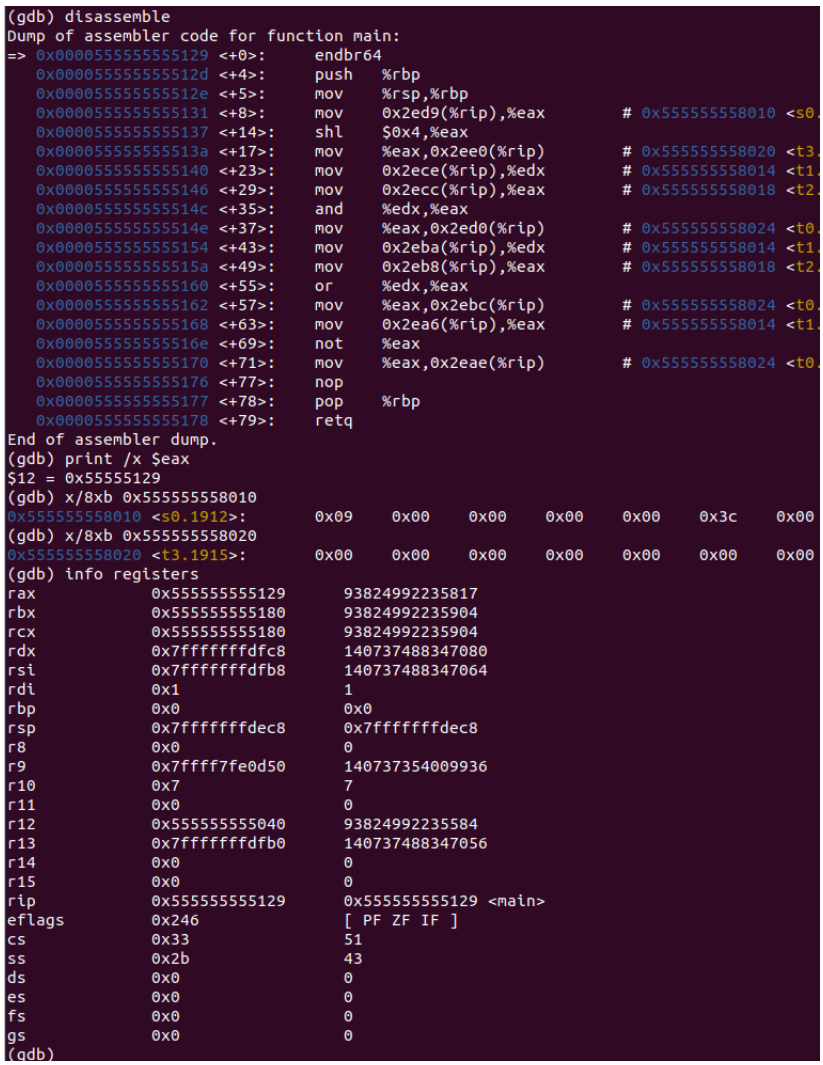
```
(gdb) disassemble
Dump of assembler code for function main:
=> 0x0000555555555129 <+0>:     endbr64
   0x000055555555512d <+4>:     push   %rbp
   0x000055555555512e <+5>:     mov    %rsp,%rbp
   0x0000555555555131 <+8>:     mov    0x2ed9(%rip),%eax        # 0x555555558010 <s0.1912>
   0x0000555555555137 <+14>:    shl    $0x4,%eax
   0x000055555555513a <+17>:    mov    %eax,0x2ee0(%rip)        # 0x555555558020 <t3.1915>
   0x0000555555555140 <+23>:    mov    0x2ece(%rip),%edx        # 0x555555558014 <t1.1913>
   0x0000555555555146 <+29>:    mov    0x2ecc(%rip),%eax        # 0x555555558018 <t2.1914>
   0x000055555555514c <+35>:    and    %edx,%eax
   0x000055555555514e <+37>:    mov    %eax,0x2ed0(%rip)        # 0x555555558024 <t0.1916>
   0x0000555555555154 <+43>:    mov    0x2eba(%rip),%edx        # 0x555555558014 <t1.1913>
   0x000055555555515a <+49>:    mov    0x2eb8(%rip),%eax        # 0x555555558018 <t2.1914>
   0x0000555555555160 <+55>:    or     %edx,%eax
   0x0000555555555162 <+57>:    mov    %eax,0x2ebc(%rip)        # 0x555555558024 <t0.1916>
   0x0000555555555168 <+63>:    mov    0x2ea6(%rip),%eax        # 0x555555558014 <t1.1913>
   0x000055555555516e <+69>:    not    %eax
   0x0000555555555170 <+71>:    mov    %eax,0x2eae(%rip)        # 0x555555558024 <t0.1916>
   0x0000555555555176 <+77>:    nop
   0x0000555555555177 <+78>:    pop    %rbp
   0x0000555555555178 <+79>:    retq
End of assembler dump.
(gdb) print /x $rsp
$5 = 0x7fffffffdec8
(gdb) print /x $rbp
$6 = 0x0
(gdb) info registers
rax            0x555555555129      93824992235817
rbx            0x555555555180      93824992235904
rcx            0x555555555180      93824992235904
rdx            0x7fffffffdfc8      140737488347080
rsi            0x7fffffffdfb8      140737488347064
rdi            0x1                 1
rbp            0x0                 0x0
rsp            0x7fffffffdec8      0x7fffffffdec8
r8             0x0                 0
r9             0x7ffff7fe0d50      140737354009936
r10            0x7                 7
r11            0x0                 0
r12            0x555555555040      93824992235584
r13            0x7fffffffdfb0      140737488347056
r14            0x0                 0
r15            0x0                 0
rip            0x555555555129      0x555555555129 <main>
eflags         0x246               [ PF ZF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb)
```

Figure 107: Shameem_2-6_1.c code in gdb before line 1

Disassembly: This shows that the code runs line 1 and that stack pointer and base pointer are set.

Registers: This shows the value that's stored in each register which is the stack pointer at 0x7fffffffdf78 and the base pointer at 0x0.

Figure 107: Shameem_2-6_1.c code in gdb line 1-6



Disassembly: This shows that the code runs lines 1-6 and that the value is moved to the register so can be shifted to the left by 4 and the result of that operation will be stored in memory.

Registers: This shows the value that's stored in each register.

The value of s0 in memory is moved to eax.

The s0 value that is left by 4 is stored in the same register.

The value in that register is moved to the memory and stored in the address of t0 where the new t0 overwrites the previous t0.

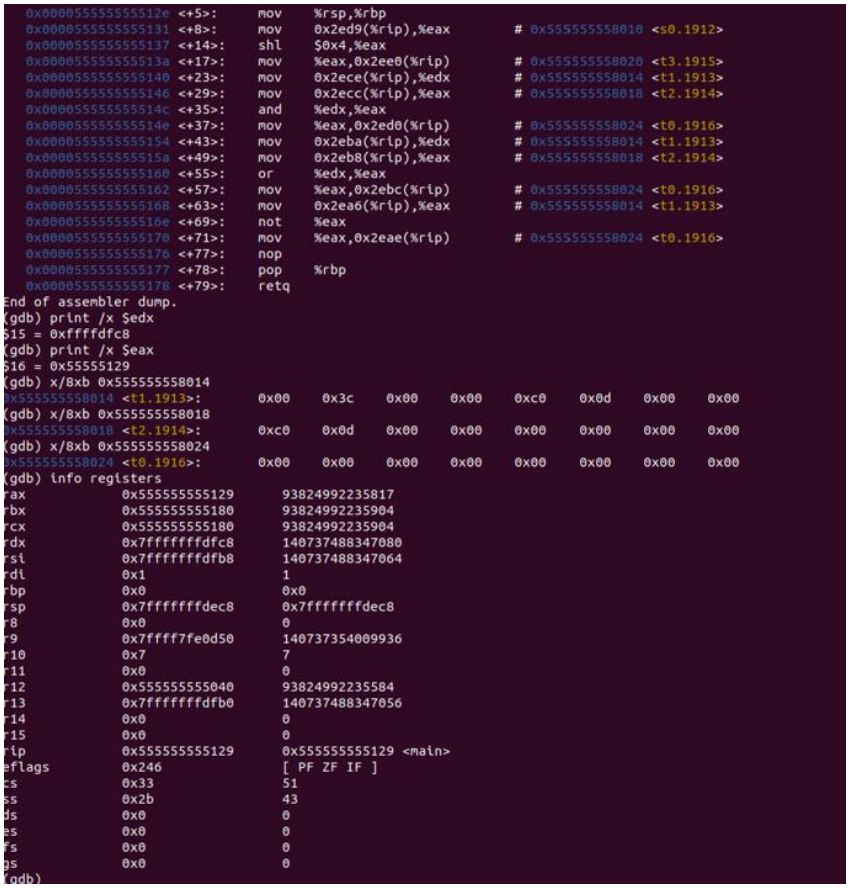
Memory: This shows the values stored in an address.

The address 0x555555558010 has the value 0x09 0x00 0x00 0x00 0x00 0x3c 0x00 0x00.

The address 0x555555558020 has the value 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00.

Figure 108: Shameem_2-6_1.c code in gdb line 7-8

Disassembly: This shows that the code runs lines 7-8 and that values are moved into



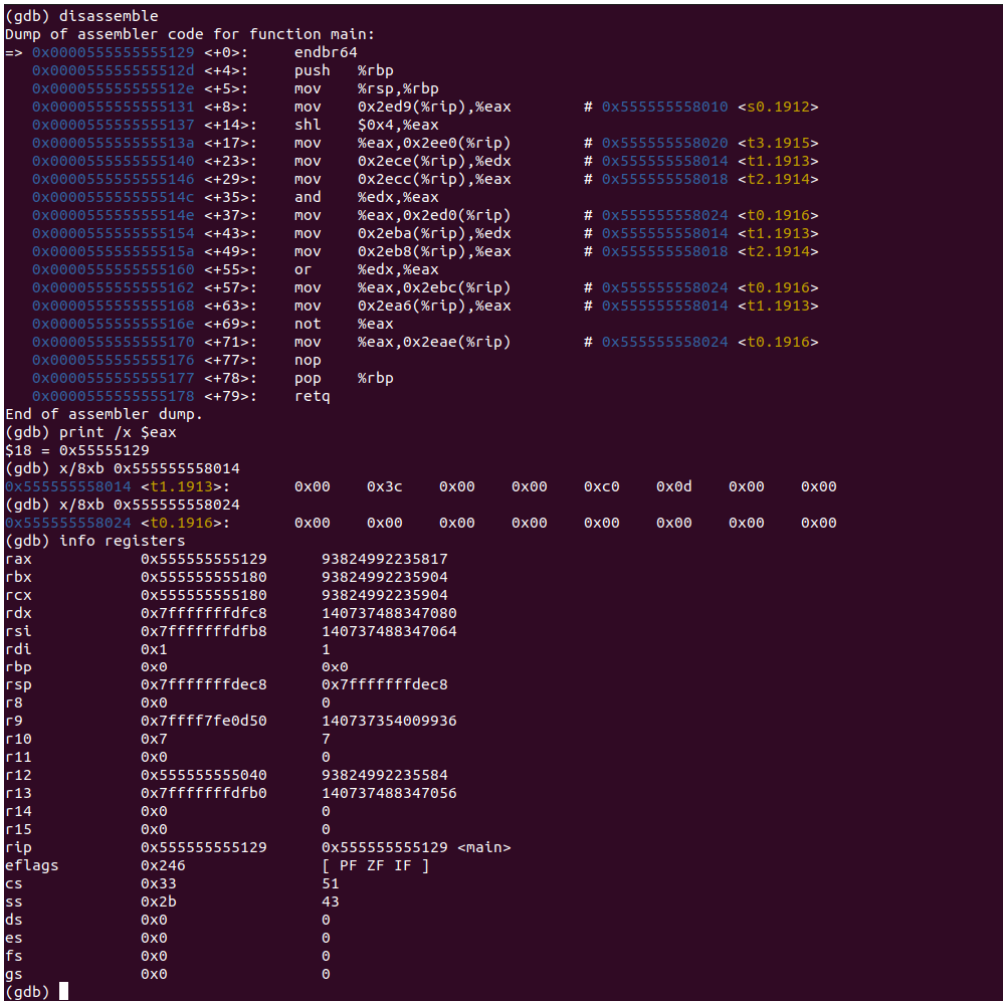
registers for bitwise comparisons and the result of the bitwise comparisons is stored in memory.

Registers: This shows the value that's stored in each register.

The value of t1 in memory is moved to edx.
The value of t2 in memory is moved to eax.
The bitwise comparison is then performed on the two registers edx and eax and the result is stored in eax.
The value in eax is then moved into the memory and stored in the address of t0 which
overwrites the value stored at t0.

Memory: This shows the values stored in an address.
The address 0x555555558014 has value 0x00 0x3c 0x00 0x00 0xc0 0x0d 0x00 0x00.
The address 0x555555558018 has value 0xc0 0x0d 0x00 0x00 0x00 0x00 0x00 0x00.
The address 0x555555558024 has value 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00.

Figure 109: Shameem_2-6_1.c code in gdb line 10-11



Disassembly: This shows that the code runs lines 10- 11 and that values are moved into registers for bitwise comparison and the result of the bitwise comparison will be stored in memory.

Registers: This shows the value that's stored in each register.

The value of t1 in memory is moved to edx.

The value of t2 in memory is moved to eax.

The bitwise comparison is then performed on the two registers edx and eax and the result is stored in eax.

The value in eax is then moved into the memory and stored in the address of t0 which

overwrites the value stored at t0.

Memory: This shows the values stored in an address.

The address 0x555555558014 has value 0x00 0x3c 0x00 0x00 0xc0 0x0d 0x00 0x00.

The address 0x555555558024 has value 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00.

2-7_1.c

```
(gdb) list
1       void main() {
2               static int a = 1;
3               static int b = 5;
4               while (a != b) {
5                       a += 1;
6               }
7       }
(gdb)
```

Figure 110: Shameem_2-7_1.c code in gdb

The code has three static variables a and b with the respective values 1 and 5. The code then goes into a while loop where for each loop the variable a is incremented by 1 and continues that process until a equals b. Once variable a equals variable b the function then stops.

```
(gdb) disassemble
Dump of assembler code for function main:
=> 0x0000555555555129 <+0>:     endbr64
   0x000055555555512d <+4>:     push   %rbp
   0x000055555555512e <+5>:     mov    %rsp,%rbp
   0x0000555555555131 <+8>:     jmp    0x555555555142 <main+25>
   0x0000555555555133 <+10>:    mov    0x2ed7(%rip),%eax        # 0x555555558010 <a.1912>
   0x0000555555555139 <+16>:    add    $0x1,%eax
   0x000055555555513c <+19>:    mov    %eax,0x2ece(%rip)        # 0x555555558010 <a.1912>
   0x0000555555555142 <+25>:    mov    0x2ec8(%rip),%edx        # 0x555555558010 <a.1912>
   0x0000555555555148 <+31>:    mov    0x2ec6(%rip),%eax        # 0x555555558014 <b.1913>
   0x000055555555514e <+37>:    cmp    %eax,%edx
   0x0000555555555150 <+39>:    jne    0x555555555133 <main+10>
   0x0000555555555152 <+41>:    nop
   0x0000555555555153 <+42>:    nop
   0x0000555555555154 <+43>:    pop    %rbp
   0x0000555555555155 <+44>:    retq
End of assembler dump.
(gdb) print /x $rsp
$3 = 0x7fffffffdec8
(gdb) print /x $rbp
$4 = 0x0
(gdb) info registers
rax            0x555555555129      93824992235817
rbx            0x555555555160      93824992235872
rcx            0x555555555160      93824992235872
rdx            0x7fffffffdfc8      140737488347080
rsi            0x7fffffffdfb8      140737488347064
rdi            0x1                 1
rbp            0x0                 0x0
rsp            0x7fffffffdec8      0x7fffffffdec8
r8             0x0                 0
r9             0x7ffff7fe0d50      140737354009936
r10            0x7                 7
r11            0x0                 0
r12            0x555555555040      93824992235584
r13            0x7fffffffdfb0      140737488347056
r14            0x0                 0
r15            0x0                 0
rip            0x555555555129      0x555555555129 <main>
eflags         0x246               [ PF ZF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb)
```

Figure 111: Shameem_2-7_1.c code in gdb before line 1

Disassembly: This shows that the code runs line 1 and that stack pointer and base pointer are set.

Registers: This shows the value that's stored in each register which is the stack pointer at 0x7fffffffdf78 and the base pointer at 0x0.

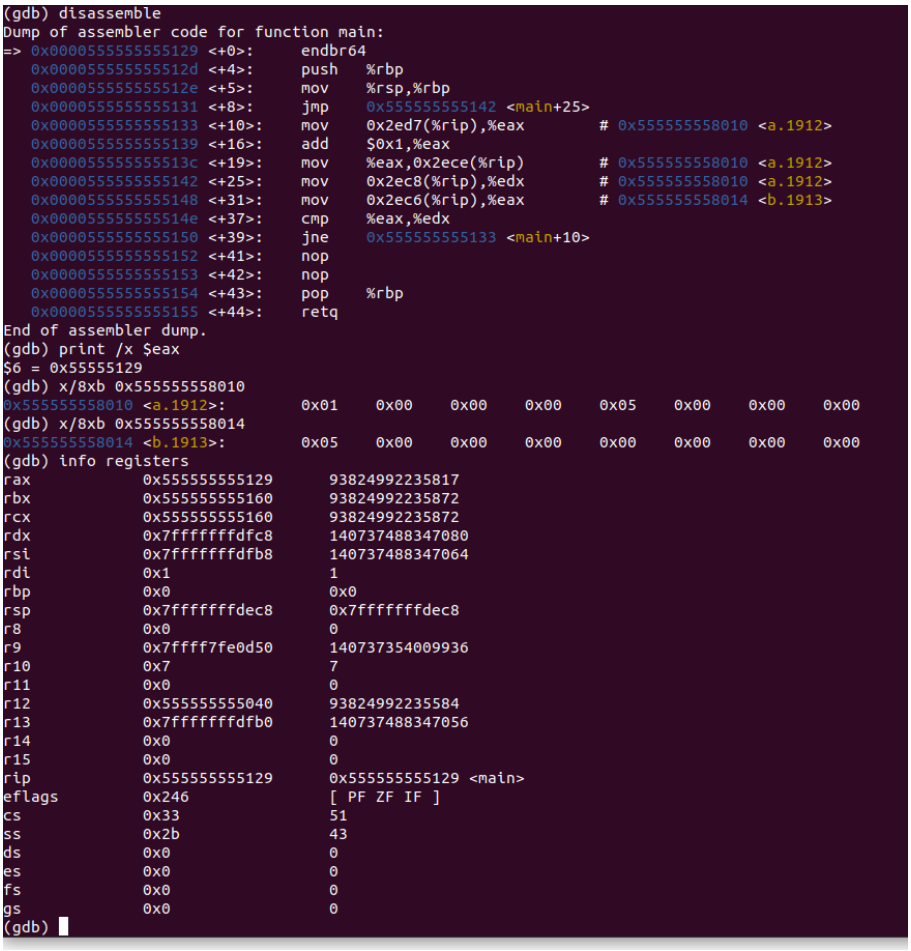


Figure 112: Shameem_2-6_1.c code in gdb line 1-6

Disassembly: This shows that the code runs lines 1-6 and that the value is moved to the register so can be shifted to the left by 4 and the result of that operation will be stored in memory.

Registers: This shows the value that's stored in each register.

The value of a in memory is moved to eax, so that it can compare with the value b.

Then the value of eax is incremented if the value of a does not equal b.

Memory: This shows the values stored in an address.

The address 0x555555558010 has the value 0x01 0x00 0x00 0x00 0x05 0x00 0x00 0x00.

The address 0x555555558014 has the value 0x05 0x00 0x00 0x00 0x00 0x00 0x00 0x00.

natural_generator.c

```
(gdb) list
1       /* static.c */
2       #include <stdio.h>
3       int natural_generator()
4       {
5               int a = 1;
6               static int b = -1;
7               b += 1;
8               return a + b;
9       }
10
(gdb)
```

Figure 113: Shameem_natural_generator.c code in gdb

The function natural_generator() has an variable integer a and a static integer b with values 1 and -1 respectively. The function natural_generator() increments the static variable b every time its called and then returns the value of the variable a plus the value of the static variable b.

```
Dump of assembler code for function main:
=> 0x0000555555555174 <+0>:     endbr64
   0x0000555555555178 <+4>:     push   %rbp
   0x0000555555555179 <+5>:     mov    %rsp,%rbp
   0x000055555555517c <+8>:     mov    $0x0,%eax
   0x0000555555555181 <+13>:    callq  0x555555555149 <natural_generator>
   0x0000555555555186 <+18>:    mov    %eax,%esi
   0x0000555555555188 <+20>:    lea    0xe75(%rip),%rdi        # 0x555555556004
   0x000055555555518f <+27>:    mov    $0x0,%eax
   0x0000555555555194 <+32>:    callq  0x555555555050 <printf@plt>
   0x0000555555555199 <+37>:    mov    $0x0,%eax
   0x000055555555519e <+42>:    callq  0x555555555149 <natural_generator>
   0x00005555555551a3 <+47>:    mov    %eax,%esi
   0x00005555555551a5 <+49>:    lea    0xe58(%rip),%rdi        # 0x555555556004
   0x00005555555551ac <+56>:    mov    $0x0,%eax
   0x00005555555551b1 <+61>:    callq  0x555555555050 <printf@plt>
   0x00005555555551b6 <+66>:    mov    $0x0,%eax
   0x00005555555551bb <+71>:    callq  0x555555555149 <natural_generator>
   0x00005555555551c0 <+76>:    mov    %eax,%esi
   0x00005555555551c2 <+78>:    lea    0xe3b(%rip),%rdi        # 0x555555556004
   0x00005555555551c9 <+85>:    mov    $0x0,%eax
   0x00005555555551ce <+90>:    callq  0x555555555050 <printf@plt>
   0x00005555555551d3 <+95>:    mov    $0x0,%eax
   0x00005555555551d8 <+100>:   pop    %rbp
   0x00005555555551d9 <+101>:   retq
End of assembler dump.
(gdb) print /x $rsp
$1 = 0x7fffffffdeb8
(gdb) print /x $rbp
$2 = 0x0
(gdb) info registers
rax            0x555555555174      93824992235892
rbx            0x5555555551e0      93824992236000
rcx            0x5555555551e0      93824992236000
rdx            0x7fffffffdfb8      140737488347064
rsi            0x7fffffffdfa8      140737488347048
rdi            0x1                 1
rbp            0x0                 0x0
rsp            0x7fffffffdeb8      0x7fffffffdeb8
r8             0x0                 0
r9             0x7ffff7fe0d50      140737354009936
r10            0x7                 7
r11            0x0                 0
r12            0x555555555060      93824992235616
r13            0x7fffffffdfa0      140737488347040
r14            0x0                 0
r15            0x0                 0
rip            0x555555555174      0x555555555174 <main>
eflags         0x246               [ PF ZF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
```

Figure 114: Shameem_natural_generator.c code in gdb

Disassembly: This shows that the code runs line 1 and that stack pointer and base pointer are set.

Registers: This shows the value that's stored in each register which is the stack pointer at 0x7fffffffdf78 and the base pointer at 0x0.

```
Dump of assembler code for function natural_generator:
=> 0x0000555555555149 <+0>:     endbr64
   0x000055555555514d <+4>:     push   %rbp
   0x000055555555514e <+5>:     mov    %rsp,%rbp
   0x0000555555555151 <+8>:     movl   $0x1,-0x4(%rbp)
   0x0000555555555158 <+15>:    mov    0x2eb2(%rip),%eax        # 0x555555558010 <b.2316>
   0x000055555555515e <+21>:    add    $0x1,%eax
   0x0000555555555161 <+24>:    mov    %eax,0x2ea9(%rip)        # 0x555555558010 <b.2316>
   0x0000555555555167 <+30>:    mov    0x2ea3(%rip),%edx        # 0x555555558010 <b.2316>
   0x000055555555516d <+36>:    mov    -0x4(%rbp),%eax
   0x0000555555555170 <+39>:    add    %edx,%eax
   0x0000555555555172 <+41>:    pop    %rbp
   0x0000555555555173 <+42>:    retq
End of assembler dump.
```

Figure 115: Shameem_natural_generator.c code in gdb

Disassembly: This shows that the code runs line 1 and that stack pointer and base pointer are set.

Registers: This shows the value that's stored in each register which is the stack pointer at 0x7fffffffdf78 and the base pointer at 0x0.

```
(gdb) x/8xb 0x555555558010
0x555555558010 <b.2316>:        0xff    0xff    0xff    0xff    0x00    0x00    0x00    0x00
(gdb)
(gdb) next
5               int a = 1;
(gdb) next
7               b += 1;
(gdb) next
8               return a + b;
(gdb) next
9       }
(gdb) x/8xb 0x555555558010
0x555555558010 <b.2316>:        0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
(gdb)
(gdb) x/8xb 0x555555558010
0x555555558010 <b.2316>:        0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
(gdb) next
1
main () at Shameem_natural_generator.c:14
14              printf("%d\n", natural_generator());
(gdb) next

Breakpoint 2, natural_generator () at Shameem_natural_generator.c:4
4       {
(gdb) next
5               int a = 1;
(gdb) next
7               b += 1;
(gdb) next
8               return a + b;
(gdb) next
9       }
(gdb) x/8xb 0x555555558010
0x555555558010 <b.2316>:        0x01    0x00    0x00    0x00    0x00    0x00    0x00    0x00
(gdb)
```

Figure 116: Shameem_natural_generator.c code in gdb

Memory: The address 0x555555558010 has the value 0xff 0xff 0xff 0xff 0x00 0x00 0x00 0x00. The address 0x555555558010 has the value 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00. The address 0x555555558010 has the value 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00.

myadd.c

```
(gdb) list
1       static int a = 1;
2       static int b = 2;
3       static int c = 3;
4
5       int myAdd(int a, int b)
6       {
7               int c = a + b;
8               return c;
9       }
10
(gdb)
```

Figure 117: Shameem_myadd.c code in gdb

The code has three static variables a, b and c whose values are 1, 2, 3 respectively. The function myAdd() takes variables a and b and adds them together then stores that result into the variable c and returns. The void main() next sets c equal to the value of the function myAdd() and then exits the program.

```
(gdb) disassemble
Dump of assembler code for function main:
=> 0x0000555555555147 <+0>:     endbr64
   0x000055555555514b <+4>:     push   %rbp
   0x000055555555514c <+5>:     mov    %rsp,%rbp
   0x000055555555514f <+8>:     mov    0x2ebf(%rip),%edx        # 0x555555558014 <b>
   0x0000555555555155 <+14>:    mov    0x2eb5(%rip),%eax        # 0x555555558010 <a>
   0x000055555555515b <+20>:    mov    %edx,%esi
   0x000055555555515d <+22>:    mov    %eax,%edi
   0x000055555555515f <+24>:    callq  0x555555555129 <myAdd>
   0x0000555555555164 <+29>:    mov    %eax,0x2eae(%rip)        # 0x555555558018 <c>
   0x000055555555516a <+35>:    nop
   0x000055555555516b <+36>:    pop    %rbp
   0x000055555555516c <+37>:    retq
End of assembler dump.
(gdb) print /x $rsp
$1 = 0x7fffffffdec8
(gdb) print /x $rbp
$2 = 0x0
(gdb) info registers
rax            0x555555555147      93824992235847
rbx            0x555555555170      93824992235888
rcx            0x555555555170      93824992235888
rdx            0x7fffffffdfc8      140737488347080
rsi            0x7fffffffdfb8      140737488347064
rdi            0x1                 1
rbp            0x0                 0x0
rsp            0x7fffffffdec8      0x7fffffffdec8
r8             0x0                 0
r9             0x7ffff7fe0d50      140737354009936
r10            0x7                 7
r11            0x0                 0
r12            0x555555555040      93824992235584
r13            0x7fffffffdfb0      140737488347056
r14            0x0                 0
r15            0x0                 0
rip            0x555555555147      0x555555555147 <main>
eflags         0x246               [ PF ZF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
```

Figure 118: Shameem_myadd.c code in gdb

Disassembly: This shows that the code runs line 1 and that stack pointer and base pointer are set.

Registers: This shows the value that's stored in each register which is the stack pointer at 0x7fffffffdf78 and the base pointer at 0x0.

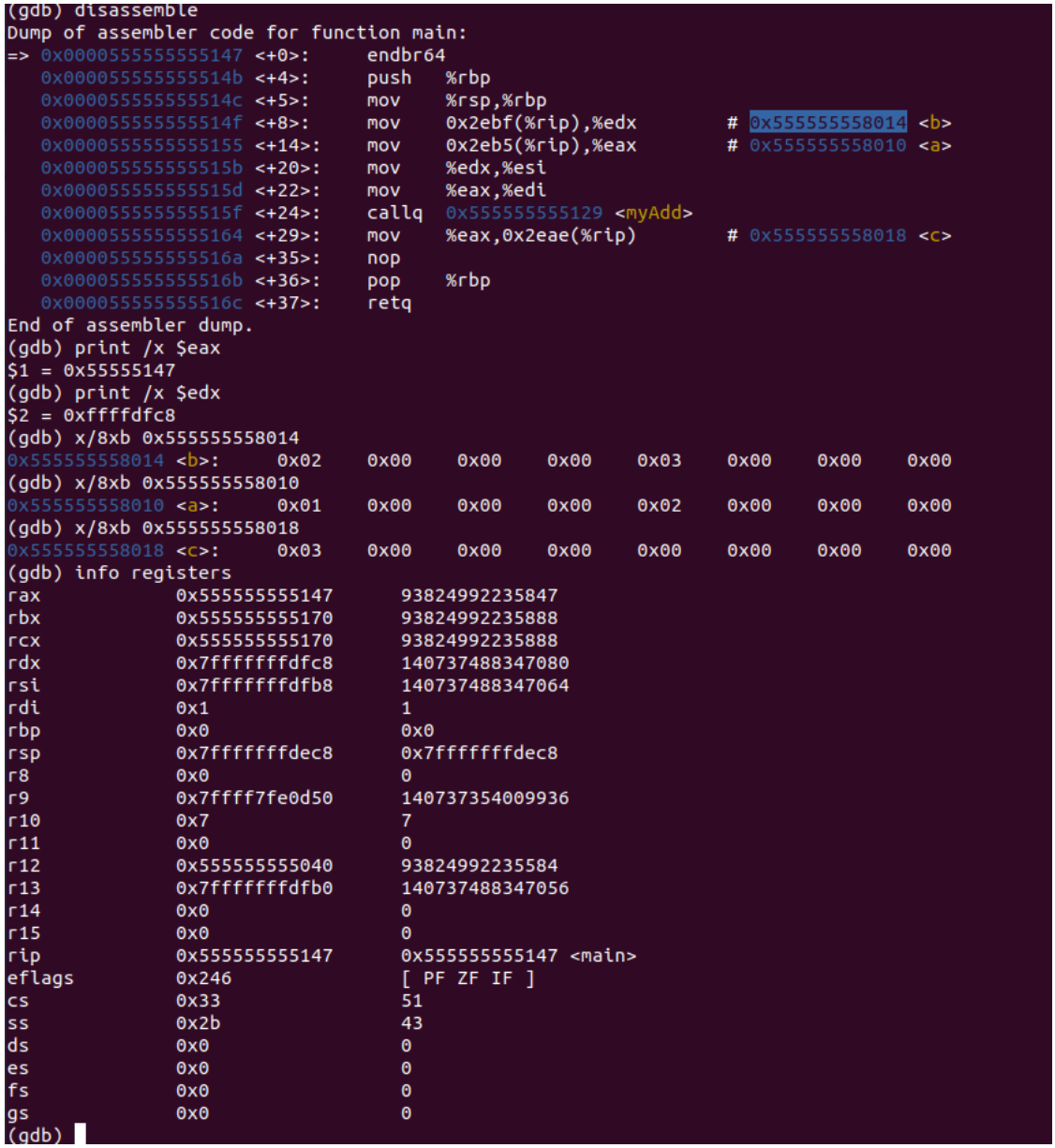


Figure 119: Shameem_myadd.c code in gdb

Memory: The address 0x555555558014 has the value 0x02 0x00 0x00 0x00 0x03 0x00 0x00 0x00. The address 0x555555558010 has the value 0x01 0x00 0x00 0x00 0x02 0x00 0x00 0x00. The address 0x555555558018 has the value 0x03 0x00 0x00 0x00 0x00 0x00 0x00 0x00

Explanation

MIPS

In the MIPS part we simulated the code using the MARS simulator to disassemble and analyze how the code works. In the MARS simulator we look at the registers, memory and the generated assembly instructions to understand the program better. One thing that was noticed is how the data was being stored during runtime. Also we know MIPS uses the Big-endian notation where the most significant byte is stored in a smaller memory address than at larger address. Furthermore, a select few of registers were seemingly common. The common registers are the $pc register which points to the address of the next instruction, the $fp register which points to the bottom of the stack and $sp which points to the top of the stack. In fact, the $ra register stored the return address after the function. Another detail is that the MIPS instructions are either fixed or fixed sized specifically 32 bits.

INTEL X32 ISA Windows 32-bit

In the Intel 32-bit Windows Visual Studio we simulated the code running on an there are a lot of differences between the INTEL compiler vs the MIPS processor compiler. One example is that the data is stored in LITTLE endian notation, which means that the most significant byte is stored in a larger address than in a smaller address, which is the opposite of MIPS. Furthermore, the size of the instructions is dynamic, not fixed. Not everything is different though, MIPS data and data in INTEL is stored in the same way.

X86_64 ISA Linux 64-bit

In Linux 64-bit we simulated the code running on a LINUX 64-bit compiler using GCC and GDB. Similarly, to the Intel 32-bit compiler and MIPS, the data is stored in the same way. Another similarity is Linux like in MIPS and Intel, the static variables are

initialized before the program is rann and then is stored in memory. One difference that is apparent is how the instructions look compared to MIPS and Intel. The Linux 64-bit compiler's look shows the source and destination operands very differently than MIPS and Intel. The destination operand in Linux is to the right and the source operand is to the left, which is very different from INTEL and MIPS because the source is to the right and the destination is on the left.

Conclusion

In conclusion, this was a great assignment to learn and test ourselves about the comparisons that we see on different compilers and systems. In our comparison we saw that Intel and Linux share the most similarities compared to MIPS. The assignment helped a lot to learn about dealing with assembly generated code for each platform and how different platforms and scompilers handles data and how it can be used to debug in the respective platforms.