CS 342000 | CS343000
Instructor: Professor Izidor Gertner
Spring 2022
Azwad Shameem, 4/25/2022
Take Home Test 2

Please hand write and sign statements affirming that you will not cheat here and in your submission:

"I will neither give nor receive unauthorized assistance on this LAB. I will use only one computing device to perform this LAB".

I will neither give nor recieve unathorized assistance on this exam. I will only use one computing device to perform this test
Azwad Shameem

Start Time and Date: 8:00 AM 04/24/22
End Time and Date: 8:00 PM 04/24/22

Table of Contents

Objective

The objective of this Take Home Test was to test, understand and demonstrate the recursive calls and stack frames in the three different architecture MARS Simulator, Visual Studio and Linux GDB and GCC. The Take Home Test objective includes testing and explaining the MIPS instructions on MARS Simulator, explaining the registers and memory on Visual Studio and explaining the debugger on Linux GDB and GCC. Furthermore, in this Take Home Test, we will be using the recursive version of the Eucledean algorithm for GCD(a, b) where integers $a > 0$ and $b > 0$.

Solutions:

MIPS:

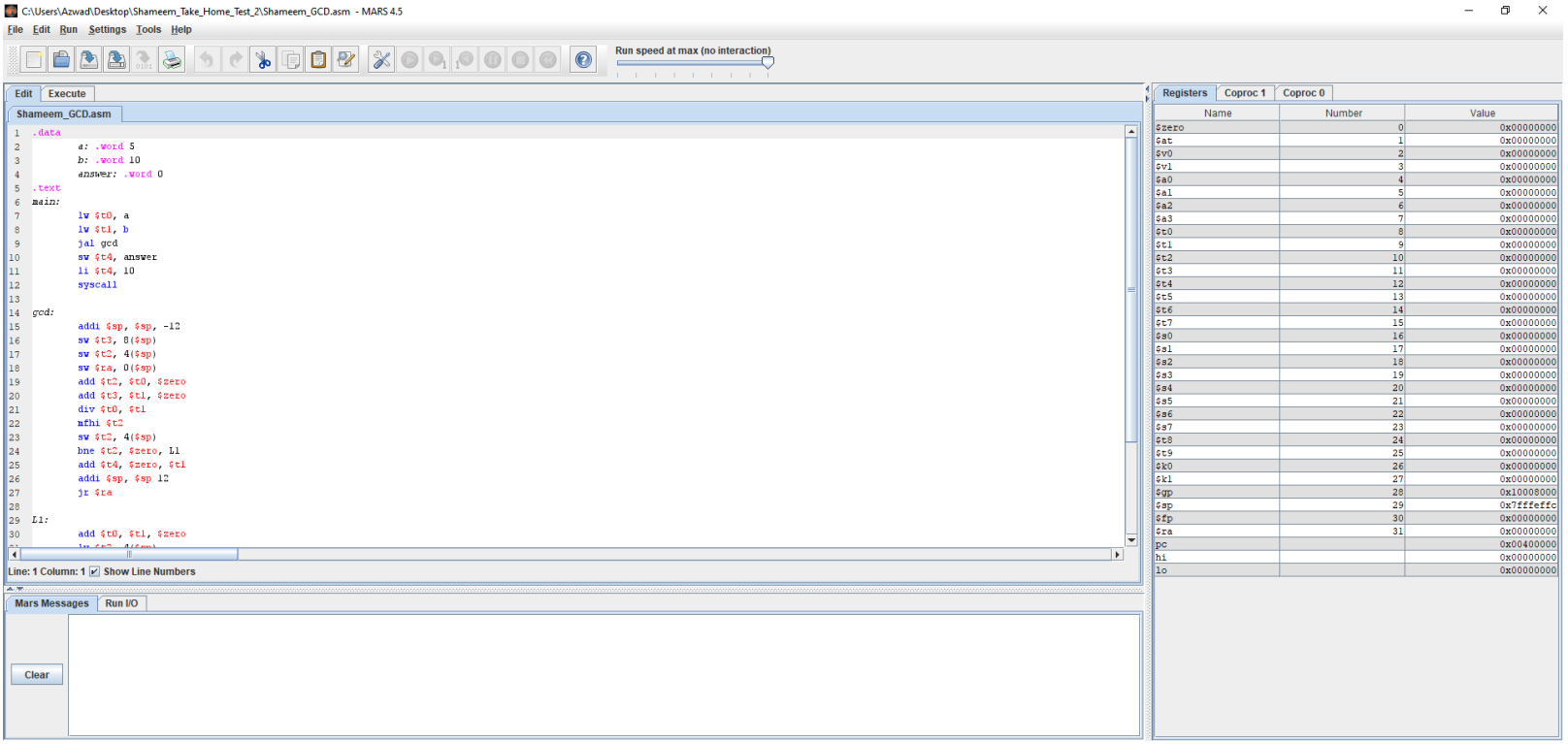

Figure 1: Shameem_GCD.asm code in MARS

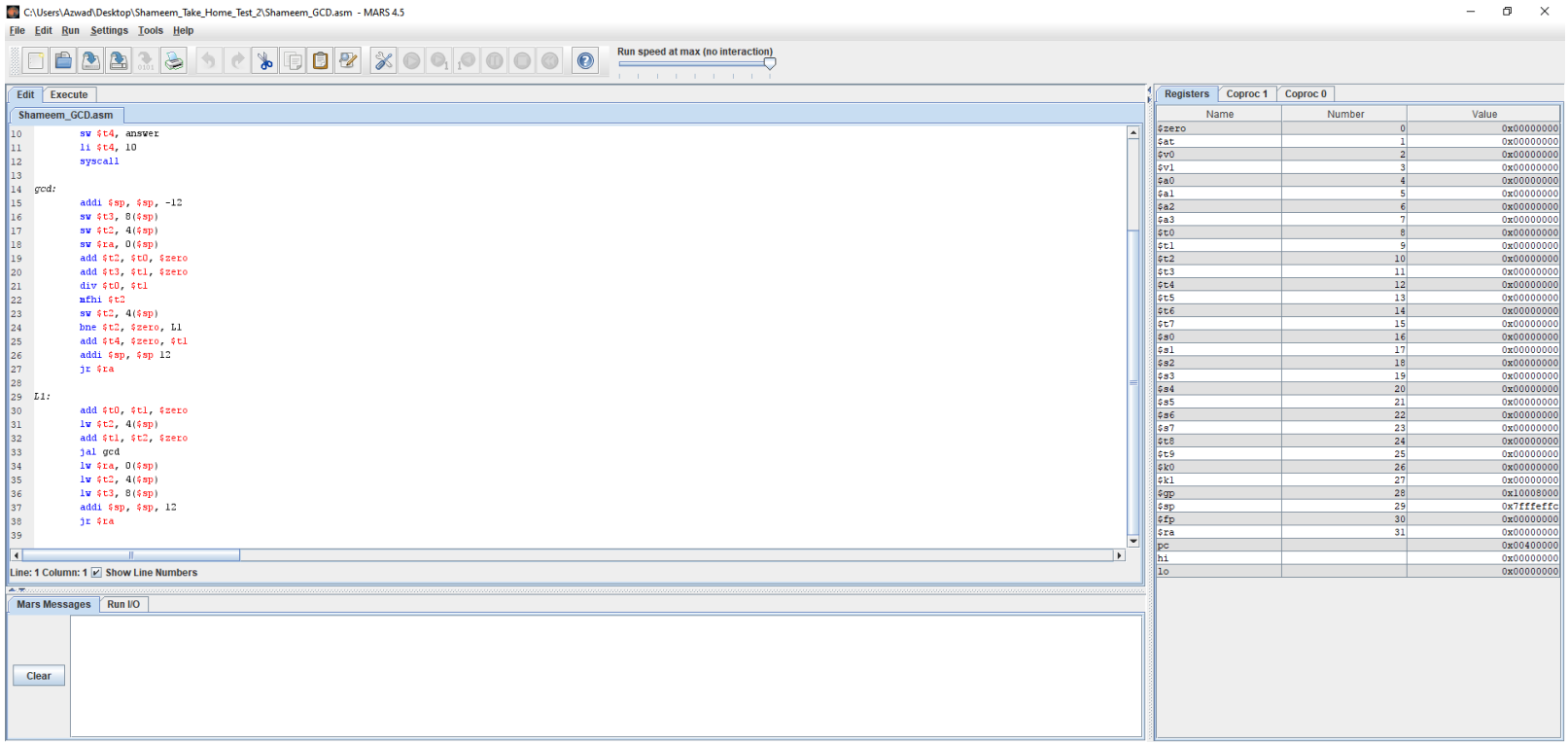

Figure 2: Shameem_GCD.asm code in MARS continued

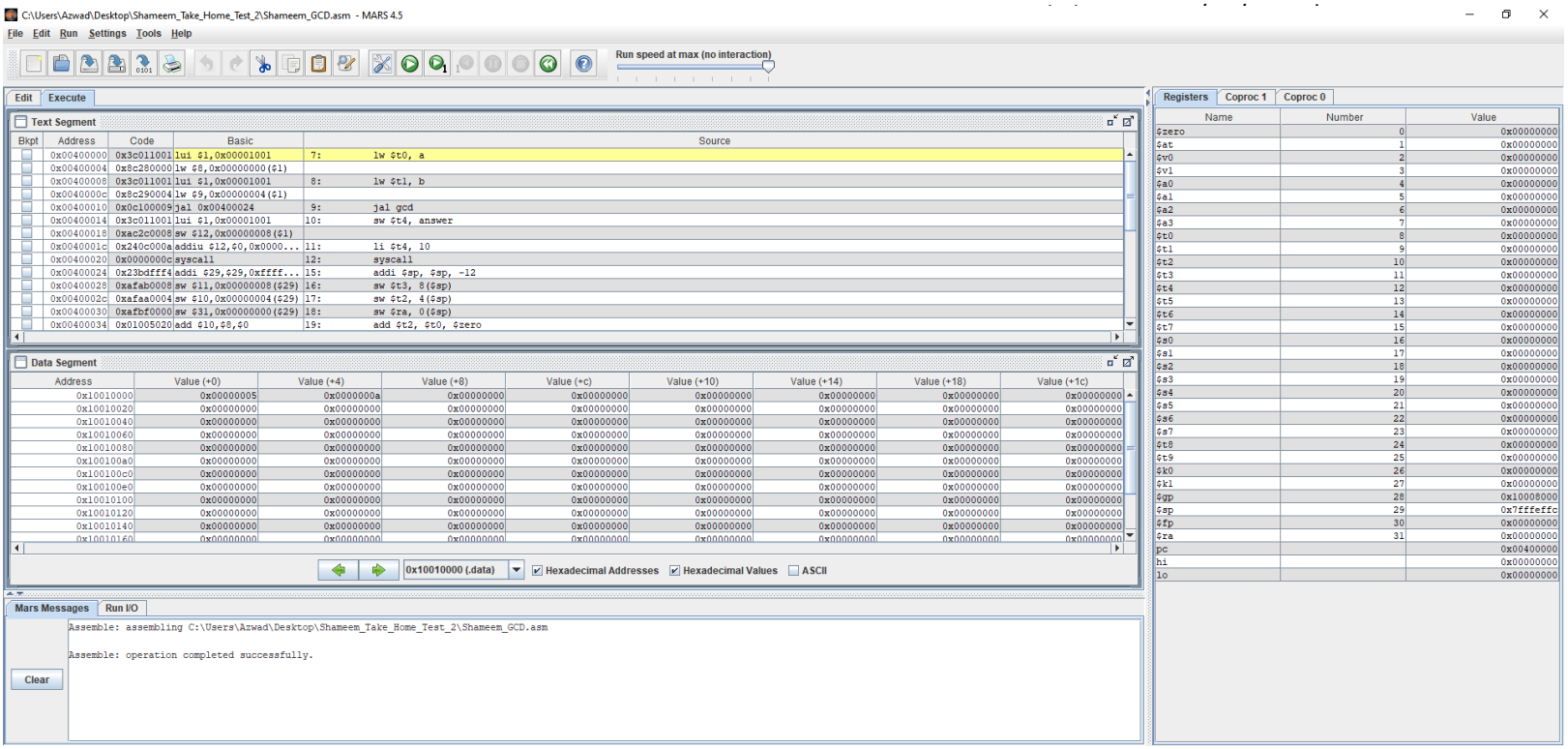

Figure 3: Shameem_GCD.asm code assembled in MARS (Lines 1-6)

Text Segment: This window shows that the code starts from line 7 because the previous lines don't use registers.

Registers: This window shows that the stack pointer is 0x7FFFEFFC.

Data Segment: This window shows the variables stored in the addresses.

Variable a is stored at address 0x10010000, with a value of 0x00000005.

Variable b is stored at address 0x10010000, with a value of 0x0000000a.

Variable answer is stored at address 0x10010000, with a value of 0x00000000.

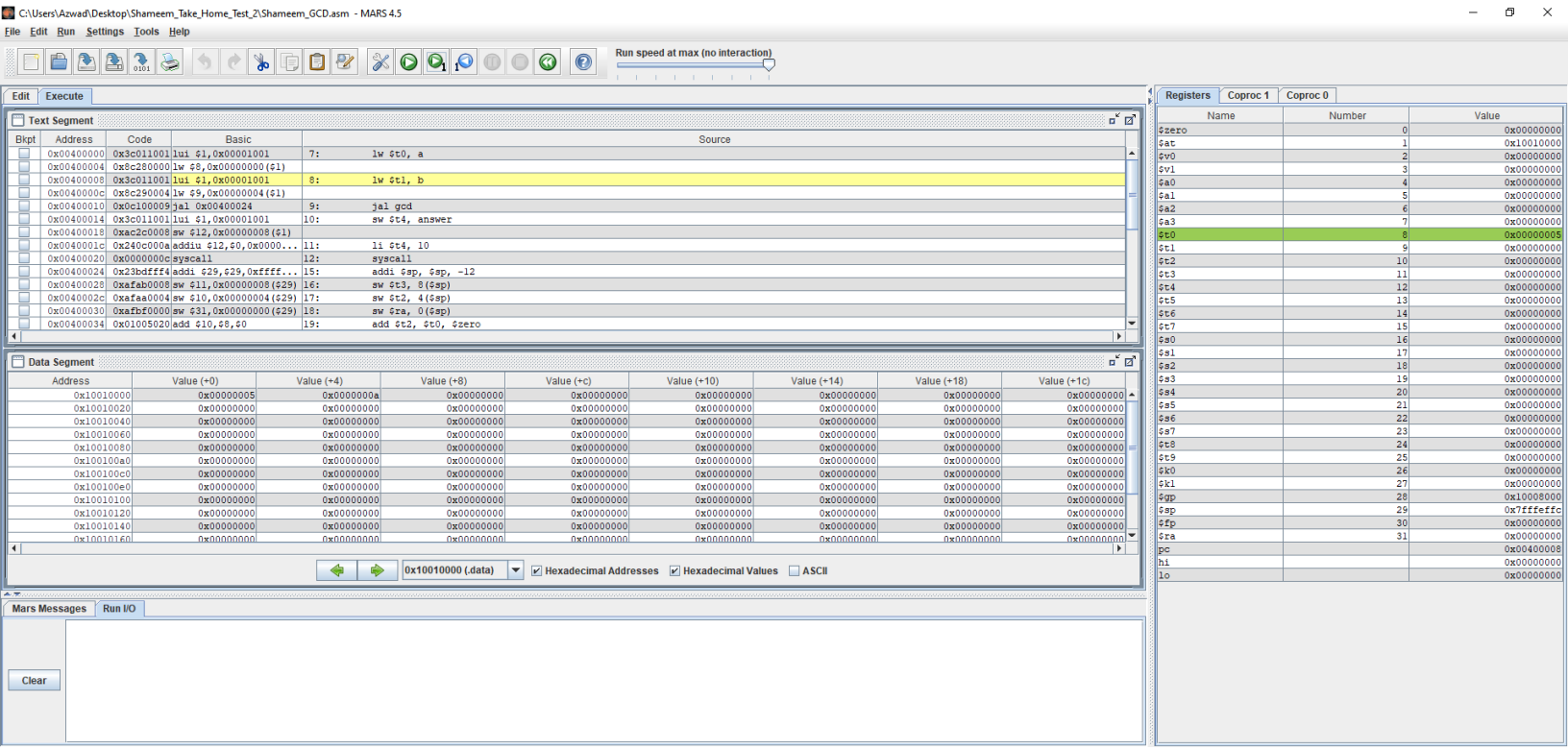

Figure 4: Shameem_GCD.asm code assembled in MARS (Line 7)

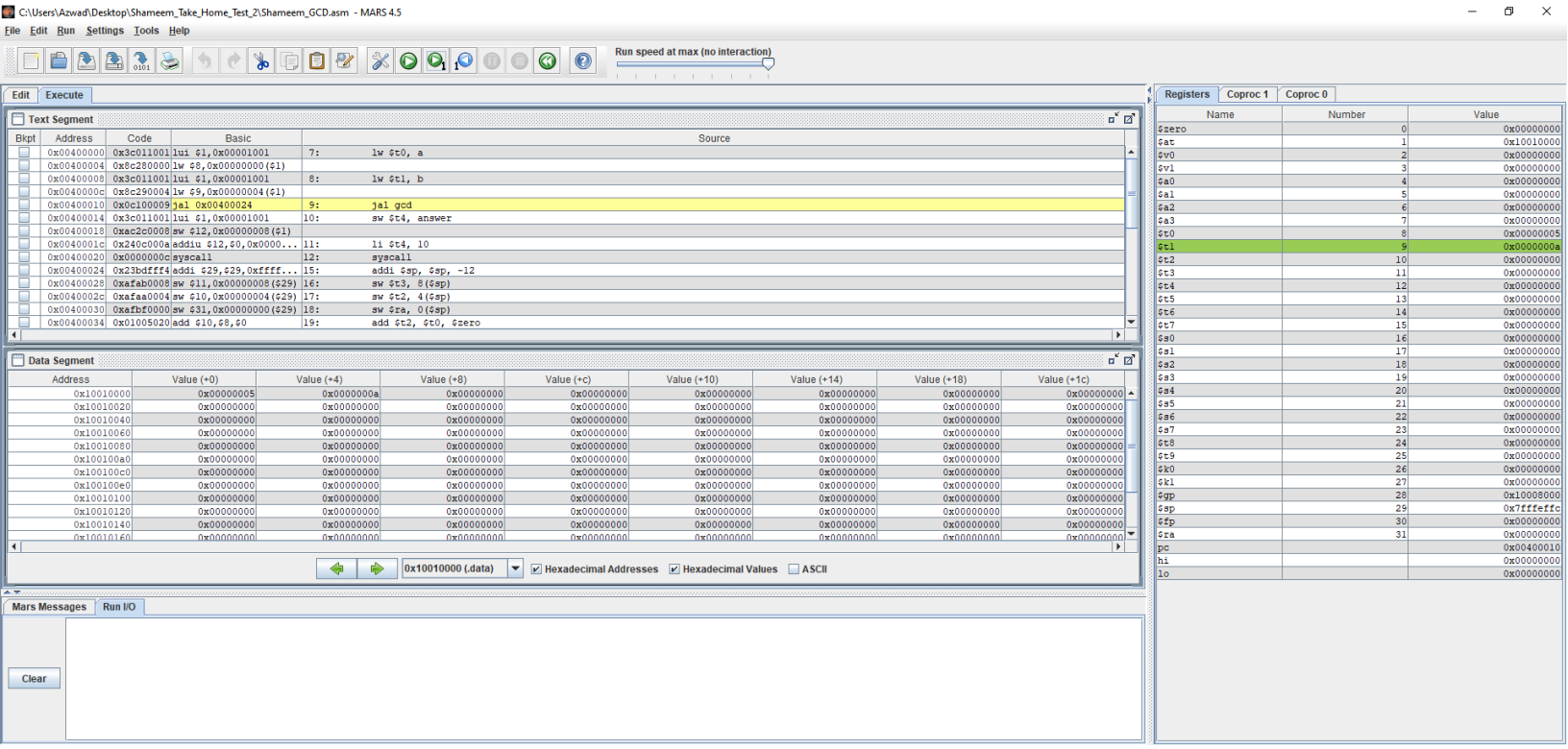

Figure 5: Shameem_GCD.asm code assembled in MARS (Line 8)

For Figure 4 and Figure 5.

Text Segment: This window shows that MARS has ran lines 7 and 8.

Registers: This window shows that the stack pointer is 0x7FFFEFFC. This window also shows that the variables have been loaded into the registers at $t0 and $t1. In the registers window in figure 4, $t0 is highlighted with the value of 0x00000005 because variable a's value has recently been put into the registers. In the registers window in figure 5, $t1 is highlighted with the value of 0x0000000a because variable b's value has recently been put into the registers window.

Data Segment: This window shows the variables stored in the addresses.

Variable a is stored at address 0x10010000, with a value of 0x00000005.

Variable b is stored at address 0x10010000, with a value of 0x0000000a.

Variable answer is stored at address 0x10010000, with a value of 0x00000000.

There has been no change in the Data Segment window figure 5 and figure 6, remains the same as figure 6.

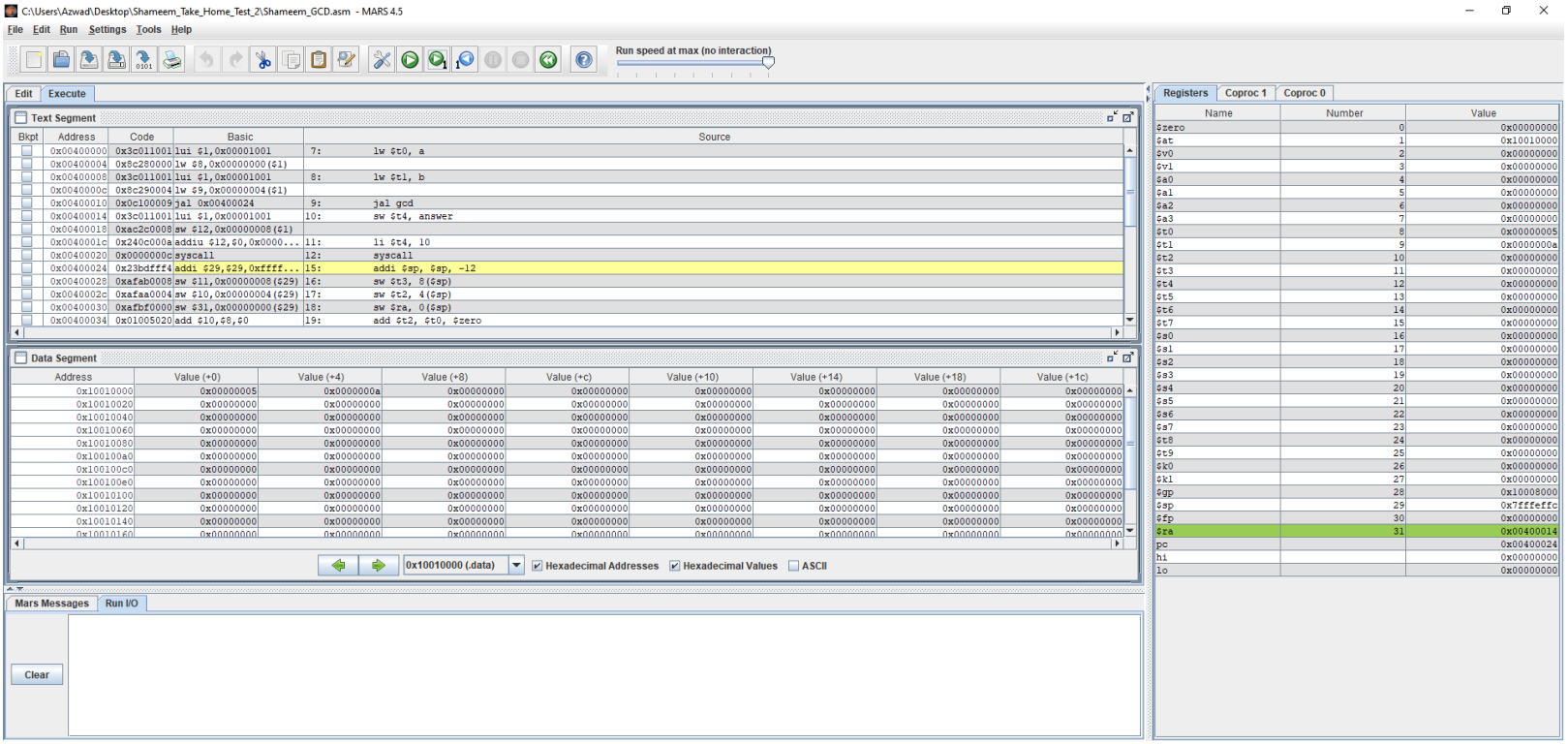

Figure 6: Shameem_GCD.asm code assembled in MARS (Line 9)

Text Segment: This window shows that MARS has ran line 9 and jumped to line 15 where the gcd function starts.

Registers: This window shows that the $ra register has been updated to 0x00400014. The other values in the registers window remain the same from figure 4 and figure 5.

Data Segment: This window shows the variables stored in the addresses.

Variable a is stored at address 0x10010000, with a value of 0x00000005.

Variable b is stored at address 0x10010000, with a value of 0x0000000a.

Variable answer is stored at address 0x10010000, with a value of 0x00000000.

There has been no change in the Data Segment window figure 5 and figure 6, remains the same as figure 6.

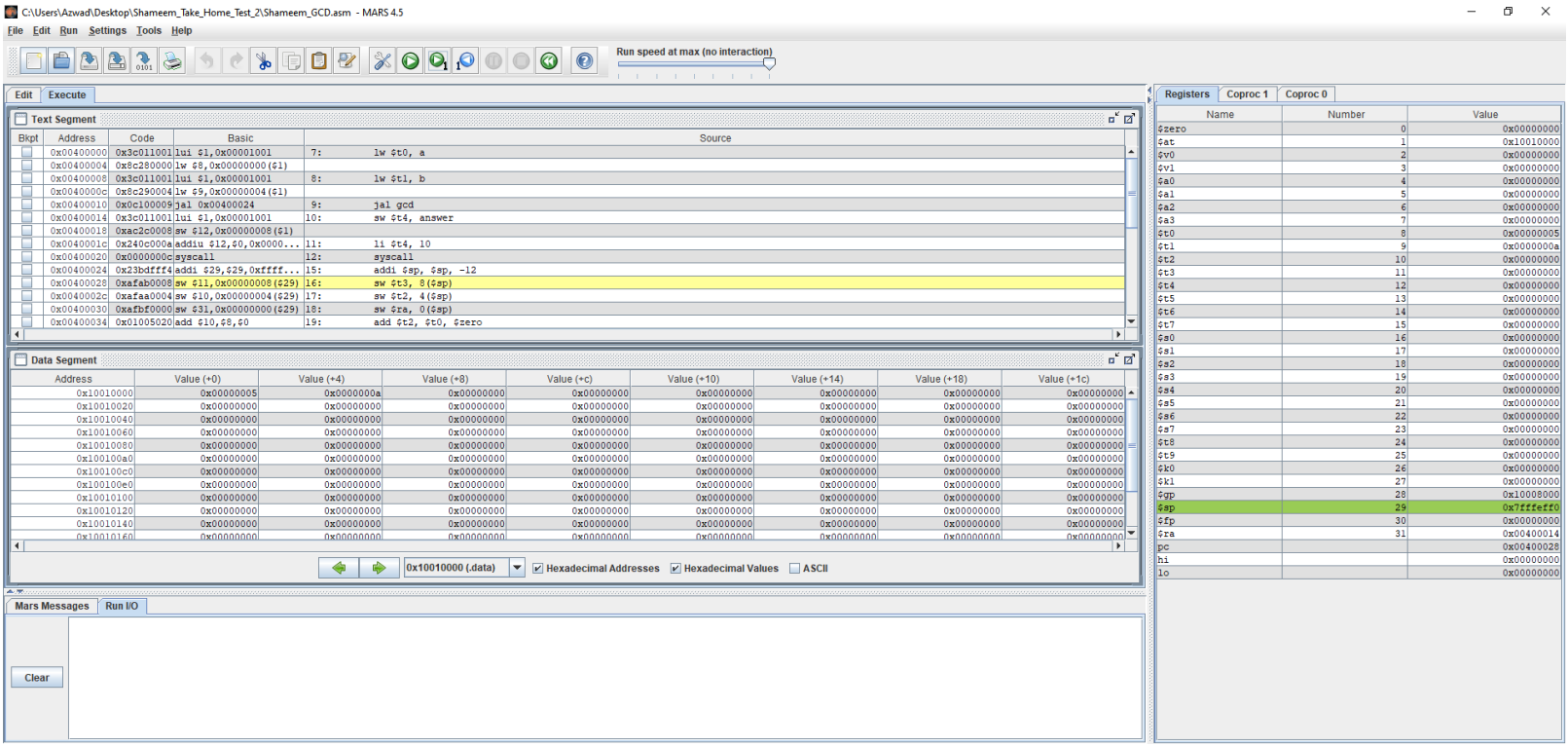

Figure 7: Shameem_GCD.asm code assembled in MARS (Line 15)

Text Segment: This window shows that MARS has ran line 15.

Registers: This window shows that the stack pointer, $sp register has been decreased by the value of 12. The other values in the registers window remain the same from figure 6.

Data Segment: This window shows the variables stored in the addresses.

Variable a is stored at address 0x10010000, with a value of 0x00000005.

Variable b is stored at address 0x10010004, with a value of 0x0000000a.

Variable answer is stored at address 0x10010008, with a value of 0x00000000.

There has been no change in the Data Segment window figure 5 and figure 6, remains the same as figure 6.

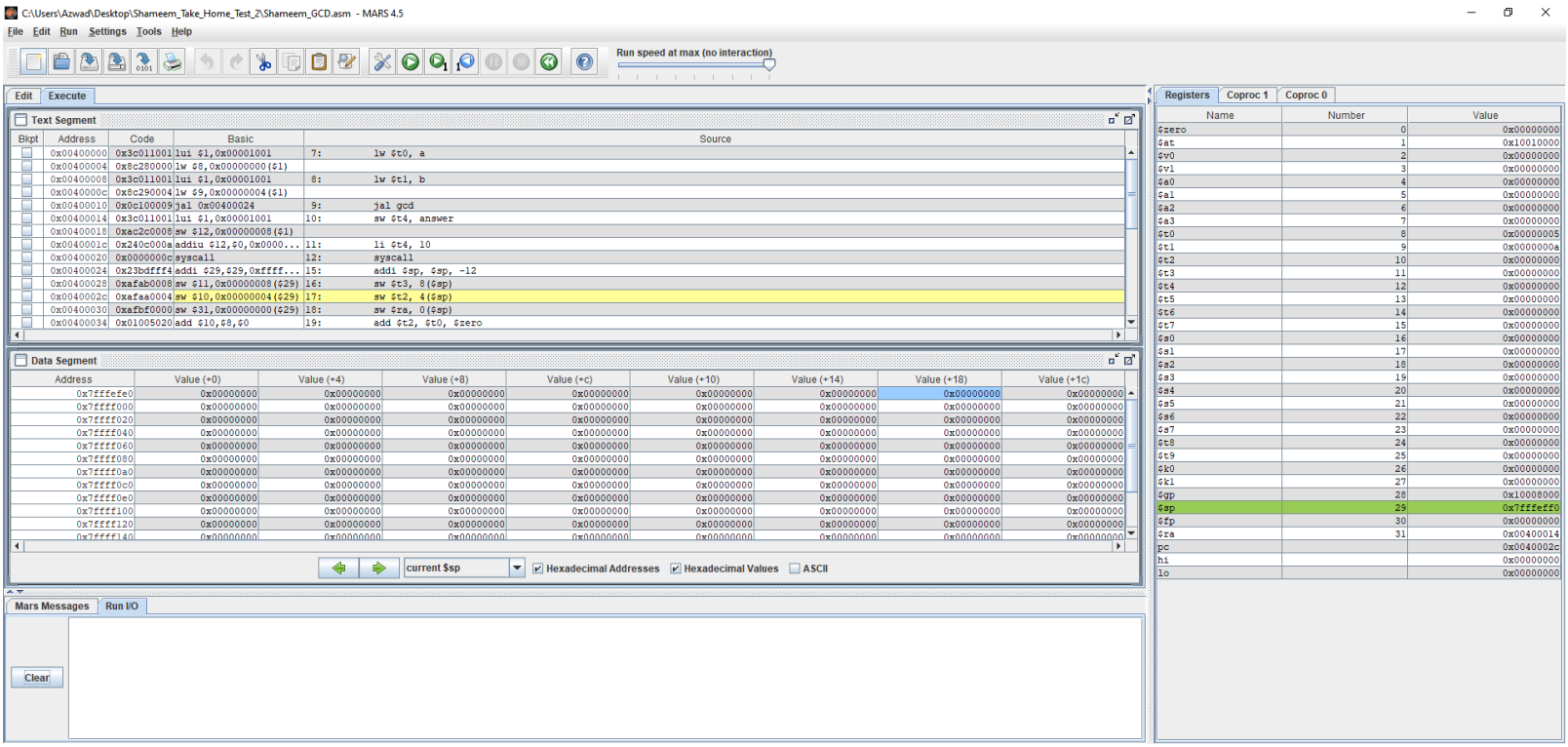

Figure 8: Shameem_GCD.asm code assembled in MARS (Line 16)

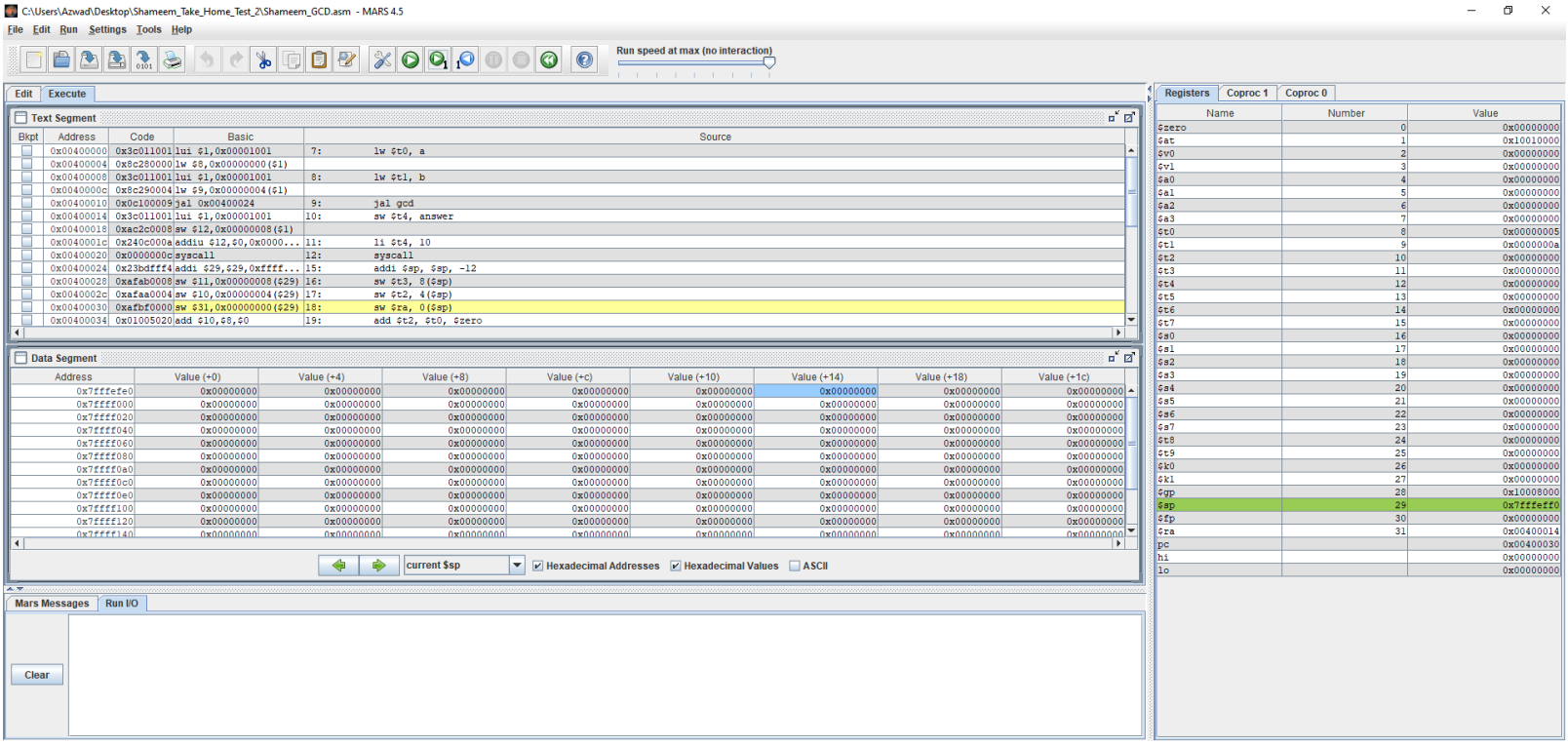

Figure 9: Shameem_GCD.asm code assembled in MARS (Line 17)

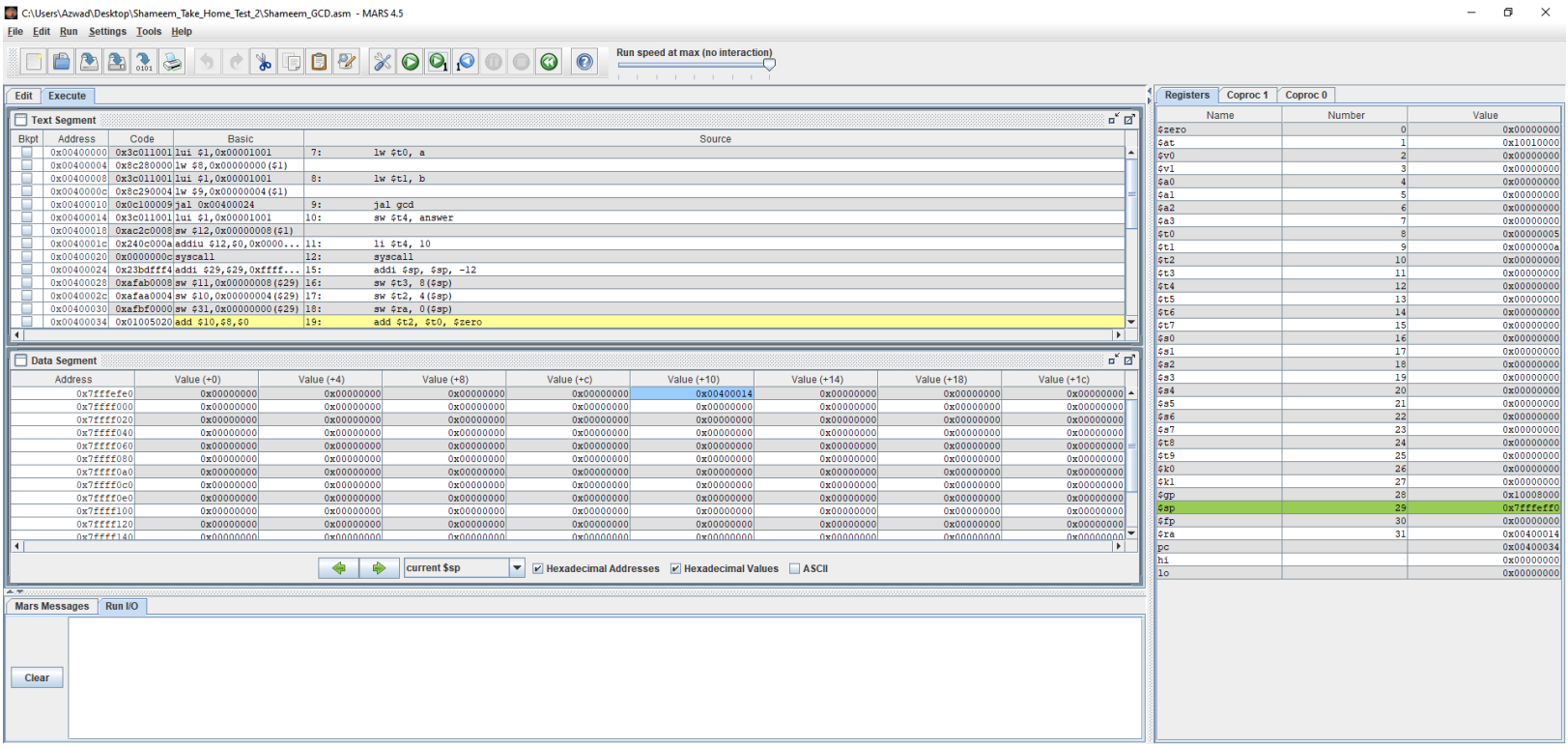

Figure 10: Shameem_GCD.asm code assembled in MARS (Line 18)

For Figure 8, Figure 9 and Figure 10.

Text Segment: This window shows that MARS has ran lines 16-18.

Registers: This window shows no change from Figure 7.

Data Segment: This window shows the variables stored in the addresses.

The value of $t3 and $t2, which is 0, is stored at the addresses 0x7FFFF010 and 0x7FFFEFFE respectively. The value of $ra, which is 0x00400014 is stored at addresses 0x7FFFEFF0. There has been no change in the previous values that have been added into the Data Segment window from figure 5 and figure 6.

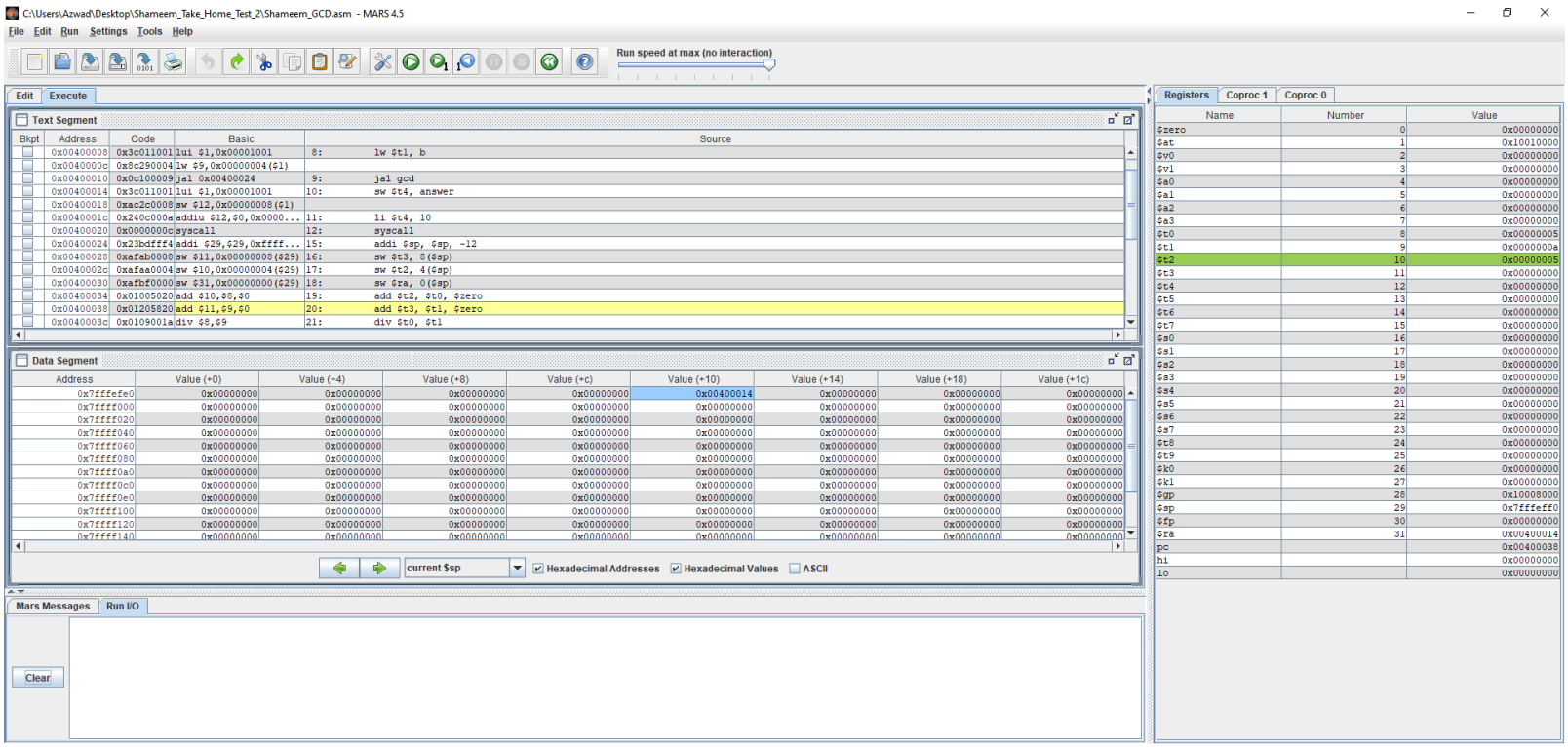

Figure 11: Shameem_GCD.asm code assembled in MARS (Line 19)

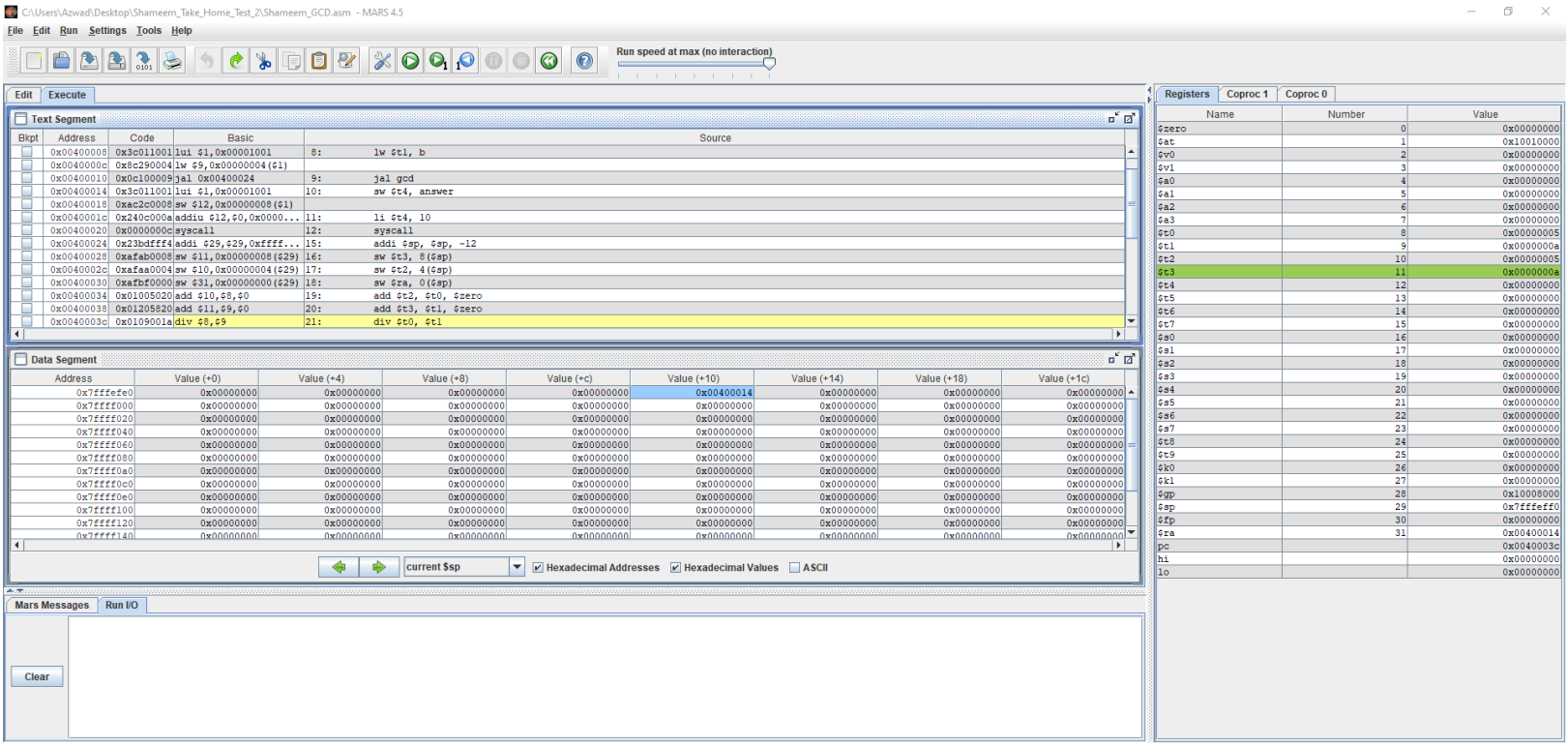

Figure 12: Shameem_GCD.asm code assembled in MARS (Line 20)

For Figure 11 and Figure 12.

Text Segment: This window shows that MARS has ran lines 16-18.

Registers: This window shows that the registers have made a copy of the values of variables a and variable b in order to do computation. The value of the $t0 register, which is 0x00000005, plus the value of $zero register, which is 0x00000000, is added into the highlighted $t3 register in figure 11. The $t3 register is now updated to the value of 0x00000005, which is the value of the variable a. The value of the $t3 register, which is 0x0000000a, plus the value of the $zero register, which is 0x00000000, is added into the highlighted $t4 register in figure 12. The value of the $t4 with the value of 0x0000000a, which is the value of the variable b.

Data Segment: This window shows no change since Figure 10.

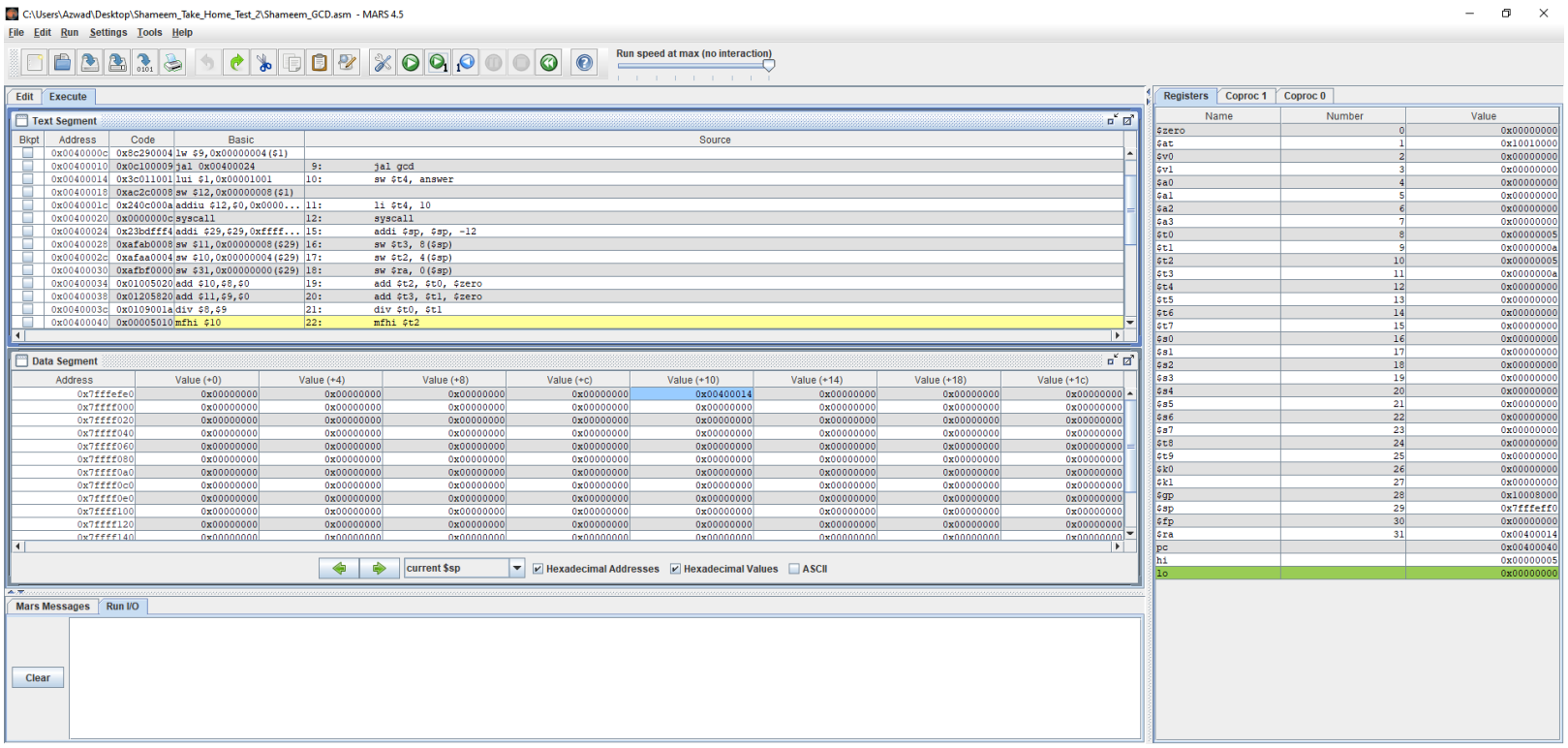

Figure 13: Shameem_GCD.asm code assembled in MARS (Line 21)

Text Segment: This window shows that MARS has ran line 21.

Registers: This window shows that the registers have been updated, the hi register shows the value of 0x00000005 and the lo register is highlighted and shows the value of 0x00000000.

Data Segment: This window shows no change since Figure 10.

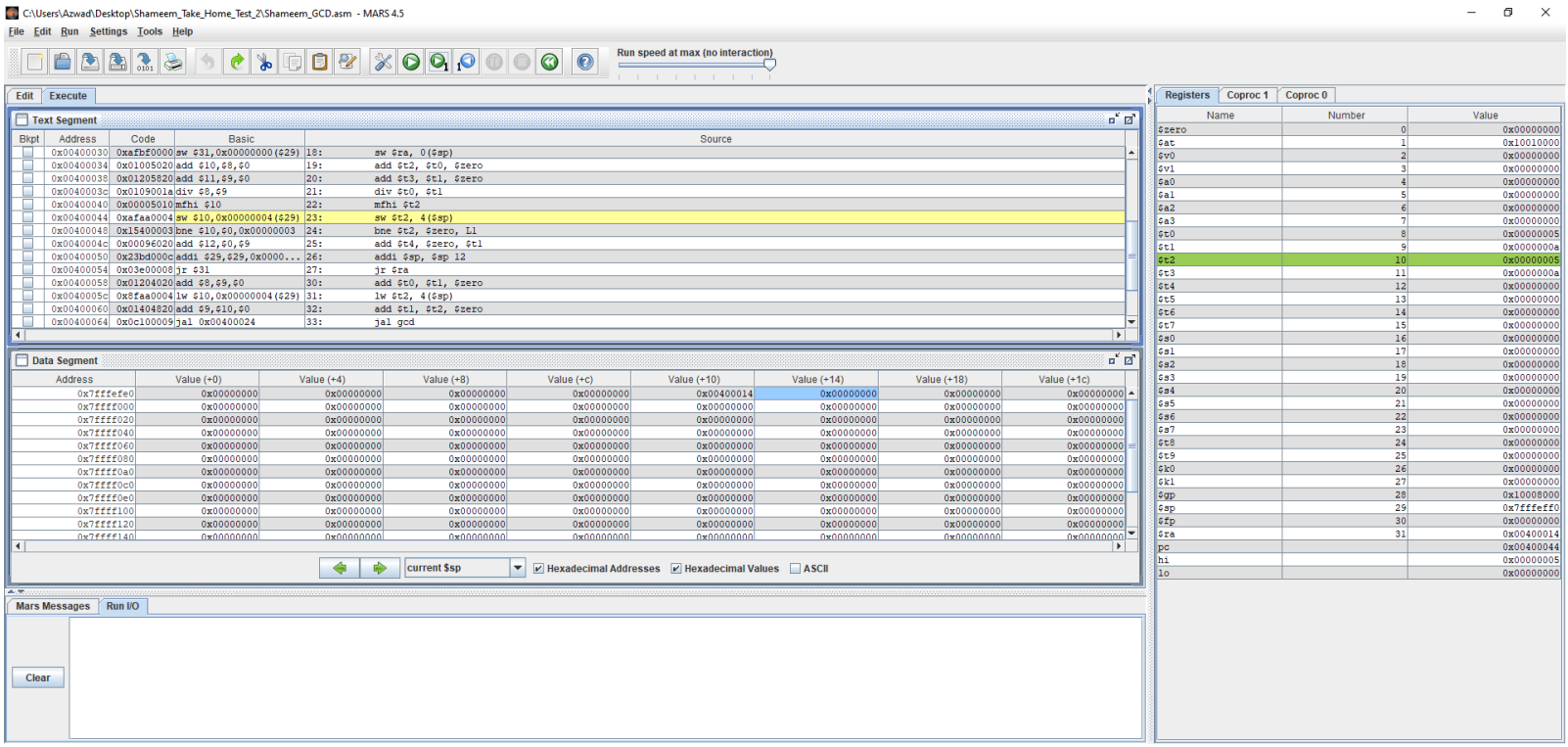

Figure 14: Shameem_GCD.asm code assembled in MARS (Line 22)

Text Segment: This window shows that MARS has ran line 22.

Registers: This window shows that the registers have been updated. The register $t2 is highlighted and has been inputted with the remainder of the division of $t0 and $t1 which is 0x00000005.

Data Segment: This window shows no change since Figure 10.

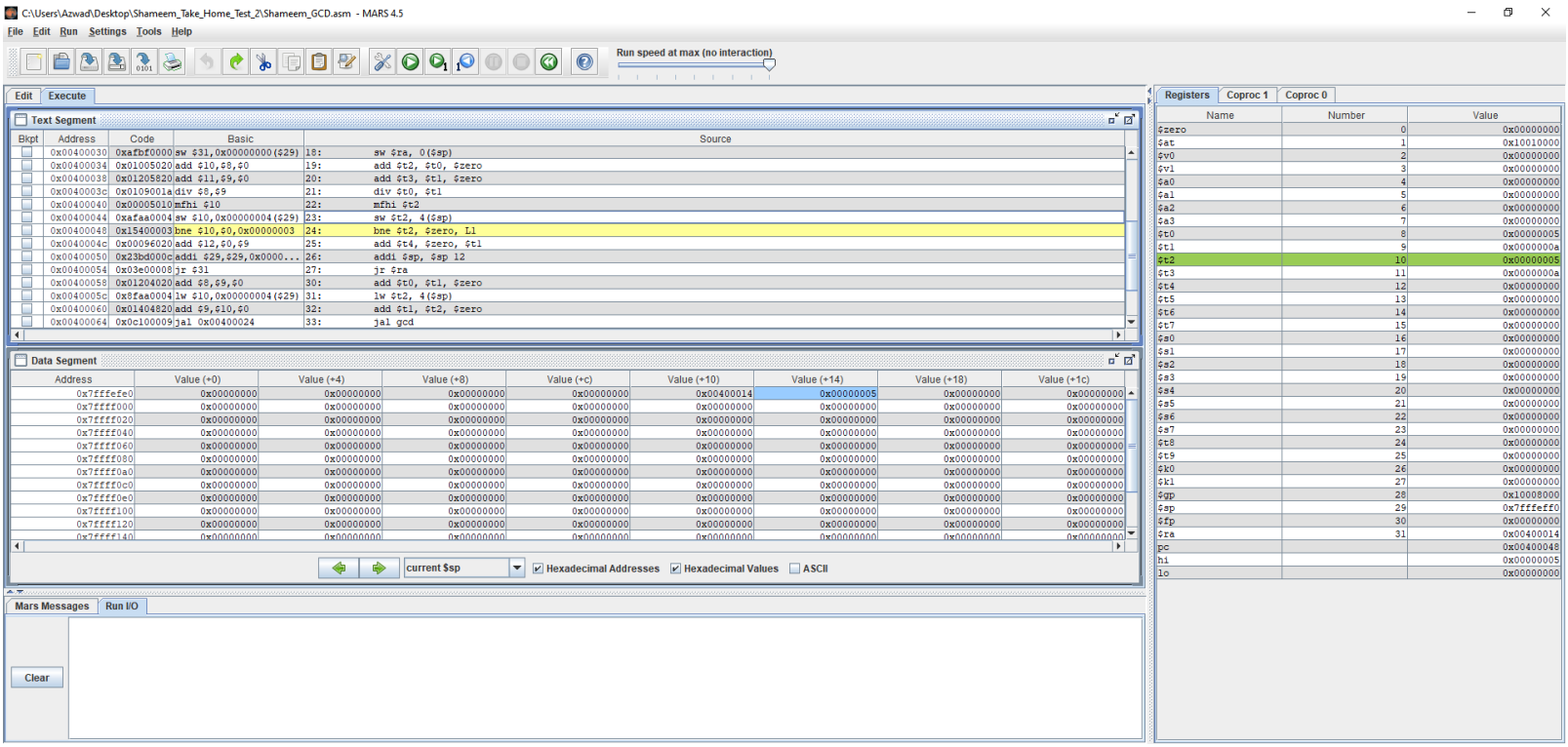

Figure 15: Shameem_GCD.asm code assembled in MARS (Line 23)

Text Segment: This window shows that MARS has ran line 23.

Registers: This window shows that the registers have not changed since figure 14.

Data Segment: This window shows that the values at addresses have been updated. At the highlighted address 0x7FFFEFFE, the value of $t2 is stored, which is 0x00000005, which is also the value of the remainder of the division of variable a and variable b.

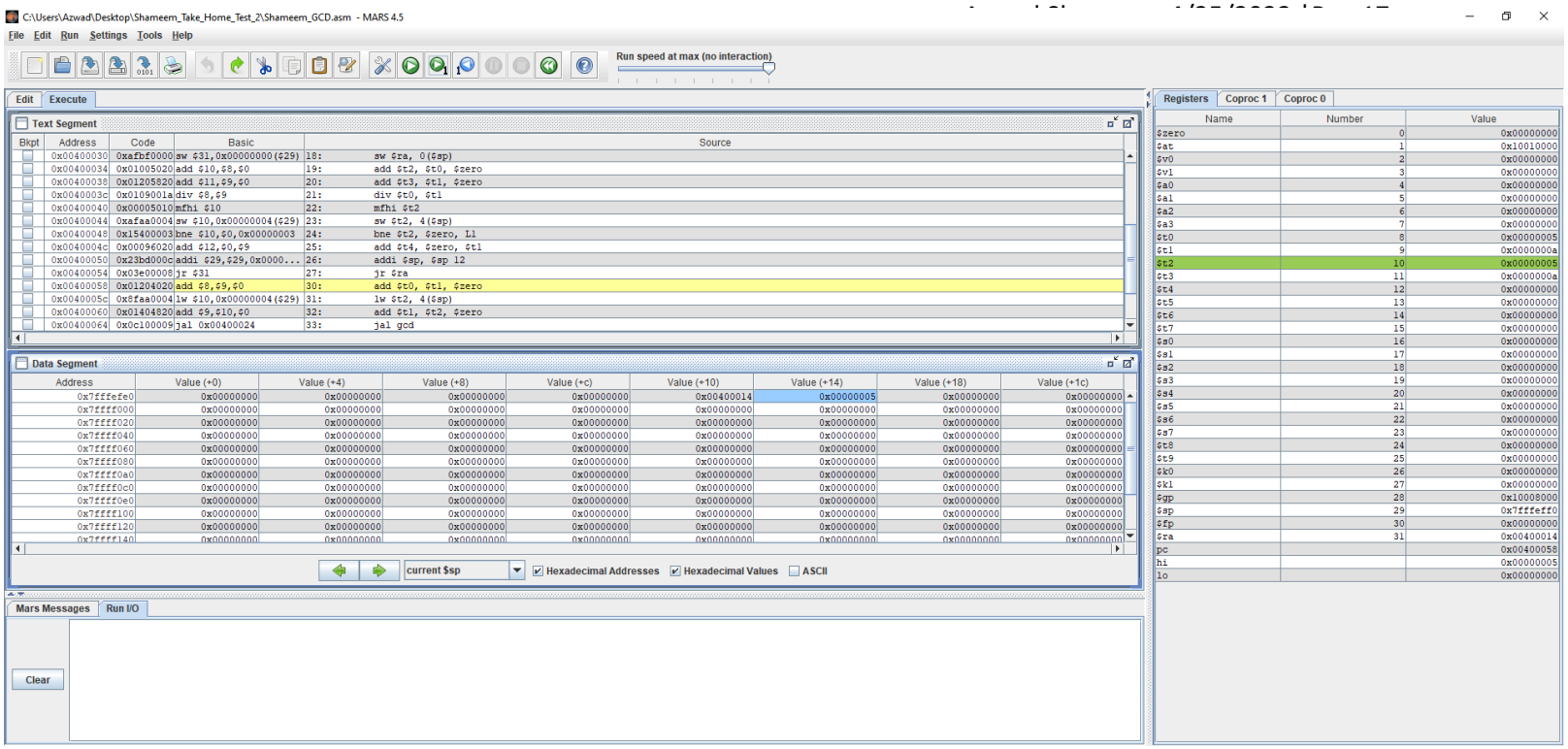

Figure 16: Shameem_GCD.asm code assembled in MARS (Line 24)

Text Segment: This window shows that MARS has ran line 24, which has made it jump to line 30 because of the BNE instruction.

Registers: This window shows that the registers have not changed since figure 14.

Data Segment: This window shows that there has been no change since figure 15.

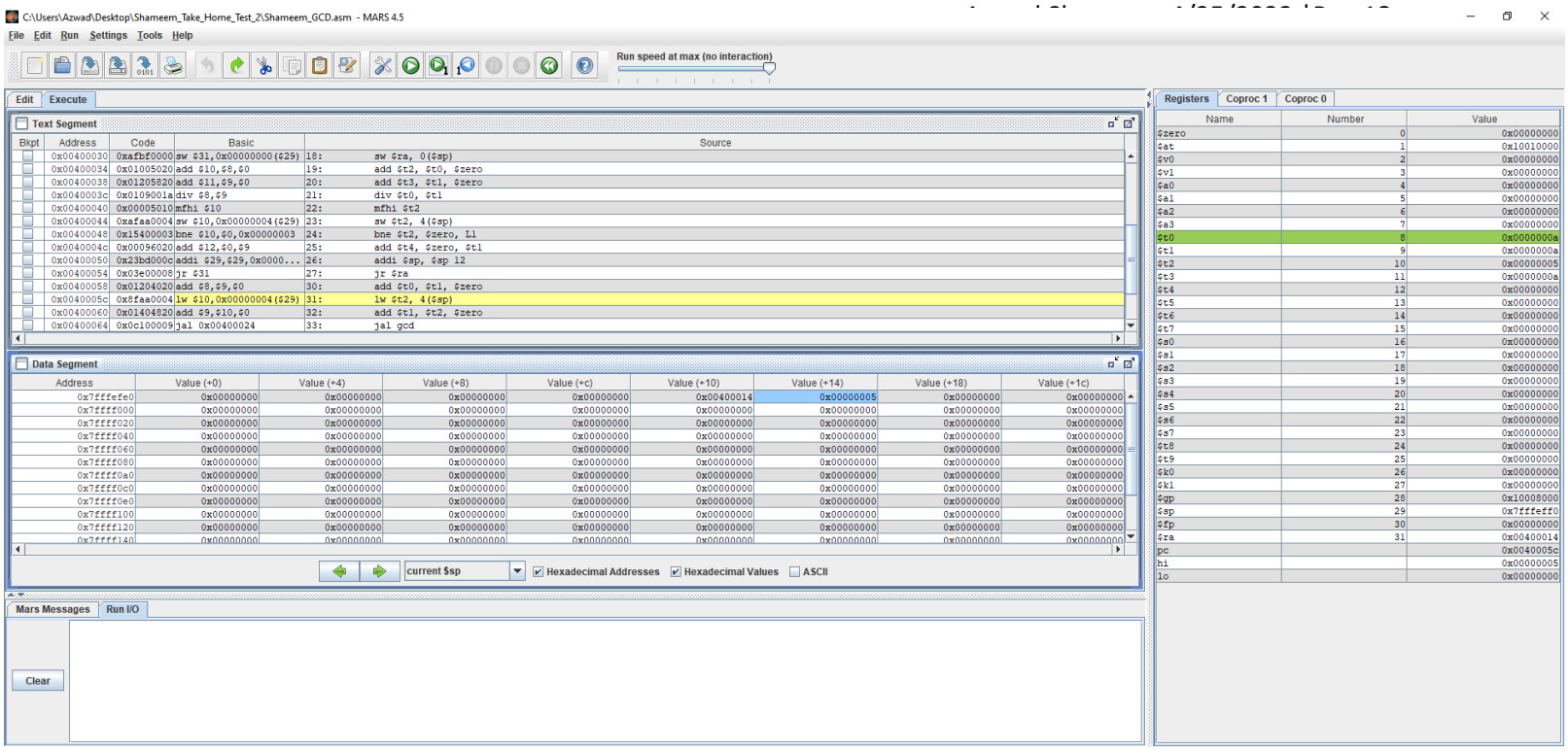

Figure 17: Shameem_GCD.asm code assembled in MARS (Line 30)

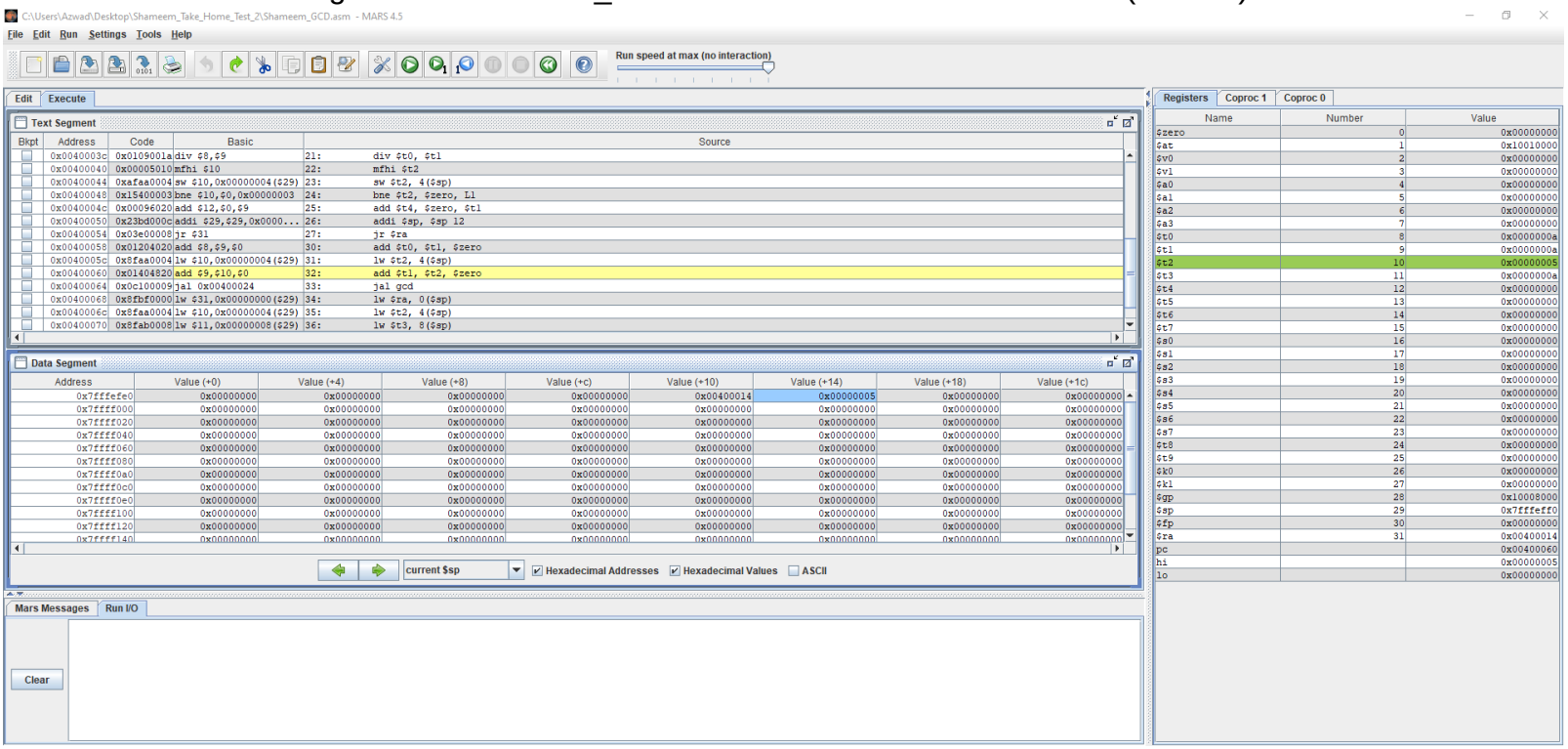

Figure 18: Shameem_GCD.asm code assembled in MARS (Line 31)

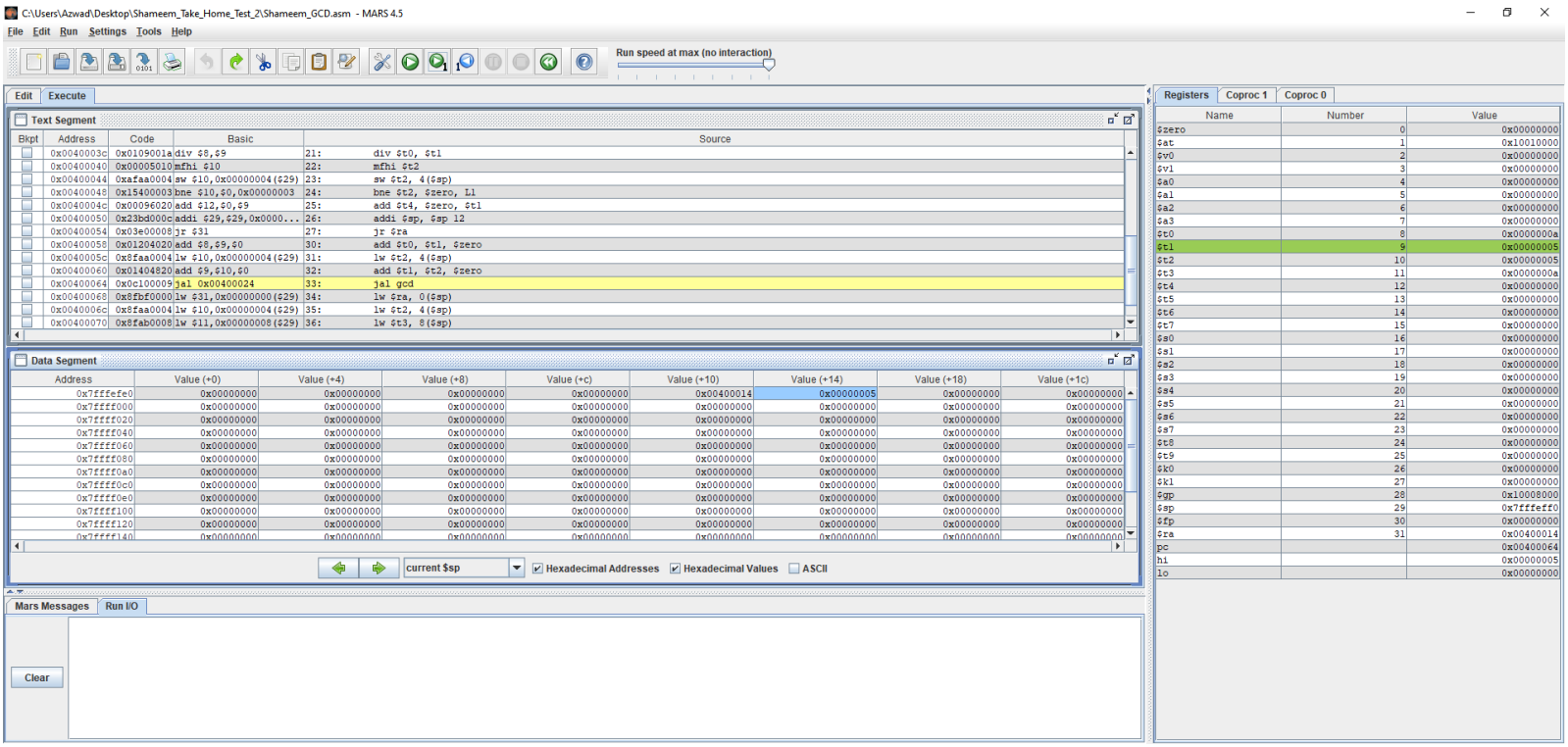

Figure 19: Shameem_GCD.asm code assembled in MARS (Line 32)

For Figure 17, Figure 18 and Figure 19.

Text Segment: This window shows that MARS has ran lines 30-32.

Registers: This window shows that the registers have been updated. The value of $t0 register is now 0x0000000a and the value of the $t1 register is now 0x00000005.

Data Segment: This window shows that there has been no change since figure 15.

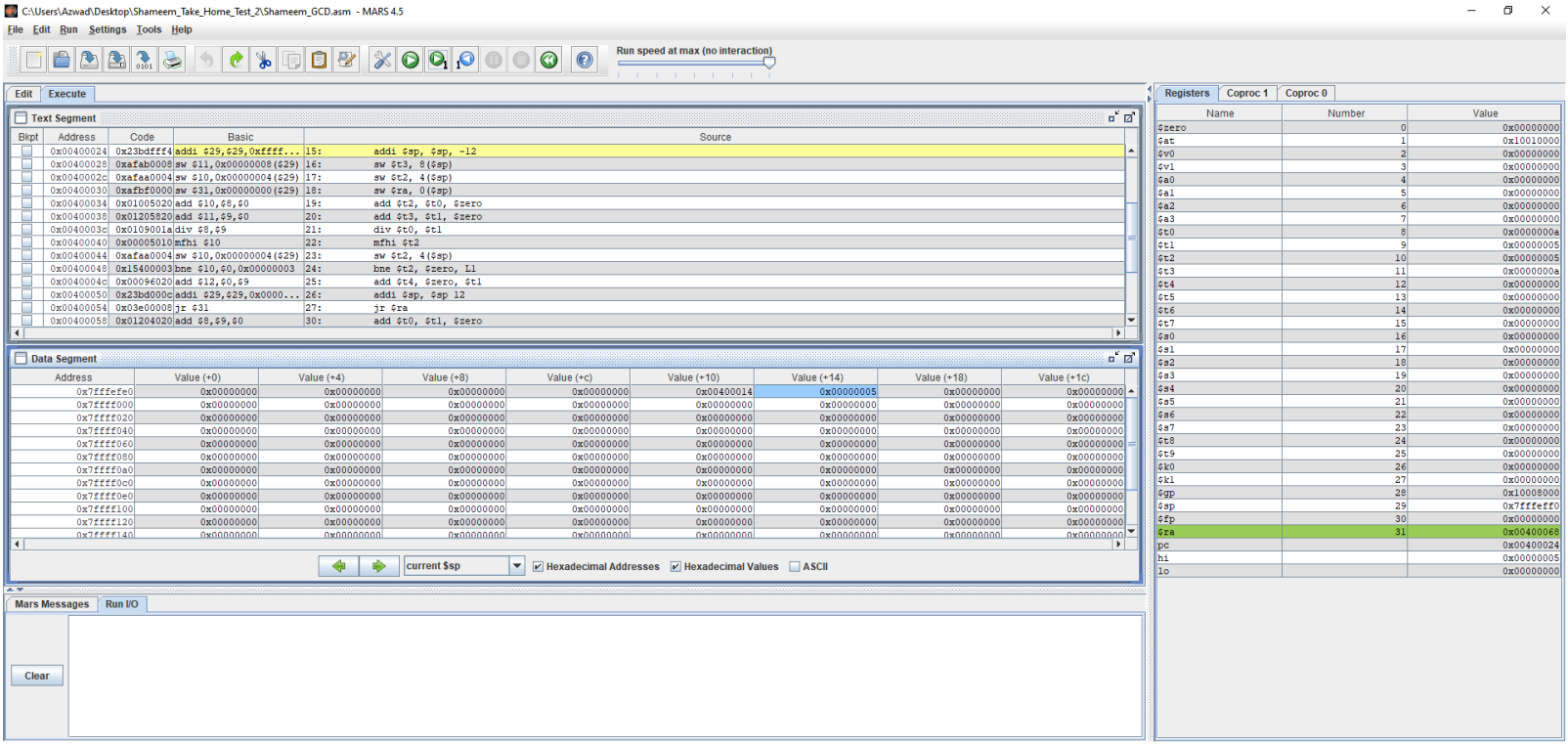

Figure 20: Shameem_GCD.asm code assembled in MARS (Line 33 -> jumped to 15)

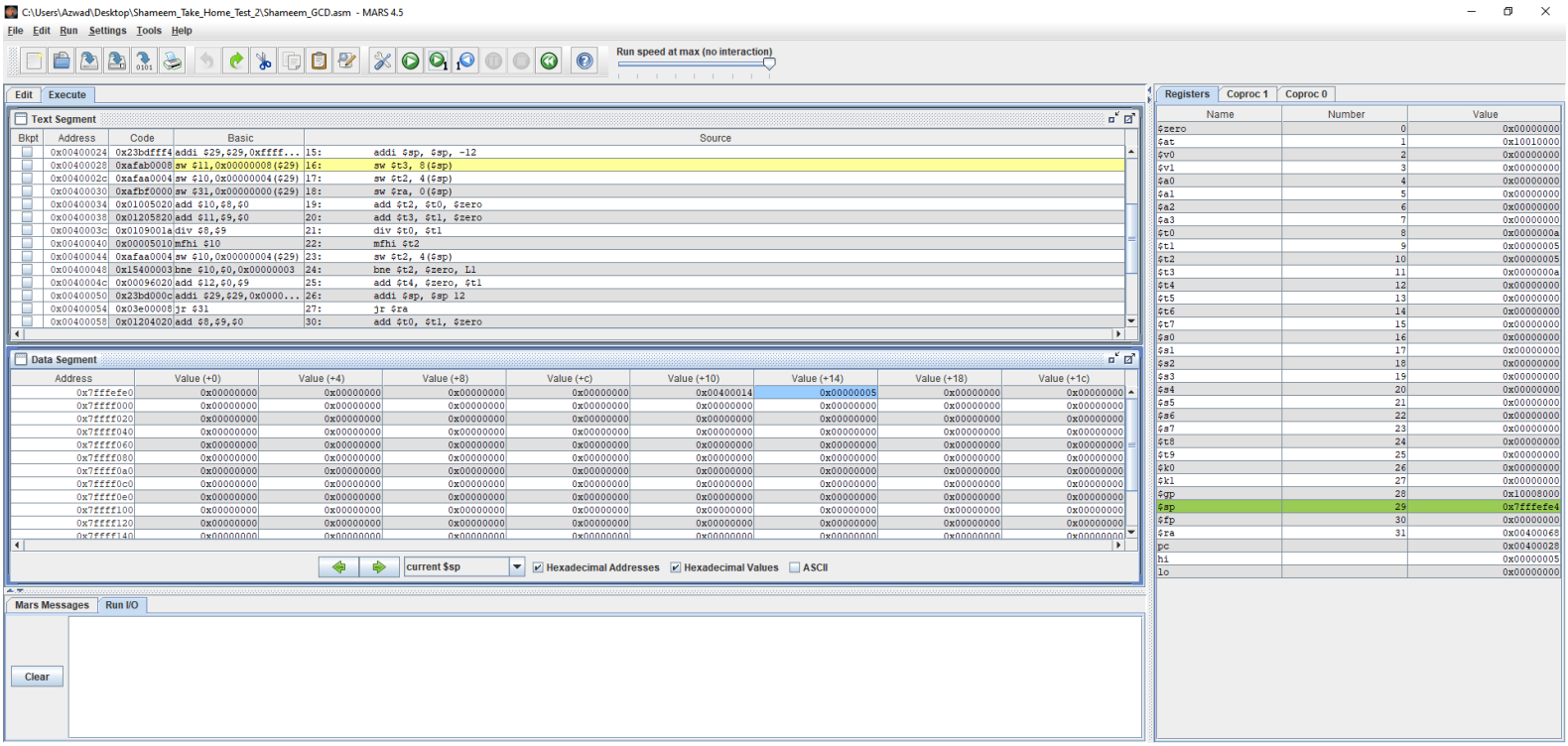

Figure 21: Shameem_GCD.asm code assembled in MARS (Line 15)

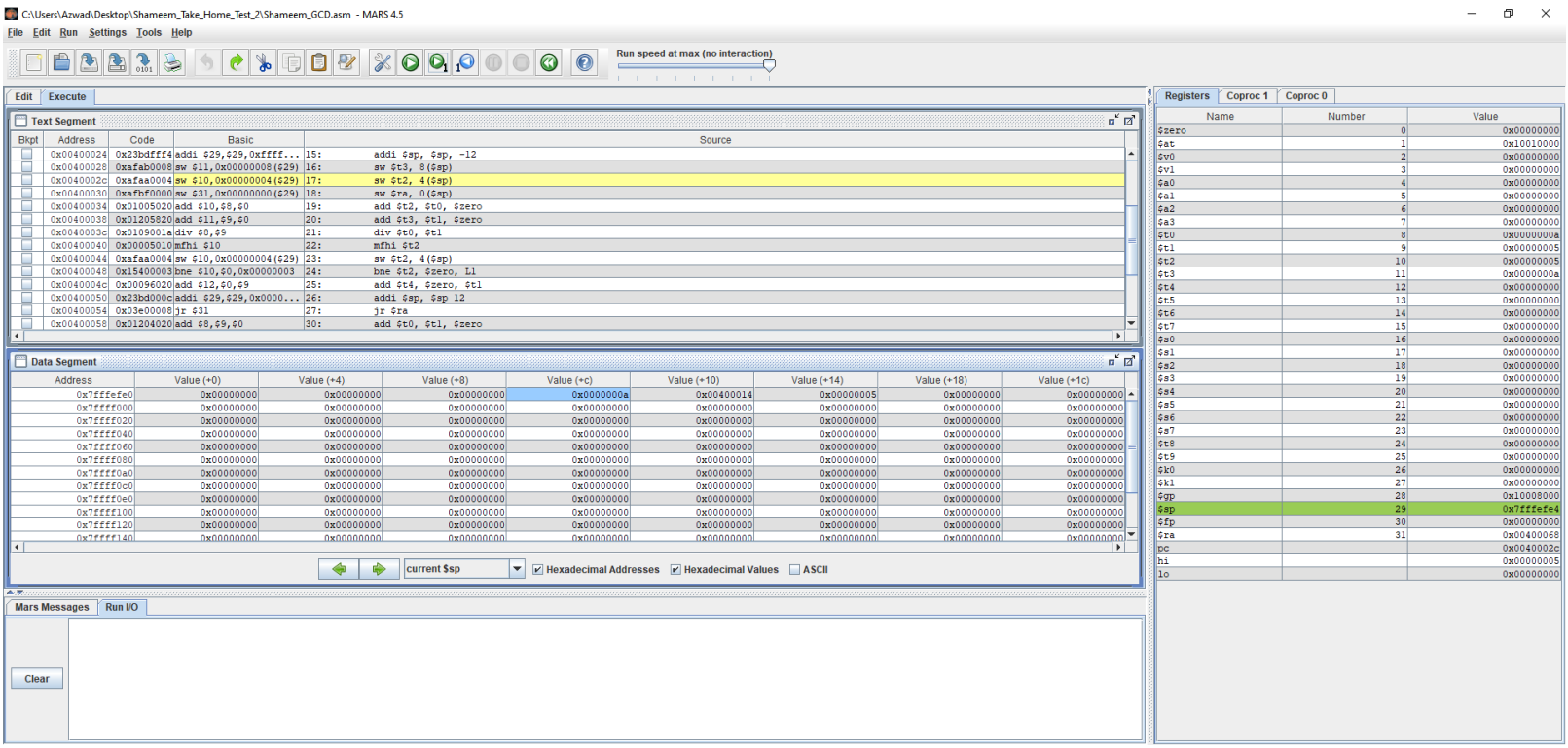

Figure 22: Shameem_GCD.asm code assembled in MARS (Line 16)

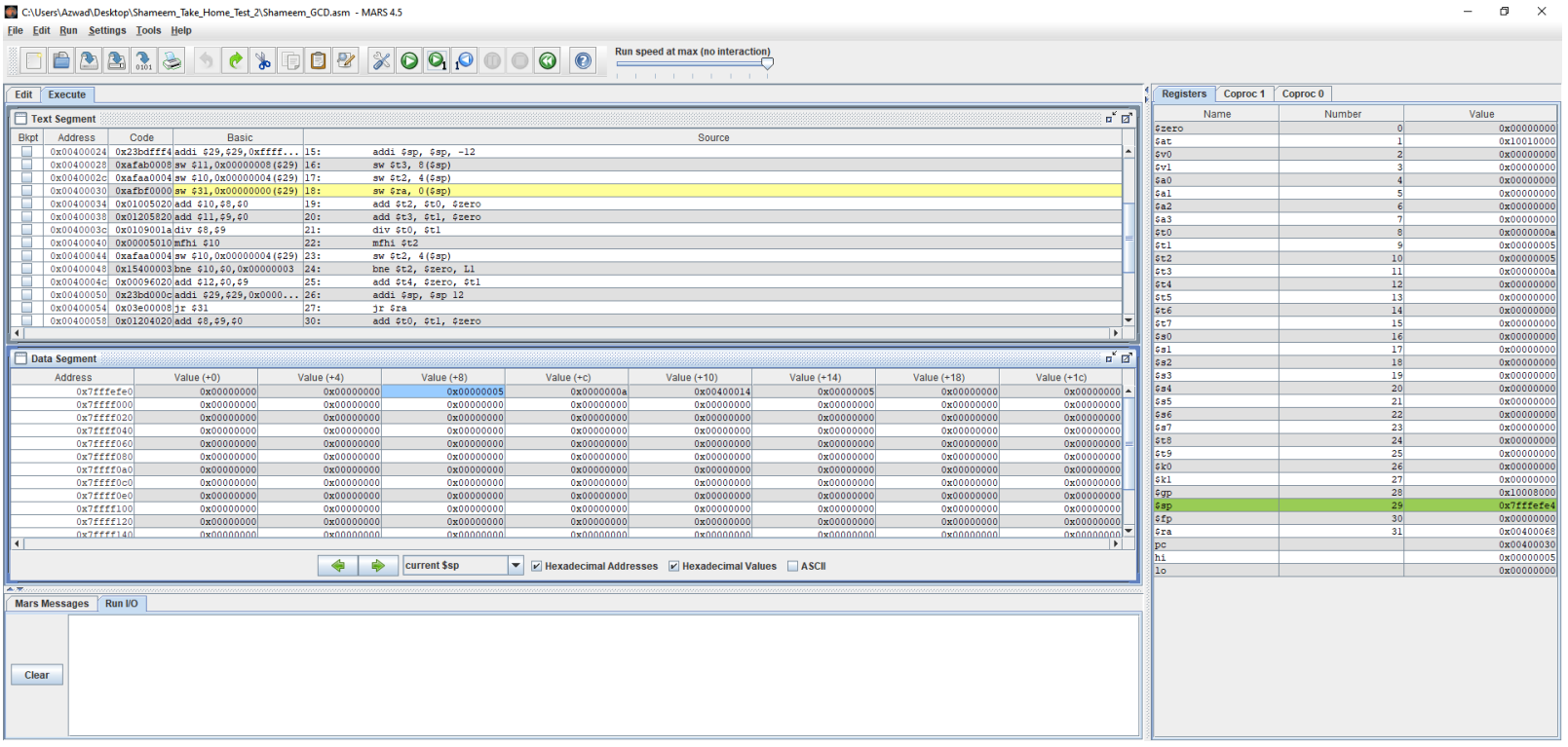

Figure 23: Shameem_GCD.asm code assembled in MARS (Line 17)

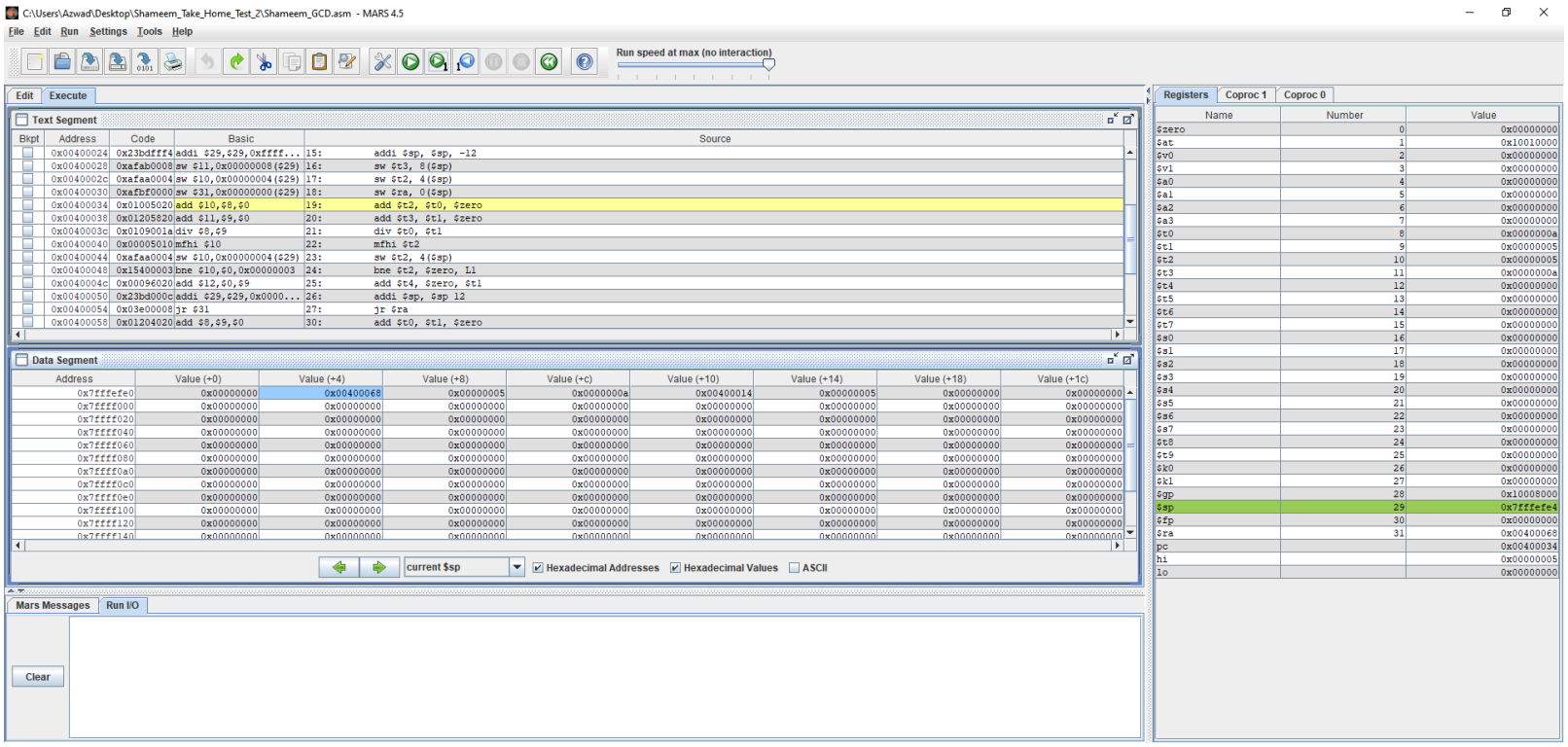

Figure 24: Shameem_GCD.asm code assembled in MARS (Line 18)

For Figure 20, Figure 21, Figure 22, Figure 23, Figure 24.

Text Segment: This window shows that MARS has ran line 30 and jumped to 15 and then ran lines 15-18.

Registers: This window shows that the registers have been updated. The $ra register is now 0x00400068 and the stack pointer register, $sp is now 0x7FFFEFE4.

Data Segment: This window shows that at addresses 0x7FFFEFEC, 0x7FFFEF8, 0x7FFFEFE4, the values have been updated with 0x00400068, 0x0000000A and 0x00000005 respectively.

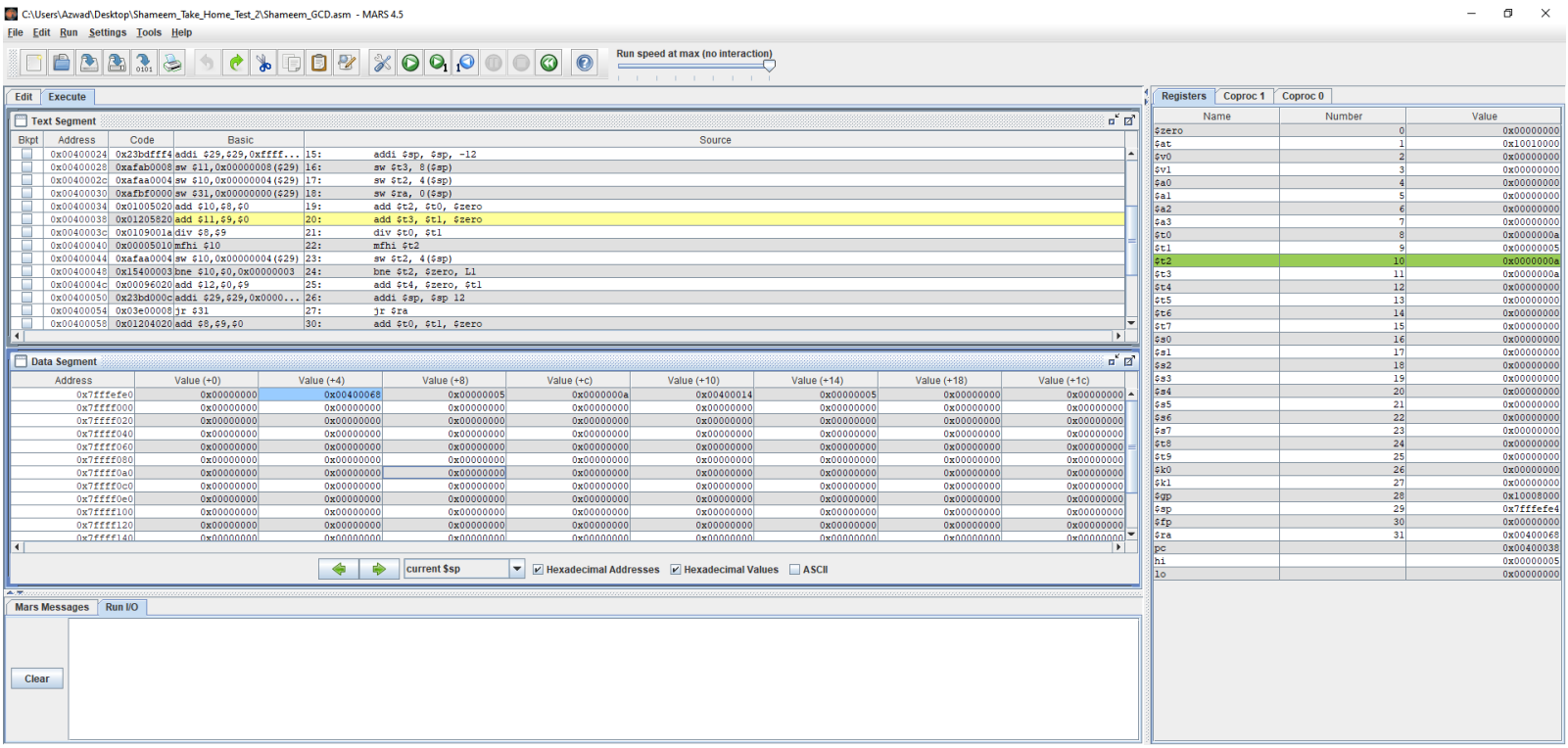

Figure 25: Shameem_GCD.asm code assembled in MARS (Line 19)

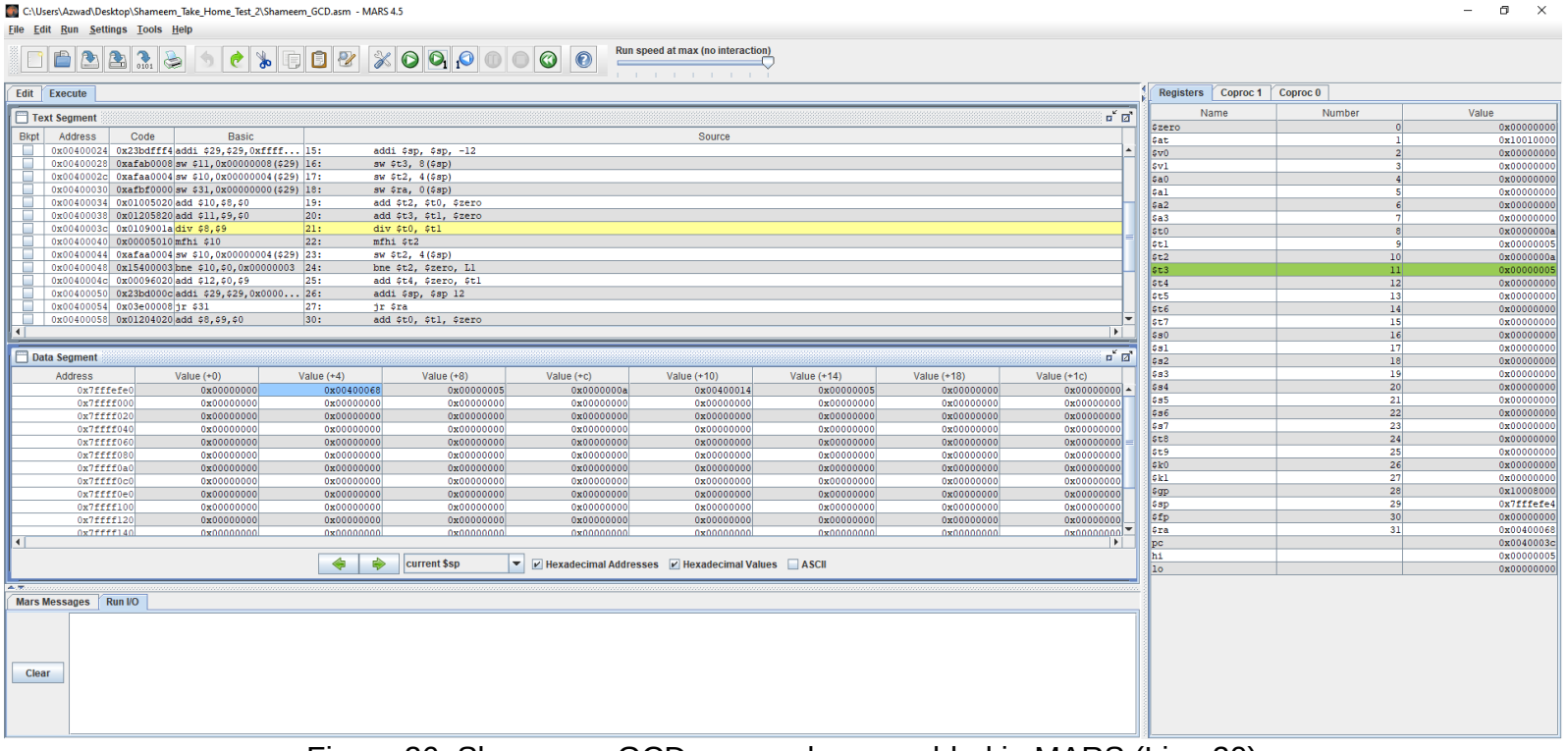

Figure 26: Shameem_GCD.asm code assembled in MARS (Line 20)

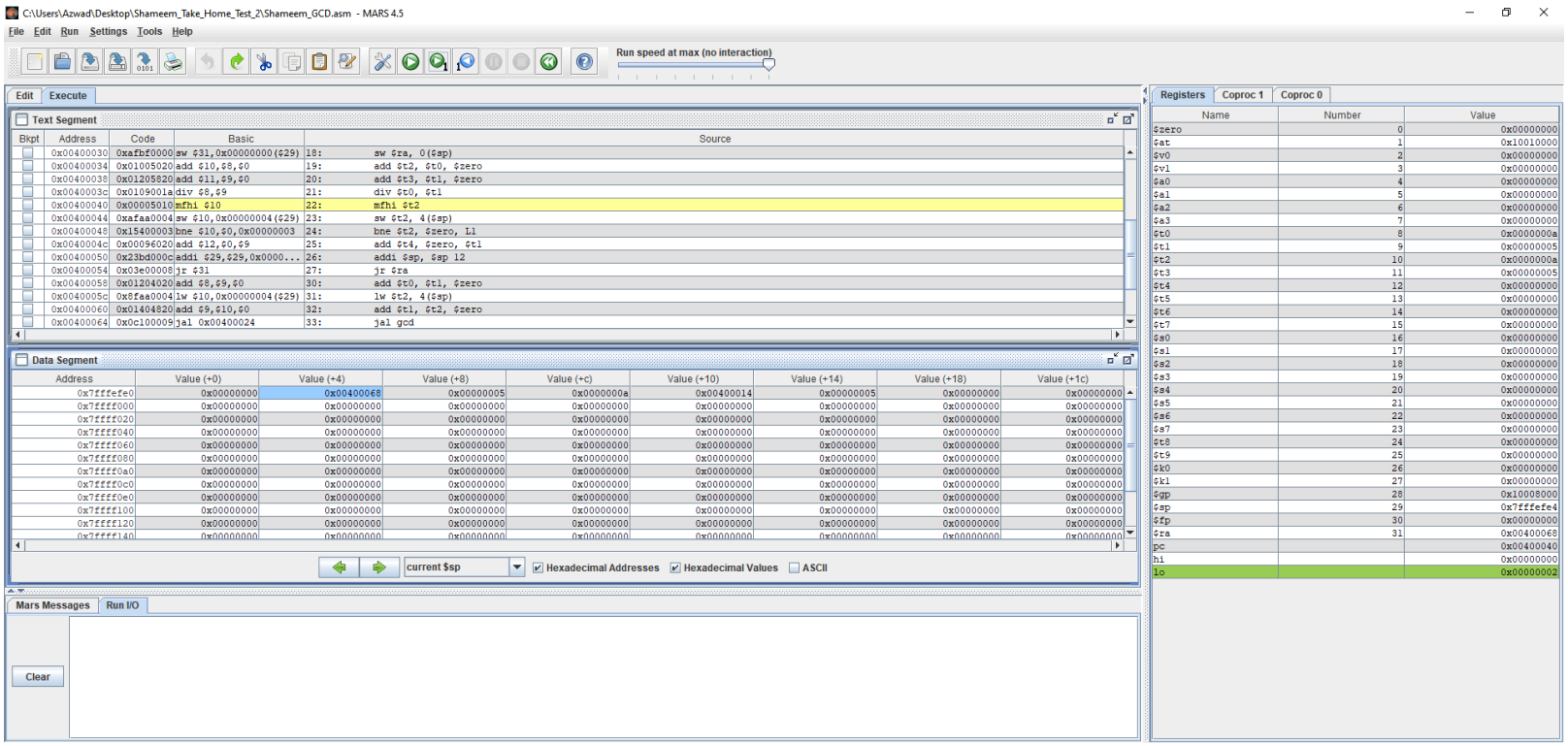

Figure 27: Shameem_GCD.asm code assembled in MARS (Line 21)

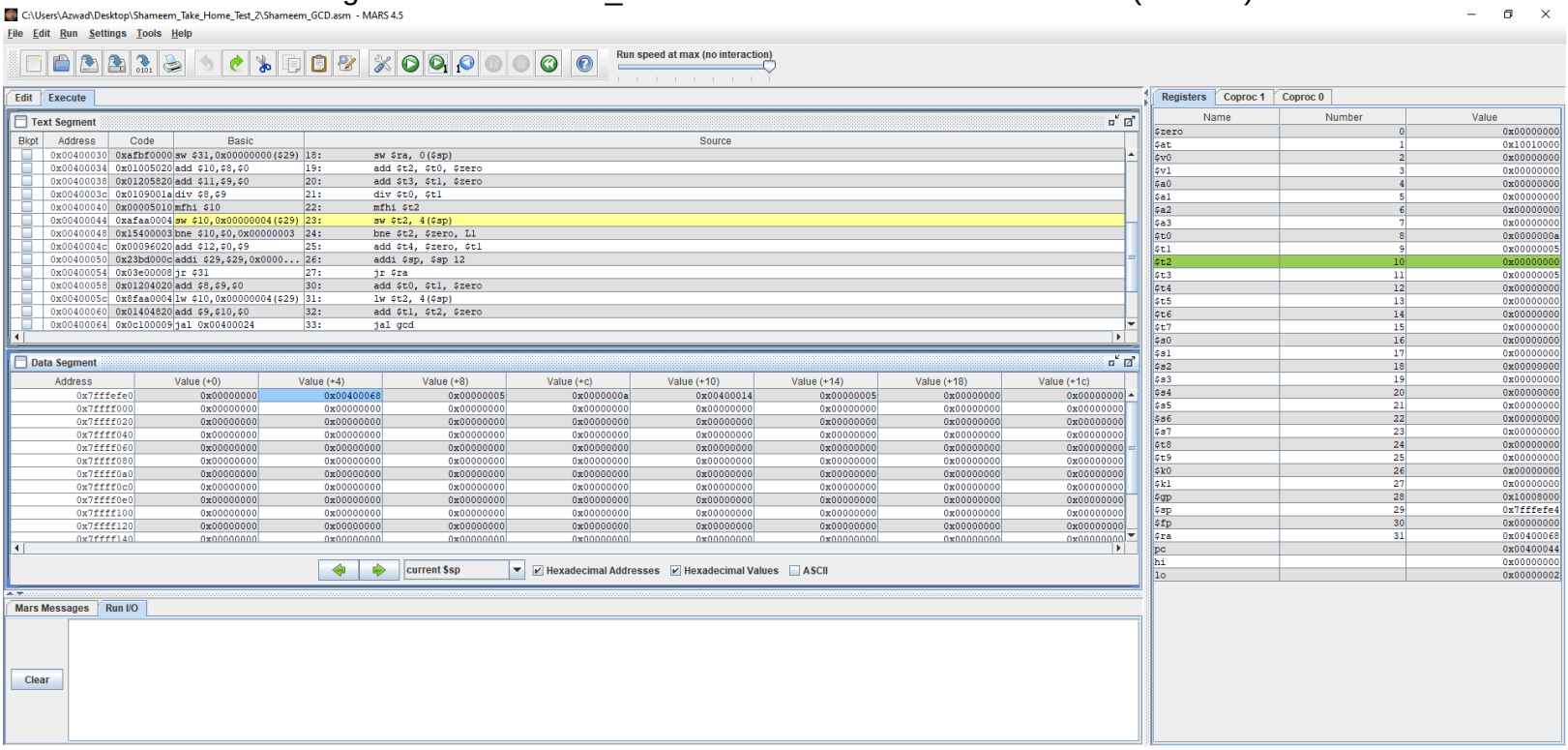

Figure 28: Shameem_GCD.asm code assembled in MARS (Line 22)

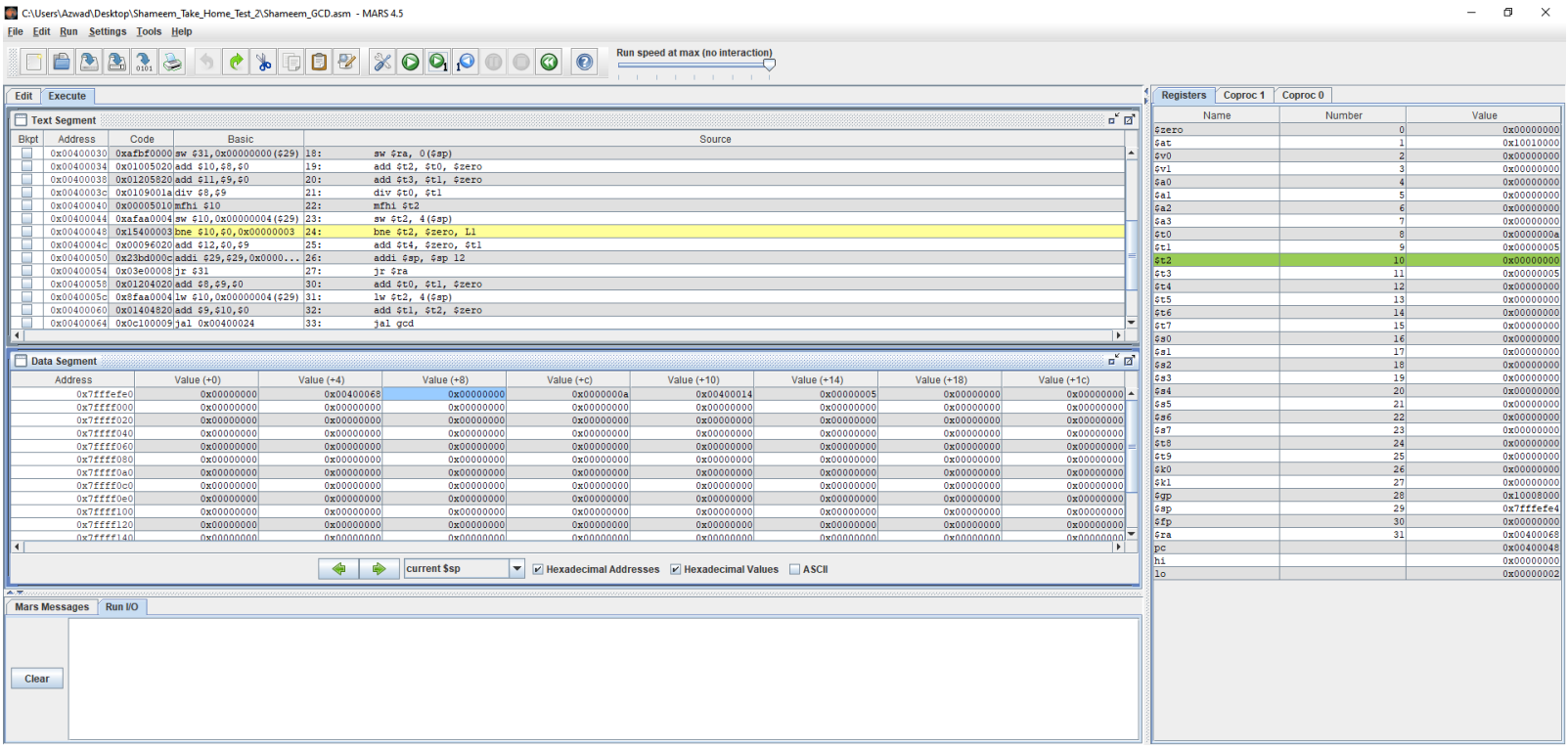

Figure 29: Shameem_GCD.asm code assembled in MARS (Line 23)

For Figure 25, Figure 26, Figure 27, Figure 28, Figure 29

Text Segment: This window shows that MARS has ran lines 19-23.

Registers: This window shows that the registers have been updated. The $t2 register is now 0x00000000 and the $t1 register is 0x00000005 and the lo register is now 0x00000002.

Data Segment: This window shows that at addresses 0x7FFFEF8,the values have been updated with 0x00000000.

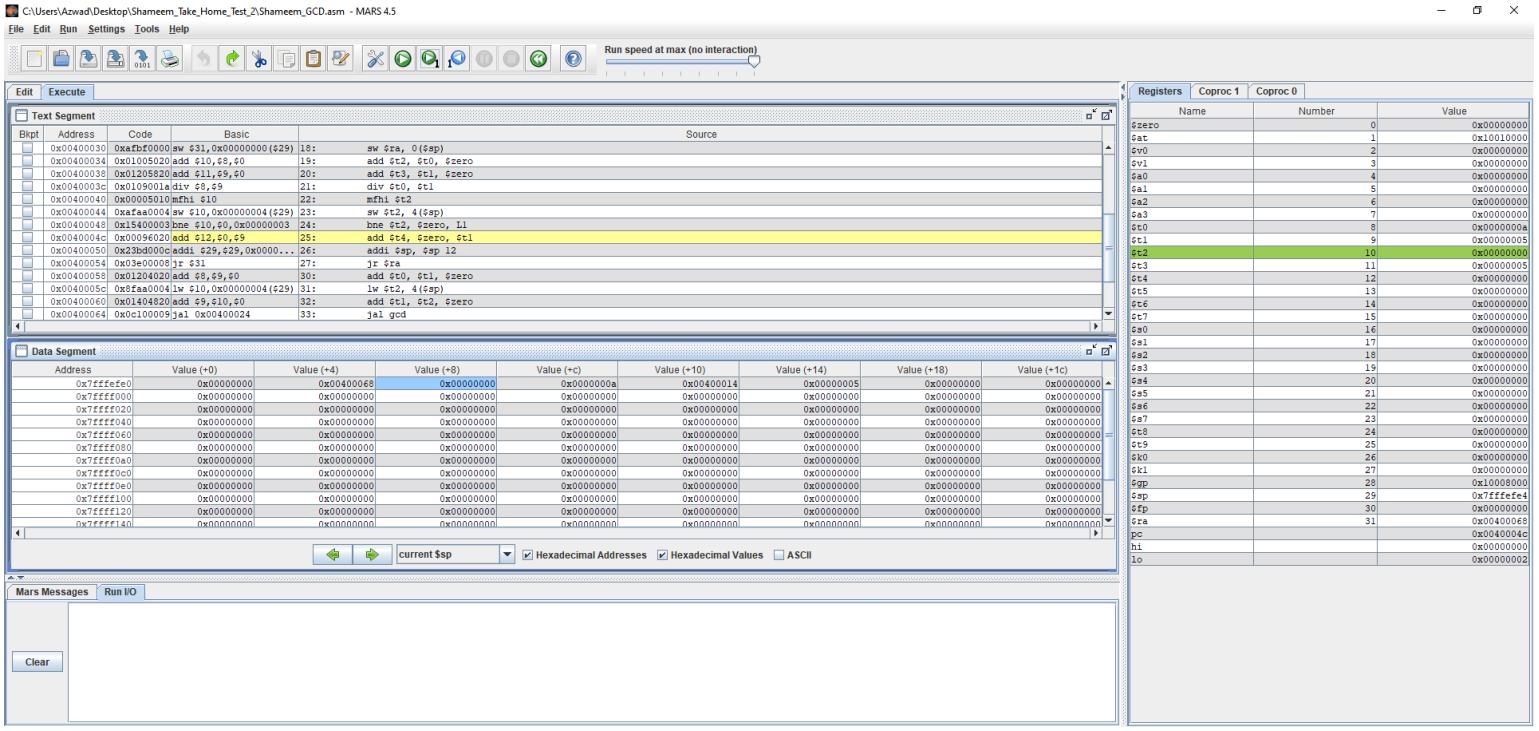

Figure 30: Shameem_GCD.asm code assembled in MARS (Line 24)

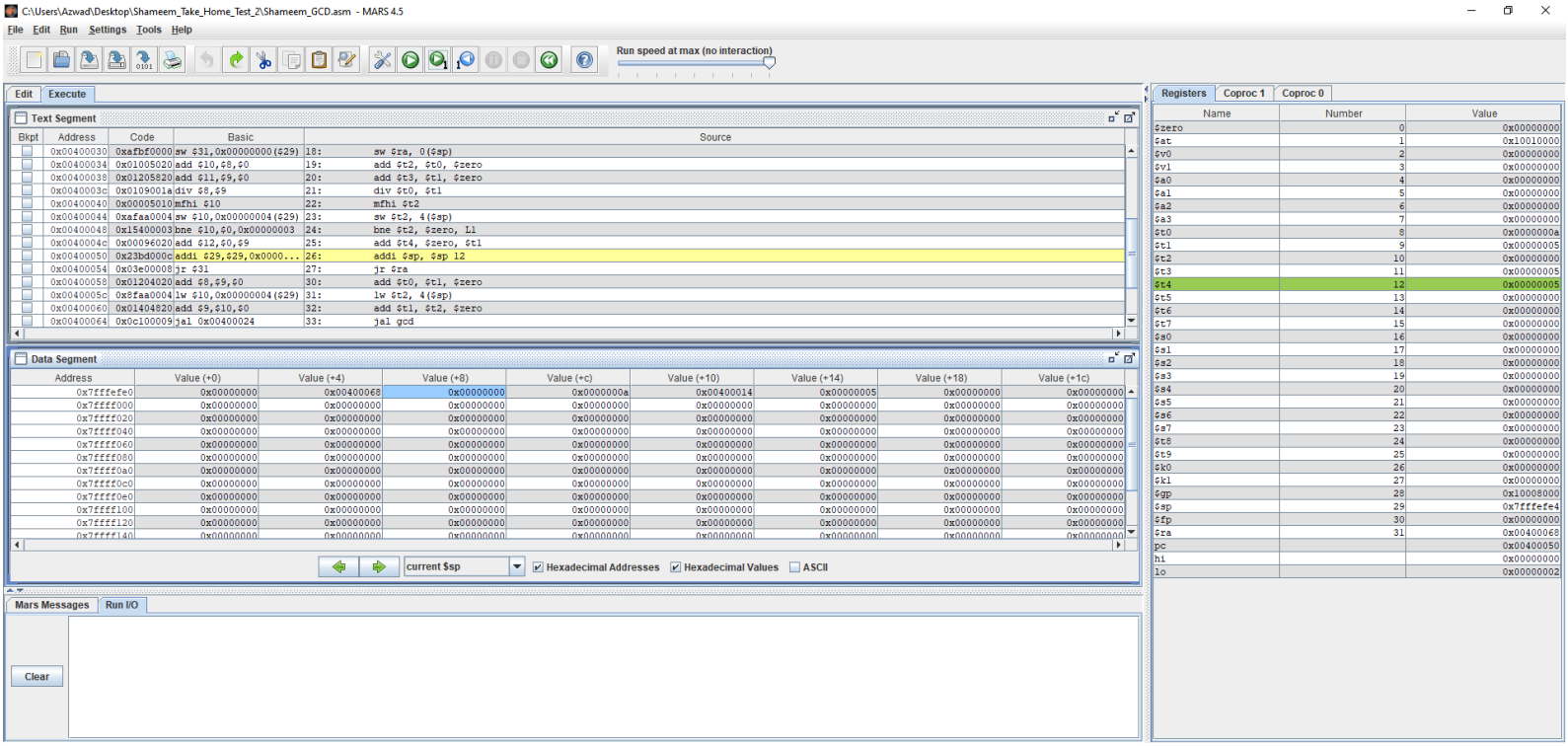

Figure 31: Shameem_GCD.asm code assembled in MARS (Line 25)

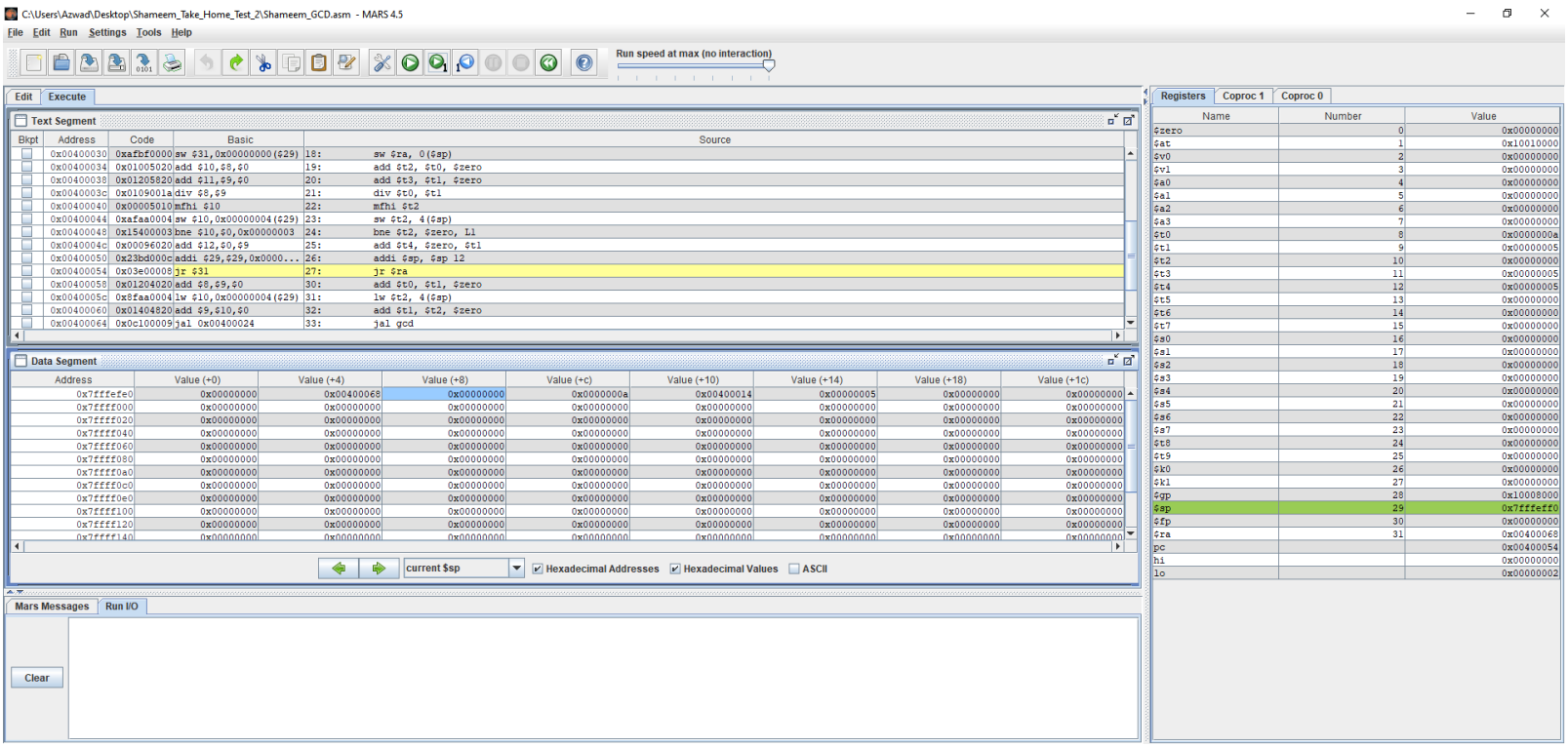

Figure 32: Shameem_GCD.asm code assembled in MARS (Line 26)

For Figure 30, Figure 31, Figure 32

Text Segment: This window shows that MARS has ran lines 24-26.

Registers: This window shows that the $sp register is now 0x7FFFEF0.

Data Segment: This window shows no change since figure 29.

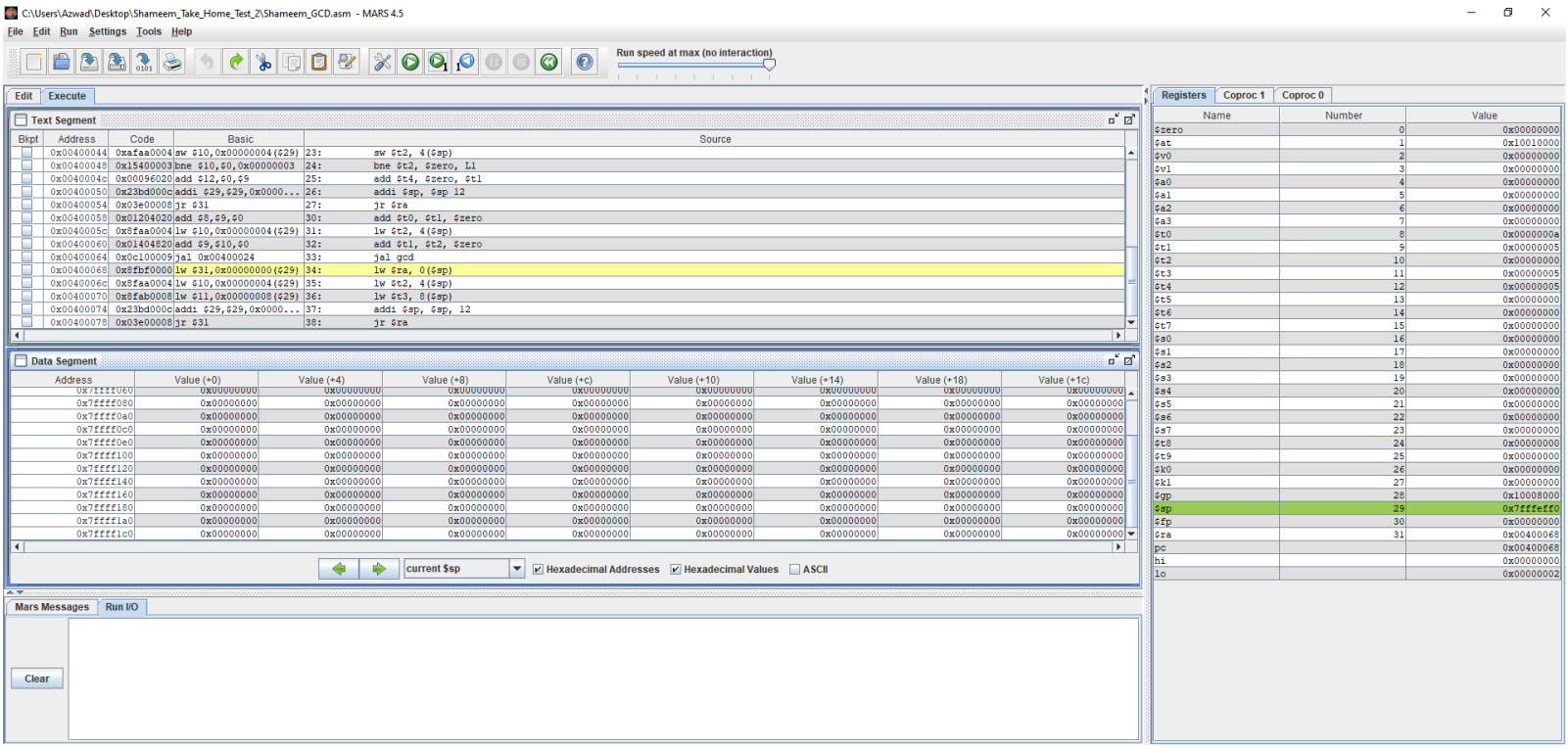

Figure 33: Shameem_GCD.asm code assembled in MARS (Line 34, jumped from 27)

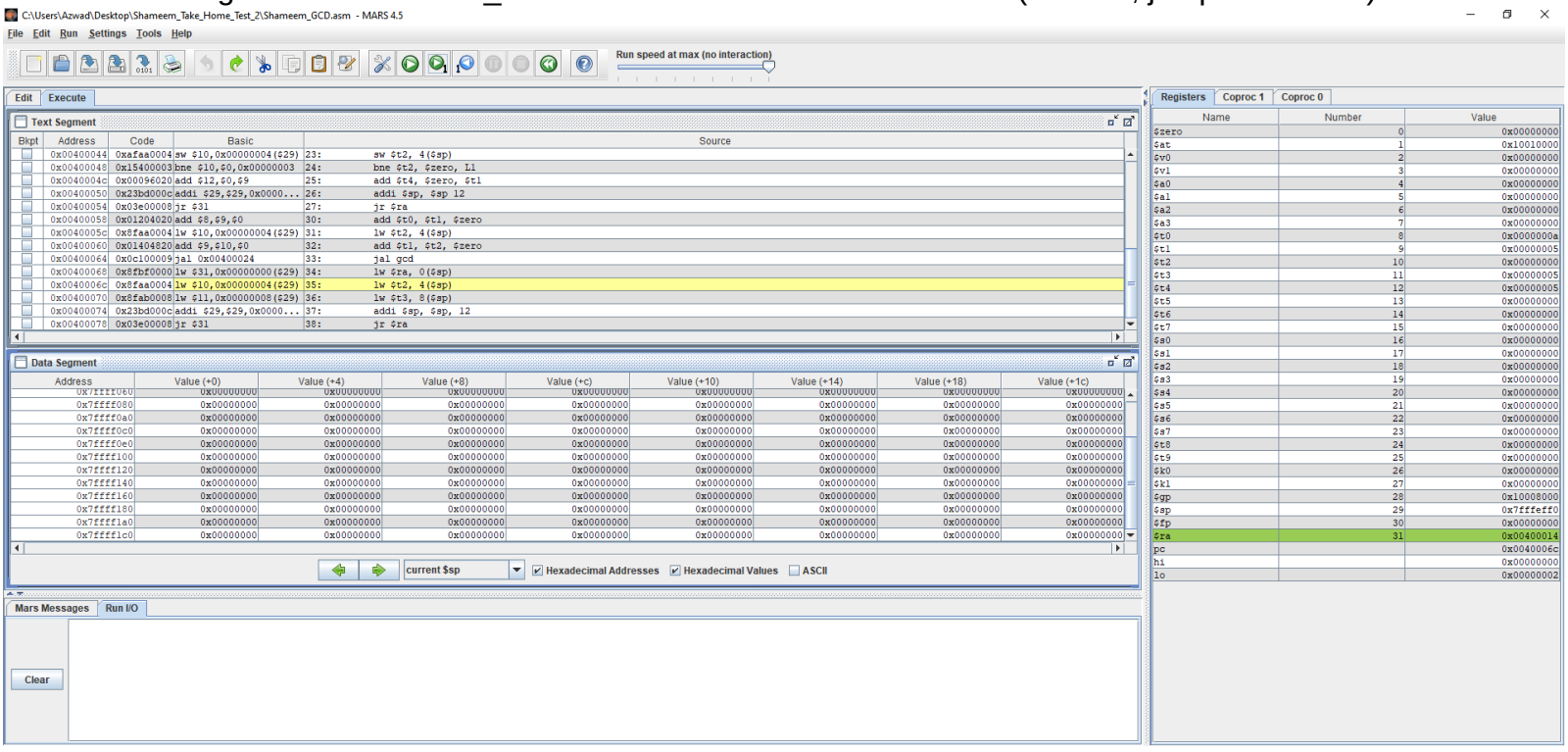

Figure 34: Shameem_GCD.asm code assembled in MARS (Line 35)

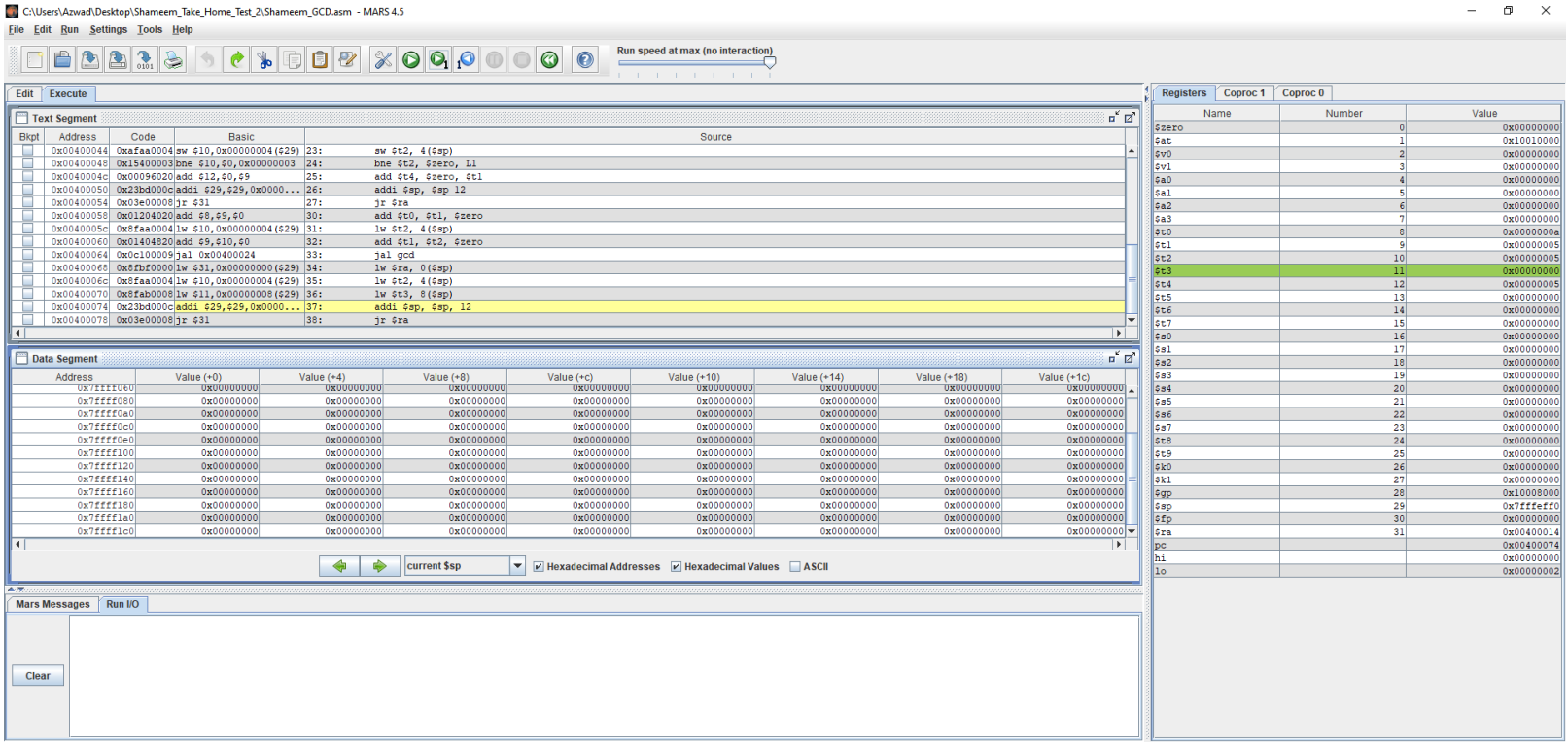

Figure 35: Shameem_GCD.asm code assembled in MARS (Line 36)

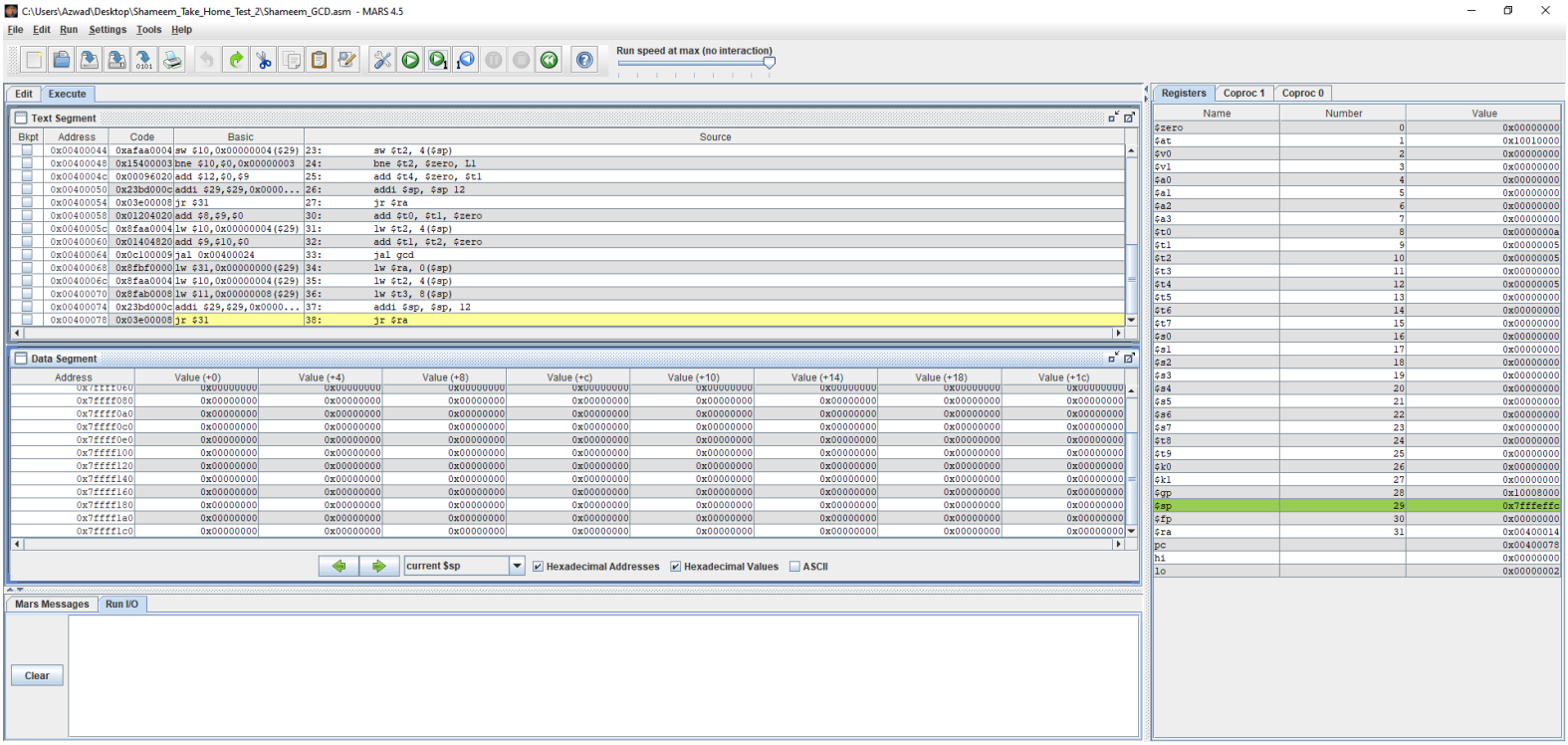

Figure 36: Shameem_GCD.asm code assembled in MARS (Line 37)

For Figure 33, Figure 34, Figure 35, Figure 36

Text Segment: This window shows that MARS has ran lines 34-37.

Registers: This window shows that the $sp register is now 0x7FFFEFC. The window also shows that $t3 is now 0x00000000 and $ra is now 0x00400014.

Data Segment: This window shows no change since figure 29.

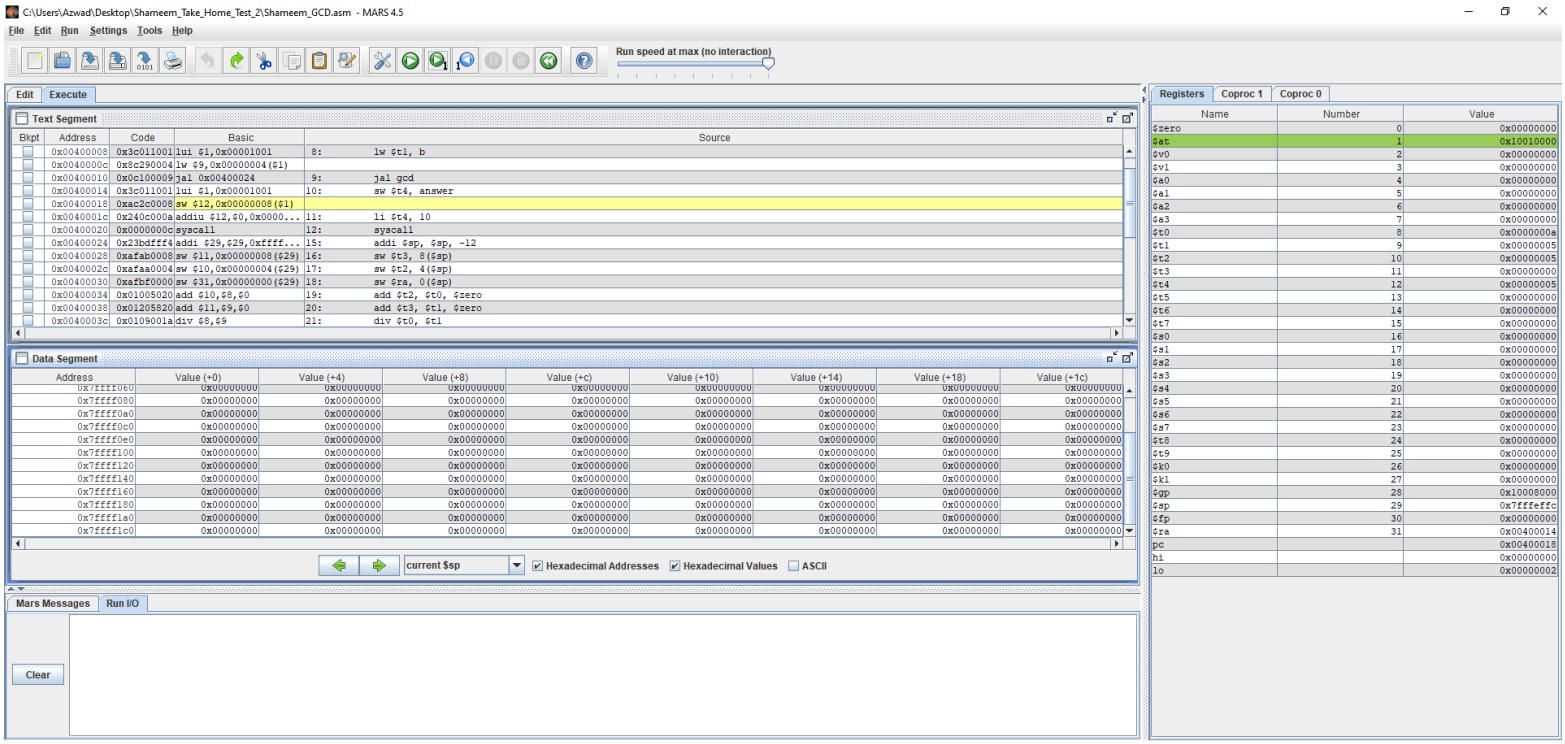

Figure 37: Shameem_GCD.asm code assembled in MARS (jumped to Line 10)

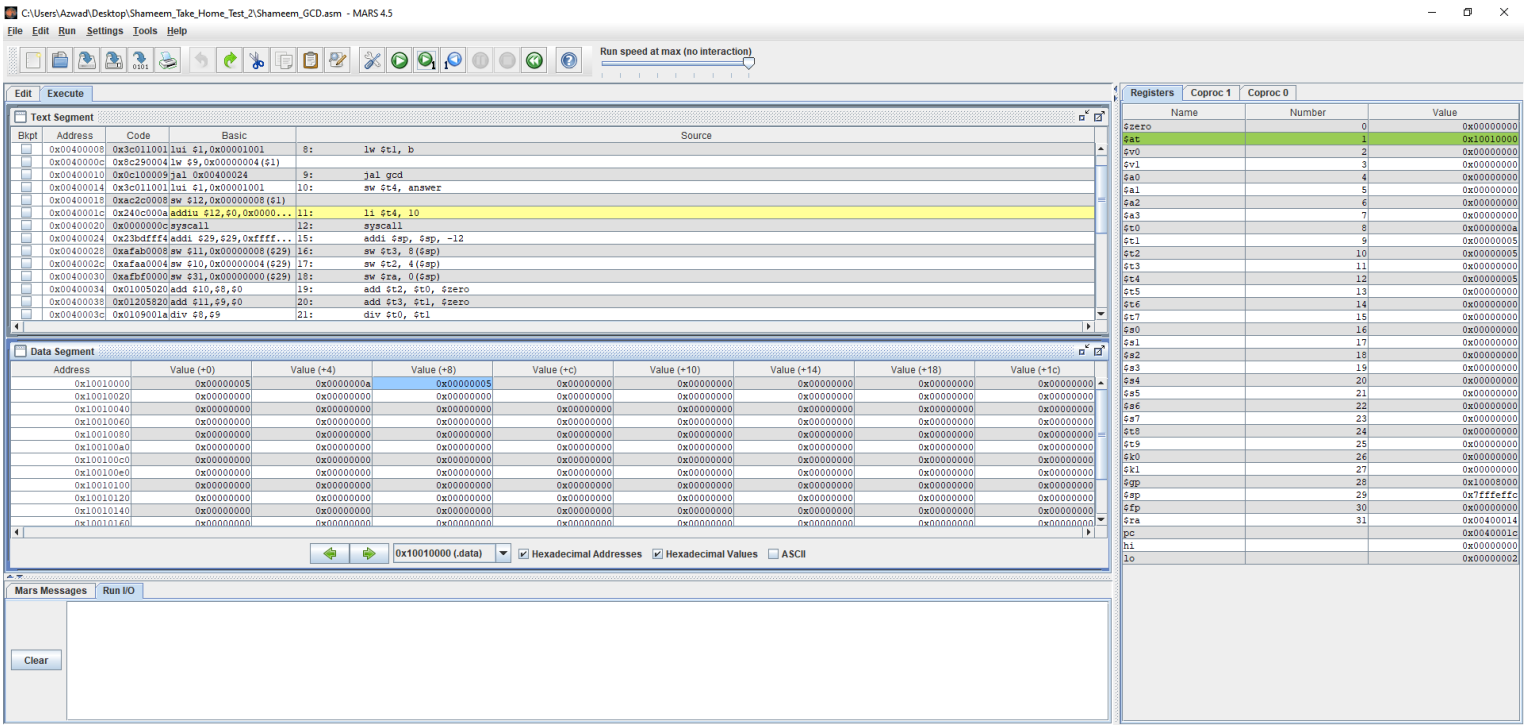

Figure 38: Shameem_GCD.asm code assembled in MARS (Line 10)

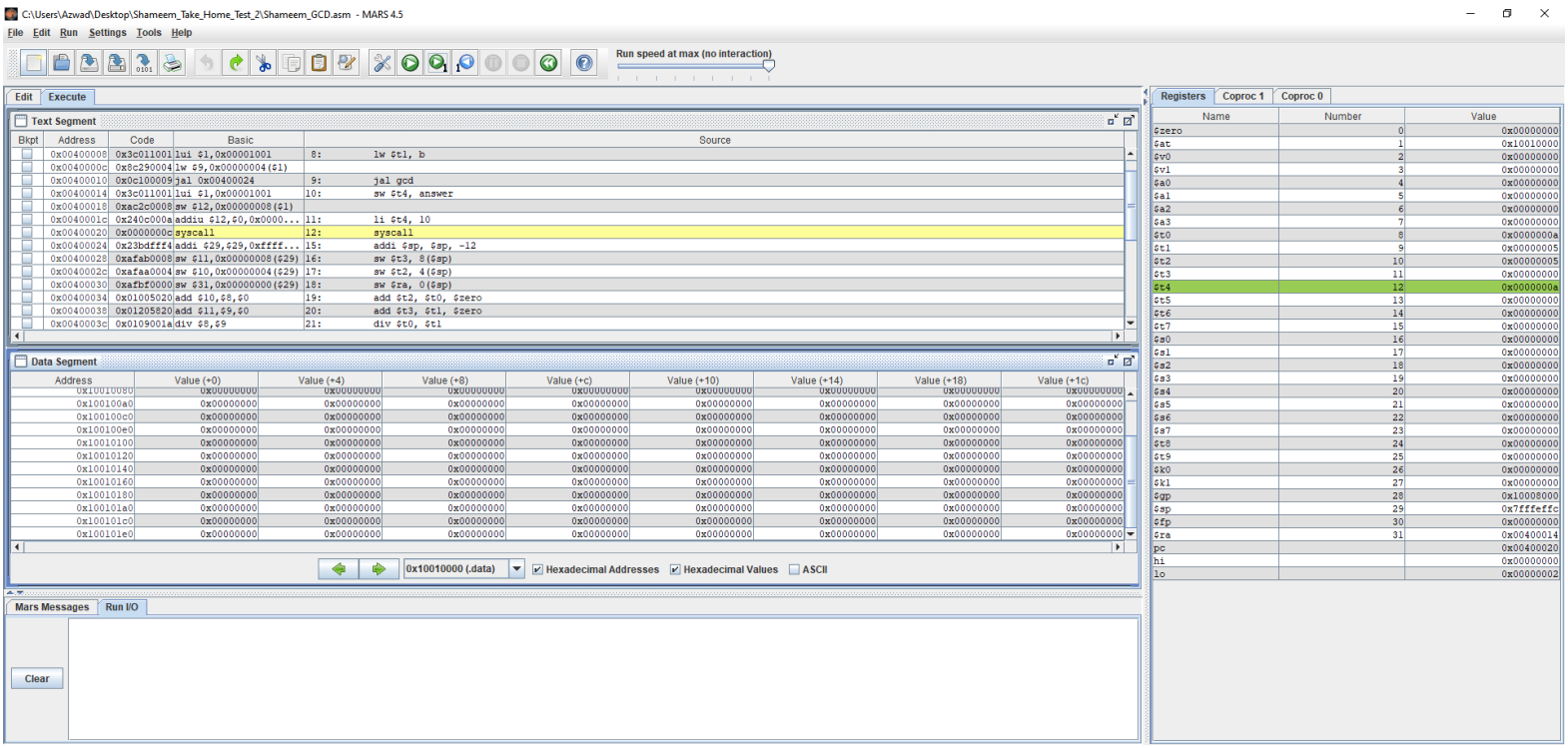

Figure 39: Shameem_GCD.asm code assembled in MARS (line 11)

For Figure 37, Figure 38, Figure 39

Text Segment: This window shows that MARS has ran lines 10-12 and exited.

Registers: This window shows that the $t4 register's value is now 0x0000000A.

Data Segment: This window shows the answer is stored in 0x10100008 with the value of 0x00000005.

Intel X32 ISA Windows 32-bit:

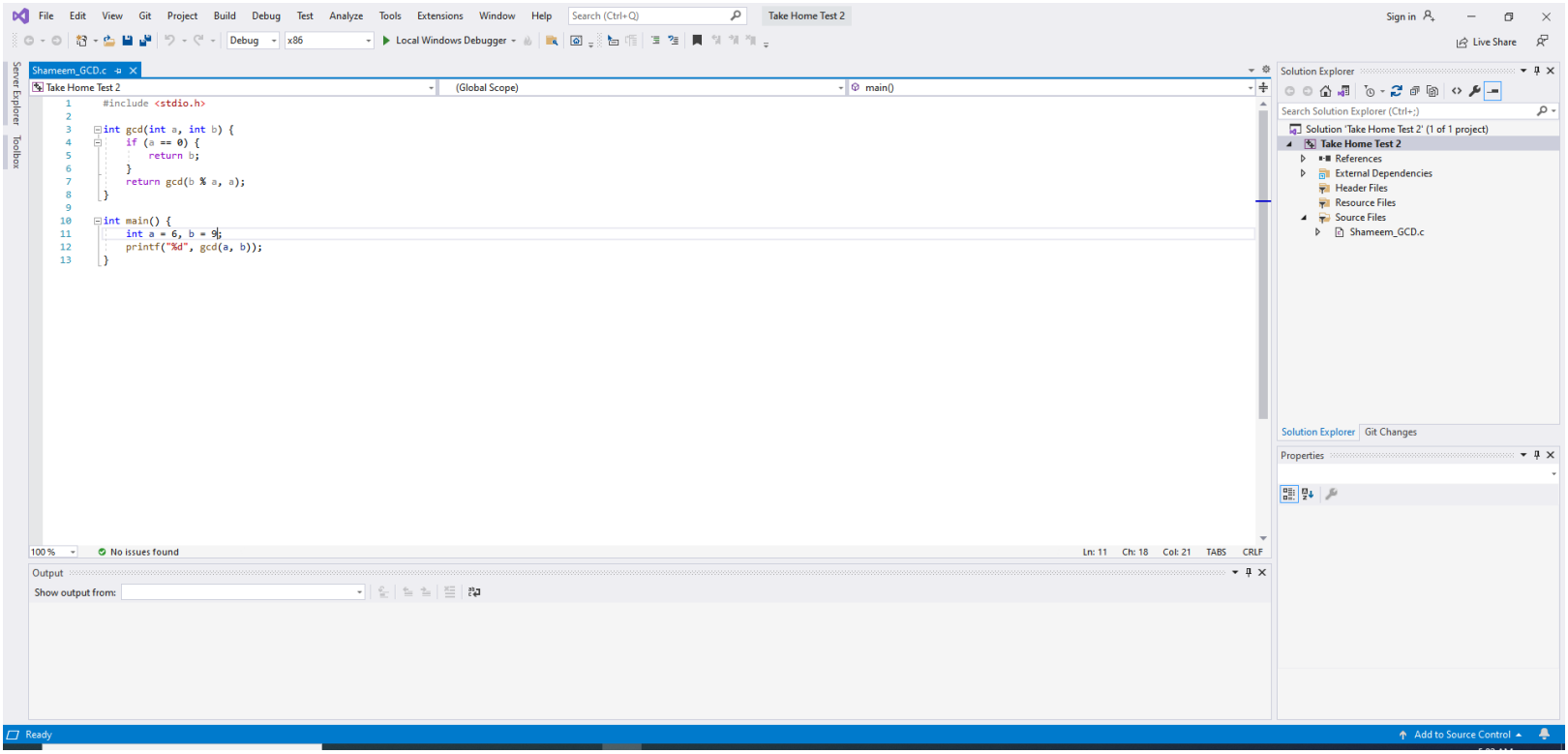

Figure 40: Shameem_GCD.c code in Visual Studio

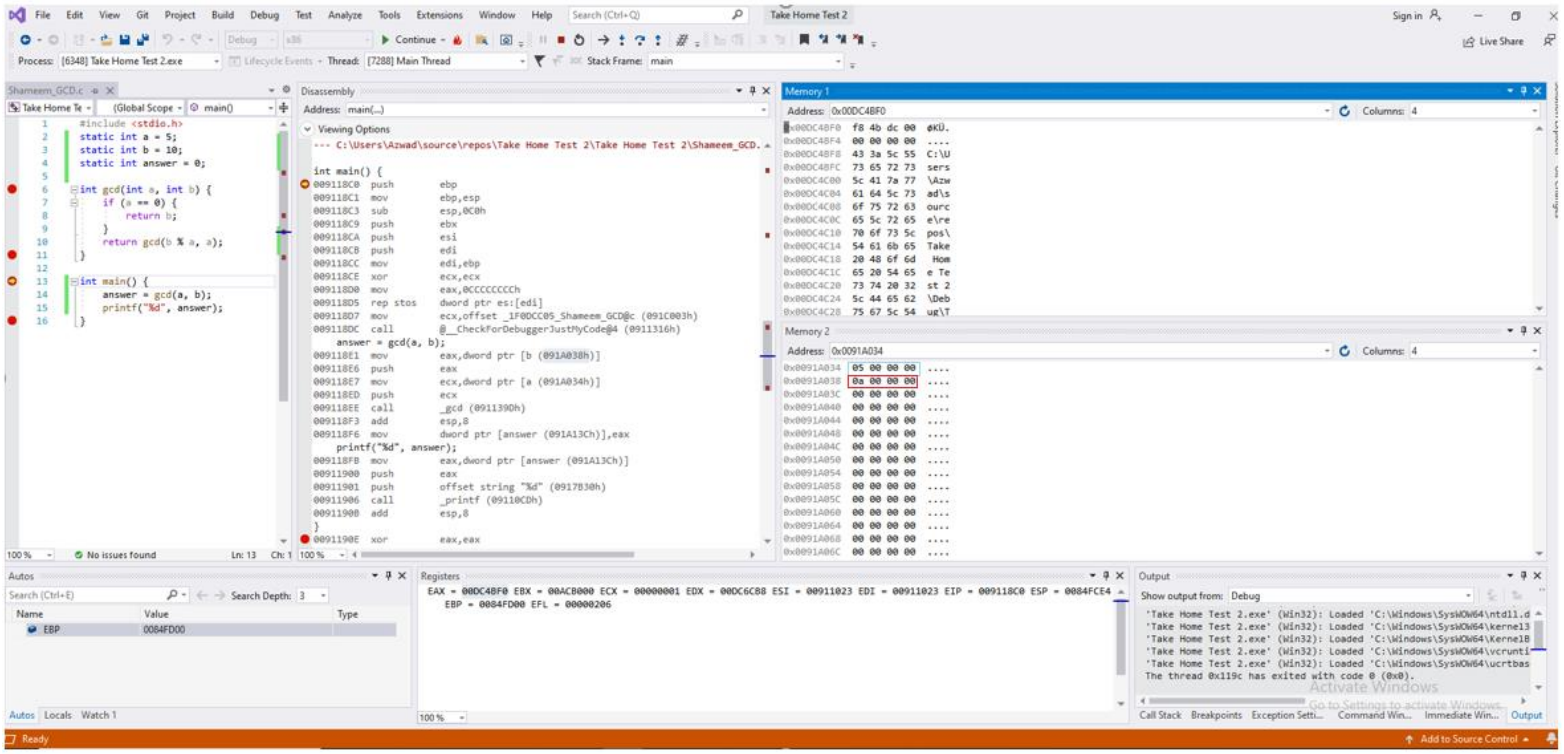

Figure 41: Shameem_GCD.c code in Visual Studio

Disassembly: This window shows that the code starts at line 12.

Registers: This window shows that the value of ESP, the stack pointer, is 0x00084FCE4.

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 0x0091A034 and has the value 05 00 00 00.

Variable b is stored in address 0x0091A038 and has the value 0a 00 00 00.

Variable answer is stored in address 0x0091A03C and has the value 00 00 00 00.

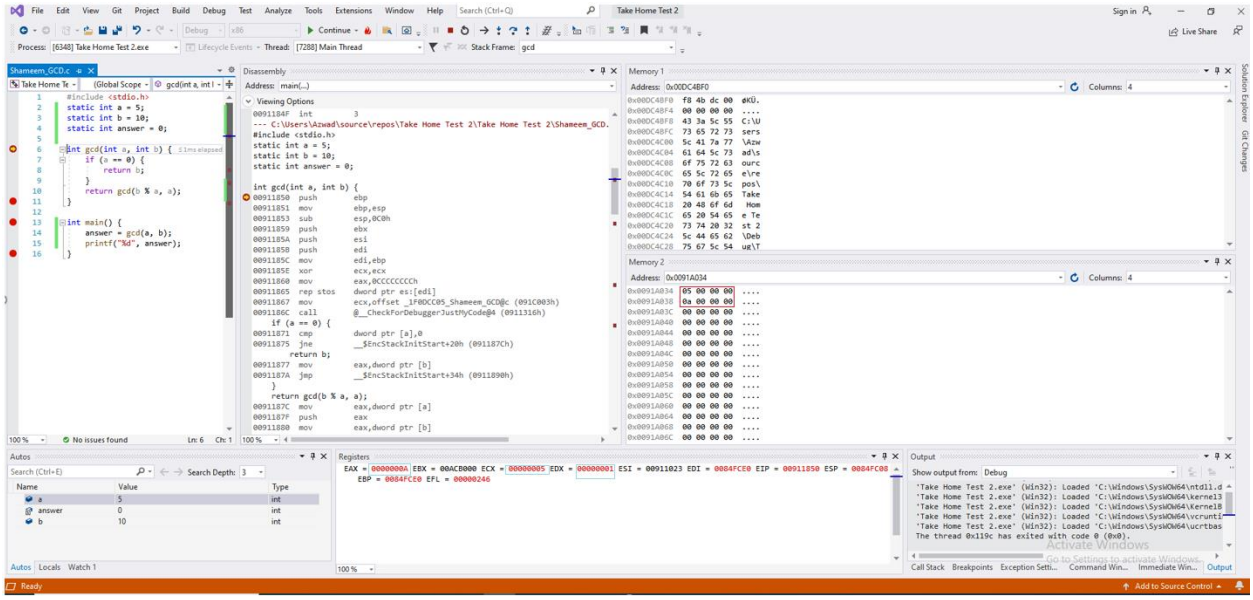

Figure 42: Shameem_GCD.c code in Visual Studio

Disassembly: This window shows that the code runs line 14 which calls the gcd() function at line 6.

Registers: This window shows that the value of EAX is now the value of variable b, which is 0x0000000A, and the value of ECX is now the value of variable a, which is 0x00000005. This means the value of the variables are copied into the register to do computation.

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 in address 0x0091A034 and has the value 05 00 00 00.

Variable b is stored in address 0x0091A038 and has the value 0a 00 00 00.

Variable answer is stored in address 0x0091A03C and has the value 00 00 00 00.

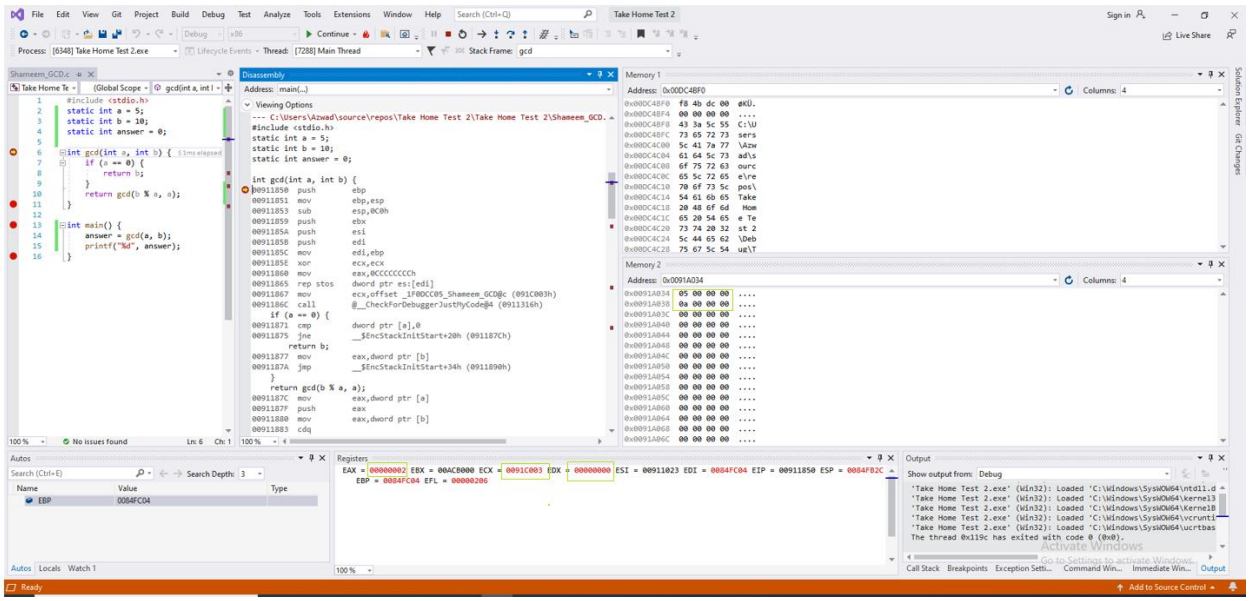

Figure 43: Shameem_GCD.c code in Visual Studio

Disassembly: This window shows that the code runs line 14 which calls the gcd() function at line 6.

Registers: This window shows that the value of EAX is now the value of variable b, which is 0x00000002, and the value of ECX is now the value 0x0091C003.

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 in address 0x0091A034 and has the value 05 00 00 00.

Variable b is stored in address 0x0091A038 and has the value 0a 00 00 00.

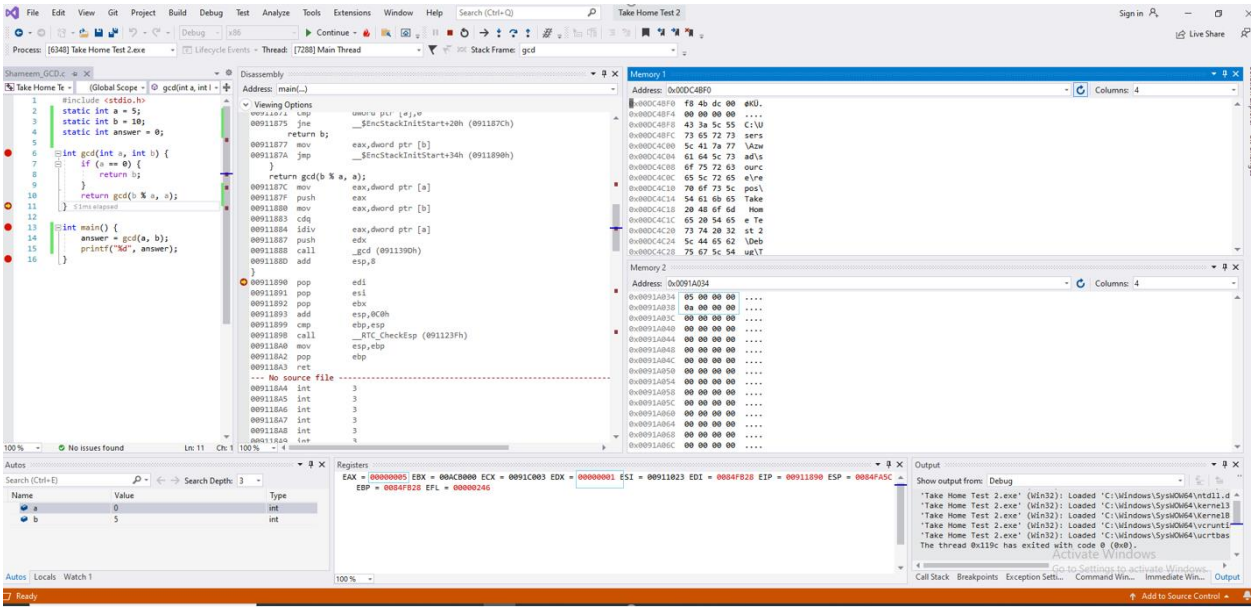

Figure 44: Shameem_GCD.c code in Visual Studio

Disassembly: This window shows that the code runs line 6-11.

Registers: This window shows that the value of EAX is now the value of variable b, which is 0x00000005, and the value of EDX is now the value 0x00000001.

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 in address 0x0091A034 and has the value 05 00 00 00.

Variable b is stored in address 0x0091A038 and has the value 0a 00 00 00.

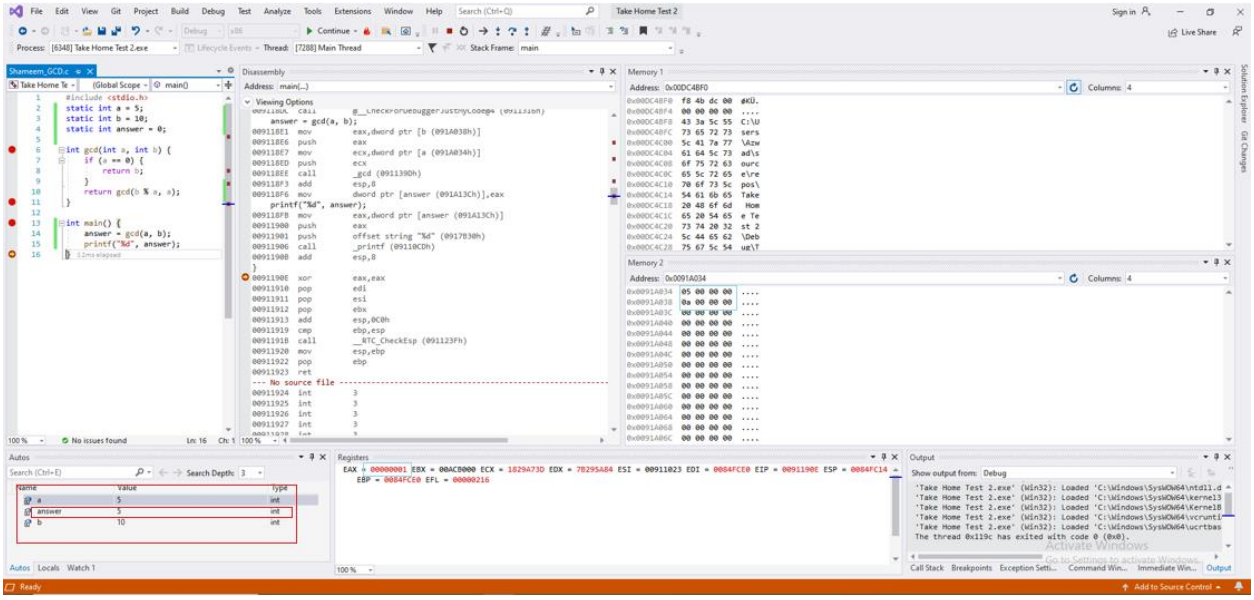

Figure 45: Shameem_GCD.c code in Visual Studio

Disassembly: This window shows that the code runs line 15.

Registers: This window shows that the value of EAX is now the value of variable b, which is 0x00000001, and the value of the other registers have been updated.

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 in address 0x0091A034 and has the value 05 00 00 00.

Variable b is stored in address 0x0091A038 and has the value 0a 00 00 00.

Also notice how the value of answer in the bottom left shows 5, which means the answer is printed out.

X86_64 ISA Linux 64-bit:

```
(gdb) list
1       #include <stdio.h>
2       static int a = 5;
3       static int b = 10;
4       static int answer = 0;
5
6       int gcd(int a, int b) {
7               if (a == 0) {
8                       return b;
9               }
10              return gcd(b % a, a);
(gdb)
11      }
12
13      int main() {
14              answer = gcd(a, b);
15              printf("%d", answer);
16      }
(gdb)
```

Figure 46: Shameem_GCD.c code in GDB

```
(gdb) disassemble
Dump of assembler code for function main:
   0x0000555555555163 <+0>:     push   %rbp
   0x0000555555555164 <+1>:     mov    %rsp,%rbp
=> 0x0000555555555167 <+4>:     mov    0x2ec7(%rip),%edx        # 0x555555558034 <b>
   0x000055555555516d <+10>:    mov    0x2ebd(%rip),%eax        # 0x555555558030 <a>
   0x0000555555555173 <+16>:    mov    %edx,%esi
   0x0000555555555175 <+18>:    mov    %eax,%edi
   0x0000555555555177 <+20>:    call   0x555555555135 <gcd>
   0x000055555555517c <+25>:    mov    %eax,0x2eba(%rip)        # 0x55555555803c <answer>
   0x0000555555555182 <+31>:    mov    0x2eb4(%rip),%eax        # 0x55555555803c <answer>
   0x0000555555555188 <+37>:    mov    %eax,%esi
   0x000055555555518a <+39>:    lea    0xe73(%rip),%rdi        # 0x555555556004
   0x0000555555555191 <+46>:    mov    $0x0,%eax
   0x0000555555555196 <+51>:    call   0x555555555030 <printf@plt>
   0x000055555555519b <+56>:    mov    $0x0,%eax
   0x00005555555551a0 <+61>:    pop    %rbp
   0x00005555555551a1 <+62>:    ret
End of assembler dump.
(gdb) print /x $rsp
$1 = 0x7fffffffe5a0
(gdb) print /x $rbp
$2 = 0x7fffffffe5a0
(gdb) info registers
rax            0x555555555163      93824992235875
rbx            0x0                 0
rcx            0x7ffff7fae718      140737353803544
rdx            0x7fffffffe6a8      140737488348840
rsi            0x7fffffffe698      140737488348824
rdi            0x1                 1
rbp            0x7fffffffe5a0      0x7fffffffe5a0
rsp            0x7fffffffe5a0      0x7fffffffe5a0
r8             0x0                 0
r9             0x7ffff7fe21b0      140737354015152
r10            0xf                 15
r11            0x2                 2
r12            0x555555555050      93824992235600
r13            0x0                 0
r14            0x0                 0
r15            0x0                 0
rip            0x555555555167      0x555555555167 <main+4>
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 47: Shameem_GCD.c code in GDB

Disassembly: This window shows that the code starts to run main().

Registers: This window shows the value that's stored $rsp which is 0x7fffffffe5a0 and the value that's stored in $rbp which is at 0x0.

.

```
Dump of assembler code for function main:
   0x0000555555555163 <+0>:     push   %rbp
   0x0000555555555164 <+1>:     mov    %rsp,%rbp
   0x0000555555555167 <+4>:     mov    0x2ec7(%rip),%edx        # 0x555555558034 <b>
   0x000055555555516d <+10>:    mov    0x2ebd(%rip),%eax        # 0x555555558030 <a>
   0x0000555555555173 <+16>:    mov    %edx,%esi
   0x0000555555555175 <+18>:    mov    %eax,%edi
   0x0000555555555177 <+20>:    call   0x555555555135 <gcd>
   0x000055555555517c <+25>:    mov    %eax,0x2eba(%rip)        # 0x55555555803c <answer>
=> 0x0000555555555182 <+31>:    mov    0x2eb4(%rip),%eax        # 0x55555555803c <answer>
   0x0000555555555188 <+37>:    mov    %eax,%esi
   0x000055555555518a <+39>:    lea    0xe73(%rip),%rdi        # 0x555555556004
   0x0000555555555191 <+46>:    mov    $0x0,%eax
   0x0000555555555196 <+51>:    call   0x555555555030 <printf@plt>
   0x000055555555519b <+56>:    mov    $0x0,%eax
   0x00005555555551a0 <+61>:    pop    %rbp
   0x00005555555551a1 <+62>:    ret
End of assembler dump.
(gdb) print /x $eax
$6 = 0x5
(gdb) print /x $edx
$7 = 0x0
(gdb)
```

Figure 48: Shameem_GCD.c code in GD

Registers: The value of a is stored into $eax. This is done to do computation of division with the value of b.

```
(gdb) disassemble
Dump of assembler code for function main:
   0x0000555555555163 <+0>:     push   %rbp
   0x0000555555555164 <+1>:     mov    %rsp,%rbp
   0x0000555555555167 <+4>:     mov    0x2ec7(%rip),%edx        # 0x555555558034 <b>
   0x000055555555516d <+10>:    mov    0x2ebd(%rip),%eax        # 0x555555558030 <a>
   0x0000555555555173 <+16>:    mov    %edx,%esi
   0x0000555555555175 <+18>:    mov    %eax,%edi
   0x0000555555555177 <+20>:    call   0x555555555135 <gcd>
   0x000055555555517c <+25>:    mov    %eax,0x2eba(%rip)        # 0x55555555803c <answer>
=> 0x0000555555555182 <+31>:    mov    0x2eb4(%rip),%eax        # 0x55555555803c <answer>
   0x0000555555555188 <+37>:    mov    %eax,%esi
   0x000055555555518a <+39>:    lea    0xe73(%rip),%rdi        # 0x555555556004
   0x0000555555555191 <+46>:    mov    $0x0,%eax
   0x0000555555555196 <+51>:    call   0x555555555030 <printf@plt>
   0x000055555555519b <+56>:    mov    $0x0,%eax
   0x00005555555551a0 <+61>:    pop    %rbp
   0x00005555555551a1 <+62>:    ret
End of assembler dump.
(gdb) x/8xb 0x555555558034
0x555555558034 <b>:     0x0a    0x00    0x00    0x00    0x00    0x00    0x00    0x00
(gdb) x/8xb 0x555555558030
0x555555558030 <a>:     0x05    0x00    0x00    0x00    0x0a    0x00    0x00    0x00
(gdb) x/8xb 0x55555555803c
0x55555555803c <answer>:        0x05    0x00    0x00    0x00    0x00    0x00    0x00    0x00
(gdb)
```

Figure 49: Shameem_GCD.c code in GDB

Memory: This shows the address 0x555555558034 which has the value 0x0a 0x00 0x00 0x00 0x00 0x00 0x00 0x00 which is the variable b. Also, shows the address 0x555555558034 which has the value 0x05 0x000x00 0x00 0x00 0x00 0x00 0x00 which is the value of variable a. Lastly, shows the address 0x555555558034 which has the value 0x05 0x000x00 0x00 0x00 0x00 0x00 0x00, which is the value of the answer.

Conclusions

In conclusion the Take Home Test helped us to test, understand and demonstrate the recursive calls and stack frames in the three different architecture MARS Simulator, Visual Studio and Linux GDB and GCC. Therefore, by explaining the recursive steps of the GCD(a, b) function, we can gain a good understanding how the function works in three different architectures, MIPS on MARS Simulator, Intel x86 using Windows MS 32-bit compiler on Visual Studio and Intel X86_64 bit on Linux GDB and GCC. After explaining the recursive GCD(a, b) function we understand the difference in how the function runs step by step in the different architectures.