

**Purpose**

The purpose of lab 3 was to further grasp a familiarity of MIPS instructions and registers. The lab included two tasks. The first task was with regards to how C++ can be translated into MIPS. The second task included translation of a C++ factorial algorithm into MIPS. The provided practice with assembly language programming and required variable initialization, arithmetic, and a while loop for a factorial computation. Additionally, the assignment provided an introduction to MIPS ISA control structures and how to use the $HI and $LO registers. Furthermore, the lab allowed practice with verifying the values within registers and provided further a foundation of how to use the MARS Tool.

**Approach**

Each task was approached initially from a foundation of C++. After understanding the C++ implementation, each line was translated into MIPS assembly as depicted in Appendix A for both tasks. After single-stepping for each instruction, the register values were verified. Table 1 in Appendix B depicts the output results of Task 1. Meanwhile, the test log in Table 2 of Appendix B depicts the results for Task 2. Each task was accomplished successfully.

For task 1, the goal was to convert given C++ code to MIPS assembly, while keeping the number of instructions under 28. We completed this task successfully while keeping our number of instructions to 26. We accomplished this by taking each line of C++ at a time and writing MIPS instructions to accomplish what that line would perform in C++. We also made every effort to avoid using pseudo-instructions while also staying under the 28 line limit. For example, instead of using *mul Rd, Rs, Rt* we used *mult Rt, Rs.* This allowed us to learn how to use the $HI and $LO registers. Figures 1 and 2 in Appendix C show verification of the accomplished task.

For task 2, we assembled our code and single-stepped through the execution. Table 1 in Appendix B shows our code and how each register is updated after each instruction is executed. Table 2 in Appendix B also shows the contents of the specified memory locations.

For task 3, the goal was to write a MIPS assembly program to calculate the factorial of an integer n. We were given the pseudocode for the factorial algorithm. Using this and the knowledge of how to design a while loop from task 1, we successfully wrote a program that performs the same function as the given pseudocode. Using n = 5 as the input integer, we calculated the output f as 120. Figures 3 and 4 in Appendix C show verification of the accomplished task.

For task 4, we repeated the same steps as in task 2 and single-stepped through the execution of the assembled code and recorded the results in a test log. This test log is visible in Table 3 of Appendix B.

**Accomplished Tasks**

*1. Programmed MIPS assembly for performing an arithmetic computation*

*2. Recorded output results of the arithmetic calculation*

*3. Programmed MIPS assembly or a factorial calculation*

*4. Recorded output results of the factorial calculation*

**Conclusion**

The purpose of the lab was fulfilled successfully with all tasks accomplished. A foundation of how to program MIPS assembly was further built upon. In Assignment 3, we learned how to translate C++ into MIPS assembly language. The first task was accomplished with initialization of variables, arithmetic, and a while loop. Meanwhile, the second task was accomplished with a factorial computation. We learned bit shifting with MIPS instructions, and further about how different instructions can rely upon other instructions to store into the relevant registers. Rather than relying only on mul and div instructions to store the results to the ideal register, we included mflo to provide the result into the ideal register. Furthermore, we learned how to truncate the amount of instructions for an ideal length of assembly code. The lab was accomplished successfully.

### Appendix A: MIPS Assembly Code

|  |
| --- |
| **Assignment3Task1.asm** |
| main: ori $4, $0, 0x8000 # a = 0x8000  ori $5, $0, 0x00A9 # b = 0x00A9  ori $16, $0, 1974 # c = 1974    mult $4, $4 # x = a \* a  mflo $17 # Move lower 32 bits of mult into x  sw $17, 0x20($0) # Store x into 0x20  mult $17, $5 # y = x \* b  mflo $18 # Move lower 32 bits of mult into y  sw $18, 0x24($0) # Store y into 0x24  mfhi $19 # Move upper 32 bits of mult (x \* b) into temp (s3)  sw $19, 0x28($0) # Store temp into 0x28 (0x24 is lower 32 bits  of y = x \* b, 0x28 is upper 32 bits)  sll $19, $19, 16 # Shift upper 32 bits of y left 16 bits  srl $18, $18, 16 # Shift lower 32 bits of y right 16 bits  or $18, $18, $19 # Goal: y = 0x002a4000    # Performing c = (c + y / c) / 2  loop: div $18, $16 # First, y / c  mflo $19 # Store quotient from LO to temp  add $19, $19, $16 # Second, c + temp  li $20, 2 # Load 2 for division  div $19, $20 # Divide temp by 2  mflo $16 # Store quotient from LOW in c  sw $16, 0x2C($0) # Store c into 0x2C  li $19, 1665 # Load 1665 for comparison  slt $20, $19, $16 # Set s4 to 1 if c >= 1665, else 0  bne $20, $0, loop # Branch if s4 not equal to 0  e\_loop:sll $16, $16, 8 # Shift c left by 8 bits  sw $16, 0x30($0) # Save c to 0x30 |

### 

|  |
| --- |
| **Assignment3Task2.asm** |
| main: ori $4, $0, 5 # Initialize n = 5 ($4 = 0 OR 5)  sw $4, ($0) # Save n to memory location 0  ori $16, $0, 1 # Initialize f = 1  loop: li $19, 1 # Load 1 for comparison and for decrementing n  mult $16, $4 # Multiply f \* n  mflo $16 # Move calculated value from LO to f  sub $4, $4, $19 # n - 1  slt $20, $19, $4 # Set s4 to 1 if n >= 1, else store 0 into 20  bne $20, $0, loop # Branch to loop if s4 != 0 (for while loop)  sw $16, 0x10($0) # Store final value of f into memory location 10  (f = n!) |

### Appendix B: Test Logs

Table 1. Test Log for Task 1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Adr | MIPS Instruction | Machine Code | **Registers (in hex)** | | | | |
| $a0 | $a1 | $s0 | $s1 | $s2 |
| 3000 | ori $4, $0, 0x8000 | 34048000 | 8000 | 0 | 0 | 0 | 0 |
| 3004 | ori $5, $0, 0x00A9 | 340500a9 | 8000 | a9 | 0 | 0 | 0 |
| 3008 | ori $16, $0, 1974 | 341007b6 | 8000 | a9 | 7b6 | 0 | 0 |
| 300c | mult $4, $4 | 00840018 | 8000 | a9 | 7b6 | 0 | 0 |
| 3010 | mflo $17 | 00008812 | 8000 | a9 | 7b6 | 4000  0000 | 0 |
| 3014 | sw $17, 0x20($0) | ac110020 | 8000 | a9 | 7b6 | 4000  0000 | 0 |
| 3018 | mult $17, $5 | 02250018 | 8000 | a9 | 7b6 | 4000  0000 | 0 |
| 301c | mflo $18 | 00009012 | 8000 | a9 | 7b6 | 4000  0000 | 4000  0000 |
| 3020 | sw $18, 0x24($0) | ac120024 | 8000 | a9 | 7b6 | 4000  0000 | 4000  0000 |
| 3024 | mfhi $19 | 00009810 | 8000 | a9 | 7b6 | 4000  0000 | 4000  0000 |
| 3028 | sw $19, 0x28($0) | ac130028 | 8000 | a9 | 7b6 | 4000  0000 | 4000  0000 |
| 302c | sll $19, $19, 16 | 00139c00 | 8000 | a9 | 7b6 | 4000  0000 | 4000  0000 |
| 3030 | srl $18, $18, $19 | 00129402 | 8000 | a9 | 7b6 | 4000  0000 | 4000 |
| 3034 | or $18, $18, $19 | 02539025 | 8000 | a9 | 7b6 | 4000  0000 | 2a4000 |
| 3038 | div $18, $16 | 0250001a | 8000 | a9 | 7b6 | 4000  0000 | 2a4000 |
| 303c | mflo $19 | 00009812 | 8000 | a9 | 7b6 | 4000  0000 | 2a4000 |
| 3040 | add $19, $19, $16 | 02709820 | 8000 | a9 | 7b6 | 4000  0000 | 2a4000 |
| 3044 | addiu $20, $0, 2 | 24140002 | 8000 | a9 | 7b6 | 4000  0000 | 2a4000 |
| 3048 | div $19, $20 | 0274001a | 8000 | a9 | 7b6 | 4000  0000 | 2a4000 |
| 304c | mflo $16 | 00008012 | 8000 | a9 | 698 | 4000  0000 | 2a4000 |
| 3050 | sw $16, 44($0) | ac10002c | 8000 | a9 | 698 | 4000  0000 | 2a4000 |
| 3054 | addiu $19, $0, 1665 | 24130681 | 8000 | a9 | 698 | 4000  0000 | 2a4000 |
| 3058 | slt $20, $19, $16 | 270a02a | 8000 | a9 | 698 | 4000  0000 | 2a4000 |
| 305C | bne $20, $0, -10 | 1680fff6 | 8000 | a9 | 698 | 4000  0000 | 2a4000 |
| 3060 | sll $16, $16, 8 | 108200 | 8000 | a9 | 68000 | 4000  0000 | 2a4000 |
| 3064 | sw $16, 48($0) | ac100030 | 8000 | a9 | 68000 | 4000  0000 | 2a4000 |
| 3068 |  |  |  |  |  |  |  |
| 306C |  |  |  |  |  |  |  |

Table 2. Final Memory Contents of Task 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Memory contents** | | | | |
| Word @ 0x20 | Word @ 0x24 | Word @ 0x28 | Word @ 0x2C | Word @ 0x30 |
| 40000000 | 40000000 | 0000002a | 00000680 | 00068000 |

# Table 3: Test Log for Task 3

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Adr | MIPS Instruction | Machine Code | Registers (in hex) | | | | Memory Content (in hex) | |
| **$a0** | **$s0** | **$s3** | **$s4** | Word @ 0x00 | Word @ 0x10 |
| 3000 | main: ori $4, $0, 5 | 34040005 | 5 | 0 | 0 | 0 | 0 | 0 |
| 3004 | sw $4, ($0) | ac040000 | 5 | 0 | 0 | 0 | 5 | 0 |
| 3008 | ori $16, $0, 1 | 34100001 | 5 | 1 | 0 | 0 | 5 | 0 |
| 300c | loop: li $19, 1 | 24130001 | 5 | 1 | 1 | 0 | 5 | 0 |
| 3010 | mult $16, $4 | 02040018 | 5 | 1 | 1 | 0 | 5 | 0 |
| 3014 | mflo $16 | 00008012 | 5 | 5 | 1 | 0 | 5 | 0 |
| 3018 | sub $4, $4, $19 | 00932022 | 4 | 5 | 1 | 0 | 5 | 0 |
| 301c | slt $20, $19, $4 | 0264a02a | 4 | 5 | 1 | 1 | 5 | 0 |
| 3020 | bne $20, $0, loop | 1680fffa | 4 | 5 | 1 | 1 | 5 | 0 |
| 3024 | sw $16, 0x10($0) | ac100010 | 1 | 78 | 1 | 0 | 5 | 78 |
| 3028 |  |  |  |  |  |  |  |  |

### Appendix C: Screenshots

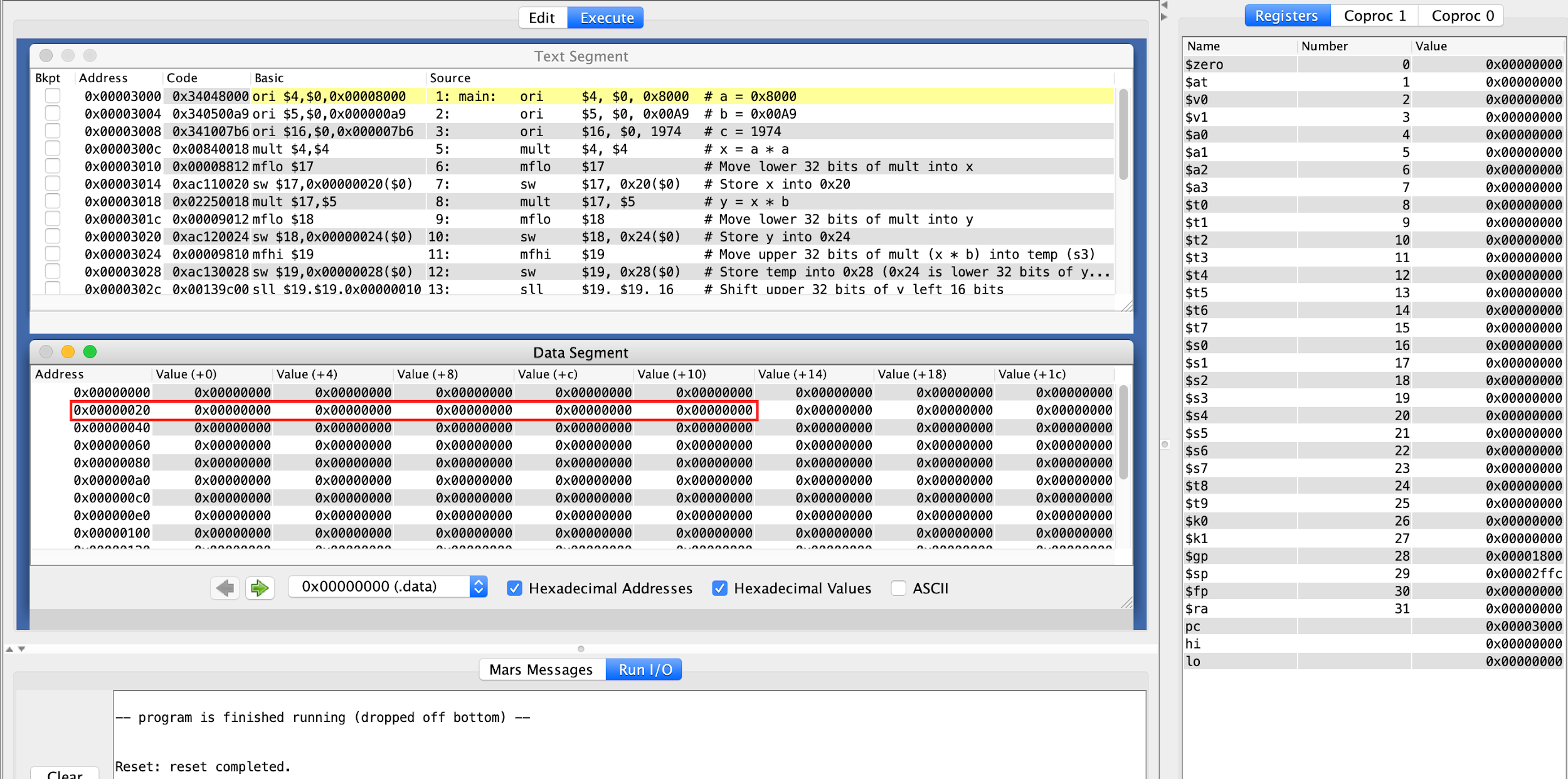


Figure 1. Screenshot showing assembled Task 1 code. The code has not been run yet.

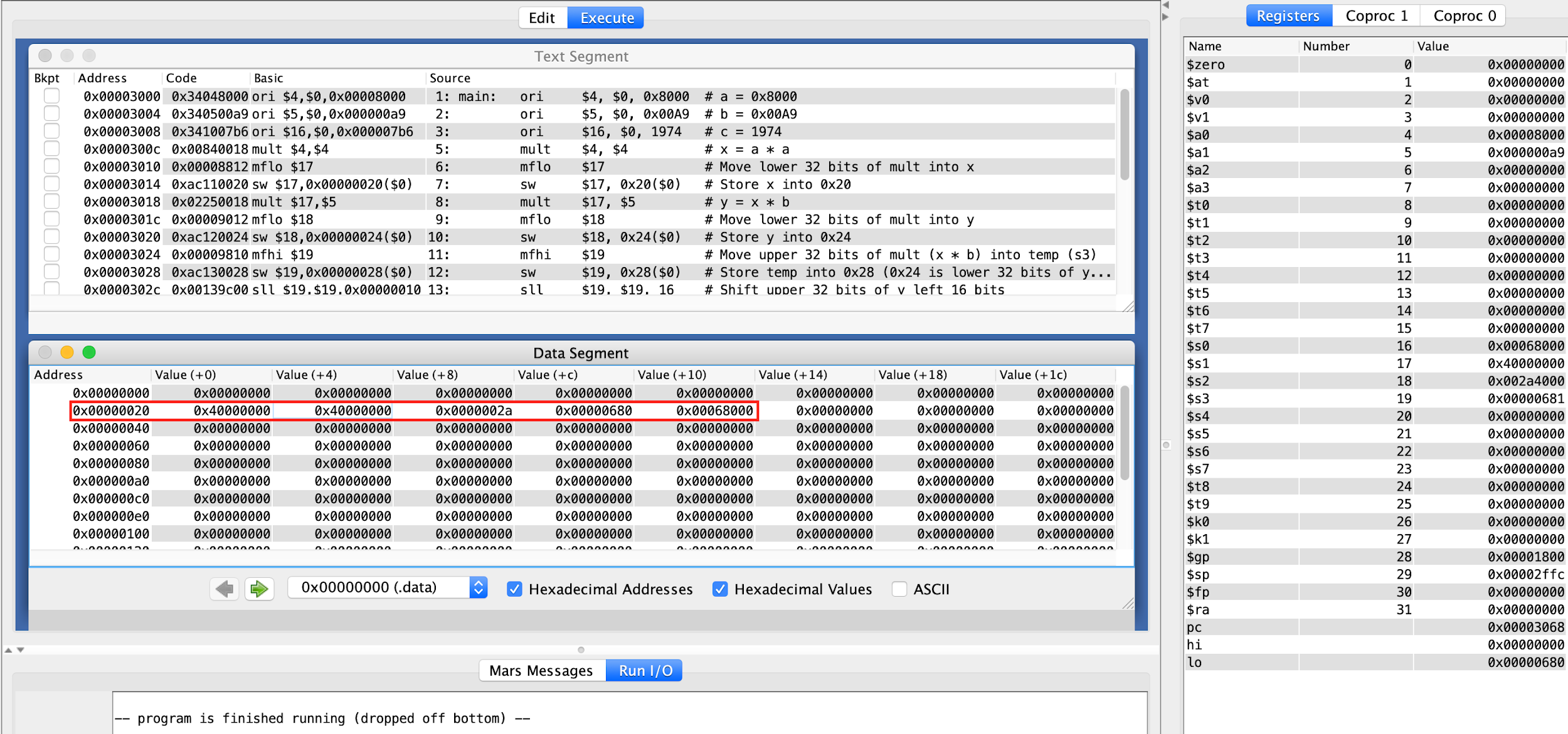


Figure 2. Screenshot showing contents of memory addresses 0x20-0x23, 0x24-0x27, 0x28-0x2b, 0x2c-0x2f, 0x30-0x33 after running assembled Task 1 code.

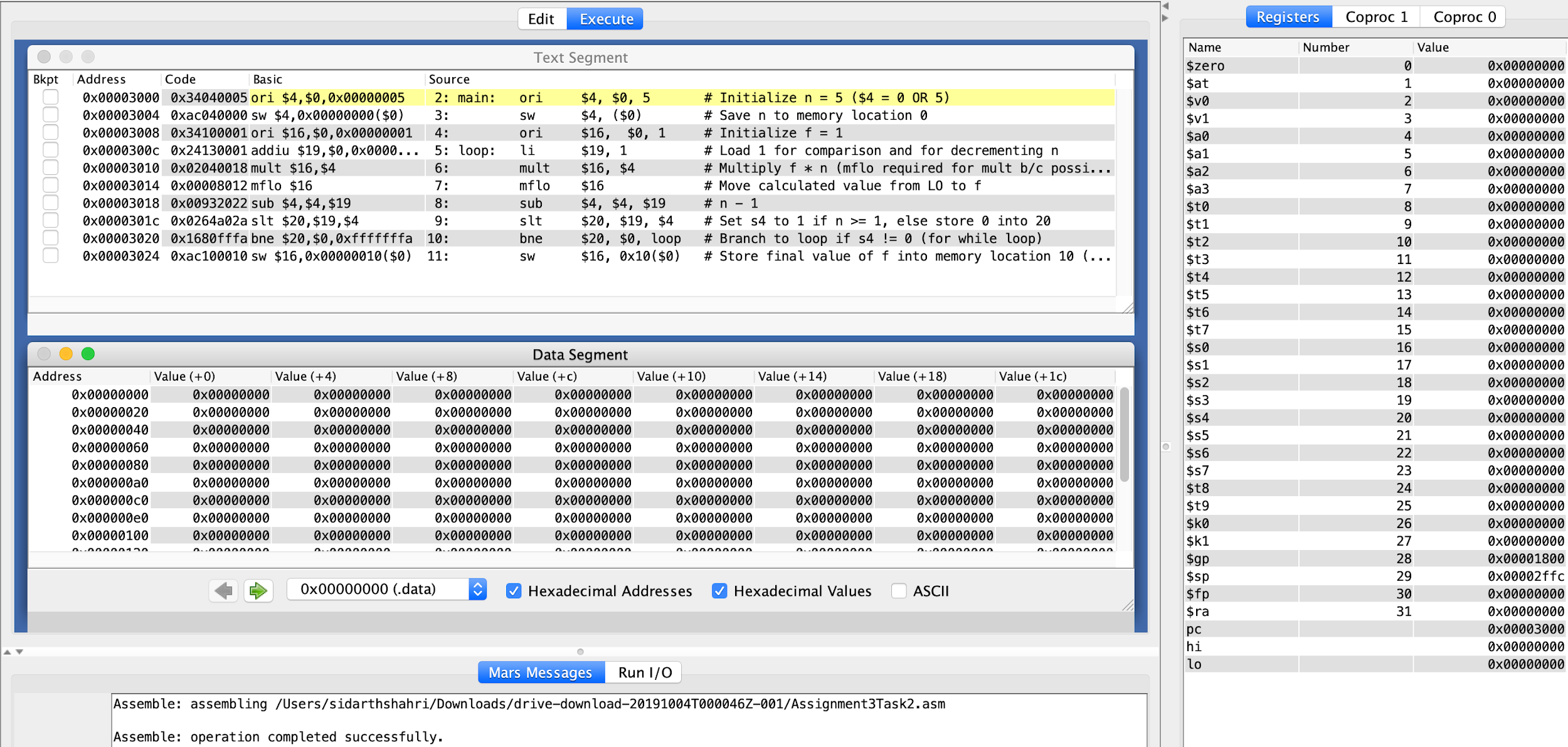


Figure 3. Screenshot showing assembled Task 3 code. The code has not been run yet.

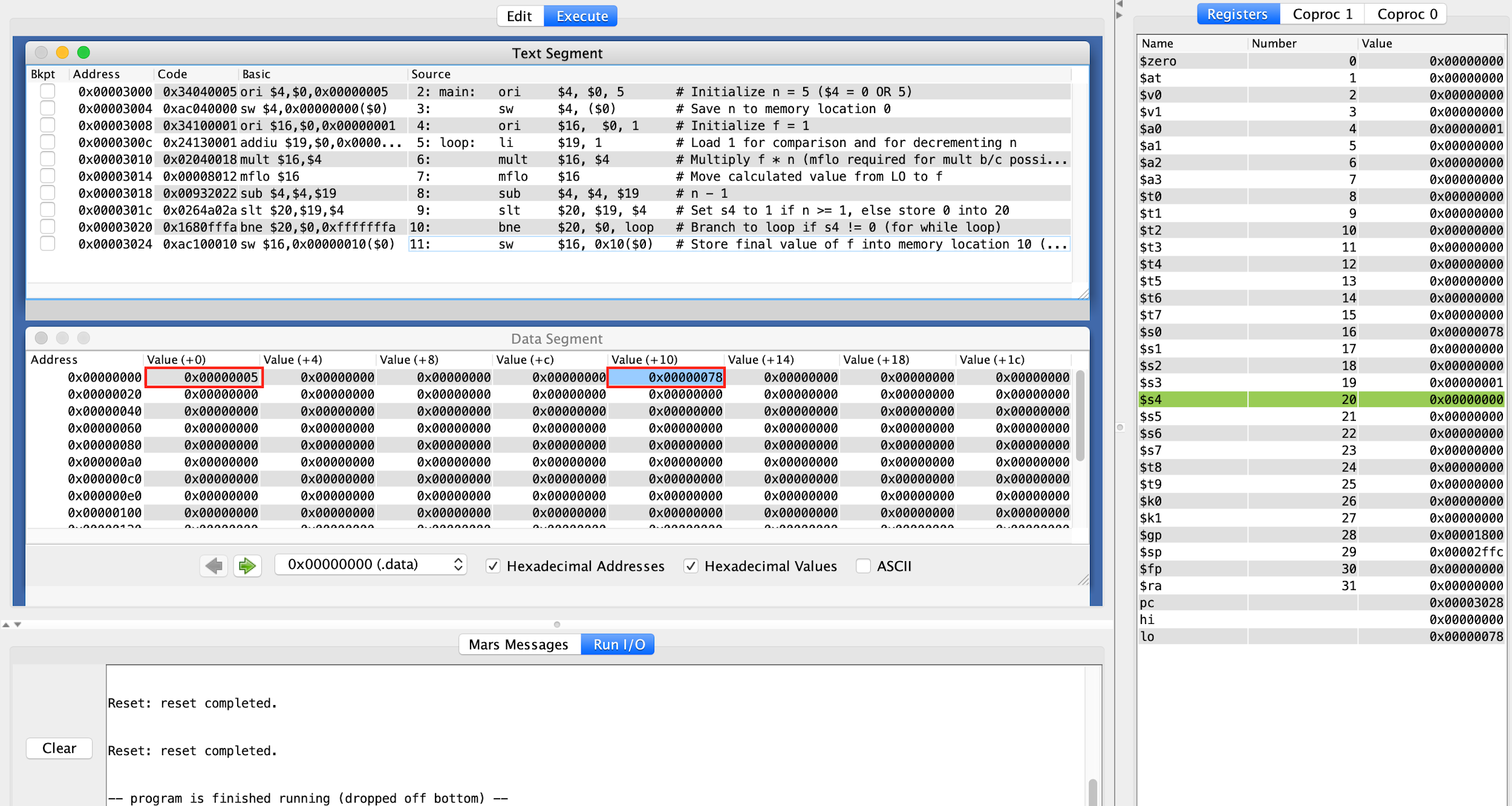


Figure 4. Screenshot depicting contents of memory addresses 0x00-0x03, 0x10-0x13 after running assembled Task 3 code.