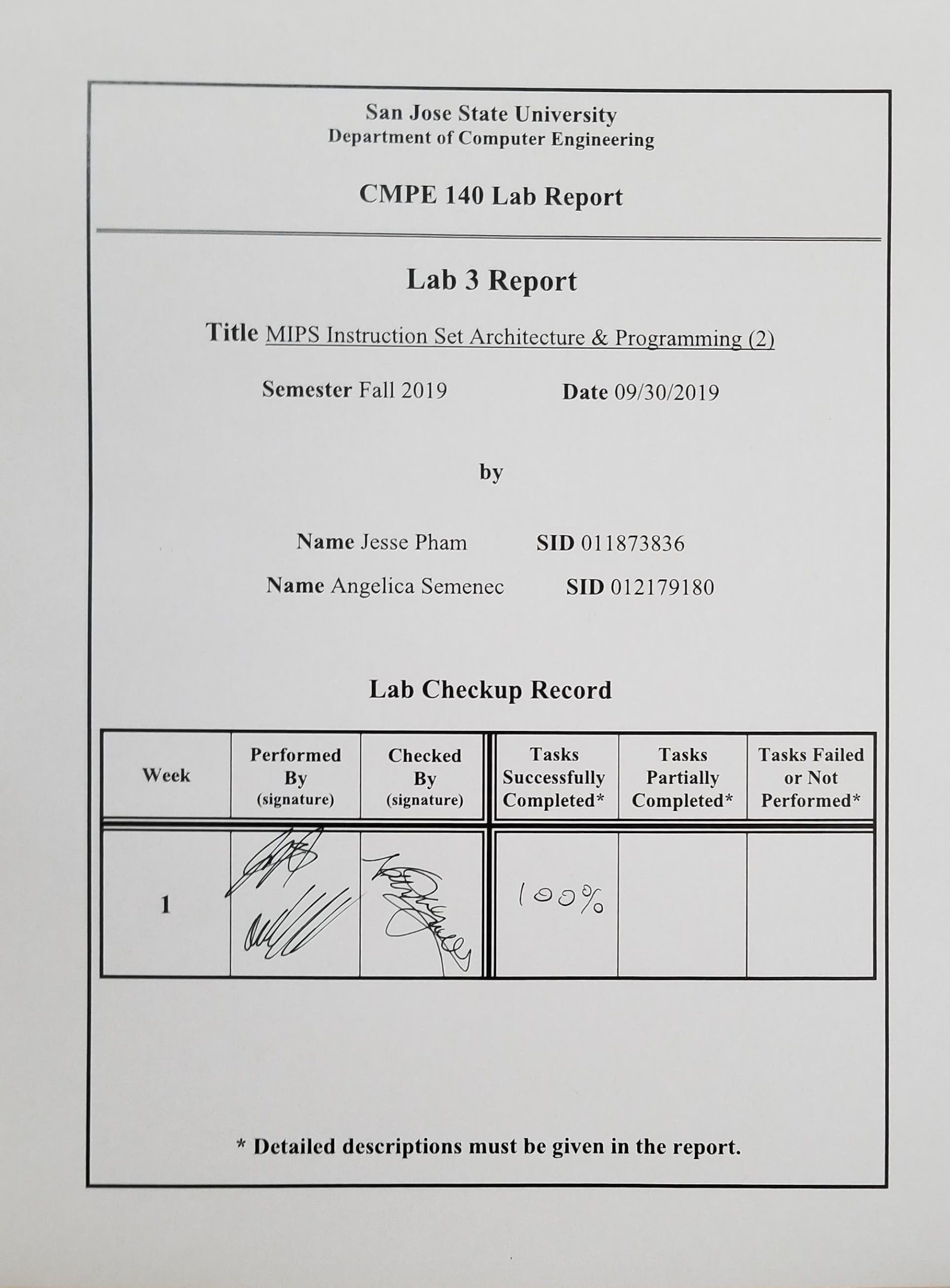
**Purpose**

The purpose of this lab is to become familiar with creating a program using the MIPS instruction set architecture by using mathematical operations, loops, and storage to memory in the first part of the assignment and to create a program that can calculate the factorial of a given value in the second part of the assignment by using a while loop.

**Design Methodology**

For the first part of the assignment, the program instructions were provided to follow and create using the MIPS language. One constraint was to use no more than 28 lines of MIPS code, and so the program needed to be optimized in order to fit within the given space. Multiplication and division were used, so the HI and LO registers were accessed for computational results and the desired output was stored to the specified locations in memory.

In the second part of the assignment, a program that calculates the factorial of a given number using a while loop was created. The algorithm was provided to be translated into the MIPS language. The input to the function and calculated output were stored to the specified memory addresses and the constraints of the program were to use no more than 11 MIPS instructions.

**Tasks Successfully Accomplished**

1. The MIPS code was successfully generated using the given formula, register, memory and number of instructions restrictions for part 1 and part 2.
2. The correct values were produced and stored to their appropriate addresses in memory.
3. The test logs were filled with the correct values.

**Source Code**

|  |
| --- |
| **assignment3part1.asm** |
| #Variable Initiation  ori $a0, $0, 0x8000 #a = 0x8000  ori $a1, $0, 0x00A9 #b = 0x00A9  ori $s0, $0, 1974 #c = 1974  #Arithmetic computation  multu $a0, $a0 #x = a \* a  mflo $s1  sw $s1, 0x20  multu $s1, $a1 #y = x \* b  mfhi $s2  mflo $s3  sw $s2, 0x24  sw $s3, 0x28  #y = y >> 16  sll $s2, $s2, 16 #shift high register by 32 - 16 left  srl $s3, $s3, 16 #shift low register right by 16  add $s2, $s2, $s3 #add the results  #c = (c + y / c) / 2  Math: divu $s2, $s0 #y/c  mflo $t0 #low bits contain the quotient  add $t0, $t0, $s0 #add y/c + c  srl $s0, $t0, 1 #shift right by 1 (division by 2) and store in c  beq $t2, 1, Loop  sw $s0, 0x2C #store results in 0x2c  ori $t2, $0, 1  Loop: slti $t1, $s0, 1665 #while loop  bnez $t1, Done  j Math  Done: sll $s0, $s0, 8  sw $s0, 0x30 |

|  |
| --- |
| **factorial.asm** |
| main: addi $a0, $0 , 5 # input n = 5 to $a0  sw $a0, 0($0) # input n stored into mem add 0x00  addi $16, $0, 1 # f = 1  factorial: slti $t2, $a0, 2 # is a0 < 2  bnez $t2, end  mult $a0, $16 # f = f \* n  addiu $a0, $a0, -1 # n = n - 1  mflo $16 # Stores product to s0  j factorial # Go back to factorial  end: sw $s0, 16($0) # Stores n! into mem add location 0x10 |

**Test Results**

Table 1: Displaying the Test Log Table for the first MIPS program experiment.

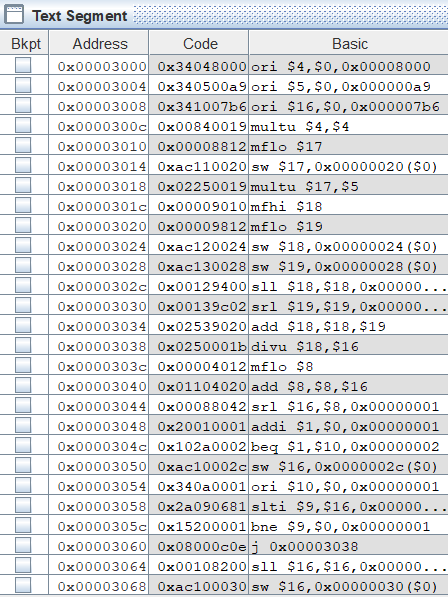
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Adr | MIPS Instruction | Machine Code | **Registers** | | | | |
| $a0 | $a1 | $s0 | $s1 | $s2 |
| 3000 | ori $4, $0, 0x8000 | 0x340500a9 | 8000 | 0 | 0 | 0 | 0 |
| 3004 | ori $5, $0, 0xa9 | 0x340500a9 | 8000 | a9 | 0 | 0 | 0 |
| 3008 | ori $16, $0, 0x7b6 | 0x341007b6 | 8000 | a9 | 7b6 | 0 | 0 |
| 300c | multu $4, $4 | 0x840019 | 8000 | a9 | 7b6 | 40000000 | 0 |
| 3010 | mflo $17 | 0x8812 | 8000 | a9 | 7b6 | 40000000 | 0 |
| 3014 | sw $17, 0x20($0) | 0xac110020 | 8000 | a9 | 7b6 | 40000000 | 0 |
| 3018 | multu $17, $5 | 0x02250019 | 8000 | a9 | 7b6 | 40000000 | 0 |
| 301c | mfhi $18 | 0x9010 | 8000 | a9 | 7b6 | 40000000 | 2a |
| 3020 | mflo $19 | 0x9812 | 8000 | a9 | 7b6 | 40000000 | 2a |
| 3024 | sw $18, 0x24($0) | 0xac120024 | 8000 | a9 | 7b6 | 40000000 | 2a |
| 3028 | sw $19, 0x28($0) | 0xac130028 | 8000 | a9 | 7b6 | 40000000 | 2a |
| 302c | sll $18, $18, 0x10 | 0x129400 | 8000 | a9 | 7b6 | 40000000 | 2a0000 |
| 3030 | srl $19, $19, 0x10 | 0x139c02 | 8000 | a9 | 7b6 | 40000000 | 2a0000 |
| 3034 | add $18, $18, $19 | 0x2539020 | 8000 | a9 | 7b6 | 40000000 | 2a4000 |
| 3038 | divu $18, $16 | 0x250001b | 8000 | a9 | 7b6 | 40000000 | 2a0000 |
| 303c | mflo $8 | 0x4012 | 8000 | a9 | 7b6 | 40000000 | 2a0000 |
| 3040 | add $8, $8, $16 | 0x0114020 | 8000 | a9 | 7b6 | 40000000 | 2a4000 |
| 3044 | srl $16, $8, 0x1 | 0x88042 | 8000 | a9 | [698] [680] | 40000000 | 2a4000 |
| 3048 | addi $1, $0, 0x1 | 0x20010001 | 8000 | a9 | [698] [680] | 40000000 | 2a4000 |
| 304c | beq $1, $10, 0x2 | 0x102a0002 | 8000 | a9 | [698] [680] | 40000000 | 2a4000 |
| 3050 | sw $16, 0x2c($0) | 0xac10002c | 8000 | a9 | 698 | 40000000 | 2a4000 |
| 3054 | ori $19, $0, 0x1 | 0x340a0001 | 8000 | a9 | 698 | 40000000 | 2a4000 |
| 3058 | slti $9, $16, 0x681 | 0x2a090681 | 8000 | a9 | [698] [680] | 40000000 | 2a4000 |
| 305C | bne $9, $0, 0x1 | 0x15200001 | 8000 | a9 | [698] [680] | 40000000 | 2a4000 |
| 3060 | j 0x3038 | 0x8000c0e | 8000 | a9 | 698 | 40000000 | 2a4000 |
| 3064 | sll $16, $16, 0x8 | 0x108200 | 8000 | a9 | 68000 | 40000000 | 2a4000 |
| 3068 | sw $16, 0x30($0) | 0xac100030 | 8000 | a9 | 68000 | 40000000 | 2a4000 |
| 306C |  |  |  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Memory contents** | | | | |
| Word @ 0x20 | Word @ 0x24 | Word @ 0x28 | Word @ 0x2C | Word @ 0x30 |
| 0x40000000 | 0x0000002a | 0x40000000 | 0x698 | 0x68000 |

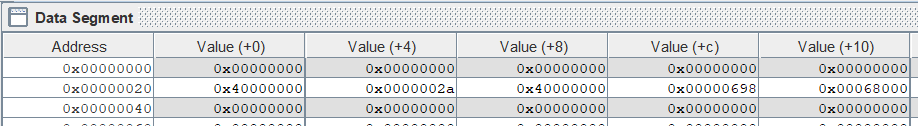
Table 2: Displaying the Test Log Table for the factorial MIPS program. The values in brackets are from different iterations running through the same MIPS instructions.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Adr | MIPS Instruction | Machine Code | Registers | | | | Memory Content | |
| **$a0** | **$s0** | **$t2** | **$** | Word @ 0x00 | Word @ 0x10 |
| 3000 | addi $a0, $0, 5 | 0x20040005 | 5 | 0 | 0 |  | 0x00000000 | 0x00000000 |
| 3004 | sw $a0, 0($0) | 0xac040000 | 5 | 0 | 0 |  | 0x00000005 | 0x00000000 |
| 3008 | addi $16, $0, 1 | 0x20100001 | 5 | 1 | 0 |  | 0x00000005 | 0x00000000 |
| 300c | slti $t2, $a0, 2 | 0x288a0002 | 5  [4]  [3]  [2]  [1] | 1  [5]  [0x14]  [0x3c]  [0x78] | 0  [0]  [0]  [0]  [1] |  | 0x00000005 | 0x00000000 |
| 3010 | bnez $t2, end | 0x15400004 | 5  [4]  [3]  [2] | 1  [5]  [0x14]  [0x3c] | 0  [0]  [0]  [0] |  | 0x00000005 | 0x00000000 |
| 3014 | mult $a0, $16 | 0x00900018 | 5  [4]  [3]  [2] | 1  [5]  [0x14]  [0x3c] | 0  [0]  [0]  [0] |  | 0x00000005 | 0x00000000 |
| 3018 | addiu $a0, $a0, -2 | 0x2484ffff | 4  [3]  [2]  [1] | 1  [5]  [0x14]  [0x3c] | 0  [0]  [0]  [0] |  | 0x00000005 | 0x00000000 |
| 301c | mflo $16 | 0x00008012 | 4  [3]  [2]  [1] | 5  [0x14]  [0x3c]  [0x78] | 0  [0]  [0]  [0] |  | 0x00000005 | 0x0000000 |
| 3020 | j factorial | 0x08000c03 |  |  |  |  | 0x00000005 | 0x00000000 |
| 3024 | sw $s0, 16($0) | 0xac100010 | 1 | 0x78 | 1 |  | 0x00000005 | 0x00000078 |
| 3028 |  |  |  |  |  |  |  |  |

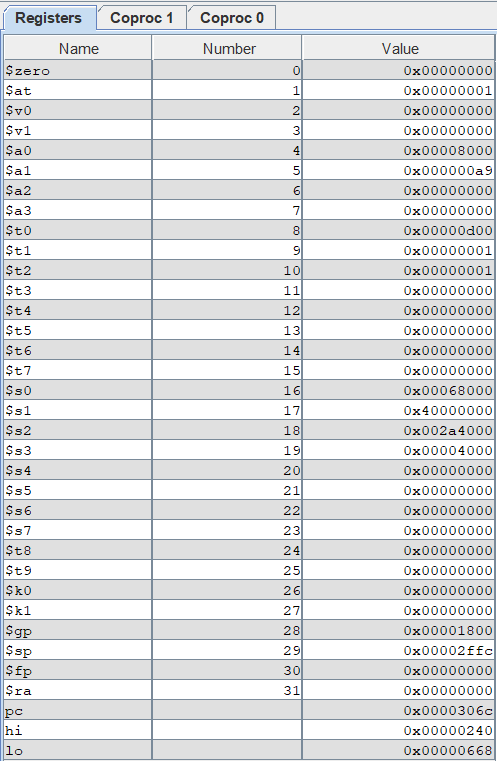
**Screen Captures**

****

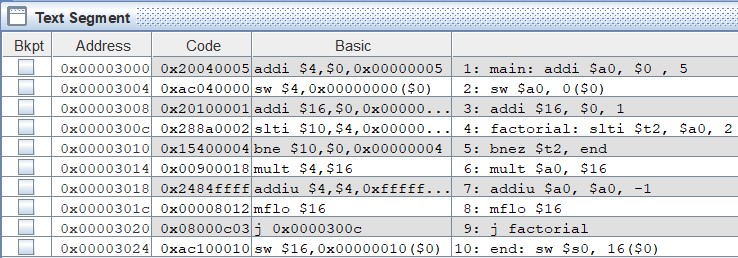
*Figure 1: Displaying the Text Segment table after completely running through the first experiment*

****

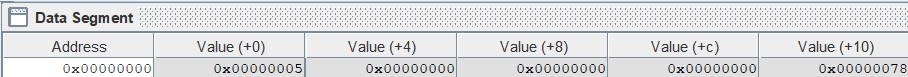
*Figure 2: Displaying the Data Segment table after running through the first experiment. The values of each address from 0x20 to 0x30 are shown above.\*

****

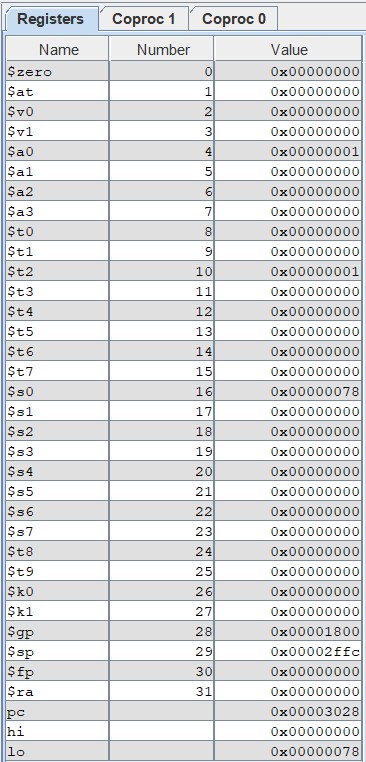
*Figure 3: Displaying the Register table after completing the execution of the first MIPS program.*

****

*Figure 4: Displaying the Text Segment table after assembling the factorial MIPS program. There are 10 assembly instructions in total.*

**

*Figure 5: Displaying the Data Segment table after completely running the MIPS program. At memory location 0, the value stored is the initial n value, which is 0x05 (510). At memory address 0x10, the value stored is the product of 5!, which is 0x78 (12010).*

**

*Figure 6: Displaying the Register values for the factorial MIPS program after completing its run. Register $s0 contains the n! value, register $a0 contains the n value, and $t2 contains the condition-checking value, where the value 0 means n is greater than 1 and the value 1 means n is less than or equal to 1.*

**Discussion/ Conclusion**

When the MIPS code was assembled, a few of the instructions were broken up into separate instructions, which increased the number of lines of MIPS instructions were actually read and executed. The instructions had to be optimized to reduce the number of actual MIPS instructions that were executed. For instance, the code for sd (store double) breaks up into 4 separate instructions, increasing the overall instruction count by 3. This was optimized by reducing the code to 2 instances of sw (store word), which was executed as only 2 instructions.

To further reduce the amount of instructions, the program branches to a previous mathematical operation that had been created in code earlier. This conditional branching removed the need to repeat the mathematical instruction code, which would have added several lines to the overall instruction count.

The factorial function was created using a while loop. The while loop was created by first checking the condition to determine if the program would enter the loop or exit the loop. When the program finished executing the loop, a jump instruction brings the program back to the top of the loop. This reflects the behavior of the whole loop, which always checks conditions as the first instruction.

When the input parameter is changed from 5 to 0, the program outputs the appropriate value of 1 for 0!. This is because the value for the factorial output is set to 1 before executing the while loop that handles the factorial calculation. The program does not enter the loop because the condition for the loop is false and so the value remains unchanged.

This lab utilized loop functionality and instruction count restrictions to gain a better understanding of the MIPS instruction set architecture. The limited instruction count helped to understand how instructions may be further translated and broken up into multiple instructions and the loop functionality helped to gain understanding about branch and jump instructions as well as conditional statements in the MIPS language.