

San Jose State University
Department of Computer Engineering

CMPE 140 Lab Report

Lab 3 Report

Title MIPS Instruction Set Architecture & Programming (2)

Semester Spring 2019

Date 02/27/19

by



Name Nickolas Schiffer

SID 012279319

Name Salvatore Nicosia

SID 012013599

Lab Checkup Record

Week	Performed By (signature)	Checked By (signature)	Tasks Successfully Completed*	Tasks Partially Completed*	Tasks Failed or Not Performed*
1	SN 		100%		

* Detailed descriptions must be given in the report.

I. INTRODUCTION

The purpose of this lab is to become familiar with the MIPS ISA control structures and the \$hi and \$lo registers. In particular, two algorithms were programmed to understand how multiplication and division store the results in \$hi and \$lo registers. The first algorithm is a MIPS assembly program to perform a few arithmetic operations. The second algorithm is a MIPS assembly program to compute the factorial of any given integer.

II. TESTING PROCEDURE

In order to write these MIPS program, the MIPS Assembler/Simulator software was utilized to assemble the code. For the first program the C++ pseudo code shown in figure 1 was converted into MIPS assembly language. The code for the first algorithm named *Algorithm 1.smd* can be in the appendix.

- $\$a0 \leftarrow a$
 - $\$a1 \leftarrow b$
 - $\$s0 \leftarrow c$
 - $\$s1 \leftarrow x$
 - $\$s2 \leftarrow y$
- Variables initialization
 1. $a = 0x8000;$ #MIPS instruction: `addiu $a0, $0, 0x8000`
 2. $b = 0x00A9;$
 3. $c = 1974;$
 - Arithmetic computation
 4. $x = a * a;$
 5. store the value of x to memory location at address 0x20;
 6. $y = x * b;$
 7. store the value of y to memory location at address 0x24;
 8. $y = y \gg 16;$
 9. $c = (c + y / c) / 2;$
 10. store the value of c to memory location at address 0x2C;
 - While loop
 11. `while(c >= 1665){`
 12. $c = (c + y / c) / 2;$
 13. `}`
 14. $c = c \ll 8;$
 15. store the value of c to memory location at 0x30;

Figure 1. C++ pseudo code for algorithm 1

The MIPS assembly code was then assembled and executed to verify the correct behavior of each instruction and verify the contents of the relevant registers. After verifying the correct behavior of the algorithm the execution results and the values of the memory addresses 0x20, 0x24, 0x2C, and 0x30 were recorded in the test log table (see *Table 1 and 2*) in the testing result section.

The second MIPS assembly program calculates the factorial of a given integer n . The assembly code was written in accordance to the C++ pseudo code shown in figure 2.

```

1. INPUT n = 5; //given number n
2. f = 1;
3. while (n > 1)
{
    f = f * n;
    n = n - 1;
}
4. OUTPUT f; //factorial f = n!

```

Figure 2. C++ pseudo code for algorithm 2

The registers \$a0 and \$s0 were used to hold the value of n and $n!$ respectively. After converting the pseudo code into MIPS assembly code it was then assembled and executed. Through the execution of each instruction the contents of the relevant registers were verified. After verifying the correct behavior of the algorithm the execution results and the values of the memory addresses 0x00 and 0x10 corresponding to n and $n!$ were recorded in the test log table (see Table 3) in the testing result section. The code for the second algorithm named *Algorithm 2.smd* can be in the appendix.

III. TESTING RESULTS

Table 1. Test Log Algorithm 1

Adr	MIPS Instruction	Machine Code	Registers				
			\$a0	\$a1	\$s0	\$s1	\$s2
00	# addiu \$a0, \$zero, 32768 (\$a0 = 32768)	0x24048000	0x80000	0	0	0	0
04	# addiu \$a1, \$zero, 169 (\$a1 = 169)	0x240500A9	0x80000	0xA9	0	0	0
08	# addiu \$s0, \$zero, 1974 (\$s0 = 1974)	0x241007B6	0x80000	0xA9	0x7B6	0	0
0C	# mult \$a0, \$a0 (\$hi = High(\$a0 * \$a0); \$lo = Low(\$a0 * \$a0))	0x00840018	0x80000	0xA9	0x7B6	0	0
10	# mflo \$s1 (\$s1 = \$lo)	0x00008812	0x80000	0xA9	0x7B6	0x40000000	0
14	# sw \$s1, 32(\$zero) (mem[\$zero + 32] = \$s1)	0xAC110020	0x80000	0xA9	0x7B6	0x40000000	0
18	# mult \$s1, \$a1 (\$hi = High(\$s1 * \$a1); \$lo = Low(\$s1 * \$a1))	0x02250018	0x80000	0xA9	0x7B6	0x40000000	0
1C	# mfhi \$s2 (\$s2 = \$hi)	0x00009010	0x80000	0xA9	0x7B6	0x40000000	0x2A
20	# mflo \$s3 (\$s3 = \$lo)	0x00009812	0x80000	0xA9	0x7B6	0x40000000	0x2A
24	# srl \$s3, \$s3, 16 (\$s3 = \$s3 >> 16)	0x00139C02	0x80000	0xA9	0x7B6	0x40000000	0x2A
28	# sll \$s2, \$s2, 16 (\$s2 = \$s2 << 16)	0x00129400	0x80000	0xA9	0x7B6	0x40000000	0x2A0000
2C	# or \$s2, \$s2, \$s3 (\$s2 = \$s2 \$s3)	0x02539025	0x80000	0xA9	0x7B6	0x40000000	0x2A4000
30	# sw \$s2, 36(\$zero) (mem[\$zero + 36] = \$s2)	0xAC120024	0x80000	0xA9	0x7B6	0x40000000	0x2A4000
34	# divu \$s2, \$s0 (\$Hi = \$s2 div \$s0; \$Lo = \$s2 mod \$s0)	0x0250001B	0x80000	0xA9	0x7B6	0x40000000	0x2A4000
38	# mflo \$t0 (\$t0 = \$lo)	0x00004012	0x80000	0xA9	0x7B6	0x40000000	0x2A4000
3C	# addu \$s0, \$t0, \$s0 (\$s0 = \$t0 + \$s0)	0x01108021	0x80000	0xA9	0xD30	0x40000000	0x2A4000
40	# srl \$s0, \$s0, 1 (\$s0 = \$s0 >> 1)	0x00108042	0x80000	0xA9	0x698	0x40000000	0x2A4000
44	# sw \$s0, 44(\$zero) (mem[\$zero + 44] = \$s0)	0xAC10002C	0x80000	0xA9	0x698	0x40000000	0x2A4000
48	# slti \$RD, \$s0, 1665 (if (\$s0 < 1665) \$RD = 1 else \$RD = 0)	0x2A0B0681	0x80000	0xA9	0x680	0x40000000	0x2A4000
4C	# bne \$t3, \$zero, 5 (if (\$t3 != \$zero) goto 5)	0x140B0005	0x80000	0xA9	0x698	0x40000000	0x2A4000
50	# divu \$s2, \$s0 (\$Hi = \$s2 div \$s0; \$Lo = \$s2 mod \$s0)	0x0250001B	0x80000	0xA9	0x698	0x40000000	0x2A4000
54	# mflo \$t0 (\$t0 = \$lo)	0x00004012	0x80000	0xA9	0x698	0x40000000	0x2A4000
58	# addu \$s0, \$t0, \$s0 (\$s0 = \$t0 + \$s0)	0x01108021	0x80000	0xA9	0xD00	0x40000000	0x2A4000
5C	# srl \$s0, \$s0, 1 (\$s0 = \$s0 >> 1)	0x00108042	0x80000	0xA9	0x680	0x40000000	0x2A4000
60	# j 0x0012 (jump to addr 0x0048)	0x08000012	0x80000	0xA9	0x680	0x40000000	0x2A4000
64	# sll \$s0, \$s0, 8 (\$s0 = \$s0 << 8)	0x00108200	0x80000	0xA9	0x68000	0x40000000	0x2A4000
68	# sw \$s0, 48(\$zero) (mem[\$zero + 48] = \$s0)	0xAC100030	0x80000	0xA9	0x68000	0x40000000	0x2A4000
6C	# j 0x0000 (jump to addr 0x0000)	0x08000000	0x80000	0xA9	0x68000	0x40000000	0x2A4000

Table 2. Test Log Algorithm 1

Memory Contents			
Word @ 0x20	Word @ 0x24	Word @ 0x2C	Word @ 0x30
0x40000000	0x00000024	0x00000698	0x00068000

Figures 1, and 2 show the registers values generated by the assembler/simulator for algorithm 1. In addition, it shows the values of \$lo and \$hi after the multiplication and division and the content in memory for addresses 0x20, 0x24, 0x2C, and 0x30.

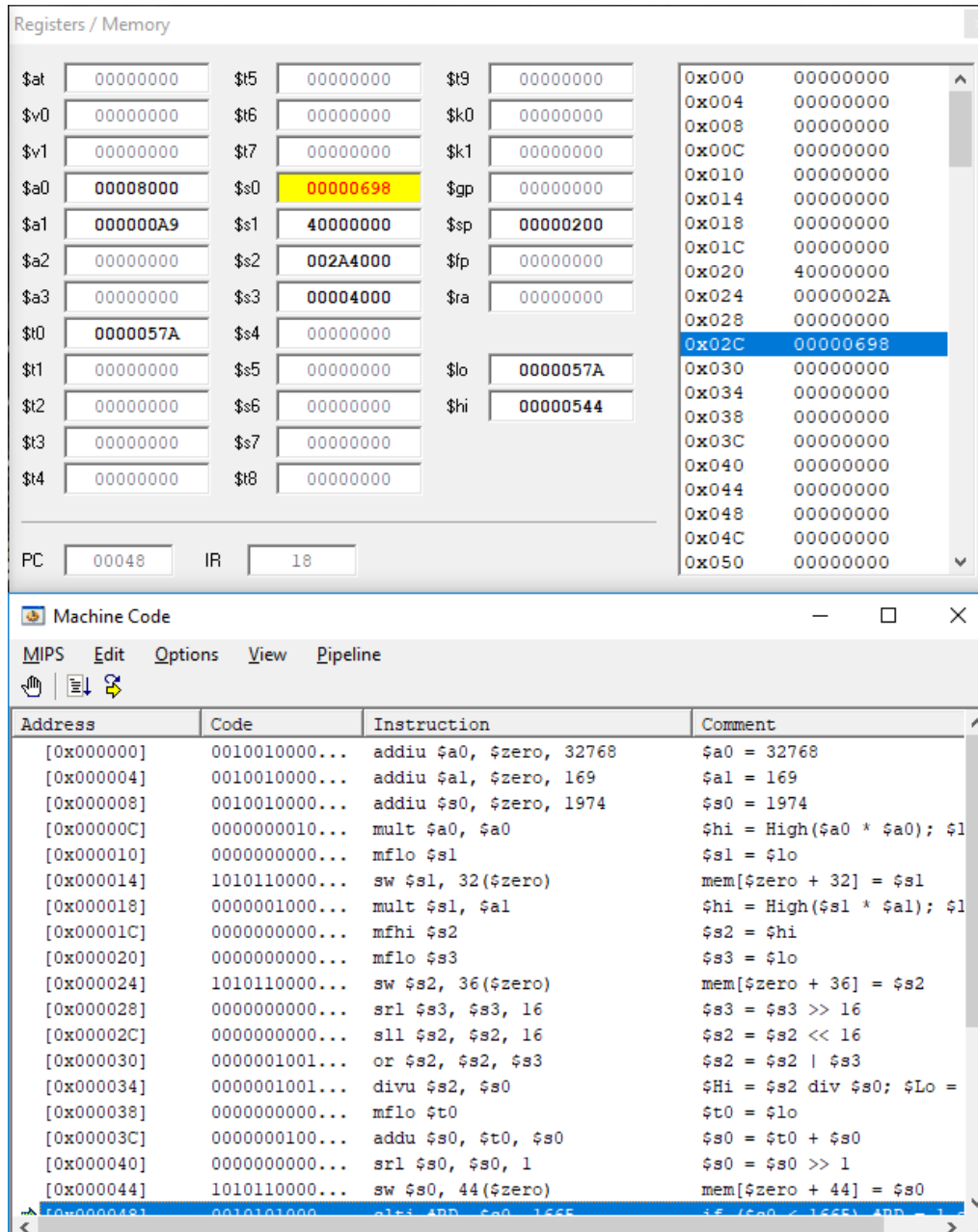


Figure 1: Algorithm 1 execution result for `sw $s0, 44($zero)` showing the stored value = 00000678 at memory address 0x02C

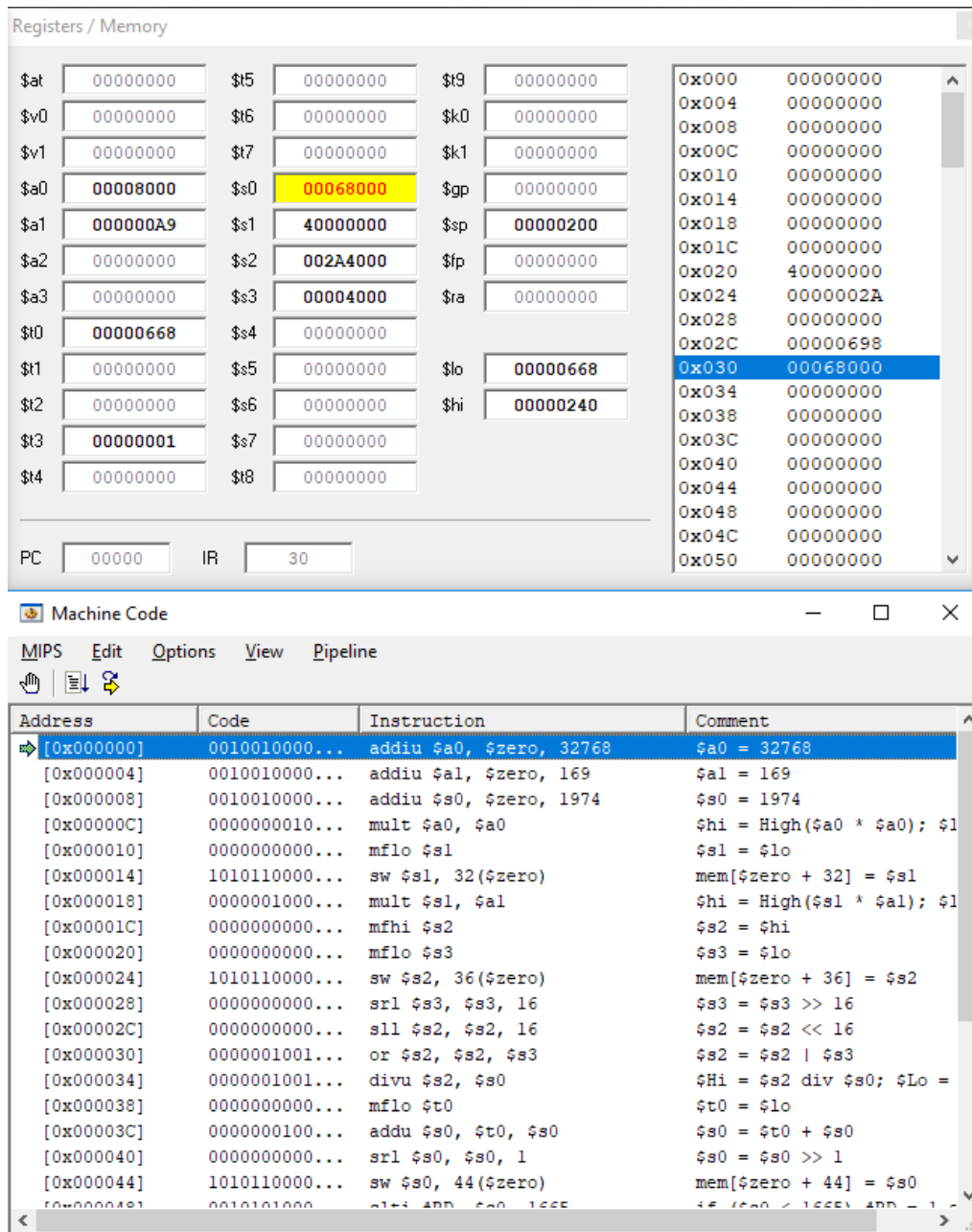


Figure 2: Algorithm 1 final execution result showing memory value = 40000000 at address 0x020, value = 00000024 at address 0x024, value = 00000698 at address 0x02C, and value = 00068000 at address 0x030

The values recorded in Table 3 for the factorial computation show the content of the registers after the execution of each instruction and the relative content at memory address 0x00 (n) and 0x01 (n!). Figures 2 and 3 show the execution of instructions where it shows the value of n and n! in memory and the content of the \$lo and \$hi registers after the multiplication.

Table 3. Test Log Algorithm 2

Adr	MIPS Instruction	Machine Code	Registers				Memory Content	
			\$a0	\$s0	\$t0	\$	Word @ 0x00	Word @ 0x10
00	# addi \$a0, \$zero, 5 (\$a0 = 5)	0x20040005	0x5	0	0	-	0	0
04	# addi \$s0, \$zero, 1 (\$s0 = 1)	0x20100001	0x5	0x1	0	-	0	0
08	# sw \$a0, 0(\$zero) (mem[\$zero + 0] = \$a0)	0xAC040000	0x5	0x1	0	-	0x5	0
0C	# slti \$RD, \$a0, 1 (if (\$a0 < 1) #RD = 1 else #RD = 0)	0x28880001	0x5	0x1	0	-	0x5	0
10	# bne \$t0, \$zero, 4 (if (\$t0 != \$zero) goto 4)	0x14080004	0	0x78	0x1	-	0x5	0
14	# mult \$s0, \$a0 (\$hi = High(\$s0 * \$a0); \$lo = Low(\$s0 * \$a0))	0x02040018	0	0x78	0x1	-	0x5	0
18	# mflo \$s0 (\$s0 = \$lo)	0x00008012	0	0x78	0x1	-	0x5	0
1C	# addi \$a0, \$a0, -1 (\$a0 = \$a0 + -1)	0x2084FFFF	0	0x78	0x1	-	0x5	0
20	# j 0x0003 (jump to addr 0x000C)	0x08000003	0	0x78	0x1	-	0x5	0
24	# sw \$s0, 16(\$zero) (mem[\$zero + 16] = \$s0)	0xAC100010	0	0x78	0x1	-	0x5	0x78
28	# j 0x0000 (jump to addr 0x0000)	0x08000000	0	0x78	0x1	-	0x5	0x78

Registers / Memory

\$a0: 00000004

\$a1: 00000000

\$a2: 00000000

\$a3: 00000000

\$t0: 00000000

\$t1: 00000000

\$t2: 00000000

\$t3: 00000000

\$t4: 00000000

\$t5: 00000000

\$t6: 00000000

\$t7: 00000000

\$s0: 00000005

\$s1: 00000000

\$s2: 00000000

\$s3: 00000000

\$s4: 00000000

\$s5: 00000000

\$s6: 00000000

\$s7: 00000000

\$s8: 00000000

\$t9: 00000000

\$k0: 00000000

\$k1: 00000000

\$gp: 00000000

\$sp: 00000200

\$fp: 00000000

\$ra: 00000000

\$lo: 00000005

\$hi: 00000000

0x000: 00000005

0x004: 00000000

0x008: 00000000

0x00C: 00000000

0x010: 00000000

0x014: 00000000

0x018: 00000000

0x01C: 00000000

0x020: 00000000

0x024: 00000000

0x028: 00000000

0x02C: 00000000

0x030: 00000000

0x034: 00000000

0x038: 00000000

0x03C: 00000000

0x040: 00000000

0x044: 00000000

0x048: 00000000

0x04C: 00000000

0x050: 00000000

PC: 0000C IR: 9

Machine Code

MIPS Edit Options View Pipeline

Address	Code	Instruction	Comment
[0x000000]	0010000000...	addi \$a0, \$zero, 5	\$a0 = 5
[0x000004]	0010000000...	addi \$s0, \$zero, 1	\$s0 = 1
[0x000008]	1010110000...	sw \$a0, 0(\$zero)	mem[\$zero + 0] = \$a0
[0x00000C]	0010100010...	slti \$RD, \$a0, 1	if (\$a0 < 1) #RD = 1 else .
[0x000010]	0001010000...	bne \$t0, \$zero, 4	if (\$t0 != \$zero) goto 4
[0x000014]	0000001000...	mult \$s0, \$a0	\$hi = High(\$s0 * \$a0); \$lo.
[0x000018]	0000000000...	mflo \$s0	\$s0 = \$lo
[0x00001C]	0010000010...	addi \$a0, \$a0, -1	\$a0 = \$a0 + -1
[0x000020]	0000100000...	j 0x0003	jump to addr 0x000C
[0x000024]	1010110000...	sw \$s0, 16(\$zero)	mem[\$zero + 16] = \$s0
[0x000028]	0000100000...	j 0x0000	jump to addr 0x0000

Figure 3: Algorithm 2 execution result showing memory value = 5 at address 0x00

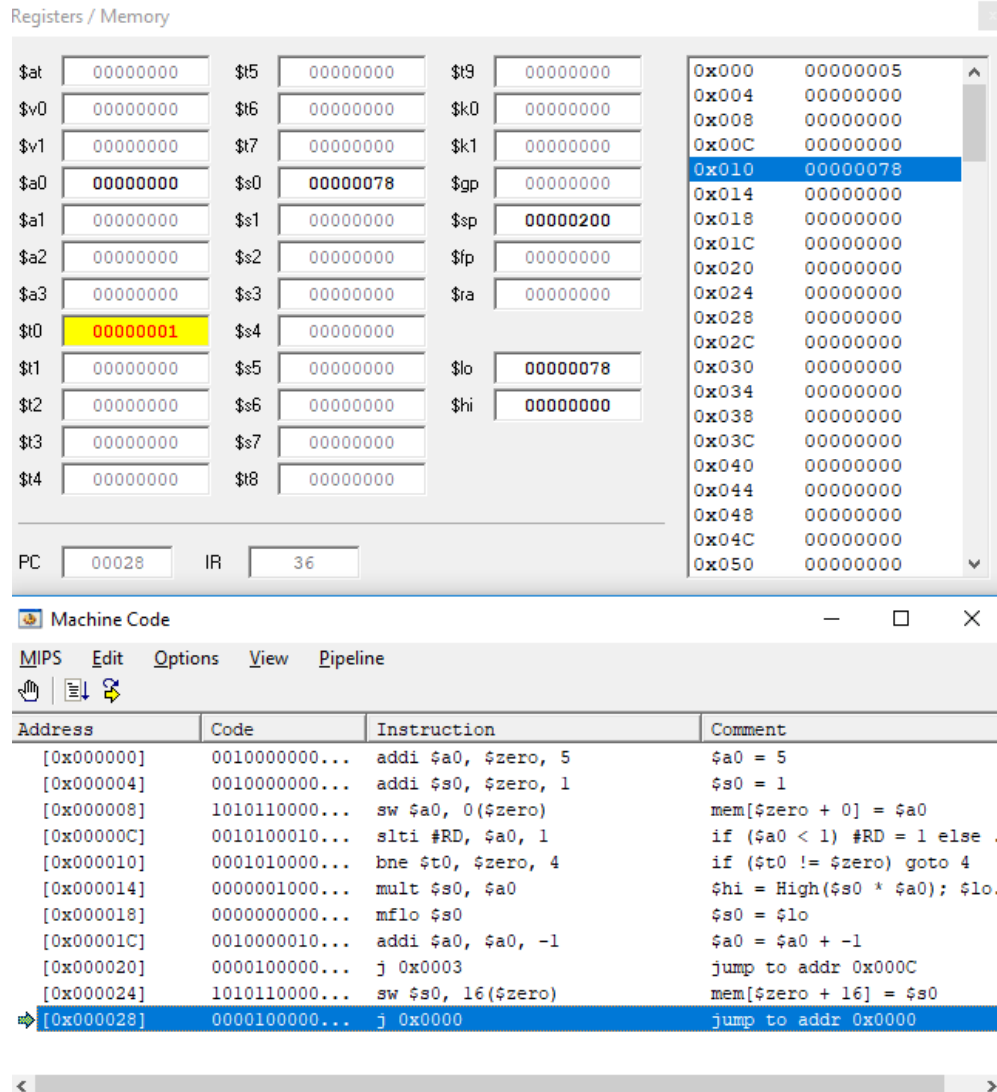


Figure 4: Algorithm 2 execution result for the factorial of 5 showing memory value = 78 at address 0x010)

IV. CONCLUSION

This lab further solidified how to use the MIPS instruction set to implement basic algorithms in a restricted number of instructions. An important note is being able to extract resultant information from the high and low registers from division and multiplication.

V. SUCCESSFUL TASKS

1. Implement the algorithms with the MIPS instruction set.
2. Write algorithms in the specified limit of instructions of 28 and 11 for algorithm 1 and 2.
3. Verification of the register's content and memory value for each execution of the MIPS instructions.
4. Recorded execution results in the test log tables.

VI. APPENDIX

A. SOURCE CODE:

Algorithm 1.smd

```
# $a0 = a, $a1 = b, $s0 = c, $s1 = x, $s2 = y

main:
    addiu $a0, $0, 0x8000      # a = 0x8000
    addiu $a1, $0, 0x00A9     # b = 0x00A9
    addiu $s0, $0, 1974       # c = 1974

    mult $a0, $a0              # x=a*a;
    mflo $s1
    sw $s1, 0x20($0)          # store the value of x to memory location at address 0x20

    mult $s1, $a1              # y=x*b;
    mfhi $s2
    mflo $s3
    sw $s2, 0x24($0)          # store the value of y to memory location at address 0x24

    # y=y>>16;

    srl $s3, $s3, 16          # shift low right 16
    sll $s2, $s2, 16          # shift high left 16
    or $s2, $s2, $s3          # or results and store into y

    # c=(c+(y/c))/2;

    divu $s2, $s0              # y/c
    mflo $t0
    addu $s0, $t0, $s0         # c + (y/c)
    srl $s0, $s0, 1           # c /= 2
    sw $s0, 0x2C($0)          # store the value of c to memory location at address 0x2C

while:
    slti $t3, $s0, 1665       # while(c >= 1665){ c = (c + y / c) / 2;}
    bne $t3, $0, done         # if $t2 < $s0, $t3 = 1 else $t3 = 0
                                # $t3 = 1 branch to done

    # c=(c+(y/c))/2;

    divu $s2, $s0              # y/c
    mflo $t0
    addu $s0, $t0, $s0         # c + (y/c)
    srl $s0, $s0, 1           # c /= 2
    j while

done:
    sll $s0, $s0, 8           # c <= 8;
    sw $s0, 0x30($0)          # store the value of c to memory location 0x30
    j main
```

Algorithm 2.smd

```
# $a0 = n, $s0 = n!

main:
    addi $a0, $0, 5           # n = 5
    addi $s0, $0, 1           # n! = 1

    sw $a0, 0x00($0)          # store the value of n to memory location 0x00

while:
    slti $t0, $a0, 1          # if n < 1, $t0 = 1 else $t0 = 0;
    bne $t0, $zero, done      # if $t0 = 1 branch to done
    mult $s0, $a0              # n! = n! * n
    mflo $s0                  # move multiplication result to $s0
```