**San Jose State University**
**Department of Computer Engineering**

**CMPE 140 Lab Report**

# Lab 2 Report

**Title** MIPS Instruction Set Architecture & Programming (2)

**Semester** Spring 2019 **Date** 02/20/19

**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 instruction set by assembling, simulating, and analyzing a sample program.

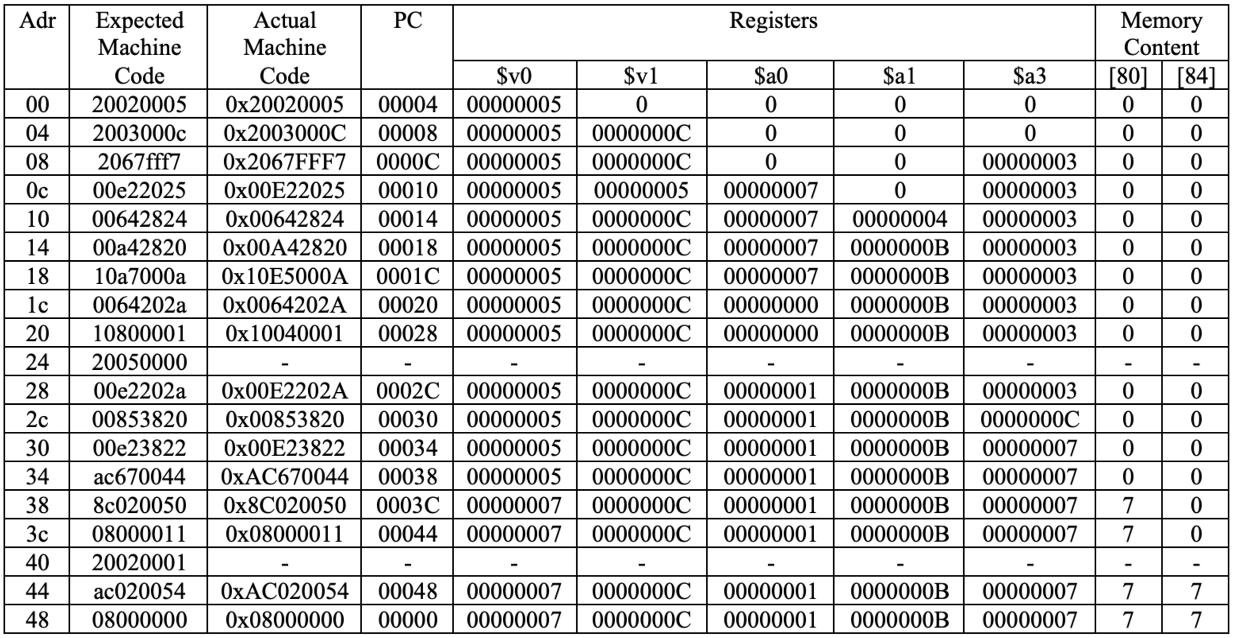
## II. TESTING PROCEDURE

In order to test the sample MIPS program, it was utilized the MIPS Assembler/Simulator software to assemble the code. The MIPS assembly code contained in the file “miptest.asm” can be found in the appendix section. After assembling the code, for each MIPS instruction each machine code generated was compared to the machine code provided. Through the MIPS simulator, for each of the executions of the corresponding instruction, the content of each register was verified. The execution results were recorded in the test log table (see *Table 1*). These results include the program counter of the relevant registers and the memory value at address 80 (0x50) and 84 (0x54) after the execution of the program was completed.

## III. TESTING RESULTS

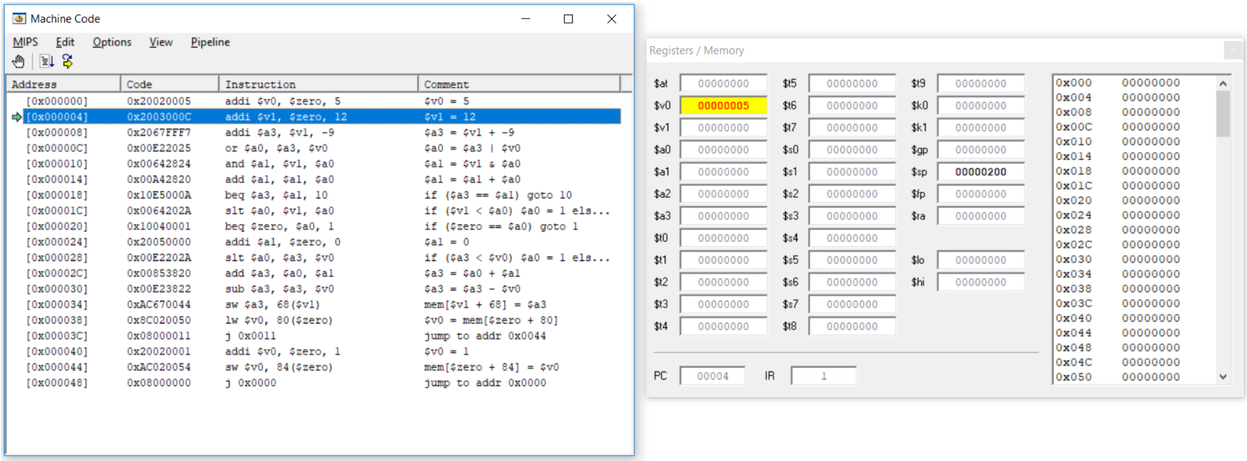
During the execution of each MIPS instruction the actual machine code, program counter, register values and memory content were recorded in *Table 1*.

**Table 1. Test Log**



| Adr | Expected Machine Code | Actual Machine Code | PC | Registers | | | | | Memory Content | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | $v0 | $v1 | $a0 | $a1 | $a3 | [80] | [84] |
| 00 | 20020005 | 0x20020005 | 00004 | 00000005 | 0 | 0 | 0 | 0 | 0 | 0 |
| 04 | 2003000c | 0x2003000C | 00008 | 00000005 | 0000000C | 0 | 0 | 0 | 0 | 0 |
| 08 | 2067fff7 | 0x2067FFF7 | 0000C | 00000005 | 0000000C | 0 | 0 | 00000003 | 0 | 0 |
| 0c | 00e22025 | 0x00E22025 | 00010 | 00000005 | 00000005 | 00000007 | 0 | 00000003 | 0 | 0 |
| 10 | 00642824 | 0x00642824 | 00014 | 00000005 | 0000000C | 00000007 | 00000004 | 00000003 | 0 | 0 |
| 14 | 00a42820 | 0x00A42820 | 00018 | 00000005 | 0000000C | 00000007 | 0000000B | 00000003 | 0 | 0 |
| 18 | 10a7000a | 0x10E5000A | 0001C | 00000005 | 0000000C | 00000007 | 0000000B | 00000003 | 0 | 0 |
| 1c | 0064202a | 0x0064202A | 00020 | 00000005 | 0000000C | 00000000 | 0000000B | 00000003 | 0 | 0 |
| 20 | 10800001 | 0x10040001 | 00028 | 00000005 | 0000000C | 00000000 | 0000000B | 00000003 | 0 | 0 |
| 24 | 20050000 | - | - | - | - | - | - | - | - | - |
| 28 | 00e2202a | 0x00E2202A | 0002C | 00000005 | 0000000C | 00000001 | 0000000B | 00000003 | 0 | 0 |
| 2c | 00853820 | 0x00853820 | 00030 | 00000005 | 0000000C | 00000001 | 0000000B | 0000000C | 0 | 0 |
| 30 | 00e23822 | 0x00E23822 | 00034 | 00000005 | 0000000C | 00000001 | 0000000B | 00000007 | 0 | 0 |
| 34 | ac670044 | 0xAC670044 | 00038 | 00000005 | 0000000C | 00000001 | 0000000B | 00000007 | 0 | 0 |
| 38 | 8c020050 | 0x8C020050 | 0003C | 00000007 | 0000000C | 00000001 | 0000000B | 00000007 | 7 | 0 |
| 3c | 08000011 | 0x08000011 | 00044 | 00000007 | 0000000C | 00000001 | 0000000B | 00000007 | 7 | 0 |
| 40 | 20020001 | - | - | - | - | - | - | - | - | - |
| 44 | ac020054 | 0xAC020054 | 00048 | 00000007 | 0000000C | 00000001 | 0000000B | 00000007 | 7 | 7 |
| 48 | 08000000 | 0x08000000 | 00000 | 00000007 | 0000000C | 00000001 | 0000000B | 00000007 | 7 | 7 |

*Figures 1, 2,* and *3* show a few of the execution windows generated by the assembler/simulator.



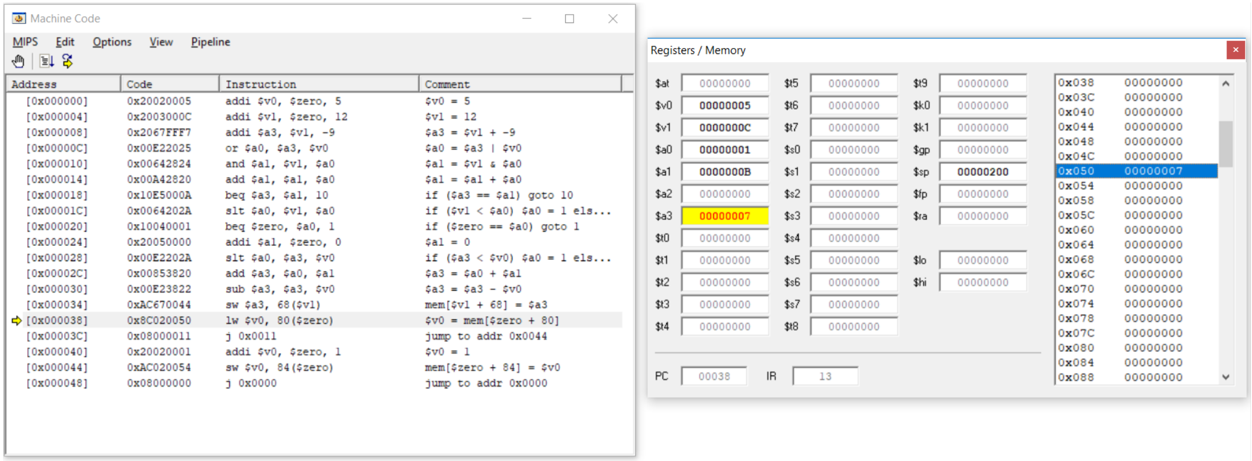
```
Address       Code          Instruction               Comment
[0x000000]    0x20020005    addi $v0, $zero, 5        $v0 = 5
[0x000004]    0x2003000C    addi $v1, $zero, 12       $v1 = 12
[0x000008]    0x2067FFF7    addi $a3, $v1, -9         $a3 = $v1 + -9
[0x00000C]    0x00E22025    or $a0, $a3, $v0          $a0 = $a3 | $v0
[0x000010]    0x00642824    and $a1, $v1, $a0         $a1 = $v1 & $a0
[0x000014]    0x00A42820    add $a1, $a1, $a0         $a1 = $a1 + $a0
[0x000018]    0x10E5000A    beq $a3, $a1, 10          if ($a3 == $a1) goto 10
[0x00001C]    0x0064202A    slt $a0, $v1, $a0         if ($v1 < $a0) $a0 = 1 els...
[0x000020]    0x10040001    beq $zero, $a0, 1         if ($zero == $a0) goto 1
[0x000024]    0x20050000    addi $a1, $zero, 0        $a1 = 0
[0x000028]    0x00E2202A    slt $a0, $a3, $v0         if ($a3 < $v0) $a0 = 1 els...
[0x00002C]    0x00853820    add $a3, $a0, $a1         $a3 = $a0 + $a1
[0x000030]    0x00E23822    sub $a3, $a3, $v0         $a3 = $a3 - $v0
[0x000034]    0xAC670044    sw $a3, 68($v1)           mem[$v1 + 68] = $a3
[0x000038]    0x8C020050    lw $v0, 80($zero)         $v0 = mem[$zero + 80]
[0x00003C]    0x08000011    j 0x0011                  jump to addr 0x0044
[0x000040]    0x20020001    addi $v0, $zero, 1        $v0 = 1
[0x000044]    0xAC020054    sw $v0, 84($zero)         mem[$zero + 84] = $v0
[0x000048]    0x08000000    j 0x0000                  jump to addr 0x0000
```

```
$at  00000000    $t5  00000000    $t9  00000000
$v0  00000005    $t6  00000000    $k0  00000000
$v1  00000000    $t7  00000000    $k1  00000000
$a0  00000000    $s0  00000000    $gp  00000000
$a1  00000000    $s1  00000000    $sp  00000200
$a2  00000000    $s2  00000000    $fp  00000000
$a3  00000000    $s3  00000000    $ra  00000000
$t0  00000000    $s4  00000000
$t1  00000000    $s5  00000000    $lo  00000000
$t2  00000000    $s6  00000000    $hi  00000000
$t3  00000000    $s7  00000000
$t4  00000000    $t8  00000000

PC   00004       IR   1
```

```
0x000   00000000
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
```

*Figure 1: Execution result of the first MIPS instruction with Machine Code 20020005*



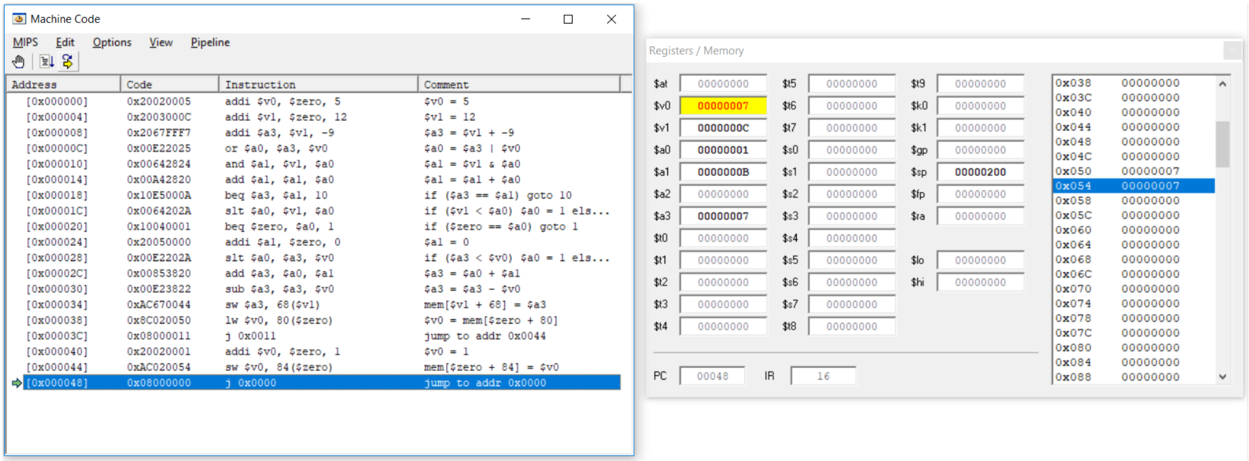
```
Address       Code          Instruction               Comment
[0x000000]    0x20020005    addi $v0, $zero, 5        $v0 = 5
[0x000004]    0x2003000C    addi $v1, $zero, 12       $v1 = 12
[0x000008]    0x2067FFF7    addi $a3, $v1, -9         $a3 = $v1 + -9
[0x00000C]    0x00E22025    or $a0, $a3, $v0          $a0 = $a3 | $v0
[0x000010]    0x00642824    and $a1, $v1, $a0         $a1 = $v1 & $a0
[0x000014]    0x00A42820    add $a1, $a1, $a0         $a1 = $a1 + $a0
[0x000018]    0x10E5000A    beq $a3, $a1, 10          if ($a3 == $a1) goto 10
[0x00001C]    0x0064202A    slt $a0, $v1, $a0         if ($v1 < $a0) $a0 = 1 els...
[0x000020]    0x10040001    beq $zero, $a0, 1         if ($zero == $a0) goto 1
[0x000024]    0x20050000    addi $a1, $zero, 0        $a1 = 0
[0x000028]    0x00E2202A    slt $a0, $a3, $v0         if ($a3 < $v0) $a0 = 1 els...
[0x00002C]    0x00853820    add $a3, $a0, $a1         $a3 = $a0 + $a1
[0x000030]    0x00E23822    sub $a3, $a3, $v0         $a3 = $a3 - $v0
[0x000034]    0xAC670044    sw $a3, 68($v1)           mem[$v1 + 68] = $a3
[0x000038]    0x8C020050    lw $v0, 80($zero)         $v0 = mem[$zero + 80]
[0x00003C]    0x08000011    j 0x0011                  jump to addr 0x0044
[0x000040]    0x20020001    addi $v0, $zero, 1        $v0 = 1
[0x000044]    0xAC020054    sw $v0, 84($zero)         mem[$zero + 84] = $v0
[0x000048]    0x08000000    j 0x0000                  jump to addr 0x0000
```

```
$at  00000000    $t5  00000000    $t9  00000000
$v0  00000005    $t6  00000000    $k0  00000000
$v1  0000000C    $t7  00000000    $k1  00000000
$a0  00000001    $s0  00000000    $gp  00000000
$a1  0000000B    $s1  00000000    $sp  00000200
$a2  00000000    $s2  00000000    $fp  00000000
$a3  00000007    $s3  00000000    $ra  00000000
$t0  00000000    $s4  00000000
$t1  00000000    $s5  00000000    $lo  00000000
$t2  00000000    $s6  00000000    $hi  00000000
$t3  00000000    $s7  00000000
$t4  00000000    $t8  00000000

PC   00038       IR   13
```

```
0x038   00000000
0x03C   00000000
0x040   00000000
0x044   00000000
0x048   00000000
0x04C   00000000
0x050   00000007
0x054   00000000
0x058   00000000
0x05C   00000000
0x060   00000000
0x064   00000000
0x068   00000000
0x06C   00000000
0x070   00000000
0x074   00000000
0x078   00000000
0x07C   00000000
0x080   00000000
0x084   00000000
0x088   00000000
```

*Figure 2: Execution result for Machine code AC670044 showing memory value = 7 at address 80 (0x050)*



```
Address       Code          Instruction               Comment
[0x000000]    0x20020005    addi $v0, $zero, 5        $v0 = 5
[0x000004]    0x2003000C    addi $v1, $zero, 12       $v1 = 12
[0x000008]    0x2067FFF7    addi $a3, $v1, -9         $a3 = $v1 + -9
[0x00000C]    0x00E22025    or $a0, $a3, $v0          $a0 = $a3 | $v0
[0x000010]    0x00642824    and $a1, $v1, $a0         $a1 = $v1 & $a0
[0x000014]    0x00A42820    add $a1, $a1, $a0         $a1 = $a1 + $a0
[0x000018]    0x10E5000A    beq $a3, $a1, 10          if ($a3 == $a1) goto 10
[0x00001C]    0x0064202A    slt $a0, $v1, $a0         if ($v1 < $a0) $a0 = 1 els...
[0x000020]    0x10040001    beq $zero, $a0, 1         if ($zero == $a0) goto 1
[0x000024]    0x20050000    addi $a1, $zero, 0        $a1 = 0
[0x000028]    0x00E2202A    slt $a0, $a3, $v0         if ($a3 < $v0) $a0 = 1 els...
[0x00002C]    0x00853820    add $a3, $a0, $a1         $a3 = $a0 + $a1
[0x000030]    0x00E23822    sub $a3, $a3, $v0         $a3 = $a3 - $v0
[0x000034]    0xAC670044    sw $a3, 68($v1)           mem[$v1 + 68] = $a3
[0x000038]    0x8C020050    lw $v0, 80($zero)         $v0 = mem[$zero + 80]
[0x00003C]    0x08000011    j 0x0011                  jump to addr 0x0044
[0x000040]    0x20020001    addi $v0, $zero, 1        $v0 = 1
[0x000044]    0xAC020054    sw $v0, 84($zero)         mem[$zero + 84] = $v0
[0x000048]    0x08000000    j 0x0000                  jump to addr 0x0000
```

```
$at  00000000    $t5  00000000    $t9  00000000
$v0  00000007    $t6  00000000    $k0  00000000
$v1  0000000C    $t7  00000000    $k1  00000000
$a0  00000001    $s0  00000000    $gp  00000000
$a1  0000000B    $s1  00000000    $sp  00000200
$a2  00000000    $s2  00000000    $fp  00000000
$a3  00000007    $s3  00000000    $ra  00000000
$t0  00000000    $s4  00000000
$t1  00000000    $s5  00000000    $lo  00000000
$t2  00000000    $s6  00000000    $hi  00000000
$t3  00000000    $s7  00000000
$t4  00000000    $t8  00000000

PC   00048       IR   16
```

```
0x038   00000000
0x03C   00000000
0x040   00000000
0x044   00000000
0x048   00000000
0x04C   00000000
0x050   00000007
0x054   00000007
0x058   00000000
0x05C   00000000
0x060   00000000
0x064   00000000
0x068   00000000
0x06C   00000000
0x070   00000000
0x074   00000000
0x078   00000000
0x07C   00000000
0x080   00000000
0x084   00000000
0x088   00000000
```

*Figure 3: Execution result for Machine code AC020054 showing memory value = 7 at address 80 (0x050) and 84 (0x054)*

## IV. CONCLUSION

This lab taught us that when branching occurs in the assembler, rs and rt get flipped resulting in different machine code than what was expected. For example, in the instruction `beq  $5, $7, end` the expected machine code is 10A7000A. However, since the MIPS assembler flips rs and rt, this becomes `beq  $7, $5, end` and the resulting machine becomes 10E5000A.

## V. SUCCESSFUL TASKS

1. Install the MIPS Assembler/Simulator software.
2. Compare the machine code generated by assembler with machine code provided.
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 table.

## VI. APPENDIX

### A. SOURCE CODE:

*mipstest.asm*

```
# mipstest.asm
# Test the following MIPS instructions.
# add, sub, and, or, slt, addi, lw, sw, beq, j
#       Assembly                  Description           Address Machine
main:   addi $2, $0, 5          # initialize $2 = 5     0       20020005
        addi $3, $0, 12         # initialize $3 = 12    4       2003000c
        addi $7, $3, -9         # initialize $7 = 3     8       2067fff7
        or   $4, $7, $2         # $4 <= 3 or 5 = 7      c       00e22025
        and  $5, $3, $4         # $5 <= 12 and 7 = 4    10      00642824
        add  $5, $5, $4         # $5 = 4 + 7 = 11       14      00a42820
        beq  $5, $7, end        # shouldn't be taken    18      10a7000a
        slt  $4, $3, $4         # $4 = 12 < 7 = 0       1c      0064202a
        beq  $4, $0, around     # should be taken       20      10800001
        addi $5, $0, 0          # shouldn't execute     24      20050000
around: slt  $4, $7, $2         # $4 = 3 < 5 = 1        28      00e2202a
        add  $7, $4, $5         # $7 = 1 + 11 = 12      2c      00853820
        sub  $7, $7, $2         # $7 = 12 - 5 = 7       30      00e23822
        sw   $7, 68($3)         # [80] = 7              34      ac670044
        lw   $2, 80($0)         # $2 = [80] = 7         38      8c020050
        j    end                # should be taken       3c      08000011
        addi $2, $0, 1          # shouldn't execute     40      20020001
end:    sw   $2, 84($0)         # write adr 84 = 7      44      ac020054
        j    main               # go back to beginning  48      08000000
```