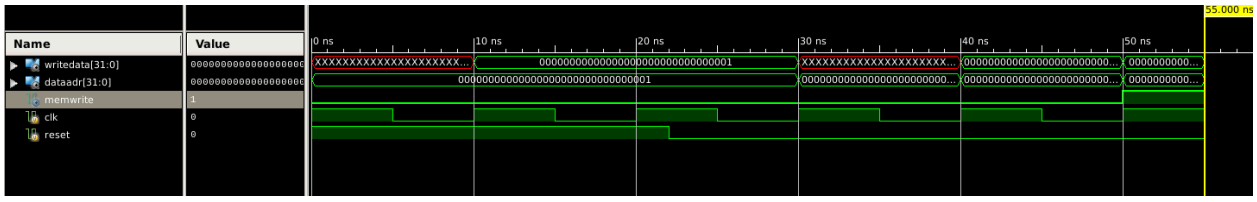
PHYS 234 Lab 10: MIPS Microarchitecture

Ethan Yang

In this lab, we will dive into the MIPS microarchitecture and explain the simulated circuit behavior of two assembly/machine code snippets: one shown in Figure 7.60, and the other (from Lab 9) shown directly below, with their own simulated circuit attached immediately after.

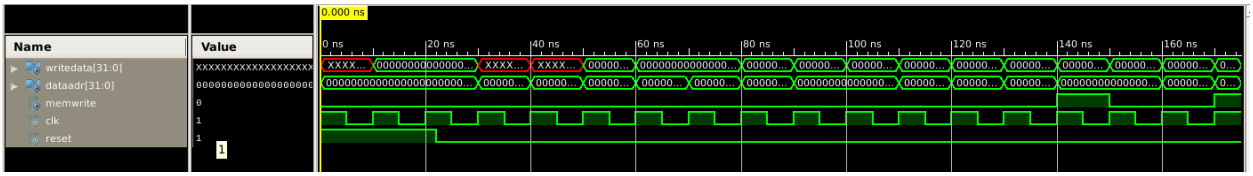
```
Assembly                    Machine
addi $s0, $0, 1             0x20100001
addi $s1, $0, 2             0x20110002
add $s2, $s0, $s1           0x02119020
sw $s2, 32($0)              0xAC120020
```




```
# mipstest.asm
# David_Harris@hmc.edu 9 November 2005
#
# Test the MIPS processor.
# add, sub, and, or, slt, addi, lw, sw, beq, j
# If successful, it should write the value 7 to address 84

#           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 happen       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 happen       40         20020001
end:        sw   $2, 84($0)       # write adr 84 = 7       44         ac020054
```

Figure 7.60 **Assembly and machine code for MIPS test program**



I will explain why each register attains its current value at each rising edge for both simulations. Both programs use writedata, dataadr, memwrite, clk, and reset, where writedata stores rt, dataadr stores the ALU result, memwrite indicates whether or not values are written into the memory (1 when written, 0 when not written), and reset reinitializes the circuit when it's 1.

For the assembly code in Figure 7.6, the circuit started with reset = 1. Therefore, instruction 0 is run until reset becomes 0, and instruction 4 only executes at the first clock edge after reset becomes 0. Memwrite only turns to 1 during the lw and sw instructions.

The list below shows which values are stored during each instruction and why.

- addi $2, $0, 5: dataadr (result of the ALU) is 5. At the first clock edge, writedata is uninitialized, which is the previous register value of register $2, and at the second clock edge it updates to 5, which is the current value of $2. This is because writedata is delayed by 1 clock edge.
- addi $3, $0, 12: dataadr is 12 and writedata is uninitialized (XXXXX) (shows the previous uninitialized value of $3)
- addi $7, $3, -9: dataadr is 3 (12-9=3). Writedata is uninitialized (previous value of $7)
- or $4, $7, $2: rs = $7 = 3, rt = $2 = 5, dataaddr is 7 because 3 OR 5 in binary (0011 OR 0101) is 7 (0111). The previous value of rt is $2 = 5 so writedata shows 5.
- and $5, $3, $4: rs = $3 = 12, rt = $4 = 7. dataadr is 4 because 12 AND 7 (1100 AND 0111) in binary is 4 (0100). Writedata is 7 because rt=$4=7.
- add $5, $5, $4: rs = $5 = 4, rt = $4 = 7. dataaddr is 4+7=11. Writedata is $4=7.
- beq $5, $7, end: rs = $5 = 11, rt = $7 = 3. 11 doesn't equal 3 so the code doesn't branch to end. Dataaddr is rs - rt = 11-3=8. And Writedata retains the previous $7 value 3.
- slt $4, $3, $4: This line computes $4 = ($3 < $4) ? 1:0. Therefore, even though the ALU computes $3-$4=12-7=5, dataaddr stores 0 because 12 is not < 7. Writedata stores the previous $4 value 7.
- beq $4, $0, around: rs = $4 = 0, rt = $0 = 0. 0 equals 0 so the program branches to the around flag. Dataaddr is rs - rt = 0-0=0. And Writedata retains the $0 value 0.
- addi $5, $0, 0: This line is not executed because the beq instruction earlier skipped this line.
- slt $4, $7, $2: This line is executed because it is the around flag target, and it computes $4 = ($7 < $2) ? 1:0. Therefore, even though the ALU computes $7-$2=2-5= -2, dataaddr stores 1 because 3 < 5. Writedata stores the previous $2 value 5.
- add $7, $4, $5: Dataaddr stores the value 1 ($4 + $5 = 1 + 0 = 1) and writedata stores previous $5 value 11.
- sub $7, $7, $2: Dataaddr stores the value -4 ($7 - $2 = 1 - 5 = -4) and writedata stores previous $2 value 5.

- sw $7, 68($3): Dataaddr stores the ALU computation result of the address, which is 68 + $3 = 68 + 12 = 80 and writedata stores previous $7 value 7.
- lw $2, 80($0): Again, dataaddr stores the computed address 80 (80 + $0 = 80) and writedata stores previous $2 value 5. The current $2 value becomes the value in address 80, which is 7.
- j end: the program jumps to the end flag so this instruction itself doesn't have a dataaddr or writedata value.
- addi $2, $0, 1: This line is not executed because the program jumped to the end flag and this line was skipped.
- sw $2, 84($0): This line is executed because it is the end flag target. Dataaddr stores the computed address 84 (84 + $0 = 84) and writedata stores previous $2 value 7.

For the assembly code from Lab 9, the circuit also started with reset = 1 and became 0 during the third clock edge. This means instruction 4 is executed at the fourth clock edge. Memwrite turns into 1 at the last instruction sw.

- addi $s0, $0, 1 : ALU computes $0 + 1 = 1 and dataaddr stores the result 1. Writedata stores the previous value of $s0, which is still uninitialized and the simulation shows XXXXXX….
- addi $s1, $0, 2 : ALU computes $0 + 2 = 2 and dataaddr stores the result 2. Writedata stores the previous value of $s1, which, similarly, is still uninitialized and the simulation shows XXXXXX….
- add $s2, $s0, $s1: ALU computes $0 + $1 = 1 + 2 = 4 and dataaddr stores the result 3. Writedata stores the previous value of rt = $s1 = 2.
- sw $s2, 32($0): Dataaddr stores the computed address 32 (32 + $0 = 32) and writedata stores previous $s2 value 3.