

**Course Name**: Computer Architecture Lab

**Course Number and Section**: **14:332:361:05**

**Experiment**: Lab #4 - CPU Structure, Pipeline Programming and Hazards, Exceptions and Interrupts

**Lab Instructor**: Christos Mitropoulos

**Date Performed**: 10/27/2016

**Date Submitted**: 11/17/2016

**Submitted by**: Pawel Derkacz - 151005994

--------------------------For Lab Instructor Use ONLY--------------------------

GRADE: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

COMMENTS:

Electrical and Computer Engineering Department

School of Engineering

Rutgers University, Piscataway, NJ 08854

Assignment 1:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instruction | RegDst | ALUSrc | MemtoReg | RegWrite | MemRead | MemWrite | Branch | ALUOp [1:0] |
| addiu | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1:0 |
| lb | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0:0 |
| or | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1:0 |
| bne | X | 0 | X | 0 | 0 | 0 | 1 | 0:1 |
| j | X | 1? | X | 0 | 0 | 0 | 1 | 0:1 |

Assignment 2:

1)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instruction/cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| lw $t0, 0($t3) | IF | ID | EX | MEM | WB |  |  |  |  |
| add $t1, $t0, $t2 |  | IF | ID | EX | MEM | WB |  |  |  |
| sub $t3, $t3, $t1 |  |  | IF | ID | EX | MEM | WB |  |  |
| addi $t4, $t4, 4 |  |  |  | IF | ID | EX | MEM | WB |  |
| sub $t5, $t5, $t4 |  |  |  |  | IF | ID | EX | MEM | WB |

2)

$t0 = 2, $t1 = 5, $t2 = 8, $t3 = 2, $t4 = 4 (Assuming full forwarding when possible)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Instruction/register | $t0 | $t1 | $t2 | $t3 | $t4 |
| lw $t0, 0($t3) | 2 | 5 | 8 | 2 | 4 |
| add $t1, $t0, $t2 | 2 | 5 | 8 | 2 | 4 |
| sub $t3, $t3, $t1 | 0($t3) | 10 | 8 | 2 | 4 |
| addi $t4, $t4, 4 | 0($t3) | 10 | 8 | -8 | 4 |
| sub $t5, $t5, $t4 | 0($t3) | 10 | 8 | -8 | 8 |

3) (Assuming full forwarding)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instruction/cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| lw $t0, 0($t3) | IF | ID | EX | MEM | WB |  |  |  |  |  |
| NOP |  | X | X | X | X | X |  |  |  |  |
| add $t1, $t0, $t2 |  |  | IF | ID | EX | MEM | WB |  |  |  |
| sub $t3, $t3, $t1 |  |  |  | IF | ID | EX | MEM | WB |  |  |
| addi $t4, $t4, 4 |  |  |  |  | IF | ID | EX | MEM | WB |  |
| sub $t5, $t5, $t4 |  |  |  |  |  | IF | ID | EX | MEM | WB |

4)

Let the instruction after the add go first (the sub), then add. This will remove the hazard without needing to use a NOP. Since full forwarding was assumed, the rest of the commands should work as prescribed.

5)

By use of forwarding, the data needed can be ready for the next command before that command's execution if the previous command did not have to write from memory.

Assignment 3:

1)

If full forwarding is assumed, it would take 7 commands, therefore it would take 11 cycles.

2 & 3)

Because full forwarding is assumed, no hazards exist.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instr./cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| addiu | IF | ID | EX | MEM | WB |  |  |  |  |  |  |
| addiu |  | IF | ID | EX | MEM | WB |  |  |  |  |  |
| bne |  |  | IF | ID | EX | MEM | WB |  |  |  |  |
| addiu |  |  |  | IF | ID | EX | MEM | WB |  |  |  |
| bne |  |  |  |  | IF | ID | EX | MEM | WB |  |  |
| addiu |  |  |  |  |  | IF | ID | EX | MEM | WB |  |
| bne |  |  |  |  |  |  | IF | ID | EX | MEM | WB |

4)

Since forwarding was used, the addiu's can be done in rapid succession, along with the bne. All information is forwarded from the previous execution into the register for the coming execution.

Assignment 4:

It was, in general, hard to understand what was exactly going on with the status and cause registers only because it took so long for the program to get to where it needed, and I usually skipped part accidentally due to impatience. However, what seems to be happening is that for polling there was a loop that consisted of 3 commands, one of which checked the input to see if a key was pressed, and then it would eventually echo that to the console. With interrupts, it also went into a type of loop, but it seems that the loop was needed only to be able to start waiting for an interrupt. Once an interrupt signal was sent to the program, it would echo the correct key press to the console. Interrupts seemed to be disabled while polling, or at least the polling did not get interrupted whatsoever. If interrupts were enabled, then there would be no need to poll. Interrupts are probably enabled by the underlying system which has a thread that is constantly waiting for user input. Once user input is read, that signal is relayed to any program waiting on an interrupt signal, and the information is copied from terminal.

Assignment 5:

1)

Code added at end.

2)

If the program did not support full forwarding, 7 cycles would be saved. Otherwise, no cycles would need saving, since the program does not have memory calls followed by calls that use that memory now stored in a register.