Lab 11

Jiasen Zhou, Jon Johnston

April 14, 2019

1 Executive Summary

This lab had three parts: creating the Write Back stage, integrating it into the datapath, and then testing the datapath with code that performs a simple division. For the Write Back stage, a new module did not have to be created, a mux was simply added to the current datapath. However, an iWriteBack module was created to nest the mux for ease of reading. The division test did not require a new module to be created either, machine code for the division simply had to be created and implemented in the instruction memory file. The reg data and memory files also had to be edited to ensure correct operation. The division of 21 by 7 correctly produced the result of 3, demonstrating that all three parts of the lab were successful.

2 Test Report

To verify operation of these modules, this lab required two test benches.

1. Datapath Test Bench
2. Division Test Bench

2.1 Datapath Test Bench

The Datapath Test Bench contains:

1. Inputs
   (a) reset - reset button that sets PC source to 0
2. Outputs
   (a) read_data1 - data stored in read register 1
   (b) read_data2 - data stored in read register 2
   (c) write_data - the output of the Write Back stage

2.2 Division Test Bench

The Divison Test Bench contains:

1. Inputs
   (a) reset - reset button that sets PC source to 0
2. Outputs
   (a) read_data1 - data stored in read register 1
   (b) read_data2 - data stored in read register 2
   (c) write_data - the output of the Write Back stage

Figure 1: Expected Results of the Datapath test.

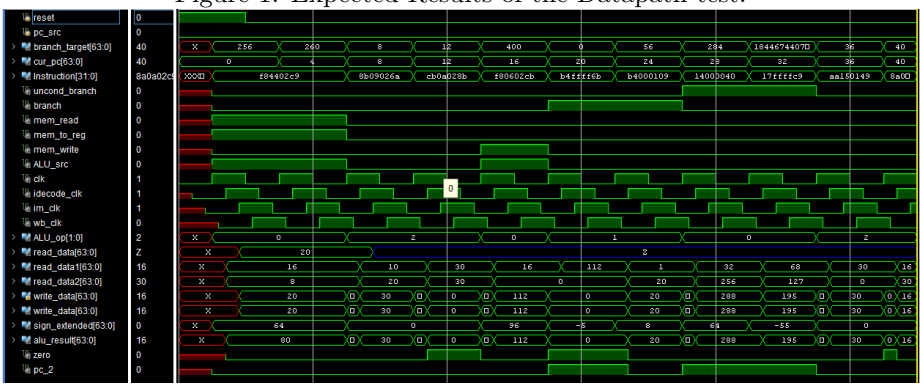

Figure 2: Timing diagram for the Division test.

3 Code Appendix

Listing 1: Verilog code for testing a register.

```
`include "definitions.vh"

module datapath;

    reg reset;
    wire [`INSTR_LEN-1:0] Instruction;
    wire uncond_branch, branch,
         mem_read, mem_to_reg,
         mem_write, ALU_src, clk,
         fetch_clk, idecode_clk,
         im_clk, wb_clk;
    wire [1:0] ALU_op;
```

```
    wire [`WORD-1:0] branch_target, cur_pc,
                     read_data, read_data1, read_data2,
                     write_data, sign_extended, alu_result;
    wire zero, pc_src;

    oscillator r_clk(.clk(clk));       //base clock
    delay #(.DELAYAMT(1)) ClkPlus3(.a(clk),
            .a_delayed(fetch_clk));    //fetch clock
    delay #(.DELAYAMT(2)) ClkPlus2(.a(clk),
            .a_delayed(idecode_clk)); //decode clock
    delay #(.DELAYAMT(4)) ClkPlus6(.a(clk),
            .a_delayed(im_clk));       //memory clock
    delay #(.DELAYAMT(6)) ClkPlus8(.a(clk),
            .a_delayed(wb_clk));       // writeback clock


    fetch fetch_mod(
                .clk(clk),
                .fetch_clk(fetch_clk),
                .reset(reset),
                .branch_target(branch_target),
                .pc_src(pc_src),
                .instruction(Instruction),
                .cur_pc(cur_pc));

    iDecode decode_mod(
                .write_data(write_data),
                .Instruction(Instruction),
                .uncond_branch(uncond_branch),
                .branch(branch),
                .mem_read(mem_read),
                .mem_to_reg(mem_to_reg),
                .mem_write(mem_write),
                .ALU_src(ALU_src),
                .read_clk(idecode_clk),
                .write_clk(wb_clk),
                .ALU_op(ALU_op),
                .read_data1(read_data1),
                .read_data2(read_data2),
                .sign_extended(sign_extended));

    iExecute execute_mod(
                .pc_in(cur_pc),
                .read_data1(read_data1),
                .read_data2(read_data2),
                .sign_extend(sign_extended),
```

```
                .opcode(Instruction[31:21]),
                .alu_op(ALU_op),
                .alu_src(ALU_src),
                .alu_result(alu_result),
                .zero(zero),
                .branch_target(branch_target));

    iMemory memory_mod(
                .im_clk(im_clk),
                .alu_result(alu_result),
                .read_data2(read_data2),
                .mem_read(mem_read),
                .mem_write(mem_write),
                .zero(zero),
                .branch(branch),
                .uncondbranch(uncond_branch),
                .read_data(read_data),
                .pc_src(pc_src));

    iWrite_back writeback_m(
                .read_data(read_data),
                .alu_result(alu_result),
                .MemtoReg(mem_to_reg),
                .write_data(write_data));

initial
    begin
        reset = 1; #10
        reset = 0; #190
   $finish;
    end

endmodule
```