Lab 12: Pipeline Fetch and Decode

Jiasen Zhou, Jon Johnston

April 27, 2019

1 Executive Summary

Fill.

2 Test Report

To verify operation of these modules, this lab required one test bench.

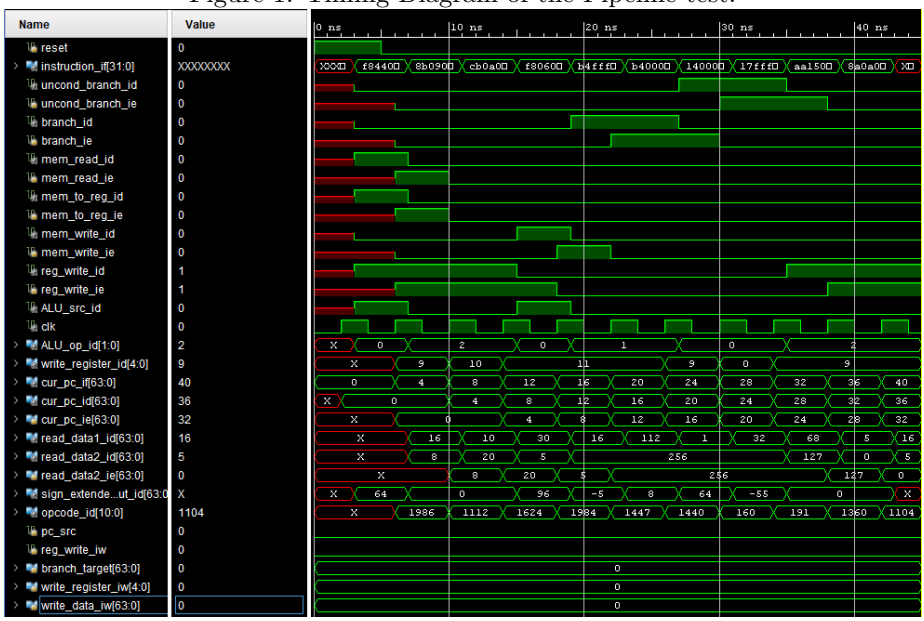
1. Pipeline Test Bench

2.1 Pipeline Test Bench

The Pipeline Test Bench contains:

1. Inputs
   (a) Fill.
2. Outputs
   (a) Fill.

Figure 1: Timing Diagram of the Pipeline test.



3 Code Appendix

Listing 1: Verilog code for testing the pipelined datapath

```
`include "definitions.vh"

module pipeline;

    reg reset;
    wire [`INSTR_LEN-1:0] instruction_if;
    wire uncond_branch_id, branch_id,
         mem_read_id, mem_to_reg_id,
         mem_write_id, reg_write_id,
         ALU_src_id, clk;
         // Future
         /* zero_ie, pc_src, uncond_branch_ie,
         branch_ie, mem_read_ie, mem_to_reg_ie,
         mem_to_reg_im, mem_write_ie, reg_write_ie,
         reg_write_im, reg_write_iw,*/
    wire [1:0] ALU_op_id;
    wire [4:0] write_register_id;
         // Future
         /*write_register_ie, write_register_im,
```

```
        write_register_iw*/
    wire [`WORD-1:0]
        cur_pc_if, cur_pc_id,
        read_data1_id,
        read_data2_id,
        sign_extended_output_id;
        // Future
        /* branch_target, cur_pc_ie, cur_pc_im,
        read_data_im, read_data2_ie, write_data_iw,
        alu_result_ie*/
    wire [10:0] opcode_id;

    // Temporary Registers for Simulation
    reg pc_src, uncond_branch_ie, branch_ie,
        mem_read_ie, mem_to_reg_ie, mem_write_ie,
        reg_write_ie, reg_write_iw;
    reg [4:0] write_register_iw;
    reg [`WORD-1:0] branch_target, cur_pc_ie,
        read_data2_ie, write_data_iw;

    // Base Clock
    oscillator r_clk(.clk(clk));

    // Fetch Stage
    fetch fetch_mod(
        .clk(clk),
        .reset(reset),
        .branch_target(branch_target),
        .pc_src(pc_src),
        .instruction_if(instruction_if),
        .cur_pc_if(cur_pc_if));

    // Decode Stage
    iDecode decode_mod(
        .cur_pc_if(cur_pc_if),
        .cur_pc_id(cur_pc_id),
        .write_data_iw(write_data_iw),
        .write_register_iw(write_register_iw),
        .write_register_id(write_register_id),
        .reg_write_iw(reg_write_iw),
        .reg_write_id(reg_write_id),
        .instruction_if(instruction_if),
        .uncond_branch_id(uncond_branch_id),
        .branch_id(branch_id),
        .mem_read_id(mem_read_id),
        .mem_to_reg_id(mem_to_reg_id),
```

```
        .mem_write_id(mem_write_id),
        .ALU_src_id(ALU_src_id),
        .write_clk(clk),
        .ALU_op_id(ALU_op_id),
        .read_data1_id(read_data1_id),
        .read_data2_id(read_data2_id),
        .sign_extended_output_id(sign_extended_output_id),
        .opcode_id(opcode_id));

    // iExecute Buffer Simulation
    always @(posedge clk)
    begin
        uncond_branch_ie <= uncond_branch_id;
        branch_ie <= branch_id;
        mem_read_ie <= mem_read_id;
        mem_to_reg_ie <= mem_to_reg_id;
        mem_write_ie <= mem_write_id;
        reg_write_ie <= reg_write_id;
        cur_pc_ie <= cur_pc_id;
        read_data2_ie <= read_data2_id;
    end

//    Future Modules
//    iExecute execute_mod(
//        .pc_in(cur_pc),
//        .read_data1(read_data1),
//        .read_data2(read_data2),
//        .sign_extend(sign_extended_output_id),
//        .opcode(opcode_id),
//        .alu_op(ALU_op),
//        .alu_src(ALU_src),
//        .alu_result(alu_result),
//        .zero(zero),
//        .branch_target(branch_target));

//    iMemory memory_mod(
//        .im_clk(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;
        pc_src = 0;
        branch_target = 0;
        reg_write_iw = 0;
        write_register_iw = 0;
        write_data_iw = 0; #5
        reset = 0; #40
   $finish;
    end

endmodule
```