Lab 9: Integrating Fetch and Decode

Jiasen Zhou, Jon Johnston

April 1, 2019

1 Executive Summary

The purpose of this lab is to design and simulate the datapath with the iExecute module integrated into it. Datapath.v now contains the fetch, decode, and execute stages of the datapath. With those modules, the datapath can retrieve instructions, interpret what needs to be done, and pass the necessary information to the ALU. Some signals are declared in the testbench to ensure operation since there are some stages that remain unintegrated. After comparing the Expected Results Table with each module's simulation, the lab was successful.

2 Test Report

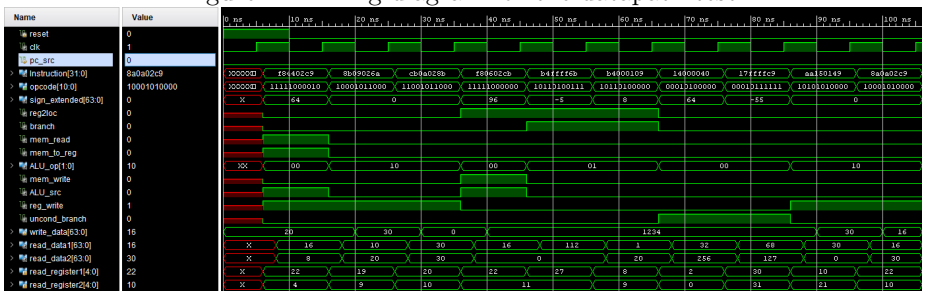
To verify operation of these module, this lab requires one test bench.

1. Datapath Test Bench

Figure 1: Expected Results of the datapath test.

	Instruction 1	Instruction 2	Instruction 3	Instruction 4	Instruction 5	Instruction 6	Instruction 7	Instruction 8	Instruction 9	Instruction 10
instruction	LDUR X9, [X22, #64]	ADD X10, X19, X9	SUB X11, X20, X10	STUR X11, [X22, #96]	CBZ X11, -5	CBZ X9, 8	B 64	B -55	ORR X9, X10, X21	AND X9, X22, X10
Machine Instruction (hex)	F84402C9	8B09026A	CB0A028B	F80602CB	B4FFFF6B	B4000109	14000040	17FFFFC9	AA150149	8A0A02C9
opcode (binary)	11111000010	10001011000	11001011000	11111000000	10110100	10110100	000101	000101	10101010000	10001010000
sign_extended_output (hex)	0000000000040	0000000000000	0000000000000	0000000000060	FFFFFFFFFFFFFB	0000000000008	0000000000040	FFFFFFFFFFFFC9	0000000000000	0000000000000
reg2_loc	0	0	0	1	1	1	0	0	0	0
branch	0	0	0	0	1	1	0	0	0	0
mem_read	1	0	0	0	0	0	0	0	0	0
mem_to_reg	1	0	0	0	0	0	0	0	0	0
alu_op	00	10	10	00	01	01	00	00	10	10
mem_write	0	0	0	1	0	0	0	0	0	0
alu_src	1	0	0	1	0	0	0	0	0	0
reg_write	1	1	1	0	0	0	0	0	1	1
uncondbranch	0	0	0	0	0	0	1	1	0	0
write_data (decimal)	20	30	0	X	X	X	X	X	30	16
read_data1 (decimal)	16	10	30	16	X	X	X	X	30	16
read_data2 (decimal)	X	20	30	0	0	20	X	X	0	30
branch_target	X	4	8	396	-4	52	280	-192	32	36
alu_result	80	30	0	112	0	20	X	X	30	16
zero	0	0	1	0	1	0	0	0	0	0

Figure 2: Timing diagram for the datapath test.



3 Code Appendix

Listing 1: Verilog code for testing the datapath.

```
`include "definitions.vh"

module datapath;

    reg reset, pc_src;
    wire [`WORD-1:0] branch_target, cur_pc;
    wire [`INSTR_LEN-1:0] Instruction;
    reg [`WORD-1:0] write_data;
    wire uncond_branch, branch, mem_read, mem_to_reg, mem_write, ALU_src;
    wire clk, clkplus1, clkplus2, clkplus3, clkplus4, clkplus5, clkplus6;
    wire [1:0] ALU_op;
    wire [`WORD-1:0] read_data1, read_data2, sign_extended;
    wire [`WORD-1:0] alu_result;
    wire zero;

    //fetch takes 2ns to complete. So clk and clkplus1 are used in fetch
```

```
    oscillator r_clk(.clk(clk));
    delay ClkPlus1(.a(clk), .a_delayed(clkplus1));
    delay #(.DELAYAMT(2)) ClkPlus2(.a(clk), .a_delayed(clkplus2));
    delay #(.DELAYAMT(3)) ClkPlus3(.a(clk), .a_delayed(clkplus3));
    delay #(.DELAYAMT(4)) ClkPlus4(.a(clk), .a_delayed(clkplus4));
    delay #(.DELAYAMT(5)) ClkPlus5(.a(clk), .a_delayed(clkplus5));
    delay #(.DELAYAMT(6)) ClkPlus6(.a(clk), .a_delayed(clkplus6));


    fetch iFetch(.clk(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(clkplus3),
                .write_clk(clkplus6),
                .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));
initial
    begin
       reset = 1;
```

```
        pc_src= 0;
        write_data = 20;#10;
        reset = 0;
        write_data = 20;#10;
        write_data = 30;#10;
        write_data = 0;#10;
        write_data = 1234; #50;
        write_data = 30; #10;
        write_data = 16; #6;
    $finish;
     end

endmodule
```