Lab6: Finishing Decode

Justin Roessler and Jon Johnston

February 27, 2019

1 Executive Summary

The purpose of this lab is to finish the decode stage of the processor and to implement a datapath.v module that will be the top module of our proccessor. For this lab datapath.v will only include the fetch stage and the decode stage of the proccessor. Other needed signals for the datapath to operate were explicitly stated in the module so datapath will operate as expected.

The Decode stage combines the different pieces of the stage into one module. This module will break down the instruction and assign all the control signals to the processor and store the needed values into the registers. The decode stage is the final stage before an instruction is executed and holds the register memory of the processor. After comparing the Expected Results Table with each module's simulation, the lab was successful.

2 Test Report

To verify operation of these modules, this lab requires 2 test benches.

1. iDecode Test Bench
2. Datapath Test Bench

Figure 1: Expected Results for Lab 6.

	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	01	01	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

Figure 2: Simulation Results for the Decode.

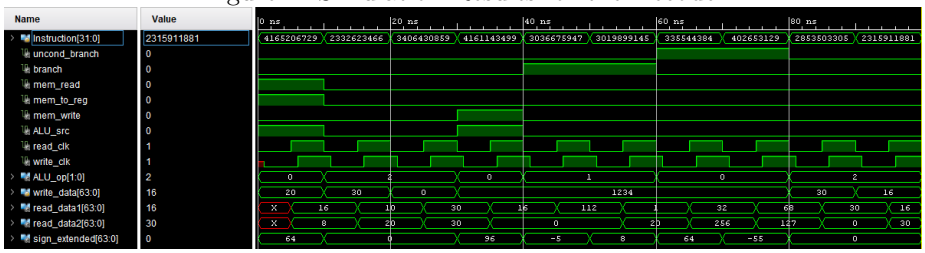
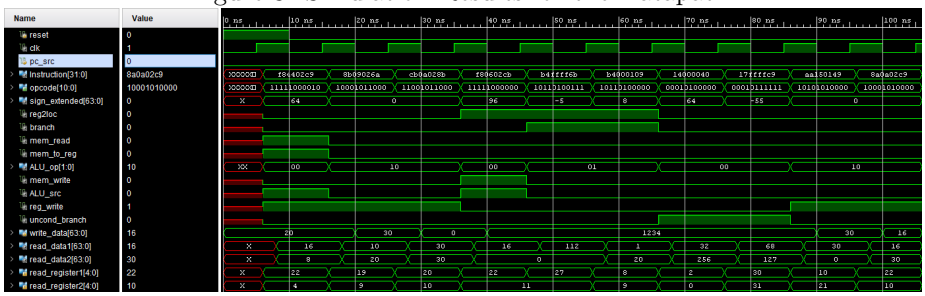


Figure 3: Simulation Results for the Datapath.



3 Code Appendix

Listing 1: Verilog code for testing the Decode Stage.

```
`include "definitions.vh"

module decode_test;

    reg [`INSTR_LEN-1:0] Instruction;
    reg [`WORD-1:0] write_data;
    wire uncond_branch, branch, mem_read, mem_to_reg, mem_write, ALU_src, read_c
    wire [1:0] ALU_op;
    wire [`WORD-1:0] read_data1, read_data2, sign_extended;
    wire [10:0] opcode;

    oscillator r_clk(.clk(read_clk));

    delay w_clk(.a(read_clk), .a_delayed(write_clk));

    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(read_clk),
                       .write_clk(write_clk),
                       .ALU_op(ALU_op),
                       .read_data1(read_data1),
                       .read_data2(read_data2),
                       .sign_extended(sign_extended),
                       .opcode(opcode));


    initial
        begin
        write_data = 20;
        Instruction = 32'hF84402C9; #10;
        write_data = 30;
        Instruction = 32'h8B09026A; #10;
        write_data = 0;
        Instruction = 32'hCB0A028B; #10;
        write_data = 1234;
        Instruction = 32'hF80602CB; #10;
```

```
        write_data = 1234;
        Instruction = 32'hB4FFFF6B; #10;
        write_data = 1234;
        Instruction = 32'hB4000109; #10;
        write_data = 1234;
        Instruction = 32'h14000040; #10;
        write_data = 1234;
        Instruction = 32'h17FFFFC9; #10;
        write_data = 30;
        Instruction = 32'hAA150149; #10;
        write_data = 16;
        Instruction = 32'h8A0A02C9; #10;
        $finish;
        end
endmodule
```

Listing 2: Verilog code for implementing the Decode Stage.

```
`include "definitions.vh"

module iDecode(
    input [`WORD-1:0] write_data,
    input [`INSTR_LEN-1:0] Instruction,
    output uncond_branch, branch, mem_read, mem_to_reg, mem_write, ALU_src, read
    output [1:0] ALU_op,
    output reg [10:0] opcode,
    output [`WORD-1:0] read_data1, read_data2, sign_extended
    );

    wire reg2loc, reg_write;
    wire [4:0] read_reg2;

    always @*
        begin
            opcode <= Instruction[31:21];
        end

    control cont_mod(.instruction(Instruction),
                     .Reg2Loc(reg2loc),
                     .Branch(branch),
                     .MemRead(mem_read),
                     .MemtoReg(mem_to_reg),
                     .MemWrite(mem_write),
                     .ALUSrc(ALU_src),
                     .RegWrite(reg_write),
                     .UncondBranch(uncond_branch),
```

```
                        .ALUOp(ALU_op));

    sign_extender sign_ex(.Instruction(Instruction), .Sign_Extended(sign_extende

    mux #(.SIZE(5)) read_mux(.a_in(Instruction[20:16]), .b_in(Instruction[4:0]),

    regfile reg_mod(.read_register1(Instruction[9:5]),
                    .read_register2(read_reg2),
                    .write_register(Instruction[4:0]),
                    .RegWrite(reg_write),
                    .read_clk(read_clk),
                    .write_clk(write_clk),
                    .write_data(write_data),
                    .read_data1(read_data1),
                    .read_data2(read_data2));
endmodule
```

Listing 3: Verilog code for implementing the Datapath.

```
`include "definitions.vh"

module datapath;

    reg reset, pc_src;
    reg [`WORD-1:0] branch_target;
    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;

    //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));

    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));

    initial
    begin
        reset = 1;
        pc_src= 0;
        write_data = 20;
        branch_target= 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
```