Lab 9: Integrating Fetch and Decode

Jiasen Zhou, Jon Johnston

March 27, 2019

1 Executive Summary

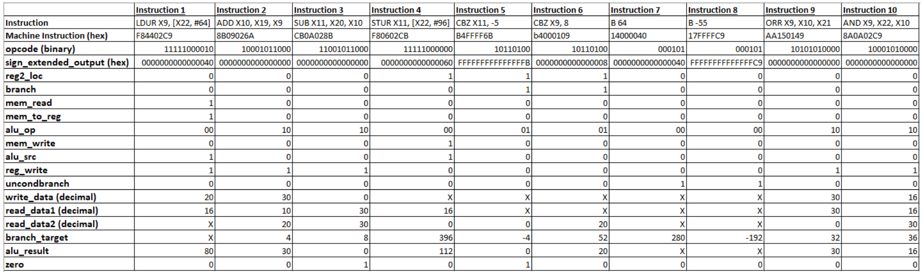
The Executive Summary section should be a single concise paragraph that describes the following items. Please note that this should be a paragraph, not an itemized list like I have below. The list is just to tell you what I'm looking for...it is not there for you to fill it in on your lab report. See the Lab 1 Example Report.pdf on Canvas to see a good example of what I'm looking for.

2 Test Report

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

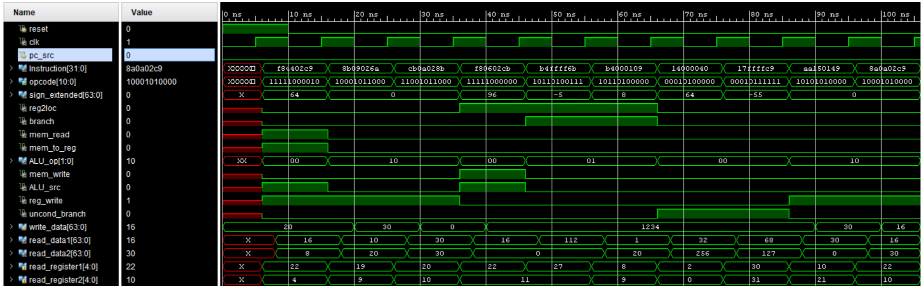
1. datapath

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)	0000000000000040	0000000000000000	0000000000000000	0000000000000060	FFFFFFFFFFFFFFFB	0000000000000008	0000000000000040	FFFFFFFFFFFFFFC9	0000000000000000	0000000000000000
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.



Name	Value
reset	0
clk	1
pc_src	0
Instruction[31:0]	8a0a02c9
opcode[10:0]	10001010000
sign_extended[63:0]	0
reg2loc	0
branch	0
mem_read	0
mem_to_reg	0
ALU_op[1:0]	10
mem_write	0
ALU_src	0
reg_write	1
uncond_branch	0
write_data[63:0]	16
read_data1[63:0]	16
read_data2[63:0]	30
read_register1[4:0]	22
read_register2[4:0]	10

0 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
80 ns
90 ns
100 ns
XXXXX f84402c9 8b09026a cb0a028b f80602cb b4ffff6b b4000109 14000040 17ffffc9 aa150149 8a0a02c9
XXXXX 11111000010 10001011000 11001011000 11111000000 10110100111 10110100000 00010100000 00010111111 10101010000 10001010000
X 64 0 96 -5 8 64 -55 0
XX 00 10 00 01 00 10
20 30 0 1234 30 16
X 16 10 30 16 112 1 32 68 30 16
X 8 20 30 0 20 256 127 0 30
X 22 19 20 22 27 8 2 30 10 22
X 4 9 10 11 9 0 31 21 10

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
```