Lab Assignment 6

Due: 10 Nov 2020

By: Alex Oswald
EECE 2323

Verilog Code for `pdatapath_top_lab5.v`:

```
1.  `timescale 1ns / 1ps
2.  //////////////////////////////////////////////////////////////////////////////////
3.  // Company:
4.  // Engineer: Majid Sabbagh (sabbagh.m@husky.neu.edu)
5.  //
6.  // Create Date: 08/17/2014 02:18:36 PM
7.  // Design Name:
8.  // Module Name: eightbit_alu_top
9.  // Project Name:
10. // Target Devices:
11. // Tool Versions:
12. // Description:
13. //
14. // Dependencies:
15. //
16. // Revision:
17. // Revision 0.01 - File Created
18. // Additional Comments:
19. //
20. //////////////////////////////////////////////////////////////////////////////////
21. module pdatapath_top(
22.         input wire clk,
23.         input wire top_pb_clk,
24.         input wire rst_general,
25.         output [7:0] led
26.     );
27.
28.     wire [7:0] alu_1st_input, alu_2nd_input;
29.     wire [7:0] alu_output;
30.     wire [2:0] ALUOp;
31.     wire       alu_ovf_flag;
32.     wire       alu_take_branch_output;
33.
34.     wire [15:0] instruction;
35.     //insturction fields
36.     wire [3:0] opcode;
37.     wire [1:0] rs_addr;
38.     wire [1:0] rt_addr;
39.     wire [1:0] rd_addr;
40.     wire [7:0] immediate;
41.     //control signals
42.     wire RegDst;
43.     wire RegWrite;
44.     wire ALUSrc1;
45.     wire ALUSrc2;
46.     wire MemWrite;
47.     wire MemToReg;
48.
49.     wire [1:0] regfile_write_address;//destination register address
50.     wire [8:0] regfile_write_data;//result data
51.     wire [8:0] read_data1;//source register1 data
52.     wire [8:0] read_data2;//source register2 data
53.
54.     wire [8:0] alu_result;
55.     wire [8:0] zero_register;
56.     wire [8:0] data_mem_out;
57.
58.     wire pb_clk_debounced;
```

```
59.
60.
61.     assign alu_result = {alu_ovf_flag, alu_output};
62.     assign led = alu_result;
63.     debounce debounce_clk(
64.         .clk_in(clk),
65.         .rst_in(rst_general),
66.         .sig_in(top_pb_clk),
67.         .sig_debounced_out(pb_clk_debounced)
68.     );
69.
70.
71.     //Instantiate Your instruction decoder here
72.     decoder decode (
73.         .instruction(instruction),
74.         .opcode(opcode),
75.         .rs_addr(rs_addr),
76.         .rt_addr(rt_addr),
77.         .rd_addr(rd_addr),
78.         .immediate(immediate),
79.         .RegDst(RegDst),
80.         .RegWrite(RegWrite),
81.         .ALUSrc1(ALUSrc1),
82.         .ALUSrc2(ALUSrc2),
83.         .ALUOp(ALUOp),
84.         .MemWrite(MemWrite),
85.         .MemToReg(MemToReg)
86.         );
87.
88.
89.     //Instantiate Your alu-regfile here
90.     alu_regfile ALU_RegFile (
91.         .ReadData1(read_data1),
92.         .ReadData2(read_data2),
93.         .ALUsrc1(ALUSrc1),
94.         .ALUsrc2(ALUSrc2),
95.         .ALUop(ALUOp),
96.         .Instr_i(immediate),
97.         .zero_register(zero_register),
98.
99.         .rst(rst_general),
100.                .clk(pb_clk_debounced),
101.                .wr_en(RegWrite),
102.                .rd0_addr(rs_addr),
103.                .rd1_addr(rt_addr),
104.                .wr_addr(regfile_write_address),
105.                .wr_data(regfile_write_data),
106.
107.                .input_1(alu_1st_input),
108.                .input_2(alu_2nd_input),
109.                .result(alu_result),
110.                .ovf(alu_ovf_flag),
111.                .take_branch(alu_take_branch_output)
112.                );
113.
114.           //Instantiate Your data memory here
115.           data_memory datamem (
116.               .rst(rst_general),
117.               .clk(pb_clk_debounced),
118.               .write_enable(MemWrite),
119.               .addr(alu_output),  // ALU_RESULT or ALU_OUTPUT ??
```

```
120.                .write_data(read_data2),
121.                .read_data(data_mem_out)
122.                );
123.
124.            //Mux for regfile_write_data (MemToReg)
125.            assign regfile_write_data = MemToReg ? data_mem_out : alu_result;
126.
127.            //Mux for regfile_write_address (RegDst)
128.            assign regfile_write_address = RegDst ? rt_addr : rd_addr;
129.
130.            //Instantiate Your VIO core here
131.            vio_0 vio (
132.                .clk(clk),                          //inputwireclk
133.
134.                .probe_in0(regfile_write_data),     //[8:0]WriteData
135.                .probe_in1(read_data1),             //[7:0]ReadData1
136.                .probe_in2(read_data2),             //[7:0]ReadData2
137.                .probe_in3(alu_1st_input),          //[7:0]input1
138.                .probe_in4(alu_2nd_input),          //[7:0]input2
139.                .probe_in5(alu_take_branch_output), //[0:0]take_branch
140.                .probe_in6(alu_ovf_flag),           //[0:0]alu_ovf
141.                .probe_in7(opcode),                 //[3:0]opcode
142.                .probe_in8(alu_output),             //[7:0]alu_out
143.                .probe_in9(data_mem_out),           //[8:0]DataMemOut
144.
145.                .probe_out0(instruction)            //[15:0]instruction
146.                );
147.
148.        endmodule
```

Verilog Code for `decoder.v`:

```
1.  module decoder(
2.      input [15:0] instruction,
3.      output [3:0] opcode,
4.      output [1:0] rs_addr,
5.      output [1:0] rt_addr,
6.      output [1:0] rd_addr,
7.      output [7:0] immediate,
8.      output reg RegDst,
9.      output reg RegWrite,
10.     output reg ALUSrc1,
11.     output reg ALUSrc2,
12.     output reg [2:0] ALUOp,
13.     output reg MemWrite,
14.     output reg MemToReg);
15.
16.
17.     assign opcode = instruction[15:12];
18.     assign rs_addr = instruction[11:10];
19.     assign rt_addr = instruction[9:8];
20.     assign rd_addr = instruction[7:6];
21.     assign immediate = instruction[7:0];
22.
23.
24.     always @(opcode)
25.     begin
26.         case(opcode)
27.             4'b0000:  begin     // LW
```

```
28.                     RegDst <= 1'b0;  RegWrite <= 1'b1;    ALUSrc1 <= 1'b0; ALUSrc2 <= 1
    'b1; ALUOp <= 3'b000; MemWrite <= 1'b0;    MemToReg <= 1'b1;
29.                 end
30.             4'b0001:  begin     // SW
31.                     RegDst <= 1'b0;  RegWrite <= 1'b0;    ALUSrc1 <= 1'b0; ALUSrc2 <= 1
    'b0; ALUOp <= 3'b000; MemWrite <= 1'b1;    MemToReg <= 1'b0;
32.                 end
33.             4'b0010:  begin     // ADD
34.                     RegDst <= 1'b1;  RegWrite <= 1'b1;    ALUSrc1 <= 1'b0; ALUSrc2 <= 1
    'b0; ALUOp <= 3'b000; MemWrite <= 1'b0;    MemToReg <= 1'b0;
35.                 end
36.             4'b0011:  begin     // ADDI
37.                     RegDst <= 1'b0;  RegWrite <= 1'b1;    ALUSrc1 <= 1'b0; ALUSrc2 <= 1
    'b1; ALUOp <= 3'b000; MemWrite <= 1'b0;    MemToReg <= 1'b0;
38.                 end
39.             4'b0100:  begin     // INV
40.                     RegDst <= 1'b1;  RegWrite <= 1'b1;    ALUSrc1 <= 1'b0; ALUSrc2 <= 1
    'b0; ALUOp <= 3'b001; MemWrite <= 1'b0;    MemToReg <= 1'b0;
41.                 end
42.             4'b0101:  begin     // AND
43.                     RegDst <= 1'b1;  RegWrite <= 1'b1;    ALUSrc1 <= 1'b0; ALUSrc2 <= 1
    'b0; ALUOp <= 3'b010; MemWrite <= 1'b0;    MemToReg <= 1'b0;
44.                 end
45.             4'b0110:  begin     // ANDI
46.                     RegDst <= 1'b0;  RegWrite <= 1'b1;    ALUSrc1 <= 1'b0; ALUSrc2 <= 1
    'b1; ALUOp <= 3'b010; MemWrite <= 1'b0;    MemToReg <= 1'b0;
47.                 end
48.             4'b0111:  begin     // OR
49.                     RegDst <= 1'b1;  RegWrite <= 1'b1;    ALUSrc1 <= 1'b0; ALUSrc2 <= 1
    'b0; ALUOp <= 3'b011; MemWrite <= 1'b0;    MemToReg <= 1'b0;
50.                 end
51.             4'b1000:  begin     // ORI
52.                     RegDst <= 1'b0;  RegWrite <= 1'b1;    ALUSrc1 <= 1'b0; ALUSrc2 <= 1
    'b1; ALUOp <= 3'b011; MemWrite <= 1'b0;    MemToReg <= 1'b0;
53.                 end
54.             4'b1001:  begin     // SRA
55.                     RegDst <= 1'b0;  RegWrite <= 1'b1;    ALUSrc1 <= 1'b0; ALUSrc2 <= 1
    'b0; ALUOp <= 3'b100; MemWrite <= 1'b0;    MemToReg <= 1'b0;
56.                 end
57.             4'b1010:  begin     // SLL
58.                     RegDst <= 1'b0;  RegWrite <= 1'b1;    ALUSrc1 <= 1'b0; ALUSrc2 <= 1
    'b0; ALUOp <= 3'b101; MemWrite <= 1'b0;    MemToReg <= 1'b0;
59.                 end
60.             4'b1011:  begin     // BEQ
61.                     RegDst <= 1'b0;  RegWrite <= 1'b0;    ALUSrc1 <= 1'b0; ALUSrc2 <= 1
    'b0; ALUOp <= 3'b110; MemWrite <= 1'b0;    MemToReg <= 1'b0;
62.                 end
63.             4'b1100:  begin     // BNE
64.                     RegDst <= 1'b0;  RegWrite <= 1'b0;    ALUSrc1 <= 1'b0; ALUSrc2 <= 1
    'b0; ALUOp <= 3'b111; MemWrite <= 1'b0;    MemToReg <= 1'b0;
65.                 end
66.             4'b1101:  begin     // CLR
67.                     RegDst <= 1'b1;  RegWrite <= 1'b1;    ALUSrc1 <= 1'b1; ALUSrc2 <= 1
    'b0; ALUOp <= 3'b010; MemWrite <= 1'b0;    MemToReg <= 1'b0;
68.                 end
69.             default:  begin     // just do adding i guess
70.                     RegDst <= 1'b0;  RegWrite <= 1'b1;    ALUSrc1 <= 1'b0; ALUSrc2 <= 1
    'b1; ALUOp <= 3'b000; MemWrite <= 1'b0;    MemToReg <= 1'b1;
71.                 end
72.         endcase
73.     end
```

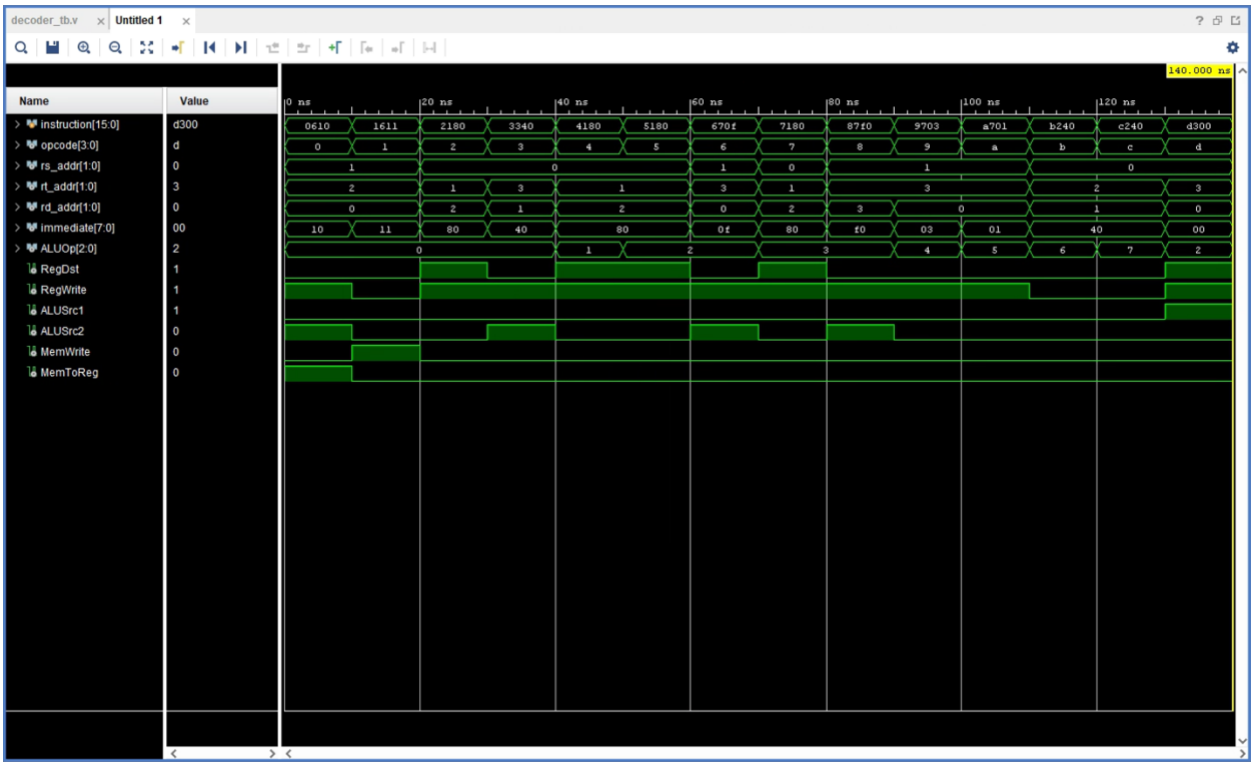
```
74.
75. endmodule
```

Verilog Code for `decoder_tb.v`:

```
1.  module decoder_tb();
2.
3.      // inputs
4.      reg [15:0] instruction;
5.      // outputs
6.      wire [3:0] opcode;
7.      wire [1:0] rs_addr, rt_addr, rd_addr;
8.      wire [7:0] immediate;
9.      wire [2:0] ALUOp;
10.     wire RegDst, RegWrite, ALUSrc1, ALUSrc2, MemWrite, MemToReg;
11.
12.
13.     decoder uut (
14.         .instruction(instruction),
15.         .opcode(opcode),
16.         .rs_addr(rs_addr),
17.         .rt_addr(rt_addr),
18.         .rd_addr(rd_addr),
19.         .immediate(immediate),
20.         .RegDst(RegDst),
21.         .RegWrite(RegWrite),
22.         .ALUSrc1(ALUSrc1),
23.         .ALUSrc2(ALUSrc2),
24.         .ALUOp(ALUOp),
25.         .MemWrite(MemWrite),
26.         .MemToReg(MemToReg));
27.
28.
29.     initial
30.     begin
31.         instruction = 16'b0000_01_10_00_010000;#10; // load reg[1] <= mem[2+0x10]
32.         instruction = 16'b0001_01_10_00_010001;#10; // store mem[2+0x11] <= reg[1]
33.         instruction = 16'b0010_00_01_10_000000;#10; // add reg[2] <= reg[0] + reg[1]
34.         instruction = 16'b0011_00_11_01_000000;#10; // addi reg[3] <= reg[0] + 0x40
35.         instruction = 16'b0100_00_01_10_000000;#10; // inv reg[2] <= ~reg[1]
36.         instruction = 16'b0101_00_01_10_000000;#10; // and reg[2] <= reg[0] & reg[1]
37.         instruction = 16'b0110_01_11_00_001111;#10; // andi reg[3] <= reg[1] & 0xF
38.         instruction = 16'b0111_00_01_10_000000;#10; // or reg[2] <= reg[0] | reg[1]
39.         instruction = 16'b1000_01_11_11_110000;#10; // ori reg[3] <= reg[1] | 0xF0
40.         instruction = 16'b1001_01_11_00_000011;#10; // sra reg[3] <= reg[1] << 0x3;
41.         instruction = 16'b1010_01_11_00_000001;#10; // sll reg[3] <= reg[1] >>> 0x1;
42.         instruction = 16'b1011_00_10_01_000000;#10; // beq pc+0x40 <= reg[0] == reg[2]

43.         instruction = 16'b1100_00_10_01_000000;#10; // bne pc-
    0xC0 <= reg[0] != reg[2]
44.         instruction = 16'b1101_00_11_00_000000;#10; // clr reg[3] <= 0
45.         $finish;
46.         $monitor("%d instruction=%h opcode=%b immediate=%h rs_addr=%h rt_addr=%h rd_add
    r=%h | RegWrite=%d RegDst=%d ALUSrc1=%d ALUSrc2=%d ALUOp=%d MemWrite=%d MemToReg=%d",
47.             $time, instruction, opcode, immediate, rs_addr, rt_addr, rd_addr, RegWrite,
     RegDst, ALUSrc1, ALUSrc2, ALUOp, MemWrite, MemToReg);
48.     end
49.
50. endmodule
```

Testbench results:



VIO Output Visualization:

Evidence is located in attached `lab6-test.mp4` file. The following sequence was followed. The BTN1 was pressed between each instruction to trigger storing the resultant in either data memory of the register while BTN0 was pressed after step 5 to reset the board.

```
1. inv $1 $1              (0x4140)
2. sll $1, $1, 0x03       (0xa502)
3. sw $1, 0xFF($3)        (0x1dff)
4. addi $2, $3, 0xFF      (0x3eff)
5. ori $2, $2, 0xF0       (0x8af0)
```