**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. );
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;
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;
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
54. wire [8:0] alu\_result;
55. wire [8:0] zero\_register;
56. wire [8:0] data\_mem\_out;
58. wire pb\_clk\_debounced;

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. );

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. );

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),
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),
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. );
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. );
124. //Mux for regfile\_write\_data (MemToReg)
125. assign regfile\_write\_data = MemToReg ? data\_mem\_out : alu\_result;
127. //Mux for regfile\_write\_address (RegDst)
128. assign regfile\_write\_address = RegDst ? rt\_addr : rd\_addr;
130. //Instantiate Your VIO core here
131. vio\_0 vio (
132. .clk(clk),                          //inputwireclk
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
145. .probe\_out0(instruction)            //[15:0]instruction
146. );
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);

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];

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
75. endmodule

**Verilog Code for decoder\_tb.v:**

1. module decoder\_tb();
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;

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));

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\_addr=%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
50. endmodule

**Testbench results:**

**A screen shot of a computer

Description automatically generated**

**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)