Jack Kapino

Cs 354- Digital Systems Design

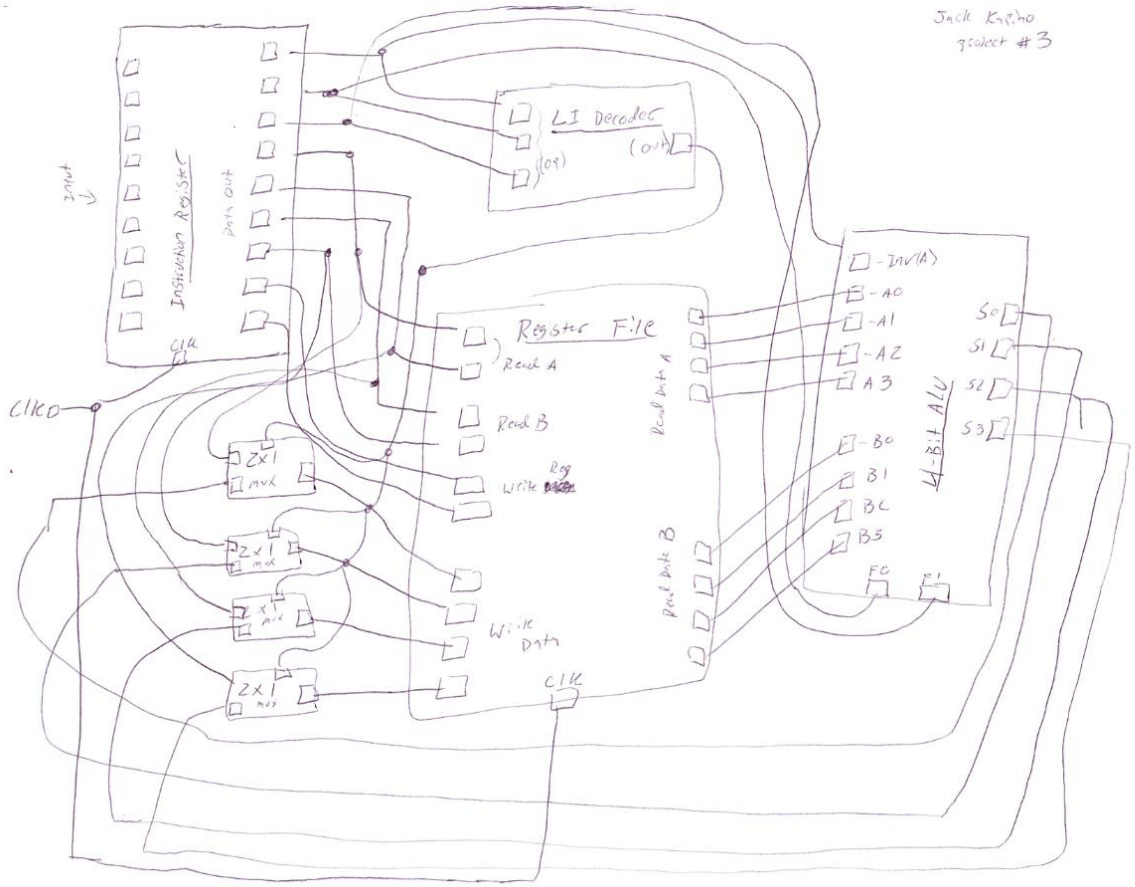
Project 3:

Report:

The Verilog implementation of a 4 -bit cpu project uses a 4- bit Alu from project 2 along with the following modules. A 9- bit instruction register using nine D Flip-Flops, including a clock pulse input that connects to all the D-Flip-Flops. A register module which includes four -bit registers. The Li instruction decoder module, the combinational circuit output is 0 when the Li opcode input is (100) otherwise output is 1. A module for the quadruple 2x1 multiplexer. Along with a test module that is created and loaded with instructions to match corresponding binary values for each instruction.

The 4-bit cpu supports the following operations (Add, Subtract, and, or, SLT, Load immediate). These instructions are loaded into the instruction register. There are two types of instructions R-Type and I-Type both of which are of size 9 bits. The load immediate decoder is used along with the multiplexer to determine which type of instruction has been loaded. Most operations are performed by the Alu, data is moved around during each clock pulse to the register file. The Register file uses a select line to determine which write data line it should connect to.

Diagram:



Verilog Code:

```
//Half adder

module half_adder(x,y,S,C);

 input x,y;

 output S, C;


 xor x1(S,x,y);

 and a1(C,x,y);


endmodule
```

```
//Full adder
module full_adder(x,y,z,S,C);
 input x,y,z;
 output S, C;

 half_adder ha1(x,y,u,v);
 half_adder ha2(u,z,S,w);

 or o1(C,v,w);

endmodule

//2:1 Mux project 2
module two_oneMux(sel,op,out);
 input sel;
 input [1:0] op;
 output out;

 not n1(inverted, sel);
 and a1(and1, op[0], inverted);
 and a2(and2, op[1], sel);
 or o1(out, and1, and2);

endmodule
```

```
//4:1 Mux Project 2
module muxFour_One(sel,operation,out);
 input [1:0] sel;
 input [3:0] operation;
 output out;

 two_oneMux muxOne(sel[0], {operation[0], operation[1]}, one);
 two_oneMux muxTwo(sel[0], {operation[2], operation[3]}, two);
 two_oneMux muxThree(sel[1], {one, two}, out);

endmodule

//1bit alu from project 2
module One_Bit_Alu(a,b,cin,opcode,binvert,less,result,cout);
 input a,b,cin;
 input [1:0] opcode;
 input binvert,less;
 output result,cout;

  not notb(bbar, b);
  two_oneMux mux2_1(binvert, {bbar, b}, binverted);
  muxFour_One mux(opcode, {and0, or1, sum2, less}, result);
```

```
 full_adder fa1( a,binverted, cin, sum2, cout);
 and a1( and0, a, binverted);
 or o1( or1, a, binverted);

endmodule

module Overflow(a,b,total,overflow);
 input a, b, total;
 output overflow;

 not n1(aprime, a);
 not n2(bprime, b);
 not n3(totalprime, total);

 and a1(and1, aprime, bprime);
 and a2(negative_sum, and1, total);
 and a3(and2, a,b);
 and a4(positive_sum, and2, totalprime);
 or o1(overflow, negative_sum, positive_sum);

endmodule

module Msb_Alu(a,b,cin,opcode,binvert,less,set,result,overflow);
 input a,b, cin;
 input [1:0] opcode;
```

```
 input binvert,less;
 output set,result,overflow;

  not n1(bbar, b);
  two_oneMux inverter(binvert, {bbar, b}, binverted);
  muxFour_One mux(opcode, {out1, out2, sum3, less}, result);
  full_adder fa1(a, binverted, cin, sum3, cout);
  or or_msb(out2, a, binverted);
  and and_msb(out1, a, binverted);

  not n2(outpt,sum3);
  not n3(set,outpt);
  Overflow over1(a, binverted, sum3, overflow);

endmodule


module Four_bitAlu(a,b,opcode,result,overflow,zero);
 input [3:0] a,b;
 input [2:0] opcode;
 output [3:0] result;
 output overflow,zero;

  One_Bit_Alu alu0(a[0],b[0],opcode[2],opcode[1:0],opcode[2],set,result[0],carry1);
  One_Bit_Alu alu1(a[1],b[1],carry1,opcode[1:0],opcode[2],1'b0,result[1],carry2);
```

```
  One_Bit_Alu alu2(a[2],b[2],carry2,opcode[1:0],opcode[2],1'b0,result[2],carry3);
  Msb_Alu alu3(a[3],b[3],carry3, opcode[1:0],opcode[2], 1'b0 , set, result[3], overflow_out);

 and a1(overflow, overflow_out, opcode[1]);
  or o1(left, result[0], result[1]);
  or o2(right, result[2], result[3]);
  or o3(notzero, left, right);
  not n1(zero, notzero);

endmodule

// Used for 9 bit instruction register
module D_latch(D,C,Q);  //Code D_Flip_Flop.vl
   input D,C;
   output Q;
   wire x,y,D1,Q1;
   nand nand1 (x,D, C),
        nand2 (y,D1,C),
        nand3 (Q,x,Q1),
        nand4 (Q1,y,Q);
   not  not1  (D1,D);
endmodule


module D_flip_flop(D,CLK,Q);  //Code from D_Flip_Flop.Vl
```

```
  input D,CLK;
  output Q;
  wire CLK1, Y;
  not  not1 (CLK1,CLK);
  D_latch D1(D,CLK, Y),
        D2(Y,CLK1,Q);
endmodule


module regfile (ReadReg1,ReadReg2,WriteReg,WriteData,ReadData1,ReadData2,CLK);
 input [1:0] ReadReg1,ReadReg2,WriteReg;
 input [3:0] WriteData;
 input CLK;
 output [3:0] ReadData1,ReadData2;
 reg [3:0] Regs[0:3];
 assign ReadData1 = Regs[ReadReg1];
 assign ReadData2 = Regs[ReadReg2];
 initial Regs[0] = 0;
 always @(negedge CLK)
   Regs[WriteReg] <= WriteData;
endmodule


module instr_reg(Instruction,IR,clk);
 input [8:0] Instruction;
```

```
  input clk;
  output [8:0] IR;

  D_flip_flop D0(Instruction[0],clk, IR[0]);
  D_flip_flop D1(Instruction[1],clk, IR[1]);
  D_flip_flop D2(Instruction[2],clk, IR[2]);
  D_flip_flop D3(Instruction[3],clk, IR[3]);
  D_flip_flop D4(Instruction[4],clk, IR[4]);
  D_flip_flop D5(Instruction[5],clk, IR[5]);
  D_flip_flop D6(Instruction[6],clk, IR[6]);
  D_flip_flop D7(Instruction[7],clk, IR[7]);
  D_flip_flop D8(Instruction[8],clk, IR[8]);

endmodule

// Part 4 - Li Instruction decoder Module:
module Li_Instruction_Decoder(op,out);
   input [2:0] op;
   output out;

   not n0(invert0, op[0]);
   not n1(invert1, op[1]);
   and a1(Low, invert0, invert1);
   and a2(out, Low, op[2]);
```

```
endmodule


module cpu(instruction,clk,writeData);
 input [8:0] instruction;
 input clk;
 output [3:0] writeData;
 wire [8:0] IR;
 wire [3:0] a,b,result;

 //Instruction Register
 instr_reg instruct_Reg1(instruction, IR,clk);
 //Register file
 regfile RegisterFile1(IR[5:4], IR[3:2], IR[1:0], writeData, a, b,clk);

 //   WriteData multiplexer
 two_oneMux twoOneMux0(load_Result,{IR[2], result[0]}, writeData[0] );
 two_oneMux twoOneMux1(load_Result,{IR[3], result[1]}, writeData[1] );
 two_oneMux twoOneMux2(load_Result,{IR[4], result[2]}, writeData[2] );
 two_oneMux twoOneMux3(load_Result,{IR[5], result[3]}, writeData[3] );

 //ALU
 Four_bitAlu FourBit_Alu(a,b,IR[8:6],result,overflow,zero);

//load immediate Decoder
Li_Instruction_Decoder liDecoder(IR[8:6], load_Result);
```

```
endmodule



module test_cpu;
 reg [8:0] Instruction;
 reg clk;
 wire signed [3:0] WriteData;

 cpu cpu1( Instruction,clk, WriteData);

 initial
  begin

   #1; Instruction = 9'b100111110;  // Load 15 -> $2
     clk = 1;
   #1; clk = 0; // negative edge - execute instruction



   #1; Instruction = 9'b100100011; // Load 8 -> $3
     clk = 1;
   #1; clk = 0;  // negative edge - execute instruction



   #1; Instruction = 9'b000101101; // $2 & $3 -> $1
```

```
  clk = 1;
#1; clk = 0;    // negative edge - execute instruction


#1; Instruction = 9'b110011011;  // $1 - $2 -> $3
  clk = 1;
#1; clk = 0; // negative edge - execute instruction


#1; Instruction = 9'b111110010; // SLT $2, $3, $0
  clk = 1;
#1; clk = 0;    // negative edge - execute instruction


#1; Instruction = 9'b010101101; // $2 + $3 -> $1
  clk = 1;
#1; clk = 0;  // negative edge - execute instruction


#1; Instruction = 9'b110000101; // $0 - $1 -> $1
  clk = 1;
#1; clk = 0;   // negative edge - execute instruction


#1; Instruction = 9'b001011011; // $1 | $2 -> $3
```

```
    clk = 1;
  #1; clk = 0;  // negative edge - execute instruction


  end


 initial
 begin
    $display("\nCLK Instruction WriteData\n------------------------");
    $monitor("%b   %b   %b  %d", clk, Instruction, WriteData,WriteData);
 end


endmodule
```

Program output:

Result

CPU Time: 0.00 sec(s), Memory: 7588 kilobyte(s)

```
CLK Instruction WriteData
------------------------
x   xxxxxxxx   xxxx   x
1   100111110   xxxx   x
0   100111110   1111  -1
1   100100011   1111  -1
0   100100011   1000  -8
1   000101101   1000  -8
0   000101101   1000  -8
1   110011011   1000  -8
0   110011011   1001  -7
1   111110010   1001  -7
0   111110010   0001   1
1   010101101   0001   1
0   010101101   1010  -6
1   110000101   1010  -6
0   110000101   0110   6
1   001011011   0110   6
0   001011011   0111   7
```