CECS 225

Lab 6 : MIPS Datapath

Sotheanith Sok

1. **Datapath Verilog Module source**

`timescale 1ns / 1ps

module datapath( clk,

reset,

memtoreg,

pcsrc,

alusrc,

regdst,

regwrite,

jump,

alucontrol,

zero,

pc,

instr,

aluresult,

writedata,

readdata);

input clk, reset, memtoreg, pcsrc, alusrc, regdst, regwrite, jump;

input [2:0] alucontrol;

input [31:0] instr, readdata;

output zero;

output [31:0] pc, aluresult, writedata;

wire [4:0] writereg;

wire [31:0] pcnext, pcnextbr, pcplus4, pcbranch;

wire [31:0] signimm, signimmsh;

wire [31:0] srca, srcb, result;

/\*\*\*\*\*\*\*\*\*\* next PC logic \*\*\*\*\*\*\*\*\*\*/

// The program counter

flopr pcreg ( .clk (clk),

.reset (reset),

.d (pcnext[7:0]),

.q (pc[7:0])

);

// Adder to increment the PC value

Adder\_32 pcadd1 ( .augend (pc),

.addend (31'h4),

.sum (pcplus4)

);

// shift left the sign extended constant

s12 immsh ( .a (signimm),

.y (signimmsh)

);

// branch address adder

Adder\_32 pcadd2 ( .augend (signimmsh),

.addend (pcplus4),

.sum (pcbranch)

);

// branch mux

mux2\_32 pcbrmux( .in0 (pcnextbr),

.in1 ({pcplus4[31:28], instr[25:0], 2'b00}),

.sel (jump),

.out (pcnext)

);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\* register file logic \*\*\*\*\*\*\*\*\*\*/

// The register file

regfile rf ( .clk (clk),

.we3 (regwrite),

.ra1 (instr[25:21]),

.ra2 (instr[20:16]),

.wa3 (writereg),

.wd3 (result),

.rd1 (srca),

.rd2 (writedata)

);

// destination register selection mux

mux2\_5 wrmux ( .in0 (instr[20:16]),

.in1 (instr[15:11]),

.sel (regdst),

.out (writereg)

);

// register data input selection mux

mux2\_32 resmux( .in0 (aluresult),

.in1 (readdata),

.sel (memtoreg),

.out (result)

);

// sign extender to make 32-bit constant

signext se ( .a (instr[15:0]),

.y (signimm)

);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\* ALU logic \*\*\*\*\*\*\*\*\*\*/

// ALU input B selection mux

mux2\_32 srcbmux( .in0 (writedata),

.in1 (signimm),

.sel (alusrc),

.out (srcb)

);

// The Arithmetic Logic Unit

ALU alu ( .ALUControl(alucontrol),

.A(srca),

.B(srcb),

.Result(aluresult),

.Zero\_Flag(zero)

);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

endmodule

1. **Datapath text fixture**

`timescale 1ns / 1ps

module datapath\_tester;

// Inputs

reg clk;

reg reset;

reg memtoreg;

reg pcsrc;

reg alusrc;

reg regdst;

reg regwrite;

reg jump;

reg [2:0] alucontrol;

reg [31:0] instr;

reg [31:0] readdata;

// Outputs

wire zero;

wire [31:0] pc;

wire [31:0] aluresult;

wire [31:0] writedata;

//internal variables

reg [5:0] Opcode, func;

reg [4:0] rs, rt, rd;

reg [15:0] imm;

// Instantiate the Unit Under Test (UUT)

datapath uut (

.clk(clk),

.reset(reset),

.memtoreg(memtoreg),

.pcsrc(pcsrc),

.alusrc(alusrc),

.regdst(regdst),

.regwrite(regwrite),

.jump(jump),

.alucontrol(alucontrol),

.zero(zero),

.pc(pc),

.instr(instr),

.aluresult(aluresult),

.writedata(writedata),

.readdata(readdata)

);

assign pc[31:8] = 0; //upper 24 bits of program counter not used

always #5 clk = ~clk; //clock pulse generation

integer i;

initial begin

//initialize contents of registers

uut.rf.rf[0] = 0;

uut.rf.rf[11] = 50;

uut.rf.rf[12] = 100;

uut.rf.rf[13] = 150;

uut.rf.rf[14] = 200;

uut.rf.rf[15] = 250;

uut.rf.rf[31] = 32'hFFFFFFFF;

// Initialize Inputs

clk = 0; // initialzie clock signal to 0

reset = 0; // reset PC so it is initialized to 0

memtoreg = 0;

pcsrc = 0;

alusrc = 0;

regdst = 0;

regwrite = 0;

jump = 0;

alucontrol = 0;

instr = 0;

readdata = 0;

// Wait until positive clock edge for program counter reset to occur

@(posedge clk);

#1 reset = 0; // de-assert the reset signal for the program counter

@(negedge clk)#1;

$display("Begin MIPS datapath Test");

$display("");

$display("Show initial register contents");

regdump(); //call reg dump task to display register contents

//Initialize control signal values fro R-Type Instructions

memtoreg = 0; // pass ALUResult to Register File

pcsrc = 0; // not a branch so load PC with pcplus4

alusrc = 0; // use read data 2 output from Register File

regdst = 1; // rd field determines destination register

regwrite = 1; // R-Type instruction will write to register file

jump = 0; // not a J-Type instruction

readdata = 32'hX; // Data from DMEM only used in lw

Opcode = 6'h00; // R-Type instruction opcode

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Test case 0: add $16, $14, $12

@(posedge clk)

alucontrol = 2; // make ALU do 'add' - Table 7.2 2

rs = 14; // $14 is first source register

rt= 12; // $12 is the second source register

rd = 16; // $16 is the destination register

func = 6'h20; // 'add' function code

instr = {Opcode, rs, rt, rd, 5'h0, func};

@(negedge clk) #1;

$display("Test case 0");

$display("check result of add $16, $14, $12");

$display("Expected Result: $16 = %h", uut.rf.rf[14]+uut.rf.rf[12]);

$display("Actual Result: $%d = %h", rd, uut.rf.rf[16] );

testCaseOutcome( uut.rf.rf[14]+uut.rf.rf[12], uut.rf.rf[16]);

$display("" );

//Test case 1: sub $17, $15, $11

@(posedge clk)

alucontrol = 6; // make ALU do 'sub' - Table 7.2 2

rs = 15; // $15 is first source register

rt= 11; // $11 is the second source register

rd = 17; // $17 is the destination register

func = 6'h22; // ''sub'' function code

instr = {Opcode, rs, rt, rd, 5'h0, func};

@(negedge clk) #1;

$display("Test case 1");

$display("check result of sub $17, $15, $11");

$display("Expected Result: $17 = %h", uut.rf.rf[15]-uut.rf.rf[11]);

$display("Actual Result: $%d = %h", rd, uut.rf.rf[rd] );

testCaseOutcome( uut.rf.rf[15]-uut.rf.rf[11], uut.rf.rf[rd]);

$display("" );

//Test case 2: and $18, $31, $0

@(posedge clk)

alucontrol = 0; // make ALU do 'and' - Table 7.2 2

rs = 31; // $31 is first source register

rt= 0; // $0 is the second source register

rd = 18; // $18 is the destination register

func = 6'h24; // ''and'' function code

instr = {Opcode, rs, rt, rd, 5'h0, func};

@(negedge clk) #1;

$display("Test case 2");

$display("check result of and $18, $31, $0");

$display("Expected Result: $18 = %h", uut.rf.rf[31]&uut.rf.rf[0]);

$display("Actual Result: $%d = %h", rd, uut.rf.rf[rd] );

testCaseOutcome( uut.rf.rf[31]&uut.rf.rf[0], uut.rf.rf[rd]);

$display("" );

//Test case 3: or $19, $17, $31

@(posedge clk)

alucontrol = 1; // make ALU do 'or' - Table 7.2 2

rs = 17; // $17 is first source register

rt= 31; // $31 is the second source register

rd = 19; // $19 is the destination register

func = 6'h25; // ''or'' function code

instr = {Opcode, rs, rt, rd, 5'h0, func};

@(negedge clk) #1;

$display("Test case 3");

$display("check result of or $19, $17, $31");

$display("Expected Result: $19 = %h", uut.rf.rf[19]|uut.rf.rf[31]);

$display("Actual Result: $%d = %h", rd, uut.rf.rf[rd] );

testCaseOutcome( uut.rf.rf[19]|uut.rf.rf[31], uut.rf.rf[rd]);

$display("" );

//Test case 4 : slt $20, $17, $16

@(posedge clk)

alucontrol = 7; // make ALU do 'slt' - Table 7.2 2

rs = 17; // $17 is first source register

rt= 16; // $16 is the second source register

rd = 20; // $20 is the destination register

func = 6'h2A; // ''slt'' function code

instr = {Opcode, rs, rt, rd, 5'h0, func};

@(negedge clk) #1;

$display("Test case 4");

$display("check result of slt $20, $17, $16");

$display("Expected Result: $20 = %h", {{31'b0},uut.rf.rf[17]<uut.rf.rf[16]});

$display("Actual Result: $%d = %h", rd, uut.rf.rf[rd] );

testCaseOutcome( {{31'b0},uut.rf.rf[17]<uut.rf.rf[16]}, uut.rf.rf[rd]);

$display("" );

//Test case 5 : slt $21, $16, $17

@(posedge clk)

alucontrol = 7; // make ALU do 'slt' - Table 7.2 2

rs = 16; // $16 is first source register

rt= 17; // $17 is the second source register

rd = 21; // $21 is the destination register

func = 6'h2A; // ''slt'' function code

instr = {Opcode, rs, rt, rd, 5'h0, func};

@(negedge clk) #1;

$display("Test case 5");

$display("check result of slt $21, $16, $17");

$display("Expected Result: $21 = %h", {{31'b0},{uut.rf.rf[16]<uut.rf.rf[17]}});

$display("Actual Result: $%d = %h", rd, uut.rf.rf[21] );

testCaseOutcome( {{31'b0},{uut.rf.rf[16]<uut.rf.rf[17]}}, uut.rf.rf[21]);

$display("" );

$display("Show final register contents");

regdump(); //call reg dump task to display register contents

$display("End MIPS datapath Test");

//halt simulation

@(negedge clk) #1;

$finish;

end

task regdump;

begin

$display("Scan Registers for valid data");

$display("The following registers have data");

for (i=0; i<32; i=i+1)

if(uut.rf.rf[i]||uut.rf.rf[i] == 32'h0)

$display("register%d = 32'h%h", i, uut.rf.rf[i]);

$display("");

end

endtask //end of regdump

task testCaseOutcome;

input [31:0] input1, input2;

if(input1 == input2)

$display("Test Passed!");

else

begin

$display("Test Failed");

checksignals();

end

endtask // end of test case outcome task

task checksignals;

begin //check signal values for correctness within datapath

//Check Register Destination Selection Mux values

$display("All signals for tested instruction");

$display("RegDst Mux Signals");

$display("\tData Inputs");

$display("\t\tIn0 = %h\tIn1 = %h", uut.instr[20:16], uut.instr[15:11]);

$display("\tControl Inputs");

$display("\t\tSel = %b", uut.regdst);

$display("\tOutputs");

$display("\t\tOut = %h", uut.writereg);

//Check Register Destination Selection Mux values

$display("Register FIle Signals");

$display("\Data Inputs");

$display("\t\trs = %h\trt + %h\tWriteReg = %h\tWriteData = %h",uut.instr[25:21], uut.instr[20:16], uut.writereg, uut.writedata);

$display("\tControl Inputs");

$display("\t\tRegwrite = %b", uut.regwrite);

$display("\tOutputs");

$display("\t\tRD1 = %h\tRD2 = %h", uut.srca, uut.writedata);

//Sign Extender

$display("Sign Extender Signals");

$display("\tData Inputs");

$display("\t\tInstr[15:0] = %h", uut.instr[15:0]);

$display("\tOutputs");

$display("\t\tSignImm = %h", uut.signimm);

//ALU Source B selection mux

$display("ALUSrc Mux Signals");

$display("\tData Inputs");

$display("\t\tIn0 = %hIn1 = %h",uut.writedata, uut.signimm);

$display("\tControl Inputs");

$display("\t\tSel = %h", uut.alusrc);

$display("\tOutputs");

$display("\t\tOUt = %h", uut.srcb);

//ALU data signals

$display("ALU Signals:");

$display("\tData Inputs");

$display("\t\tSrcA = %h \tSrcb = %h", uut.srca, uut.srcb);

$display("\tControl Inputs");

$display("\t\tALUControl = %h",uut.alucontrol);

$display("\tOutputs");

$display("\t\tALUResult = %h\tZero = %h",uut.aluresult, uut.zero);

$display("");

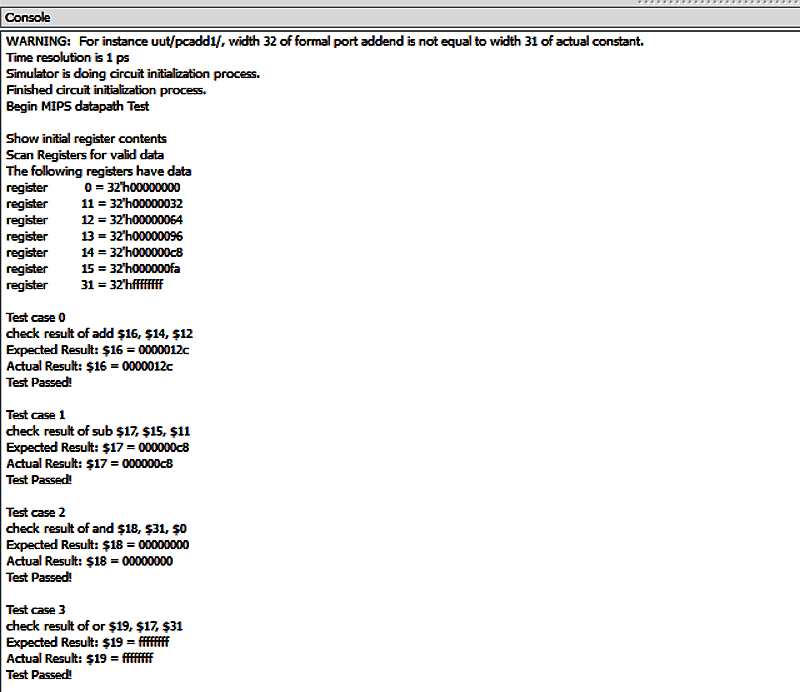
$display("");

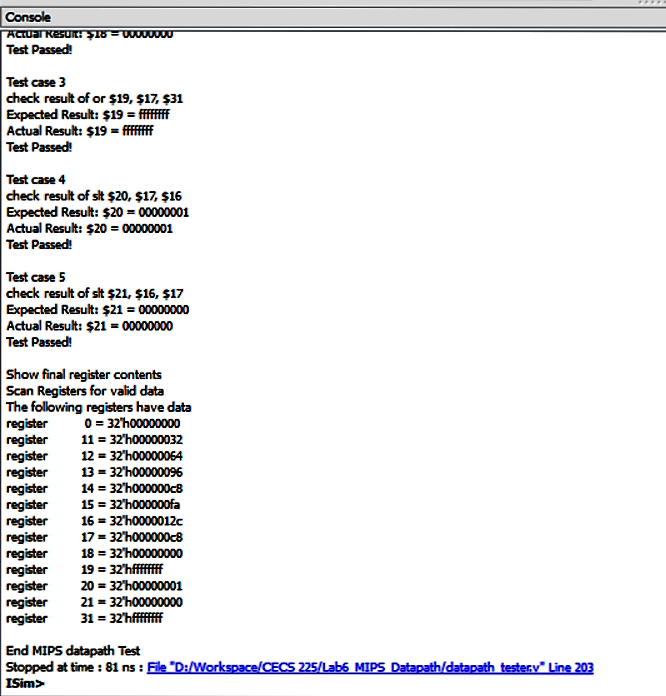
end

endtask //end of check signals

endmodule

1. **Datapath Test Fixture Simulation Console Messages**



****