**CS 224**

**Lab 3**

**Preliminary Design Report**

Group

Cüneyt EREM

21202398

**jalr:**

IM[PC]

PC ← RF[rs]

RF[rd] ← PC + 4

**push:**

IM[PC]

DM[RF[rs] +SignExt(imm)] ←RF[rt]

RF[rs] ←RF[rs] – 4

PC ← PC + 4

**bge:**

IM[PC]

PC ← RF[rs] >= RF[rt] ? PC + 4 + SignExt(imm) || 00: PC + 4

**swapRM:**

IM[PC]

RF[rs] ← M[RF[rt] + immed[15:0]] ← RF[rt]

PC ← PC + 4

**List of The Modules that Changed**

mips

controller

maindec

datapath

**Base mips module**

module mips (input clk, reset,

output [31:0] pc,

input [31:0] instr,

output memwrite,

output [31:0] aluout, writedata,

input [31:0] readdata);

wire memtoreg, pcsrc, zero,

alusrc, regdst, regwrite, jump;

wire [2:0] alucontrol;

controller c (instr[31:26], instr[5:0], zero,

memtoreg, memwrite, pcsrc,

alusrc, regdst, regwrite, jump,

alucontrol);

datapath dp (clk, reset, memtoreg, pcsrc,

alusrc, regdst, regwrite, jump,

alucontrol, zero, pc, instr,

aluout, writedata, readdata);

endmodule

**Modified mips module**

module mips (input clk, reset,

output [31:0] pc,

input [31:0] instr,

output memwrite,

output [31:0] aluout, writedata,

input [31:0] readdata);

wire pcsrc, zero,alusrc, regdst, regwrite;

wire [1:0] memtoreg, jump;

wire [2:0] alucontrol;

controller c (instr[31:26], instr[5:0], zero,

memwrite, pcsrc,

alusrc, regdst, regwrite, jump, memtoreg,

alucontrol);

datapath dp (clk, reset, pcsrc,

alusrc, regdst, regwrite, memtoreg, jump,

alucontrol, zero, pc, instr,

aluout, writedata, readdata);

endmodule

**Base controller module**

module controller(input [5:0] op, funct,

input zero,

output memtoreg, memwrite,

output pcsrc, alusrc,

output regdst, regwrite,

output jump,

output [2:0] alucontrol);

wire [1:0] aluop;

wire branch;

maindec md (op, regwrite, regdst, alusrc, branch,

memwrite, memtoreg, aluop, jump);

aludec ad (funct, aluop, alucontrol);

assign pcsrc = branch & zero;

endmodule

**Modified controller module**

module controller(input [5:0] op, funct,

input zero,

output memwrite,

output pcsrc, alusrc,

output regdst, regwrite,

output[1:0] jump, memtoreg,

output [2:0] alucontrol);

wire [1:0] aluop;

wire branch;

maindec md (op, regwrite, regdst, alusrc, branch,

memwrite, memtoreg, aluop, jump);

aludec ad (funct, aluop, alucontrol);

assign pcsrc = branch & zero;

endmodule

**Base maindec module**

module maindec(input [5:0] op,

output regwrite, regdst,

output alusrc, branch,

output memwrite, memtoreg,

output [1:0] aluop,

output jump);

reg [8:0] controls;

assign {regwrite, regdst, alusrc, branch,

memwrite, memtoreg, aluop, jump} = controls;

always @(\*)

case(op)

6'b000000: controls <= 9'b110000100; //R-type

6'b100011: controls <= 9'b101001000; //LW

6'b101011: controls <= 9'b001010000; //SW

6'b000100: controls <= 9'b000100010; //BEQ

6'b001000: controls <= 9'b101000000; //ADDI

6'b000010: controls <= 9'b000000001; //J

default: controls <= 9'bxxxxxxxxx; //???

endcase

endmodule

**Modified maindec module**

module maindec(input [5:0] op,

output regwrite, regdst,

output alusrc, branch,

output memwrite,

output [1:0] memtoreg ,aluop, jump

reg [10:0] controls;

assign {regwrite, regdst, alusrc, branch,

memwrite, memtoreg, aluop, jump} = controls;

always @(\*)

case(op)

6'b000000: controls <= 11'b11000001000; //R-type

6'b000001: controls <= 11'b00000000000; //nop

6'b100011: controls <= 11'b10100100000; //LW

6'b101011: controls <= 11'b00101000000; //SW

6'b000100: controls <= 11'b00010000100; //BEQ

6'b001000: controls <= 11'b10100000000; //ADDI

6'b000010: controls <= 11'b00000000001; //J

6'b000010: controls <= 11'b11000010010; //jalr

6'b000110: controls <= 11'b00010001100; //bge

6'b101100: controls <= 11'b11101110000; //push

//6'b101100: controls <= 11'b00000000000; //swaprm

default: controls <= 11'bxxxxxxxxxxx; //???

endcase

endmodule

**Base datapath module**

module datapath(input clk, reset,

input memtoreg, pcsrc,

input alusrc, regdst,

input regwrite, jump,

input [2:0] alucontrol,

output zero,

output [31:0] pc,

input [31:0] instr,

output [31:0] aluout, writedata,

input [31:0] readdata);

wire [4:0] writereg;

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

wire [31:0] signimm, signimmsh;

wire [31:0] srca, srcb;

wire [31:0] result;

// next PC logic

flopr #(32) pcreg(clk, reset, pcnext, pc);

adder pcadd1(pc, 32'b100, pcplus4);

sl2 immsh(signimm, signimmsh);

adder pcadd2(pcplus4, signimmsh, pcbranch);

mux2 #(32) branchmux(pcplus4, pcbranch, pcsrc,

pcnextbr);

mux2 #(32) jumpmux(pcnextbr, {pcplus4[31:28],

instr[25:0], 2'b00},

jump, pcnext);

// register file logic

regfile rf(clk, regwrite, instr[25:21],

instr[20:16], writereg,

result, srca, writedata);

mux2 #(5) w\_addrmux(instr[20:16], instr[15:11],

regdst, writereg);

mux2 #(32) w\_datamux(aluout, readdata,

memtoreg, result);

signext se(instr[15:0], signimm);

// ALU logic

mux2 #(32) srcbmux(writedata, signimm, alusrc,

srcb);

alu alu(srca, srcb, alucontrol,

aluout, zero);

endmodule

**Modified datapath module**

module datapath(input clk, reset,

input pcsrc,

input alusrc, regdst,

input regwrite,

input [1:0] memtoreg, jump,

input [2:0] alucontrol,

output zero,

output [31:0] pc,

input [31:0] instr,

output [31:0] aluout, writedata,

input [31:0] readdata);

wire [4:0] writereg;

wire [31:0] pcnext, pcnextbr, pcplus4, pcbranch,readminus4;

wire [31:0] signimm, signimmsh;

wire [31:0] srca, srcb;

wire [31:0] result;

// next PC logic

flopr #(32) pcreg(clk, reset, pcnext, pc);

adder pcadd1(pc, 32'b100, pcplus4);

sl2 immsh(signimm, signimmsh);

adder pcadd2(pcplus4, signimmsh, pcbranch);

adder rmin4( 32'hFFFFFFFC, aluout, readminus4); //for push

mux2 #(32) branchmux(pcplus4, pcbranch, pcsrc,

pcnextbr);

mux3 #(32) jumpmux(pcnextbr, {pcplus4[31:28], instr[25:0], 2'b00}, srca, jump, pcnext);

regfile rf(clk, regwrite, instr[25:21],

instr[20:16], writereg,

result, srca, writedata);

mux2 #(5) w\_addrmux(instr[20:16], instr[15:11],

regdst, writereg);

mux4 #(32) w\_datamux(aluout, readdata, pcplus4, readminus4, memtoreg, result);

signext se(instr[15:0], signimm);

// ALU logic

mux2 #(32) srcbmux(writedata, signimm, alusrc,

srcb);

alu alu(srca, srcb, alucontrol,

aluout, zero);

endmodule

**Test Code**

8'h00: instr = 32'h20020005; //addi $2,$0,5

8'h04: instr = 32'h2003000c; //addi $3,$0,12

8'h08: instr = 32'h2067fff7; //addi $7,$3,-9

8'h0c: instr = 32'h00e22025; //or $4,$7,$2

8'h10: instr = 32'h00642824; //and $5,$3,$4

8'h14: instr = 32'h00a42820; //add $5,$5,$4

8'h18: instr = 32'h10a7000a; //beq $5,$7, 0x10

8'h1c: instr = 32'h0064202a; //slt $4,$3,$4

8'h20: instr = 32'h10800001; //beq $5, $0, 0x28

8'h24: instr = 32'h20050000; //addi $5,$0,0

8'h28: instr = 32'h00e2202a; //slt $4,$7,$2

8'h2c: instr = 32'h00853820; //add $7,$4,$5

8'h30: instr = 32'h00e23822; //sub $7,$7,$2

8'h34: instr = 32'hac670044; //sw $7,68($3)

8'h38: instr = 32'h8c020050; //lw $2,80($0)

8'h3c: instr = 32'h08000011; //j 0x44

8'h40: instr = 32'h20020001; //addi $2,$0,1

8'h44: instr = 32'hac020054; //sw $2,84($0)

//end of base instructions

8'h48: instr = 32'h2008000a; //addi t0, $0, 10

8'h4c: instr = 32'h20090005; //addi t1, $0, 5

8'h50: instr = 32'h19280001; //bge t1, t0, 2 //it shouldn't jbranch

8'h54: instr = 32'h19090002; //bge t0, t1, 1

8'h58: instr = 32'h201d003c; //addi sp, zero, 60

8'h5c: instr = 32'hb3a90000; //push t1

default: instr = {32{1'bx}}; // unknown instruction

endcase

endmodule

**Testbench**

module testbench();

reg clk;

reg reset;

wire [31:0] writedata, dataadr, pc, readdata,instr;

wire memwrite;

// instantiate device to be tested

top dut(clk, reset, writedata, dataadr, pc, readdata,instr, memwrite);

// initialize test

initial

begin

reset <= 1; # 22; reset <= 0;

end

// generate clock to sequence tests

always

begin

clk <= 1; # 5; clk <= 0; # 5;

end

endmodule