

***Design Methodology***

The purpose of this lab is to understand how a single-cycle MIPS processor is designed. The files provided within the archive included the basic design, a testbench, and a memfile. The memfile contained machine code, allowing the processor to be functionally verified. The signals which required to be checked and probed during the simulations are: clk, rst, pc\_current, instr, alu\_out, we\_dm, wd\_dm, rd\_dm. Provided below are block diagrams for the control unit, datapath, complete processor, and instruction memory and data. Also provided are screenshots of simulation, and console output for the eye-balling test bench. The learning outcome for this lab was to understand the basic building blocks of a MIPS processor, and to fully understand how to integrate machine code instructions within the Xilinx simulation software.

Tasks which were completed in lab:

* Review MIPS instructions.
* Create block diagrams for:
  + Datapath
  + Control Unit
  + Instruction Memory and Data Memory
  + Processor Core
  + Complete Processor
* Create a DUT and an eyeballing test bench to verify the program through waveform simulation.

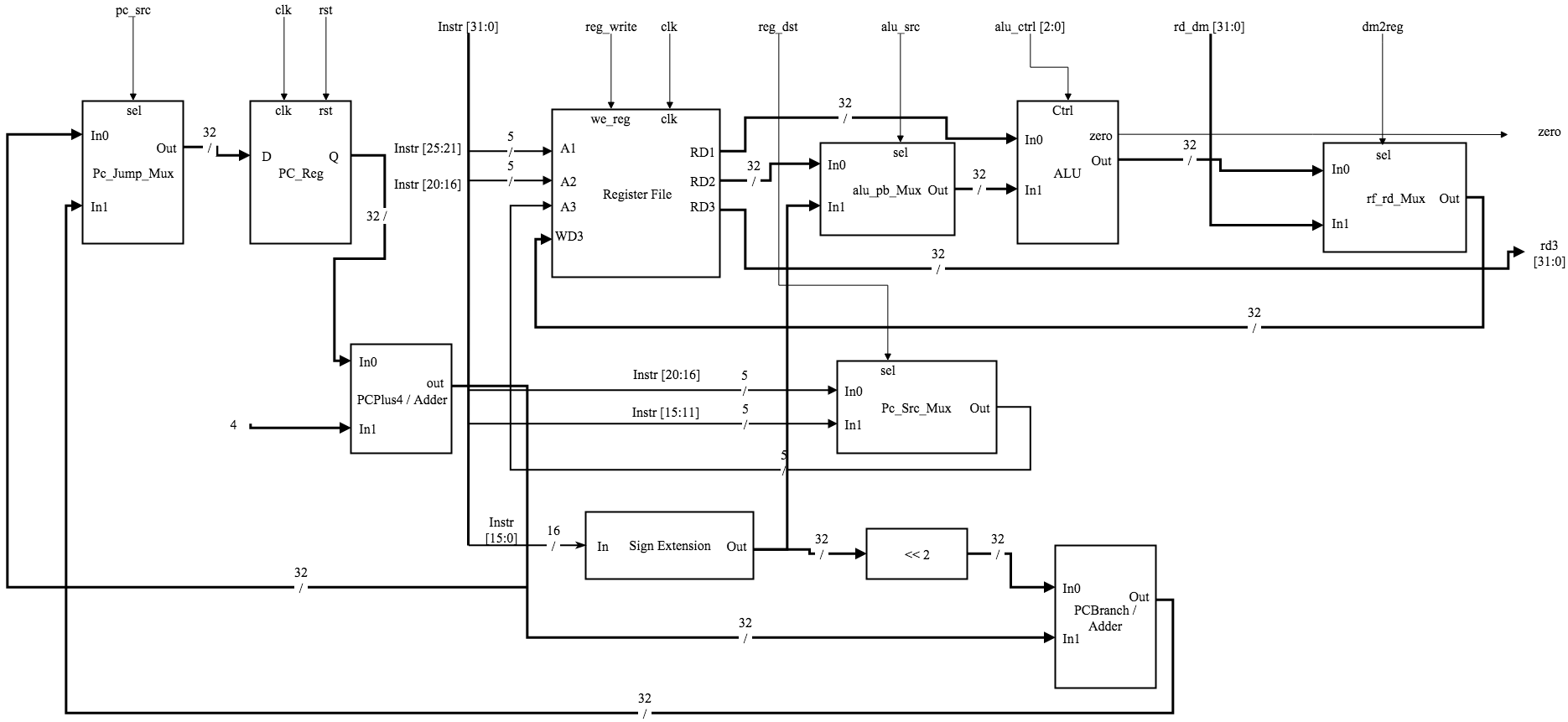


Figure 1. Data Path Diagram

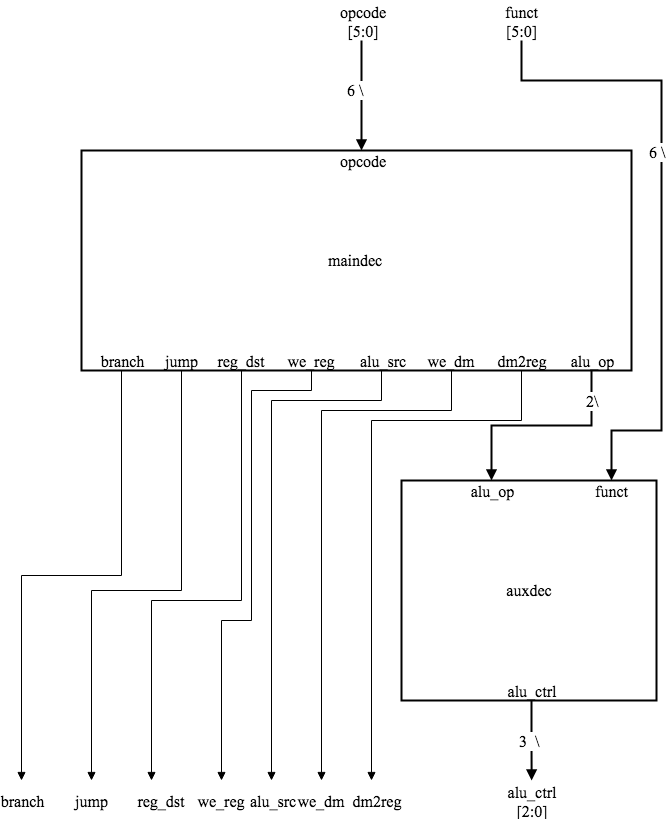


Figure 2. Control Unit Diagram

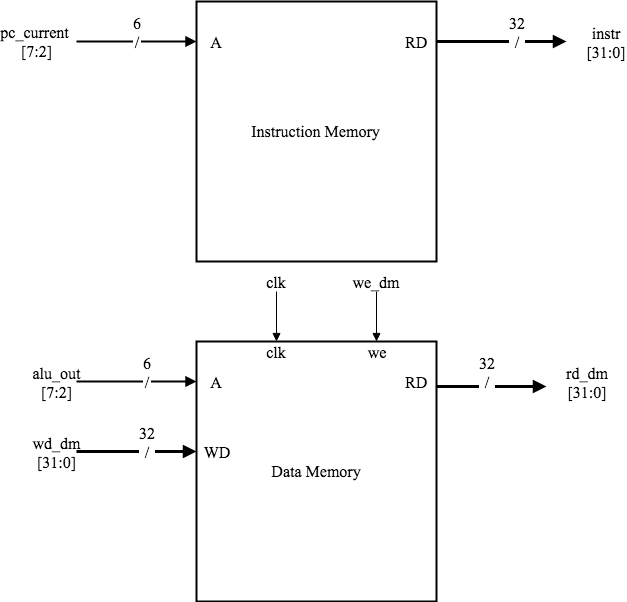


Figure 3. Instruction Memory and Data Memory Diagram

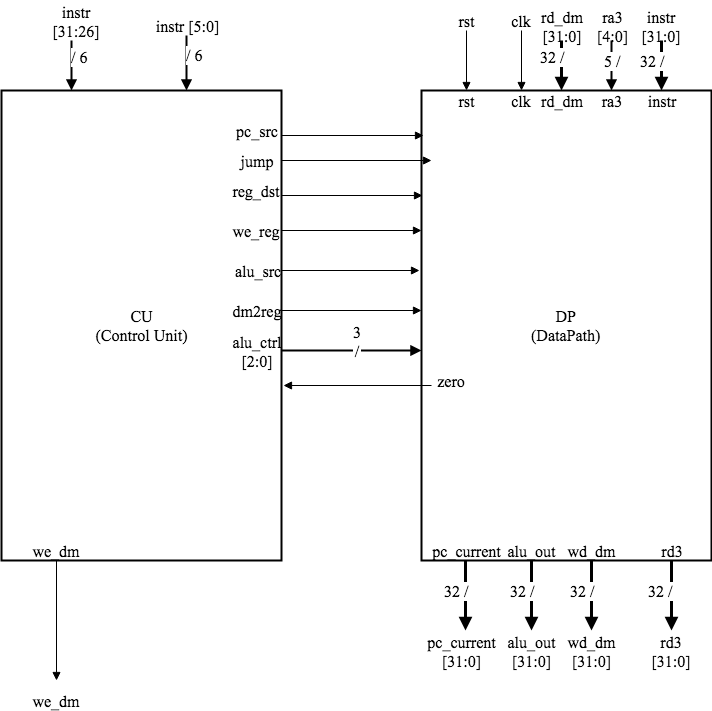


Figure 4. Control Unit and Data Path Diagram

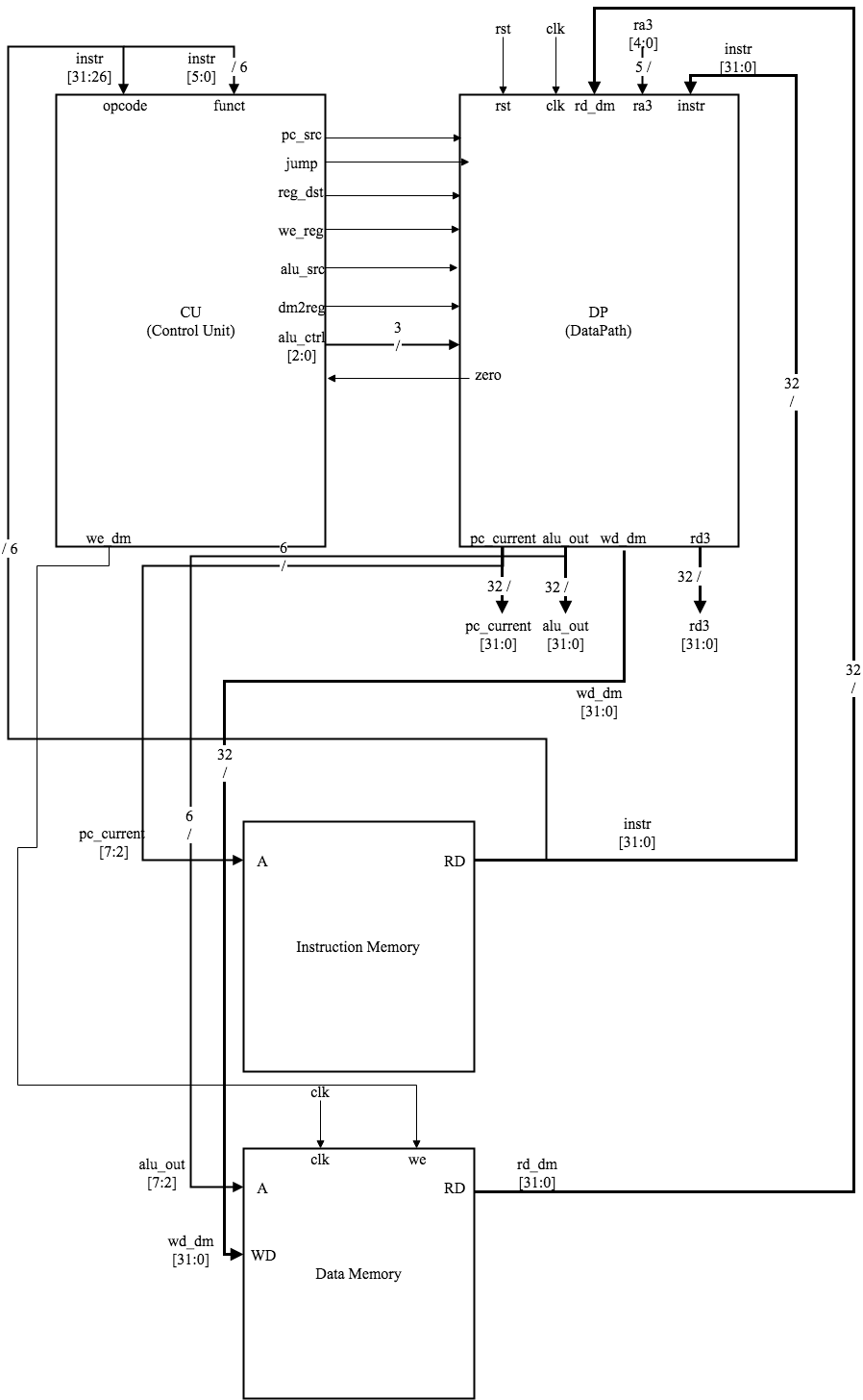


Figure 5. Complete Processor Diagram

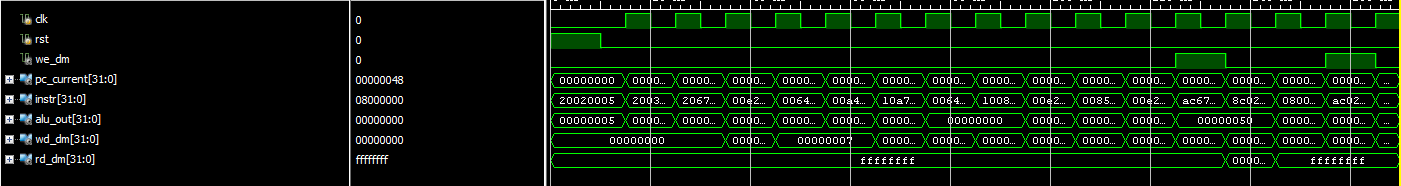


Figure 5. Test Bench Waveform

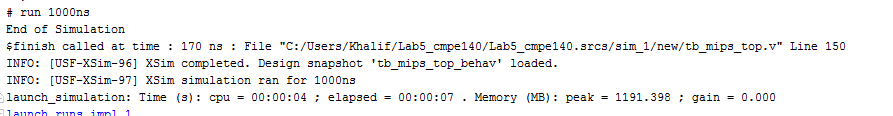


Figure 6. Test Bench Console

**Source Code:**

**mips\_top.v**

module mips\_top

(input clk, rst, output we\_dm, [31:0] pc\_current, instr, alu\_out, wd\_dm, rd\_dm);

wire [31:0] DONT\_USE;

mips mips (clk, rst, 0, instr, rd\_dm, we\_dm, pc\_current, alu\_out, wd\_dm, DONT\_USE);

imem imem (pc\_current[7:2], instr);

dmem dmem (clk, we\_dm, alu\_out[7:2], wd\_dm, rd\_dm);

endmodule

**mips.v**

module mips

(input clk, rst, [4:0] ra3, [31:0] instr, rd\_dm, output we\_dm, [31:0] pc\_current, alu\_out, wd\_dm, rd3);

wire pc\_src, jump, reg\_dst, we\_reg, alu\_src, dm2reg;

wire [2:0] alu\_ctrl;

datapath dp (clk, rst, pc\_src, jump, reg\_dst, we\_reg, alu\_src, dm2reg, alu\_ctrl, ra3, instr, rd\_dm, zero, pc\_current, alu\_out, wd\_dm, rd3);

controlunit cu (zero, instr[31:26], instr[5:0], pc\_src, jump, reg\_dst, we\_reg, alu\_src, we\_dm, dm2reg, alu\_ctrl);

endmodule

**Mem\_parts.v**

module imem

(input [5:0] a, output [31:0] y);

reg [31:0] rom [0:63];

initial begin

$readmemh ("memfile.dat", rom);

end

assign y = rom[a];

endmodule

module dmem

(input clk, we, [5:0] a, [31:0] d, output [31:0] q);

reg [31:0] ram [0:63];

integer n;

initial begin

for (n = 0; n < 64; n = n + 1) ram[n] = 32'hFFFFFFFF;

end

always @ (posedge clk)

begin

if (we) ram[a] <= d;

end

assign q = ram[a];

endmodule

**datapath.v**

module datapath

(input clk, rst, pc\_src, jump, reg\_dst, we\_reg, alu\_src, dm2reg, [2:0] alu\_ctrl, [4:0] ra3, [31:0] instr, rd\_dm, output zero, [31:0] pc\_current, alu\_out, wd\_dm, rd3);

wire [4:0] rf\_wa;

wire [31:0] pc\_plus4, pc\_pre, pc\_next, sext\_imm, ba, bta, jta, alu\_pa, alu\_pb, wd\_rf;

assign ba = {sext\_imm[29:0], 2'b00};

assign jta = {pc\_plus4[31:28], instr[25:0], 2'b00};

// --- PC Logic --- //

dreg pc\_reg (clk, rst, pc\_next, pc\_current);

adder pc\_plus\_4 (pc\_current, 4, pc\_plus4);

adder pc\_plus\_br (pc\_plus4, ba, bta);

mux2 #(32) pc\_src\_mux (pc\_src, pc\_plus4, bta, pc\_pre);

mux2 #(32) pc\_jmp\_mux (jump, pc\_pre, jta, pc\_next);

// --- RF Logic --- //

mux2 #(5) rf\_wa\_mux (reg\_dst, instr[20:16], instr[15:11], rf\_wa);

regfile rf (clk, we\_reg, instr[25:21], instr[20:16], ra3, rf\_wa, wd\_rf, alu\_pa, wd\_dm, rd3);

signext se (instr[15:0], sext\_imm);

// --- ALU Logic --- //

mux2 #(32) alu\_pb\_mux (alu\_src, wd\_dm, sext\_imm, alu\_pb);

alu alu (alu\_ctrl, alu\_pa, alu\_pb, zero, alu\_out);

// --- MEM Logic --- //

mux2 #(32) rf\_wd\_mux (dm2reg, alu\_out, rd\_dm, wd\_rf);

endmodule

**dp\_parts.v**

module mux2 #(parameter wide = 8)

(input sel, [wide-1:0] a, b, output [wide-1:0] y);

assign y = (sel) ? b : a;

endmodule

module adder

(input [31:0] a, b, output [31:0] y);

assign y = a + b;

endmodule

module signext

(input [15:0] a, output [31:0] y);

assign y = {{16{a[15]}}, a};

endmodule

module alu

(input [2:0] op, [31:0] a, b, output zero, reg [31:0] y);

assign zero = (y == 0);

always @ (op, a, b)

begin

case (op)

3'b000: y = a & b;

3'b001: y = a | b;

3'b010: y = a + b;

3'b110: y = a - b;

3'b111: y = (a < b) ? 1 : 0;

endcase

end

endmodule

module dreg

(input clk, rst, [31:0] d, output reg [31:0] q);

always @ (posedge clk, posedge rst)

begin

if (rst) q <= 0;

else q <= d;

end

endmodule

module regfile

(input clk, we, [4:0] ra1, ra2, ra3, wa, [31:0] wd, output [31:0] rd1, rd2, rd3);

reg [31:0] rf [0:31];

integer n;

initial begin

for (n = 0; n < 32; n = n + 1) rf[n] = 32'h0;

end

always @ (posedge clk)

begin

if (we) rf[wa] <= wd;

end

assign rd1 = (ra1 == 0) ? 0 : rf[ra1];

assign rd2 = (ra2 == 0) ? 0 : rf[ra2];

assign rd3 = (ra3 == 0) ? 0 : rf[ra3];

endmodule

**controlunit.v**

module controlunit

(input zero, [5:0] opcode, funct, output pc\_src, jump, reg\_dst, we\_reg, alu\_src, we\_dm, dm2reg, [2:0] alu\_ctrl);

wire [1:0] alu\_op;

assign pc\_src = branch & zero;

maindec md (opcode, branch, jump, reg\_dst, we\_reg, alu\_src, we\_dm, dm2reg, alu\_op);

auxdec ad (alu\_op, funct, alu\_ctrl);

endmodule

**cu\_parts.v**

module maindec

(input [5:0] opcode, output branch, jump, reg\_dst, we\_reg, alu\_src, we\_dm, dm2reg, [1:0] alu\_op);

reg [8:0] ctrl;

assign {branch, jump, reg\_dst, we\_reg, alu\_src, we\_dm, dm2reg, alu\_op} = ctrl;

always @ (opcode)

begin

case (opcode)

6'b00\_0000: ctrl = 9'b0\_0\_1\_1\_0\_0\_0\_10; // R-type

6'b00\_1000: ctrl = 9'b0\_0\_0\_1\_1\_0\_0\_00; // ADDI

6'b00\_0100: ctrl = 9'b1\_0\_0\_0\_0\_0\_0\_01; // BEQ

6'b00\_0010: ctrl = 9'b0\_1\_0\_0\_0\_0\_0\_00; // J

6'b10\_1011: ctrl = 9'b0\_0\_0\_0\_1\_1\_0\_00; // SW

6'b10\_0011: ctrl = 9'b0\_0\_0\_1\_1\_0\_1\_00; // LW

default: ctrl = 9'bx\_x\_x\_x\_x\_x\_x\_xx;

endcase

end

endmodule

module auxdec

(input [1:0] alu\_op, [5:0] funct, output [2:0] alu\_ctrl);

reg [2:0] ctrl;

assign {alu\_ctrl} = ctrl;

always @ (alu\_op, funct)

begin

case (alu\_op)

2'b00: ctrl = 3'b010; // add

2'b01: ctrl = 3'b110; // sub

default: case (funct)

6'b10\_0100: ctrl = 3'b000; // AND

6'b10\_0101: ctrl = 3'b001; // OR

6'b10\_0000: ctrl = 3'b010; // ADD

6'b10\_0010: ctrl = 3'b110; // SUB

6'b10\_1010: ctrl = 3'b111; // SLT

default: ctrl = 3'bxxx;

endcase

endcase

end

endmodule

**tb\_mips\_top.v**

module tb\_mips\_top;

reg clk, rst;

wire we\_dm;

wire [31:0] pc\_current, instr, alu\_out, wd\_dm, rd\_dm;

mips\_top DUT (clk, rst, we\_dm, pc\_current, instr, alu\_out, wd\_dm, rd\_dm);

task tick; begin #5 clk = 1; #5 clk = 0; end endtask

task rest; begin #5 rst = 1; #5 rst = 0; end endtask

initial begin

clk = 0;

rst = 1;

rest;

while(pc\_current != 32'h48)

begin

tick;

case(pc\_current)

32'h0: begin

if(alu\_out != 5) $display("Error with alu at instruction addr: 0");

if(instr != 32'h20020005) $display("Error with instr at instruction addr: 0");

if(rd\_dm != 32'hffffffff) $display("Error with rd\_dm at instruction addr: 0");

if(we\_dm != 1) $display("Error with we\_dm at instruction addr: 0");

end

32'h4: begin

if(alu\_out != 12) $display("Error with alu at instruction addr: 4");

if(instr != 32'h2003000c) $display("Error with instr at instruction addr: 4");

if(rd\_dm != 32'hffffffff) $display("Error with rd\_dm at instruction addr: 4");

if(we\_dm != 0) $display("Error with we\_dm at instruction addr: 4");

end

32'h8: begin

if(alu\_out != 3) $display("Error with alu at instruction addr: 8");

if(instr != 32'h2067fff7) $display("Error with instr at instruction addr: 8");

if(we\_dm != 0) $display("Error with we\_dm at instruction addr: 8");

if(rd\_dm != 32'hffffffff) $display("Error with rd\_dm at instruction addr: 8");

end

32'hc: begin

if(alu\_out != 7) $display("Error with alu at instruction addr: c");

if(instr != 32'h00e22025) $display("Error with instr at instruction addr: c");

if(we\_dm != 0) $display("Error with we\_dm at instruction addr: c");

if(rd\_dm != 32'hffffffff) $display("Error with rd\_dm at instruction addr: c");

end

32'h10: begin

if(alu\_out != 4) $display("Error with alu at instruction addr: 10");

if(instr != 32'h00642824) $display("Error with instr at instruction addr: 10");

if(we\_dm != 0) $display("Error with we\_dm at instruction addr: 10");

if(rd\_dm != 32'hffffffff) $display("Error with rd\_dm at instruction addr: 10");

end

32'h14: begin

if(alu\_out != 11) $display("Error with alu at instruction addr: 14");

if(instr != 32'h00a42820) $display("Error with instr at instruction addr: 14");

if(we\_dm != 0) $display("Error with we\_dm at instruction addr: 14");

if(rd\_dm != 32'hffffffff) $display("Error with rd\_dm at instruction addr: 14");

end

32'h18: begin

if(instr != 32'h10a7000a) $display("Error with instr at instruction addr: 18");

if(we\_dm != 0) $display("Error with we\_dm at instruction addr: 18");

if(rd\_dm != 32'hffffffff) $display("Error with rd\_dm at instruction addr: 18");

end

32'h1c: begin

if(alu\_out != 0) $display("Error with alu at instruction addr: 1c");

if(instr != 32'h0064202a) $display("Error with instr at instruction addr: 1c");

if(we\_dm != 0) $display("Error with we\_dm at instruction addr: 1c");

if(rd\_dm != 32'hffffffff) $display("Error with rd\_dm at instruction addr: 1c");

end

32'h20: begin

if(instr != 32'h10080001) $display("Error with instr at instruction addr: 20");

if(we\_dm != 0) $display("Error with we\_dm at instruction addr: 20");

if(rd\_dm != 32'hffffffff) $display("Error with rd\_dm at instruction addr: 20");

end

32'h24: begin $display("Error with beq instr at instruction addr: 24"); end

32'h28: begin

if(instr != 32'h00e2202a) $display("Error with instr at instruction addr: 28");

if(alu\_out != 1) $display("Error with alu instr at instruction addr: 28");

if(we\_dm != 0) $display("Error with we\_dm at instruction addr: 28");

if(rd\_dm != 32'hffffffff) $display("Error with rd\_dm at instruction addr: 28");

end

32'h2c: begin

if(instr != 32'h00853820) $display("Error with instr at instruction addr: 2c");

if(alu\_out != 12) $display("Error with alu instr at instruction addr: 2c");

if(we\_dm != 0) $display("Error with we\_dm at instruction addr: 2c");

if(rd\_dm != 32'hffffffff) $display("Error with rd\_dm at instruction addr: 2c");

end

32'h30: begin

if(instr != 32'h00e23822) $display("Error with instr at instruction addr: 30");

if(alu\_out != 7) $display("Error with alu instr at instruction addr: 30");

if(we\_dm != 0) $display("Error with we\_dm at instruction addr: 30");

if(rd\_dm != 32'hffffffff) $display("Error with rd\_dm at instruction addr: 30");

end

32'h34: begin

if(instr != 32'hac670044) $display("Error with instr at instruction addr: 34");

if(wd\_dm != 32'h7) $display("Error with wd\_dm at instruction addr: 34");

if(we\_dm != 1) $display("Error with we\_dm at instruction addr: 34");

if(alu\_out != 32'h50) $display("Error with alu at instruction addr: 34");

if(rd\_dm != 32'hffffffff) $display("Error with rd\_dm at instruction addr: 34");

end

32'h38: begin

if(instr != 32'h8c020050) $display("Error with instr at instruction addr: 38");

if(rd\_dm != 32'h7) $display("Error with wd\_dm at instruction addr: 38");

if(we\_dm != 0) $display("Error with we\_dm at instruction addr: 38");

if(alu\_out != 32'h50) $display("Error with alu at instruction addr: 38");

end

32'h3c: begin

if(instr != 32'h08000011) $display("Error with instr at instruction addr: 3c");

if(we\_dm != 0) $display("Error with we\_dm at instruction addr: 3c");

if(alu\_out != 32'h0) $display("Error with alu at instruction addr: 3c");

if(rd\_dm != 32'hffffffff) $display("Error with rd\_dm at instruction addr: 3c");

end

32'h40: begin

$display("Error with jump instr at instruction addr: 3c");

end

32'h44: begin

if(we\_dm != 1) $display("Error with we\_dm at instruction addr: 44");

if(alu\_out != 32'h54) $display("Error with alu\_out at instruction addr: 44");

if(wd\_dm != 32'h7) $display("Error with wd\_dm at instruction addr: 44");

if(instr != 32'hac020054) $display("Error with instr at instruction addr: 44");

if(rd\_dm != 32'hffffffff) $display("Error with rd\_dm at instruction addr: 44");

end

32'h48: begin

if(instr != 32'h08000000) $display("Error with instr at instruction addr: 48");

if(we\_dm != 0) $display("Error with we\_dm at instruction addr: 48");

if(alu\_out != 32'h0) $display("Error with alu\_out at instruction addr: 48");

if(rd\_dm != 32'hffffffff) $display("Error with rd\_dm at instruction addr: 48");

end

endcase

end

$display("End of Simulation");

$finish;

end

endmodule

**memfile.dat**

20020005

2003000C

2067FFF7

00E22025

00642824

00A42820

10A7000A

0064202A

10080001

20050000

00E2202A

00853820

00E23822

AC670044

8C020050

08000011

20020001

AC020054

08000000