Final Project

Rithvik Shetty

CMPEN 331

Section Rec 006

**Abstract**

In this project I have designed and implemented a Single cycle CPU. This CPU takes R-type, I-type, and J-type instructions and then implements them using two memory modules, a program counter, an ALU, and a control unit. This project was created in Vivado using the xc7s100fgga676-1 device.

**Introduction**

This CPU works by executing a R-type, I-type or J-type instruction every clock cycle. It has three main modules: The instruction memory (sc\_inst\_mem), the data memory (scdatamem), and the CPU (sc\_cpu). Within the CPU, the Control Unit looks at the opcode and function received from the PC. Then it uses the ALU for different operations based on the address. The Program Counter handles sending address to the instruction memory, control unit and ALU.

Based on the type of instruction, the CPU does different things. For R-type instructions, the regfile, control unit and ALU are used. For I-type instructions, the regfile, data memory, sign extend, and control unit are used. For J-type instructions, instead of the above, it replaces the lower 28 bits of the address with 26 bits from instruction that is shifted 2 bits to the left. This is then sent to the regfile.

Every clock cycle starts with fetching the instruction from PC. It then reads the instruction by reading the opcode, values, registers, etc. It then loads the values from the registers that the instruction reference. From here, it executes what the instruction specifies and writes these new values back into the registers. Afterwards, it updates the PC to fetch the next instruction. This cycle will continue to repeat until the instructions in PC are exhausted.

Some advantages of implementing the single cycle CPU are that it is easier to design and requires less complex hardware. However, they have long, inefficient clock cycles since they are single cycle.

**Schematics**

**A screenshot of a computer

Description automatically generated**

**A picture containing diagram, text, plan, technical drawing

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer program

Description automatically generated with medium confidence**

**A screen shot of a cell phone

Description automatically generated with low confidence A screenshot of a game

Description automatically generated with medium confidence**

**Code**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/03/2023 09:51:49 PM

// Design Name:

// Module Name: single\_cycle\_computer\_tb

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module single\_cycle\_computer\_tb();

reg clk;

reg clrn;

wire[31:0] pc;

wire[31:0] inst\_out;

wire[31:0] alu;

wire[31:0] dataout;

single\_cycle\_computer single\_cycle\_computer1\_tb (clk, clrn, pc , inst\_out, alu, dataout);

initial begin

clk = 1;

clrn = 1;

#20 clrn = 0;

end

always

#10 clk = ~clk;

Endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/03/2023 09:51:49 PM

// Design Name:

// Module Name: single\_cycle\_computer\_tb

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module single\_cycle\_computer\_tb();

reg clk;

reg clrn;

wire[31:0] pc;

wire[31:0] inst\_out;

wire[31:0] alu;

wire[31:0] dataout;

single\_cycle\_computer single\_cycle\_computer1\_tb (clk, clrn, pc , inst\_out, alu, dataout);

initial begin

clk = 1;

clrn = 1;

#20 clrn = 0;

end

always

#10 clk = ~clk;

Endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/03/2023 09:03:05 PM

// Design Name:

// Module Name: sc\_inst\_mem

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module sc\_inst\_mem(

input [31:0] a,

output [31:0] inst\_out

);

reg [31:0] inst\_mem [0:31];

initial begin

// instruction // (pc) label instruction

inst\_mem[5'h00] = 32'h3c010000; // (00) main: lui $1, 0

inst\_mem[5'h01] = 32'h34240050; // (04) ori $4, $1, 80

inst\_mem[5'h02] = 32'h20050004; // (08) addi $5, $0, 4

inst\_mem[5'h03] = 32'h0c000018; // (0c) call: jal sum

inst\_mem[5'h04] = 32'hac820000; // (10) sw $2, 0($4)

inst\_mem[5'h05] = 32'h8c890000; // (14) lw $9, 0($4)

inst\_mem[5'h06] = 32'h01244022; // (18) sub $8, $9, $4

inst\_mem[5'h07] = 32'h20050003; // (1c) addi $5, $0, 3

inst\_mem[5'h08] = 32'h20a5ffff; // (20) loop2: addi $5, $5, -1

inst\_mem[5'h09] = 32'h34a8ffff; // (24) ori $8, $5, 0xffff

inst\_mem[5'h0A] = 32'h39085555; // (28) xori $8, $8, 0x5555

inst\_mem[5'h0B] = 32'h2009ffff; // (2c) addi $9, $0, -1

inst\_mem[5'h0C] = 32'h312affff; // (30) andi $10,$9, 0xffff

inst\_mem[5'h0D] = 32'h01493025; // (34) or $6, $10, $9

inst\_mem[5'h0E] = 32'h01494026; // (38) xor $8, $10, $9

inst\_mem[5'h0F] = 32'h01463824; // (3c) and $7, $10, $6

inst\_mem[5'h10] = 32'h10a00001; // (40) beq $5, $0, shift

inst\_mem[5'h11] = 32'h08000008; // (44) j loop2

inst\_mem[5'h12] = 32'h2005ffff; // (48) shift: addi $5, $0, -1

inst\_mem[5'h13] = 32'h000543c0; // (4c) sll $8, $5, 15

inst\_mem[5'h14] = 32'h00084400; // (50) sll $8, $8, 16

inst\_mem[5'h15] = 32'h00084403; // (54) sra $8, $8, 16

inst\_mem[5'h16] = 32'h000843c2; // (58) srl $8, $8, 15

inst\_mem[5'h17] = 32'h08000017; // (5c) finish: j finish

inst\_mem[5'h18] = 32'h00004020; // (60) sum: add $8, $0, $0

inst\_mem[5'h19] = 32'h8c890000; // (64) loop: lw $9, 0($4)

inst\_mem[5'h1A] = 32'h20840004; // (68) addi $4, $4, 4

inst\_mem[5'h1B] = 32'h01094020; // (6c) add $8, $8, $9

inst\_mem[5'h1C] = 32'h20a5ffff; // inst\_mem[5'h00] = (70) addi $5, $5, -1

inst\_mem[5'h1D] = 32'h14a0fffb; // (74) bne $5, $0, loop

inst\_mem[5'h1E] = 32'h00081000; // (78) sll $2, $8, 0

inst\_mem[5'h1F] = 32'h03e00008; // (7c) jr $31

end

assign inst\_out = inst\_mem[a[6:2]];

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/03/2023 09:05:13 PM

// Design Name:

// Module Name: sc\_datamem

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module scdatamem(

input clk,

input [31:0] addr,

input we,

input [31:0] datain,

output [31:0] dataout

);

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

assign dataout = ram[addr[6:2]];

always@(posedge clk)

if(we)

ram[addr[6:2]] = datain;

integer i;

initial begin // initialize memory

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

ram[i] = 0;

// ram[word\_addr] = data // (byte\_addr)

ram[5'h14] = 32'h000000a3; // (50hex)

ram[5'h15] = 32'h00000027; // (54hex)

ram[5'h16] = 32'h00000079; // (58hex)

ram[5'h17] = 32'h00000115; // (5chex)

ram[5'h18] = 32'h00000258; // the sum stored by sw instruction

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/03/2023 09:07:26 PM

// Design Name:

// Module Name: sc\_cpu

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module sc\_cpu(

input clk,

input clrn,

input[31:0] inst\_out,

input[31:0] mem,

output[31:0] PCOUT,

output[31:0] r,

output[31:0] data,

output wmem

);

wire[5:0] op = inst\_out[31:26];

wire[5:0] func = inst\_out[5:0];

wire[15:0] imm = inst\_out[15:0];

wire[25:0] addr = inst\_out[25:0];

wire[4:0] rd = inst\_out[15:11];

wire[4:0] rs = inst\_out[25:21];

wire[4:0] rt = inst\_out[20:16];

wire[4:0] sa = {27'b0, inst\_out [10:6]};

wire[31:0] PCIN;

wire[31:0] q\_pcmux;

wire[31:0] pc\_adder;

wire[31:0] p4;

wire[31:0] qa;

wire z;

wire m2reg;

wire shift;

wire aluimm;

wire wreg;

wire regrt;

wire sext;

wire[1:0] pcsrc;

wire[3:0] aluc;

wire jal;

wire[31:0] out\_mux;

wire[4:0] q\_mux\_inst\_mem;

wire[31:0] out\_mux3;

wire[31:0] D\_muxout;

wire[4:0] out\_f;

wire[31:0] d;

wire[31:0] qb;

wire [31:0]e;

wire [27:0]shifter;

wire [31:0]q\_mux\_shiftline;

wire [31:0]out\_mux2;

wire [31:0]sext\_shift;

wire [31:0]sext\_adder;

PC pc1 (clk, clrn, q\_pcmux, PCOUT);

PC\_Adder pcadd(PCOUT, p4);

control\_unit controlunit1 (op, func, z, m2reg, pcsrc, wmem, aluc, shift, aluimm, wreg, regrt, sext, jal);

mux\_inst\_mem min1 (regrt , rd, rt, q\_mux\_inst\_mem);

D\_mux dmux1(out\_mux3, p4, jal, D\_muxout);

f f\_jal(jal, q\_mux\_inst\_mem, out\_f);

regfile regfile1 (D\_muxout, rs, rt, wreg, out\_f, clk, qa, data);

sign\_extender snext(imm, sext, e);

shifter shifter1(addr, shifter);

mux\_shiftline muxshift (shift, qa, {27'b0, sa}, q\_mux\_shiftline);

mux mux2 (aluimm, data, e, out\_mux2);

ALU alu1 (aluc, q\_mux\_shiftline, out\_mux2, z, r);

sextandshift sextandshift1 (e, sext\_shift);

sext\_adder sext\_add1 (p4, sext\_shift, sext\_adder);

PC\_mux pc\_mux1 (pcsrc, p4, sext\_adder, qa, {p4[31:28], shifter[27:0]}, q\_pcmux);

mux datamem\_mux (m2reg, r, mem, out\_mux3);

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/03/2023 09:15:42 PM

// Design Name:

// Module Name: PC

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module PC(

input clk,

input clrn,

input[31:0] PCIN,

output reg[31:0] PCOUT

);

always@ (negedge clk or posedge clrn) begin

if (clrn == 1)

PCOUT <= 0;

else

PCOUT <= PCIN;

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/03/2023 09:18:03 PM

// Design Name:

// Module Name: PC\_Adder

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module PC\_Adder(

input[31:0] pc\_adder,

output reg[31:0] pc4

);

always @(\*)

pc4 = pc\_adder + 32'd4;

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/03/2023 09:19:03 PM

// Design Name:

// Module Name: control\_unit

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module control\_unit(

input[5:0] op,

input[5:0] func,

input z,

output reg[1:0] pcsrc,

output reg[3:0] aluc,

output reg m2reg,

output reg wmem,

output reg shift,

output reg aluimm,

output reg wreg,

output reg regrt,

output reg sext,

output reg jal

);

always @(\*) begin

case (op)

6'b000000: begin

case (func)

6'b100010: begin

aluc <= 4'b0100;

shift <= 1'b0;

wreg <= 1'b1;

pcsrc <= 2'b00;

m2reg <= 1'b0;

wmem <= 1'b0;

aluimm <= 1'b0;

regrt <= 1'b0;

sext <= 1'b0;

jal <= 1'b0;

end

6'b100101: begin

aluc <= 4'b0101;

shift <= 1'b0;

wreg <= 1'b1;

pcsrc <= 2'b00;

m2reg <= 1'b0;

wmem <= 1'b0;

aluimm <= 1'b0;

regrt <= 1'b0;

sext <= 1'b0;

jal <= 1'b0;

end

6'b100000: begin

aluc <= 4'b0000;

shift <= 1'b0;

wreg <= 1'b1;

pcsrc <= 2'b00;

m2reg <= 1'b0;

wmem <= 1'b0;

aluimm <= 1'b0;

regrt <= 1'b0;

sext <= 1'b0;

jal <= 1'b0;

end

6'b100110: begin

aluc <= 4'b0010;

shift <= 1'b0;

wreg <= 1'b1;

pcsrc <= 2'b00;

m2reg <= 1'b0;

wmem <= 1'b0;

aluimm <= 1'b0;

regrt <= 1'b0;

sext <= 1'b0;

jal <= 1'b0;

end

6'b100100: begin

aluc <= 4'b0001;

shift <= 1'b0;

wreg <= 1'b1;

pcsrc <= 2'b00;

m2reg <= 1'b0;

wmem <= 1'b0;

aluimm <= 1'b0;

regrt <= 1'b0;

sext <= 1'b0;

jal <= 1'b0;

end

6'b001000: begin

aluc <= 4'b0000;

shift <= 1'b0;

wreg <= 1'b0;

pcsrc <= 2'b10;

m2reg <= 1'b0;

wmem <= 1'b0;

aluimm <= 1'b0;

regrt <= 1'b0;

sext <= 1'b0;

jal <= 1'b0;

end

6'b000010: begin

aluc <= 4'b0111;

shift <= 1'b1;

wreg <= 1'b1;

pcsrc <= 2'b00;

m2reg <= 1'b0;

wmem <= 1'b0;

aluimm <= 1'b0;

regrt <= 1'b0;

sext <= 1'b0;

jal <= 1'b0;

end

6'b000011: begin

aluc <= 4'b1111;

shift <= 1'b1;

wreg <= 1'b1;

pcsrc <= 2'b00;

m2reg <= 1'b0;

wmem <= 1'b0;

aluimm <= 1'b0;

regrt <= 1'b0;

sext <= 1'b0;

jal <= 1'b0;

end

6'b000000: begin

aluc <= 4'b0011;

shift <= 1'b1;

wreg <= 1'b1;

pcsrc <= 2'b00;

m2reg <= 1'b0;

wmem <= 1'b0;

aluimm <= 1'b0;

regrt <= 1'b0;

sext <= 1'b0;

jal <= 1'b0;

end

endcase

end

6'b100011: begin

aluc <= 4'b0000;

shift <= 1'b0;

wreg <= 1'b1;

pcsrc <= 2'b00;

m2reg <= 1'b1;

wmem <= 1'b0;

aluimm <= 1'b1;

regrt <= 1'b1;

sext <= 1'b1;

jal <= 1'b0;

end

6'b101011: begin

aluc <= 4'b0000;

shift <= 1'b0;

wreg <= 1'b1;

pcsrc <= 2'b00;

m2reg <= 1'b1;

wmem <= 1'b1;

aluimm <= 1'b1;

regrt <= 1'b1;

sext <= 1'b1;

jal <= 1'b0;

end

6'b001000: begin

aluc <= 4'b0000;

shift <= 1'b0;

wreg <= 1'b1;

pcsrc <= 2'b00;

m2reg <= 1'b0;

wmem <= 1'b0;

aluimm <= 1'b1;

regrt <= 1'b1;

sext <= 1'b1;

jal <= 1'b0;

end

6'b001100: begin

aluc <= 4'b0001;

shift <= 1'b0;

wreg <= 1'b1;

pcsrc <= 2'b00;

m2reg <= 1'b0;

wmem <= 1'b0;

aluimm <= 1'b1;

regrt <= 1'b1;

sext <= 1'b0;

jal <= 1'b0;

end

6'b001101: begin

aluc <= 4'b0101;

shift <= 1'b0;

wreg <= 1'b1;

pcsrc <= 2'b00;

m2reg <= 1'b0;

wmem <= 1'b0;

aluimm <= 1'b1;

regrt <= 1'b1;

sext <= 1'b0;

jal <= 1'b0;

end

6'b001110: begin

aluc <= 4'b0010;

shift <= 1'b0;

wreg <= 1'b1;

pcsrc <= 2'b00;

m2reg <= 1'b0;

wmem <= 1'b0;

aluimm <= 1'b1;

regrt <= 1'b1;

sext <= 1'b0;

jal <= 1'b0;

end

6'b001111: begin

aluc <= 4'b0110;

shift <= 1'b0;

wreg <= 1'b1;

pcsrc <= 2'b00;

m2reg <= 1'b0;

wmem <= 1'b0;

aluimm <= 1'b1;

regrt <= 1'b1;

sext <= 1'b0;

jal <= 1'b0;

end

6'b000100: begin

aluc <= 4'b0010;

shift <= 1'b0;

wreg <= 1'b0;

m2reg <= 1'b0;

wmem <= 1'b0;

aluimm <= 1'b0;

regrt <= 1'b0;

sext <= 1'b1;

jal <= 1'b0;

if (z == 1) begin

pcsrc <= 2'b01;

end

else

pcsrc <= 2'b00;

end

6'b000101: begin

aluc <= 4'b0010;

shift <= 1'b0;

wreg <= 1'b0;

m2reg <= 1'b0;

wmem <= 1'b0;

aluimm <= 1'b0;

regrt <= 1'b0;

sext <= 1'b1;

jal <= 1'b0;

if (z == 1) begin

pcsrc <= 2'b00;

end

else

pcsrc <= 2'b01;

end

6'b000010: begin

aluc <= 4'b0000;

shift <= 1'b0;

wreg <= 1'b0;

pcsrc <= 2'b11;

m2reg <= 1'b0;

wmem <= 1'b0;

aluimm <= 1'b0;

regrt <= 1'b0;

sext <= 1'b0;

jal <= 1'b0;

end

6'b000011: begin

aluc <= 4'b0000;

shift <= 1'b0;

wreg <= 1'b1;

pcsrc <= 2'b11;

m2reg <= 1'b0;

wmem <= 1'b0;

aluimm <= 1'b0;

regrt <= 1'b0;

sext <= 1'b0;

jal <= 1'b1;

end

default: begin

aluc <= 4'b0000;

shift <= 1'b0;

wreg <= 1'b0;

pcsrc <= 2'b00;

m2reg <= 1'b0;

wmem <= 1'b0;

aluimm <= 1'b0;

regrt <= 1'b0;

sext <= 1'b0;

jal <= 1'b0;

end

endcase

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/03/2023 09:26:17 PM

// Design Name:

// Module Name: mux\_inst\_mem

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module mux\_inst\_mem(

input regrt,

input[4:0] mux\_0,

input[4:0] mux\_1,

output[4:0] q\_mux\_inst\_mem

);

assign q\_mux\_inst\_mem = regrt? mux\_1 : mux\_0;

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/03/2023 09:27:08 PM

// Design Name:

// Module Name: D\_mux

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module D\_mux(

input [31:0] mux\_0,

input [31:0] mux\_1,

input jal,

output reg[31:0] D\_out

);

always @(\*) begin

if (jal == 0) begin

D\_out <= mux\_0;

end

else begin

D\_out <= mux\_1;

end

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/03/2023 09:29:09 PM

// Design Name:

// Module Name: f

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module f(

input jal,

input[4:0]fi,

output reg[4:0] fout

);

always @(\*) begin

if (jal == 1) begin

fout <= 5'b1111;

end

else begin

fout <= fi;

end

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/03/2023 09:30:48 PM

// Design Name:

// Module Name: regfile

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module regfile(

input clk,

input wreg,

input [31:0] d,

input [4:0] rna,

input [4:0] rnb,

input [4:0] wn,

output [31:0] qa,

output [31:0] qb

);

reg [31:0] register[0:31];

initial begin

register[0] = 32'h00000000;

register[1] = 32'h00000000;

register[2] = 32'h00000000;

register[3] = 32'h00000000;

register[4] = 32'h00000000;

register[5] = 32'h00000000;

register[6] = 32'h00000000;

register[7] = 32'h00000000;

register[8] = 32'h00000000;

register[9] = 32'h00000000;

register[10] = 32'h00000000;

end

assign qa = register[rna];

assign qb = register[rnb];

always @(negedge clk) begin

if (wreg == 1)

register[wn] = d;

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/03/2023 09:32:13 PM

// Design Name:

// Module Name: sign\_extender

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module sign\_extender(

input [15:0] imm,

input sext,

output reg[31:0] o

);

always @ (\*) begin

if (sext == 1) begin

o <= {{16{imm[15]}},imm[15:0]};

end

else begin

o[31:16] <= 16'b0000000000000000;

o[15:0] <= imm[15:0];

end

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/03/2023 09:34:16 PM

// Design Name:

// Module Name: shifter

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module shifter(

input [25:0] address,

output reg [27:0] shift

);

always @(\*) begin

shift <= address << 2;

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/03/2023 09:35:05 PM

// Design Name:

// Module Name: mux\_shiftline

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module mux\_shiftline(

input shift,

input[31:0] mux\_0,

input[31:0] mux\_1,

output[31:0] mux\_shiftline\_q

);

assign mux\_shiftline\_q = shift? mux\_1 : mux\_0;

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/03/2023 09:38:25 PM

// Design Name:

// Module Name: mux

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module mux(

input select,

input[31:0] mux\_0,

input[31:0] mux\_1,

output[31:0] mout

);

assign mout = select? mux\_1 : mux\_0;

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/03/2023 09:35:56 PM

// Design Name:

// Module Name: ALU

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module ALU(

input [3:0] aluc,

input [31:0] aluA,

input [31:0] aluB,

output reg z,

output reg [31:0] r

);

always @(\*) begin

if (aluA == aluB) begin

z = 1'b1;

end

if (aluA != aluB) begin

z = 1'b0;

end

case (aluc)

4'b0101: r = aluA | aluB;

4'b0100: r = aluA - aluB;

4'b0001: r = aluA & aluB;

4'b0010: r = aluA ^ aluB;

4'b0000: r = aluA + aluB;

4'b0110: r = aluB << 16;

4'b0111: r = aluA << aluB;

4'b1111: r = $signed(aluB) >>> aluA;

4'b0011: r = aluB << aluA ;

default : begin

r <= 0;

end

endcase

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/03/2023 09:39:47 PM

// Design Name:

// Module Name: sextandshift

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module sextandshift(

input [31:0] i,

output [31:0] sextshift

);

assign sextshift = i << 2;

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/03/2023 09:40:38 PM

// Design Name:

// Module Name: sext\_adder

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module sext\_adder(

input[31:0] p4,

input[31:0] sextshift,

output reg[31:0] sextadder

);

always @(\*) begin

sextadder <= p4 + sextshift;

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/03/2023 09:41:41 PM

// Design Name:

// Module Name: PC\_mux

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module PC\_mux(

input [1:0] pcsrc,

input [31:0] pc\_mux0,

input [31:0] pc\_mux1,

input [31:0] pc\_mux2,

input [31:0] pc\_mux3,

output reg [31:0] pcmux\_q

);

always @(\*) begin

if (pcsrc == 2'b00)

pcmux\_q = pc\_mux0;

else if (pcsrc == 2'b01)

pcmux\_q = pc\_mux1;

else if (pcsrc == 2'b10)

pcmux\_q = pc\_mux2;

else if (pcsrc == 2'b11)

pcmux\_q = pc\_mux3;

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/03/2023 09:38:25 PM

// Design Name:

// Module Name: mux

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module mux(

input select,

input[31:0] mux\_0,

input[31:0] mux\_1,

output[31:0] mout

);

assign mout = select? mux\_1 : mux\_0;

endmodule