Single cycle MIPS processor

Team 3 :

1. Rana Wagdy
2. Omnia Mohammed
3. Youmna Mohammed
4. Kerolos Youssef

CODE:

Top module:

module MIPS(clk,reset,register\_data1,register\_data2,ALU\_out,result,A1,A2,A3);

input clk,reset;

output [31:0] register\_data1,register\_data2,ALU\_out,result;

output [4:0] A1,A2,A3;

Signals:

wire [31:0] pc\_branch,pc\_jump,pc\_plus4,pc,pc\_before\_clk,final\_pc;//PC wires

wire [31:0]int\_four;//adder

wire [31:0] int31;

assign int31=32'd31;

assign int\_four=32'b100;

wire [31:0] instruction;//instruction memory

wire [4:0]write\_to\_register;//register file

wire[31:0] regfile\_WD,register\_RD1,register\_RD2;//register\_file

wire[31:0]extended\_imm,extended\_imm\_by\_four; //extension

wire[1:0]mem\_to\_reg,reg\_dest,alu\_op;//control unit

wire jump,mem\_write\_en,reg\_write\_en,branch,ori\_ctrl,alu\_src,zero,pc\_source,sourceB;

wire[2:0] alu\_control;//alu

wire[31:0] alu\_output,mem\_data\_out,data\_result;//result

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* connection\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Mux\_2 PCSource(pc\_plus4, pc\_branch, pc, pc\_source);

assign pc\_jump[31:28]=pc\_plus4[31:28];

Mux\_2 PCJump(pc, pc\_jump, pc\_before\_clk, jump);

PC final\_PC (final\_pc,pc\_before\_clk,clk,reset);

adder PCPlus4 (final\_pc,int\_four,pc\_plus4);

instr\_mem instruction\_mem(final\_pc,instruction);

Shift\_Left\_Two\_Jump shifter1 (instruction[25:0], pc\_jump[27:0]);

extension extend\_address(instruction[15:0], extended\_imm,ori\_ctrl);

Shift\_Left\_Two shifter2(extended\_imm\_by\_four,extended\_imm);

adder PCBranch(extended\_imm\_by\_four,pc\_plus4,pc\_branch);

controlunit main\_decoder(instruction[31:26],instruction[5:0], mem\_write\_en,mem\_to\_reg,branch,alu\_src,reg\_dest,reg\_write\_en,alu\_op,jump,ori\_ctrl);

aludec decoder (instruction[31:26],instruction[5:0],alu\_op,alu\_control);

Mux\_2 Source\_B(register\_RD2, extended\_imm, sourceB, alu\_src);

alu ALU(register\_RD1, sourceB,alu\_control,alu\_output,zero);

assign pc\_source=branch && zero;

Data\_Memory memory (mem\_data\_out,clk,mem\_write\_en,alu\_output,register\_RD2);

Mux\_3 resulted\_data(alu\_output,mem\_data\_out,pc\_plus4, data\_result, mem\_to\_reg);

register\_file RegisterFile (instruction[25:21], instruction[20:16], write\_to\_register, register\_RD1, register\_RD2,data\_result,reg\_write\_en, clk);

Mux\_3 register\_dst(instruction[20:16], instruction[15:11],int31, write\_to\_register, reg\_dest);

Endmodule

Program counter:

module PC(out,in,clk,reset);

output [31:0] out;

input [31:0] in ;

input clk,reset;

reg [31:0] out;

initial begin

out =32'b0;

end

always@(posedge clk or posedge reset)

begin

if (reset == 1'b1)

out<=32'b0;

else

out <= in;

end

endmodule

Register file:

module register\_file (A1, A2, A3, RD1, RD2, WD3, WE3, clk);

input [4:0] A1, A2, A3;

input [31:0] WD3;

input clk, WE3;

output [31:0] RD1, RD2;

reg [31:0] regfile [31:0];

initial

begin

regfile[0]=32'b0;

end

assign RD1 = regfile[A1];

assign RD2 = regfile[A2];

always @(posedge clk )

begin

if (WE3 == 1'b1) regfile[A3] = WD3;

end

endmodule

ALU:

module alu

(

input [31:0] srcA , srcB,

input [2:0] alucontrol,

output reg [31:0] aluresult,

output reg zero

);

always @(\*) begin

case (alucontrol)

3'b010: aluresult <= srcA + srcB; // add

3'b110: aluresult <= srcA + (~srcB) + 1; // sub

3'b000: aluresult <= srcA & srcB; // and

3'b001: aluresult <= srcA | srcB; // or

3'b111: aluresult <=((srcA[31] != srcB[31]) && srcA<srcB ) ? 1 : 0; // slt

default: aluresult <=0;

endcase

end

always @(aluresult) begin

if(aluresult==0)

begin

zero<=1;

end

else begin

zero<=0;

end

end

endmodule

ALU decoder:

module aludec(

input [5:0]opcode,

input [5:0]funct,

input [1:0]aluop,

output reg [2:0] alucontrol

);

always @(\*)

begin

case(aluop)

2'b00:alucontrol<=3'b010;//ADD\_LW\_SW\_ADDI

2'b01:alucontrol<=3'b110;//SUB\_BEQ

2'b10:case(funct) //R\_Type

6'b100000:alucontrol<=3'b010;//ADD

6'b100010:alucontrol<=3'b110;//SUB

6'b100100:alucontrol<=3'b000;//AND

6'b100101:alucontrol<=3'b001;//OR

6'b101010:alucontrol<=3'b111;//SLT

default:alucontrol<=3'bxxx;

endcase

2'b11:case(opcode)

6'b001000:alucontrol<=3'b010;//ADDI

6'b001101:alucontrol<=3'b001;//ORI

default:alucontrol<=3'bxxx;//NOT Used

endcase

endcase

end

endmodule

Control unit:

module controlunit

(

input [5:0] opcode,

input [5:0] funct, //we describe main decoder not ALU decoder so this variable is not used

output reg mem\_write,

output reg [1:0]mem\_toreg,

output reg branch,

output reg alu\_src,

output reg[1:0] reg\_dst,

output reg reg\_write,

output reg [1:0] alu\_op,

output reg jump,

output reg logic

);

always@(\*)

begin

if(opcode==6'b000011)

begin

mem\_toreg[1]=1'b1;

reg\_dst[1]=1'b1;

end

else begin

mem\_toreg[1]=1'b0;

reg\_dst[1]=1'b0;

end

if (opcode==6'b001101)

logic=1'b1;

else logic=1'b0;

case(opcode)

6'b000000: //R\_Type

begin

reg\_dst[0]=1;

alu\_src=0;

mem\_toreg[0]=0;

reg\_write=1;

mem\_write=0;

branch=0;

alu\_op=2'b10;

jump=0;

end

6'b100011: //lw

begin

reg\_dst[0]=0;

alu\_src=1;

mem\_toreg[0]=1;

reg\_write=1;

mem\_write=0;

branch=0;

alu\_op=2'b00;

jump=0;

end

6'b101011: //sw

begin

reg\_dst[0]=1'bx;

alu\_src=1;

mem\_toreg[0]=1'bx;

reg\_write=0;

mem\_write=1;

branch=0;

alu\_op=2'b00;

jump=0;

end

6'b000010: //jump

begin

reg\_dst[0]=1'bx;

alu\_src=1'bx;

mem\_toreg[0]=1'bx;

reg\_write=0;

mem\_write=0;

branch=1'bx;

alu\_op=2'bxx;

jump=1;

end

6'b000011: //jal

begin

reg\_dst[0]=1'b0;

alu\_src=1'bx;

mem\_toreg[0]=1'b0;

reg\_write=1;

mem\_write=0;

branch=1'bx;

alu\_op=2'b00;

jump=1;

end

6'b000100: //beq

begin

reg\_dst[0]=1'bx;

alu\_src=0;

mem\_toreg[0]=1'bx;

reg\_write=0;

mem\_write=0;

branch=1;

alu\_op=2'b01;

jump=0;

end

6'b001000: //addi

begin

reg\_dst[0]=0;

alu\_src=1;

mem\_toreg[0]=0;

reg\_write=1;

mem\_write=0;

branch=0;

alu\_op=2'b11;

jump=0;

end

6'b001101: //ori

begin

reg\_dst[0]=0;

alu\_src=1;

mem\_toreg[0]=0;

reg\_write=1;

mem\_write=0;

branch=0;

alu\_op=2'b11;

jump=0;

end

default://look @ function

begin

reg\_dst[0]=1;

alu\_src=0;

mem\_toreg[0]=0;

reg\_write=1;

mem\_write=0;

branch=0;

alu\_op=2'b10;

end

endcase

end

endmodule

Data memory:

module Data\_Memory(output [31:0] Read\_Data,input clk,WE,output reg[31:0] ALU\_Result,output reg[31:0]Write\_Data);

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

//wire [31:2] memory\_add;

assign Read\_Data=Memory[ALU\_Result[31:2]];

always @(posedge clk)

begin

if(WE)

Memory[ALU\_Result[31:2]]<=Write\_Data;

end

endmodule

Instruction memory:

module instr\_mem // a synthesisable rom implementation

(

input [31:0] pc,

output wire [31:0] instruction

);

wire [4: 0] rom\_addr = pc[5: 1];

reg [31:0] rom[21:0];

initial

begin

rom[0] <= 32'b00100000000010000000000000100000; //addi $t0, $zero, 32

rom[1] <= 32'b00100000000010010000000000110111; //addi $t1, $zero, 55

rom[2] <= 32'b00000001000010011000000000100100; //and $s0, $t0, $t1

rom[3] <= 32'b00000001000010011000000000100101; //or $s0, $t0, $t1

rom[4] <= 32'b10101100000100000000000000000100; //sw $s0, 4($zero)

rom[5] <= 32'b00000001000010011001000000100010; //sub $s2, $t0, $t1

rom[6] <= 32'b10101100000010000000000000001000; //sw $t0, 8($zero)

rom[7] <= 32'b00000001000010011001000000100010; //sub $s2, $t0, $t1

rom[8] <= 32'b00010010001100100000000000001001; //beq $s1, $s2, error0

rom[9] <= 32'b10001100000100010000000000000100; //lw $s1, 4($zero)

rom[10] <= 32'b00110010001100100000000001001000; //andi $s2, $s1, 48

rom[11] <= 32'b00010010001100100000000000001001; //beq $s1, $s2, error1

rom[12] <= 32'b00000010010100011010000000101010; //slt $s4, $s2, $s1 (Last)

rom[13] <= 32'b00000010001000001001000000100000; //add $s2, $s1, $0

rom[14] <= 32'b00001100000000000000000000001110; //jal last;

rom[15] <= 32'b00100000000010000000000000000000; //addi $t0, $0, 0(error0)

rom[16] <= 32'b00100000000010010000000000000000; //addi $t1, $0, 0

rom[17] <= 32'b00100000000010000000000000000001; //addi $t0, $0, 1(error1);

rom[18] <= 32'b00100000000010010000000000000001; //addi $t1, $0, 1 ;

rom[19] <= 32'b00110100000010000000000000000000; //ori $t0, $0, 0(error0)

rom[20] <= 32'b00001000000000000000000000011111; //j EXIT;

end

assign instruction = (pc[31:0] != 0 )? pc[31:0]:32'b0;

endmodule

module Shift\_Left\_Two(output [31:0] Output,input[31:0] Input);

assign Output={Input[29:0] ,2'b00};

endmodule

module Shift\_Left\_Two\_Jump (shift\_in, shift\_out);

input [25:0] shift\_in;

output [27:0] shift\_out;

assign shift\_out[27:0]={shift\_in[25:0],2'b00};

endmodule

//######################################################################

module Mux\_3 (input0, input1,input2, mux\_out, control);

input [31:0] input0, input1,input2;

output reg [31:0] mux\_out;

input[1:0]control;

always@(\*) begin

if (control[1])

mux\_out=input2;

else if (control[1]!=1)

mux\_out=control[0]?input1:input0;

end

endmodule

//############################################################################

module Mux\_2 (input0, input1, mux\_out, control);

input [31:0] input0, input1;

output [31:0] mux\_out;

input control;

assign mux\_out=control?input1:input0;

endmodule

//############################################################################

module adder (input1,input2,out);

input [31:0] input1, input2;

output [31:0] out;

assign out=input1+input2;

endmodule

//################################################

module extension (imm, sign\_imm,logic);

input [15:0] imm;

input logic;

output reg[31:0] sign\_imm;

always@(\*)begin

sign\_imm[15:0]=imm[15:0];

if(logic) sign\_imm[31:16]=16'b0;

else

sign\_imm[31:16]=imm[15]?16'b1111111111111111:16'b0;

end

endmodule