//backup

module MainModule( input clk, input reset, input [5:0] opCode , input [5:0] funcField ,

input overflowflag, ///comm

output [28:0] controlWord

, output overFlowDetect //commm

);

wire [4:0] addressNext;

wire [2:0] addressCtrl;

wire [4:0] seq;

wire [4:0] dt1;

wire [4:0] dt2;

wire [4:0] dt3;

wire [4:0] wbAddr;

//wire [28:0] controlWord;

DT1 dispTbl1(opCode , funcField, dt1);

DT2 dispTbl2(opCode , funcField, dt2);

DT3 dispTbl3(opCode , dt3);

WriteBack wb(opCode , funcField, wbAddr);

adder add0(addressNext,seq);

NxtAddrMux mux1(clk, reset, addressCtrl , wbAddr, dt3, dt2, dt1,

seq, 5'd0 , addressNext );

controlUnit c1( addressNext,

clk,

overflowflag ,//comm

addressCtrl,

controlWord,

overFlowDetect //comm

);

endmodule

module DT1(input [5:0] opCode , input [5:0] funcField, output reg[4:0] addressNext

);

always@(\*)

begin

case(opCode)

6'd0: begin

if(funcField == 6'd16) //mfhi

addressNext=5'd2;

else if(funcField == 6'd18) //mflo

addressNext=5'd3;

else if(funcField == 6'd17) //mthi

addressNext=5'd4;

else if(funcField == 6'd19) //mtlo

addressNext=5'd5;

else if(funcField == 6'd8)//jr

addressNext=5'd10;

else if(funcField == 6'd9) //jalr

addressNext=5'd11;

else if(funcField == 6'd32) //add

addressNext=5'd12;

else if(funcField == 6'd0) //sll

addressNext=5'd13;

else if(funcField == 6'd4) //sllv

addressNext=5'd14;

else if(funcField == 6'd26) //div

addressNext=5'd15;

else if(funcField == 6'd24) //mult

addressNext=5'd16;

end

6'd2: addressNext=5'd8; //j

6'd3: addressNext=5'd9; //jal

6'd4: addressNext=5'd7; //beq

6'd15:addressNext=5'd6; //lui

6'd28:addressNext=5'd16; // madd msub

6'd13:addressNext=5'd18; //ori

6'd8:addressNext =5'd17; //addi

6'd35:addressNext=5'd17; //lw

6'd43:addressNext=5'd17; //sw

default: begin

addressNext=5'd27; //invalid

end

endcase

end

endmodule

module DT2(input [5:0] opCode , input [5:0] funField, output reg[4:0] addressNext

);

always@(\*)

begin

case(opCode) //WB stages of resp instructions

6'd0: begin

if(funField == 6'd24) //mult

addressNext=5'd20;

end

6'd28: begin

if(funField == 6'd0) //madd

addressNext=5'd21;

else if(funField == 6'd4)//msub

addressNext=5'd22;

end

endcase

end

endmodule

module DT3(input [5:0] opCode,output reg[4:0] addressNext

);

always@(\*)

begin

case(opCode) //MEM / WB stages of resp instructions

6'd35: addressNext=5'd23; //lw

6'd43: addressNext=5'd24; //sw

6'd8: addressNext=5'd25; //addi (WB)

endcase

end

endmodule

module WriteBack(input [5:0] opCode , input [5:0] funcField, output reg[4:0] addressNext

);

always@(\*)

begin //WB stages of resp instructions

case(opCode)

6'd0: begin

if(funcField == 6'd32) //add

addressNext=5'd19;

else if(funcField == 6'd0) //sll

addressNext=5'd19;

else if(funcField == 6'd4) //sllv

addressNext=5'd19;

else if(funcField == 6'd26) //div

addressNext=5'd20;

end

6'd35: addressNext=5'd26; //lw

6'd13: addressNext=5'd25; //ori

endcase

end

endmodule

module adder(input [4:0] address, output reg [4:0] sequenceAddr

);

always@(address)

begin

sequenceAddr=address+1;

end

endmodule

module NxtAddrMux(input clk, input reset,input [2:0] addressCtrl , input [4:0] wbAddr, input [4:0] dt3, input [4:0] dt2, input [4:0] dt1,

input [4:0] seq, input [4:0] fetch , output reg [4:0] nextAddress

);

always@(posedge clk)

begin

if(reset)

nextAddress= fetch;

else

case(addressCtrl)

3'd0:nextAddress= fetch;

3'd1:nextAddress= seq;

3'd2:nextAddress= dt1;

3'd3:nextAddress= dt2;

3'd4:nextAddress= dt3;

3'd5:nextAddress= wbAddr;

endcase

end

endmodule

module controlUnit(input [4:0] addressNext,

input clk,

input overflowflag ,//commm

output reg[2:0] addressControl,

output reg [28:0] controlWord,

output reg overFlowDetect //comm

);

reg [31:0] controlSignalsArray [27:0];

reg [31:0] controlComp;

always@(negedge clk)

begin

//pcWr pcWrCond pcSrc2b IorD MemRd MemWr IRWr srcA2b srcB3b aluCtrl3b hiSel2b loSel2b hiWR loWr malu memtoREg regdst regWr addrCtrl3b

controlSignalsArray[0] =32'b1\_0\_00\_0\_1\_0\_1\_00\_000\_000\_00\_00\_0\_0\_0\_000\_00\_0\_001; // IF

controlSignalsArray[1] =32'b0\_0\_00\_0\_0\_0\_0\_00\_010\_000\_00\_00\_0\_0\_0\_000\_00\_0\_010; // ID

controlSignalsArray[2] =32'b0\_0\_00\_0\_0\_0\_0\_00\_000\_000\_00\_00\_0\_0\_0\_100\_00\_1\_000; // EX mfhi

controlSignalsArray[3] =32'b0\_0\_00\_0\_0\_0\_0\_00\_000\_000\_00\_00\_0\_0\_0\_101\_00\_1\_000; // EX mflo

controlSignalsArray[4] =32'b0\_0\_00\_0\_0\_0\_0\_00\_000\_000\_01\_00\_1\_0\_0\_000\_00\_0\_000; // EX mthi

controlSignalsArray[5] =32'b0\_0\_00\_0\_0\_0\_0\_00\_000\_000\_00\_01\_0\_1\_0\_000\_00\_0\_000; // EX mtlo

controlSignalsArray[6] =32'b0\_0\_00\_0\_0\_0\_0\_00\_000\_000\_00\_00\_0\_0\_0\_011\_00\_1\_000; // EX lui

controlSignalsArray[7] =32'b0\_1\_01\_0\_0\_0\_0\_01\_001\_001\_00\_00\_0\_0\_0\_000\_00\_0\_000; // EX beq

controlSignalsArray[8] =32'b1\_0\_10\_0\_0\_0\_0\_00\_000\_000\_00\_00\_0\_0\_0\_000\_00\_0\_000; // EX j

controlSignalsArray[9] =32'b1\_0\_10\_0\_0\_0\_0\_00\_000\_000\_00\_00\_0\_0\_0\_010\_10\_1\_000; // EX jal

controlSignalsArray[10] =32'b1\_0\_11\_0\_0\_0\_0\_00\_000\_000\_00\_00\_0\_0\_0\_000\_00\_0\_000; // EX jr

controlSignalsArray[11] =32'b1\_0\_11\_0\_0\_0\_0\_00\_000\_000\_00\_00\_0\_0\_0\_010\_00\_1\_000; // EX jalr

controlSignalsArray[12] =32'b0\_0\_00\_0\_0\_0\_0\_01\_001\_000\_00\_00\_0\_0\_0\_000\_00\_0\_101; // EX add

controlSignalsArray[13] =32'b0\_0\_00\_0\_0\_0\_0\_10\_001\_100\_00\_00\_0\_0\_0\_000\_00\_0\_101; // EX sll

controlSignalsArray[14] =32'b0\_0\_00\_0\_0\_0\_0\_01\_001\_100\_00\_00\_0\_0\_0\_000\_00\_0\_101; // EX sllv

controlSignalsArray[15] =32'b0\_0\_00\_0\_0\_0\_0\_01\_001\_100\_00\_00\_0\_0\_0\_000\_00\_0\_101; // EX div

controlSignalsArray[16] =32'b0\_0\_00\_0\_0\_0\_0\_01\_001\_011\_00\_00\_0\_0\_0\_000\_00\_0\_011; // EX mult, madd , msub

controlSignalsArray[17] =32'b0\_0\_00\_0\_0\_0\_0\_01\_011\_000\_00\_00\_0\_0\_0\_000\_00\_0\_100; // EX addi, lw, sw

controlSignalsArray[18] =32'b0\_0\_00\_0\_0\_0\_0\_01\_010\_101\_00\_00\_0\_0\_0\_000\_00\_0\_101; // EX ori

controlSignalsArray[19] =32'b0\_0\_00\_0\_0\_0\_0\_00\_000\_000\_00\_00\_0\_0\_0\_000\_00\_1\_000; // WB add, sll , sllv

controlSignalsArray[20] =32'b0\_0\_00\_0\_0\_0\_0\_00\_000\_000\_00\_00\_1\_1\_0\_000\_00\_0\_000; // WB div , mult

controlSignalsArray[21] =32'b0\_0\_00\_1\_1\_0\_0\_00\_000\_000\_00\_00\_0\_0\_0\_000\_00\_0\_000; // WB madd

controlSignalsArray[22] =32'b0\_0\_00\_1\_1\_0\_0\_00\_000\_000\_00\_00\_0\_0\_0\_000\_00\_0\_000; // WB msub

controlSignalsArray[23] =32'b0\_0\_00\_1\_0\_1\_0\_00\_000\_000\_00\_00\_0\_0\_0\_000\_00\_0\_101; // MEM lw

controlSignalsArray[24] =32'b0\_0\_00\_0\_0\_0\_0\_00\_000\_000\_00\_00\_0\_0\_0\_000\_01\_1\_000; // MEM sw

controlSignalsArray[25] =32'b0\_0\_00\_0\_0\_0\_0\_00\_000\_000\_00\_00\_0\_0\_0\_001\_01\_1\_000; // WB addi ori

controlSignalsArray[26] =32'b0\_0\_00\_0\_0\_0\_0\_00\_000\_000\_00\_00\_0\_0\_0\_001\_01\_1\_000; // WB lw --> CHECK

controlSignalsArray[27] =32'b1\_1\_11\_1\_1\_1\_1\_11\_111\_111\_11\_11\_1\_1\_1\_111\_11\_1\_111; // Invalid Instruction exception

controlComp=controlSignalsArray[addressNext];

controlWord=controlComp[31:3];

addressControl=controlComp[2:0];

overFlowDetect=overflowflag; //comm

end

endmodule