|  |
| --- |
| B10502007 郭曜銘 Computer Organization Homework IV |
| module Pipe\_CPU\_1(  clk\_i,  rst\_n  );    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Parameter  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  I/O ports  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  input clk\_i;  input rst\_n;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Internal signal  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  wire [31:0]PC\_4;  wire [31:0]branch\_in;  wire PCSrc;  wire [31:0]pc\_in;  wire [31:0]pc\_out;  wire [31:0]instr;  wire [31:0]IF\_ID\_PC4;  wire [31:0]IF\_ID\_instr;  wire [31:0]rdata1;  wire [31:0]rdata2;  //wire [4:0]writereg;  //wire [31:0]writedata;  //  wire regwrite;  //  wire RegWrite\_o;  wire [2:0]ALU\_op\_o;  wire ALUSrc\_o;  wire RegDst\_o;  wire Branch\_o;  //wire Flag\_ORI;  wire MemToReg\_o;  wire MemRead\_o;  wire MemWrite\_o;  //  wire [31:0]signex;  wire ID\_EX\_RegWrite;  wire ID\_EX\_MemToReg;  wire ID\_EX\_Branch;  wire ID\_EX\_MemRead;  wire ID\_EX\_MemWrite;  wire ID\_EX\_RegDst;  wire [2:0]ID\_EX\_ALU\_op;  wire ID\_EX\_ALUSrc;  wire [31:0]ID\_EX\_PC4;  wire [31:0]add2\_o;  wire [31:0]AluIn1;  wire [31:0]ID\_EX\_rd2;  wire [31:0]ID\_EX\_signex;  wire [4:0]ID\_EX\_RD;  wire [4:0]ID\_EX\_RT;  wire [31:0]shift2;  wire [31:0]AluIn2;  wire [31:0]Alu\_result;  wire [3:0]Aluctrl;  wire zero;  wire cout;  wire overflow;  wire [4:0]ID\_EX\_Write\_R;  wire EX\_M\_RegWrite;  wire EX\_M\_MemToReg;  wire EX\_M\_Branch;  wire EX\_M\_MemRead;  wire EX\_M\_MemWrite;  wire M\_zero;  wire [31:0]M\_Alu\_result;  wire [31:0]WriteData; //W  wire [4:0]EX\_M\_Write\_R; //R  wire [31:0]Memdata\_o;  wire [31:0]WB\_Memdata\_o;  wire [31:0]WB\_Alu\_result;  wire MemToReg;  wire [4:0]final\_RD;  wire [31:0]final\_data;  /\*\*\*\* IF stage \*\*\*\*/  MUX\_2to1 #(.size(32)) Mux\_PC(  .data0\_i(PC\_4),  .data1\_i(branch\_in),  .select\_i(PCSrc),  .data\_o(pc\_in)  );  ProgramCounter PC(  .clk\_i(clk\_i),  .rst\_n (rst\_n),  .pc\_in\_i(pc\_in) ,  .pc\_out\_o(pc\_out)  );  Adder Adder1( // keep doing PC + 4  .src1\_i(pc\_out),  .src2\_i(32'd4),  .sum\_o(PC\_4)  );  Instr\_Memory IM( // Get Instruction from IM by PC address  .pc\_addr\_i(pc\_out),  .instr\_o(instr)  );  /\*\*\*\* IF stage \*\*\*\*/  /\*\*\*\* IF stage \*\*\*\*/  Pipe\_Reg #(.size(32)) IF\_PC(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(PC\_4),  .data\_o(IF\_ID\_PC4)  );  Pipe\_Reg #(.size(32)) IF\_instr(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(instr),  .data\_o(IF\_ID\_instr)  );  /\*\*\*\* IF stage \*\*\*\*/  /\*\*\*\* ID stage \*\*\*\*/  Reg\_File RF( // Read or write register data  .clk\_i(clk\_i),  .rst\_n(rst\_n) ,  .RSaddr\_i(IF\_ID\_instr[25:21]) ,  .RTaddr\_i(IF\_ID\_instr[20:16]) ,  .RDaddr\_i(final\_RD) ,  .RDdata\_i(final\_data) ,  .RegWrite\_i (regwrite),  .RSdata\_o(rdata1) ,  .RTdata\_o(rdata2)  );  Decoder Decoder( // Decode instruct to ALU / MUX / RF / BRANCH  .instr\_op\_i(IF\_ID\_instr[31:26]),  .RegWrite\_o(RegWrite\_o),  .ALU\_op\_o(ALU\_op\_o),  .ALUSrc\_o(ALUSrc\_o),  .RegDst\_o(RegDst\_o),  .Branch\_o(Branch\_o),  .MemToReg\_o(MemToReg\_o),  .Jump\_o(), //多餘  .MemRead\_o(MemRead\_o),  .MemWrite\_o(MemWrite\_o)  );  Sign\_Extend SE( // Sign extend from 16 bits to 32 bits  //.ALUop(),  .data\_i(IF\_ID\_instr[15:0]),  .data\_o(signex)  );  /\*\*\*\* ID stage \*\*\*\*/  /\*\*\*\* ID stage \*\*\*\*/  Pipe\_Reg #(.size(1))ID\_WB(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(RegWrite\_o),  .data\_o(ID\_EX\_RegWrite)  );  Pipe\_Reg #(.size(1))ID\_WB2(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(MemToReg\_o),  .data\_o(ID\_EX\_MemToReg)  );  Pipe\_Reg #(.size(1))ID\_M1(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(Branch\_o),  .data\_o(ID\_EX\_Branch)  );  Pipe\_Reg #(.size(1))ID\_M2(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(MemRead\_o),  .data\_o(ID\_EX\_MemRead)  );  Pipe\_Reg #(.size(1))ID\_M3(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(MemWrite\_o),  .data\_o(ID\_EX\_MemWrite)  );  Pipe\_Reg #(.size(1))ID\_EX1(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(ALUSrc\_o),  .data\_o(ID\_EX\_ALUSrc)  );  Pipe\_Reg #(.size(3))ID\_EX2(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(ALU\_op\_o),  .data\_o(ID\_EX\_ALU\_op)  );  Pipe\_Reg #(.size(1))ID\_EX3(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(RegDst\_o),  .data\_o(ID\_EX\_RegDst)  );  Pipe\_Reg #(.size(32))ID\_PC(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(IF\_ID\_PC4),  .data\_o(ID\_EX\_PC4)  );  Pipe\_Reg #(.size(32))Rd1(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(rdata1),  .data\_o(AluIn1)  );  Pipe\_Reg #(.size(32))Rd2(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(rdata2),  .data\_o(ID\_EX\_rd2)  );  Pipe\_Reg #(.size(32))ID\_SE(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(signex),  .data\_o(ID\_EX\_signex)  );  Pipe\_Reg #(.size(5))ID\_RD(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(IF\_ID\_instr[15:11]),  .data\_o(ID\_EX\_RD)  );  Pipe\_Reg #(.size(5))ID\_RT(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(IF\_ID\_instr[20:16]),  .data\_o(ID\_EX\_RT)  );    /\*\*\*\* ID stage \*\*\*\*/  /\*\*\*\* EX stage \*\*\*\*/  Shift\_Left\_Two\_32 Shifter(  .data\_i(ID\_EX\_signex),  .data\_o(shift2)  );  Adder Adder2( // PC = PC + 4 + branch\_addr \* 4  .src1\_i(shift2),  .src2\_i(ID\_EX\_PC4),  .sum\_o(add2\_o)  );  MUX\_2to1 #(.size(32)) Mux\_ALU( // Select source of ALU\_IN\_2  .data0\_i(ID\_EX\_rd2),  .data1\_i(ID\_EX\_signex),  .select\_i(ID\_EX\_ALUSrc),  .data\_o(AluIn2)  );  alu ALU(  .clk(clk\_i),  .rst(rst\_n),  .src1(AluIn1),  .src2(AluIn2),  .ALU\_control(Aluctrl),  .result(Alu\_result),  .zero(zero),  .cout(),  .overflow()  );  ALU\_Ctrl AC( // Decode ALUCtrl\_o  .funct\_i(ID\_EX\_signex[5:0]),  .ALUOp\_i(ID\_EX\_ALU\_op),  .ALUCtrl\_o(Aluctrl)  );  MUX\_2to1 #(.size(5)) Mux\_Dst(  .data0\_i(ID\_EX\_RT),  .data1\_i(ID\_EX\_RD),  .select\_i(ID\_EX\_RegDst),  .data\_o(ID\_EX\_Write\_R)  );  /\*\*\*\* EX stage \*\*\*\*/  /\*\*\*\* EX stage \*\*\*\*/  Pipe\_Reg #(.size(1))EX\_WB(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(ID\_EX\_RegWrite),  .data\_o(EX\_M\_RegWrite)  );  Pipe\_Reg #(.size(1))EX\_WB2(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(ID\_EX\_MemToReg),  .data\_o(EX\_M\_MemToReg)  );  Pipe\_Reg #(.size(1))EX\_M1(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(ID\_EX\_Branch),  .data\_o(EX\_M\_Branch)  );  Pipe\_Reg #(.size(1))EX\_M2(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(ID\_EX\_MemRead),  .data\_o(EX\_M\_MemRead)  );  Pipe\_Reg #(.size(1))EX\_M3(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(ID\_EX\_MemWrite),  .data\_o(EX\_M\_MemWrite)  );  Pipe\_Reg #(.size(32))EX\_PC(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(add2\_o),  .data\_o(branch\_in)  );  Pipe\_Reg #(.size(1))EX\_Zero(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(zero),  .data\_o(M\_zero)  );  Pipe\_Reg #(.size(32))EX\_Alu(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(Alu\_result),  .data\_o(M\_Alu\_result)  );  Pipe\_Reg #(.size(32))EX\_Rd2(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(ID\_EX\_rd2),  .data\_o(WriteData)  );  Pipe\_Reg #(.size(5))EX\_Write\_R(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(ID\_EX\_Write\_R),  .data\_o(EX\_M\_Write\_R)  );  /\*\*\*\* EX stage \*\*\*\*/  /\*\*\*\* MEM stage \*\*\*\*/  and(PCSrc,EX\_M\_Branch,M\_zero);  Data\_Memory DM(  .clk\_i(clk\_i),  .addr\_i(M\_Alu\_result),  .data\_i(WriteData),  .MemRead\_i(EX\_M\_MemRead),  .MemWrite\_i(EX\_M\_MemWrite),  .data\_o(Memdata\_o)  );  /\*\*\*\* MEM stage \*\*\*\*/  /\*\*\*\* MEM stage \*\*\*\*/  Pipe\_Reg #(.size(1))M\_WB(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(EX\_M\_RegWrite),  .data\_o(regwrite)  );  Pipe\_Reg #(.size(1))M\_WB2(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(EX\_M\_MemToReg),  .data\_o(MemToReg)  );  Pipe\_Reg #(.size(32))M\_DM(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(Memdata\_o),  .data\_o(WB\_Memdata\_o)  );  Pipe\_Reg #(.size(32))M\_Alu(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(M\_Alu\_result),  .data\_o(WB\_Alu\_result)  );  Pipe\_Reg #(.size(5))M\_Write\_R(  .rst\_n(rst\_n),  .clk\_i(clk\_i),  .data\_i(EX\_M\_Write\_R),  .data\_o(final\_RD)  );  /\*\*\*\* MEM stage \*\*\*\*/  /\*\*\*\* WB stage \*\*\*\*/  MUX\_2to1 #(.size(32)) Mux\_MtoR(  .data0\_i(WB\_Memdata\_o),  .data1\_i(WB\_Alu\_result),  .select\_i(MemToReg),  .data\_o(final\_data)  );  /\*\*\*\* WB stage \*\*\*\*/  endmodule |
|  |
|  |
| C:\Users\1q099\AppData\Local\Microsoft\Windows\INetCache\Content.Word\1.png |