《计算机组成原理》实验报告

年级、专业、班级	2023 级计算机科学与技术 06 班 2023 级计算机科学与技术 04 班 2023 级计算机科学与技术 04 班	**姓名**	饶格奇 刘雨霜 李隆征
实验题目	实验四简单五级流水线 CPU		
实验时间	2025 年 5 月 13 日	**实验地点**	DS1410
实验成绩	优秀/良好/中等	**实验性质**	□验证性 ☑设计性 □综合性

教师评价：

□算法/实验过程正确； □源程序/实验内容提交； □程序结构/实验步骤合理；

□实验结果正确； □语法、语义正确； □报告规范；

其他:

评价教师: 任骜

实验目的

（1）掌握流水线 (Pipelined) 处理器的思想。

（2）掌握单周期处理中执行阶段的划分。

（3）了解流水线处理器遇到的冒险。

（4）掌握数据前推、流水线暂停等冒险解决方式。

报告完成时间: 2025 年 5 月 14 日

1 实验内容

阅读实验原理实现以下模块：

(1) Datapath，所有模块均可由实验三复用，需根据不同阶段，修改 mux2 为 mux3(三选一选择器)，以及带有 enable(使能)、clear(清除流水线) 等信号的触发器，
(2) Controller，其中 main decoder 与 alu decoder 可直接复用，另需增加触发器在不同阶段进行信号传递
(3) 指令存储器 inst_mem(Single Port Ram)，数据存储器 data_mem(Single Port Ram)；同实验三一致，无需改动，
(4) 参照实验原理，在单周期基础上加入每个阶段所需要的触发器，重新连接部分信号。实验给出 top 文件，需兼容 top 文件端口设定。
(5) 实验给出仿真程序，最终以仿真输出结果判断是否成功实现要求指令。

2 实验设计

2.1 数据通路模块

2.1.1 功能描述

Datapath 模块实现指令的取指、译码、执行、访存与回写五大阶段的数据流与运算逻辑。

2.1.2 接口定义

表 1: Datapath 接口

信号名	方向	位宽	功能描述
clk	input	1	时钟信号
reset	input	1	同步复位信号
instr	input	32	当前取出的指令,来自指令存储器
pc	output	32	当前 PC 地址,供指令存储器使用
memtoregD	input	1	ID 阶段:写回阶段是否从内存读取
memwriteD	input	1	ID 阶段:是否写入内存
branchD	input	1	ID 阶段:是否为条件分支指令
alusrcD	input	1	ID 阶段:ALU 第二个操作数来源选择(寄存器或立即数)
regdstD	input	1	ID 阶段:写寄存器地址选择(rd 或 rt)
regwriteD	input	1	ID 阶段:是否写寄存器
jumpD	input	1	ID 阶段:是否为跳转指令
alucontrolD	input	3	ID 阶段:ALU 操作控制信号
EqualD	output	1	ID 阶段:比较 rs 与 rt 是否相等(用于分支判断)
StallD	output	1	ID 阶段:是否暂停流水线(冒险检测结果)
pcsrcD	output	1	IF 阶段:是否进行条件跳转(EqualD & branchD)
op	output	6	ID 阶段: 指令操作码字段 instr[31:26]
funct	output	6	ID 阶段:R 型指令功能码字段 instr[5:0]
readdata	input	32	MEM 阶段:从数据存储器读取的数据
aluout_M	output	32	MEM 阶段:ALU 计算结果,写数据存储器地址或写回用
Writedata_M	output	32	MEM 阶段:写入数据存储器的数据
memwriteM	output	1	MEM 阶段:是否进行数据存储器写操作

2.2 冒险处理模块

2.2.1 功能描述

Hazard 模块用于检测和处理 MIPS 五级流水线中的数据冒险与控制冒险，通过转发（Forwarding）与暂停（Stall）机制保证数据正确性和指令正确执行。

2.2.2 接口定义

表 2: Hazard 接口定义

信号名	方向	位宽	功能描述
writereg_e	input	5	EX 阶段要写回的寄存器号（用于检测 EX 阶段的 RAW 冒险）
writereg_m	input	5	MEM 阶段要写回的寄存器号（用于前推判断）
writereg_w	input	5	WB 阶段要写回的寄存器号（用于前推判断）
regwriteE	input	1	EX 阶段写回使能信号
regwriteM	input	1	MEM 阶段写回使能信号
regwriteW	input	1	WB 阶段写回使能信号
branchD	input	1	ID 阶段当前指令是否为分支指令
rsE	input	5	EX 阶段的源寄存器 rs
rtE	input	5	EX 阶段的源寄存器 rt
rsD	input	5	ID 阶段的源寄存器 rs
rtD	input	5	ID 阶段的源寄存器 rt
memtoregE	input	1	EX 阶段指令是否从内存读取数据（lw）
memtoregM	input	1	MEM 阶段指令是否从内存读取数据（lw）
forwardAE	output	2	EX 阶段 ALU 第一个操作数的前推控制信号
forwardBE	output	2	EX 阶段 ALU 第二个操作数的前推控制信号
forwardAD	output	1	ID 阶段用于分支判断的 rs 是否需要前推
forwardBD	output	1	ID 阶段用于分支判断的 rt 是否需要前推
StallF	output	1	是否暂停 IF 阶段（冒险检测结果）
StallD	output	1	是否暂停 ID 阶段（冒险检测结果）
FlushE	output	1	是否清空 EX 阶段寄存器（防止错误执行）

3 实验过程记录

3.1 问题 1: 仿真多数据出现不定态 X

问题描述:refile 的内部信号未初始化 0,加之 controller 中的控制信号在某些情况规定为 x,虽然逻辑上在某些情况这些控制信号是 0 是 1 都是可以的,但是会影响到其他运算,导致不定态在运算中不断传递。

解决方案:将 controller 中的控制信号为 x 改成 0。

3.2 问题 2:branch 跳转错误

问题描述:

解决方案:

3.3 问题 3:jump 跳转错误

问题描述:

解决方案:

4 实验结果及分析

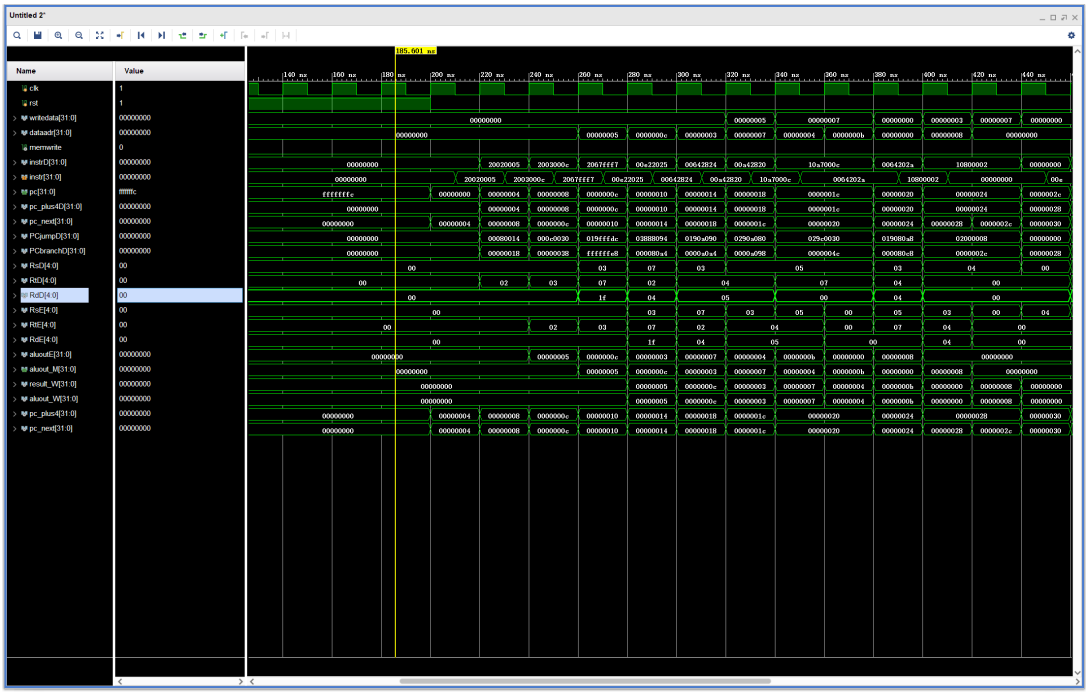

图 1: 仿真结果 1

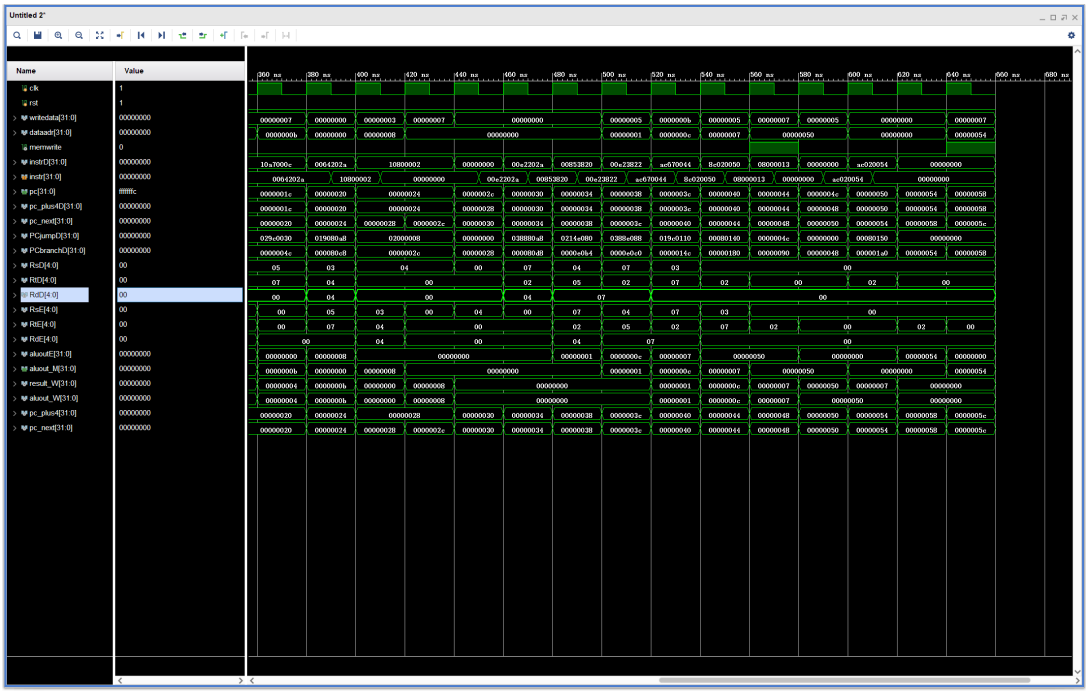

图 2: 仿真结果 2

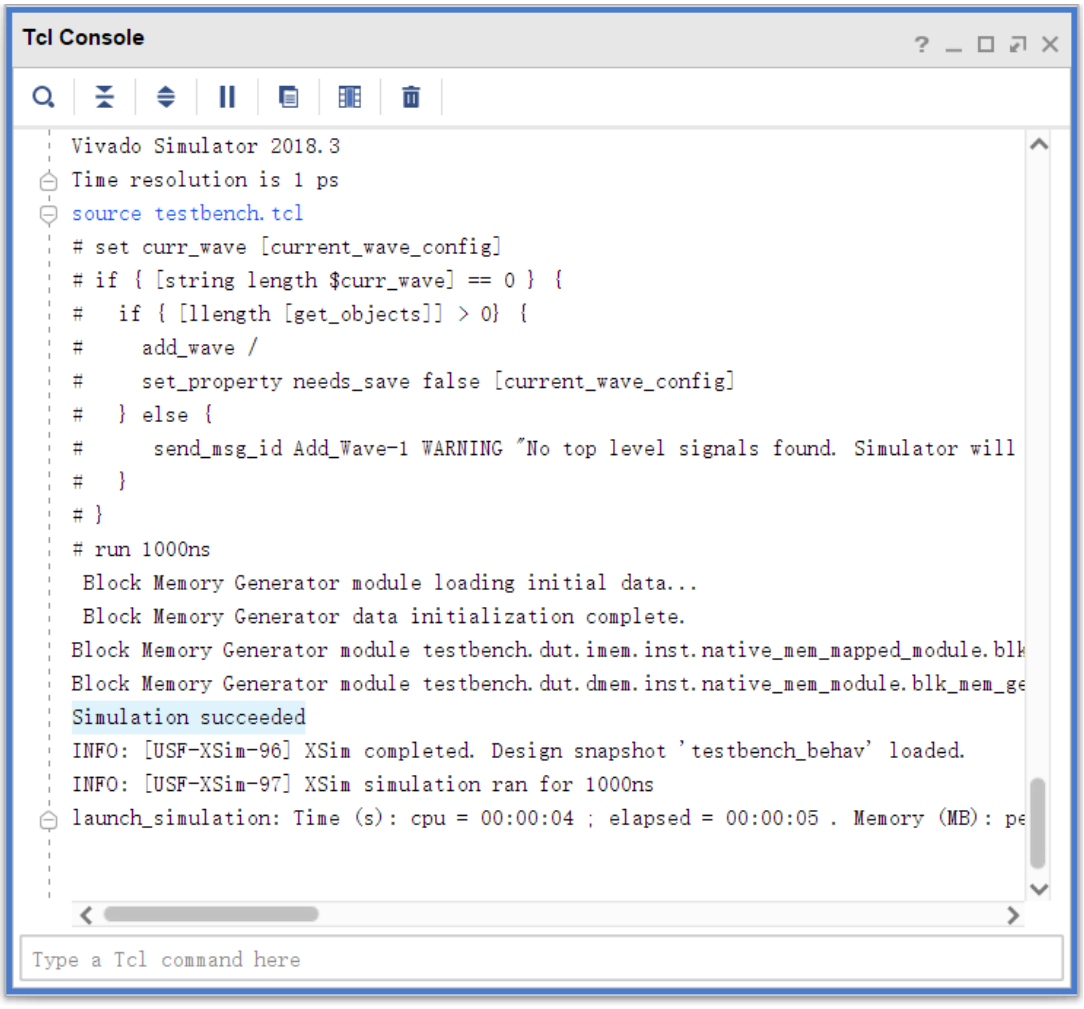

图 3: Simulation succeeded

A Datapath 代码

```
`timescale 1ns / 1ps

module datapath(
input clk,reset,

//Fetch

input [31:0]instr,
output [31:0] pc,

//Decode
input memtoregD,memwriteD,branchD,alusrcD,regdstD,regwriteD,jumpD,
input [2:0] alucontrolD,
output wire EqualD,StallD,pcsrcD,
output wire [5:0] op,funct,
//Execute
// No IO signals

//Memory
input [31:0]readdata,
//output wire zero,
output [31:0] aluout_M, Writedata_M,
output wire memwriteM
//Writeback
// No IO signals
);
wire [31:0]instrD;
wire [31:0] extended_dataD,shifted_data;
wire [31:0] pc_plus4,PCjumpD;
wire [31:0] RD1,RD2;
wire [31:0] pc_next; //pc+4或 ? branch
wire [31:0] pc__next; //pc_next或 ? jump

//assign writedata=RD2;

//提前声明的变 ?
//Hazard
wire [1:0] forwardAE,forwardBE;
wire StallF,FlushE;

// Fetch -> Decode
wire [31:0] PCbranchD, pc_plus4D;
wire [4:0] RsD,RtD,RdD;
wire memtoregD,memwriteD,branchD,alusrcD,regdstD,regwriteD,jumpD;
wire [2:0] alucontrolD;
wire [31:0] Forward_RD1,Forward_RD2;
wire forwardAD,forwardBD;
```

```
wire EqualD;



// Decode -> Excute
wire [31:0] RD1_E,RD2_E;
wire [4:0] RsE, RtE, RdE, WriteregE;
wire [31:0] SrcAE, SrcBE, SrcBE_temp;
wire [31:0] extended_dataE;
wire [31:0] aluoutE;
//wire [31:0] pc_plus4_E;

wire memtoregE,memwriteE,alusrcE,regdstE,regwriteE;
wire [2:0] alucontrolE;
// Excute -> Memory
wire [31:0] aluout_M;
//wire [31:0] PCbranch_M;
wire [4:0]  WriteregM;
//wire zero_M;

wire memtoregM,memwriteM,regwriteM;



wire [31:0] result_W;
// Memory -> Writeback
wire regwriteW;
wire [4:0] WriteregW;
wire [31:0] readdata_W;
wire [31:0] aluout_W;
wire  memtoregW;
//Fetch stage #1
/*
PC+取指 ?
*/
assign PCjumpD={pc_plus4D[31:28],instrD[25:0],2'b00};

pc PC(          //修改
.clk(clk),   //out:pc
.rst(reset),     //input:pc__next
.pc(pc),
.ena(~StallF),
.pc_next(pc__next)
);

mux21 #(32)pc_mux(.data_in0(pc_plus4),.data_in1(PCbranchD),.sel(pcsrcD),.
    data_out(pc_next));
mux21 #(32)pc2_mux(.data_in0(pc_next),.data_in1(PCjumpD),.sel(jumpD),.data_out(
    pc__next));
adder pc_plus_4(.a(pc),.b(32'h4),.y(pc_plus4));
```

```
// Fetch -> Decode
//////////////////////////////////////////////////////////
//flopr #(32)fd_inst (.clk(clk), .rst(reset), .din(instr), .dout(instrD));
flopenrc #(32)fd_instr (.clk(clk), .rst(reset), .clear(1'b0), .en(~StallD), .
    din(instr), .dout(instrD));
flopenrc #(32)fd_pc_plus4 (.clk(clk), .rst(reset), .clear(1'b0), .en(~StallD),
    .din(pc_plus4), .dout(pc_plus4D));
//////////////////////////////////////////////////////////


//Decode stage #2

//signals
assign op=instrD[31:26];
assign funct=instrD[5:0];
//assign {memtoregD,memwriteD,branchD,alusrcD,regdstD,regwriteD,jumpD,
    alucontrolD}={memtoreg,memwrite,branch,alusrc,regdst,regwrite,jump,
    alucontrol};
assign {RsD,RtD,RdD}={instrD[25:21],instrD[20:16],instrD[15:11]};
regfile R(
.clk(clk),
.we3(regwriteW),
.ra1(instrD[25:21]),.ra2(instrD[20:16]),.wa3(WriteregW),
.wd3(result_W),
.rd1(RD1),.rd2(RD2)
);

sign_extension SE(
.data_in(instrD[15:0]),
.data_out(extended_dataD)
);

shift_2 #(32)sh2 (
.data_in(extended_dataD),
.data_out(shifted_data)
);
adder A(.a(shifted_data),.b(pc_plus4D),.y(PCbranchD));

mux21 #(32)forward1_mux(
.data_in0(RD1),
.data_in1(aluout_M),
.sel(forwardAD),
.data_out(Forward_RD1)
);

mux21 #(32)forward2_mux(
.data_in0(RD2),
```

```
.data_in1(aluout_M),
.sel(forwardBD),
.data_out(Forward_RD2)
);

assign EqualD = (Forward_RD1==Forward_RD2) ? 1'b1 : 1'b0;
assign pcsrcD = (EqualD & branchD);

// Decode -> Execute
////////////////////////////////////////////////////////////
floprc #(1)de_memtoreg (.clk(clk), .rst(reset), .clear(FlushE), .din(memtoregD)
    , .dout(memtoregE));
floprc #(1)de_memwrite (.clk(clk), .rst(reset), .clear(FlushE), .din(memwriteD)
    , .dout(memwriteE));
floprc #(1)de_alusrc (.clk(clk), .rst(reset), .clear(FlushE), .din(alusrcD), .
    dout(alusrcE));
floprc #(1)de_reg_dst (.clk(clk), .rst(reset), .clear(FlushE), .din(regdstD), .
    dout(regdstE));
floprc #(1)de_reg_write (.clk(clk), .rst(reset), .clear(FlushE), .din(regwriteD
    ), .dout(regwriteE));
floprc #(3)de_alucontrol (.clk(clk), .rst(reset), .clear(FlushE), .din(
    alucontrolD), .dout(alucontrolE));

floprc #(5)de_inst3 (.clk(clk), .rst(reset), .clear(FlushE), .din(RsD), .dout(
    RsE));
floprc #(5)de_inst1 (.clk(clk), .rst(reset), .clear(FlushE), .din(RtD), .dout(
    RtE));
floprc #(5)de_inst2 (.clk(clk), .rst(reset), .clear(FlushE), .din(RdD), .dout(
    RdE));
floprc #(32)de_signDate (.clk(clk), .rst(reset), .clear(FlushE), .din(
    extended_dataD), .dout(extended_dataE));
floprc #(32)de_reg_RD1 (.clk(clk), .rst(reset), .clear(FlushE), .din(RD1), .
    dout(RD1_E));
floprc #(32)de_reg_RD2 (.clk(clk), .rst(reset), .clear(FlushE), .din(RD2), .
    dout(RD2_E));
////////////////////////////////////////////////////////////
//Execute stage #3
mux31 #(32) alu_srcA(.a(RD1_E),.b(result_W),.c(aluout_M),.sel(forwardAE),.out(
    SrcAE));
mux31 #(32) alu_srcB(.a(RD2_E),.b(result_W),.c(aluout_M),.sel(forwardBE),.out(
    SrcBE_temp));
mux21 #(5)reg_mux(
.data_in0(RtE),
.data_in1(RdE),
.sel(regdstE),
.data_out(WriteregE)
);
mux21 #(32)alu_mux(
.data_in0(SrcBE_temp),
```

```
.data_in1(extended_dataE),
.sel(alusrcE),
.data_out(SrcBE)
);
//Data来自extended_data或 ? shifted_data
ALU alu(.SrcA(SrcAE),.SrcB(SrcBE),.opcode(alucontrolE),.aluresult(aluoutE));


// Excute -> Memory
//////////////////////////////////////////////////////////////
floprc #(1)em_memtoreg (.clk(clk), .rst(reset), .clear(1'b0), .din(memtoregE),
    .dout(memtoregM));
floprc #(1)em_memwrite (.clk(clk), .rst(reset), .clear(1'b0), .din(memwriteE),
    .dout(memwriteM));
floprc #(1)em_regwrite (.clk(clk), .rst(reset), .clear(1'b0), .din(regwriteE),
    .dout(regwriteM));
//floprc #(1)em_zero (.clk(clk), .rst(reset), .clear(1'b0), .din(zero), .dout(
    zero_M));

floprc #(32)em_aluout (.clk(clk), .rst(reset), .clear(1'b0), .din(aluoutE), .
    dout(aluout_M));
floprc #(32)em_writedata (.clk(clk), .rst(reset), .clear(1'b0), .din(SrcBE_temp
    ), .dout(Writedata_M));
floprc #(5)em_writereg (.clk(clk), .rst(reset), .clear(1'b0), .din(WriteregE),
    .dout(WriteregM));
//floprc #(32)em_PCbranch (.clk(clk), .rst(reset), .clear(1'b0), .din(PCbranch)
    , .dout(PCbranch_M));
//////////////////////////////////////////////////////////////




//Memory stage #4


// Memory -> Writeback
//////////////////////////////////////////////////////////////
flopr #(1)mw_regwrite (.clk(clk), .rst(reset), .din(regwriteM), .dout(regwriteW
    ));
flopr #(1)mw_memtoreg (.clk(clk), .rst(reset), .din(memtoregM), .dout(memtoregW
    ));

flopr #(5)mw_write_reg (.clk(clk), .rst(reset), .din(WriteregM), .dout(
    WriteregW));
flopr #(32)mw_readdata (.clk(clk), .rst(reset), .din(readdata), .dout(
    readdata_W));
flopr #(32)mw_aluout (.clk(clk), .rst(reset), .din(aluout_M), .dout(aluout_W));
//////////////////////////////////////////////////////////////
```

```
//Writeback stage #5

mux21 #(32)mem_mux(
.data_in0(aluout_W),
.data_in1(readdata_W),
.sel(memtoregW),
.data_out(result_W)
);

Hazard h(.regwriteE(regwriteE), .regwriteM(regwriteM), .regwriteW(regwriteW), .
    writereg_e(WriteregE), .writereg_w(WriteregW), .writereg_m(WriteregM),
.memtoregE(memtoregE), .memtoregM(memtoregM),
.rsE(RsE), .rtE(RtE), .rsD(RsD), .rtD(RtD),
.forwardAE(forwardAE), .forwardBE(forwardBE), .StallF(StallF), .StallD(StallD),
     .FlushE(FlushE),
.branchD(branchD), .forwardAD(forwardAD), .forwardBD(forwardBD)
);


endmodule
```

B Hazard 代码

```
‘timescale 1ns / 1ps

module Hazard(
    input [4:0] writereg_e, writereg_w, writereg_m,
    input regwriteE, regwriteM, regwriteW, branchD,
    input [4:0] rsE, rtE, rsD, rtD,
    input memtoregE, memtoregM,
    output reg [1:0] forwardAE,forwardBE,
    output wire forwardAD, forwardBD,
    output wire StallF, StallD, FlushE
    );
    //Forwarding
    wire branchstall, lwstall;
    always @(*) begin
        forwardAE = 2'b00;
        forwardBE = 2'b00;
        if (rsE != 0) begin
            if ((rsE == writereg_m) & regwriteM) begin
                forwardAE = 2'b10;
            end //ALU数据前推
            else if ((rsE == writereg_w) & regwriteW) begin
                forwardAE = 2'b01;
            end //寄存？
            else begin
```

```
                forwardAE = 2'b00;
            end
        end

        if (rtE != 0) begin
            if ((rtE == writereg_m) & regwriteM) begin
                forwardBE = 2'b10;
            end
            else if ((rtE == writereg_w) & regwriteW) begin
                forwardBE = 2'b01;
            end
            else begin
                forwardBE = 2'b00;
            end
        end
    end
    //Stall
    assign lwstall = (rsD == rtE | rtD == rtE) & memtoregE;

    //branch prediction

    assign forwardAD = (rsD != 0) & (rsD == writereg_m) & regwriteM;
    assign forwardBD = (rtD != 0) & (rtD == writereg_m) & regwriteM;


    assign branchstall = branchD & ( regwriteE & (writereg_e == rsD |
        writereg_e == rtD)
                        | memtoregM & ( writereg_m == rsD | writereg_m == rtD ))
                            ;
    assign StallD = branchstall | lwstall;
    assign StallF = branchstall | lwstall;
    assign FlushE = branchstall | lwstall;
endmodule
```

C Controller 代码

```
`timescale 1ns / 1ps

module controller(
input [5:0] Inst_part_31_26,
input [5:0] Inst_part_5_0,
input zero,
output wire jump,alusrc,memwrite,memtoreg,regwrite,regdst,branch,
output wire [2:0]Alcontrol
);
```

```
wire [1:0]Alop;

maindec MD(
.Inst_part_31_26(Inst_part_31_26),
.jump(jump),
.zero(zero),
.branch(branch),
.alusrc(alusrc),
.memwrite(memwrite),
.memtoreg(memtoreg),
.regwrite(regwrite),
.regdst(regdst),
.Alop(Alop)
);

aludec AL(
.Inst_part_5_0(Inst_part_5_0),
.Alop(Alop),
.Alcontrol(Alcontrol)
);

endmodule
```