计算机组成原理 实验报告

姓名：杨博涵　学号：PB20000328　实验日期：2022.5.7

一、实验题目：

Lab05 流水线 CPU

二、实验源码：

IF：

```
module IF
(
      input clk,
      input rst,
      input IFIDwrite,
      input PCsrc,
      input [31:0] addsum,
      output reg [31:0] pc_reg,
      output reg [31:0] instruction_reg,
      output [31:0] pc
);

      wire [31:0] instruction;
      wire [31:0] next_pc;
      wire [31:0] in0;
      wire [31:0] in1;

mux2 jalmux(in0,in1,PCsrc,pc);
instruction_mem instruction_mem(
  .a(pc[9:2]),      // input wire [7 : 0] a
  .spo(instruction)  // output wire [31 : 0] spo
);

assign in0=pc_reg+4;
assign in1=addsum;
```

```
//if-id reg
always@(posedge clk or posedge rst)
begin
      if(rst==1) begin pc_reg<=32'h2ffc;   instruction_reg<=0;end
      else if(IFIDwrite==1) begin pc_reg<=pc;
instruction_reg<=instruction;end
end


endmodule
```

ID：

```
`define I_type 3'b000
`define S_type 3'b001
`define SB_type 3'b010
`define UJ_type 3'b011
`define U_type 3'b100

`define R_type 7'b0110011
`define addi 7'b0010011
`define lw  7'b0000011
`define sw 7'b0100011
`define beq 7'b1100011
`define jal 7'b1101111
`define nop 7'b0
```

```
module ID
(
	input clk,
	input rst,
	input RegWritein,
	input [31:0] pc,
	input [31:0] INSTRUCTION,
	input [4:0] wb_addrin,
	input [31:0] writeback,
	output [31:0] rf_data,
	input [7:0] m_rf_addr,
	input flush_idex,
	output reg [31:0] rd0_reg,
	output reg [31:0] rd1_reg,
	output reg [31:0] imm_reg,
	output reg [4:0] wb_addr_reg,
	output reg [31:0] pc_reg,
	output reg [31:0] ctrl_reg,
	output reg [4:0] rs0_reg,
	output reg [4:0] rs1_reg
);


wire [31:0] rd1,rd0,imm;
wire [4:0] wb_addr;
reg
RegWrite,Branch,jal,ALUctr1,ALUctr2,branch_sel,jalr,RegScr,MemWrite,Mem
Wread;
reg [2:0] Imm_gen;
reg [2:0] ALUop;
wire [31:0] ctrl,ctrl_sel;
```

```
ImmGen ImmGen(Imm_gen,INSTRUCTION,imm);
rf
Register(.rst(rst),.clk(clk),.ra0(INSTRUCTION[19:15]),.ra1(INSTRUCTION[
24:20]),.ra2(m_rf_addr[4:0]),.wa(wb_addrin),.we(RegWritein),.rd0(rd0),.
rd1(rd1),.rd2(rf_data),.wd(writeback));
mux2 flushmux(ctrl,32'b0,flush_idex,ctrl_sel);


assign wb_addr=INSTRUCTION[11:7];


//control unit
always@(*)
begin
      case(INSTRUCTION[6:0])
            `R_type: begin
jal=0;Branch=0;Imm_gen=3'b111;RegScr=1'b0;ALUop=3'b000;MemWrite=0;ALUct
r2=0;RegWrite=1; MemWread=0;end
            `addi: begin
jal=0;Branch=0;Imm_gen=`I_type;RegScr=1'b0;ALUop=3'b000;MemWrite=0;ALUc
tr2=1;RegWrite=1; MemWread=0;end
            `lw:begin
jal=0;Branch=0;Imm_gen=`I_type;RegScr=1'b1;ALUop=3'b000;MemWrite=0;ALUc
tr2=1;RegWrite=1; MemWread=1;end
            `sw:begin
jal=0;Branch=0;Imm_gen=`S_type;RegScr=1'b0;ALUop=3'b0;MemWrite=1;ALUctr
2=1;RegWrite=0; MemWread=0; end
            `beq:begin
jal=0;Branch=1;Imm_gen=`SB_type;RegScr=1'b0;ALUop=3'b001;MemWrite=0;ALU
ctr2=0;RegWrite=0;MemWread=0; end
```

```
            `jal:begin
jal=1;Branch=0;Imm_gen=`UJ_type;RegScr=1'b0;ALUop=3'b000;MemWrite=0;ALU
ctr2=1;RegWrite=1; MemWread=0;end
            `nop:begin
jal=0;Branch=0;Imm_gen=3'b111;RegScr=1'b0;ALUop=3'b0;MemWrite=0;ALUctr2
=0;RegWrite=0; MemWread=0;end
            default:begin
jal=0;Branch=0;Imm_gen=3'b111;RegScr=1'b0;ALUop=3'b00;MemWrite=0;ALUctr
2=0;RegWrite=0;MemWread=0;end
      endcase
end

//ex
assign ctrl[2:0]=ALUop;
assign ctrl[3]=0;
assign ctrl[4]=ALUctr2;
assign ctrl[7:5]=0;
assign ctrl[8]=Branch;
assign ctrl[9]=jal;
assign ctrl[11:10]=0;

//mem
assign ctrl[12]=MemWrite;
assign ctrl[13]=MemWread;
assign ctrl[15:14]=0;

//wb
assign ctrl[16]=RegScr;
assign ctrl[17]=RegWrite;
assign ctrl[31:18]=0;
```

```
//id-ex reg
always@(posedge clk or posedge rst ) //flush 不能异步！！
begin
      if(rst==1||flush_idex==1) begin rd0_reg<=0;
                                rd1_reg<=0;
                                imm_reg<=0;
                                wb_addr_reg<=0;
                                pc_reg<=0;
                                ctrl_reg<=0;
                                rs0_reg<=0;
                                rs1_reg<=0;
                          end
      else  begin rd0_reg<=INSTRUCTION[19:15]==0 ?0 :rd0;
                               rd1_reg<=INSTRUCTION[24:20]==0 ?0 :rd1;
                               imm_reg<=imm;
                               wb_addr_reg<=wb_addr;
                               pc_reg<=pc;
                               ctrl_reg<=ctrl_sel;
                               rs0_reg<=INSTRUCTION[19:15];
                               rs1_reg<=INSTRUCTION[24:20];
                          end
end
endmodule
```

```
module ImmGen
(
	input [2:0] Imm_gen,
	input [31:0] INSTRUCTION,
	output reg [31:0] imm
);

always@(*)
begin
	case(Imm_gen)
		`I_type: begin
					if(INSTRUCTION[31]==0)
					    imm={20'b0,INSTRUCTION[31:20]};
					else
					      imm={20'hFFFFF,INSTRUCTION[31:20]};
				end
		`S_type: begin
					if(INSTRUCTION[31]==0)

imm={20'b0,INSTRUCTION[31:25],INSTRUCTION[11:7]};
					else

	imm={20'hFFFFF,INSTRUCTION[31:25],INSTRUCTION[11:7]};
				end

		`SB_type: begin
					if(INSTRUCTION[31]==0)

imm={20'b0,INSTRUCTION[31],INSTRUCTION[7],INSTRUCTION[30:25],INSTRUCTIO
N[11:8]};
```

```
                                else

        imm={20'hFFFFF,INSTRUCTION[31],INSTRUCTION[7],INSTRUCTION[30:25],
INSTRUCTION[11:8]};
                            end
                `UJ_type: begin
                                if(INSTRUCTION[31]==0)

imm={12'b0,INSTRUCTION[31],INSTRUCTION[19:12],INSTRUCTION[20],INSTRUCTI
ON[30:21]};
                                else

        imm={12'hFFF,INSTRUCTION[31],INSTRUCTION[19:12],INSTRUCTION[20],I
NSTRUCTION[30:21]};
                            end
                `U_type: imm={INSTRUCTION[31:12],12'b0};
                default: imm=0;
        endcase
end
endmodule
```

EX：

```
module EX
(
        input clk,
        input rst,
        input [31:0] pc,
        input [4:0] wb_addr,
        input [31:0] rd0,
        input [31:0] rd1,
        input [31:0] imm,
```

```
        input [31:0] ctrl,
        input [31:0] src_mem,
        input [31:0] src_wb,
        input [1:0] afwd,
        input [1:0] bfwd,
        output reg [31:0] wd_reg,
        output reg [31:0] sum_reg,
        output reg [4:0] wb_addr_reg,
        output reg [31:0] ctrm_reg,
        output PCsrc,
        output [31:0] addsum,
        output flush_idex
);

wire z;
wire [31:0] sum,alu1,alu2,src;


alu alu(alu1,alu2,ctrl[2:0],sum,z);
mux2 alusrc(src,imm,ctrl[4],alu2);
mux3 afwdmux(rd0,src_mem,src_wb,afwd,alu1);
mux3 bfwdmux(rd1,src_mem,src_wb,bfwd,src);
//mux2 alusrc(rd1,imm,ctrl[4],src);
//mux3 afwdmux(rd0,src_mem,src_wb,afwd,alu1);
//mux3 bfwdmux(src,src_mem,src_wb,bfwd,alu2);

assign addsum=pc+(imm<<1);
assign PCsrc=ctrl[9]|(ctrl[8]&z);
assign flush_idex=PCsrc;

//ex-mem reg
always@(posedge clk or posedge rst)
```

```
begin
      if(rst==1)  begin
                              sum_reg<=0;
                              wb_addr_reg<=0;
                              ctrm_reg<=0;
                              wd_reg<=0;
                        end
      else        begin
                              sum_reg<=sum;
                              wb_addr_reg<=wb_addr;
                              ctrm_reg<=ctrl;
                              wd_reg<=src;
                        end
end


endmodule
```

MEM：

```
module MEM
(
      input clk,
      input rst,

      input [4:0] wb_addr,
      input [31:0] wd,
      input [31:0] sum,
      input [31:0] ctrm,
      output reg [31:0] memd_reg,
      output reg[31:0] sum_reg,
      output reg[31:0] ctrwb_reg,
      output reg[4:0] wb_addr_reg,
      output [31:0] m_data,
```

```
        output [7:0] io_addr,       //led 和 seg 的地址
        output io_we,                  //输出 led 和 seg 数据时的使能信号
        input [31:0] io_din,         //来自 sw 的输入数据
        input [7:0] m_rf_addr
);


wire [31:0] memd,mem_sel;
wire we;

data_mem date_mem (
  .a(sum[9:2]),        // input wire [7 : 0] a
  .d(wd),         // input wire [31 : 0] d
  .dpra(m_rf_addr),  // input wire [7 : 0] dpra
  .clk(clk),    // input wire clk
  .we(we),       // input wire we
  .spo(memd),    // output wire [31 : 0] spo
  .dpo(m_data)    // output wire [31 : 0] dpo
);

assign io_addr=sum[7:0];
assign we=ctrm[12]&~sum[10];
assign io_we=sum[10]&ctrm[12];
mux2 iomux(memd,io_din,sum[10],mem_sel);


//mem-wb reg
always@(posedge clk or posedge rst)
begin
      if(rst==1)begin memd_reg<=0;
                                sum_reg<=0;
                                wb_addr_reg<=0;
```

```
                                ctrwb_reg<=0;
                        end
        else  begin memd_reg<=mem_sel;
                                sum_reg<=sum;
                                wb_addr_reg<=wb_addr;
                                ctrwb_reg<=ctrm;
                        end
end




endmodule
```

WB：

```
module WB
(
        input clk,
        input rst,
        input [31:0] memd,
        input [31:0] sum,
        input [31:0] ctrwb,
        output [31:0] Regsrc,
        output Regwrite

);

mux2 regsel(sum,memd,ctrwb[16],Regsrc);

assign Regwrite=ctrwb[17];

endmodule
```

```
forwarding
module forward
(
	input [4:0] rs0,
	input [4:0] rs1,
	input [4:0] rdm,
	input [4:0] rdw,
	input regwrite_mem,
	input regwrite_wb,
	output reg [1:0] afwd,
	output reg [1:0] bfwd
);

always@(*)
begin
	if((rs0==rdm)&&(regwrite_mem==1)&&(rdm!=0))  afwd=2'b01;
	else if((rs0==rdw)&&(regwrite_wb==1&&(rdw!=0))) afwd=2'b10;
	else afwd=2'b00;

	if((rs1==rdm)&&(regwrite_mem==1)&&(rdm!=0))  bfwd=2'b01;
	else if((rs1==rdw)&&(regwrite_wb==1)&&(rdw!=0)) bfwd=2'b10;
	else bfwd=2'b00;
end

endmodule


module hazard
(
	input Memread,
	input [4:0] rde,
```

```
      input [31:0] instruction,
      output reg IFIDwrite,
      output reg flush_ldex

);

always@(*)
      begin

      if((rde==instruction[19:15]||instruction[24:20]==rde)&&Memread&&(
rde!=0))
                  begin
                        IFIDwrite=0;
                        flush_ldex=1;
                  end
            else
                  begin
                        IFIDwrite=1;
                        flush_ldex=0;
                  end
      end

endmodule
```

CPU 模块：

```
module  cpu (
  input clk,
  input rst,

  //IO_BUS
  output [7:0] io_addr,       //led 和 seg 的地址
  output [31:0] io_dout,      //输出 led 和 seg 的数据
```

```
    output io_we,                    //输出 led 和 seg 数据时的使能信号
    input [31:0] io_din,           //来自 sw 的输入数据

    //Debug_BUS
    input [7:0] m_rf_addr,    //存储器(MEM)或寄存器堆(RF)的调试读口地址
    output [31:0] rf_data,     //从 RF 读取的数据
    output [31:0] m_data,     //从 MEM 读取的数据
    output [31:0] pc,
    output [31:0] pcd,
    output [31:0] ir,
    output [31:0] pcin,

    // ID/EX 流水段寄存器
    output [31:0] pce,
    output [31:0] a,
    output [31:0] b,
    output [31:0] imm,
    output [4:0] rd,
    output [31:0] ctrl,

    // EX/MEM 流水段寄存器
    output [31:0] y,
    output [31:0] bm,
    output [4:0] rdm,
    output [31:0] ctrlm,

    // MEM/WB 流水段寄存器
    output [31:0] summ,
    output [31:0] memd,
    output [4:0] rdw,
    output [31:0] ctrlwb
);
```

```
wire PCsrc,Regwrite,flush_idex,flush_idex1,flush_idex2,IFIDwrite;
wire [31:0] addsum,instruction,wb_addr,wd,sume;
wire [31:0] Regsrc;
wire [4:0] rs0,rs1;
wire [1:0] afwd,bfwd;
//wire [4:0] wb_addre,wb_addrm,wb_addrwb;
//wire [19:0] ctrlm;
//wire [15:0] ctrlwb;

//assign rd=wb_addre;
//assign rdm=wb_addrm;
//assign rdw=wb_addrwb;
assign y=sume;
assign bm=wd;
assign io_dout=wd;
assign ir=instruction;
assign pcin=addsum;
assign flush_idex=flush_idex1|flush_idex2;

IF IF
(
      .clk(clk),
      .rst(rst),
      .pc(pc),
      .PCsrc(PCsrc),
      .IFIDwrite(IFIDwrite),
      .addsum(addsum),
      .pc_reg(pcd),
      .instruction_reg(instruction)
);
```

```
ID ID
(
      .clk(clk),
      .rst(rst),

      .RegWritein(Regwrite),
      .pc(pcd),
      .rf_data(rf_data),
      .m_rf_addr(m_rf_addr),
      .INSTRUCTION(instruction),
      .wb_addrin(rdw),
      .writeback(Regsrc),
      .rd0_reg(a),
      .rd1_reg(b),
      .flush_idex(flush_idex),
      .imm_reg(imm),
      .wb_addr_reg(rd),
      .pc_reg(pce),
      .ctrl_reg(ctrl),
      .rs0_reg(rs0),
      .rs1_reg(rs1)

);


EX EX
(
      .clk(clk),
      .rst(rst),

      .pc(pce),
      .wb_addr(rd),
```

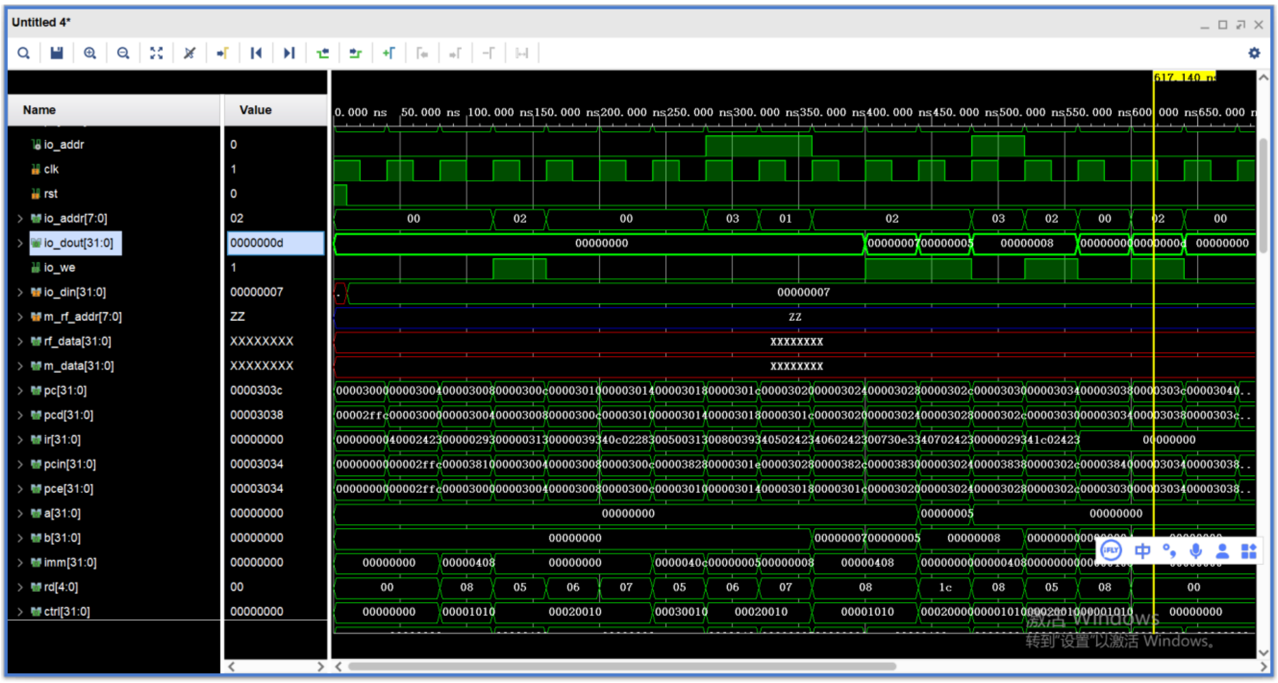
```
        .rd0(a),
        .rd1(b),
        .imm(imm),
        .ctrl(ctrl),
        .src_mem(sume),
        .src_wb(Regsrc),
        .afwd(afwd),
        .bfwd(bfwd),
        .wd_reg(wd),
        .sum_reg(sume),
        .wb_addr_reg(rdm),
        .ctrm_reg(ctrlm),
        .PCsrc(PCsrc),
        .addsum(addsum),
        .flush_idex(flush_idex1)
);

MEM MEM
(
        .clk(clk),
        .rst(rst),
        .m_data(m_data),
        .m_rf_addr(m_rf_addr),
        .wb_addr(rdm),
        .wd(wd),
        .sum(sume),
        .ctrm(ctrlm),
        .memd_reg(memd),
        .sum_reg(summ),
        .ctrwb_reg(ctrlwb),
        .wb_addr_reg(rdw),
        .io_addr(io_addr),      //led 和 seg 的地址
```

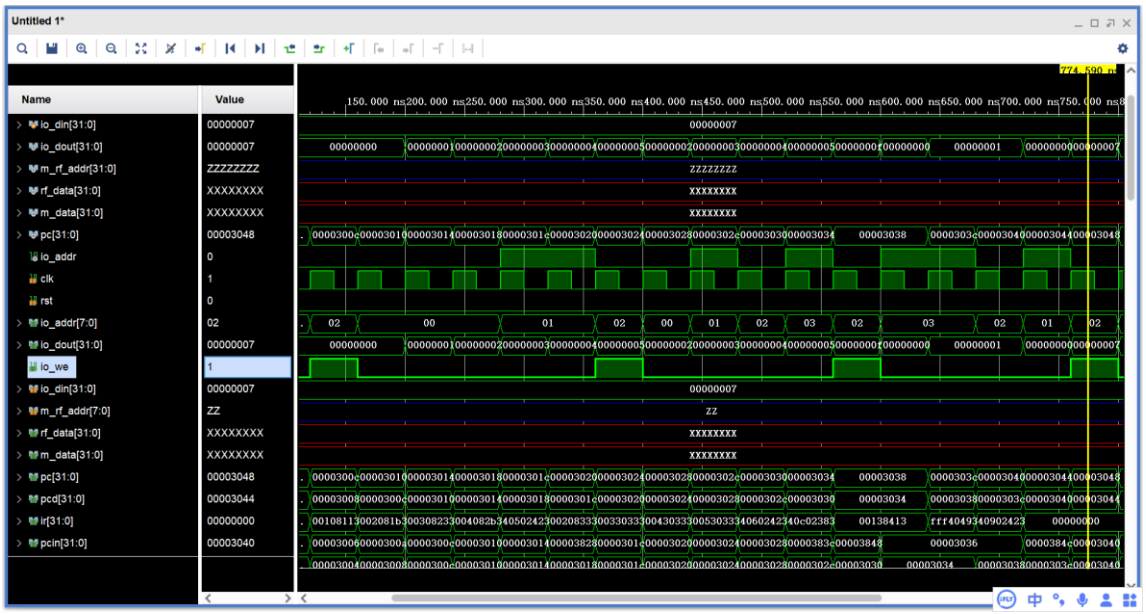
```
      .io_we(io_we),                    //输出 led 和 seg 数据时的使能信号
      .io_din(io_din)
);

WB WB
(
      .clk(clk),
      .rst(rst),

      .memd(memd),
      .sum(summ),
      .ctrwb(ctrlwb),
      .Regsrc(Regsrc),
      .Regwrite(Regwrite)

);


forward forward
(
      .regwrite_mem(ctrlm[17]),
      .regwrite_wb(ctrlwb[17]),
      .rs0(rs0),
      .rs1(rs1),
      .rdm(rdm),
      .rdw(rdw),
      .afwd(afwd),
      .bfwd(bfwd)
);

hazardhazard
(
```

```
        .Memread(ctrl[13]),
        .rde(rd),
        .instruction(instruction),
        .IFIDwrite(IFIDwrite),
        .flush_ldex(flush_idex2)

);
endmodule
```

三、实验思路:

下图为一些在测试指令时的仿真结果:





可以看到，io_out 在 io_we 的采样下输出了 50，5，15，7，27，与预期一致。

四、实验记录：

实验中出现两大问题：

1. flush_idex 与 flush_Idex 编译器不区别，语法检测通过，但是线却没有连上。最后人工排除。
2. 在 3054 读取 x0 的值时，读出结果为 4，无法找到逻辑问题，最后通过冗余解除。