**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,MemWread;

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;ALUctr2=0;RegWrite=1; MemWread=0;end

`addi: begin jal=0;Branch=0;Imm\_gen=`I\_type;RegScr=1'b0;ALUop=3'b000;MemWrite=0;ALUctr2=1;RegWrite=1; MemWread=0;end

`lw:begin jal=0;Branch=0;Imm\_gen=`I\_type;RegScr=1'b1;ALUop=3'b000;MemWrite=0;ALUctr2=1;RegWrite=1; MemWread=1;end

`sw:begin jal=0;Branch=0;Imm\_gen=`S\_type;RegScr=1'b0;ALUop=3'b0;MemWrite=1;ALUctr2=1;RegWrite=0; MemWread=0; end

`beq:begin jal=0;Branch=1;Imm\_gen=`SB\_type;RegScr=1'b0;ALUop=3'b001;MemWrite=0;ALUctr2=0;RegWrite=0;MemWread=0; end

`jal:begin jal=1;Branch=0;Imm\_gen=`UJ\_type;RegScr=1'b0;ALUop=3'b000;MemWrite=0;ALUctr2=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;ALUctr2=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],INSTRUCTION[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],INSTRUCTION[30:21]};

else

imm={12'hFFF,INSTRUCTION[31],INSTRUCTION[19:12],INSTRUCTION[20],INSTRUCTION[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)

);

hazard hazard

(

.Memread(ctrl[13]),

.rde(rd),

.instruction(instruction),

.IFIDwrite(IFIDwrite),

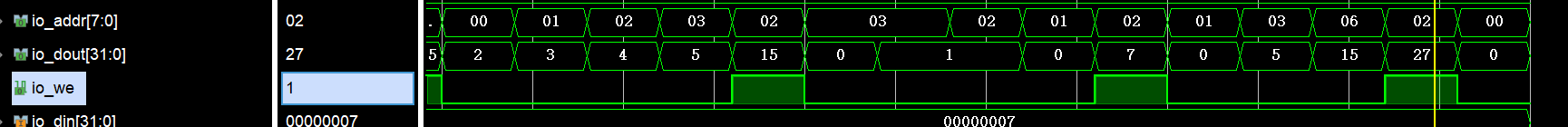
.flush\_ldex(flush\_idex2)

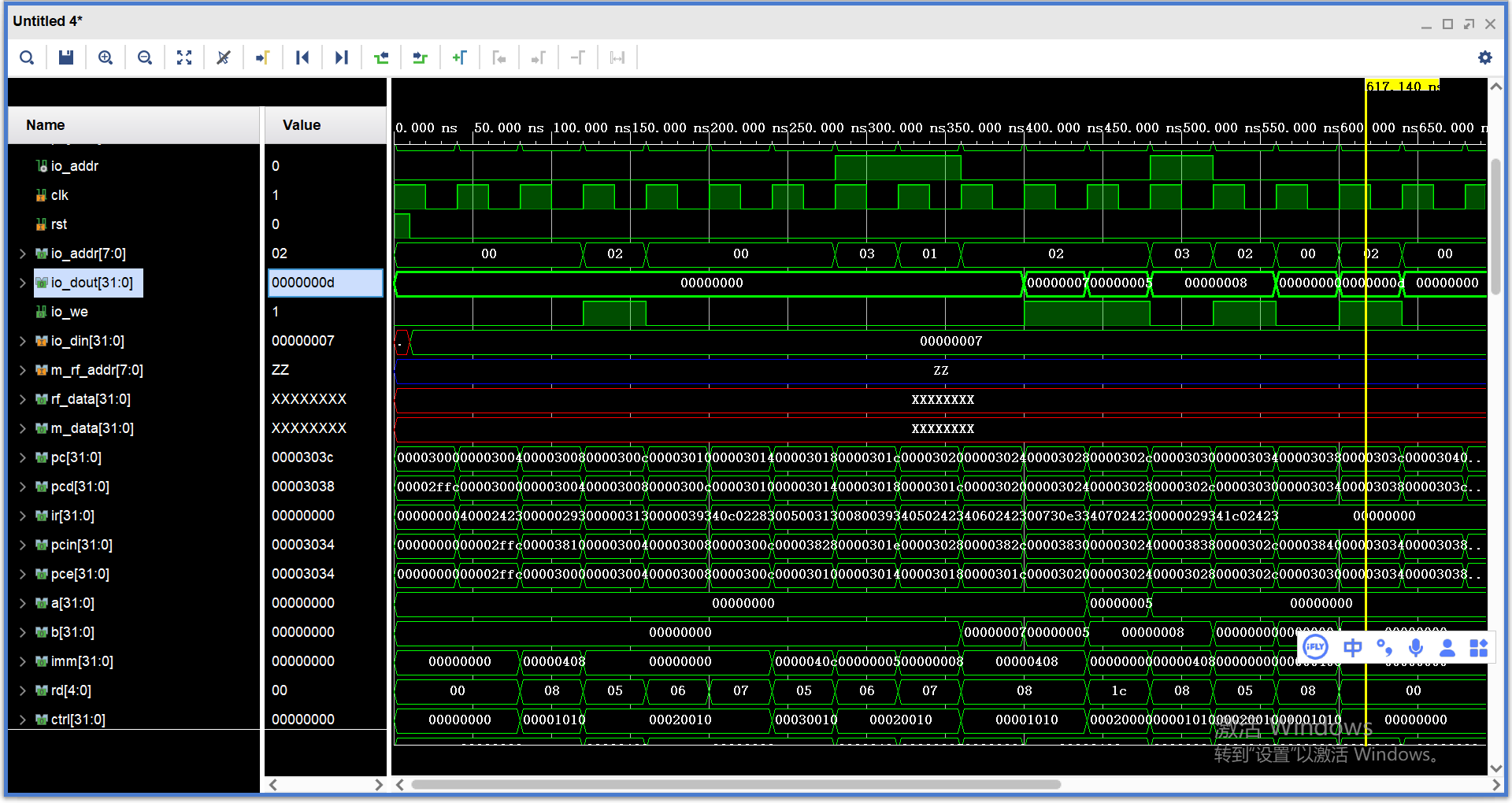
);

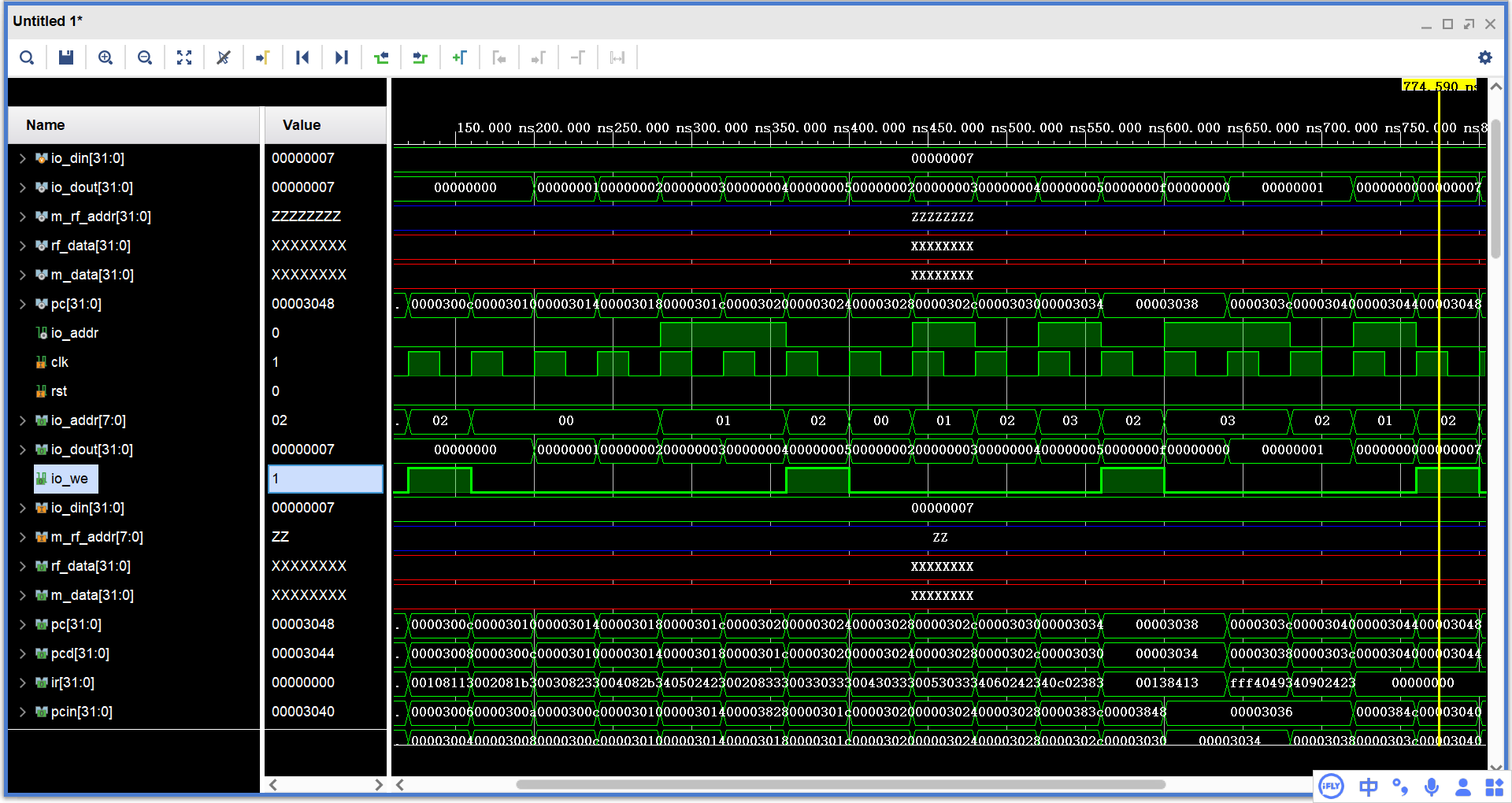
endmodule

**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\_ldex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。