**计算机组成原理 实验报告**

姓名：杨博涵 学号：PB20000328 实验日期：2022.4.13

**一、实验题目：**

Lab04 单周期CPU

**二、实验源码：**

**汇编：**

.data

n0: .word 1

n1: .word 2

n: .space 30

.text

la s0,n

lw a0,n0

lw a1,n1

li s1,20

loop: add a2,a1,a0

sw a2,0(s0)

mv a0,a1

mv a1,a2

addi s0,s0,4

addi s1,s1,-1

beq s1,x0,a

jal loop

a: jal a

CPU模块：

Decoder：

`define I\_type 3'b000 //ISA：冗余、分类、规整

`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 auipc 7'b0010111

`define sw 7'b0100011

`define br 7'b1100011

`define beq 3'b000

`define blt 3'b100

`define jal 7'b1101111

`define jalr 7'b1100111

`define nop 7'b0

module Decoder

(

input [31:0] pc,

input clk,

input zero,

input less,

input rst,

input [31:0] writeback,

input [7:0] m\_rf\_address,

output [2:0] ALUsel,

output [31:0] ALUscr1,

output [31:0] ALUscr2,

output [31:0] wd,

output reg [1:0] RegScr,

output [31:0] rf\_data,

output reg MemWrite,

output [1:0] jalmux\_sel,

output [31:0] imm,

output [31:0] io\_dout

);

wire [31:0] INSTRUCTION,rd1,rd0;

wire judge;

reg RegWrite,Branch,jal,ALUctr1,ALUctr2,branch\_sel,jalr;

reg [2:0] Imm\_gen;

reg [1:0] ALUop;

instruction\_mem instruction\_mem(.a(pc[9:2]),.spo(INSTRUCTION));

rf Register(.rst(rst),.clk(clk),.ra0(INSTRUCTION[19:15]),.ra1(INSTRUCTION[24:20]),.ra2(m\_rf\_address[4:0]),.wa(INSTRUCTION[11:7]),.we(RegWrite),.rd0(rd0),.rd1(rd1),.rd2(rf\_data),.wd(writeback));

ImmGen ImmGen(Imm\_gen,INSTRUCTION,imm);

mux2 ALUscr2\_mux(rd1,imm,ALUctr2,ALUscr2);

mux2 ALUscr1\_mux(rd0,pc,ALUctr1,ALUscr1);

ALUctr ALUctr(INSTRUCTION,ALUop,ALUsel);

assign judge=branch\_sel==0 ? zero: less;

assign wd=rd1;

assign jalmux\_sel[0]=jal|(Branch&judge);

assign jalmux\_sel[1]=jalr;

assign io\_dout=rd1;

//control unit

always@(\*)

begin

case(INSTRUCTION[6:0])

`R\_type: begin jal=0;Branch=0;Imm\_gen=3'b111;RegScr=2'b00;ALUop=2'b10;MemWrite=0;ALUctr1=0;ALUctr2=0;RegWrite=1; branch\_sel=0;jalr=0;end

`addi: begin jal=0;Branch=0;Imm\_gen=`I\_type;RegScr=2'b00;ALUop=2'b00;MemWrite=0;ALUctr1=0;ALUctr2=1;RegWrite=1; branch\_sel=0;jalr=0;end

`lw:begin jal=0;Branch=0;Imm\_gen=`I\_type;RegScr=2'b01;ALUop=2'b00;MemWrite=0;ALUctr1=0;ALUctr2=1;RegWrite=1; branch\_sel=0;jalr=0;end

`auipc:begin jal=0;Branch=0;Imm\_gen=`U\_type;RegScr=2'b00;ALUop=2'b00;MemWrite=0;ALUctr1=1;ALUctr2=1;RegWrite=1; branch\_sel=0;jalr=0;end

`sw:begin jal=0;Branch=0;Imm\_gen=`S\_type;RegScr=2'b00;ALUop=2'b00;MemWrite=1;ALUctr1=0;ALUctr2=1;RegWrite=0; branch\_sel=0;jalr=0;end

`br:begin

case(INSTRUCTION[14:12])

`beq:begin jal=0;Branch=1;Imm\_gen=`SB\_type;RegScr=2'b00;ALUop=2'b01;MemWrite=0;ALUctr1=0;ALUctr2=0;RegWrite=0; branch\_sel=0;jalr=0;end

default:begin jal=0;Branch=1;Imm\_gen=`SB\_type;RegScr=2'b00;ALUop=2'b01;MemWrite=0;ALUctr1=0;ALUctr2=0;RegWrite=0; branch\_sel=1;jalr=0;end

endcase

end

`jal:begin jal=1;Branch=0;Imm\_gen=`UJ\_type;RegScr=2'b10;ALUop=2'b00;MemWrite=0;ALUctr1=0;ALUctr2=1;RegWrite=1; branch\_sel=1;jalr=0;end

`jalr:begin jal=1;Branch=0;Imm\_gen=`I\_type;RegScr=2'b10;ALUop=2'b00;MemWrite=0;ALUctr1=0;ALUctr2=1;RegWrite=1; branch\_sel=1;jalr=1;end

`nop:begin jal=0;Branch=0;Imm\_gen=3'b111;RegScr=2'b00;ALUop=2'b00;MemWrite=0;ALUctr1=0;ALUctr2=0;RegWrite=0; branch\_sel=0;jalr=0;end

default:begin jal=0;Branch=0;Imm\_gen=3'b111;RegScr=2'b00;ALUop=2'b00;MemWrite=0;ALUctr1=0;ALUctr2=0;RegWrite=0; branch\_sel=0;jalr=0;end

endcase

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

module ALUctr

(

input [31:0] INSTRUCTION,

input [1:0] ALUop,

output reg [2:0] ALUsel

);

always@(\*)

begin

case(ALUop)

2'b00: ALUsel=3'b000;

2'b01: ALUsel=3'b001;

2'b10: begin

case(INSTRUCTION[31:25])

7'b0000000:ALUsel=3'b000;

7'b0100000:ALUsel=3'b001;

default: ALUsel=0;

endcase

end

default: ALUsel=0;

endcase

end

endmodule

Execute：

module Execute

(

input [2:0] ALUsel,

input [31:0] ALUscr1,

input [31:0] ALUscr2,

input [31:0] wd,

input [1:0] RegScr,

input MemWrite,

input clk,

input [31:0] io\_din,

input [31:0] in0,

input [7:0] m\_rf\_address,

output zero,

output [31:0] m\_data,

output less,

output [31:0] writeback,

output [7:0] io\_address,

output io\_we,

output [31:0] ALUresult

);

wire [31:0] rd,mem\_sel;

wire we;

alu alu(ALUscr1,ALUscr2,ALUsel,ALUresult,zero,less);

data\_mem data\_mem(.a(ALUresult[9:2]),.d(wd),.dpra(m\_rf\_address),.clk(clk),.we(we),.dpo(m\_data),.spo(rd)); //假设全为字寻址

mux2 iomux(rd,io\_din,ALUresult[10],mem\_sel);

mux3 regsrc(ALUresult,mem\_sel,in0,RegScr,writeback);

assign io\_address=ALUresult[7:0];

assign we=MemWrite&~ALUresult[10];

assign io\_we=ALUresult[10]&MemWrite;

endmodule

pc：

module PC

(

input rst,

input clk,

input [31:0] in2,

input [1:0] jalmux\_sel,

input [31:0] jalimm,

output [31:0] pc,

output [31:0] in0

);

reg [31:0] PCreg;

wire [31:0] in1,next\_pc;

assign pc=PCreg;

assign in0=PCreg+4;

assign in1=PCreg+(jalimm<<1);

mux3 jalmux(in0,in1,in2,jalmux\_sel,next\_pc);

always@(posedge clk or posedge rst)

begin

if(rst==1) PCreg<=32'h3000;

else PCreg<=next\_pc;

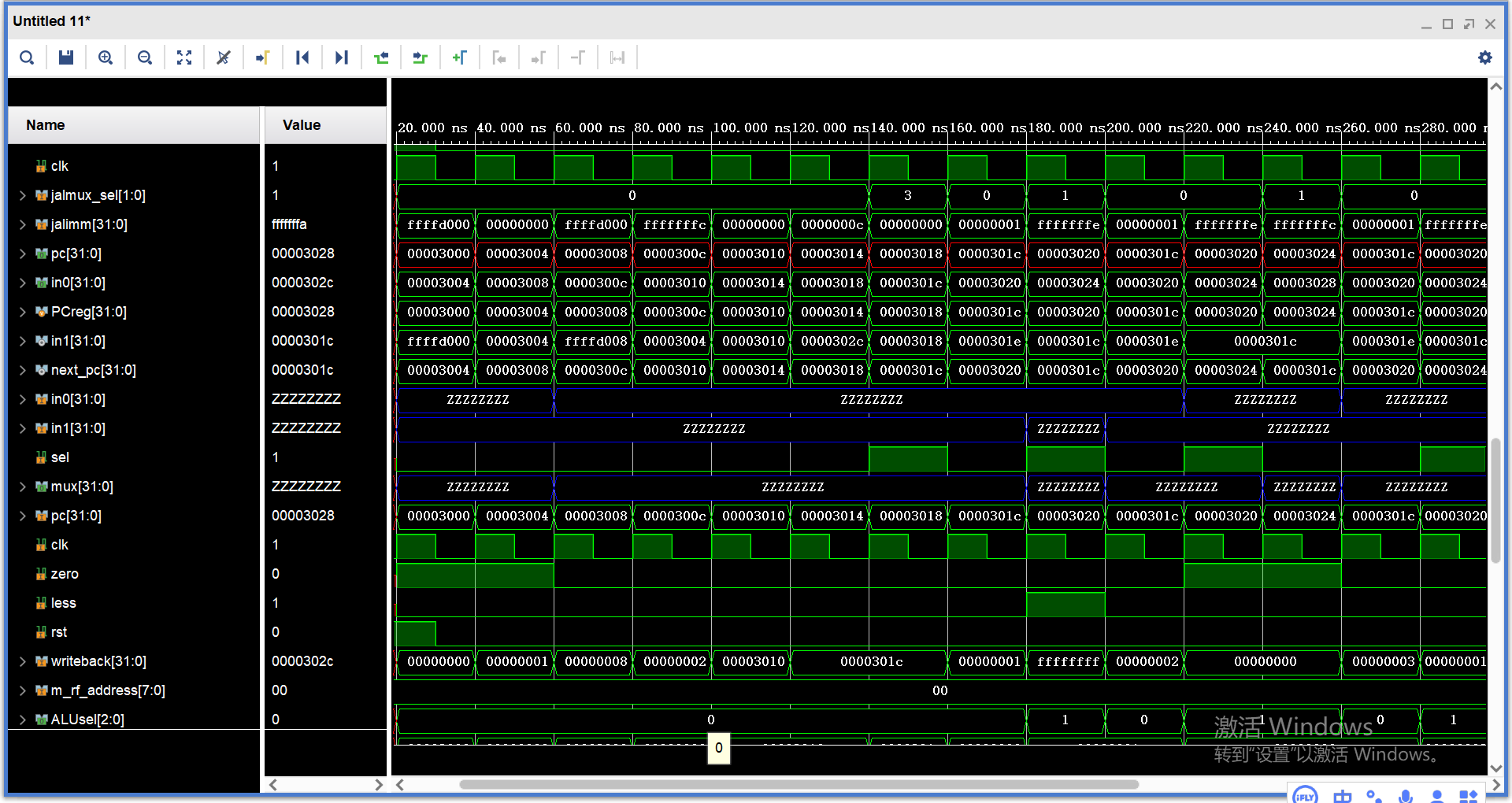
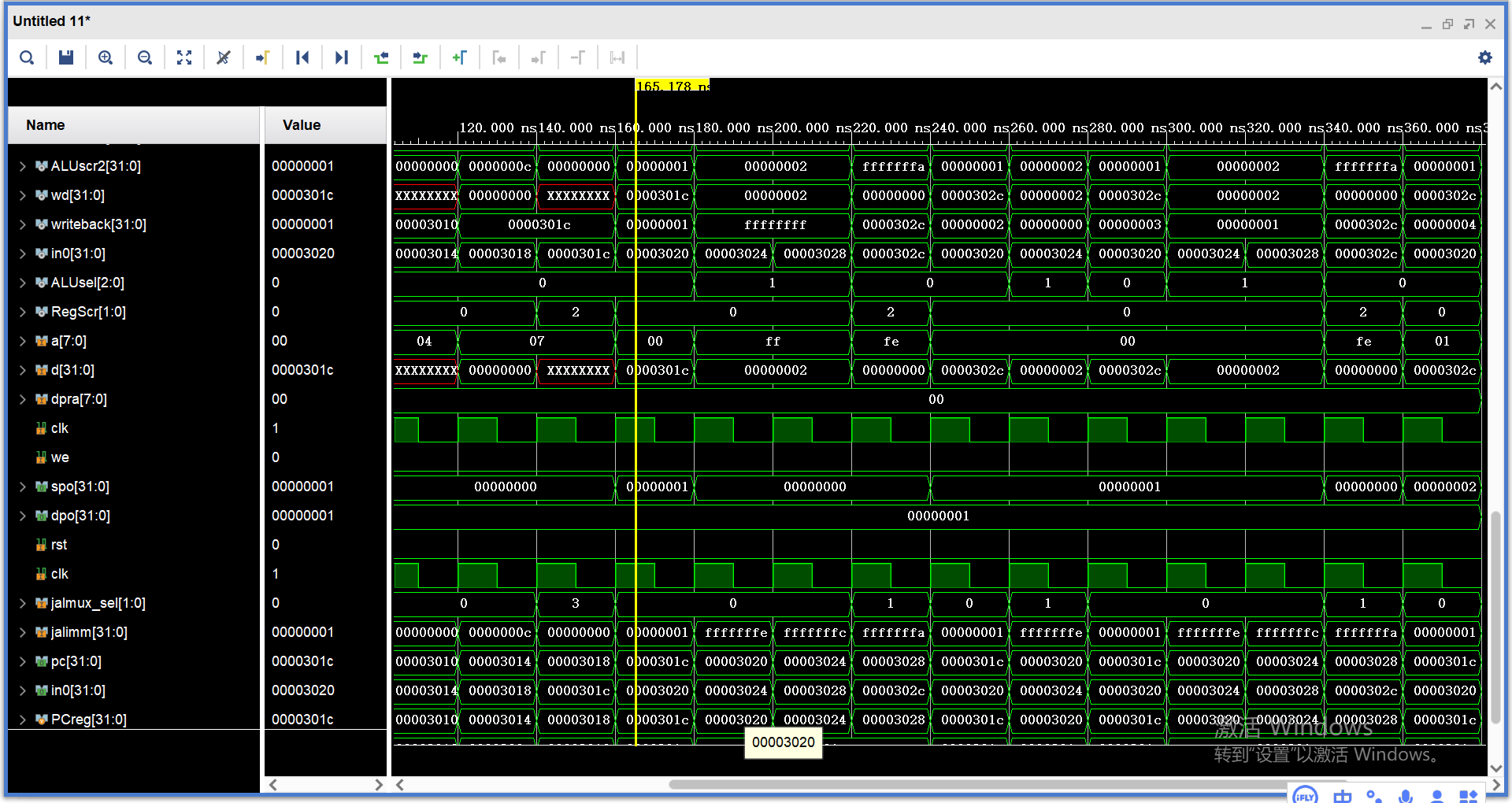
end

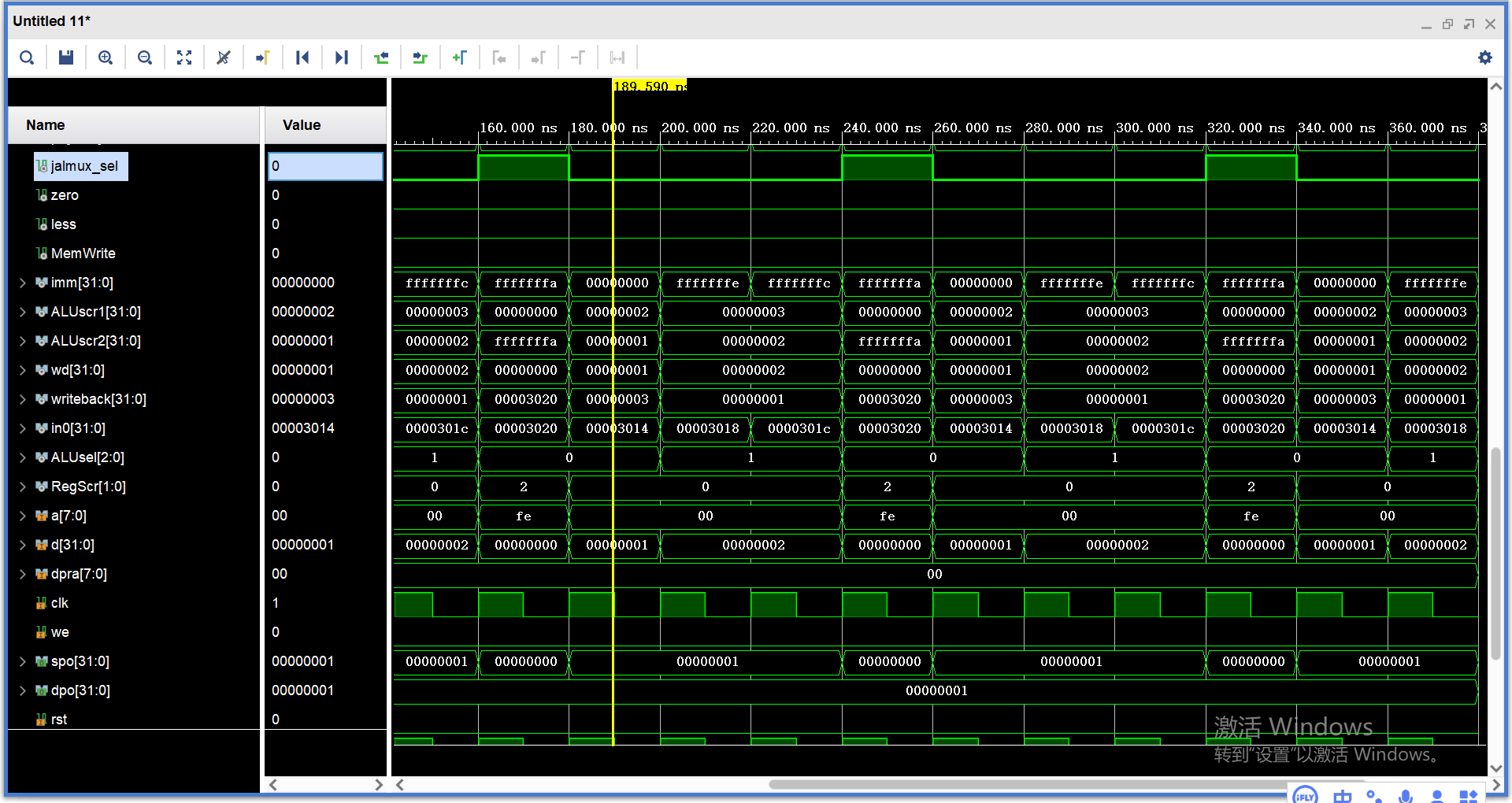
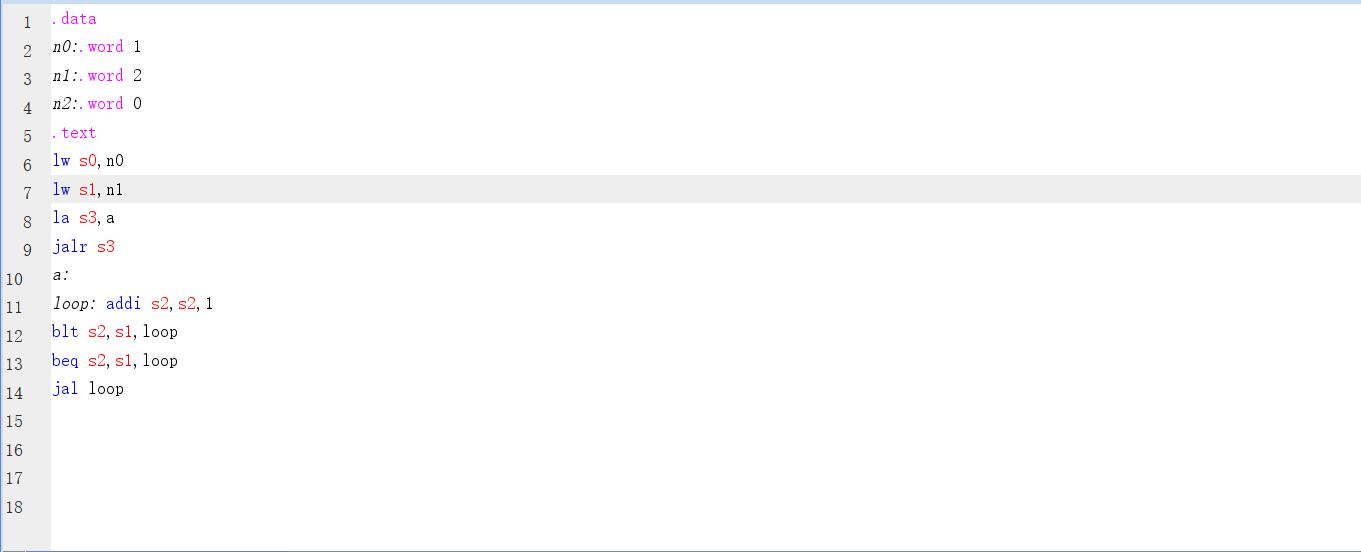
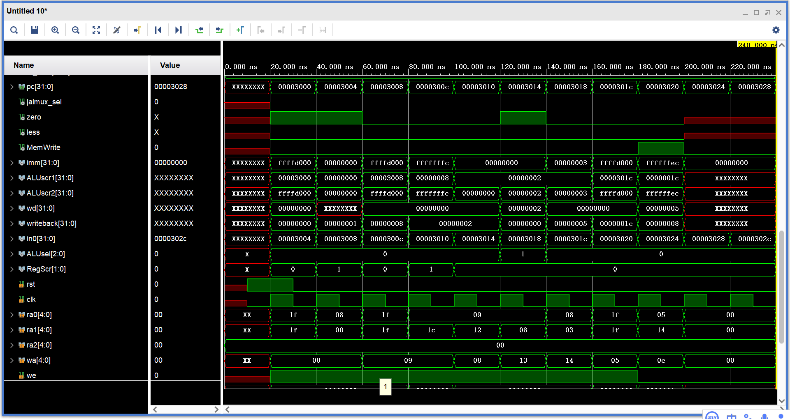
endmodule

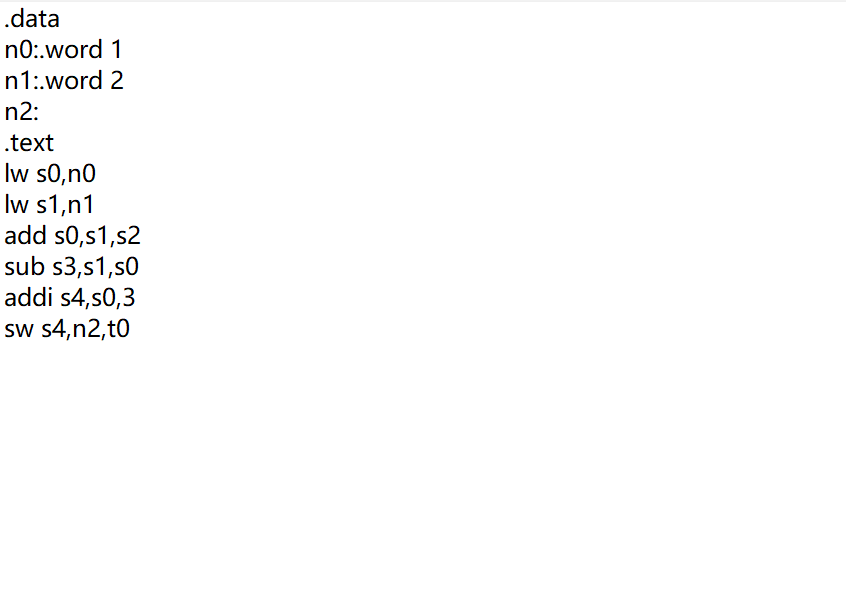
**三、实验思路：**

照着数据通路连线即可，注意控制器的信号赋值即可，宏定义各种指令状态。并且需要在给出的数据通路中加入一些MUX以达到跳转的效果。

注意接口连线和位宽即可，排查高阻态。

 下图为一些在测试指令时的仿真结果：





**四、实验记录：**

Mainv1.bit:

1. PC在run=0时仍在变化----CPU直接输入clk
2. data存储器只在1处非零，值为2----
3. 寄存器无值---

Update：CPU改为cpu\_clk; pc改为pc[9:2]

Mainv2.bit:

1.上述2,3仍未解决

2.auipc未支持

3.addi不正确，

4.一开始2是正常的，后来归零---指令问题

Update：mux2接口缺少; auipc支持。

Main\_test.v

1. lw正常
2. auipc错误
3. Imm错误
4. alu接口又漏接。。。。
5. 新增stop（nop）指令
6. Jalr已支持
7. Blt 和beq的opcode似乎一样（fun3区别）

8.x0不定（真reg不用always组合逻辑）

已完成指令：auipc、add、sub、addi、sw、lw、jal、nop、jalr、blt、beq

（vivado的小于0似乎有点问题）

Main\_finished:

去除时钟与门，所有系统正常。