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),.r
d0(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;ALUct
r1=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;ALUc
tr1=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;ALUc
tr1=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;ALUc
tr1=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;ALUc
tr1=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;ALU
ctr1=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;ALU
ctr1=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;ALU
ctr1=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;ALUc
tr1=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;ALUct
r1=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;ALUct
r1=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],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
```

```
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);
```

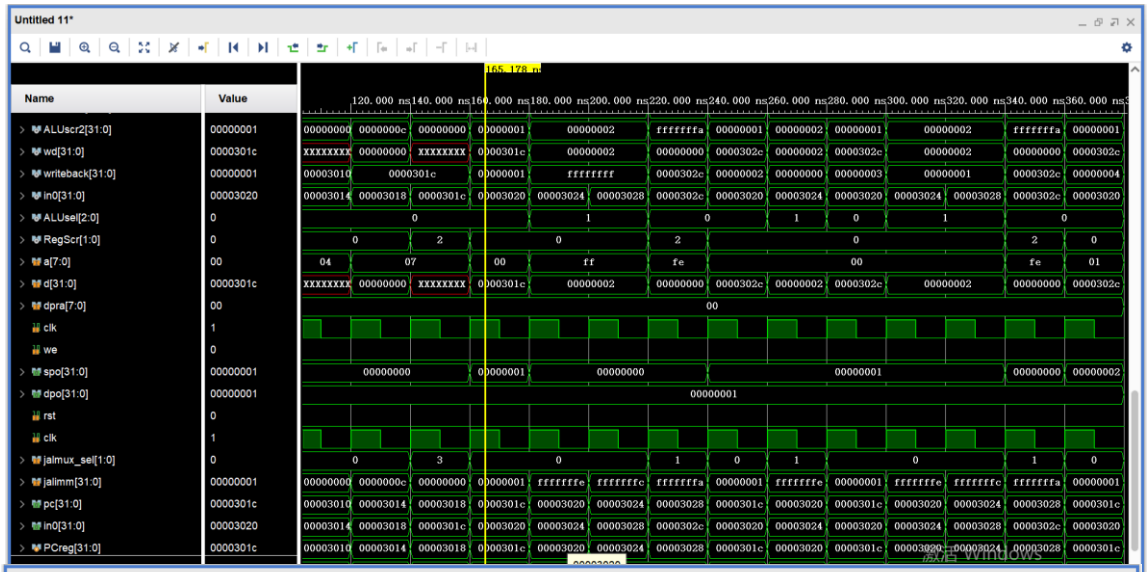
```
mux3 jalmux(in0,in1,in2,jalmux_sel,next_pc);


always@(posedge clk or posedge rst)
begin
        if(rst==1) PCreg<=32'h3000;
        else PCreg<=next_pc;
end


endmodule
```

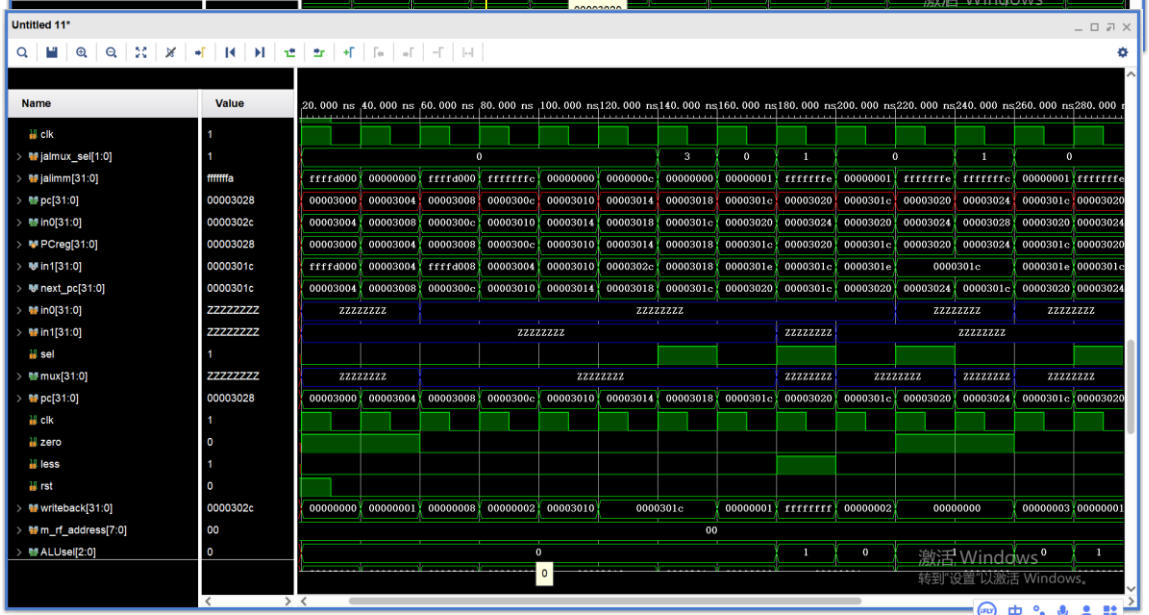
三、实验思路：

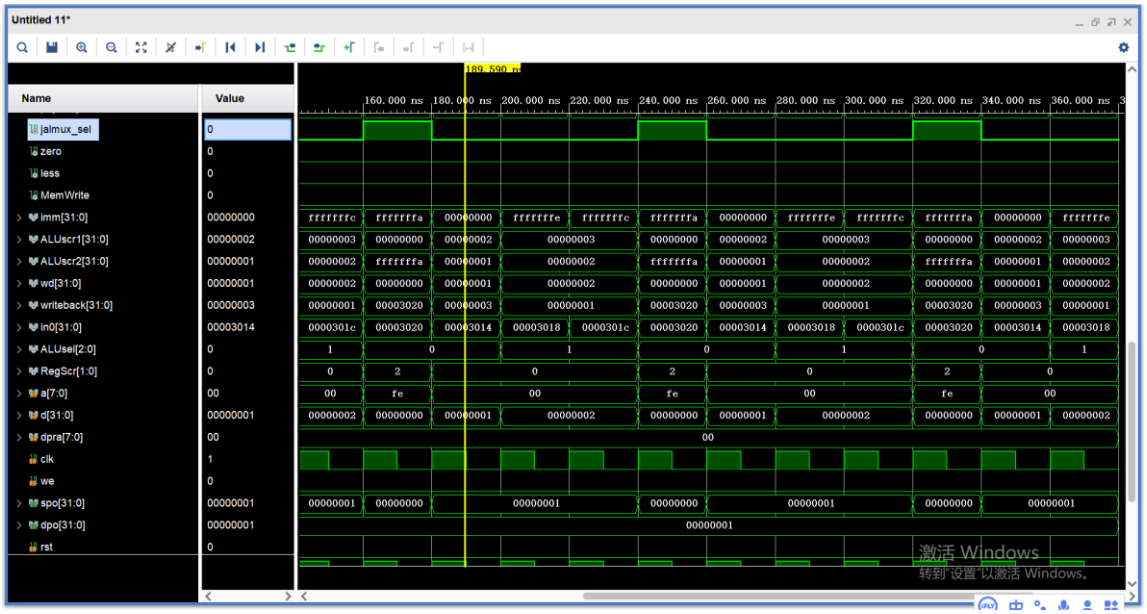
照着数据通路连线即可，注意控制器的信号赋值即可，宏定义各种指令状态。并且需要在给出的数据通路中加入一些 MUX 以达到跳转的效果。

注意接口连线和位宽即可，排查高阻态。

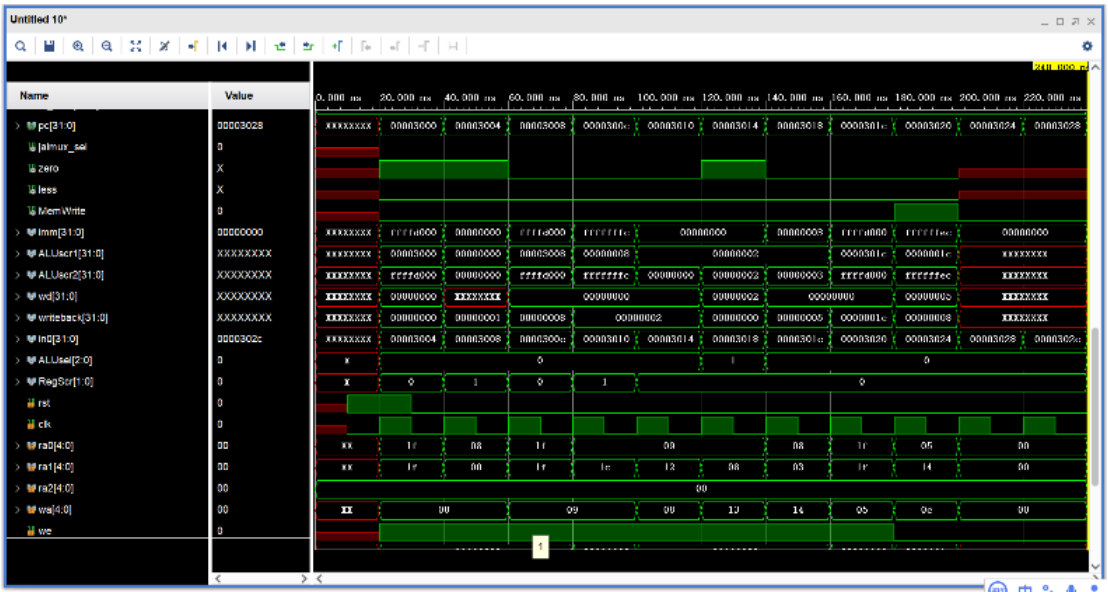
下图为一些在测试指令时的仿真结果：








```
.data
n0:.word 1
n1:.word 2
n2:.word 0
.text
lw s0,n0
lw s1,n1
la s3,a
jalr s3
a:
loop: addi s2,s2,1
blt s2,s1,loop
beq s2,s1,loop
jal loop
```



```
.data
n0:.word 1
n1:.word 2
n2:
.text
lw s0,n0
lw s1,n1
add s0,s1,s2
sub s3,s1,s0
addi s4,s0,3
sw s4,n2,t0
```

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。