

计算机组成原理实验报告

姓名：付斯珂 学号：PB21061224 实验日期：2023-6-14

一、实验题目：

Lab6 综合设计

二、实验目的与内容：

- 理解计算机硬件系统的组成结构和工作原理
- 掌握软硬件综合系统的设计和调试方法
- 在lab5的基础上实现流水线CPU的指令集拓展以及选做内容

三、实验平台：

Vivado, FPGAOL, vscode

四、逻辑设计

数据通路：即为助教给出的datapath.png

核心代码：

1. 分支指令

实现了bne、bge、bltu、bgeu

```

case (br_type)
  3'b000: br=0;
  3'b001: //beq
begin
  if(op1==op2) br=1;
  else br=0;
end
3'b010: //blt //有符号数比较
begin
  if(~(op1[31]^op2[31]))//均为正或均为负
    br=(op1<op2?1'b1:1'b0); //和无符号数比较相同
  else if(op1[31] && !op2[31])
    br=1;
  else
    br=0;
end
3'b011://bne 不等时分支
begin
  if(op1!=op2) br=1;
  else br=0;
end
3'b100://bgeu 无符号大于等于时分支
begin
  if(op1>=op2) br=1;
  else br=0;
end
3'b101://bltu 无符号小于时分支
begin
  if(op1<op2) br=1;
  else br=0;
end
3'b110://bge 有符号大于等于时分支
begin
  if(~(op1[31]^op2[31]))//均为正或均为负
    br=(op1>=op2?1'b1:1'b0); //和无符号数比较相同
  else if(op1[31] && !op2[31])
    br=0;
  else
    br=1;
end
default: br=0;
endcase

```

2. 算数与逻辑指令和条件指令

拓展指令为sll、slli、srl、srli、sra、srai、sub、xor、xori、or、ori、and、andi

以下为ctrl.v的部分代码

对I-type指令，通过inst[14:12]区分具体指令，并选择相应的alu功能，输出相应的控制信号。

```

7'b0010011: begin
    if(inst[14:12]==3'b000)//addi
    begin
        jal=0;
        jalr=0;
        lb=0;lh=0;lw=0;lbu=0;lhu=0;
        sw_sh_sb=3'b000;
        br_type=3'b00;//not branch
        if(inst[11:7]==3'b000) wb_en=1'b0;//write back
        else wb_en=1'b1;
        wb_sel=2'b0;//alu out
        alu_op1_sel=2'b00;//inst[19:15]
        alu_op2_sel=1'b1;//imm
        alu_ctrl=4'b0000;//+
        imm_type=3'b001;//I-type
        mem_we=1'b0;//do not write mem

        if(inst[19:15]==3'b000) rf_re0=1'b0;
        else rf_re0=1'b1;
        rf_re1=1'b0;

        float=0;
        rf_read_sel_id=0;
        rf_wb_sel_id=0;
    end
    else if(inst[14:12]==3'b111)//andi
    begin
        jal=0;
        jalr=0;
        lb=0;lh=0;lw=0;lbu=0;lhu=0;
        sw_sh_sb=3'b000;
        br_type=3'b00;//not branch
        if(inst[11:7]==3'b000) wb_en=1'b0;//write back
        else wb_en=1'b1;
        wb_sel=2'b0;//alu out
        alu_op1_sel=2'b00;//inst[19:15]
        alu_op2_sel=1'b1;//imm
        alu_ctrl=4'b0101;//&
        imm_type=3'b001;//I-type
        mem_we=1'b0;//do not write mem

        if(inst[19:15]==3'b000) rf_re0=1'b0;
        else rf_re0=1'b1;
        rf_re1=1'b0;

        float=0;
        rf_read_sel_id=0;
        rf_wb_sel_id=0;
    end

```

3. 访存指令

6条：lb、lh、lw、lbu、lhu、sb、sh

lb, lh, lw, lbu, lhu, 和sw_sh_sb信号为1时，分别说明指令为相应的访存指令。

对load指令：

dm_dout_wb_sel.v模块处理load指令访存后写回寄存器的值。若lb有效，只使用dm_dout_wb的低8位的符号拓展作为写回值。lh则为低16位，等等。

```
always @(*) begin
    case({lw_wb,lh_wb,lb_wb,lbu_wb,lhu_wb})
        5'b10000:dm_dout_wb_load=dm_dout_wb;
        5'b01000:begin
            if(dm_dout_wb[15]) dm_dout_wb_load={16'hFFFF,dm_dout_wb[15:0]};
            else dm_dout_wb_load={16'b0,dm_dout_wb[15:0]};
        end
        5'b00100:begin
            if(dm_dout_wb[7]) dm_dout_wb_load={28'hFFFFFFF,dm_dout_wb[7:0]};
            else dm_dout_wb_load={28'b0,dm_dout_wb[7:0]};
        end
        5'b00010: dm_dout_wb_load={28'b0,dm_dout_wb[7:0]};
        5'b00001: dm_dout_wb_load={16'b0,dm_dout_wb[15:0]};
        default: dm_dout_wb_load=dm_dout_wb;
    endcase
end
```

对store指令：

将data mem由原本的一个ip核32位拆为4个ip核，每个各8位。每个代表data mem的一字中的一字节。

它们输入地址相同，输出数据分别赋给dm_din的不同字节。

根据sw_sh_sb和we的值决定写使能，从而决定写入哪个data mem。比如sb只写入Data_mem0。

```

Data_mem0 Data_mem0 (
    .a(dm_addr[9:2]),           // input wire [7 : 0] a
    .d(dm_din[7:0]),           // input wire [7 : 0] d
    .dpra(mem_check_addr[7:0]), // input wire [7 : 0] dpra
    .clk(clk),                // input wire clk
    .we(dm_we),               // input wire we
    .spo(dm_dout[7:0]),         // output wire [7 : 0] spo
    .dpo(mem_check_data[7:0])   // output wire [7 : 0] dpo
);

Data_mem1 Data_mem1 (
    .a(dm_addr[9:2]),           // input wire [7 : 0] a
    .d(dm_din[15:8]),          // input wire [7 : 0] d
    .dpra(mem_check_addr[7:0]), // input wire [7 : 0] dpra
    .clk(clk),                // input wire clk
    .we(dm_we & (~sw_sh_sb[0])), // input wire we
    .spo(dm_dout[15:8]),        // output wire [7 : 0] spo
    .dpo(mem_check_data[15:8])  // output wire [7 : 0] dpo
);

Data_mem2 Data_mem2 (
    .a(dm_addr[9:2]),           // input wire [7 : 0] a
    .d(dm_din[23:16]),          // input wire [7 : 0] d
    .dpra(mem_check_addr[7:0]), // input wire [7 : 0] dpra
    .clk(clk),                // input wire clk
    .we(dm_we & sw_sh_sb[2]),   // input wire we
    .spo(dm_dout[23:16]),        // output wire [7 : 0] spo
    .dpo(mem_check_data[23:16]) // output wire [7 : 0] dpo
);

Data_mem3 Data_mem3 (
    .a(dm_addr[9:2]),           // input wire [7 : 0] a
    .d(dm_din[31:24]),          // input wire [7 : 0] d
    .dpra(mem_check_addr[7:0]), // input wire [7 : 0] dpra
    .clk(clk),                // input wire clk
    .we(dm_we & sw_sh_sb[2]),   // input wire we
    .spo(dm_dout[31:24]),        // output wire [7 : 0] spo
    .dpo(mem_check_data[31:24]) // output wire [7 : 0] dpo
);

```

[选做] 4. 浮点指令的实现

- **浮点寄存器**

增加浮点寄存器fRF.v。浮点寄存器的实现基本与整数寄存器相同，不同之处在于x0寄存器可被写入。

- **浮点运算器**

增加浮点运算器falu.v。

实现指令 fadd.s：浮点数相加， fsub.s：浮点数相减， fcvt.s.w：整数转换浮点

数, `fcvt.w.s` : 浮点数转化为整数。

考虑由于浮点数位数有限, 整数和浮点数转换, 以及浮点数计算结果均需要舍去或进位。舍入方式由浮点指令中不同的舍入模式(rounding mode)决定。`fadd.s`、`fsub.s`为动态舍入(dyn), 本应根据`fcsr`的值实现舍入, 此处默认为`rne`(四舍五入)。

舍入模式说明:

1. 就近舍入Round to nearest, ties to even (`rne`)

四舍五入的结果是最接近无限精确的结果。如果小数部分为0.5, 若整数部分为奇数则进一, 若整数部分为偶数则舍去小数0.5。则结果为偶数(即最低有效位为0)。

2. 朝0舍入Round toward zero (`rtz`)

正数或负数均直接舍去小数部分, 这样结果最接近0。

3. 朝正无穷舍入Round up (toward $+\infty$) (`rup`)

正数多余位不全为0进位1, 正数多余位全为0直接截尾; 负数直接截尾。

4. 朝负无穷舍入Round down (toward $-\infty$) (`rdn`)

正数直接截尾; 负数多余位全为0直接截尾, 负数多余位不全为0进位1。

具体浮点指令计算根据浮点数格式实现:

```

always @(*) begin
  case (func)
    4'b1011: begin//fadd //rne
      exp1 = a[30:23];
      exp2 = b[30:23];
      frac1 = a[22:0];
      frac2 = b[22:0];
      width1=exp1-127;
      width2=exp2-127;

      if(exp1>=exp2) begin
        i=exp2;
        tmp={8'd0,1'b1,frac2}>>(exp1-exp2); //指数相同时的有效数字部分

        if(a[31]==b[31]) begin//shift right
          tmp1=tmp+{8'd0,1'b1,frac1}; //add
          if(tmp1[24]) begin
            if(tmp1[0]==0) frac=tmp1[23:1]; //直接舍去
            else begin
              if(tmp1[1]==0) frac=tmp1[23:1]; //直接舍去
              else frac=tmp1[23:1]+1; //进一
            end
            exp=exp1+1;
          end
          else begin frac=tmp1[22:0];exp=exp1;end
          y={a[31],exp,frac};
        end
        else if(a[31]&(~b[31])) begin//shift left
          tmp1=-tmp+{8'd0,1'b1,frac1}; //sub
          j=0;
          // while(!tmp1[i] && i>=0) i=i-1; //i位置为第一个1
          for(i=0;i<=23;i=i+1) if(tmp1[i] && i>j) j=i;
          for(i=22;i>=0;i=i-1)
            if(j+i-23>=0) frac[i]=tmp1[j+i-23];
            else frac[i]=0; //无舍入
          exp=exp1-(23-j);
          y={1'b1,exp,frac};
        end
        else begin//shift left
          tmp1=-tmp+{8'd0,1'b1,frac1}; //sub
          j=0;
          for(i=0;i<=23;i=i+1) if(tmp1[i] && i>j) j=i;
          for(i=22;i>=0;i=i-1)
            if(j+i-23>=0) frac[i]=tmp1[j+i-23];
            else frac[i]=0; //无舍入
          exp=exp1-(23-j);
          y={1'b0,exp,frac};
        end
      end
    end
    else begin

```

```

//exp1<exp2
//省略
end
end

4'b1100: begin//fsub //rne
//省略
end

4'b1101: begin//fcvt.w.s
// 把寄存器 f[rs1]中的单精度浮点数转化为 32 位二进制补码表示的整数，再写入 x[rd]中
    exp = a[30:23];
    frac = a[22:0];
    width=a[30:23]-127;//尾数被截取的宽度，可能大于23
    tmp=0;

    if(rm==3'b000 || rm==3'b111) begin//Round to Nearest, ties to Even ,rne
        if(width>=23) begin
            tmp=0;
            tmp=frac<<(width-23);
            tmp[width]=1;
            y=(a[31]?-tmp:tmp);
        end
        else if( width<23 && frac[22-width]==0) begin//舍去
            j=23-width;
            tmp=0;
            for(i=0;i<width && i<32;i=i+1) begin
                tmp[i]=frac[j+i];
            end
            tmp[i]=1;
            y=(a[31]?-tmp:tmp);
        end
        else begin//进位
            if((frac<<(width+1))==0) begin//小数部分==0.5
                if(frac[23-width]) begin//奇数，进一
                    j=23-width;
                    tmp=0;
                    for(i=0;i<width && i<32;i=i+1) begin
                        tmp[i]=frac[j];
                        j=j+1;
                    end
                    tmp[i]=1;
                    y=(a[31]?-(tmp+1):tmp+1);
                end
                else begin//偶数， 截去
                    j=23-width;
                    tmp=0;
                    for(i=0;i<width && i<32;i=i+1) begin
                        tmp[i]=frac[j];
                        j=j+1;
                    end
                    tmp[i]=1;
                end
            end
        end
    end
end

```

```

        y=(a[31]?-tmp:tmp);
    end
end
else//小数部分>0.5, 进一
    j=23-width;
    tmp=0;
    for(i=0;i<width && i<32;i=i+1) begin
        tmp[i]=frac[j];
        j=j+1;
    end
    tmp[i]=1;
    y=(a[31]?-(tmp+1):tmp+1);
end
end

else if(rm==3'b001) begin//Round towards Zero
// 直接截尾
if(width>=23) begin
    tmp=0;
    tmp=frac<<(width-23);
    tmp[width]=1;
    y=(a[31]?-tmp:tmp);
end
else begin
    j=23-width;
    tmp=0;
    for(i=0;i<width && i<32;i=i+1) begin
        tmp[i]=frac[j];
        j=j+1;
    end
    tmp[i]=1;
    y=(a[31]?-tmp:tmp);
end
end

else if(rm==3'b010) begin//Round Down (towards -∞)
if(width>=23) begin
    tmp=0;
    tmp=frac<<(width-23);
    tmp[width]=1;
    y=(a[31]?-tmp:tmp);
end
else begin
    if(!a[31]) begin//正数直接截尾
        j=23-width;
        y=0;
        for(i=0;i<width && i<32;i=i+1) begin
            y[i]=frac[j];
            j=j+1;
        end
        y[i]=1;
    end
end

```

```

        end
    else begin
        //负数多余位全为0直接截尾
        if((frac<<width) == 0) begin
            j=23-width;
            tmp=0;
            for(i=0;i<width && i<32;i=i+1) begin
                tmp[i]=frac[j];
                j=j+1;
            end
            tmp[i]=1;
            y=-tmp;
        end
        //负数多余位不全为0进位1
        else begin
            j=23-width;
            tmp=0;
            for(i=0;i<width && i<32;i=i+1) begin
                tmp[i]=frac[j];
                j=j+1;
            end
            tmp[i]=1;
            y=-(tmp+1);
        end
    end
end

else if(rm==3'b011) begin//Round Up (towards +∞)
    //省略
end

else begin
    tmp=0;
    y=0;//随便
end
end

4'b1110: begin//fcvt.s.w
// 把寄存器 x[rs1]中的 32 位二进制补码表示的整数转化为单精度浮点数，再写入 f[rd]中
y =32'd0;
tmp=0;
tmp1=0;
tmp2=0;
if (a == 32'b0) begin // 特殊情况：输入为零
    y = 32'h00000000;
end
else begin
    // 提取符号位
    y[31] = a[31];
    // 指数部分

```

```

tmp =$signed(a);
if (tmp[31]) tmp1 = -tmp; // 取绝对值
else tmp1=tmp;
j=0;
for(i=0;i<32;i=i+1) begin//为1的最高位exp
    if(tmp1[i] && i>j) j=i;
end
exp=j;
y[30:23] = exp + 23'd127;

// 尾数部分
if(rm==3'b000 || rm==3'b111) begin//Round to Nearest, ties to Even ,rne
    if(j<=23||tmp1[j-23]==0) begin//直接舍去
        for (i = 22; i >= 0; i = i - 1)
            if((j+i-23)>=0) y[i] = tmp1[j+i-23];
            else y[i]=0;
    end
    else if(tmp1<<(31-exp+23+1)==0) begin//0.5
        if(tmp1[j-22]) begin//奇数
            for (i = 22; i >= 0; i = i - 1)
                tmp2[i] = tmp1[j+i-23];
            tmp2[31:23]=0;
            y[22:0]=tmp2+1;//进一
        end
        else begin//偶数
            for (i = 22; i >= 0; i = i - 1)
                y[i] = tmp1[j+i-23];//直接舍去
        end
    end
    else begin//进一
        for (i = 22; i >= 0; i = i - 1)
            tmp2[i] = tmp1[j+i-23];
        tmp2[31:23]=0;
        y[22:0]=tmp2+1;
    end
end
else if(rm==3'b001) begin//Round towards Zero
    for (i = 22; i >= 0; i = i - 1)
        if(j+i-23>=0) y[i] = tmp1[j+i-23];//直接舍去
        else y[i]=0;
end
else if(rm==3'b010) begin//Round Down (towards -∞)
    //正数直接截尾//负数多余位全为0直接截尾
    if(a>0 || (a<0 && tmp1<<(32-exp)==0)) begin
        for (i = 22; i >= 0; i = i - 1)
            if(j+i-23>=0) y[i] = tmp1[j+i-23];
            else y[i]=0;
    end
    else begin//负数多余位不全为0进位1
        for (i = 22; i >= 0; i = i - 1)
            if(j+i-23>=0) tmp2[i] = tmp1[j+i-23];

```

```

        else tmp2[i]=0;
        tmp2[31:23]=0;
        y[22:0]=tmp2+1;
    end
end
else if(rm==3'b011) begin//Round Up (towards +∞)
    //省略
end
end
default :begin
    y=32'b0;
    exp=0;
    tmp=0;
end
endcase
end

```

• 浮点指令产生的数据选择

- 读寄存器选择：rf_rd0_raw_id根据rf_read_sel_id选择从浮点还是整型寄存器中数据。

```

assign rf_rd0_raw_id=(rf_read_sel_id?rf_rd0_raw_id_float:rf_rd0_raw_id_no_float),
      rf_rd1_raw_id=(rf_read_sel_id?rf_rd1_raw_id_float:rf_rd1_raw_id_no_float);

```

- ALU计算结果选择：两种ALU同时在EX阶段计算，根据float_ex信号（即EX阶段是否为浮点指令）选择alu_ans_ex的ALU来源。

```

assign alu_ans_ex=(float_ex?alu_ans_ex_float:alu_ans_ex_no_float);

```

- 写寄存器选择：

整型寄存器的写使能例化为 .we(rf_we_wb & (~rf_wb_sel_wb))，浮点寄存器的写使能例化为 .we(rf_we_wb & rf_wb_sel_wb)，这样rf_wb_sel_wb==0时写入整型寄存器，否则写入浮点寄存器。rf_wb_sel由ctrl模块产生。

• 浮点指令的冒险

本次实验浮点指令只有fadd.s、fsub.s、fcvt.s.w、fcvt.w.s四条。因此只增加了数据冒险。

hazard.v前递部分增加如 rf_read_sel_ex==rf_wb_sel_mem 的判断条件，即ex阶段读的寄存器和mem或wb阶段写的寄存器是否同为浮点寄存器或整型寄存器，这样才会产生冒险。否则像写浮点寄存器和读整型寄存器并不会产生冒险。

```

// forwarding
always @(*) begin //for rd0
    // mem
    if(rf_we_mem==1 && rf_re0_ex!=0 && rf_ra0_ex==rf_wa_mem && rf_wd_sel_mem!=2'b10
    && rf_read_sel_ex==rf_wb_sel_mem)begin
        rf_rd0_fe=1;
        if(rf_wd_sel_mem==2'b00) rf_rd0_fd=alu_ans_mem;
        else if(rf_wd_sel_mem==2'b01) rf_rd0_fd=pc_add4_mem;
        else rf_rd0_fd=imm_mem;
    end
    //.....

```

同理load-use也增加判断条件 `rf_read_sel_ex==rf_wb_sel_i`。

```

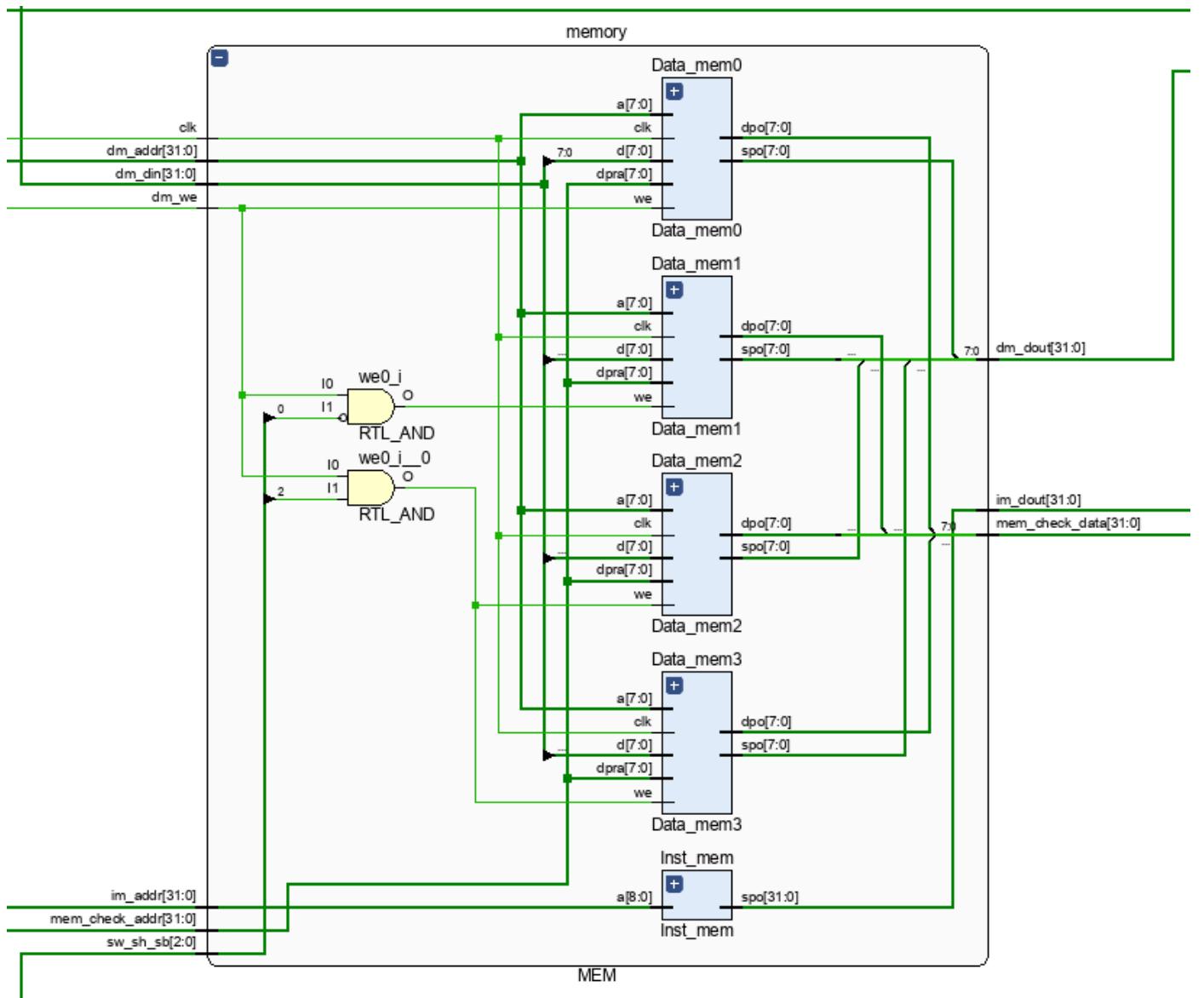
// load use
always @(*) begin
    // 假定 WB_SEL_MEM 代表寄存器堆写回选择器选择 MEM_dout 的结果
    if (rf_we_ex && rf_re0_id && rf_wa_ex == rf_ra0_id && rf_wd_sel_ex == 2'b10 && rf_read_:
    || rf_we_ex && rf_re1_id && rf_wa_ex == rf_ra1_id && rf_wd_sel_ex == 2'b10 && rf_read_se
begin
    stall_if = 1'b1;
    stall_id = 1'b1;
    stall_ex = 1'b0;
end

```

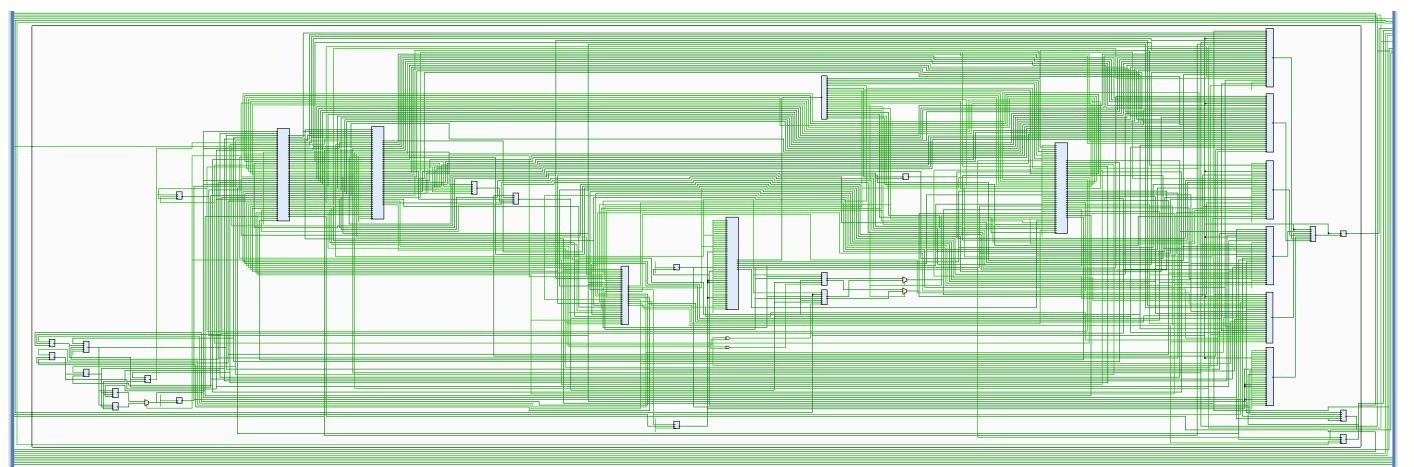
电路设计与分析

RTL电路图：

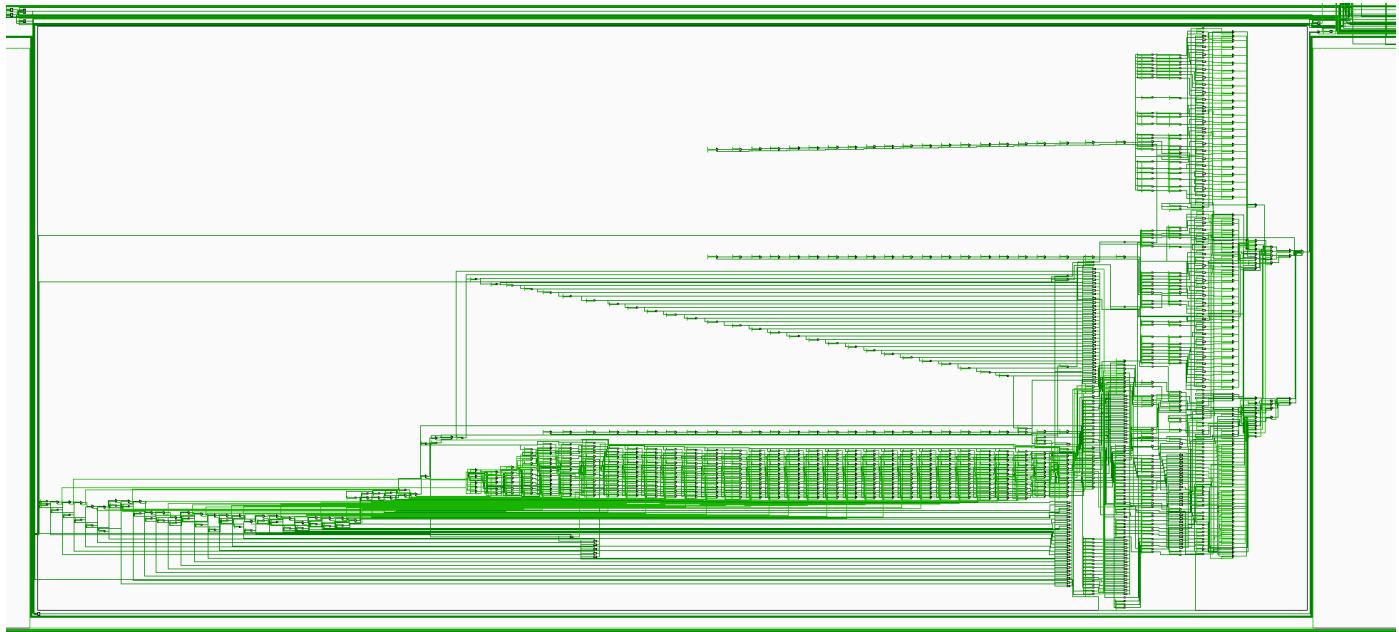
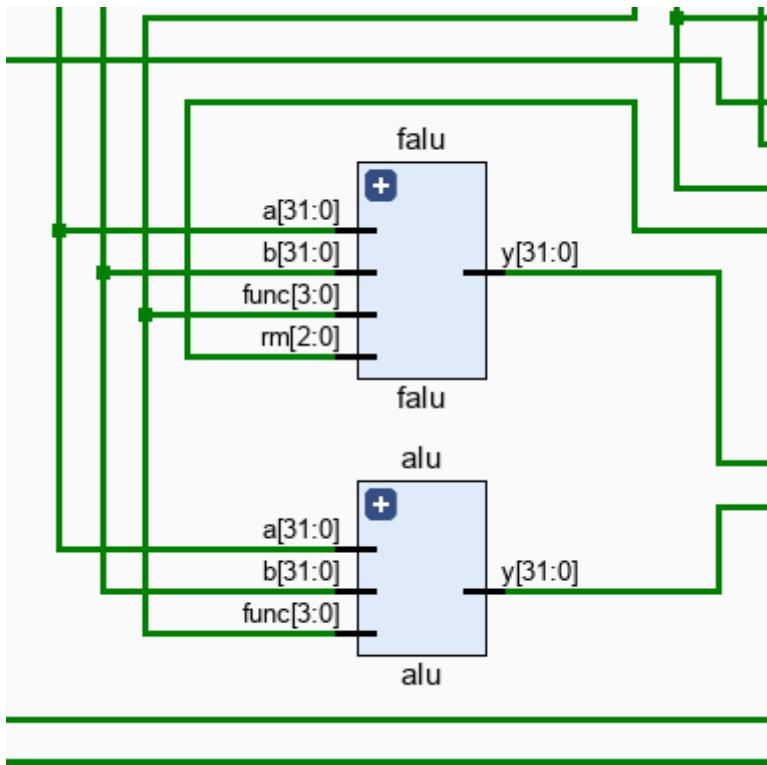
1. MEM



2. CPU



3. falu 浮点ALU

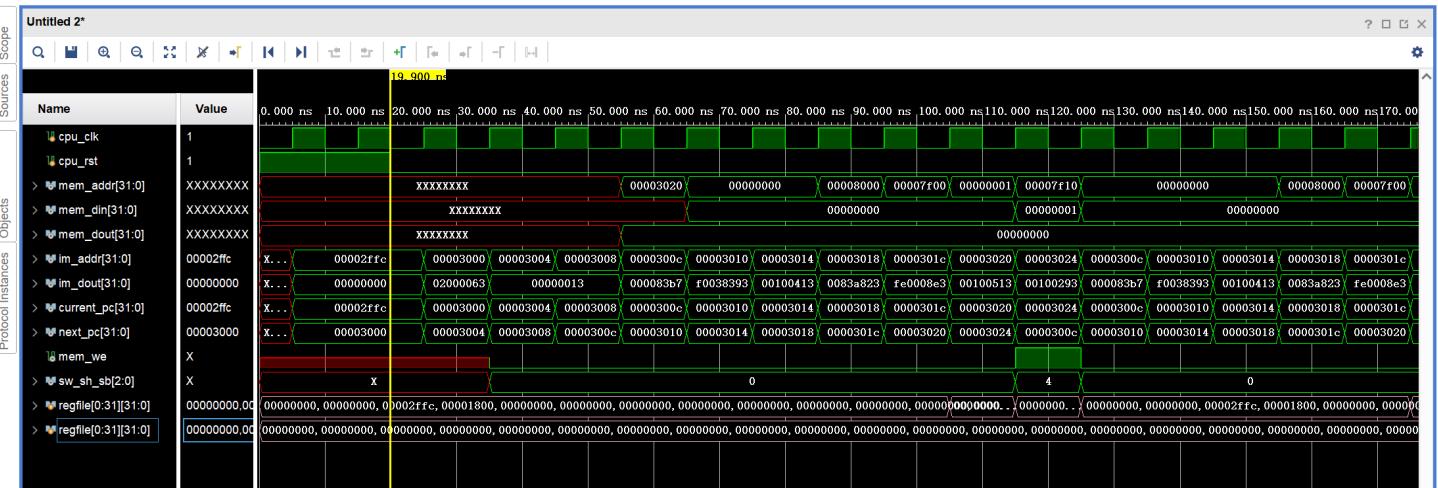


仿真结果与分析

仿真所用coe文件即为提供的float.asm测试程序。

最下面两行标为粉色的信号，第一个为rf(整型寄存器堆)的各个寄存器值，第二个为rf(浮点型寄存器堆)的各个寄存器值。

im_addr为指令地址，im_dout为指令值。

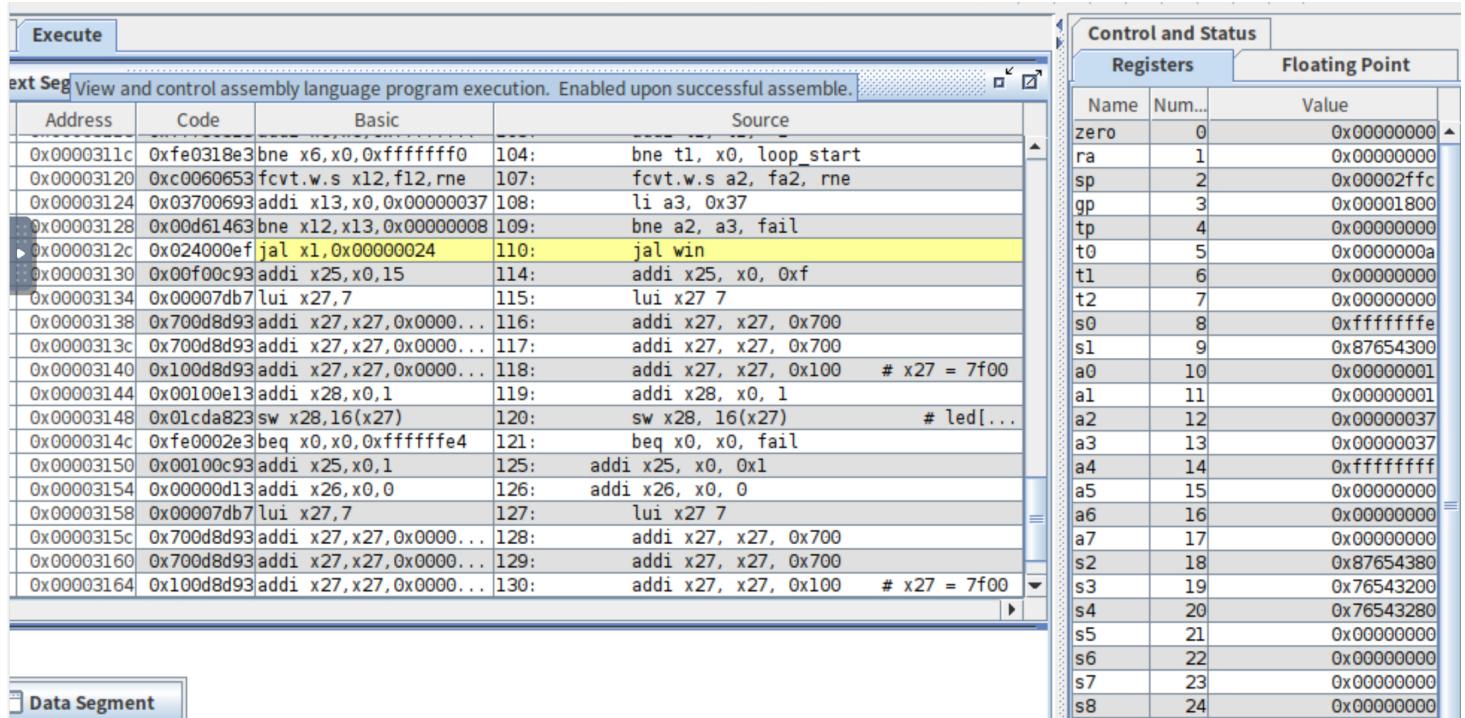


查看运行到不同指令时两种regfile的值，和RARS软件的对比，相同则仿真通过。

如运行到 jal win 指令

RARS:

整型寄存器堆的值

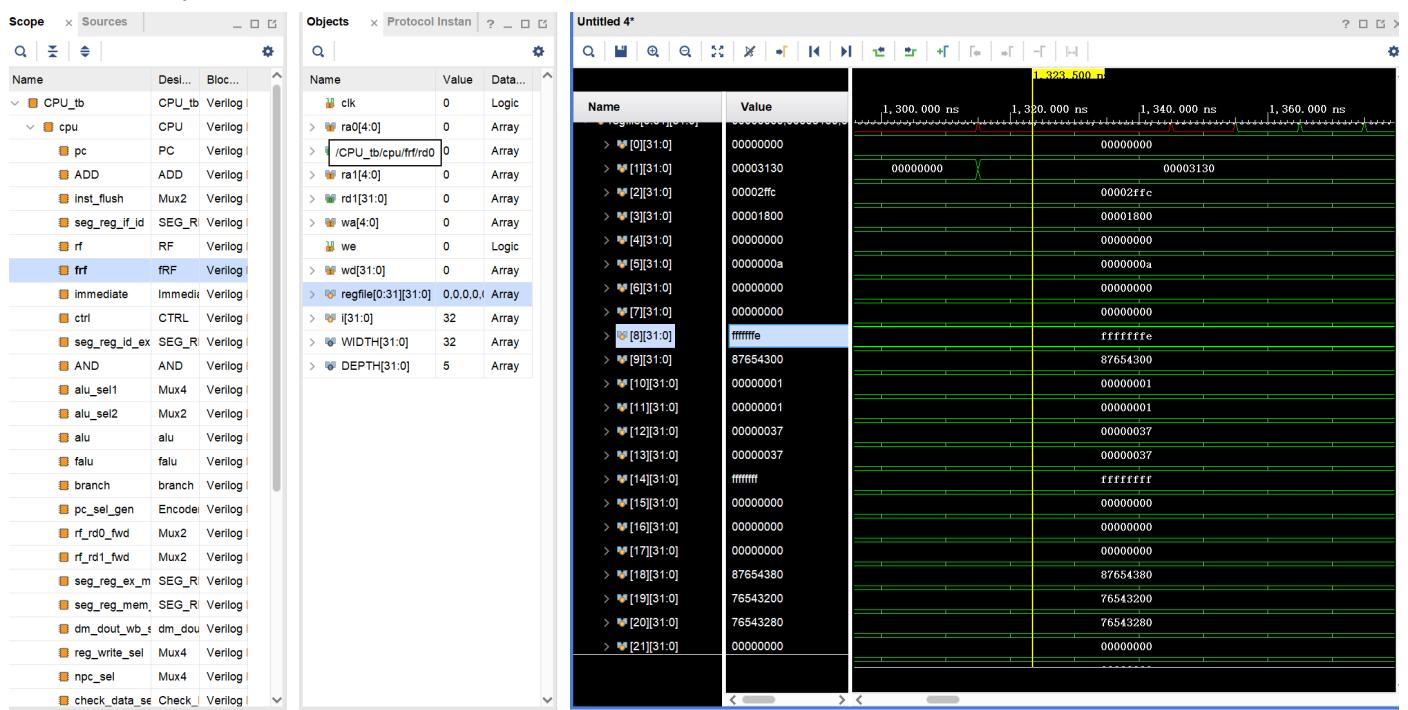


浮点型寄存器堆的值

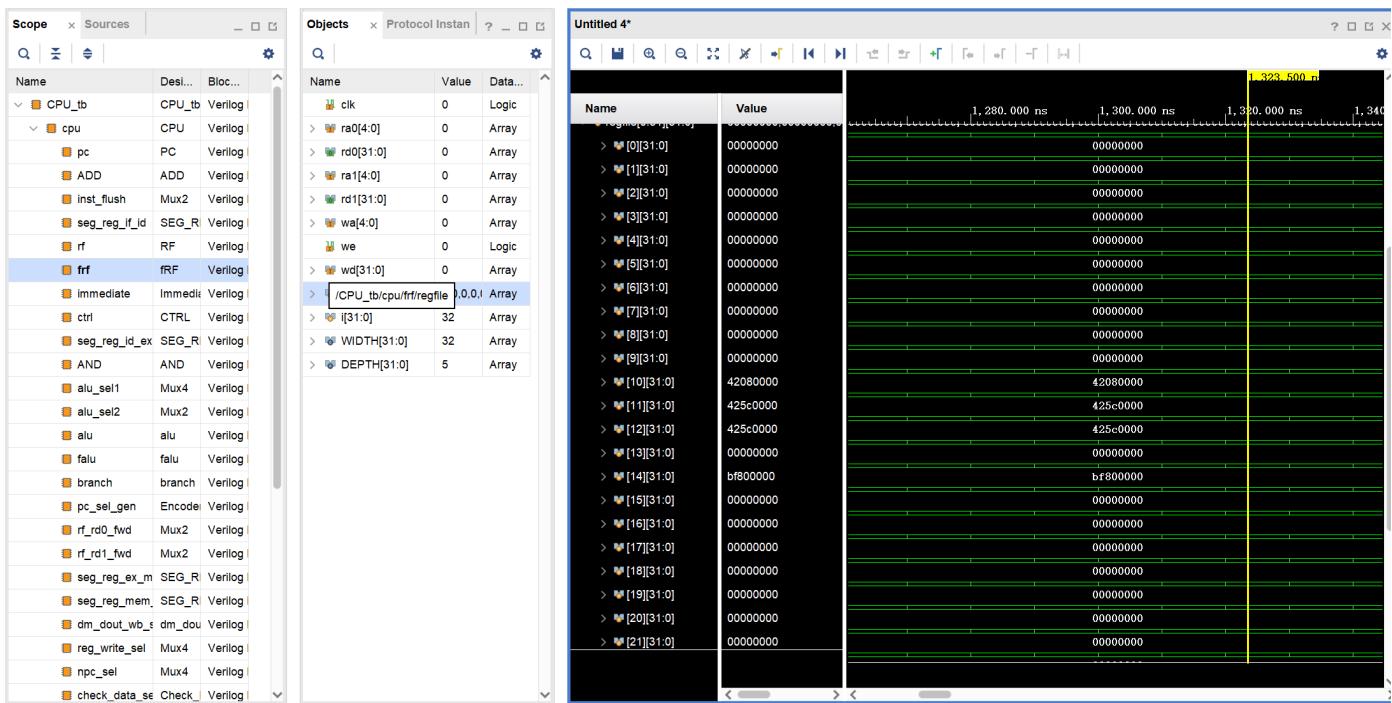
Text Segment				Control and Status
Address	Code	Basic	Source	Registers
0x00000311c	0xfe0318e3	bne x6,x0,0xffffffff	104: bne t1, x0, loop_start	ft0 0x0000000000000000
0x000003120	0xc0060653	fcvt.w.s x12,f12,rne	107: fcvt.w.s a2, fa2, rne	ft1 10x0000000000000000
0x000003124	0x03700693	addi x13,x0,0x00000037	108: li a3, 0x37	ft2 20x0000000000000000
0x000003128	0x00d61463	bne x12,x13,0x00000008	109: bne a2, a3, fail	ft3 30x0000000000000000
0x00000312c	0x024000ef	jal x1,0x00000024	110: jal win	ft4 40x0000000000000000
0x000003130	0x00f00c93	addi x25,x0,15	114: addi x25, x0, 0xf	ft5 50x0000000000000000
0x000003134	0x00007db7	lui x27,7	115: lui x27 7	ft6 60x0000000000000000
0x000003138	0x700d8d93	addi x27,x27,0x0000...	116: addi x27, x27, 0x700	ft7 70x0000000000000000
0x00000313c	0x700d8d93	addi x27,x27,0x0000...	117: addi x27, x27, 0x700	fs0 80x0000000000000000
0x000003140	0x100d8d93	addi x27,x27,0x0000...	118: addi x27, x27, 0x100 # x27 = 7f00	fs1 90x0000000000000000
0x000003144	0x00100e13	addi x28,x0,1	119: addi x28, x0, 1	fa0 10 0x42080000
0x000003148	0x01cda823	sw x28,16(x27)	120: sw x28, 16(x27) # led[...	fa1 11 0x425c0000
0x00000314c	0xfe0002e3	beq x0,x0,0xffffffe4	121: beq x0, x0, fail	fa2 12 0x425c0000
0x000003150	0x00100c93	addi x25,x0,1	125: addi x25, x0, 0x1	fa3 13 0x00000000
0x000003154	0x000000d13	addi x26,x0,0	126: addi x26, x0, 0	fa4 14 0xbfb80000
0x000003158	0x00007db7	lui x27,7	127: lui x27 7	fa5 15 0x0000000000000000
0x00000315c	0x700d8d93	addi x27,x27,0x0000...	128: addi x27, x27, 0x700	fa6 16 0x0000000000000000
0x000003160	0x700d8d93	addi x27,x27,0x0000...	129: addi x27, x27, 0x700	fa7 17 0x0000000000000000
0x000003164	0x100d8d93	addi x27,x27,0x0000...	130: addi x27, x27, 0x100 # x27 = 7f00	fs2 18 0x0000000000000000

仿真结果：

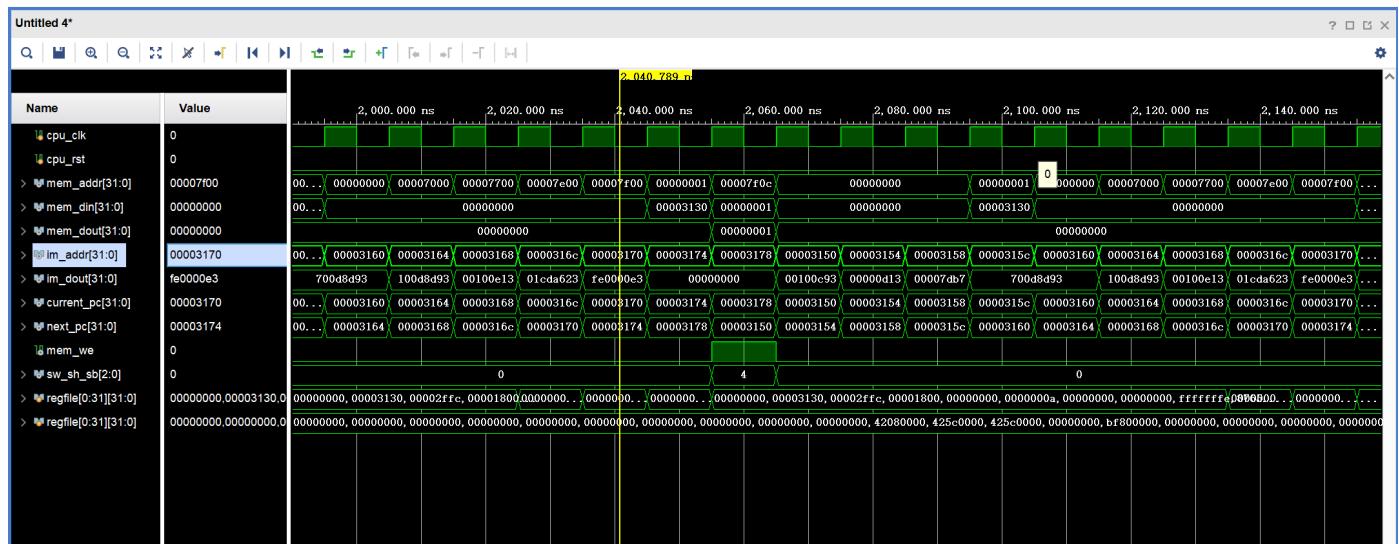
1. 整型寄存器堆



2. 浮点型寄存器堆



查看最后指令循环情况，可知指令进入了win循环，im_addr为0x3150~3170。



0x0000314c	0xfe0002e3	beq x0, x0, 0xfffffffffe4	121:	beq x0, x0, fail
0x00003150	0x00100c93	addi x25, x0, 1	125:	addi x25, x0, 0x1
0x00003154	0x00000d13	addi x26, x0, 0	126:	addi x26, x0, 0
0x00003158	0x00007db7	lui x27, 7	127:	lui x27 7
0x0000315c	0x700d8d93	addi x27, x27, 0x0000...	128:	addi x27, x27, 0x700
0x00003160	0x700d8d93	addi x27, x27, 0x0000...	129:	addi x27, x27, 0x700
0x00003164	0x100d8d93	addi x27, x27, 0x0000...	130:	addi x27, x27, 0x100 # x27 = 7f00
0x00003168	0x00100e13	addi x28, x0, 1	131:	addi x28, x0, 1
0x0000316c	0x01cda623	sw x28, 12(x27)	132:	sw x28 12(x27) # led[...]
0x00003170	0xfe0000e3	beq x0, x0, 0xfffffffffe0	133:	beq x0, x0, win

测试结果与分析

上板结果已检查，led0亮起。

总结

更了解了RISC-V指令集以及流水线CPU。

最后一个实验哈哈