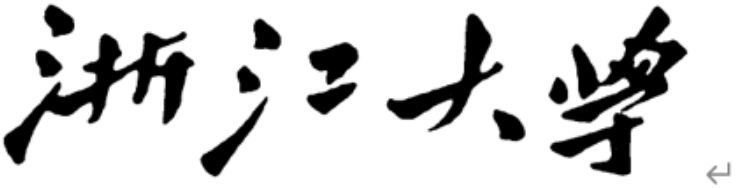


课程名称:	**计算机组成与设计**
姓 名:	杨吉祥
学 院:	计算机科学与技术学院
系:	竺可桢学院图灵班
专 业:	计算机科学与技术
学 号:	3230106222
指导教师:	刘海风

浙江大学实验报告

课程名称：计算机组成与设计　　实验类型： 综合

实验项目名称：单周期CPU

学生姓名： 杨吉祥 专业： 计算机科学与技术 学号： 3230106222

报告日期：11.27

一、操作方法与实验步骤

1.1 SCPU_ctrl.v

以下是本次实验用到的所有控制信号

1. `ALUop[3:0]`:控制ALU的运算
2. `ALUsrc`:控制ALU的B的值是立即数还是read data2
3. `Branch[1:0]`:Branch[1]为1表示指令是分支跳转指令,反之则不是;因为分支跳转指令都是比较大小决定是否跳转,所以我们可以通过ALU_zero的结果来判断是否跳转,所以Branch[0]的值设为发生跳转时ALU_zero的值
4. `Jump[1:0]`:Jump=2'b01表示jal,Jump=2'b10表示jalr,Jump=2'b0表示不是无条件跳转指令
5. `Memread`:1表示要从内存中读取数据,0则不用
6. `MemtoReg[2:0]`:写入寄存器的值,0表示值来源于ALU,1表示值来源于数据内存,2表示值为PC+4,3表示值为imm,4表示值为PC+imm,5表示值来源于csr寄存器堆
7. `RegWrite`:通用寄存器堆的写使能
8. `load_type[2:0]`:表示不同的load类型指令,load指令按读取的数据位数分为3种,然后又分为符号位扩展和无符号扩展,所以用load[2]表示是否为符号位扩展,load[1:0]为0表示不是load指令,1表示取一个字节,2表示取半字,3表示取一个字
9. `store_type[1:0]`:与load指令类似,store指令按存储的数据位数也能分为3种,0表示不是store指令,1表示存一个字节,2表示存半字,3表示存一个字
10. `is_auipc`:指令是否为auipc
11. `is_lui`:指令是否为lui
12. `ecall`:指令是否为ecall
13. `mret`:指令是否为mret
14. `illegal_inst`:指令是否为非法指令
15. `is_csr`:指令是否为csr类型的指令
16. `mstatus`:mstatus为0表示正常,为1表示发生中断或异常
17. `mtvec`:储存trap处理程序的首条指令地址
18. `mcause`:进入trap处理程序的原因,mcause最高位为1表示发生硬件中断,最高位为0,最低位为1表示ecall指令导致的异常,最低位为0,下一位为1表示非法指令导致的异常
19. `mtval`:储存导致发生异常的非法指令
20. `mepc`:存储 trap 触发时将要执行的指令地址,在 `mret` 时作为返回地址

```
`timescale 1ns/1ps
`define CODE{OPcode,Fun3,Fun7}
```

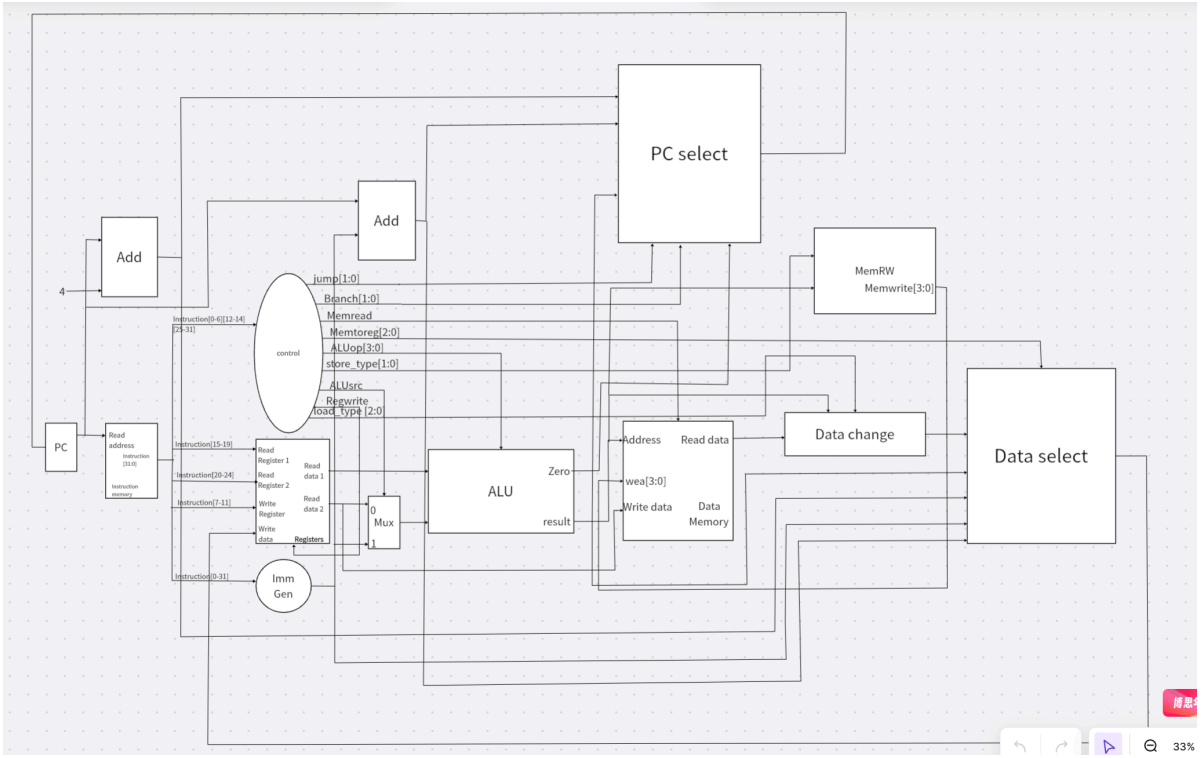
```
`define
SCPU_ctrl{ALUop,ALUsrc,Branch,Jump,MemRead,MemtoReg,RegWrite,load_type,store_type
,is_auipc,is_lui,ecall,mret,illegal_inst,is_csr}

module SCPU_ctrl(
    input wire [6:0]OPcode,
    input wire [2:0]Fun3,
    input wire [6:0]Fun7,
    output reg [3:0]ALUop,
    output reg ALUsrc,
    output reg [1:0]Branch,//Branch[1]:branch,Branch[0]:ALU_zero
    output reg [1:0]Jump,//1:jal,2:jalr
    output reg MemRead,
    output reg
[2:0]MemtoReg,//0:alu,1:mem,2:pc+4,3:immediate,4:pc+immediate,5:csr
    output reg RegWrite,
    output reg
[2:0]load_type,//load[2]:0:unsigned,1:signed;load[1:0]:0:none,1:byte,2:half,3:wor
d
    output reg [1:0]store_type,//0:none,1:byte,2:half,3:word
    output reg is_auipc,
    output reg is_lui,
    output reg ecall,
    output reg mret,
    output reg illegal_inst,
    output reg is_csr
);

    always @(*) begin
        casez(`CODE)

 17'b0110011_000_0000000:`SCPU_ctrl=25'b00000_0000_0_000_1_000_00_00_0000;//add

 17'b0110011_000_0100000:`SCPU_ctrl=25'b00010_0000_0_000_1_000_00_00_0000;//sub

 17'b0110011_111_0000000:`SCPU_ctrl=25'b10010_0000_0_000_1_000_00_00_0000;//and

 17'b0110011_110_0000000:`SCPU_ctrl=25'b10000_0000_0_000_1_000_00_00_0000;//or

 17'b0110011_100_0000000:`SCPU_ctrl=25'b01010_0000_0_000_1_000_00_00_0000;//xor

 17'b0110011_010_0000000:`SCPU_ctrl=25'b00110_0000_0_000_1_000_00_00_0000;//slt

 17'b0110011_101_0000000:`SCPU_ctrl=25'b01100_0000_0_000_1_000_00_00_0000;//srl

 17'b0110011_001_0000000:`SCPU_ctrl=25'b00100_0000_0_000_1_000_00_00_0000;//sll

 17'b0110011_101_0100000:`SCPU_ctrl=25'b01110_0000_0_000_1_000_00_00_0000;//sra

 17'b0110011_011_0000000:`SCPU_ctrl=25'b01000_0000_0_000_1_000_00_00_0000;//sltu


 17'b0010011_000_???????:`SCPU_ctrl=25'b00001_0000_0_000_1_000_00_00_0000;//addi

 17'b0010011_111_???????:`SCPU_ctrl=25'b10011_0000_0_000_1_000_00_00_0000;//andi
```

```
 17'b0010011_110_???????:`SCPU_ctrl=25'b10001_0000_0_000_1_000_00_00_0000;//ori

 17'b0010011_100_???????:`SCPU_ctrl=25'b01011_0000_0_000_1_000_00_00_0000;//xori

 17'b0010011_101_0000000:`SCPU_ctrl=25'b01101_0000_0_000_1_000_00_00_0000;//srli

 17'b0010011_010_???????:`SCPU_ctrl=25'b00111_0000_0_000_1_000_00_00_0000;//slti

 17'b0010011_001_0000000:`SCPU_ctrl=25'b00101_0000_0_000_1_000_00_00_0000;//slli

 17'b0010011_101_0100000:`SCPU_ctrl=25'b01111_0000_0_000_1_000_00_00_0000;//srai

 17'b0010011_011_???????:`SCPU_ctrl=25'b01001_0000_0_000_1_000_00_00_0000;//sltiu


 17'b0000011_000_???????:`SCPU_ctrl=25'b00001_0000_1_001_1_101_00_00_0000;//lb

 17'b0000011_001_???????:`SCPU_ctrl=25'b00001_0000_1_001_1_110_00_00_0000;//lh

 17'b0000011_010_???????:`SCPU_ctrl=25'b00001_0000_1_001_1_111_00_00_0000;//lw

 17'b0000011_100_???????:`SCPU_ctrl=25'b00001_0000_1_001_1_001_00_00_0000;//lbu

 17'b0000011_101_???????:`SCPU_ctrl=25'b00001_0000_1_001_1_010_00_00_0000;//lhu


 17'b0100011_000_???????:`SCPU_ctrl=25'b00001_0000_0_000_0_000_01_00_0000;//sb

 17'b0100011_001_???????:`SCPU_ctrl=25'b00001_0000_0_000_0_000_10_00_0000;//sh

 17'b0100011_010_???????:`SCPU_ctrl=25'b00001_0000_0_000_0_000_11_00_0000;//sw


 17'b1100011_000_???????:`SCPU_ctrl=25'b00010_1100_0_000_0_000_00_00_0000;//beq

 17'b1100011_001_???????:`SCPU_ctrl=25'b00010_1000_0_000_0_000_00_00_0000;//bne

 17'b1100011_100_???????:`SCPU_ctrl=25'b00110_1000_0_000_0_000_00_00_0000;//blt

 17'b1100011_101_???????:`SCPU_ctrl=25'b00110_1100_0_000_0_000_00_00_0000;//bge

 17'b1100011_110_???????:`SCPU_ctrl=25'b01000_1000_0_000_0_000_00_00_0000;//bltu

 17'b1100011_111_???????:`SCPU_ctrl=25'b01000_1100_0_000_0_000_00_00_0000;//bgeu

            17'b1101111_???
_???????:`SCPU_ctrl=25'b11110_0001_0_010_1_000_00_00_0000;//jal

 17'b1100111_000_???????:`SCPU_ctrl=25'b00001_0010_0_010_1_000_00_00_0000;//jalr
            17'b0110111_???
_???????:`SCPU_ctrl=25'b11110_0000_0_011_1_000_00_01_0000;//lui
            17'b0010111_???
_???????:`SCPU_ctrl=25'b11110_0000_0_100_1_000_00_10_0000;//auipc

 17'b1110011_000_0000000:`SCPU_ctrl=25'b11110_0000_0_000_0_000_00_00_1000;//ecall
```

```
 17'b1110011_000_0011000:`SCPU_ctrl=25'b11110_0000_0_000_0_000_00_00_0100;//mret

 17'b1110011_001_???????:`SCPU_ctrl=25'b11110_0000_0_101_1_000_00_00_0001;//csrrw

 17'b1110011_010_???????:`SCPU_ctrl=25'b11110_0000_0_101_1_000_00_00_0001;//csrrs

 17'b1110011_011_???????:`SCPU_ctrl=25'b11110_0000_0_101_1_000_00_00_0001;//csrrc

 17'b1110011_101_???????:`SCPU_ctrl=25'b11110_0000_0_101_1_000_00_00_0001;//csrrw
i

 17'b1110011_110_???????:`SCPU_ctrl=25'b11110_0000_0_101_1_000_00_00_0001;//csrrs
i

 17'b1110011_111_???????:`SCPU_ctrl=25'b11110_0000_0_101_1_000_00_00_0001;//csrrc
i
            default:
  `SCPU_ctrl=25'b11110_0000_0_000_0_000_00_00_0010;//ILLEGAL
        endcase
    end
endmodule
```

1.2 Datapath.v

datapath图如下(这是在没实现中断前画的,所以未包括mret,ecall,csr等指令),下面介绍了Datapath中的各个模块



1. PC_select.v

该函数用来选择要跳转的指令地址,当前指令地址为PC,输入的地址有PC+4,PC+imm_out(imm_out为ImmGen生成的立即数),ALU_result和mepc;用来控制的选择信号有Jump[1:0],Branch[1:0],ALU_zero和mret,根据这些控制信号选择要跳转的指令地址

```
`timescale 1ns/1ps
```

```
module PC_select(
    input [31:0]PC1,
    input [31:0]PC2,
    input [31:0]PC3,
    inout [31:0]ALU_result,
    input [1:0]jump,
    input [1:0]branch,
    input ALU_zero,
    input mret,
    output [31:0]PC_out
);
    reg [31:0]PC;
    assign PC_out = PC;
    always @(*) begin
        case({jump,branch,ALU_zero,mret})
            6'b010000,
            6'b010010,//jal
            6'b001000,//branch
            6'b001110:
                PC=PC2;
            6'b100000,
            6'b100010://jalr
                PC=ALU_result;
            6'b000001,
            6'b000011://mret
                PC=PC3;
            default:
                PC=PC1;
        endcase
    end

endmodule
```

2. ImmGen.v

根据不同的指令类型去生成立即数

```
`timescale 1ns/1ps
module ImmGen(
    input wire [31:0]inst_in,
    output wire [31:0]Imm_out
);
reg [31:0] Imm_result;
assign Imm_out=Imm_result;
always @(*) begin
    case(inst_in[6:0])
        7'b0010011,//I-type
        7'b0000011,//load
        7'b1100111://jalr
        Imm_result={{20{inst_in[31]}}, inst_in[31:20]};
        7'b0100011://store
        Imm_result={{20{inst_in[31]}},inst_in[31:25],inst_in[11:7]};
        7'b1100011://branch
        Imm_result=
{{20{inst_in[31]}},inst_in[7],inst_in[30:25],inst_in[11:8],1'b0};
```

```
        7'b0110111,//lui
        7'b0010111://auipc
        Imm_result={inst_in[31:12],12'b0};
        7'b1101111://jal
        Imm_result=
{{12{inst_in[31]}},inst_in[19:12],inst_in[20],inst_in[30:21],1'b0};
        default: Imm_result=0;
    endcase
end
endmodule
```

3. ALU.v

这里的ALU.v用我们之前实验实现的ALU即可

```
`timescale 1ns / 1ps
module ALU (
  input [31:0]  A,
  input [31:0]  B,
  input [3:0]   ALU_operation,
  output[31:0]  res,
  output        zero
);
reg [31:0]C;
always @(*)begin
    case(ALU_operation)
    4'd0:C<=A+B;
    4'd1:C<=A-B;
    4'd2:C<=A<<B[4:0];
    4'd3:C<=($signed(A)<$signed(B))?32'b1:32'b0;
    4'd4:C<=(A<B)?32'b1:32'b0;
    4'd5:C<=A^B;
    4'd6:C<=A>>B[4:0];
    4'd7:C<=$signed(A)>>>B[4:0];
    4'd8:C<=A|B;
    4'd9:C<=A&B;
    default:C<=32'b0;
    endcase
end
assign res=C;
assign zero=res?1'b0:1'b1;
endmodule
```

4. Data_change.v

因为我们要实现不同的load类型指令,而我们从数据内存中读取的数据是四个字节的,当执行lb,lh,lbu,lhu指令时,我们需要对读取的数据进行修改后再写入寄存器,可以根据load指令类型和读取的数据地址进行数据的修正

```
`timescale 1ns/1ps
module Data_change(
    input [31:0]data_in,
    input [31:0]ALU_result,
    input [2:0]load_type,
    output [31:0]data_change
```

```
);
    reg [31:0]data;
    assign data_change = data;
    always @(*) begin
        case(load_type)
            3'b000,//none
            3'b111://lw
                data=data_in;
            3'b001://lbu
                if(ALU_result%4==0)begin
                    data={24'b0,data_in[7:0]};
                end
                else if(ALU_result%4==1)begin
                    data={24'b0,data_in[15:8]};
                end
                else if(ALU_result%4==2)begin
                    data={24'b0,data_in[23:16]};
                end
                else if(ALU_result%4==3)begin
                    data={24'b0,data_in[31:24]};
                end
            3'b010://lhu
                if(ALU_result%4==0)begin
                    data={16'b0,data_in[15:0]};
                end
                else if(ALU_result%4==1)begin
                    data={16'b0,data_in[23:8]};
                end
                else if(ALU_result%4==2)begin
                    data={16'b0,data_in[31:16]};
                end
            3'b101://lb
                if(ALU_result%4==0)begin
                    data={{24{data_in[7]}},data_in[7:0]};
                end
                else if(ALU_result%4==1)begin
                    data={{24{data_in[15]}},data_in[15:8]};
                end
                else if(ALU_result%4==2)begin
                    data={{24{data_in[23]}},data_in[23:16]};
                end
                else if(ALU_result%4==3)begin
                    data={{24{data_in[31]}},data_in[31:24]};
                end
            3'b110://lh
                if(ALU_result%4==0)begin
                    data={{16{data_in[15]}},data_in[15:0]};
                end
                else if(ALU_result%4==1)begin
                    data={{16{data_in[23]}},data_in[23:8]};
                end
                else if(ALU_result%4==2)begin
                    data={{16{data_in[31]}},data_in[31:16]};
                end
        endcase
    end
```

```
endmodule
```

5. Data_select.v

该函数用于选取要写入寄存器的数据,被选择的数据有data_change(从数据内存中读取并修正后的数据),ALU_result,immediate,PC+4,

PC+immediate和csr_val(从csr寄存器堆读取的值),选择信号为Memtoreg(该信号由SCPU_ctrl模块得到)

```
`timescale 1ns/1ps
module Data_select(
    input [31:0]data_change,
    input [31:0]ALU_result,
    input [31:0]immediate,
    input [31:0]PC1,
    input [31:0]PC2,
    input [31:0]csr_val,
    input [2:0]Memtoreg,
    output [31:0]data_out
);
    reg [31:0]data;
    assign data_out = data;
    always @(*) begin
        case(Memtoreg)
            3'b000:data=ALU_result;
            3'b001:data=data_change;
            3'b010:data=PC1;
            3'b011:data=immediate;
            3'b100:data=PC2;
            3'b101:data=csr_val;
        endcase
    end

endmodule
```

6.MemRW.v

该函数用于生成数据内存的写使能信号,由于store指令的不同导致写入数据内存的数据并不都是32位,所以要将写使能信号设为4位,根据store指令的类型以及写入的内存地址决定写使能信号

```
`timescale 1ns/1ps
module MemRw(
    input [31:0]ALU_result,
    input [1:0]store_type,
    output [3:0]MemWrite
);
    reg [3:0]MemW;
    assign MemWrite = MemW;
    always @(*)begin
        case(store_type)
            2'b00://none
                MemW = 4'b0000;
            2'b01://sb
                if(ALU_result%4==0)begin
                    MemW=4'b0001;
```

```
                end
                else if(ALU_result%4==1)begin
                    MemW=4'b0010;
                end
                else if(ALU_result%4==2)begin
                    MemW=4'b0100;
                end
                else if(ALU_result%4==3)begin
                    MemW=4'b1000;
                end
            2'b10://sh
                if(ALU_result%4==0)begin
                    MemW=4'b0011;
                end
                else if(ALU_result%4==1)begin
                    MemW=4'b0110;
                end
                else if(ALU_result%4==2)begin
                    MemW=4'b1100;
                end
            2'b11://sw
                MemW=4'b1111;
        endcase
    end

endmodule
```

7. Regs.v

用之前实验实现的寄存器堆即可

```
`timescale 1ns / 1ps
module Regs(
    input clk,
    input rst,
    input [4:0] Rs1_addr,
    input [4:0] Rs2_addr,
    input [4:0] Wt_addr,
    input [31:0]Wt_data,
    input RegWrite,
    output [31:0] Rs1_data,
    output [31:0] Rs2_data,
    output [31:0] Reg00,
    output [31:0] Reg01,
    output [31:0] Reg02,
    output [31:0] Reg03,
    output [31:0] Reg04,
    output [31:0] Reg05,
    output [31:0] Reg06,
    output [31:0] Reg07,
    output [31:0] Reg08,
    output [31:0] Reg09,
    output [31:0] Reg10,
    output [31:0] Reg11,
    output [31:0] Reg12,
    output [31:0] Reg13,
```

```
    output [31:0] Reg14,
    output [31:0] Reg15,
    output [31:0] Reg16,
    output [31:0] Reg17,
    output [31:0] Reg18,
    output [31:0] Reg19,
    output [31:0] Reg20,
    output [31:0] Reg21,
    output [31:0] Reg22,
    output [31:0] Reg23,
    output [31:0] Reg24,
    output [31:0] Reg25,
    output [31:0] Reg26,
    output [31:0] Reg27,
    output [31:0] Reg28,
    output [31:0] Reg29,
    output [31:0] Reg30,
    output [31:0] Reg31
);
reg [31:0]regs[31:0];
integer i;
always @(posedge clk or posedge rst)begin
        if(rst) begin
            for(i=1;i<32;i=i+1)begin
                regs[i]<=32'b0;
            end
        end
        else if(Wt_addr!=5'd0&&RegWrite)
            regs[Wt_addr]<=Wt_data;
end
assign Rs1_data=(Rs1_addr!=5'b0)?regs[Rs1_addr]:32'b0;
assign Rs2_data=(Rs2_addr!=5'b0)?regs[Rs2_addr]:32'b0;
assign Reg00=32'b0;
assign Reg01=regs[1];
assign Reg02=regs[2];
assign Reg03=regs[3];
assign Reg04=regs[4];
assign Reg05=regs[5];
assign Reg06=regs[6];
assign Reg07=regs[7];
assign Reg08=regs[8];
assign Reg09=regs[9];
assign Reg10=regs[10];
assign Reg11=regs[11];
assign Reg12=regs[12];
assign Reg13=regs[13];
assign Reg14=regs[14];
assign Reg15=regs[15];
assign Reg16=regs[16];
assign Reg17=regs[17];
assign Reg18=regs[18];
assign Reg19=regs[19];
assign Reg20=regs[20];
assign Reg21=regs[21];
assign Reg22=regs[22];
assign Reg23=regs[23];
```

```
assign Reg24=regs[24];
assign Reg25=regs[25];
assign Reg26=regs[26];
assign Reg27=regs[27];
assign Reg28=regs[28];
assign Reg29=regs[29];
assign Reg30=regs[30];
assign Reg31=regs[31];
endmodule
```

8. CSRRegs.v

该函数用于对CSR寄存器堆进行读和写,实现方法与之前寄存器堆的实现类似,也有一个读的地址和输出读取的数据,但是写的地址和数据增加到了四个,因为如果是发生中断异常的时候需要更新mstatus,mcause,mtval,mepc这些寄存器,所以要将写入地址和数据加为四个,相应的写使能信号也变为4位

```
`timescale 1ns/1ps
//0:mstatus 1:mtvec 2:mcause 3:mtval 4:mepc
module CSRRegs(
    input clk, rst,
    input[11:0] raddr,waddr1,waddr2,waddr3,waddr4,
    input[31:0] wdata1,wdata2,wdata3,wdata4,
    input [3:0]csr_w,
    output[31:0] rdata,mstatus,mtvec,mcause,mtval,mepc
);
reg [31:0]regs[4:0];
integer i;
always @(posedge clk or posedge rst)begin
        if(rst) begin
            for(i=0;i<5;i=i+1)begin
                regs[i]<=32'b0;
            end
        end
        else if(csr_w!=4'b0)begin
            if(csr_w[3])begin
                regs[waddr1]<=wdata1;
            end
            if(csr_w[2])begin
                regs[waddr2]<=wdata2;
            end
            if(csr_w[1])begin
                regs[waddr3]<=wdata3;
            end
            if(csr_w[0])begin
                regs[waddr4]<=wdata4;
            end
        end
end
assign rdata=regs[raddr];
assign mstatus=regs[0];
assign mtvec=regs[1];
assign mcause=regs[2];
assign mtval=regs[3];
assign mepc=regs[4];
```

```
endmodule
```

9. CSRwdata.v

由于不同指令写入CSR寄存器堆的数据和写使能不同,所以该模块输出写入寄存器堆的数据以及写使能信号。需要注意的是进行trap处理程序时不接受其他trap,所以当发生硬件中断时,如果mstatus=0,则要往CSR寄存器堆写入数据,mstatus=1,即正在进行trap处理程序,这时候就不要因为硬件中断去写入数据

```
`timescale 1ns/1ps
//0:mstatus 1:mtvec 2:mcause 3:mtval 4:mepc
module CSRwdata(
    input [31:0]inst_in,
    input [31:0]PC_current,
    input [31:0]csr_val,
    input [31:0]rs1_val,
    input [31:0]mstatus,
    input INT,
    input ecall,
    input mret,
    input illegal_inst,
    input is_csr,
    output [11:0]waddr1,waddr2,waddr3,waddr4,
    output [31:0]wdata1,wdata2,wdata3,wdata4,
    output [3:0]csr_w
);

reg [11:0] Waddr1,Waddr2,Waddr3,Waddr4;
reg [31:0] Wdata1,Wdata2,Wdata3,Wdata4;
reg [3:0] CSR_w;

assign waddr1=Waddr1;
assign waddr2=Waddr2;
assign waddr3=Waddr3;
assign waddr4=Waddr4;
assign wdata1=Wdata1;
assign wdata2=Wdata2;
assign wdata3=Wdata3;
assign wdata4=Wdata4;
assign csr_w=CSR_w;

always @(*)begin
    if(mstatus==32'b0)begin
        if(INT)begin
            Waddr1=12'd0;
            Waddr2=12'd2;
            Waddr3=12'd3;
            Waddr4=12'd4;
            Wdata1=32'b1;
            Wdata2={1'b1,31'b0};
            Wdata3=32'b0;
            Wdata4=PC_current;
            CSR_w=4'b1111;
        end
```

```
        else if(ecall)begin
            Waddr1=12'd0;
            Waddr2=12'd2;
            Waddr3=12'd3;
            Waddr4=12'd4;
            Wdata1=32'b1;
            Wdata2={31'b0,1'b1};
            Wdata3=32'b0;
            Wdata4=PC_current;
            CSR_w=4'b1111;
        end
        else if(illegal_inst)begin
            Waddr1=12'd0;
            Waddr2=12'd2;
            Waddr3=12'd3;
            Waddr4=12'd4;
            Wdata1=32'b1;
            Wdata2={30'b0,2'b10};
            Wdata3=inst_in;
            Wdata4=PC_current;
            CSR_w=4'b1111;
        end
        else if(is_csr)begin
            Waddr1=inst_in[31:20];
            Waddr2=inst_in[31:20];
            Waddr3=inst_in[31:20];
            Waddr4=inst_in[31:20];
            Wdata2=32'b0;
            Wdata3=32'b0;
            Wdata4=32'b0;
            if(inst_in[19:15]==5'b0)begin
                Wdata1=32'b0;
                CSR_w=4'b0;
            end
            else begin
                CSR_w=4'b1000;
                case(inst_in[14:12])
                3'b001://CSRRW
                Wdata1=rs1_val;
                3'b010://CSRRS
                Wdata1=csr_val|rs1_val;
                3'b011://CSRRC
                Wdata1=csr_val&(~rs1_val);
                3'b101://CSRRWI
                Wdata1={27'b0,inst_in[19:15]};
                3'b110://CSRRSI
                Wdata1=csr_val|{27'b0,inst_in[19:15]};
                3'b111://CSRRCI
                Wdata1=csr_val&(~{27'b0,inst_in[19:15]});
                endcase
            end
        end
        else if(mret)begin
            Waddr1=12'd0;
            Waddr2=12'd2;
            Waddr3=12'd3;
```

```
            Waddr4=12'd4;
            Wdata1=32'b0;
            Wdata2=32'b0;
            Wdata3=32'b0;
            Wdata4=32'b0;
            CSR_w=4'b1000;
        end
        else begin
            Waddr1=12'd0;
            Waddr2=12'd2;
            Waddr3=12'd3;
            Waddr4=12'd4;
            Wdata1=32'b0;
            Wdata2=32'b0;
            Wdata3=32'b0;
            Wdata4=32'b0;
            CSR_w=4'b0;
        end
    end
    else begin
        if(is_csr)begin
            Waddr1=inst_in[31:20];
            Waddr2=inst_in[31:20];
            Waddr3=inst_in[31:20];
            Waddr4=inst_in[31:20];
            Wdata2=32'b0;
            Wdata3=32'b0;
            Wdata4=32'b0;
            if(inst_in[19:15]==5'b0)begin
                Wdata1=32'b0;
                CSR_w=4'b0;
            end
            else begin
                CSR_w=4'b1000;
                case(inst_in[14:12])
                3'b001://CSRRW
                Wdata1=rs1_val;
                3'b010://CSRRS
                Wdata1=csr_val|rs1_val;
                3'b011://CSRRC
                Wdata1=csr_val&(~rs1_val);
                3'b101://CSRRWI
                Wdata1={27'b0,inst_in[19:15]};
                3'b110://CSRRSI
                Wdata1=csr_val|{27'b0,inst_in[19:15]};
                3'b111://CSRRCI
                Wdata1=csr_val&(~{27'b0,inst_in[19:15]});
                endcase
            end
        end
        else if(mret)begin
            Waddr1=12'd0;
            Waddr2=12'd2;
            Waddr3=12'd3;
            Waddr4=12'd4;
            Wdata1=32'b0;
```

```
            Wdata2=32'b0;
            Wdata3=32'b0;
            Wdata4=32'b0;
            CSR_w=4'b1000;
        end
        else begin
            Waddr1=12'd0;
            Waddr2=12'd2;
            Waddr3=12'd3;
            Waddr4=12'd4;
            Wdata1=32'b0;
            Wdata2=32'b0;
            Wdata3=32'b0;
            Wdata4=32'b0;
            CSR_w=4'b0;
        end
    end
end

endmodule
```

10. Datapath.v

    将上面这些模块连接起来后就形成了我们的Datapath

```
`timescale 1ns/1ps

module Datapath(
    input clk,
    input rst,
    input INT,
    input ALUsrc,
    input [3:0]ALUop,
    input [1:0]Branch,
    input [1:0]Jump,
    input RegWrite,
    input MemRead,
    input ecall,
    input mret,
    input illegal_inst,
    input is_csr,
    input [2:0]MemtoReg,
    input [2:0]load_type,
    input [1:0]store_type,
    input [31:0]inst_in,
    input [31:0]Data_in,
    output [31:0]mstatus,
    output [31:0]mtvec,
    output [31:0]mcause,
    output [31:0]mtval,
    output [31:0]mepc,
    output [3:0]MemWrite,
    output [31:0]ALU_result,
    output [31:0]Data_out,
    output [31:0]PC_out,
    output [31:0]rs1_val,
```

```
    output [31:0]rs2_val,
    output [31:0]reg_i_data,
    output [31:0]imm,
    output [31:0]a_val,
    output [31:0]b_val,
    output do_branch,
    output [31:0]pc_branch,
    output [31:0]reg0,
    output [31:0]reg1,
    output [31:0]reg2,
    output [31:0]reg3,
    output [31:0]reg4,
    output [31:0]reg5,
    output [31:0]reg6,
    output [31:0]reg7,
    output [31:0]reg8,
    output [31:0]reg9,
    output [31:0]reg10,
    output [31:0]reg11,
    output [31:0]reg12,
    output [31:0]reg13,
    output [31:0]reg14,
    output [31:0]reg15,
    output [31:0]reg16,
    output [31:0]reg17,
    output [31:0]reg18,
    output [31:0]reg19,
    output [31:0]reg20,
    output [31:0]reg21,
    output [31:0]reg22,
    output [31:0]reg23,
    output [31:0]reg24,
    output [31:0]reg25,
    output [31:0]reg26,
    output [31:0]reg27,
    output [31:0]reg28,
    output [31:0]reg29,
    output [31:0]reg30,
    output [31:0]reg31
);


    reg [31:0]PC;


    wire x;
    wire [31:0]PC_next;
    wire [31:0]Imm_out;
    wire [31:0]Rs1_data;
    wire ALU_zero;
    wire [31:0]Data_change;
    wire [31:0]data_out;
    wire [31:0]rdata;
    wire [31:0]wdata1,wdata2,wdata3,wdata4;
    wire [11:0]waddr1,waddr2,waddr3,waddr4;
    wire [3:0]csr_w;
```

```
assign rs1_val=Rs1_data;
assign rs2_val=Data_out;
assign reg_i_data=data_out;
assign imm=Imm_out;
assign a_val=Rs1_data;
assign b_val=ALUsrc?Imm_out:Data_out;
assign do_branch=Branch[1]&&(~(Branch[0]^ALU_zero));
assign pc_branch=PC_next;
assign PC_out = PC;
assign x=INT||ecall||illegal_inst;


always @(posedge clk or posedge rst) begin
    if(rst) begin
        PC <= 32'b0;
    end
    else if(x==1&&mstatus==32'b0) begin
        PC <= mtvec;
    end
    else begin
        PC <= PC_next;
    end
end

ImmGen ImmGen_1(
    .inst_in(inst_in),
    .Imm_out(Imm_out)
);

Data_change Data_change_1(
    .data_in(Data_in),
    .ALU_result(ALU_result),
    .load_type(load_type),
    .data_change(Data_change)
);

Data_select Data_select_1(
    .data_change(Data_change),
    .ALU_result(ALU_result),
    .immediate(Imm_out),
    .PC1(PC+4),
    .PC2(PC+Imm_out),
    .csr_val(rdata),
    .Memtoreg(MemtoReg),
    .data_out(data_out)
);

PC_select PC_select_1(
    .PC1(PC+4),
    .PC2(PC+Imm_out),
    .PC3(mepc),
    .ALU_result(ALU_result),
    .jump(Jump),
    .branch(Branch),
```

```
        .ALU_zero(ALU_zero),
        .mret(mret),
        .PC_out(PC_next)
    );

    MemRw MemRw_1(
        .ALU_result(ALU_result),
        .store_type(store_type),
        .MemWrite(MemWrite)
    );

    Regs Regs_1(
        .clk(clk),
        .rst(rst),
        .Rs1_addr(inst_in[19:15]),
        .Rs2_addr(inst_in[24:20]),
        .Wt_addr(inst_in[11:7]),
        .Wt_data(data_out),
        .RegWrite(RegWrite),
        .Rs1_data(Rs1_data),
        .Rs2_data(Data_out),
        .Reg00(reg0),
        .Reg01(reg1),
        .Reg02(reg2),
        .Reg03(reg3),
        .Reg04(reg4),
        .Reg05(reg5),
        .Reg06(reg6),
        .Reg07(reg7),
        .Reg08(reg8),
        .Reg09(reg9),
        .Reg10(reg10),
        .Reg11(reg11),
        .Reg12(reg12),
        .Reg13(reg13),
        .Reg14(reg14),
        .Reg15(reg15),
        .Reg16(reg16),
        .Reg17(reg17),
        .Reg18(reg18),
        .Reg19(reg19),
        .Reg20(reg20),
        .Reg21(reg21),
        .Reg22(reg22),
        .Reg23(reg23),
        .Reg24(reg24),
        .Reg25(reg25),
        .Reg26(reg26),
        .Reg27(reg27),
        .Reg28(reg28),
        .Reg29(reg29),
        .Reg30(reg30),
        .Reg31(reg31)
    );

    CSRRegs CSRRegs_1(
```

```
        .clk(clk),
        .rst(rst),
        .raddr(inst_in[31:20]),
        .waddr1(waddr1),
        .waddr2(waddr2),
        .waddr3(waddr3),
        .waddr4(waddr4),
        .wdata1(wdata1),
        .wdata2(wdata2),
        .wdata3(wdata3),
        .wdata4(wdata4),
        .csr_w(csr_w),
        .rdata(rdata),
        .mstatus(mstatus),
        .mtvec(mtvec),
        .mcause(mcause),
        .mtval(mtval),
        .mepc(mepc)
    );

    CSRwdata CSRwdata_1(
        .inst_in(inst_in),
        .PC_current(PC),
        .csr_val(rdata),
        .rs1_val(Rs1_data),
        .mstatus(mstatus),
        .INT(INT),
        .ecall(ecall),
        .mret(mret),
        .illegal_inst(illegal_inst),
        .is_csr(is_csr),
        .waddr1(waddr1),
        .waddr2(waddr2),
        .waddr3(waddr3),
        .waddr4(waddr4),
        .wdata1(wdata1),
        .wdata2(wdata2),
        .wdata3(wdata3),
        .wdata4(wdata4),
        .csr_w(csr_w)
    );

    ALU ALU_1(
        .A(Rs1_data),
        .ALU_operation(ALUop),
        .B(ALUsrc?Imm_out:Data_out),
        .res(ALU_result),
        .zero(ALU_zero)
    );
endmodule
```

1.3 SCPU.v

将SCPU_ctrl与Datapath模块合在一起得到SCPU模块

```
```

```
`timescale 1ns/1ps

module SCPU(
    input clk,
    input rst,
    input INT,
    input [31:0]Data_in,
    input [31:0]inst_in,
    output [31:0]mstatus,
    output [31:0]mtvec,
    output [31:0]mcause,
    output [31:0]mtval,
    output [31:0]mepc,
    output [3:0]Mem_RW,
    output [31:0]Addr_out,
    output reg [31:0]Data_out,
    output [31:0]PC_out,
    output [4:0]rs1,
    output [31:0]rs1_val,
    output [4:0]rs2,
    output [31:0]rs2_val,
    output [4:0]rd,
    output [31:0]reg_i_data,
    output reg_wen,
    output is_imm,
    output is_auipc,
    output is_lui,
    output [31:0]imm,
    output [31:0]a_val,
    output [31:0]b_val,
    output [3:0]alu_ctrl,
    output is_branch,
    output is_jal,
    output is_jalr,
    output do_branch,
    output [31:0]pc_branch,
    output mem_ren,
    output ecall,
    output mret,
    output illegal_inst,
    output is_csr,
    output [31:0]reg0,
    output [31:0]reg1,
    output [31:0]reg2,
    output [31:0]reg3,
    output [31:0]reg4,
    output [31:0]reg5,
    output [31:0]reg6,
    output [31:0]reg7,
    output [31:0]reg8,
    output [31:0]reg9,
    output [31:0]reg10,
    output [31:0]reg11,
    output [31:0]reg12,
    output [31:0]reg13,
    output [31:0]reg14,
```

```
    output [31:0]reg15,
    output [31:0]reg16,
    output [31:0]reg17,
    output [31:0]reg18,
    output [31:0]reg19,
    output [31:0]reg20,
    output [31:0]reg21,
    output [31:0]reg22,
    output [31:0]reg23,
    output [31:0]reg24,
    output [31:0]reg25,
    output [31:0]reg26,
    output [31:0]reg27,
    output [31:0]reg28,
    output [31:0]reg29,
    output [31:0]reg30,
    output [31:0]reg31
);


    wire [1:0]Branch;
    wire [1:0]Jump;
    wire [2:0]MemtoReg;
    wire [2:0]load_type;
    wire [1:0]store_type;
    wire [3:0]MemWrite;
    wire [31:0]data_out;

    assign rs1 = inst_in[19:15];
    assign rs2 = inst_in[24:20];
    assign rd = inst_in[11:7];
    assign is_branch=Branch[1];
    assign is_jal=(Jump==2'b01)?1:0;
    assign is_jalr=(Jump==2'b10)?1:0;




    always@(*)begin
        case(Addr_out%4)
        0:Data_out=data_out;
        1:Data_out=data_out<<8;
        2:Data_out=data_out<<16;
        3:Data_out=data_out<<24;
    endcase
    end


    SCPU_ctrl SCPU_ctrl_1(
        .OPcode(inst_in[6:0]),
        .Fun3(inst_in[14:12]),
        .Fun7(inst_in[31:25]),
        .ALUop(alu_ctrl),
        .ALUsrc(is_imm),
        .Branch(Branch),
        .Jump(Jump),
```

```
        .MemRead(mem_ren),
        .MemtoReg(MemtoReg),
        .RegWrite(reg_wen),
        .load_type(load_type),
        .store_type(store_type),
        .is_auipc(is_auipc),
        .is_lui(is_lui),
        .ecall(ecall),
        .mret(mret),
        .illegal_inst(illegal_inst),
        .is_csr(is_csr)
    );

    Datapath DataPath_1(
        .clk(clk),
        .rst(rst),
        .INT(INT),
        .ALUsrc(is_imm),
        .ALUop(alu_ctrl),
        .Branch(Branch),
        .Jump(Jump),
        .RegWrite(reg_wen),
        .MemRead(mem_ren),
        .ecall(ecall),
        .mret(mret),
        .illegal_inst(illegal_inst),
        .is_csr(is_csr),
        .MemtoReg(MemtoReg),
        .load_type(load_type),
        .store_type(store_type),
        .inst_in(inst_in),
        .Data_in(Data_in),
        .mstatus(mstatus),
        .mtvec(mtvec),
        .mcause(mcause),
        .mtval(mtval),
        .mepc(mepc),
        .MemWrite(Mem_RW),
        .ALU_result(Addr_out),
        .Data_out(data_out),
        .PC_out(PC_out),
        .rs1_val(rs1_val),
        .rs2_val(rs2_val),
        .reg_i_data(reg_i_data),
        .imm(imm),
        .a_val(a_val),
        .b_val(b_val),
        .do_branch(do_branch),
        .pc_branch(pc_branch),
        .reg0(reg0),
        .reg1(reg1),
        .reg2(reg2),
        .reg3(reg3),
        .reg4(reg4),
        .reg5(reg5),
        .reg6(reg6),
```

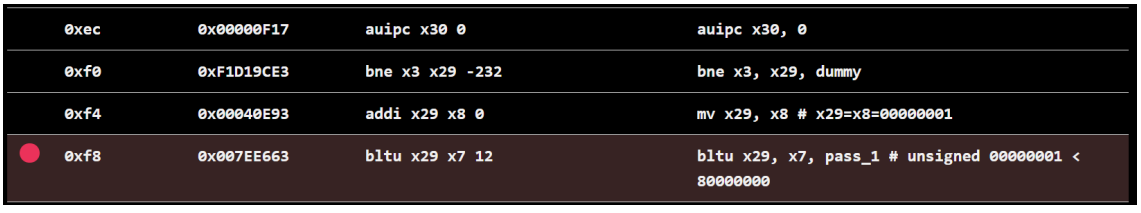
```
        .reg7(reg7),
        .reg8(reg8),
        .reg9(reg9),
        .reg10(reg10),
        .reg11(reg11),
        .reg12(reg12),
        .reg13(reg13),
        .reg14(reg14),
        .reg15(reg15),
        .reg16(reg16),
        .reg17(reg17),
        .reg18(reg18),
        .reg19(reg19),
        .reg20(reg20),
        .reg21(reg21),
        .reg22(reg22),
        .reg23(reg23),
        .reg24(reg24),
        .reg25(reg25),
        .reg26(reg26),
        .reg27(reg27),
        .reg28(reg28),
        .reg29(reg29),
        .reg30(reg30),
        .reg31(reg31)
    );

endmodule
```

二、实验结果与分析

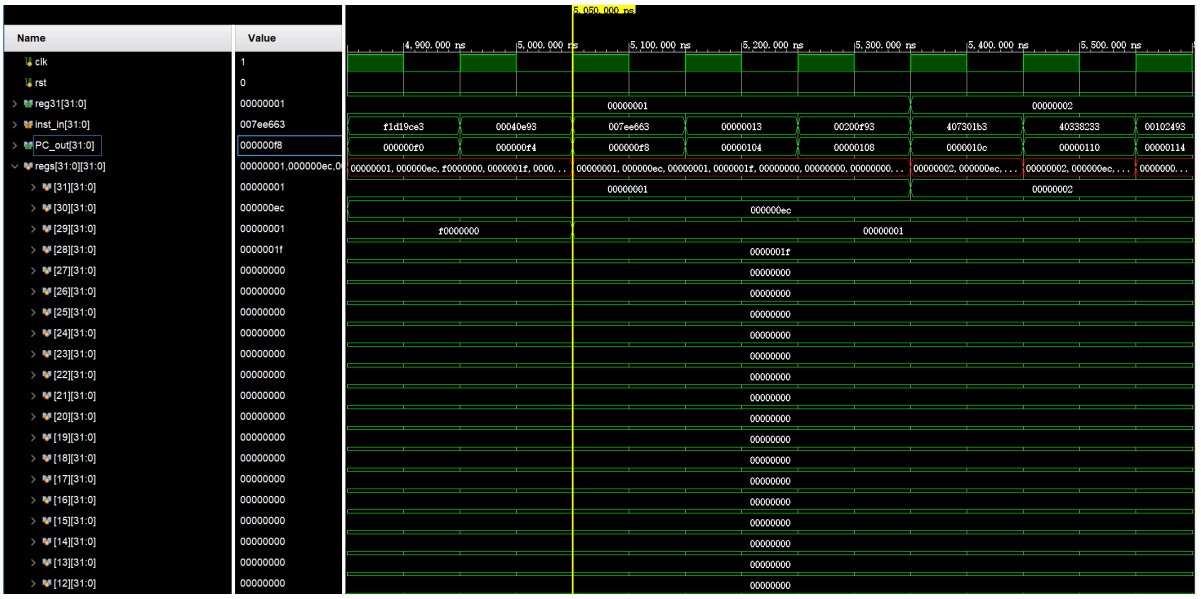
2.1无中断前的lab4-3仿真(仿真代码直接使用给的那个仿真代码)

pass_0部分写入寄存器的次数比较多,pass_4部分使用了不同类型的load和store指令,所以我主要对这两部分进行分析

- pass_0:该部分用多条指令去修改寄存器的值,我就不将这些指令一一叙述了,直接写出最后修改的结果,x1=FFFFFFFF,x3=F0000000,x4=E0000000,x6=C0000000,x7=80000000,x8=00000001,x29=00000001,发生跳转的指令地址是0xf8,对照后发现波形与预期相同



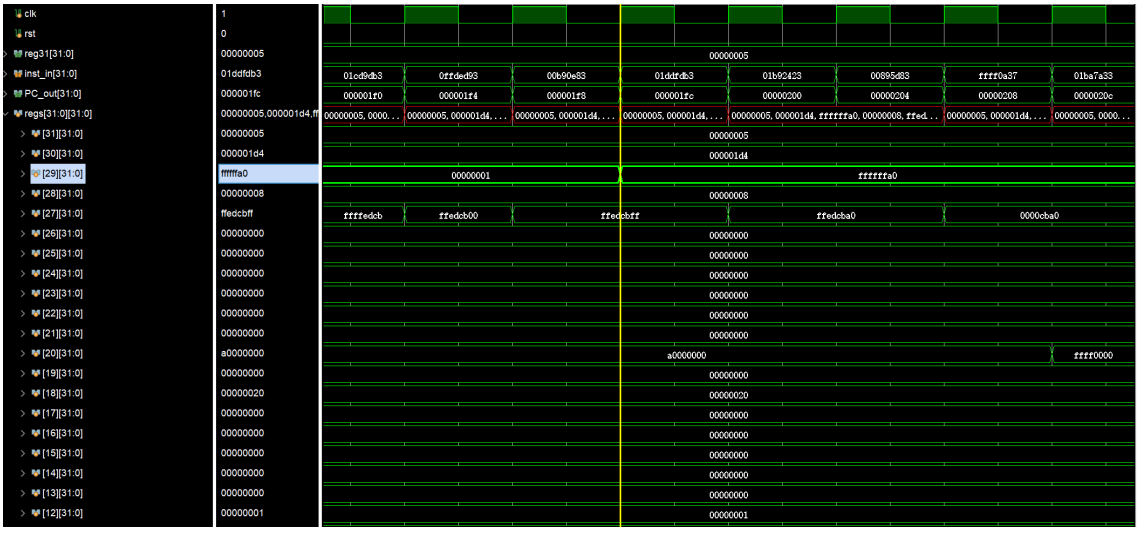
	0xec	0x00000F17	auipc x30 0	auipc x30, 0
	0xf0	0xF1D19CE3	bne x3 x29 -232	bne x3, x29, dummy
	0xf4	0x00040E93	addi x29 x8 0	mv x29, x8 # x29=x8=00000001
●	0xf8	0x007EE663	bltu x29 x7 12	bltu x29, x7, pass_1 # unsigned 00000001 < 80000000



- pass_4部分使用了多种类型的load和store指令,我挑其中几条进行分析
  - lb x29, 11(x18)

    x18为0x20,而数据内存地址0x28存储的数据为A0000000,又因为存储方式为小端存储,所以0x31对应的是高位数据0xA0,lb是取一个字节,所以将0xA0以有符号数方式扩展得到FFFFFFA0

	0x1f4	0x0FFDED93	ori x27 x27 255	ori x27, x27, 0xff # x27=FFEDCBFF
●	0x1f8	0x00B90E83	lb x29 11(x18)	lb x29, 11(x18) # x29=FFFFFFA0, little-endian, signed-ext



  - lhu x27, 8(x18)

    x18为0x20,而数据内存地址0x28存储的数据为FFEDCBA0,取半个字就是CBA0,以无符号数方式扩展后得到0000CBA0

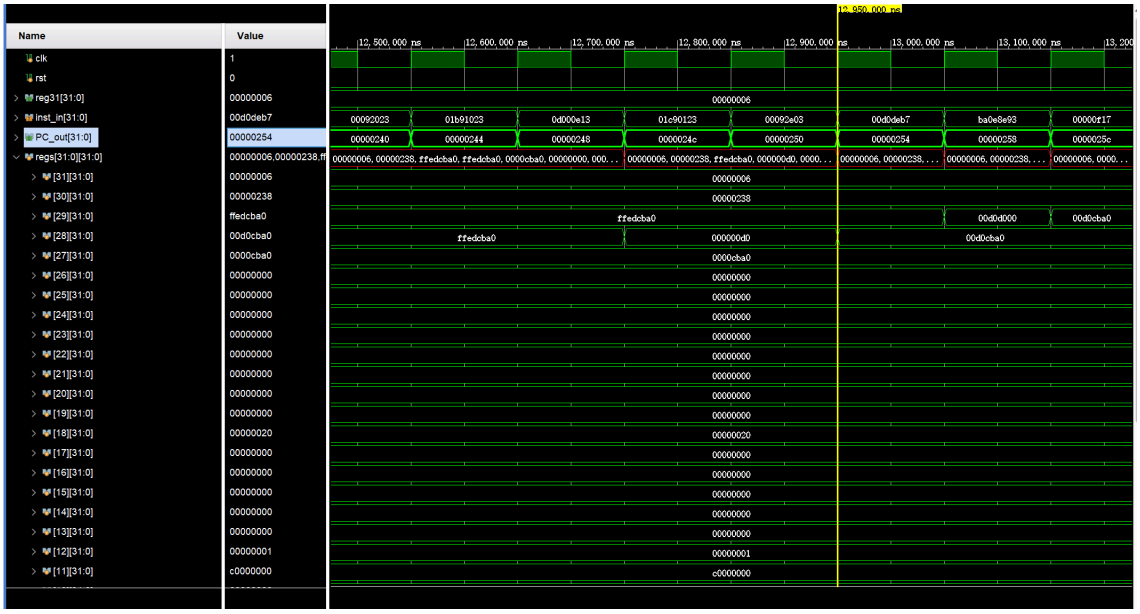
	0x200	0x01B92423	sw x27 8(x18)	sw x27, 8(x18) # mem[0x28]=FFEDCBA0
●	0x204	0x00895D83	lhu x27 8(x18)	lhu x27, 8(x18) # x27=0000CBA0

- sh x27, 0(x18)

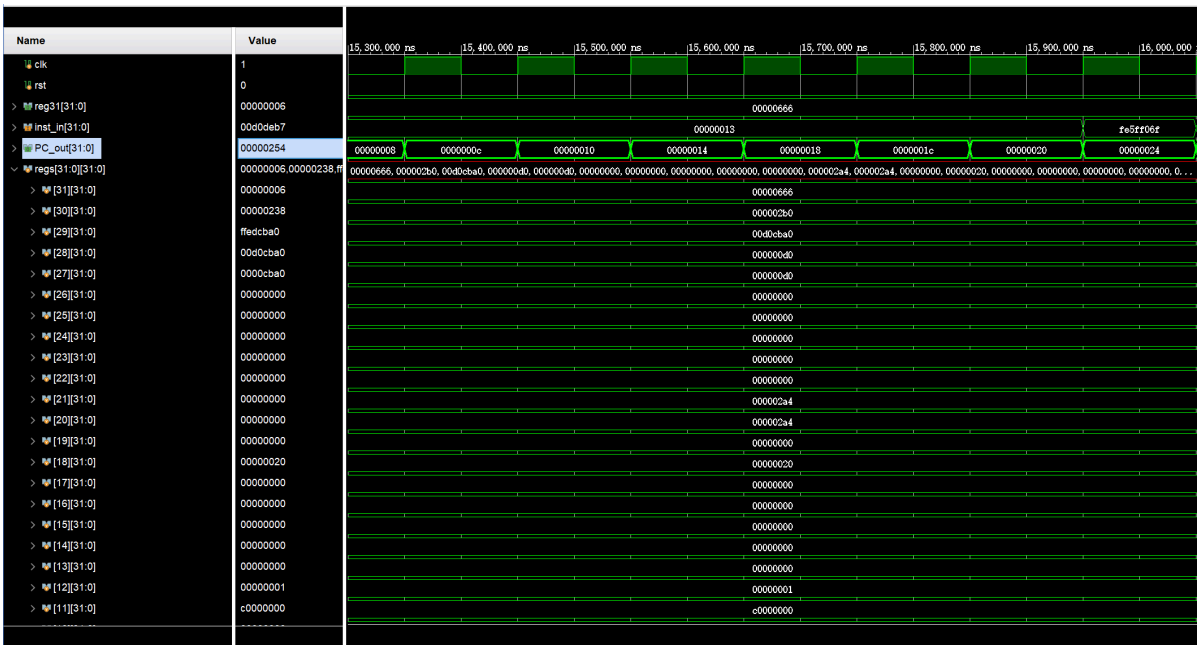
  sb x28, 2(x18)

mem[0x20]本来是0,将x27低的两个字节CBA0存入后,mem[0x20]=0000CBA0,然后又将x28的低的一个字节写入,mem[0x20]=00D0CBA0由之后的`lw x28,0(x18)`指令可知x28应该为00D0CBA0,波形结果正是如此

```
0x244    0x01B91023    sh x27 0(x18)      sh x27, 0(x18) # mem[0x20]=0000CBA0
0x248    0x0D000E13    addi x28 x0 208    li x28, 0xD0
0x24c    0x01C90123    sb x28 2(x18)      sb x28, 2(x18) # mem[0x20]=00D0CBA0
0x250    0x00092E03    lw x28 0(x18)      lw x28, 0(x18) # x28=00D0CBA0
```



- 程序最终会将0x666写入x31,同时进入dummy循环,波形符合预期



2.2加入中断后的lab4-4仿真

- 仿真代码如下

```
`timescale 1ns / 1ps
module testbench_tb();

    reg clk;
    reg rst;
    reg INT;

    testbench m0(.clk(clk), .rst(rst),.INT(INT));

    initial begin
        clk = 1'b0;
        rst = 1'b1;
        INT = 1'b0;
        #5;
        rst = 1'b0;
        #100;
        INT = 1'b1;
        #300
        INT = 1'b0;
    end

    always #10 clk = ~clk;

endmodule
```

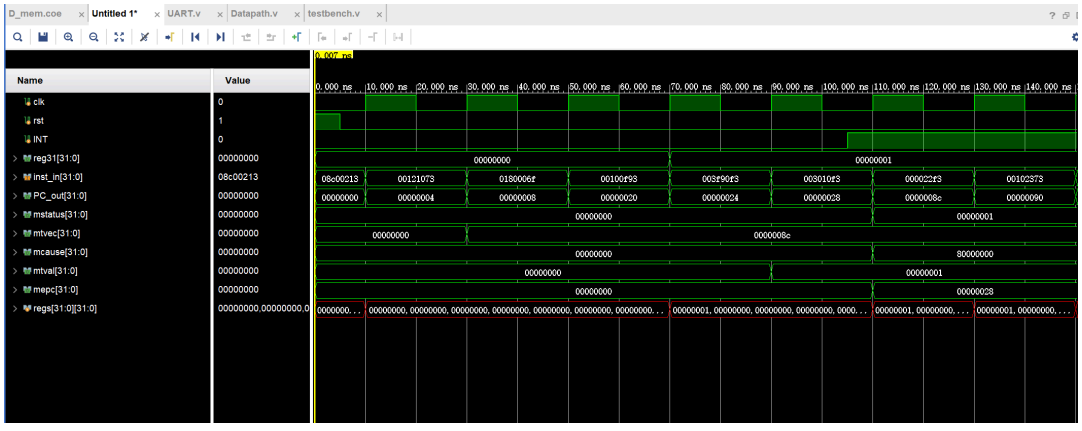
这里写一下trap处理程序是如何实现以及中间执行的指令做了什么。trap处理程序中首先要将mstatus,mtvec,mcause,mtval,mepc寄存器中的值存到通用寄存器里,我这里没有使用栈指针,只是用之前没用到的通用寄存器来存储,x7值为mcause,将其右移31位如果为1,代表是中断,为0则为异常,如果为异常,跳转后需要将mcause右移一位判断是ecall指令还是非法指令,接着再跳转到对应的处理程序。如果是异常,那么需要进行mepc=mepc+4,从而在之后返回原来程序时能够跳过ecall指令或非法指令。硬件中断则不用操作,最终都要跳转到restore_context分支,由于我是用之前没用过的通用寄存器去存值,也就不需要去恢复这些通用寄存器的值,直接使用mret指令跳转回原来程序即可。

pass_0部分与lab4-3给的仿真代码类似,每一部分都给x31一个新值,这里一共有六个部分,验证csr类型的六种指令,每部分如果该部分的csr指令未成功实现,就会跳转到dummy循环,成功实现这六种csr指令后,还有一条非法指令和ecall指令,最后再给x31赋值0x666,跳到dummy循环中,然后我这里的六种csr指令都是对mtval进行操作,是因为trap处理程序中mtval存储的是非法指令,对处理程序中的各个跳转没有影响,如果对其他csr寄存器操作就可能会影响处理程序的正常进行。

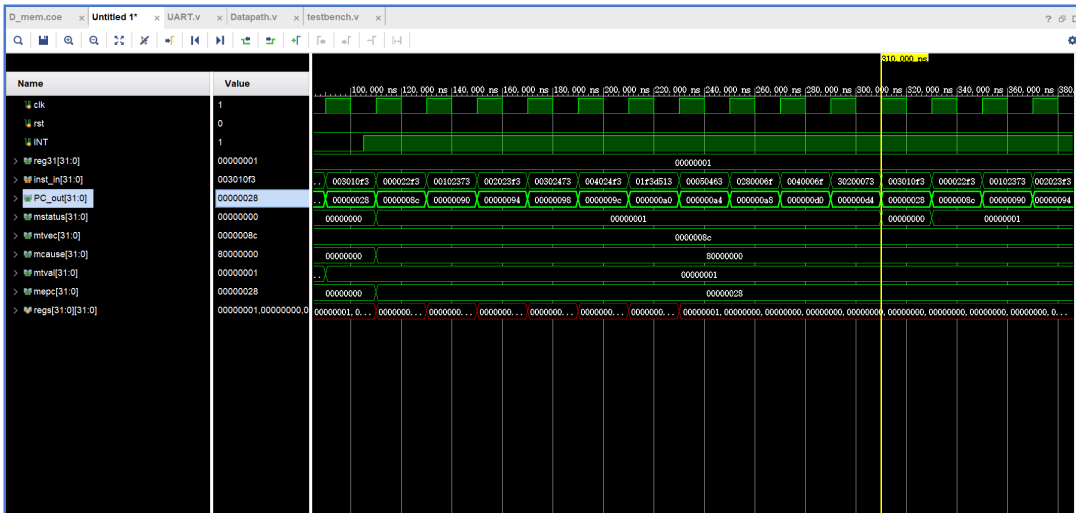
```
    addi x4,x0,0x8c
    csrrw x0,0x1,x4
    j     pass_0
dummy:
    nop
    nop
    nop
    nop
    j     dummy

pass_0:
    li x31,1        #x31=1
    csrrw x1,3,x31  #csr[3]=1,x1=0
    csrrw x1,3,x0   #x1=1
    bne x1,x31,dummy

    li x31,3        #x31=3
    csrrs x1,3,x31  #csr[3]=x31|csr[3]=3,x1=1
    csrrs x1,3,x0   #x1=3
    bne x1,x31,dummy

    li x31,2        #x31=2
    csrrc x2,3,x31  #csr[3]=1,x2=3
    csrrc x1,3,x0   #x1=1
    bge x1,x2,dummy

    li x31,6        #x31=6
    csrrwi x2,3,x5  #csr[3]=5,x2=1
    csrrwi x1,3,x0  #x1=5
    blt x1,x2,dummy

    li x31,7        #x31=7
    csrrsi x2,3,x2  #csr[3]=7,x2=5
    bne x1,x2,dummy

    li x31,8        #x31=8
    csrrci x2,3,x7  #csr[3]=0,x2=7
    csrrci x1,3,x0  #x1=0
    bne x1,x0,dummy

    非法指令
    ecall
    li x31,0x666    #x31=0x666
    j       dummy


# Trap 处理程序
trap_handler:
```

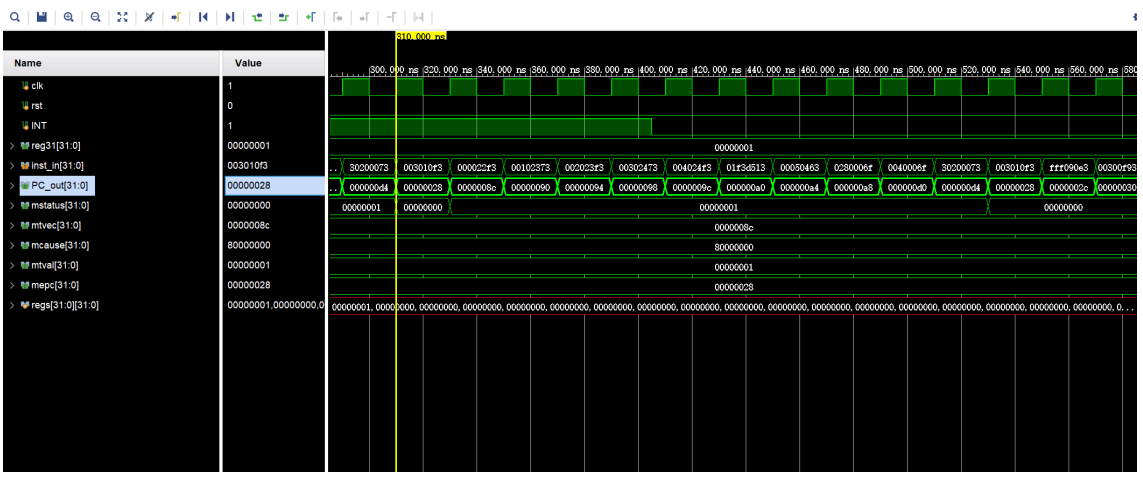
```
    # 读出异常上下文信息
    csrrs x5,0x0,x0     # 保存 mstatus
    csrrs x6,0x1,x0      # 保存 mtvec
    csrrs x7,0x2,x0         # 保存 mcause
    csrrs x8,0x3,x0       # 保存 mtval
    csrrs x9,0x4,x0        # 保存 mepc

    # 判断是中断还是异常
    srli x10,x7,31       # 提取最高位，判断是中断还是异常
    beq x10,x0,is_exception

is_interrupt:
    jal x0,handle_hardware_interrupt     # 硬件中断

is_exception:
    srli x7,x7,1
    beq x7,x0,handle_ecall  # ecall
    jal x0,handle_illegal_instruction # 非法指令

handle_ecall:
    addi x9,x9,4        # ecall: mepc += 4
    csrrw x0,0x4,x9    # 更新 mepc
    jal x0,restore_context

handle_illegal_instruction:
    addi x9,x9,4      # 非法指令: mepc += 4
    csrrw x0,0x4,x9 # 更新 mepc
    jal x0,restore_context

handle_hardware_interrupt:
    # 硬件中断: mepc 不变
    jal x0,restore_context

restore_context:
    mret                  # 返回到原来的程序
```

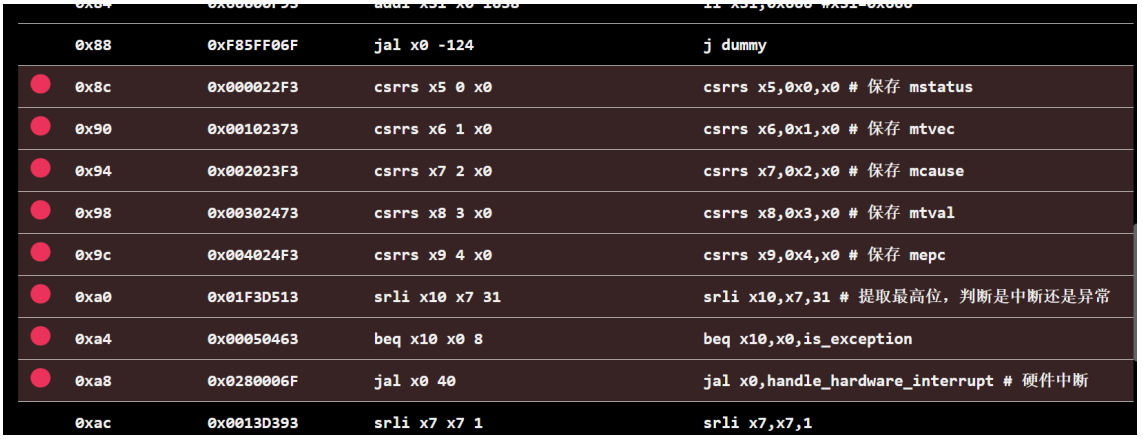
- 仿真波形分析
  - 最开始先将trap处理程序的地址0x8c写入mtvec,第一部分验证csrrw指令,在硬件中断开始前,指令地址为0x28,0x24的指令为csrrw x1,3,x31,可以看到1已经写入mtval,且发生中断后的下一个时钟上升沿,mstatus变为1,表示进入trap处理程序,mcause变为{1'b1,31'b0},表示发生中断,当前指令地址0x28被写入mepc中



- 进入trap处理程序后,应先将csr寄存器值都存起来,然后跳转到处理中断的分支,最后再跳回返回分支,可以看到波形上的指令地址与分析的指令地址一致。然后跳转回0x28地址后发现仍存在硬件中断,所以会继续跳转到trap处理程序,再将之前处理硬件中断的过程重复一遍



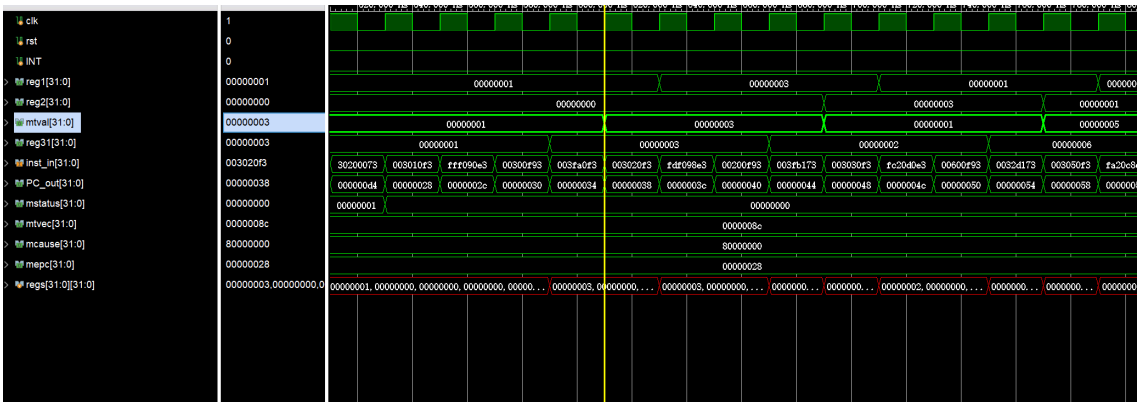




0x88	0xF85FF06F	jal x0 -124	j dummy
0x8c	0x000022F3	csrrs x5 0 x0	csrrs x5,0x0,x0 # 保存 mstatus
0x90	0x00102373	csrrs x6 1 x0	csrrs x6,0x1,x0 # 保存 mtvec
0x94	0x002023F3	csrrs x7 2 x0	csrrs x7,0x2,x0 # 保存 mcause
0x98	0x00302473	csrrs x8 3 x0	csrrs x8,0x3,x0 # 保存 mtval
0x9c	0x004024F3	csrrs x9 4 x0	csrrs x9,0x4,x0 # 保存 mepc
0xa0	0x01F3D513	srli x10 x7 31	srli x10,x7,31 # 提取最高位，判断是中断还是异常
0xa4	0x00050463	beq x10 x0 8	beq x10,x0,is_exception
0xa8	0x0280006F	jal x0 40	jal x0,handle_hardware_interrupt # 硬件中断
0xac	0x0013D393	srli x7 x7 1	srli x7,x7,1

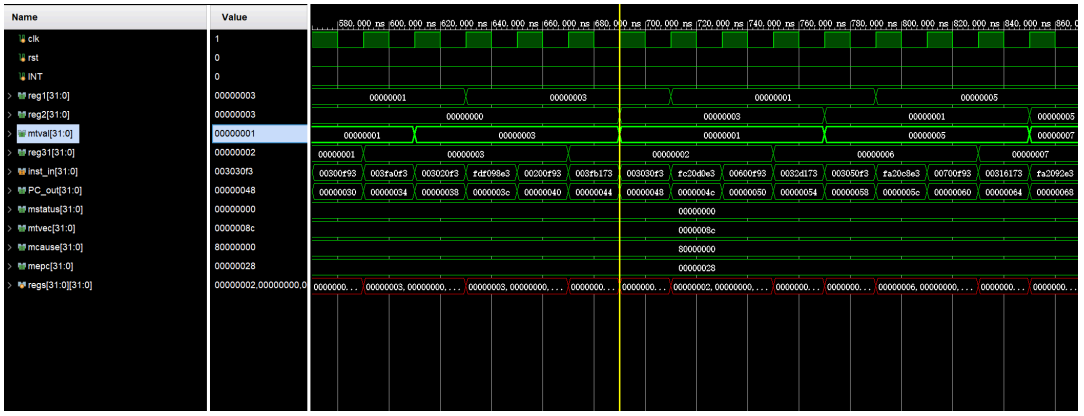
0xcc	0x0080006F	jal x0 8	jal x0,restore_context
0xd0	0x0040006F	jal x0 4	jal x0,restore_context
0xd4	0x00000013	addi x0 x0 0	nop # 返回到原来的程序

- csrrs指令:csrrs x1,3,x31,此时x1=csr[3]=1,csr[3]=x31|csr[3]=3



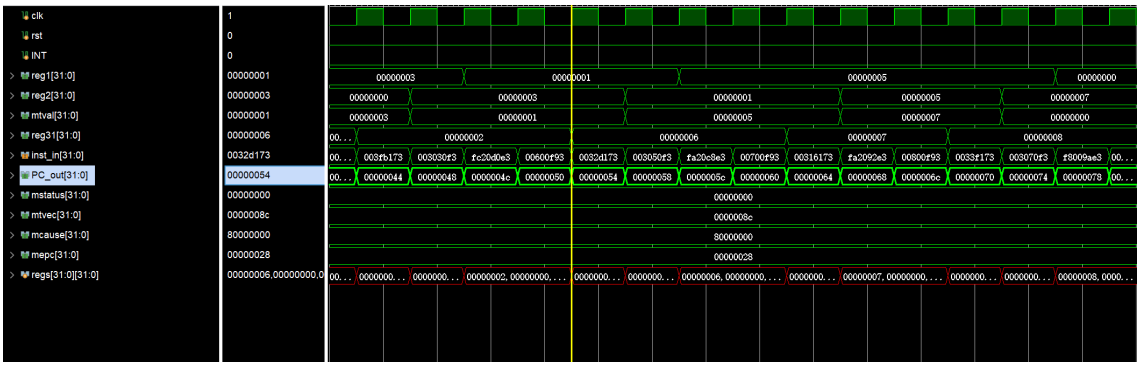
0x2c	0xFFF090E3	bne x1 x31 -32	bne x1,x31,dummy
0x30	0x00300F93	addi x31 x0 3	li x31,3 #x31=3
0x34	0x003FA0F3	csrrs x1 3 x31	csrrs x1,3,x31 #csr[3]=x31|csr[3]=3,x1=1
0x38	0x003020F3	csrrs x1 3 x0	csrrs x1,3,x0 #x1=3
0x3c	0xFDF098E3	bne x1 x31 -48	bne x1,x31,dummy
0x40	0x00200F93	addi x31 x0 2	li x31,2 #x31=2
0x44	0x003FB173	csrrc x2 3 x31	csrrc x2,3,x31 #csr[3]=1,x2=3

- csrrc指令:csrrc x2,3,x31,此时x2=csr[3]=3,csr[3]=csr[3]&(~x31)=1



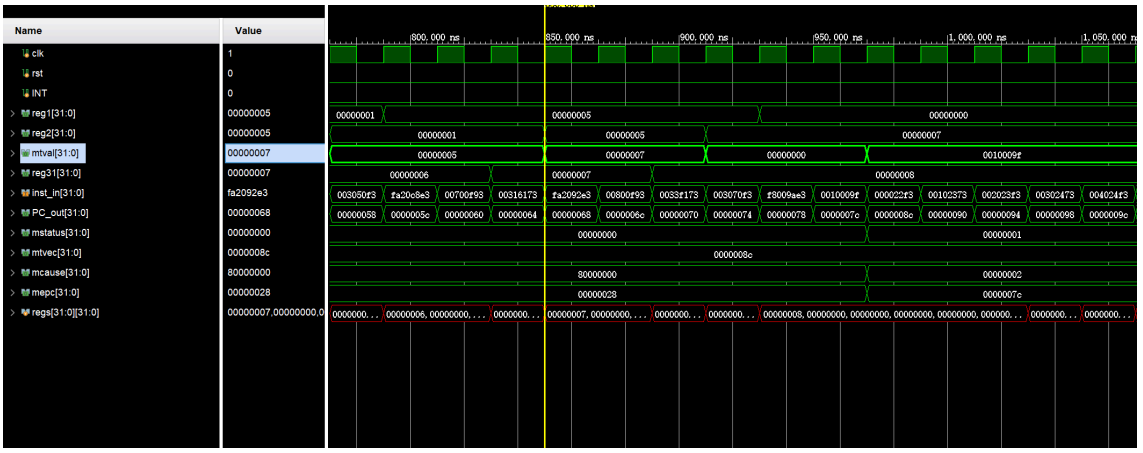
0x40	0x00200F93	addi x31 x0 2	li x31,2 #x31=2
0x44	0x003FB173	csrrc x2 3 x31	csrrc x2,3,x31 #csr[3]=1,x2=3
0x48	0x003030F3	csrrc x1 3 x0	csrrc x1,3,x0 #x1=1
0x4c	0xFC20D0E3	bge x1 x2 -64	bge x1,x2,dummy

- csrrwi指令:csrrwi x2,3,x5,此时x2=csr[3]=1,csr[3]=5



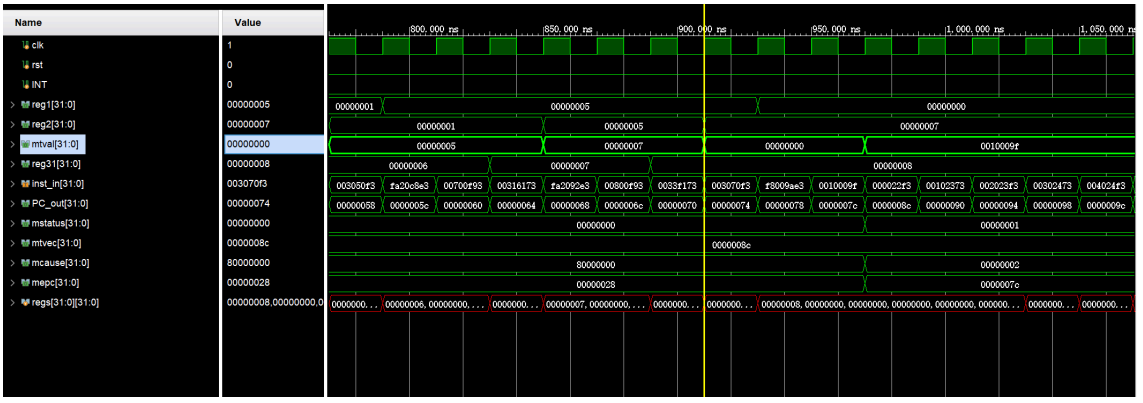
0x50	0x00600F93	addi x31 x0 6	li x31,6 #x31=6
0x54	0x0032D173	csrrwi x2 3 5	csrrwi x2,3,x5 #csr[3]=5,x2=1
0x58	0x003050F3	csrrwi x1 3 0	csrrwi x1,3,x0 #x1=5
0x5c	0xFEA20C8E3	blt x1 x2 -80	blt x1,x2,dummy

- csrrsi指令:csrrsi x2,3,x2,此时x2=csr[3]=5,csr[3]=csr[3]|2=7



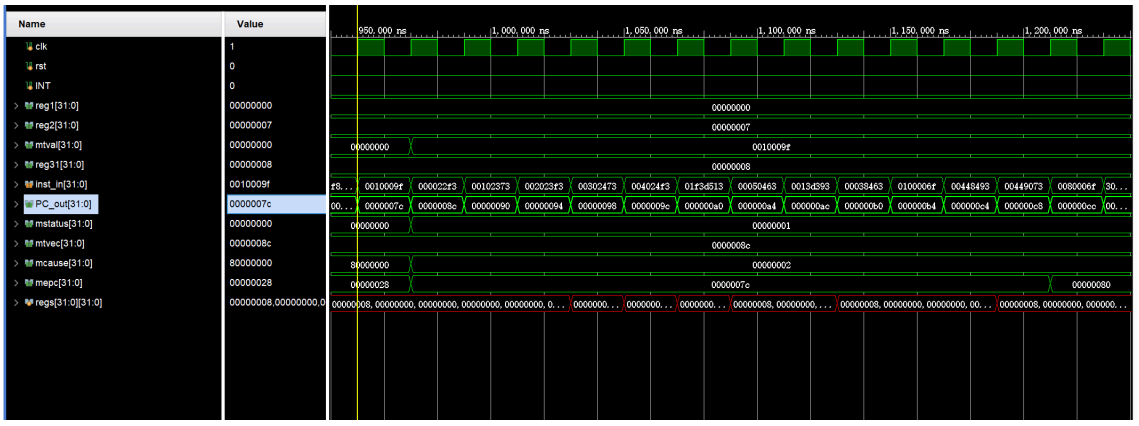
0x60	0x00700F93	addi x31 x0 7	li x31,7 #x31=7
0x64	0x00316173	csrrsi x2 3 2	csrrsi x2,3,x2 #csr[3]=7,x2=5
0x68	0xFA2092E3	bne x1 x2 -92	bne x1,x2,dummy
0x6c	0x00800F93	addi x31 x0 8	li x31,8 #x31=8

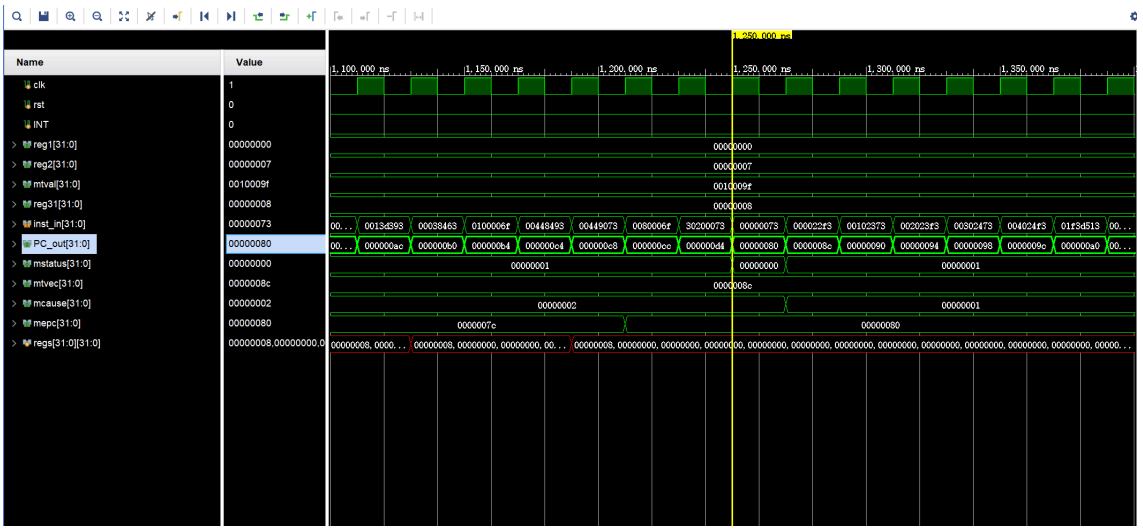
- csrrci指令:csrrci x2,3,x7,此时x2=csr[3]=7,csr[3]=csr[3]&(~7)=0



0x6c	0x00800F93	addi x31 x0 8	li x31,8 #x31=8
0x70	0x0033F173	csrrci x2 3 7	csrrci x2,3,x7 #csr[3]=0,x2=7
0x74	0x003070F3	csrrci x1 3 0	csrrci x1,3,x0 #x1=0
0x78	0xF8009AE3	bne x1 x0 -108	bne x1,x0,dummy

- 地址0x7c处的0x0010009f指令为非法指令,可以看到地址跳转到trap处理程序,mcause变为{30'b0,2'b10},判断为异常后,继续判断该异常由非法指令导致,跳转到处理非法指令部分,将mepc值加4,最后返回mepc中的地址,并且此时mtval中也存储了非法指令的内容

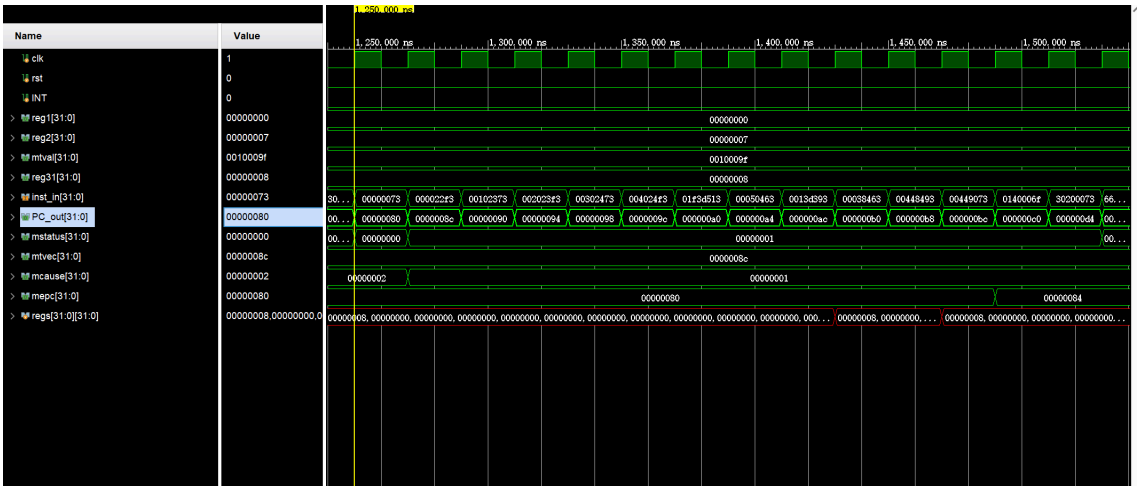


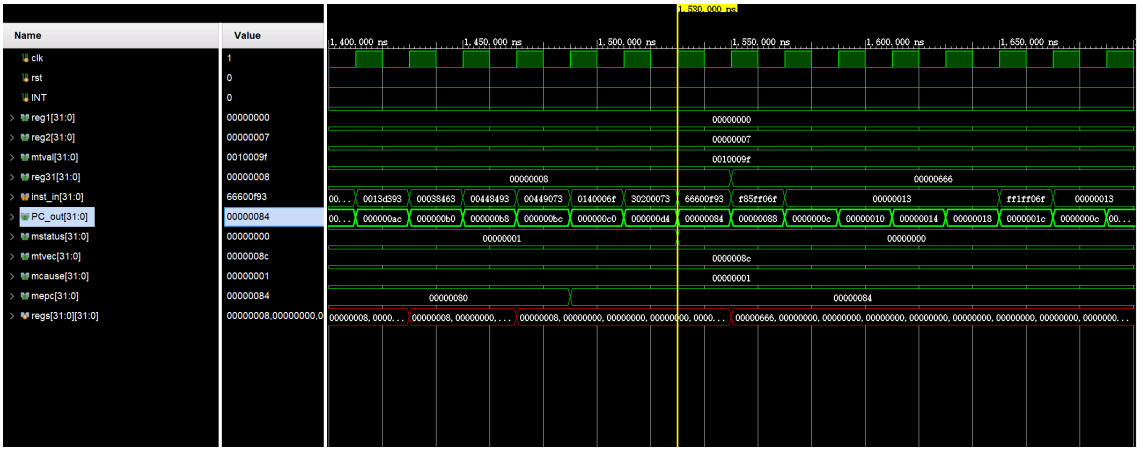


	0x88	0xF85FF06F	jal x0 -124	j dummy
●	0x8c	0x000022F3	csrrs x5 0 x0	csrrs x5,0x0,x0 # 保存 mstatus
●	0x90	0x00102373	csrrs x6 1 x0	csrrs x6,0x1,x0 # 保存 mtvec
●	0x94	0x002023F3	csrrs x7 2 x0	csrrs x7,0x2,x0 # 保存 mcause
●	0x98	0x00302473	csrrs x8 3 x0	csrrs x8,0x3,x0 # 保存 mtval
●	0x9c	0x004024F3	csrrs x9 4 x0	csrrs x9,0x4,x0 # 保存 mepc
●	0xa0	0x01F3D513	srli x10 x7 31	srli x10,x7,31 # 提取最高位，判断是中断还是异常
●	0xa4	0x00050463	beq x10 x0 8	beq x10,x0,is_exception
	0xa8	0x0280006F	jal x0 40	jal x0,handle_hardware_interrupt # 硬件中断
●	0xac	0x0013D393	srli x7 x7 1	srli x7,x7,1
●	0xb0	0x00038463	beq x7 x0 8	beq x7,x0,handle_ecall # ecall
●	0xb4	0x0100006F	jal x0 16	jal x0,handle_illegal_instruction # 非法指令
	0xb8	0x00448493	addi x9 x9 4	addi x9,x9,4 # ecall: mepc += 4

●	0xa4	0x00050463	beq x10 x0 8	beq x10,x0,is_exception
	0xa8	0x0280006F	jal x0 40	jal x0,handle_hardware_interrupt # 硬件中断
●	0xac	0x0013D393	srli x7 x7 1	srli x7,x7,1
●	0xb0	0x00038463	beq x7 x0 8	beq x7,x0,handle_ecall # ecall
●	0xb4	0x0100006F	jal x0 16	jal x0,handle_illegal_instruction # 非法指令
	0xb8	0x00448493	addi x9 x9 4	addi x9,x9,4 # ecall: mepc += 4
	0xbc	0x00449073	csrrw x0 4 x9	csrrw x0,0x4,x9 # 更新 mepc
	0xc0	0x0140006F	jal x0 20	jal x0,restore_context
●	0xc4	0x00448493	addi x9 x9 4	addi x9,x9,4 # 非法指令: mepc += 4
●	0xc8	0x00449073	csrrw x0 4 x9	csrrw x0,0x4,x9 # 更新 mepc
●	0xcc	0x0080006F	jal x0 8	jal x0,restore_context
	0xd0	0x0040006F	jal x0 4	jal x0,restore_context
●	0xd4	0x00000013	addi x0 x0 0	nop # 返回到原来的程序

- 地址0x80存储的是ecall指令,此时也是跳转到trap处理程序,mcause变为{31'b0,1'b1},判断为异常后跳转,接着判断异常由ecall指令导致,跳转到处理ecall,将mepc值加4,最终返回到mepc储存的地址

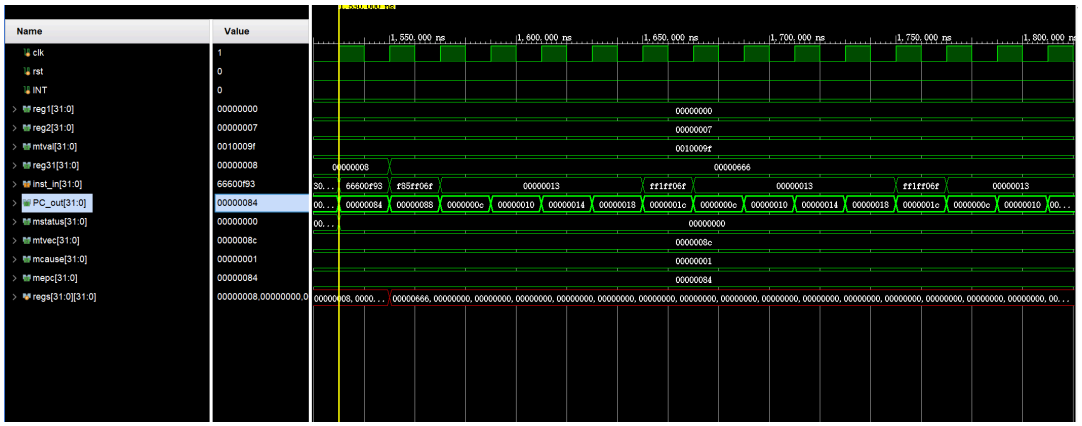


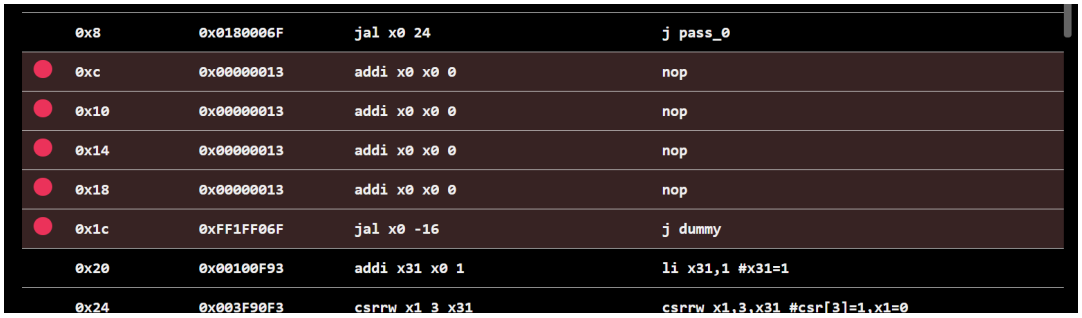


	0x88	0xF85FF06F	jal x0 -124	j dummy
●	0x8c	0x000022F3	csrrs x5 0 x0	csrrs x5,0x0,x0 # 保存 mstatus
●	0x90	0x00102373	csrrs x6 1 x0	csrrs x6,0x1,x0 # 保存 mtvec
●	0x94	0x002023F3	csrrs x7 2 x0	csrrs x7,0x2,x0 # 保存 mcause
●	0x98	0x00302473	csrrs x8 3 x0	csrrs x8,0x3,x0 # 保存 mtval
●	0x9c	0x004024F3	csrrs x9 4 x0	csrrs x9,0x4,x0 # 保存 mepc
●	0xa0	0x01F3D513	srli x10 x7 31	srli x10,x7,31 # 提取最高位，判断是中断还是异常
●	0xa4	0x00050463	beq x10 x0 8	beq x10,x0,is_exception
	0xa8	0x0280006F	jal x0 40	jal x0,handle_hardware_interrupt # 硬件中断
●	0xac	0x0013D393	srli x7 x7 1	srli x7,x7,1
●	0xb0	0x00038463	beq x7 x0 8	beq x7,x0,handle_ecall # ecall
	0xb4	0x0100006F	jal x0 16	jal x0,handle_illegal_instruction # 非法指令

●	0xac	0x0013D393	srli x7 x7 1	srli x7,x7,1
●	0xb0	0x00038463	beq x7 x0 8	beq x7,x0,handle_ecall # ecall
	0xb4	0x0100006F	jal x0 16	jal x0,handle_illegal_instruction # 非法指令
●	0xb8	0x00448493	addi x9 x9 4	addi x9,x9,4 # ecall: mepc += 4
●	0xbc	0x00449073	csrrw x0 4 x9	csrrw x0,0x4,x9 # 更新 mepc
●	0xc0	0x0140006F	jal x0 20	jal x0,restore_context
	0xc4	0x00448493	addi x9 x9 4	addi x9,x9,4 # 非法指令: mepc += 4
	0xc8	0x00449073	csrrw x0 4 x9	csrrw x0,0x4,x9 # 更新 mepc
	0xcc	0x0080006F	jal x0 8	jal x0,restore_context
	0xd0	0x0040006F	jal x0 4	jal x0,restore_context
●	0xd4	0x00000013	addi x0 x0 0	nop # 返回到原来的程序

- 最终x31变为0x666,跳转到dummy循环





	0x8	0x0180006F	jal x0 24	j pass_0
●	0xc	0x00000013	addi x0 x0 0	nop
●	0x10	0x00000013	addi x0 x0 0	nop
●	0x14	0x00000013	addi x0 x0 0	nop
●	0x18	0x00000013	addi x0 x0 0	nop
●	0x1c	0xFF1FF06F	jal x0 -16	j dummy
	0x20	0x00100F93	addi x31 x0 1	li x31,1 #x31=1
	0x24	0x003F90F3	csrrw x1 3 x31	csrrw x1,3,x31 #csr[3]=1,x1=0

2.3 未加入中断的上板验证

因为中间指令太多了,所以我就直接看最后的x31寄存器值是否为0x666,可以看到符合预期

```
pc: 00000008    inst: 00000013

x0: 00000000    ra: FFFFFFFF    sp: 00000000    gp: 40000000    tp: 40000000
t0: F8000000    t1: C0000000    t2: 80000000    s0: 00000001    s1: 00000001
a0: 00000000    a1: C0000000    a2: 00000001    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000020    s3: 00000000
s4: 000002A4    s5: 000002A4    s6: 00000000    s7: 00000000    s8: 00000000
s9: 00000000    s10:00000000    s11:000000D0    t3: 000000D0    t4: 00D0CBA0
t5: 000002B0    t6: 00000666

rs1: 00 rs1_val: 00000000
rs2: 00 rs2_val: 00000000
rd:  00 reg_i_data: 00000000    reg_wen: 1

is_imm: 1       is_auipc: 0     is_lui: 0       imm: 00000000
a_val: 00000000 b_val: 00000000 alu_ctrl: 0     cmp_ctrl: 0
alu_res: 00000000       cmp_res: 0

is_branch: 0    is_jal: 0       is_jalr: 0
do_branch: 0    pc_branch: 0000000C

mem_wen: 0      mem_ren: 0
dmem_o_data: F0000000   dmem_i_data: 00000000   dmem_addr: 00000000

csr_wen: 0      csr_ind: 000    csr_ctrl: 0     csr_r_data: 00000000
mstatus: 00000000       mcause: 00000000        mepc: 00000000  mtval: 00000000
mtvec:   00000000       mie: 00000000   mip: 00000000
```

2.4 加入中断后的上板验证

上面仿真的时候已经分析了六种csr指令对应的csr[3]的值,所以我下面就不分析了,直接把指令和对应的值写出来与上板结果比较

- csrrw x0,0x1,x4

```
pc: 00000008    inst: 0180006F

x0: 00000000    ra: 00000000    sp: 00000000    gp: 00000000    tp: 0000008C
t0: 00000000    t1: 00000000    t2: 00000000    s0: 00000000    s1: 00000000
a0: 00000000    a1: 00000000    a2: 00000000    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000000    s3: 00000000
s4: 00000000    s5: 00000000    s6: 00000000    s7: 00000000    s8: 00000000
s9: 00000000    s10:00000000    s11:00000000    t3: 00000000    t4: 00000000
t5: 00000000    t6: 00000000

rs1: 00 rs1_val: 00000000
rs2: 18 rs2_val: 00000000
rd:  00 reg_i_data: 0000000C    reg_wen: 1

is_imm: 0       is_auipc: 0     is_lui: 0       imm: 00000018
a_val: 00000000 b_val: 00000000 alu_ctrl: F     cmp_ctrl: 0
alu_res: 00000000       cmp_res: 0

is_branch: 0    is_jal: 1       is_jalr: 0
do_branch: 0    pc_branch: 00000020

mem_wen: 0      mem_ren: 0
dmem_o_data: F0000000   dmem_i_data: 00000000   dmem_addr: 00000000

csr_wen: 0      csr_ind: 000    csr_ctrl: 0     csr_r_data: 00000000
mstatus: 00000000       mcause: 00000000        mepc: 00000000  mtval: 00000000
mtvec:   0000008C       mie: 00000000   mip: 00000000
```

```
    0x0         0x08C00213      addi x4 x0 140                  addi x4,x0,0x8c
    0x4         0x00121073      csrrw x0 1 x4                   csrrw x0,0x1,x4
    0x8         0x0180006F      jal x0 24                       j pass_0
    0xc         0x00000013      addi x0 x0 0                    nop
```

- csrrs x1,3,x31 #csr[3]=x31 | csr[3]=3,x1=1

```
pc: 00000038    inst: 003020F3

x0: 00000000    ra: 00000001    sp: 00000000    gp: 00000000    tp: 0000008C
t0: 00000000    t1: 00000000    t2: 00000000    s0: 00000000    s1: 00000000
a0: 00000000    a1: 00000000    a2: 00000000    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000000    s3: 00000000
s4: 00000000    s5: 00000000    s6: 00000000    s7: 00000000    s8: 00000000
s9: 00000000    s10:00000000    s11:00000000    t3: 00000000    t4: 00000000
t5: 00000000    t6: 00000003

rs1: 00 rs1_val: 00000000
rs2: 03 rs2_val: 00000000
rd:  01 reg_i_data: 00000003    reg_wen: 1

is_imm: 0       is_auipc: 0     is_lui: 0       imm: 00000000
a_val: 00000000 b_val: 00000000 alu_ctrl: F     cmp_ctrl: 0
alu_res: 00000000       cmp_res: 0

is_branch: 0    is_jal: 0       is_jalr: 0
do_branch: 0    pc_branch: 0000003C

mem_wen: 0      mem_ren: 0
dmem_o_data: F0000000   dmem_i_data: 00000000   dmem_addr: 00000000

csr_wen: 0      csr_ind: 000    csr_ctrl: 0     csr_r_data: 00000000
mstatus: 00000000       mcause: 00000000        mepc: 00000000  mtval: 00000003
mtvec:   0000008C       mie: 00000000   mip: 00000000
```

```
    0x30        0x00300F93      addi x31 x0 3                   li x31,3 #x31=3
    0x34        0x003FA0F3      csrrs x1 3 x31                  csrrs x1,3,x31 #csr[3]=x31|csr[3]=3,x1=1
    0x38        0x003020F3      csrrs x1 3 x0                   csrrs x1,3,x0 #x1=3
    0x3c        0xFDF098E3      bne x1 x31 -48                  bne x1,x31,dummy
```

- csrrc x2,3,x31 #csr[3]=1,x2=3

```
pc: 00000048    inst: 003030F3

x0: 00000000    ra: 00000003    sp: 00000003    gp: 00000000    tp: 0000008C
t0: 00000000    t1: 00000000    t2: 00000000    s0: 00000000    s1: 00000000
a0: 00000000    a1: 00000000    a2: 00000000    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000000    s3: 00000000
s4: 00000000    s5: 00000000    s6: 00000000    s7: 00000000    s8: 00000000
s9: 00000000    s10:00000000    s11:00000000    t3: 00000000    t4: 00000000
t5: 00000000    t6: 00000002

rs1: 00 rs1_val: 00000000
rs2: 03 rs2_val: 00000000
rd:  01 reg_i_data: 00000001    reg_wen: 1

is_imm: 0       is_auipc: 0     is_lui: 0       imm: 00000000
a_val: 00000000 b_val: 00000000 alu_ctrl: F     cmp_ctrl: 0
alu_res: 00000000       cmp_res: 0

is_branch: 0    is_jal: 0       is_jalr: 0
do_branch: 0    pc_branch: 0000004C

mem_wen: 0      mem_ren: 0
dmem_o_data: F0000000   dmem_i_data: 00000000   dmem_addr: 00000000

csr_wen: 0      csr_ind: 000    csr_ctrl: 0     csr_r_data: 00000000
mstatus: 00000000       mcause: 00000000        mepc: 00000000  mtval: 00000001
mtvec:   0000008C       mie: 00000000   mip: 00000000
```

```
    0x40        0x00200F93      addi x31 x0 2                   li x31,2 #x31=2
    0x44        0x003FB173      csrrc x2 3 x31                  csrrc x2,3,x31 #csr[3]=1,x2=3
    0x48        0x003030F3      csrrc x1 3 x0                   csrrc x1,3,x0 #x1=1
    0x4c        0xFC20D0E3      bge x1 x2 -64                   bge x1,x2,dummy
```

- csrrwi x2,3,x5 #csr[3]=5,x2=1

```
pc: 00000058    inst: 003050F3

x0: 00000000    ra: 00000001    sp: 00000001    gp: 00000000    tp: 0000008C
t0: 00000000    t1: 00000000    t2: 00000000    s0: 00000000    s1: 00000000
a0: 00000000    a1: 00000000    a2: 00000000    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000000    s3: 00000000
s4: 00000000    s5: 00000000    s6: 00000000    s7: 00000000    s8: 00000000
s9: 00000000    s10:00000000    s11:00000000    t3: 00000000    t4: 00000000
t5: 00000000    t6: 00000006

rs1: 00 rs1_val: 00000000
rs2: 03 rs2_val: 00000000
rd:  01 reg_i_data: 00000005    reg_wen: 1

is_imm: 0       is_auipc: 0     is_lui: 0       imm: 00000000
a_val: 00000000 b_val: 00000000 alu_ctrl: F     cmp_ctrl: 0
alu_res: 00000000       cmp_res: 0

is_branch: 0    is_jal: 0       is_jalr: 0
do_branch: 0    pc_branch: 0000005C

mem_wen: 0      mem_ren: 0
dmem_o_data: F0000000   dmem_i_data: 00000000   dmem_addr: 00000000

csr_wen: 0      csr_ind: 000    csr_ctrl: 0     csr_r_data: 00000000
mstatus: 00000000       mcause: 00000000        mepc: 00000000  mtval: 00000005
mtvec:   0000008C       mie: 00000000   mip: 00000000
```

```
    0x50        0x00600F93      addi x31 x0 6                   li x31,6 #x31=6
    0x54        0x0032D173      csrrwi x2 3 5                   csrrwi x2,3,x5 #csr[3]=5,x2=1
    0x58        0x003050F3      csrrwi x1 3 0                   csrrwi x1,3,x0 #x1=5
    0x5c        0xFA20C8E3      blt x1 x2 -80                   blt x1,x2,dummy
```

- csrrsi x2,3,x2 #csr[3]=7,x2=5

```
pc: 00000068    inst: FA2092E3

x0: 00000000    ra: 00000005    sp: 00000005    gp: 00000000    tp: 0000008C
t0: 00000000    t1: 00000000    t2: 00000000    s0: 00000000    s1: 00000000
a0: 00000000    a1: 00000000    a2: 00000000    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000000    s3: 00000000
s4: 00000000    s5: 00000000    s6: 00000000    s7: 00000000    s8: 00000000
s9: 00000000    s10:00000000    s11:00000000    t3: 00000000    t4: 00000000
t5: 00000000    t6: 00000007

rs1: 01 rs1_val: 00000005
rs2: 02 rs2_val: 00000005
rd:  05 reg_i_data: 00000000    reg_wen: 0

is_imm: 0       is_auipc: 0     is_lui: 0       imm: FFFFFFA4
a_val: 00000005 b_val: 00000005 alu_ctrl: 1     cmp_ctrl: 0
alu_res: 00000000       cmp_res: 0

is_branch: 1    is_jal: 0       is_jalr: 0
do_branch: 0    pc_branch: 0000006C

mem_wen: 0      mem_ren: 0
dmem_o_data: F0000000   dmem_i_data: 00000005   dmem_addr: 00000000

csr_wen: 0      csr_ind: 000    csr_ctrl: 0     csr_r_data: 00000000
mstatus: 00000000       mcause: 00000000        mepc: 00000000  mtval: 00000007
mtvec:   0000008C       mie: 00000000   mip: 00000000
```

```
    0x60        0x00700F93      addi x31 x0 7                   li x31,7 #x31=7
    0x64        0x00316173      csrrsi x2 3 2                   csrrsi x2,3,x2 #csr[3]=7,x2=5
    0x68        0xFA2092E3      bne x1 x2 -92                   bne x1,x2,dummy
    0x6c        0x00800F93      addi x31 x0 8                   li x31,8 #x31=8
```

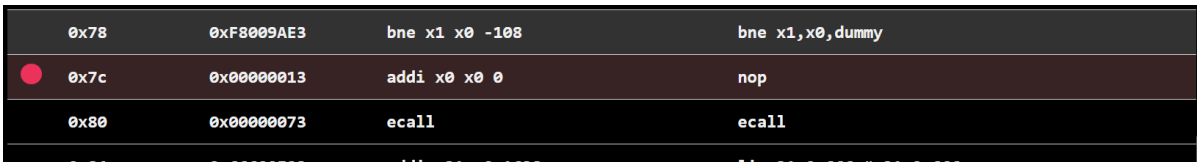
- csrrci x2,3,x7 #csr[3]=0,x2=7

```
pc: 00000074    inst: 003070F3

x0: 00000000    ra: 00000005    sp: 00000007    gp: 00000000    tp: 0000008C
t0: 00000000    t1: 00000000    t2: 00000000    s0: 00000000    s1: 00000000
a0: 00000000    a1: 00000000    a2: 00000000    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000000    s3: 00000000
s4: 00000000    s5: 00000000    s6: 00000000    s7: 00000000    s8: 00000000
s9: 00000000    s10:00000000    s11:00000000    t3: 00000000    t4: 00000000
t5: 00000000    t6: 00000008

rs1: 00 rs1_val: 00000000
rs2: 03 rs2_val: 00000000
rd:  01 reg_i_data: 00000000    reg_wen: 1

is_imm: 0       is_auipc: 0     is_lui: 0       imm: 00000000
a_val: 00000000 b_val: 00000000 alu_ctrl: F     cmp_ctrl: 0
alu_res: 00000000       cmp_res: 0

is_branch: 0    is_jal: 0       is_jalr: 0
do_branch: 0    pc_branch: 00000078

mem_wen: 0      mem_ren: 0
dmem_o_data: F0000000   dmem_i_data: 00000000   dmem_addr: 00000000

csr_wen: 0      csr_ind: 000    csr_ctrl: 0     csr_r_data: 00000000
mstatus: 00000000       mcause: 00000000        mepc: 00000000  mtval: 00000000
mtvec:   0000008C       mie: 00000000   mip: 00000000
```

```
    0x6c        0x00800F93      addi x31 x0 8                   li x31,8 #x31=8
    0x70        0x0033F173      csrrci x2 3 7                   csrrci x2,3,x7 #csr[3]=0,x2=7
    0x74        0x003070F3      csrrci x1 3 0                   csrrci x1,3,x0 #x1=0
```

上面仿真已经分析了发生中断异常时地址以及寄存器的变化,所以下面就直接放这个开始进入trap处理程序以及结束返回原来程序的图片

- 非法指令的处理过程




```
pc: 0000007C    inst: 0010009F

x0: 00000000    ra: 00000000    sp: 00000007    gp: 00000000    tp: 0000008C
t0: 00000000    t1: 00000000    t2: 00000000    s0: 00000000    s1: 00000000
a0: 00000000    a1: 00000000    a2: 00000000    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000000    s3: 00000000
s4: 00000000    s5: 00000000    s6: 00000000    s7: 00000000    s8: 00000000
s9: 00000000    s10:00000000    s11:00000000    t3: 00000000    t4: 00000000
t5: 00000000    t6: 00000008

rs1: 00 rs1_val: 00000000
rs2: 01 rs2_val: 00000000
rd:  01 reg_i_data: 00000000    reg_wen: 0

is_imm: 0       is_auipc: 0     is_lui: 0       imm: 00000000
a_val: 00000000 b_val: 00000000 alu_ctrl: F     cmp_ctrl: 0
alu_res: 00000000       cmp_res: 0

is_branch: 0    is_jal: 0       is_jalr: 0
do_branch: 0    pc_branch: 00000080

mem_wen: 0      mem_ren: 0
dmem_o_data: F0000000   dmem_i_data: 00000000   dmem_addr: 00000000

csr_wen: 0      csr_ind: 000    csr_ctrl: 0     csr_r_data: 00000000
mstatus: 00000000       mcause: 00000000        mepc: 00000000  mtval: 00000000
mtvec:   0000008C       mie: 00000000   mip: 00000000
```

```
pc: 0000008C    inst: 000022F3

x0: 00000000    ra: 00000000    sp: 00000007    gp: 00000000    tp: 0000008C
t0: 00000000    t1: 00000000    t2: 00000000    s0: 00000000    s1: 00000000
a0: 00000000    a1: 00000000    a2: 00000000    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000000    s3: 00000000
s4: 00000000    s5: 00000000    s6: 00000000    s7: 00000000    s8: 00000000
s9: 00000000    s10:00000000    s11:00000000    t3: 00000000    t4: 00000000
t5: 00000000    t6: 00000008

rs1: 00 rs1_val: 00000000
rs2: 00 rs2_val: 00000000
rd:  05 reg_i_data: 00000001    reg_wen: 1

is_imm: 0       is_auipc: 0     is_lui: 0       imm: 00000000
a_val: 00000000 b_val: 00000000 alu_ctrl: F     cmp_ctrl: 0
alu_res: 00000000       cmp_res: 0

is_branch: 0    is_jal: 0       is_jalr: 0
do_branch: 0    pc_branch: 00000090

mem_wen: 0      mem_ren: 0
dmem_o_data: F0000000   dmem_i_data: 00000000   dmem_addr: 00000000

csr_wen: 0      csr_ind: 000    csr_ctrl: 0     csr_r_data: 00000000
mstatus: 00000001       mcause: 00000002        mepc: 0000007C  mtval: 0010009F
mtvec:   0000008C       mie: 00000000   mip: 00000000
```

```
pc: 00000080    inst: 00000073

x0: 00000000    ra: 00000000    sp: 00000007    gp: 00000000    tp: 0000008C
t0: 00000001    t1: 0000008C    t2: 00000001    s0: 0010009F    s1: 00000080
a0: 00000000    a1: 00000000    a2: 00000000    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000000    s3: 00000000
s4: 00000000    s5: 00000000    s6: 00000000    s7: 00000000    s8: 00000000
s9: 00000000    s10:00000000    s11:00000000    t3: 00000000    t4: 00000000
t5: 00000000    t6: 00000008

rs1: 00 rs1_val: 00000000
rs2: 00 rs2_val: 00000000
rd:  00 reg_i_data: 00000000    reg_wen: 0

is_imm: 0       is_auipc: 0     is_lui: 0       imm: 00000000
a_val: 00000000 b_val: 00000000 alu_ctrl: F     cmp_ctrl: 0
alu_res: 00000000       cmp_res: 0

is_branch: 0    is_jal: 0       is_jalr: 0
do_branch: 0    pc_branch: 00000084

mem_wen: 0      mem_ren: 0
dmem_o_data: F0000000   dmem_i_data: 00000000   dmem_addr: 00000000

csr_wen: 0      csr_ind: 000    csr_ctrl: 0     csr_r_data: 00000000
mstatus: 00000000       mcause: 00000002        mepc: 00000080  mtval: 0010009F
mtvec:   0000008C       mie: 00000000   mip: 00000000
```

- ecall指令的处理过程

```
0x7c        0x00000013      addi x0 x0 0                    nop
0x80        0x00000073      ecall                           ecall
0x84        0x66600F93      addi x31 x0 1638                li x31,0x666 #x31=0x666
```

```
pc: 00000080    inst: 00000073

x0: 00000000    ra: 00000000    sp: 00000007    gp: 00000000    tp: 0000008C
t0: 00000001    t1: 0000008C    t2: 00000001    s0: 0010009F    s1: 00000080
a0: 00000000    a1: 00000000    a2: 00000000    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000000    s3: 00000000
s4: 00000000    s5: 00000000    s6: 00000000    s7: 00000000    s8: 00000000
s9: 00000000    s10:00000000    s11:00000000    t3: 00000000    t4: 00000000
t5: 00000000    t6: 00000008

rs1: 00 rs1_val: 00000000
rs2: 00 rs2_val: 00000000
rd:  00 reg_i_data: 00000000    reg_wen: 0

is_imm: 0       is_auipc: 0     is_lui: 0       imm: 00000000
a_val: 00000000 b_val: 00000000 alu_ctrl: F     cmp_ctrl: 0
alu_res: 00000000       cmp_res: 0

is_branch: 0    is_jal: 0       is_jalr: 0
do_branch: 0    pc_branch: 00000084

mem_wen: 0      mem_ren: 0
dmem_o_data: F0000000   dmem_i_data: 00000000   dmem_addr: 00000000

csr_wen: 0      csr_ind: 000    csr_ctrl: 0     csr_r_data: 00000000
mstatus: 00000000       mcause: 00000002        mepc: 00000080  mtval: 0010009F
mtvec:   0000008C       mie: 00000000   mip: 00000000
```

```
pc: 0000008C    inst: 000022F3

x0: 00000000    ra: 00000000    sp: 00000007    gp: 00000000    tp: 0000008C
t0: 00000001    t1: 0000008C    t2: 00000001    s0: 0010009F    s1: 00000080
a0: 00000000    a1: 00000000    a2: 00000000    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000000    s3: 00000000
s4: 00000000    s5: 00000000    s6: 00000000    s7: 00000000    s8: 00000000
s9: 00000000    s10:00000000    s11:00000000    t3: 00000000    t4: 00000000
t5: 00000000    t6: 00000008

rs1: 00 rs1_val: 00000000
rs2: 00 rs2_val: 00000000
rd:  05 reg_i_data: 00000001    reg_wen: 1

is_imm: 0       is_auipc: 0     is_lui: 0       imm: 00000000
a_val: 00000000 b_val: 00000000 alu_ctrl: F     cmp_ctrl: 0
alu_res: 00000000       cmp_res: 0

is_branch: 0    is_jal: 0       is_jalr: 0
do_branch: 0    pc_branch: 00000090

mem_wen: 0      mem_ren: 0
dmem_o_data: F0000000   dmem_i_data: 00000000   dmem_addr: 00000000

csr_wen: 0      csr_ind: 000    csr_ctrl: 0     csr_r_data: 00000000
mstatus: 00000001       mcause: 00000001        mepc: 00000080  mtval: 0010009F
mtvec:   0000008C       mie: 00000000   mip: 00000000
```

```
pc: 00000084    inst: 66600F93

x0: 00000000    ra: 00000000    sp: 00000007    gp: 00000000    tp: 0000008C
t0: 00000001    t1: 0000008C    t2: 00000000    s0: 0010009F    s1: 00000084
a0: 00000000    a1: 00000000    a2: 00000000    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000000    s3: 00000000
s4: 00000000    s5: 00000000    s6: 00000000    s7: 00000000    s8: 00000000
s9: 00000000    s10:00000000    s11:00000000    t3: 00000000    t4: 00000000
t5: 00000000    t6: 00000008

rs1: 00 rs1_val: 00000000
rs2: 06 rs2_val: 0000008C
rd:  1F reg_i_data: 00000666    reg_wen: 1

is_imm: 1       is_auipc: 0     is_lui: 0       imm: 00000666
a_val: 00000000 b_val: 00000666 alu_ctrl: 0     cmp_ctrl: 0
alu_res: 00000666       cmp_res: 0

is_branch: 0    is_jal: 0       is_jalr: 0
do_branch: 0    pc_branch: 00000088

mem_wen: 0      mem_ren: 0
dmem_o_data: 00000000   dmem_i_data: 008C0000   dmem_addr: 00000666

csr_wen: 0      csr_ind: 000    csr_ctrl: 0     csr_r_data: 00000000
mstatus: 00000000       mcause: 00000001        mepc: 00000084  mtval: 0010009F
mtvec:   0000008C       mie: 00000000   mip: 00000000
```

- 将sw[0]持续置为1,观察因为硬件中断进入trap程序后的过程,可以发现跳转回原来程序后,由于硬件中断仍然存在,又继续跳转到trap程序,与预期相符

```
pc: 00000028    inst: 003010F3

x0: 00000000    ra: 00000000    sp: 00000000    gp: 00000000    tp: 0000008C
t0: 00000000    t1: 00000000    t2: 00000000    s0: 00000000    s1: 00000000
a0: 00000000    a1: 00000000    a2: 00000000    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000000    s3: 00000000
s4: 00000000    s5: 00000000    s6: 00000000    s7: 00000000    s8: 00000000
s9: 00000000    s10:00000000    s11:00000000    t3: 00000000    t4: 00000000
t5: 00000000    t6: 00000001

rs1: 00 rs1_val: 00000000
rs2: 03 rs2_val: 00000000
rd:  01 reg_i_data: 00000001    reg_wen: 1

is_imm: 0       is_auipc: 0     is_lui: 0       imm: 00000000
a_val: 00000000 b_val: 00000000 alu_ctrl: F     cmp_ctrl: 0
alu_res: 00000000       cmp_res: 0

is_branch: 0    is_jal: 0       is_jalr: 0
do_branch: 0    pc_branch: 0000002C

mem_wen: 0      mem_ren: 0
dmem_o_data: F0000000   dmem_i_data: 00000000   dmem_addr: 00000000

csr_wen: 0      csr_ind: 000    csr_ctrl: 0     csr_r_data: 00000000
mstatus: 00000000       mcause: 00000000        mepc: 00000000  mtval: 00000001
mtvec:   0000008C       mie: 00000000   mip: 00000000
```

```
pc: 0000008C    inst: 000022F3

x0: 00000000    ra: 00000001    sp: 00000000    gp: 00000000    tp: 0000008C
t0: 00000000    t1: 00000000    t2: 00000000    s0: 00000000    s1: 00000000
a0: 00000000    a1: 00000000    a2: 00000000    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000000    s3: 00000000
s4: 00000000    s5: 00000000    s6: 00000000    s7: 00000000    s8: 00000000
s9: 00000000    s10:00000000    s11:00000000    t3: 00000000    t4: 00000000
t5: 00000000    t6: 00000001

rs1: 00 rs1_val: 00000000
rs2: 00 rs2_val: 00000000
rd:  05 reg_i_data: 00000001    reg_wen: 1

is_imm: 0       is_auipc: 0     is_lui: 0       imm: 00000000
a_val: 00000000 b_val: 00000000 alu_ctrl: F     cmp_ctrl: 0
alu_res: 00000000       cmp_res: 0

is_branch: 0    is_jal: 0       is_jalr: 0
do_branch: 0    pc_branch: 00000090

mem_wen: 0      mem_ren: 0
dmem_o_data: F0000000   dmem_i_data: 00000000   dmem_addr: 00000000

csr_wen: 0      csr_ind: 000    csr_ctrl: 0     csr_r_data: 00000000
mstatus: 00000001       mcause: 80000000        mepc: 00000028  mtval: 00000001
mtvec:   0000008C       mie: 00000000   mip: 00000000
```

```
pc: 00000028    inst: 003010F3

x0: 00000000    ra: 00000001    sp: 00000000    gp: 00000000    tp: 0000008C
t0: 00000001    t1: 0000008C    t2: 80000000    s0: 00000001    s1: 00000028
a0: 00000001    a1: 00000000    a2: 00000000    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000000    s3: 00000000
s4: 00000000    s5: 00000000    s6: 00000000    s7: 00000000    s8: 00000000
s9: 00000000    s10:00000000    s11:00000000    t3: 00000000    t4: 00000000
t5: 00000000    t6: 00000001

rs1: 00 rs1_val: 00000000
rs2: 03 rs2_val: 00000000
rd:  01 reg_i_data: 00000001    reg_wen: 1

is_imm: 0       is_auipc: 0     is_lui: 0       imm: 00000000
a_val: 00000000 b_val: 00000000 alu_ctrl: F     cmp_ctrl: 0
alu_res: 00000000       cmp_res: 0

is_branch: 0    is_jal: 0       is_jalr: 0
do_branch: 0    pc_branch: 0000002C

mem_wen: 0      mem_ren: 0
dmem_o_data: F0000000   dmem_i_data: 00000000   dmem_addr: 00000000

csr_wen: 0      csr_ind: 000    csr_ctrl: 0     csr_r_data: 00000000
mstatus: 00000000       mcause: 80000000        mepc: 00000028  mtval: 00000001
mtvec:   0000008C       mie: 00000000   mip: 00000000
```

```
pc: 0000008C    inst: 000022F3

x0: 00000000    ra: 00000001    sp: 00000000    gp: 00000000    tp: 0000008C
t0: 00000001    t1: 0000008C    t2: 80000000    s0: 00000001    s1: 00000028
a0: 00000001    a1: 00000000    a2: 00000000    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000000    s3: 00000000
s4: 00000000    s5: 00000000    s6: 00000000    s7: 00000000    s8: 00000000
s9: 00000000    s10:00000000    s11:00000000    t3: 00000000    t4: 00000000
t5: 00000000    t6: 00000001

rs1: 00 rs1_val: 00000000
rs2: 00 rs2_val: 00000000
rd:  05 reg_i_data: 00000001    reg_wen: 1

is_imm: 0       is_auipc: 0     is_lui: 0       imm: 00000000
a_val: 00000000 b_val: 00000000 alu_ctrl: F     cmp_ctrl: 0
alu_res: 00000000       cmp_res: 0

is_branch: 0    is_jal: 0       is_jalr: 0
do_branch: 0    pc_branch: 00000090

mem_wen: 0      mem_ren: 0
dmem_o_data: F0000000   dmem_i_data: 00000000   dmem_addr: 00000000

csr_wen: 0      csr_ind: 000    csr_ctrl: 0     csr_r_data: 00000000
mstatus: 00000001       mcause: 80000000        mepc: 00000028  mtval: 00000001
mtvec:   0000008C       mie: 00000000   mip: 00000000
```

- 最终x31寄存器值为0x666

```
pc: 0000001C    inst: FF1FF06F

x0: 00000000    ra: 00000000    sp: 00000007    gp: 00000000    tp: 0000008C
t0: 00000001    t1: 0000008C    t2: 00000000    s0: 0010009F    s1: 00000084
a0: 00000000    a1: 00000000    a2: 00000000    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000000    s3: 00000000
s4: 00000000    s5: 00000000    s6: 00000000    s7: 00000000    s8: 00000000
s9: 00000000    s10:00000000    s11:00000000    t3: 00000000    t4: 00000000
t5: 00000000    t6: 00000666

rs1: 1F rs1_val: 00000666
rs2: 11 rs2_val: 00000000
rd:  00 reg_i_data: 00000020    reg_wen: 1

is_imm: 0       is_auipc: 0     is_lui: 0       imm: FFFFFFF0
a_val: 00000666 b_val: 00000000 alu_ctrl: F     cmp_ctrl: 0
alu_res: 00000000       cmp_res: 0

is_branch: 0    is_jal: 1       is_jalr: 0
do_branch: 0    pc_branch: 0000000C

mem_wen: 0      mem_ren: 0
dmem_o_data: F0000000   dmem_i_data: 00000000   dmem_addr: 00000000

csr_wen: 0      csr_ind: 000    csr_ctrl: 0     csr_r_data: 00000000
mstatus: 00000000       mcause: 00000001        mepc: 00000084  mtval: 0010009F
mtvec:   0000008C       mie: 00000000   mip: 00000000
```

三、讨论、心得

思考题

```
lui t1, 0xDEADB
addi t1, t1, -273 // 0xEEF
```

该指令无法得到0xDEADBEEF的原因是addi指令将0xEEF扩展到32位的方法是以最高位进行扩展,所以扩展后的立即数是0xFFFFFEEF,与0xDEADB000相加后自然就无法得到0xDEADBEEF,此时高位B+F=A,并且产生进位,D+F+1=D,所以前面几个数字也就保持不变,得到结果为DEADAEEF,那么我们只需将DEADB改为DEADC,此时C+F=B,前面几位的运算结果不变,最终就能得到DEADBEEF

心得

本次实验属实是太折磨了,写完代码仿真就发现了各种问题,好不容易解决完仿真的问题后,上板又出现了新的问题,就一直在解决各种问题,所幸最终解决了问题。这段时间刚好又是期中周,所以最后也是抓紧时间刚好完成了,再接再厉吧,加油!