浙江大学

本科实验报告

课程名称：计算机体系结构

姓　　名：张天逸

学　　院：计算机科学与技术学院

系：计算机科学与技术系

专　　业：计算机科学与技术

学　　号：3220106424

指导教师：姜晓红

2024 年　10　　月　8　　日

浙江大学实验报告

课程名称：计算机体系结构 实验类型：综合

实验项目名称：Lab2：Pipelined CPU supporting exception & interrupt

学生姓名：张天逸 专业：计算机科学与技术 学号：3220106424

同组学生姓名： 指导老师：姜晓红

实验地点： 实验日期：2024 年 10 月 日

一、 实验目的和要求

理解 CPU 异常和中断的原理及其处理程序，包括何时发生中断和异常，以及如何在硬件层面处理它们。

掌握支持异常和中断的流水线 CPU 的设计方法，以及跳转到异常处理程序的过程。

掌握支持异常和中断的流水线 CPU 的程序验证方法，包括使用仿真测试中断引发的信号和 CSR 寄存器的值。

二、 实验内容和原理

CSR 寄存器模块

当 interrupt 信号有效时，处理器进入中断处理流程，更新 mstatus、mepc、mcause 和 mtval CSR。

当 mret 信号有效时，处理器从中断返回，恢复之前的状态。

csr_wsc_mode 是写操作的模式，可以是普通写入、或操作或与操作。

当 csr_w 信号有效时，根据 csr_wsc_mode 选择写入模式：

2'b01：正常写入。

2'b10：或操作。

2'b11：与非操作。

CSR.V

```
1.  `timescale 1ns / 1ps
2.
3.  module CSRRegs(
4.      input clk, rst,
5.      input[11:0] raddr, waddr,
6.      input[31:0] wdata,
7.      input csr_w,
8.      input[1:0] csr_wsc_mode,
9.      output[31:0] rdata,
10.     output[31:0] mstatus,
11.     output[31:0] mtvec,
12.     output[31:0] mepc,
13.     output[31:0] mie,
14.     input interrupt,
15.     input[31:0] mepc_w,
16.     input[31:0] mcause_w,
17.     input[31:0] mtval_w,
18.     input mret,
19.
20.     output[3:0] waddr_map
21. );
22.
23.     reg[31:0] CSR[0:15];
24.
25.     // Address mapping. The address is 12 bits, but only 4 bits are used in this module.
26.     wire raddr_valid = raddr[11:7] == 5'h6 && raddr[5:3] == 3'h0;
27.     wire[3:0] raddr_map = (raddr[6] << 3) + raddr[2:0];
28.     wire waddr_valid = waddr[11:7] == 5'h6 && waddr[5:3] == 3'h0;
29.
30.     assign waddr_map = (waddr[6] << 3) + waddr[2:0];
31.
32.     assign mstatus = CSR[0];
33.     assign mepc = CSR[9];
34.     assign mtvec = CSR[5];
35.     assign mie = CSR[4];
36.     assign rdata = CSR[raddr_map];
37.
38.     always@(posedge clk or posedge rst) begin
39.         if(rst) begin
40.   CSR[0] <= 32'h88;//mstatus
```

```
        CSR[1] <= 0;
        CSR[2] <= 0;
        CSR[3] <= 0;
        CSR[4] <= 32'hfff;//mie
        CSR[5] <= 0;//mtvec
        CSR[6] <= 0;
        CSR[7] <= 0;
        CSR[8] <= 0;
        CSR[9] <= 0;//mepc
        CSR[10] <= 0;//mcause
        CSR[11] <= 0;//mtval
        CSR[12] <= 0;
        CSR[13] <= 0;
        CSR[14] <= 0;
        CSR[15] <= 0;
    end
            else if(interrupt) begin
                CSR[0][7] <= CSR[0][3];//TO_BE_FILLED
                CSR[0][3] <= 0;
                CSR[9] <= mepc_w;//TO_BE_FILLED
                CSR[10] <= mcause_w;//TO_BE_FILLED
                CSR[11] <= mtval_w;//TO_BE_FILLED
            end
            else if(mret) begin
                CSR[0][3] <= CSR[0][7];//TO_BE_FILLED
                CSR[0][7] <= 1;
            end
            else if(csr_w) begin
                case(csr_wsc_mode)
                    2'b01: CSR[waddr_map] = wdata;
                    2'b10: CSR[waddr_map] = CSR[waddr_map] | wdata;
                    2'b11: CSR[waddr_map] = CSR[waddr_map] & (~wdata);
                    default: CSR[waddr_map] = wdata;
                endcase
            end
        end
endmodule
```

Exception Unit 模块

interrupt_or_exception 信号表示是否有中断或异常发生。

mepc_w、mcause_w、mtval_w 是用于写入异常处理相关信息的内部信号。

exception_index 根据异常原因信号确定异常的索引值。

PC_redirect 根据 mret 或 interrupt_or_exception 信号决定 PC 的重定向地址。

当检测到异常或中断时，根据异常类型设置 mcause_w 和 mtval_w。

如果是非法指令异常，exception_index 被设置为 2；如果是加载访问故障，被设置为 5；如果是存储访问故障，被设置为 7；如果是环境调用，被设置为 11。

EXCEPTIONUNIT.V

```
`timescale 1ns / 1ps

module ExceptionUnit(
    input clk, rst,
    input csr_rw_in,
    // write/set/clear (funct bits from instruction)
    input[1:0] csr_wsc_mode_in,
    input csr_w_imm_mux,
    input[11:0] csr_rw_addr_in,
    input[31:0] csr_w_data_reg,
    input[4:0] csr_w_data_imm,
    output[31:0] csr_r_data_out,

    input interrupt,
    input illegal_inst,
    input l_access_fault,
    input s_access_fault,
    input ecall_m,

    input mret,

    input[31:0] epc_cur,
    input[31:0] epc_next,
    output[31:0] PC_redirect,
    output redirect_mux,

    output reg_FD_flush, reg_DE_flush, reg_EM_flush, reg_MW_flush,
    output RegWrite_cancel
);

    reg[11:0] csr_waddr;
    reg[31:0] csr_wdata;
    reg csr_w;
    reg[1:0] csr_wsc;
```

```
35.         wire[11:0] csr_raddr;
36.         wire[31:0] csr_rdata;
37.
38.         wire[31:0] mstatus;
39.         wire[31:0] mepc;
40.         wire[31:0] mtvec;
41.         wire[31:0] mie;
42.
43.         wire interrupt_or_exception;
44.         wire[31:0] mepc_w;
45.         wire[31:0] mcause_w;
46.         wire[31:0] mtval_w;
47.
48.         wire[3:0] waddr_map;
49.         wire enable_exception;
50.
51.         wire [3:0] exception_index;
52.
53.         assign exception_index = illegal_inst ? 4'd2 :
54.                                  l_access_fault ? 4'd5 :
55.                                  s_access_fault ? 4'd7 :
56.                                  ecall_m ? 4'd11 : 4'd0; // 默认是0，如果没有异常
57.
58.
59.         assign enable_exception = mstatus[3];
60.
61.         assign interrupt_or_exception = illegal_inst | l_access_fault | s_access_fault | ecall_m | int
    errupt;
62.
63.         assign reg_FD_flush = interrupt_or_exception | mret;
64.         assign reg_DE_flush = interrupt_or_exception | mret;
65.         assign reg_EM_flush = interrupt_or_exception | mret;
66.         assign reg_MW_flush = interrupt_or_exception | mret;
67.         assign RegWrite_cancel = l_access_fault ;
68.         assign redirect_mux = interrupt_or_exception | mret;
69.
70.         assign PC_redirect = mret ? mepc : interrupt_or_exception ? mtvec : 32'b0;
71.         assign csr_raddr = csr_rw_addr_in;
72.         assign mepc_w = (illegal_inst | l_access_fault | s_access_fault | ecall_m) ? epc_cur : interru
    pt ? epc_next : mepc;
73.         assign mcause_w = {interrupt,{27{1'b0}},exception_index};
74.         assign mtval_w = epc_cur;
75.         assign csr_r_data_out = csr_rdata;
76.
```

```
77.       CSRRegs csr(.clk(clk),.rst(rst),.csr_w(csr_w),.raddr(csr_raddr),.waddr(csr_waddr),
78.           .wdata(csr_wdata),.rdata(csr_rdata),.mstatus(mstatus),.csr_wsc_mode(csr_wsc),
79.           .mtvec(mtvec),.mepc(mepc),.interrupt(interrupt_or_exception),.mepc_w(mepc_w),
80.           .mcause_w(mcause_w),.mtval_w(mtval_w),.mret(mret),.waddr_map(waddr_map),.mie(mie));
81.
82.       always @ (negedge clk or posedge rst) begin
83.           if(rst) begin
84.             csr_waddr<=0;
85.             csr_wdata<=0;
86.             csr_w<=0;
87.             csr_wsc<=0;
88.           end
89.           else begin
90.               if(csr_rw_in) begin
91.                   csr_waddr <= csr_rw_addr_in;
92.                   csr_wdata <= csr_w_imm_mux ? {{27{1'b0}},csr_w_data_imm} : csr_w_data_reg;
93.                   csr_w <= 1;
94.                   csr_wsc <= csr_wsc_mode_in;
95.               end
96.               else begin
97.                   csr_waddr<=0;
98.                   csr_wdata<=0;
99.                   csr_w<=0;
100.                  csr_wsc<=0;
101.              end
102.          end
103.      end
104.  endmodule
```

Control Unit 模块

exp_vector 表示异常向量，目前只处理非法指令和环境调用（ECALL）。

csr_rw 信号表示是否进行 CSR 读写操作。

csr_w_imm_mux 信号用于选择 CSR 写入数据的来源。

illegal_inst 信号用于指示当前指令是否合法。

CONTROLUNIT.V

```
1.    `timescale 1ns / 1ps
2.
3.
```

```
module CtrlUnit(
    input[31:0] inst,
    input cmp_res,
    output Branch, ALUSrc_A, ALUSrc_B, DatatoReg, RegWrite, mem_w,
        mem_r, rs1use, rs2use,
    output [1:0] hazard_optype,
    output [2:0] ImmSel, cmp_ctrl,
    output [3:0] ALUControl,
    output JALR, MRET,

    output csr_rw, csr_w_imm_mux,

    output[1:0] exp_vector
);

    wire[6:0] funct7 = inst[31:25];
    wire[2:0] funct3 = inst[14:12];
    wire[6:0] opcode = inst[6:0];

    wire Rop = opcode == 7'b0110011;
    wire Iop = opcode == 7'b0010011;
    wire Bop = opcode == 7'b1100011;
    wire Lop = opcode == 7'b0000011;
    wire Sop = opcode == 7'b0100011;
    wire CSRop = opcode == 7'b1110011;

    wire funct7_0  = funct7 == 7'h0;
    wire funct7_32 = funct7 == 7'h20;

    wire funct3_0 = funct3 == 3'h0;
    wire funct3_1 = funct3 == 3'h1;
    wire funct3_2 = funct3 == 3'h2;
    wire funct3_3 = funct3 == 3'h3;
    wire funct3_4 = funct3 == 3'h4;
    wire funct3_5 = funct3 == 3'h5;
    wire funct3_6 = funct3 == 3'h6;
    wire funct3_7 = funct3 == 3'h7;

    wire ADD  = Rop & funct3_0 & funct7_0;
    wire SUB  = Rop & funct3_0 & funct7_32;
    wire SLL  = Rop & funct3_1 & funct7_0;
    wire SLT  = Rop & funct3_2 & funct7_0;
    wire SLTU = Rop & funct3_3 & funct7_0;
    wire XOR  = Rop & funct3_4 & funct7_0;
```

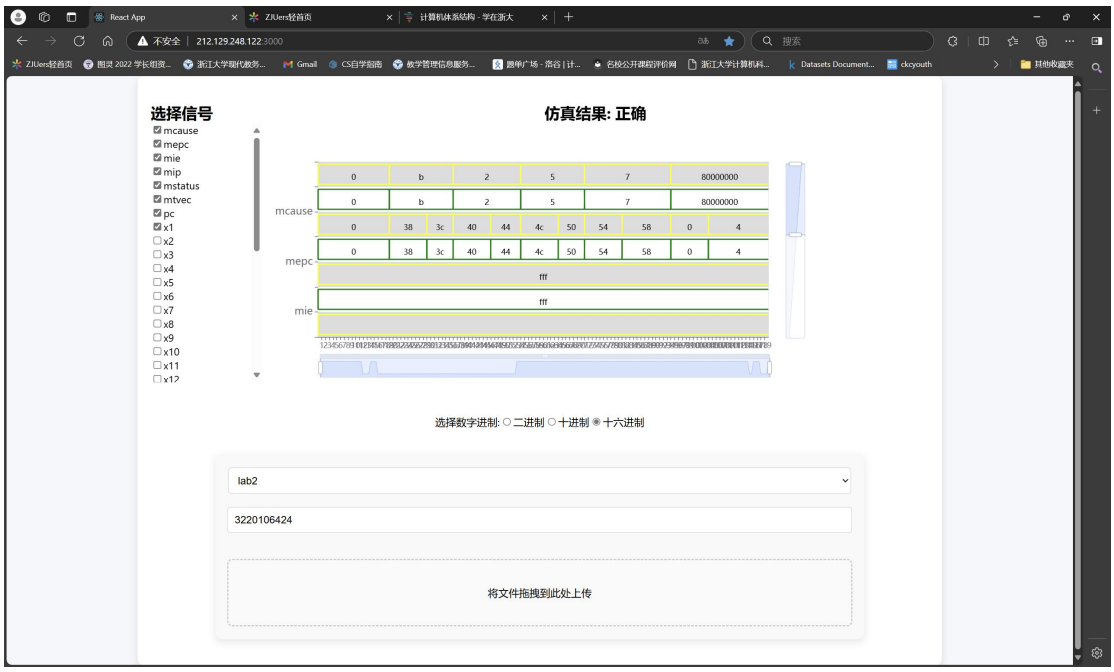
```
48.         wire SRL  = Rop & funct3_5 & funct7_0;
49.         wire SRA  = Rop & funct3_5 & funct7_32;
50.         wire OR   = Rop & funct3_6 & funct7_0;
51.         wire AND  = Rop & funct3_7 & funct7_0;
52.
53.         wire ADDI  = Iop & funct3_0;
54.         wire SLTI  = Iop & funct3_2;
55.         wire SLTIU = Iop & funct3_3;
56.         wire XORI  = Iop & funct3_4;
57.         wire ORI   = Iop & funct3_6;
58.         wire ANDI  = Iop & funct3_7;
59.         wire SLLI  = Iop & funct3_1 & funct7_0;
60.         wire SRLI  = Iop & funct3_5 & funct7_0;
61.         wire SRAI  = Iop & funct3_5 & funct7_32;
62.
63.         wire BEQ = Bop & funct3_0;
64.         wire BNE = Bop & funct3_1;
65.         wire BLT = Bop & funct3_4;
66.         wire BGE = Bop & funct3_5;
67.         wire BLTU = Bop & funct3_6;
68.         wire BGEU = Bop & funct3_7;
69.
70.         wire LB =  Lop & funct3_0;
71.         wire LH =  Lop & funct3_1;
72.         wire LW =  Lop & funct3_2;
73.         wire LBU = Lop & funct3_4;
74.         wire LHU = Lop & funct3_5;
75.
76.         wire SB = Sop & funct3_0;
77.         wire SH = Sop & funct3_1;
78.         wire SW = Sop & funct3_2;
79.
80.         wire LUI   = opcode == 7'b0110111;
81.         wire AUIPC = opcode == 7'b0010111;
82.
83.         wire JAL  =  opcode == 7'b1101111;
84.         assign JALR = (opcode == 7'b1100111) && funct3_0;
85.
86.         wire CSRRW = CSRop & funct3_1;
87.         wire CSRRS = CSRop & funct3_2;
88.         wire CSRRC = CSRop & funct3_3;
89.         wire CSRRWI = CSRop & funct3_5;
90.         wire CSRRSI = CSRop & funct3_6;
91.         wire CSRRCI = CSRop & funct3_7;
```

```
92.
93.     assign MRET = inst == 32'b0011000_00010_00000_000_00000_1110011;
94.     wire ECALL = inst == 32'b0111_0011;
95.
96.     wire R_valid = AND | OR | ADD | XOR | SLL | SRL | SRA | SUB | SLT | SLTU;
97.     wire I_valid = ANDI | ORI | ADDI | XORI | SLLI | SRLI | SRAI | SLTI | SLTIU;
98.     wire B_valid = BEQ | BNE | BLT | BGE | BLTU | BGEU;
99.     wire L_valid = LW | LH | LB | LHU | LBU;
100.    wire S_valid = SW | SH | SB;
101.    wire CSR_valid = CSRRW | CSRRS | CSRRC | CSRRWI | CSRRSI | CSRRCI;
102.
103.
104.    assign Branch = JAL | JALR | B_valid & cmp_res;
105.
106.    localparam Imm_type_I = 3'b001;
107.    localparam Imm_type_B = 3'b010;
108.    localparam Imm_type_J = 3'b011;
109.    localparam Imm_type_S = 3'b100;
110.    localparam Imm_type_U = 3'b101;
111.    assign ImmSel = {3{I_valid | JALR | L_valid}} & Imm_type_I |
112.                    {3{B_valid}}                  & Imm_type_B |
113.                    {3{JAL}}                      & Imm_type_J |
114.                    {3{S_valid}}                  & Imm_type_S |
115.                    {3{LUI | AUIPC}}              & Imm_type_U ;
116.
117.    localparam cmp_EQ  = 3'b001;
118.    localparam cmp_NE  = 3'b010;
119.    localparam cmp_LT  = 3'b011;
120.    localparam cmp_LTU = 3'b100;
121.    localparam cmp_GE  = 3'b101;
122.    localparam cmp_GEU = 3'b110;
123.    assign cmp_ctrl = {3{BEQ }} & cmp_EQ  |
124.                      {3{BNE }} & cmp_NE  |
125.                      {3{BLT }} & cmp_LT  |
126.                      {3{BLTU}} & cmp_LTU |
127.                      {3{BGE }} & cmp_GE  |
128.                      {3{BGEU}} & cmp_GEU ;
129.
130.    assign ALUSrc_A = JAL | JALR | AUIPC;
131.
132.    assign ALUSrc_B = I_valid | L_valid | S_valid | LUI | AUIPC;
133.
134.    localparam ALU_ADD  = 4'b0001;
135.    localparam ALU_SUB  = 4'b0010;
```

```
136.     localparam ALU_AND  = 4'b0011;
137.     localparam ALU_OR   = 4'b0100;
138.     localparam ALU_XOR  = 4'b0101;
139.     localparam ALU_SLL  = 4'b0110;
140.     localparam ALU_SRL  = 4'b0111;
141.     localparam ALU_SLT  = 4'b1000;
142.     localparam ALU_SLTU = 4'b1001;
143.     localparam ALU_SRA  = 4'b1010;
144.     localparam ALU_Ap4  = 4'b1011;
145.     localparam ALU_Bout = 4'b1100;
146.     assign ALUControl = {4{ADD | ADDI | L_valid | S_valid | AUIPC}} & ALU_ADD  |
147.                         {4{SUB}}                                    & ALU_SUB  |
148.                         {4{AND | ANDI}}                             & ALU_AND  |
149.                         {4{OR | ORI}}                               & ALU_OR   |
150.                         {4{XOR | XORI}}                             & ALU_XOR  |
151.                         {4{SLL | SLLI}}                             & ALU_SLL  |
152.                         {4{SRL | SRLI}}                             & ALU_SRL  |
153.                         {4{SLT | SLTI}}                             & ALU_SLT  |
154.                         {4{SLTU | SLTIU}}                           & ALU_SLTU |
155.                         {4{SRA | SRAI}}                             & ALU_SRA  |
156.                         {4{JAL | JALR}}                             & ALU_Ap4  |
157.                         {4{LUI}}                                    & ALU_Bout ;
158. 
159.     assign DatatoReg = L_valid | CSR_valid;
160. 
161.     assign RegWrite = R_valid | I_valid | JAL | JALR | L_valid | LUI | AUIPC | CSR_valid;
162. 
163.     assign mem_w = S_valid;
164. 
165.     assign mem_r = L_valid;
166. 
167.     assign rs1use = R_valid | I_valid | B_valid | JALR | L_valid | S_valid |
168.                     CSRRW | CSRRS | CSRRC ;
169. 
170.     assign rs2use = R_valid | B_valid | S_valid;
171. 
172.     localparam hazard_optype_ALU = 2'd1;
173.     localparam hazard_optype_LOAD = 2'd2;
174.     localparam hazard_optype_STORE = 2'd3;
175.     assign hazard_optype = {2{R_valid | I_valid | JAL | JALR | LUI | AUIPC}}
176.                                                 & hazard_optype_ALU  |
177.                            {2{L_valid | CSR_valid}} & hazard_optype_LOAD |
178.                            {2{S_valid}}             & hazard_optype_STORE;
179. 
```

```
    assign csr_rw = CSR_valid;

    assign csr_w_imm_mux = CSRRWI | CSRRSI | CSRRCI ;

    wire illegal_inst = ~(R_valid | I_valid | B_valid | JAL | JALR | L_valid | S_valid |
        LUI | AUIPC | CSR_valid | MRET | ECALL);

    assign exp_vector = {illegal_inst, ECALL};//TO_BE_FILLED

endmodule
```

三、 实验过程和数据记录及结果分析



在平台上进行仿真，结果正确

PC=18，指令：csrrwi x1, 0x306, 16；这条指令将立即数 16 写入 CSR 寄存器 0x306。

PC=1C，指令：csrr x1, 0x306；这条指令将 CSR 寄存器 0x306 的内容读取并写入 x1。观察仿真结果，确实看到 x1 被赋值为 16

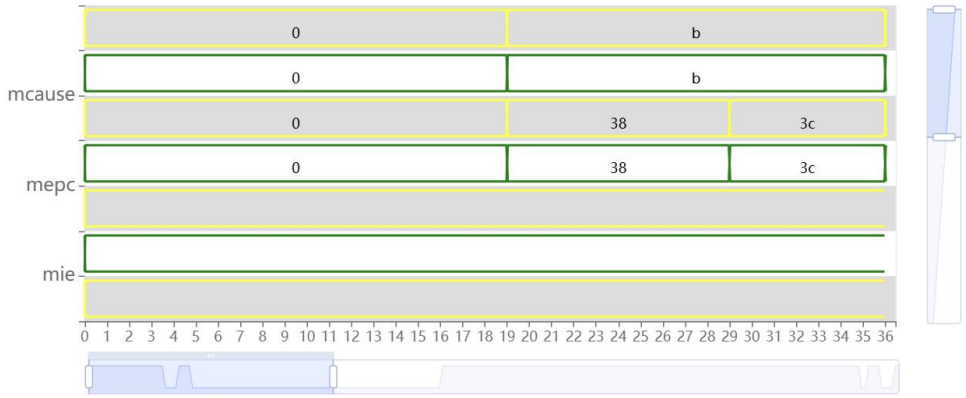
PC=20，指令：csrrw x1, 0x306, x6；这条指令将寄存器 x6 的值写入 CSR 寄存器 0x306。

PC=24，指令：csrr x1, 0x306；这条指令将 CSR 寄存器 0x306 的内容读取并写入 x1。观察仿真结果，x1 被赋值为 0xffff0000

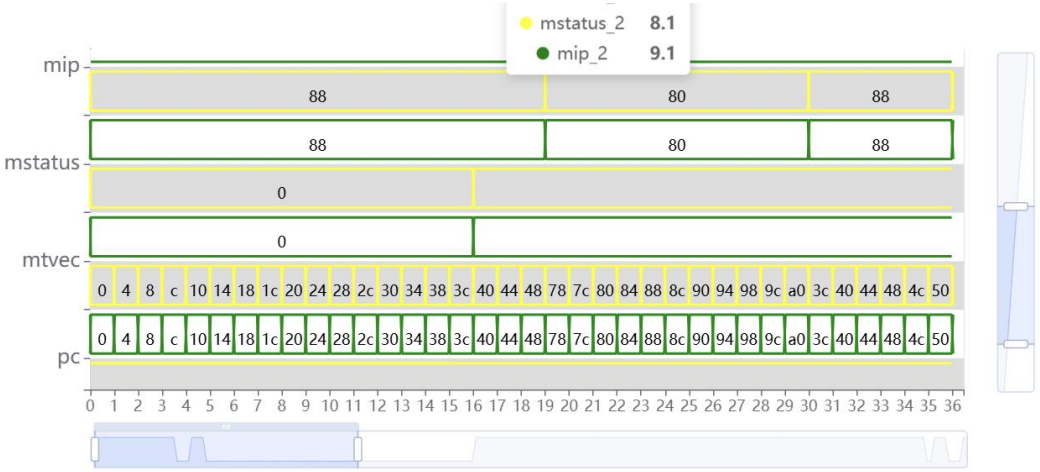
PC=30，指令：csrw 0x305, x1；这条指令将寄存器 x1 的值写入 CSR 寄存器 0x305。

PC=38 时进入指令 ecall 跳转到 0x78 地址（trap）处

仿真结果: 正确

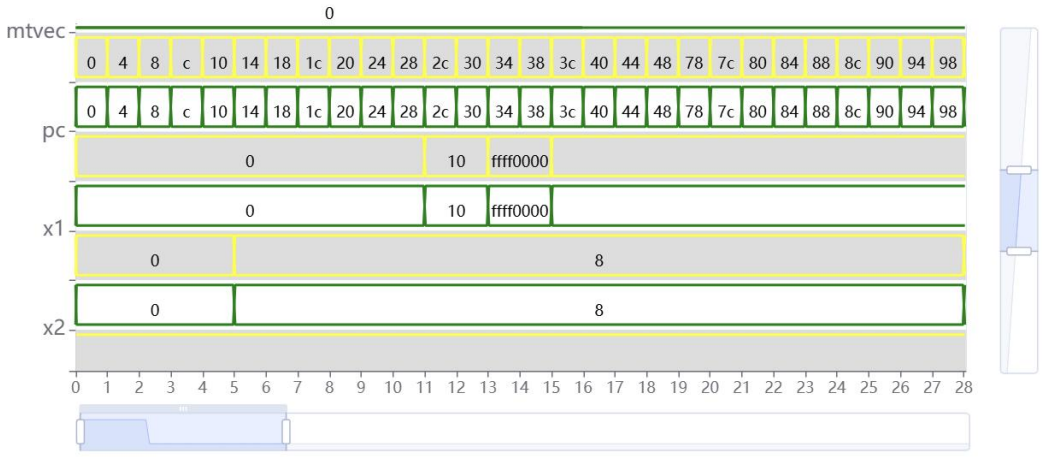


选择数字进制: ○ 二进制 ○ 十进制 ◉ 十六进制



选择数字进制: ○ 二进制 ○ 十进制 ◉ 十六进制

仿真结果: 正确



PC=78，指令：csrr x25, 0x341 ；csrr 指令读取 CSR 寄存器 0x341（即 mepc，Machine Exception Program Counter）的值，并将该值存储在寄存器 x25 中。

PC=7C，指令：csrr x27, 0x342 ；csrr 指令读取 CSR 寄存器 0x342（即 mcause，Machine Cause）的值，并将该值存储在寄存器 x27 中。mcause 用于指示导致异常或中断的原因。

PC=80，指令：csrr x28, 0x300 ；csrr 指令读取 CSR 寄存器 0x300（即 mstatus，Machine Status）的值，并将该值存储在寄存器 x28 中。mstatus 寄存器包含处理器的状态信息，如特权模式、中断使能等。

PC=84，指令：csrr x29, 0x304 ；csrr 指令读取 CSR 寄存器 0x304（即 mie，Machine Interrupt Enable）的值，并将该值存储在寄存器 x29 中。mie 寄存器用于控制哪些中断可以被处理器识别。

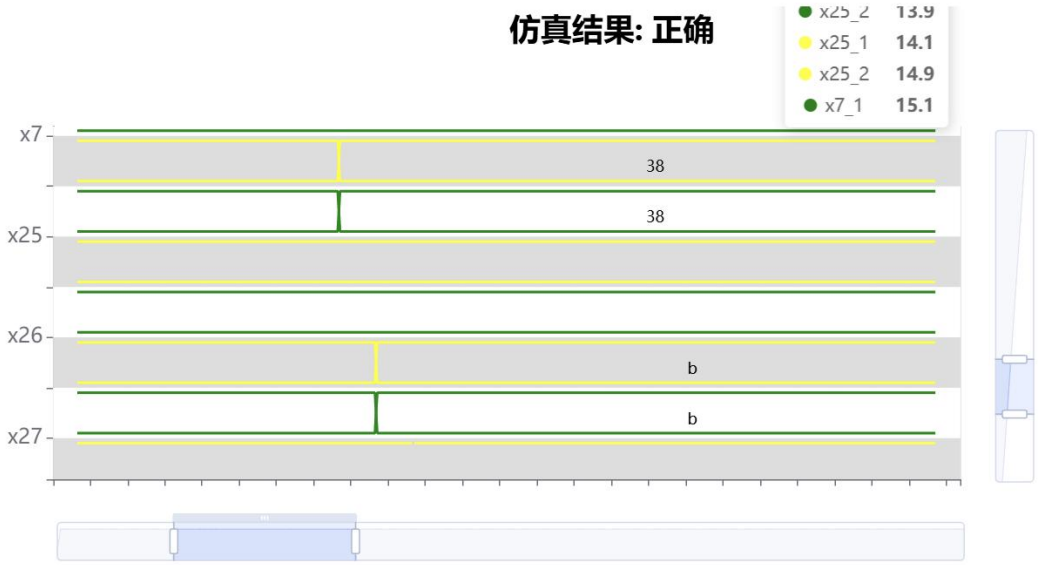
PC=88，指令：csrr x30, 0x344 ；csrr 指令读取 CSR 寄存器 0x344（即 mip，Machine Interrupt Pending）的值，并将该值存储在寄存器 x30 中。mip 寄存器包含当前待处理中断的标记。

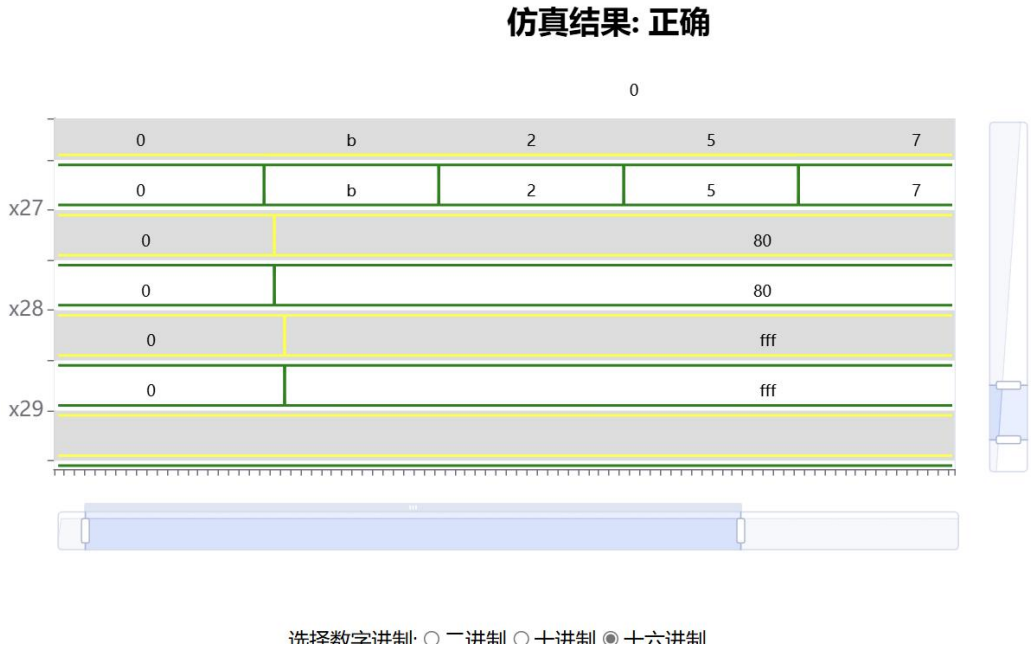
PC=90，指令：csrw 0x341, x2 ；csrw 指令将寄存器 x2 的值（更新后的 PC 值）写入 CSR 寄存器 0x341（即 mepc）。这实际上是更新 mepc 寄存器，以便在中断或异常处理完成后，处理器可以从正确的地址继续执行。

PC=94，指令：mret；mret 指令用于从中断或异常返回。它会使处理器恢复 mstatus、mepc 和 mie 寄存器的值，并将控制权返回到发生异常或中断的代码

位置。

通过对比 mcause，mepc，mie，mip，mstatus，mtvec，x25-x30 的值证明代码正确。



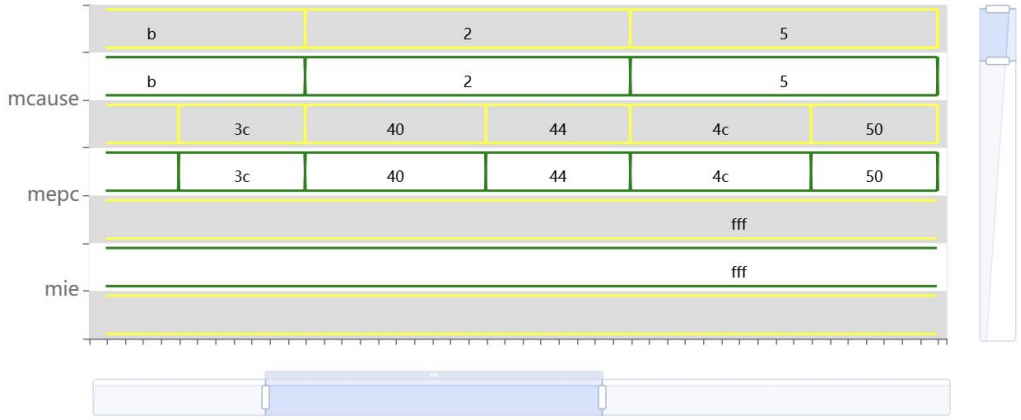


由于 mret 回到中断执行，因而返回至 PC=3C 出继续执行

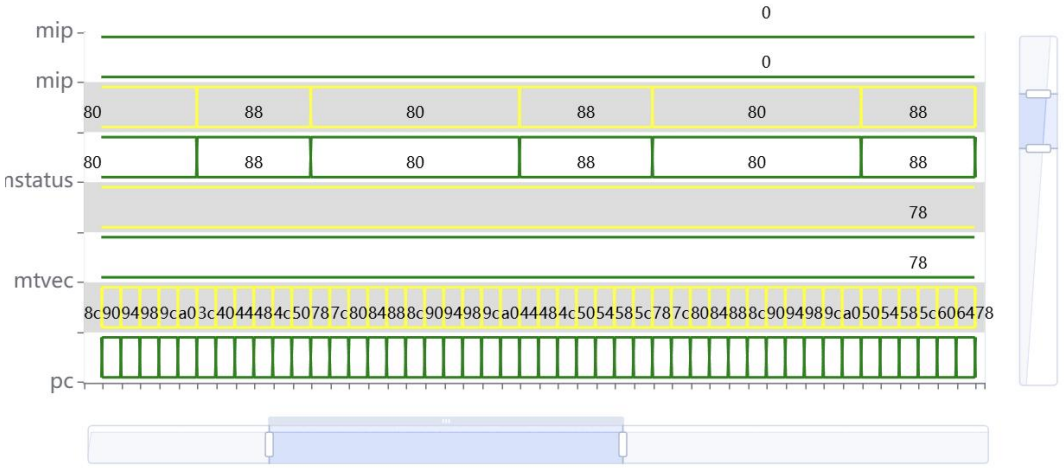
PC=40，指令：illegal ；程序检测到了异常并准备跳转至处理程序，几个周期后跳跃至 trap（0x78）依次是读取 mepc， mcause，mstatus ， mtval，mepc 为 0x40， mcause 为 2， mstatus 为 0x80，即非法指令，符合预期。最后 mret 返回 PC=44 处

PC=4C，指令：lw　x1, 128(x0)　；这条指令访问的地址超过了范围，因而进入 trap，观察 mepc， mcause，mstatus ， mtval，mepc 为 0x4C， mcause 为 5， mstatus 为 0x80， 最后 mret 返回 PC=50 处

仿真结果: 正确

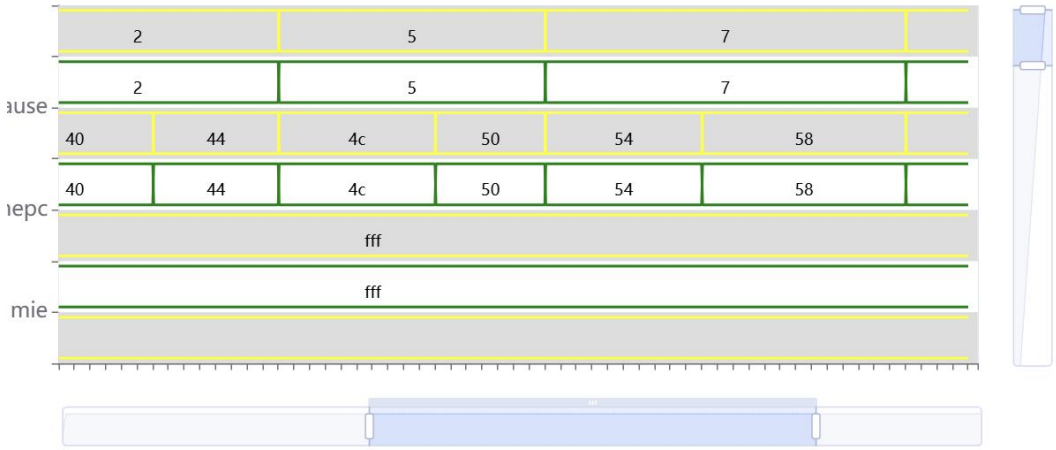


仿真结果: 正确



PC=54，指令：sw　x1, 128(x0)　；这条指令访问了非法的地址范围，因而进入 trap，观察 mepc， mcause，mstatus ， mtval，mepc 为 0x54， mcause 为 7， mstatus 为 0x80， 最后 mret 返回 PC=58 处

仿真结果: 正确



仿真结果: 正确

四、 讨论与心得

本次实验内容涉及到了 RISC-V 汇编语言的非法指令以及控制和状态寄存器（CSR）的操作。主要围绕非法指令异常处理等内容展开，需要填写部分相较上一次少了很多，真好。