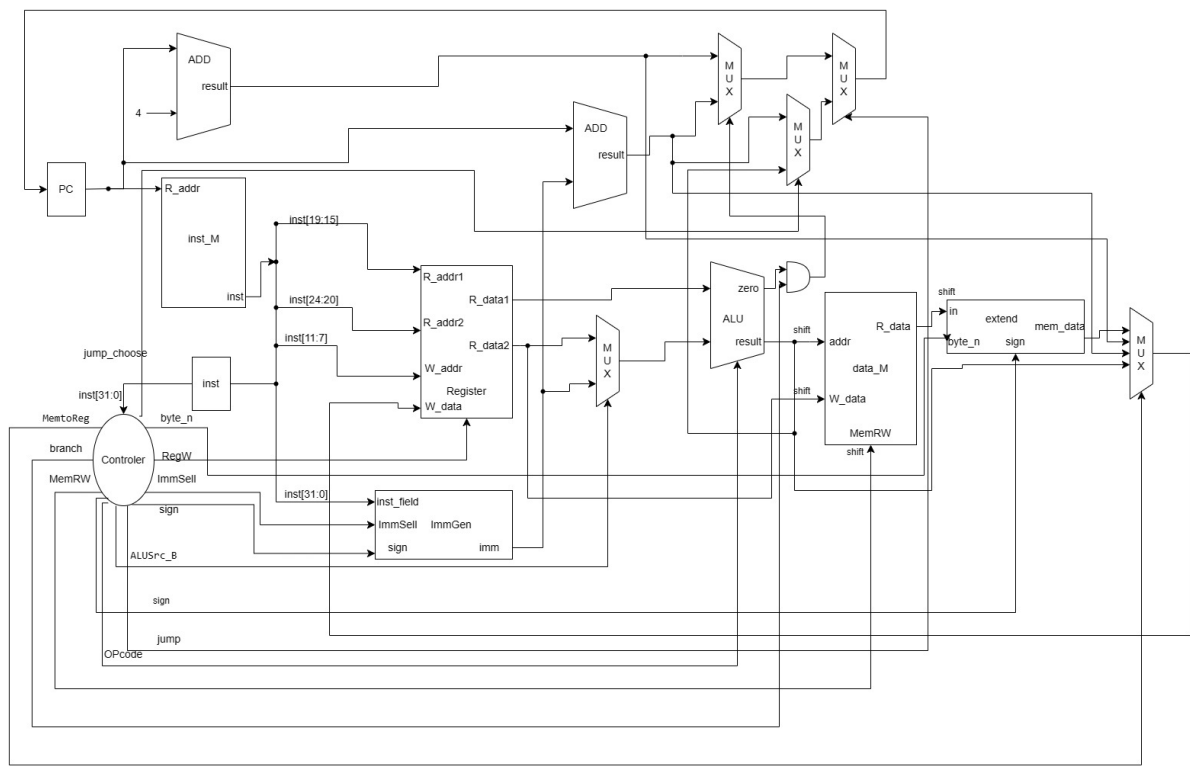


浙江大学实验报告 Lab4-3

专业：计算机科学与技术 姓名：仇国智 学号：3220102181 日期：2024/4/10
课程名称：计算机组成与设计 实验名称:实现单周期 CPU-指令拓展 指导老师：刘海风 成绩：

操作方法与实验步骤

绘制Datapath图



源文件编写

将原来lab2的工程复制一份,删除 SCPU 模块,依据原理图,自行设计 SCPU 模块(下辖 ALU , Controller , Datapath , extend , ImmGEN , Regs 模块, ALU , Regs 已经在前面实验实现,可直接复制)

SCPU 模块

SCPU 模块主要连接 Controller 和 Datapath 模块,并且包含了 Inst 指令的寄存器(实际上应该把 Regs 和 PC 的寄存器都拉到 Datapath ,这样结构会更加清晰),还计算了 MemRW (实际上这个变量没有必要存在),同时对 Data_out 和 wea 进行了左位移偏移处理(因为 RAM 的地址是四字字节对齐的,所以会对 Addr_out 造成截断, Data_out 和 wea 需要与截断后的 Addr_out 对齐,即代码 30 和 31 行).

```
1  `timescale 1ns / 1ps
2  `include "Lab4.vh"
3  module SCPU (
4      `RegFile_Regs_Outputs
5      input clk,
6      input rst,
7      input MIO_ready,
8      input [31:0] inst_in,
9      input [31:0] Data_in,
```

```

10     output CPU_MIO,
11     output MemRW,
12     output wire [31:0] PC_out,
13     output [31:0] Data_out,
14     output [31:0] Addr_out,
15     output wire [3:0] wea
16 );
17     wire [`IMM_SEL_WIDTH-1:0] ImmSel1;
18     wire ALUSrc_B;
19     wire [`MEM2REG_WIDTH-1:0] MemtoReg;
20     wire Jump;
21     wire Branch;
22     wire RegWrite;
23     wire sign;
24     wire jump_choose;
25     // 0 1byte, 1 2byte, 2 4byte
26     wire [1:0] byte_n;
27     assign MemRW = |wea;
28     wire [ 3:0] wea_temp;
29     wire [31:0] Data_out_temp;
30     assign wea = wea_temp << (Addr_out % 4);
31     assign Data_out = MemRW ? (Data_out_temp << ((Addr_out % 4) << 3)) :
Data_out_temp;
32     wire [`ALU_OP_WIDTH-1:0] ALU_Control;
33     reg [31:0] inst;
34     always @(posedge clk or posedge rst) begin
35         if (rst) begin
36             inst <= 32'h0;
37         end else begin
38             inst <= inst_in;
39         end
40     end
41     Controller v1 (
42         .OPCode(inst[6:2]),
43         .MIO_ready(MIO_ready),
44         .Fun7(inst[31:25]),
45         .Fun3(inst[14:12]),
46
47         .wea(wea_temp),
48         .CPU_MIO(CPU_MIO),
49         .ImmSel1(ImmSel1),
50         .ALUSrc_B(ALUSrc_B),
51         .MemtoReg(MemtoReg),
52         .Jump(Jump),
53         .Branch(Branch),
54         .RegWrite(RegWrite),
55         .ALU_Control(ALU_Control),
56         .sign(sign),
57         .byte_n(byte_n),
58         .jump_choose(jump_choose)
59     );
60     DataPath v2 (
61         `RegFile_Regs_Arguments
62         .clk(clk),
63         .rst(rst),
64         .inst_field(inst),

```

```

65         .Data_in(Data_in),
66         .ImmSell(ImmSell),
67         .ALUSrc_B(ALUSrc_B),
68         .MemtoReg(MemtoReg),
69         .Jump(Jump),
70         .Branch(Branch),
71         .RegWrite(RegWrite),
72         .ALU_Control(ALU_Control),
73         .sign(sign),
74         .byte_n(byte_n),
75         .jump_choose(jump_choose),
76
77         .PC_out (PC_out),
78         .Data_out(Data_out_temp),
79         .Addr_out(Addr_out)
80     );
81     endmodule

```

Controler 模块

Controler :根据指令不同要求解码即可,下面是各个输出属性的解释.代码很繁杂,或许可以通过将指令的不同输入组合然后存入一个 **ROM** 中,然后通过 **OPcode** 和 **Fun7** 和 **Fun3** 确定指令索引以读入 **ROM** 中的数据,但这样会降低代码可读性.

- **wea** : 写入地址选择
- **ImmSell** : 立即数模式选择
- **MemtoReg** : 写入寄存器数据来源选择
- **ALU_Control** : ALU控制信号
- **ALUSrc_B** : ALU第二个操作数来源选择
- **Jump** : 是否跳转
- **Branch** : 是否分支
- **RegWrite** : 是否写入寄存器
- **sign** : 字节/半字读入时是否进行有符号扩展
- **byte_n** : 读入字节数
- **jump_choose** : 跳转地址选择(PC+立即数/ALU输出)

```

1  `include "Lab4.vh"
2  module Controler (
3      input wire [4:0] OPcode,
4      input wire MIO_ready,
5      input wire [6:0] Fun7,
6      input wire [2:0] Fun3,
7      output reg CPU_MIO,
8      output reg [3:0] wea,
9      output reg [`IMM_SEL_WIDTH-1:0] ImmSell,
10     output reg [`MEM2REG_WIDTH-1:0] MemtoReg,
11     output reg [`ALU_OP_WIDTH-1:0] ALU_Control,
12     output reg ALUSrc_B,
13     output reg Jump,
14     output reg Branch,

```

```

15     output reg Regwrite,
16     output reg sign,
17     output reg [1:0] byte_n,
18     output reg jump_choose
19 );
20 always @(*) begin
21     case (OPCode)
22     `OPCODE_ALU: begin
23         CPU_MIO <= 1'b0;
24         wea <= `WEA_READ;
25         ImmSel1 <= 3'b0;
26         MemtoReg <= `MEM2REG_ALU;
27         case (Fun3)
28             `FUNC_ADD: ALU_Control <= Fun7[5] ? `ALU_OP_SUB :
`ALU_OP_ADD;
29             `FUNC_SL: ALU_Control <= `ALU_OP_SLL;
30             `FUNC_SLT: ALU_Control <= `ALU_OP_SLT;
31             `FUNC_SLTU: ALU_Control <= `ALU_OP_SLTU;
32             `FUNC_XOR: ALU_Control <= `ALU_OP_XOR;
33             `FUNC_OR: ALU_Control <= `ALU_OP_OR;
34             `FUNC_AND: ALU_Control <= `ALU_OP_AND;
35             `FUNC_SR: ALU_Control <= Fun7[5] ? `ALU_OP_SRA : `ALU_OP_SRL;
36             default: ALU_Control <= 4'b0;
37         endcase
38         ALUSrc_B <= 1'b0;
39         Jump <= 1'b0;
40         Branch <= 1'b0;
41         Regwrite <= 1'b1;
42         sign <= 1'b0;
43         byte_n <= `WORD;
44         jump_choose <= `JUMP_PC_IMM;
45     end
46     `OPCODE_ALU_IMM: begin
47         CPU_MIO <= 1'b0;
48         wea <= `WEA_READ;
49         ImmSel1 <= `IMM_SEL_I;
50         MemtoReg <= `MEM2REG_ALU;
51         case (Fun3)
52             `FUNC_ADD: ALU_Control <= `ALU_OP_ADD;
53             `FUNC_SL: ALU_Control <= `ALU_OP_SLL;
54             `FUNC_SLT: ALU_Control <= `ALU_OP_SLT;
55             `FUNC_SLTU: ALU_Control <= `ALU_OP_SLTU;
56             `FUNC_XOR: ALU_Control <= `ALU_OP_XOR;
57             `FUNC_OR: ALU_Control <= `ALU_OP_OR;
58             `FUNC_AND: ALU_Control <= `ALU_OP_AND;
59             `FUNC_SR: ALU_Control <= Fun7[5] ? `ALU_OP_SRA : `ALU_OP_SRL;
60             default: ALU_Control <= 4'b0;
61         endcase
62         ALUSrc_B <= 1'b1;
63         Jump <= 1'b0;
64         Branch <= 1'b0;
65         Regwrite <= 1'b1;
66         sign <= 1'b1;
67         byte_n <= `WORD;
68         jump_choose <= `JUMP_PC_IMM;
69     end

```

```

70     `OPCODE_LOAD: begin
71         CPU_MIO <= 1'b1;
72         wea <= `WEA_READ;
73         ImmSel1 <= `IMM_SEL_I;
74         MemtoReg <= `MEM2REG_MEM;
75         ALU_Control <= 4'b0;
76         ALUSrc_B <= 1'b1;
77         Jump <= 1'b0;
78         Branch <= 1'b0;
79         Regwrite <= 1'b1;
80         sign <= ~(Fun3 == `FUNC_BYTE_UNSIGNED || Fun3 ==
`FUNC_HALF_UNSIGNED);
81         case (Fun3)
82             `FUNC_BYTE, `FUNC_BYTE_UNSIGNED: byte_n <= `BYTE;
83             `FUNC_HALF, `FUNC_HALF_UNSIGNED: byte_n <= `HALF;
84             `FUNC_WORD: byte_n <= `WORD;
85             default: byte_n <= `WORD;
86         endcase
87         jump_choose <= `JUMP_PC_IMM;
88     end
89     `OPCODE_STORE: begin
90         CPU_MIO <= 1'b1;
91         case (Fun3)
92             `FUNC_BYTE: wea <= `WEA_BYTE;
93             `FUNC_HALF: wea <= `WEA_HALF;
94             `FUNC_WORD: wea <= `WEA_WORD;
95             default: wea <= `WEA_READ;
96         endcase
97         ImmSel1 <= `IMM_SEL_S;
98         MemtoReg <= `MEM2REG_MEM;
99         ALU_Control <= 4'b0;
100        ALUSrc_B <= 1'b1;
101        Jump <= 1'b0;
102        Branch <= 1'b0;
103        Regwrite <= 1'b0;
104        sign <= 1'b1;
105        byte_n <= `WORD;
106        jump_choose <= `JUMP_PC_IMM;
107    end
108    `OPCODE_BRANCH: begin
109        CPU_MIO <= 1'b0;
110        wea <= `WEA_READ;
111        ImmSel1 <= `IMM_SEL_B;
112        MemtoReg <= `MEM2REG_ALU;
113        case (Fun3)
114            `FUNC_EQ: ALU_Control <= `ALU_OP_SUB;
115            `FUNC_NE: ALU_Control <= `ALU_OP_EQ;
116            `FUNC_LT: ALU_Control <= `ALU_OP_SGE;
117            `FUNC_GE: ALU_Control <= `ALU_OP_SLT;
118            `FUNC_LTU: ALU_Control <= `ALU_OP_SGEU;
119            `FUNC_GEU: ALU_Control <= `ALU_OP_SLTU;
120            default: ALU_Control <= 4'b0;
121        endcase
122        ALUSrc_B <= 1'b0;
123        Jump <= 1'b0;
124        Branch <= 1'b1;

```

```

125         RegWrite <= 1'b0;
126         sign <= 1'b1;
127         byte_n <= `WORD;
128     end
129     `OPCODE_JAL: begin
130         CPU_MIO <= 1'b0;
131         wea <= `WEA_READ;
132         ImmSel1 <= `IMM_SEL_J;
133         jump_choose <= `JUMP_PC_IMM;
134         MemtoReg <= `MEM2REG_PC_PLUS;
135         ALU_Control <= `ALU_OP_ADD;
136         ALUSrc_B <= 1'b1;
137         Jump <= 1'b1;
138         Branch <= 1'b0;
139         RegWrite <= 1'b1;
140         sign <= 1'b1;
141         byte_n <= `WORD;
142     end
143     `OPCODE_JALR: begin
144         CPU_MIO <= 1'b0;
145         wea <= `WEA_READ;
146         ImmSel1 <= `IMM_SEL_I;
147         MemtoReg <= `MEM2REG_PC_PLUS;
148         ALU_Control <= `ALU_OP_ADD;
149         ALUSrc_B <= 1'b1;
150         Jump <= 1'b1;
151         Branch <= 1'b0;
152         RegWrite <= 1'b1;
153         sign <= 1'b1;
154         byte_n <= `WORD;
155         jump_choose <= `JUMP_ALU;
156     end
157     `OPCODE_LUI: begin
158         CPU_MIO <= 1'b0;
159         wea <= `WEA_READ;
160         ImmSel1 <= `IMM_SEL_U;
161         MemtoReg <= `MEM2REG_ALU;
162         ALU_Control <= `ALU_OP_R2;
163         ALUSrc_B <= 1'b1;
164         Jump <= 1'b0;
165         Branch <= 1'b0;
166         RegWrite <= 1'b1;
167         sign <= 1'b1;
168         byte_n <= `WORD;
169         jump_choose <= `JUMP_PC_IMM;
170     end
171     `OPCODE_AUIPC: begin
172         CPU_MIO <= 1'b0;
173         wea <= `WEA_READ;
174         ImmSel1 <= `IMM_SEL_U;
175         MemtoReg <= `MEM2REG_IMM_PC;
176         ALU_Control <= `ALU_OP_ADD;
177         ALUSrc_B <= 1'b1;
178         Jump <= 1'b0;
179         Branch <= 1'b0;
180         RegWrite <= 1'b1;

```

```

181         sign <= 1'b1;
182         byte_n <= `WORD;
183         jump_choose <= `JUMP_PC_IMM;
184     end
185     `OPCODE_PASS: begin
186         CPU_MIO <= 1'b0;
187         wea <= `WEA_READ;
188         ImmSel1 <= `IMM_SEL_I;
189         MemtoReg <= `MEM2REG_ALU;
190         ALU_Control <= 4'b0;
191         ALUSrc_B <= 1'b0;
192         Jump <= 1'b0;
193         Branch <= 1'b0;
194         Regwrite <= 1'b0;
195         sign <= 1'b1;
196         byte_n <= `WORD;
197         jump_choose <= `JUMP_PC_IMM;
198     end
199 endcase
200 end
201 endmodule

```

extend 模块

extend :根据是否有符号 **sign** 和读取字节数 **byte_n** 来生成读取数据的32位扩展形式

```

1  `include "Lab4.vh"
2  module extend (
3      input wire [1:0] byte_n,
4      input wire [31:0] in,
5      input wire sign,
6      output [31:0] mem_data
7  );
8      assign mem_data=byte_n==`WORD?in:
9          byte_n==`HALF?(sign?{{16{in[15]}},in[15:0]}:
10         {16'b0,in[15:0]}):
11         byte_n==`BYTE?(sign?{{24{in[7]}},in[7:0]}:{24'b0,in[7:0]}):
12         :32'b0;
13 endmodule

```

1. **ImmGEN**:根据指令类型 **ImmSel1** 进行立即数扩展.

```

1  `include "Lab4.vh"
2  module ImmGen (
3      input wire [`IMM_SEL_WIDTH-1:0] ImmSel1,
4      input wire [31:0] inst_field,
5      input wire sign,
6      output wire [31:0] Imm
7  );
8      wire [31:0] I_Imm, S_Imm, B_Imm, J_Imm, U_Imm;
9      assign I_Imm = sign ? {{20{inst_field[31]}}, inst_field[31:20]} : {20'b0,
10     inst_field[31:20]};
11     assign S_Imm = sign?{{20{inst_field[31]}}, inst_field[31:25],
12     inst_field[11:7]}:{20'b0, inst_field[31:25], inst_field[11:7]};

```

```

11     assign B_Imm = sign?{{19{inst_field[31]}}}, inst_field[31], inst_field[7],
inst_field[30:25], inst_field[11:8], 1'b0}:{19'b0, inst_field[31],
inst_field[7], inst_field[30:25], inst_field[11:8], 1'b0};

12
13     assign J_Imm = sign?{{11{inst_field[31]}}}, inst_field[19:12],
inst_field[20], inst_field[30:21], 1'b0}:
{10'b0,inst_field[31],inst_field[19:12], inst_field[20], inst_field[30:21],
1'b0};

14     assign U_Imm = {inst_field[31:12], 12'b0};
15     assign Imm = (ImmSel1 == `IMM_SEL_I) ? I_Imm :
16                 (ImmSel1 == `IMM_SEL_S) ? S_Imm :
17                 (ImmSel1 == `IMM_SEL_B) ? B_Imm :
18                 (ImmSel1 == `IMM_SEL_J) ? J_Imm :
19                 (ImmSel1 == `IMM_SEL_U) ? U_Imm : 32'b0;
20 endmodule

```

ALU 模块

ALU:修改原 ALU 模块增添了若干运算.

```

1  `timescale 1ns / 1ps
2  module ALU (
3      input  [31:0] A,
4      input  [31:0] B,
5      input  [ 3:0] ALU_operation,
6      output [31:0] res,
7      output          zero
8  );
9      wire signed [31:0] A_s = $signed(A);
10     wire signed [31:0] B_s = $signed(B);
11     wire [31:0] A_u = $unsigned(A);
12     wire [31:0] B_u = $unsigned(B);
13     wire [31:0] result0 = A_s + B_s;
14     wire [31:0] result1 = A_s - B_s;
15     wire [31:0] result2 = A << B[4:0];
16     wire [31:0] result3 = (A_s < B_s) ? 32'b1 : 32'b0;
17     wire [31:0] result4 = (A_u < B_u) ? 32'b1 : 32'b0;
18     wire [31:0] result5 = A ^ B;
19     wire [31:0] result6 = A >> B[4:0];
20     wire [31:0] result7 = A_s >>> B_s[4:0];
21     wire [31:0] result8 = A | B;
22     wire [31:0] result9 = A & B;
23     wire [31:0] result10 = ~|result1;
24     wire [31:0] result11 = ~|result3;
25     wire [31:0] result12 = ~|result4;
26     wire [31:0] result13 = B;
27     assign res = (ALU_operation==4'b0000)?result0:
28                 (ALU_operation==4'b0001)?result1:
29                 (ALU_operation==4'b0010)?result2:
30                 (ALU_operation==4'b0011)?result3:
31                 (ALU_operation==4'b0100)?result4:
32                 (ALU_operation==4'b0101)?result5:
33                 (ALU_operation==4'b0110)?result6:
34                 (ALU_operation==4'b0111)?result7:
35                 (ALU_operation==4'b1000)?result8:

```



```

36         (ALU_operation==4'b1001)?result9:
37         (ALU_operation==4'b1010)?result10:
38         (ALU_operation==4'b1011)?result11:
39         (ALU_operation==4'b1100)?result12:
40         (ALU_operation==4'b1101)?result13:
41         32'b0;
42     assign zero = ~(|res) ? 1'b1 : 1'b0;
43 endmodule

```

Datapath 模块

Datapath:将各个部件连接起来,实际上CPU的核心部件,决定了CPU运作的逻辑,代码 1 到 68 行主要是部件之间的简单连接,下面分析一下复杂逻辑部分(其中要注意 extend 模块的 in 即内存读入需要进行便宜操作以便和地址对齐,原因同上).

- 通过 ALUSrc_B 选择 ALU 的第二个操作数

```

1 | assign adder_2=ALUSrc_B?imm:Rs2_data;

```

- 进行下一条指令地址 PC_out 的选择
 - PC_temp:用于存储临时的程序计数器值.
 - PC_add_4:等于 PC_temp 加 4,即不进行跳转,正常下一条指令
 - PC_imm:通过 PC 相对偏移量进行跳转
 - Branch_final:等于 Branch 信号和 ALU 模块计算为零 zero 信号的逻辑与,这用于判断是否需要进行条件分支跳转.
 - PC_branch:根据 Branch_final 的值选择 PC_imm 或 PC_add_4
 - PC_jump:根据 Jump 信号的值选择 PC_branch 或者根据 jump_choose 的值选择 ALU_out 或 PC_imm.这用于处理无条件跳转的情况.

```

1 | wire[31:0] PC_temp;
2 | wire[31:0] PC_add_4=PC_temp+4;
3 | wire[31:0] PC_imm=imm+PC_temp;
4 | wire Branch_final=Branch&zero;
5 | wire[31:0] PC_branch=Branch_final?PC_imm:PC_add_4;
6 | wire[31:0] PC_jump=Jump?((jump_choose==`JUMP_ALU)?
    ALU_out:PC_imm):PC_branch;

```

- 根据 MemtoReg 选择写入寄存器的源数据

```

1 | assign w_data=MemtoReg==`MEM2REG_MEM?mem_out:
2 |         MemtoReg==`MEM2REG_ALU?ALU_out:
3 |         MemtoReg==`MEM2REG_PC_PLUS?PC_add_4:
4 |         MemtoReg==`MEM2REG_IMM_PC?PC_imm:32'b0;

```

下面是完整代码

```

1 | `include "Lab4.vh"
2 | module DataPath (
3 |     `RegFile_Regs_Outputs
4 |     input wire clk,

```

```

5     input wire rst,
6     input wire [31:0] inst_field,
7     input wire [31:0] Data_in,
8     input wire [`IMM_SEL_WIDTH-1:0] ImmSel1,
9     input wire ALUSrc_B,
10    input wire [`MEM2REG_WIDTH-1:0] MemtoReg,
11    input wire Jump,
12    input wire Branch,
13    input wire RegWrite,
14    input wire [`ALU_OP_WIDTH-1:0] ALU_Control,
15    input wire sign,
16    input wire[1:0] byte_n,
17    input wire jump_choose,
18    output wire [31:0] PC_out,
19    output wire [31:0] Data_out,
20    output wire [31:0] Addr_out
21 );
22 reg[31:0] PC=32'h4;
23 wire[31:0] imm;
24 ImmGen U1 (
25     .ImmSel1(ImmSel1),
26     .inst_field(inst_field),
27     .sign(1'b1),
28     .Imm(imm)
29 );
30 wire [31:0] Rs1_data, Rs2_data;
31 wire [4:0] Rs1_addr, Rs2_addr, W_addr;
32 wire[31:0]W_data;
33 assign Rs1_addr = inst_field[19:15];
34 assign Rs2_addr = inst_field[24:20];
35 assign W_addr = inst_field[11:7];
36 Regs U2 (
37     `RegFile_Regs_Arguments
38     .clk(clk),
39     .rst(rst),
40     .Rs1_addr(Rs1_addr),
41     .Rs2_addr(Rs2_addr),
42     .Wt_addr(W_addr),
43     .Wt_data(W_data),
44     .RegWrite(RegWrite),
45     .Rs1_data(Rs1_data),
46     .Rs2_data(Rs2_data)
47 );
48 wire[31:0] adder_2;
49 wire [31:0] ALU_out;
50 wire zero;
51 ALU U3 (
52     .A(Rs1_data),
53     .B(adder_2),
54     .ALU_operation(ALU_Control),
55     .res(ALU_out),
56     .zero(zero)
57 );
58 assign Addr_out=ALU_out;
59 assign Data_out=Rs2_data;
60 wire [31:0] mem_out;

```

```

61
62 extend U4(
63     .byte_n(byte_n),
64     .in(Data_in>>((Addr_out%4)<<3)),
65     .sign(sign),
66     .mem_data(mem_out)
67 );
68
69 assign adder_2=ALUSrc_B?imm:Rs2_data;
70
71 wire[31:0] PC_temp;
72 wire[31:0] PC_add_4=PC_temp+4;
73 wire[31:0] PC_imm=imm+PC_temp;
74 wire Branch_final=Branch&zero;
75 wire[31:0] PC_branch=Branch_final?PC_imm:PC_add_4;
76 wire[31:0] PC_jump=Jump?((jump_choose==`JUMP_ALU)?ALU_out:PC_imm):PC_branch;
77 always@(posedge clk or posedge rst)
78 begin
79     if(rst)
80     begin
81         PC<=32'hFFFFFFC;
82     end
83     else
84     begin
85         PC<=PC_jump;
86     end
87 end
88 assign PC_temp=PC;
89 assign PC_out=PC_jump;
90 assign w_data=MementoReg==`MEM2REG_MEM?mem_out:
91     MementoReg==`MEM2REG_ALU?ALU_out:
92     MementoReg==`MEM2REG_PC_PLUS?PC_add_4:
93     MementoReg==`MEM2REG_IMM_PC?PC_imm:32'b0;
94
95 endmodule

```

头文件宏定义

下面是一些我修改过的头文件宏定义

```

1  /* WHAT'S THIS HEADERFILE FOR? */
2  /*
3      * Reffered to code written by PanZiyue, TA of 2020_CO
4      * Macro for opcode/func3 for RV32I
5      * declaration, inputs/outputs, assignment for debug signals(RegFile)
6  */
7  /* wea */
8  `define WEA_READ 4'b0000
9  `define WEA_BYTE 4'b0001
10 `define WEA_HALF 4'b0011
11 `define WEA_WORD 4'b1111
12 /*JUMP CHOOSE*/
13 `define JUMP_PC_IMM 1'b0
14 `define JUMP_ALU 1'b1
15 /* Byte/Half/word */

```

```

16 `define BYTE 2'b00
17 `define HALF 2'b01
18 `define WORD 2'b10
19 /* ALU Operation(Using in Lab1) */
20 `define ALU_OP_WIDTH 4
21
22 `define ALU_OP_ADD      `ALU_OP_WIDTH'd0
23 `define ALU_OP_SUB      `ALU_OP_WIDTH'd1
24 `define ALU_OP_SLL      `ALU_OP_WIDTH'd2
25 `define ALU_OP_SLT      `ALU_OP_WIDTH'd3
26 `define ALU_OP_SLTU     `ALU_OP_WIDTH'd4
27 `define ALU_OP_XOR      `ALU_OP_WIDTH'd5
28 `define ALU_OP_SRL      `ALU_OP_WIDTH'd6
29 `define ALU_OP_SRA      `ALU_OP_WIDTH'd7
30 `define ALU_OP_OR        `ALU_OP_WIDTH'd8
31 `define ALU_OP_AND        `ALU_OP_WIDTH'd9
32 `define ALU_OP_EQ        `ALU_OP_WIDTH'd10
33 `define ALU_OP_SGE        `ALU_OP_WIDTH'd11
34 `define ALU_OP_SGEU       `ALU_OP_WIDTH'd12
35 `define ALU_OP_R2         `ALU_OP_WIDTH'd13
36
37 /*-----*/
38
39 /* Inst decoding(Using in Lab4/5) */
40 /* Opcode(5-bits) */
41 // R-Type
42 `define OP_CODE_ALU      5'b01100
43 // I-Type
44 `define OP_CODE_ALU_IMM  5'b00100
45 `define OP_CODE_LOAD     5'b00000
46 `define OP_CODE_JALR     5'b11001
47 `define OP_CODE_ENV      5'b11100
48 `define OP_CODE_JALR     5'b11001
49 // S-Type
50 `define OP_CODE_STORE    5'b01000
51 // B-Type
52 `define OP_CODE_BRANCH   5'b11000
53 // J-Type
54 `define OP_CODE_JAL      5'b11011
55 // U-Type
56 `define OP_CODE_LUI      5'b01101
57 `define OP_CODE_AUIPC    5'b00101
58
59 // not use
60 `define OP_CODE_PASS     5'b00000
61
62 /* Func3(3-bits) */
63 // R-Type & I-Type(ALU)
64 // For R-Type, SUB if inst[30] else ADD
65 `define FUNC_ADD         3'd0
66 // Shift Left (Logical)
67 `define FUNC_SL          3'd1
68 `define FUNC_SLT         3'd2
69 `define FUNC_SLTU        3'd3
70 `define FUNC_XOR         3'd4
71 // Shift Right Arith if inst[30] else Logical

```

```

72 `define FUNC_SR          3'd5
73 `define FUNC_OR          3'd6
74 `define FUNC_AND         3'd7
75
76 // I-Type(Load) & S-Type
77 `define FUNC_BYTE        3'd0
78 `define FUNC_HALF        3'd1
79 `define FUNC_WORD        3'd2
80 `define FUNC_BYTE_UNSIGNED 3'd4
81 `define FUNC_HALF_UNSIGNED 3'd5
82
83 // B-Type
84 `define FUNC_EQ          3'd0
85 `define FUNC_NE          3'd1
86 `define FUNC_LT          3'd4
87 `define FUNC_GE          3'd5
88 `define FUNC_LTU         3'd6
89 `define FUNC_GEU         3'd7
90 /*-----*/
91
92 // JAR
93 `define FUNC_JALR        3'd0
94 /* ImmSel signals */
95 // NOTE: You may add terms in Lab4-3 to implement more inst.
96 `define IMM_SEL_WIDTH 3
97
98 `define IMM_SEL_I `IMM_SEL_WIDTH'd0
99 `define IMM_SEL_S `IMM_SEL_WIDTH'd1
100 `define IMM_SEL_B `IMM_SEL_WIDTH'd2
101 `define IMM_SEL_J `IMM_SEL_WIDTH'd3
102 `define IMM_SEL_U `IMM_SEL_WIDTH'd4
103 /*-----*/
104
105 /* Mem2Reg signals */
106 // NOTE: You may add terms in Lab4-3 to implement more inst.
107 `define MEM2REG_WIDTH 2
108
109 `define MEM2REG_ALU `MEM2REG_WIDTH'd0
110 `define MEM2REG_MEM `MEM2REG_WIDTH'd1
111 `define MEM2REG_PC_PLUS `MEM2REG_WIDTH'd2
112 `define MEM2REG_IMM_PC `MEM2REG_WIDTH'd3
113 // `define MEM2REG_IMM `MEM2REG_WIDTH'd4
114 `define YOUR_REGS MY_REGS

```

仿真

SCPU 模块仿真

仿真激励代码 `SCPU_tb` 如下:

```

1 module SCPU_tb ();
2     reg clk;
3     reg rst;
4     reg [31:0] inst_in;
5     reg [31:0] Data_in;

```

```

6  wire MemRW;
7  wire [31:0] PC_out; // Next PC
8  wire [31:0] Data_out; //Rs2 or Imm
9  wire [31:0] Addr_out; //ALU_out
10 wire [3:0] wea;
11 SCPU SCPU (
12     .clk(clk),
13     .rst(rst),
14     .inst_in(inst_in),
15     .Data_in(Data_in),
16     .MemRW(MemRW),
17     .PC_out(PC_out),
18     .Data_out(Data_out),
19     .Addr_out(Addr_out),
20     .wea(wea)
21 );
22 always begin
23     #5 clk = ~clk;
24 end
25 //      # U型指令
26 //      lui x1, 0x625          # 加载上位立即数到0x00625000
27 //      auipc x2, 0xA38        # 加载到程序计数器上位立即数到0x00A38000
28
29 //      # 基础算术逻辑指令
30 //      add x3, x1, x2          # x3 = x1 + x2
31 //      sub x4, x3, x1          # x4 = x3 - x1
32 //      xor x5, x4, x3          # x5 = x4 ^ x3
33 //      or  x6, x5, x4          # x6 = x5 | x4
34 //      and x7, x6, x5          # x7 = x6 & x5
35 //      sll x8, x7, x1          # x8 = x7 << x1
36 //      srl x9, x8, x2          # x9 = x8 >> x2
37 //      sra x10, x9, x1         # x10 = x9 >> x1 (arithmetic)
38 //      slt x11,x10,x9
39 //      sltu x12,x11,x10
40
41 //      # 立即操作指令
42 //      addi x11, x10, 0x10 # x11 = x10 + 0x10
43 //      xori x12, x11, 0xFF # x12 = x11 ^ 0xFF
44 //      ori  x13, x12, 0x1F # x13 = x12 | 0x1F
45 //      andi x14, x13, 0x3F # x14 = x13 & 0x3F
46 //      slli x15, x14, 0x2  # x15 = x14 << 0x2
47 //      srli x16, x15, 0x2  # x16 = x15 >> 0x2
48 //      srai x17, x16, 0x2  # x17 = x16 >> 0x2 (arithmetic)
49 //      slti x18,x17,-1
50 //      sltiu x18,x17,-1
51
52 //      # 分支跳转与链接指令
53 //      jal x18, 76          # 跳转到标签end, 并将返回地址保存到x18
54 //      jalr x19, x18, 2     # 通过x18跳到返回地址, 保存下一条指令地址到x19
55
56 //      # 内存加载与存储指令
57 //      lb  x20,5(x1)        # 从x1 + arr加载一个字节到x20
58 //      lh  x21, 5(x1)        # 加载半字
59 //      lw  x22, 4(x1)        # 加载字
60 //      lbu x23, 5(x1)        # 无符号加载字节
61 //      lhu x24, 5(x1)        # 无符号加载半字

```

```

62 //      sb x20, 5(x1)          # 将x20的最低字节存回内存
63 //      sh x21, 5(x1)          # 存储半字
64 //      sw x22, 4(x1)          # 存储字
65
66 //      # 分支指令
67 //      beq x21, x22, 28
68 //      bne x21, x22, 28
69 //      blt x21, x22, 24
70 //      bge x21, x22, 16
71 //      bltu x21, x22, 16
72 //      bgeu x21, x22, 8
73 // 006250B7
74 // 00A38117
75 // 002081B3
76 // 40118233
77 // 003242B3
78 // 0042E333
79 // 005373B3
80 // 00139433
81 // 002454B3
82 // 4014D533
83 // 009525B3
84 // 00A5B633
85 // 01050593
86 // 0FF5C613
87 // 01F66693
88 // 03F6F713
89 // 00271793
90 // 0027D813
91 // 40285893
92 // FFF8A913
93 // FFF8B913
94 // 04C0096F
95 // 002909E7
96 // 00508A03
97 // 00509A83
98 // 0040AB03
99 // 0050CB83
100 // 0050DC03
101 // 014082A3
102 // 015092A3
103 // 0160A223
104 // 016A8E63
105 // 016A9E63
106 // 016ACC63
107 // 016AD863
108 // 016AE863
109 // 016AF463
110 initial begin
111     clk = 0;
112     rst = 0;
113     inst_in = 0;
114     Data_in = 0;
115     #10 rst = 1;
116     #10 rst = 0;
117     // # U型指令

```

```

118 // lui x1, 1573          # 加载上位立即数到0x00625000
119 inst_in = 32'h006250B7;
120 // MemRW = 0;
121 // PC_out = h00000004;
122 // Data_out = h00000000;
123 // Addr_out = h00625000;
124 // wea = 4'b0000;
125
126 // auipc x2, 0xA38        # 加载到程序计数器上位立即数到0x00A38000
127 #10 inst_in = 32'h00A38117;
128 // MemRW = 0;
129 // PC_out = h00000008;
130 // Data_out = h00000000;
131 // Addr_out = h00A38000;
132 // wea = 4'b0000;
133
134 // # 基础算术逻辑指令
135 // add x3, x1, x2          # x3 = x1 + x2
136 #10 inst_in = 32'h002081B3;
137 // MemRW = 0;
138 // PC_out = h0000000C;
139 // Data_out = h00a38004;
140 // Addr_out = h0105d004;
141 // wea = 4'b0000;
142
143 // sub x4, x3, x1          # x4 = x3 - x1
144 #10 inst_in = 32'h40118233;
145 // MemRW = 0;
146 // PC_out = h00000010;
147 // Data_out = h00625000;
148 // Addr_out = h00a38004;
149 // wea = 4'b0000;
150
151 // xor x5, x4, x3          # x5 = x4 ^ x3
152 #10 inst_in = 32'h003242B3;
153 // MemRW = 0;
154 // PC_out = h00000014;
155 // Data_out = h0105d004;
156 // Addr_out = h01a65000;
157 // wea = 4'b0000;
158
159 // or x6, x5, x4           # x6 = x5 | x4
160 #10 inst_in = 32'h0042E333;
161 // MemRW = 0;
162 // PC_out = h00000018;
163 // Data_out = h00a38004;
164 // Addr_out = h01a7d004;
165 // wea = 4'b0000;
166
167 // and x7, x6, x5          # x7 = x6 & x5
168 #10 inst_in = 32'h005373B3;
169 // MemRW = 0;
170 // PC_out = h0000001C;
171 // Data_out = h01a65000;
172 // Addr_out = h01a65000;
173 // wea = 4'b0000;

```



```

174
175 // sll x8, x7, x1      # x8 = x7 << x1
176 #10 inst_in = 32'h00139433;
177 // MemRW = 0;
178 // PC_out = h00000020;
179 // Data_out = h00625000;
180 // Addr_out = h01a65000;
181 // wea = 4'b0000;
182
183 // srl x9, x8, x2      # x9 = x8 >> x2
184 #10 inst_in = 32'h002454B3;
185 // MemRW = 0;
186 // PC_out = h00000024;
187 // Data_out = h00a38004;
188 // Addr_out = h001a6500;
189 // wea = 4'b0000;
190
191 // sra x10, x9, x1     # x10 = x9 >> x1 (arithmetic)
192 #10 inst_in = 32'h4014D533;
193 // MemRW = 0;
194 // PC_out = h00000028;
195 // Data_out = h00625000;
196 // Addr_out = h001a6500;
197 // wea = 4'b0000;
198
199 // slt x11,x10,x9      # x11 = x10 < x9
200 #10 inst_in = 32'h009525B3;
201 // MemRW = 0;
202 // PC_out = h0000002C;
203 // Data_out = h001a6500;
204 // Addr_out = h00000000;
205 // wea = 4'b0000;
206
207 // sltu x12,x11,x10    # x12 = x11 < x10
208 #10 inst_in = 32'h00A5B633;
209 // MemRW = 0;
210 // PC_out = h00000030;
211 // Data_out = h001a6500;
212 // Addr_out = h00000001;
213 // wea = 4'b0000;
214
215 // # 立即操作指令
216 // addi x11, x10, 0x10 # x11 = x10 + 0x10
217 #10 inst_in = 32'h01050593;
218 // MemRW = 0;
219 // PC_out = h00000034;
220 // Data_out = h00000000;
221 // Addr_out = h001a6510;
222 // wea = 4'b0000;
223
224 // xori x12, x11, 0xFF # x12 = x11 ^ 0xFF
225 #10 inst_in = 32'h0FF5C613;
226 // MemRW = 0;
227 // PC_out = h00000038;
228 // Data_out = h00000000;
229 // Addr_out = h001a65ef;

```

```

230 // wea = 4'b0000;
231
232 // ori x13, x12, 0x1F # x13 = x12 | 0x1F
233 #10 inst_in = 32'h01F66693;
234 // MemRW = 0;
235 // PC_out = h0000003c;
236 // Data_out = h00000000;
237 // Addr_out = h001a65ff;
238 // wea = 4'b0000;
239
240 // andi x14, x13, 0x3F # x14 = x13 & 0x3F
241 #10 inst_in = 32'h03F6F713;
242 // MemRW = 0;
243 // PC_out = h00000040;
244 // Data_out = h00000000;
245 // Addr_out = h0000003f;
246 // wea = 4'b0000;
247
248 // slli x15, x14, 0x2 # x15 = x14 << 0x2
249 #10 inst_in = 32'h00271793;
250 // MemRW = 0;
251 // PC_out = h00000044;
252 // Data_out = h00a38004;
253 // Addr_out = h000000fc;
254 // wea = 4'b0000;
255
256 // srli x16, x15, 0x2 # x16 = x15 >> 0x2
257 #10 inst_in = 32'h0027D813;
258 // MemRW = 0;
259 // PC_out = h00000048;
260 // Data_out = h00a38004;
261 // Addr_out = h0000003f;
262 // wea = 4'b0000;
263
264 // srai x17, x16, 0x2 # x17 = x16 >> 0x2 (arithmetic)
265 #10 inst_in = 32'h40285893;
266 // MemRW = 0;
267 // PC_out = h0000004c;
268 // Data_out = h00a38004;
269 // Addr_out = h0000000f;
270 // wea = 4'b0000;
271
272 // slti x18,x17,-1 # x18 = x17 < -1
273 #10 inst_in = 32'hFFF8A913;
274 // MemRW = 0;
275 // PC_out = h00000050;
276 // Data_out = h00000000;
277 // Addr_out = h00000000;
278 // wea = 4'b0000;
279
280 // sltiu x18,x17,-1 # x18 = x17 < -1
281 #10 inst_in = 32'hFFF8B913;
282 // MemRW = 0;
283 // PC_out = h00000054;
284 // Data_out = h00000000;
285 // Addr_out = h00000001;

```

```

286 // wea = 4'b0000;
287
288 // # 分支跳转与链接指令
289 // jal x18, 76 # 跳转到标签end, 并将返回地址保存到x18
290 #10 inst_in = 32'h04C0096F;
291 // MemRW = 0;
292 // PC_out = h000000a0;
293 // Data_out = h001a6500;
294 // Addr_out = h0000004c;
295 // wea = 4'b0000;
296
297 // jalr x19, x18, 2 # 通过x18跳到返回地址, 保存下一条指令地址到x19
298 #10 inst_in = 32'h002909E7;
299 // MemRW = 0;
300 // PC_out = h0000005a;
301 // Data_out = h00a38004;
302 // Addr_out = h0000005a;
303 // wea = 4'b0000;
304
305 // # 内存加载与存储指令
306 // lb x20, 5(x1) # 从x1 + arr加载一个字节到x20
307 #10 inst_in = 32'h00508A03;
308 Data_in = 32'h0000FF01;
309 // MemRW = 1;
310 // PC_out = h0000005e;
311 // Data_out = h01a65000;
312 // Addr_out = h00625005;
313 // wea = 4'b0000;
314
315 // lh x21, 5(x1) # 加载半字
316 #10 inst_in = 32'h00509A83;
317 // MemRW = 1;
318 // PC_out = h00000062;
319 // Data_out = h01a65000;
320 // Addr_out = h00625005;
321 // wea = 4'b0000;
322
323 // lw x22, 4(x1) # 加载字
324 #10 inst_in = 32'h0040AB03;
325 // MemRW = 1;
326 // PC_out = h00000066;
327 // Data_out = h00a38004;
328 // Addr_out = h00625004;
329 // wea = 4'b0000;
330
331 // lbu x23, 5(x1) # 无符号加载字节
332 #10 inst_in = 32'h0050CB83;
333 // MemRW = 1;
334 // PC_out = h0000006a;
335 // Data_out = h01a65000;
336 // Addr_out = h00625005;
337 // wea = 4'b0000;
338
339 // lhu x24, 5(x1) # 无符号加载半字
340 #10 inst_in = 32'h0050DC03;
341 // MemRW = 1;

```

```

342 // PC_out = h0000006e;
343 // Data_out = h01a65000;
344 // Addr_out = h00625005;
345 // wea = 4'b0000;
346
347 // sb x20, 5(x1)          # 将x20的最低字节存回内存
348 #10 inst_in = 32'h014082A3;
349 // MemRW = 1;
350 // PC_out = h00000072;
351 // Data_out = hfffffff00;
352 // Addr_out = h00625005;
353 // wea = 4'b0010;
354
355 // sh x21, 5(x1)          # 存储半字
356 #10 inst_in = 32'h015092A3;
357 // MemRW = 1;
358 // PC_out = h00000076;
359 // Data_out = 0000ff00;
360 // Addr_out = h00625005;
361 // wea = 4'b0110;
362
363 // sw x22, 4(x1)          # 存储字
364 #10 inst_in = 32'h0160A223;
365 // MemRW = 1;
366 // PC_out = h0000007a;
367 // Data_out = h0000ff01;
368 // Addr_out = h00625004;
369 // wea = 4'b1111;
370
371 // # 分支指令
372 // beq x21, x22, 28
373 #10 inst_in = 32'h016A8E63;
374 // MemRW = 0;
375 // PC_out = h0000007e;
376 // Data_out = h0000ff01;
377 // Addr_out = hffff01fe;
378 // wea = 4'b0000;
379
380 // bne x21, x22, 28
381 #10 inst_in = 32'h016A9E63;
382 // MemRW = 0;
383 // PC_out = h0000009a;
384 // Data_out = h0000ff01;
385 // Addr_out = h00000000;
386 // wea = 4'b0000;
387
388 // blt x21, x22, 24
389 #10 inst_in = 32'h016ACC63;
390 // MemRW = 0;
391 // PC_out = h000000b2;
392 // Data_out = h0000ff01;
393 // Addr_out = h00000000;
394 // wea = 4'b0000;
395
396 // bge x21, x22, 16
397 #10 inst_in = 32'h016AD863;

```

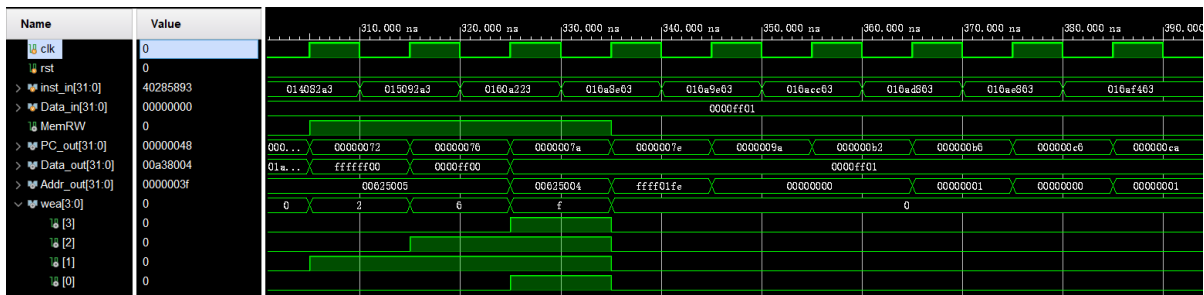
```

398 // MemRW = 0;
399 // PC_out = h000000b6;
400 // Data_out = h0000ff01;
401 // Addr_out = h00000001;
402 // wea = 4'b0000;
403
404 // bltu x21, x22, 16
405 #10 inst_in = 32'h016AE863;
406 // MemRW = 0;
407 // PC_out = h000000c6;
408 // Data_out = h0000ff01;
409 // Addr_out = h00000000;
410 // wea = 4'b0000;
411
412 // bgeu x21, x22, 8
413 #10 inst_in = 32'h016AF463;
414 // MemRW = 0;
415 // PC_out = h000000ca;
416 // Data_out = h0000ff01;
417 // Addr_out = h00000001;
418 #10;
419 $finish();
420
421 end
422 endmodule

```

仿真结果如下





仿真结果符合根据指令集编写的正确输出表

相对时间	指令 (inst_in)	MemRW	PC_out	Data_out	Addr_out	wea
0	32'h006250B7	0	h00000004	h00000000	h00625000	0000
10	32'h00A38117	0	h00000008	h00000000	h00A38000	0000
20	32'h002081B3	0	h0000000C	h00a38004	h0105d004	0000
30	32'h40118233	0	h00000010	h00625000	h00a38004	0000
40	32'h003242B3	0	h00000014	h0105d004	h01a65000	0000
50	32'h0042E333	0	h00000018	h00a38004	h01a7d004	0000
60	32'h005373B3	0	h0000001C	h01a65000	h01a65000	0000
70	32'h00139433	0	h00000020	h00625000	h01a65000	0000
80	32'h002454B3	0	h00000024	h00a38004	h001a6500	0000
90	32'h4014D533	0	h00000028	h00625000	h001a6500	0000
100	32'h009525B3	0	h0000002C	h001a6500	h00000000	0000
110	32'h00A5B633	0	h00000030	h001a6500	h00000001	0000
120	32'h01050593	0	h00000034	h00000000	h001a6510	0000
130	32'h0FF5C613	0	h00000038	h00000000	h001a65ef	0000
140	32'h01F66693	0	h0000003C	h00000000	h001a65ff	0000
150	32'h03F6F713	0	h00000040	h00000000	h0000003f	0000
160	32'h00271793	0	h00000044	h00a38004	h000000fc	0000
170	32'h0027D813	0	h00000048	h00a38004	h0000003f	0000
180	32'h40285893	0	h0000004C	h00a38004	h0000000f	0000
190	32'hFFF8A913	0	h00000050	h00000000	h00000000	0000
200	32'hFFF8B913	0	h00000054	h00000000	h00000001	0000
210	32'h04C0096F	0	h000000a0	h001a6500	h0000004c	0000
220	32'h002909E7	0	h0000005a	h00a38004	h0000005a	0000
230	32'h00508A03	1	h0000005e	h01a65000	h00625005	0000
240	32'h00509A83	1	h00000062	h01a65000	h00625005	0000

相对时间	指令 (inst_in)	MemRW	PC_out	Data_out	Addr_out	wea
250	32'h0040AB03	1	h00000066	h00a38004	h00625004	0000
260	32'h0050CB83	1	h0000006a	h01a65000	h00625005	0000
270	32'h0050DC03	1	h0000006e	h01a65000	h00625005	0000
280	32'h014082A3	1	h00000072	hffffff00	h00625005	0010
290	32'h015092A3	1	h00000076	0000ff00	h00625005	0110
300	32'h0160A223	1	h0000007a	h0000ff01	h00625004	1111
310	32'h016A8E63	0	h0000007e	h0000ff01	hffff01fe	0000
320	32'h016A9E63	0	h0000009a	h0000ff01	h00000000	0000
330	32'h016ACC63	0	h000000b2	h0000ff01	h00000000	0000
340	32'h016AD863	0	h000000b6	h0000ff01	h00000001	0000
350	32'h016AE863	0	h000000c6	h0000ff01	h00000000	0000
360	32'h016AF463	0	h000000ca	h0000ff01	h00000001	0000

datapath 模块仿真

如下是仿真代码

```

1 // Testbench for DataPath module
2 `timescale 1ns / 1ps
3 `include "../sources_1/new/Lab4.vh"
4 module DataPath_tb;
5
6     // Inputs
7     reg clk = 1'b0;
8     reg rst;
9     reg [31:0] inst_field;
10    reg [31:0] Data_in;
11    reg [`IMM_SEL_WIDTH-1:0] ImmSel1;
12    reg ALUSrc_B;
13    reg [`MEM2REG_WIDTH-1:0] MemtoReg;
14    reg Jump;
15    reg Branch;
16    reg RegWrite;
17    reg [`ALU_OP_WIDTH-1:0] ALU_Control;
18    reg sign;
19    reg [1:0] byte_n;
20    reg jump_choose;
21
22    // Outputs
23    wire [31:0] PC_out;
24    wire [31:0] Data_out;
25    wire [31:0] Addr_out;
26
27    // Instantiate the Unit Under Test (UUT)

```

```

28     DataPath uut (
29         .clk(clk),
30         .rst(rst),
31         .inst_field(inst_field),
32         .Data_in(Data_in),
33         .ImmSel1(ImmSel1),
34         .ALUSrc_B(ALUSrc_B),
35         .MemtoReg(MemtoReg),
36         .Jump(Jump),
37         .Branch(Branch),
38         .RegWrite(RegWrite),
39         .ALU_Control(ALU_Control),
40         .sign(sign),
41         .byte_n(byte_n),
42         .jump_choose(jump_choose),
43         .PC_out(PC_out),
44         .Data_out(Data_out),
45         .Addr_out(Addr_out)
46     );
47
48     // Initialize all inputs
49     initial begin
50         rst = 0;
51         #10 rst = 1;
52         #10 rst = 0;
53         // lui x1 1573
54         inst_field <= 32'h006250B7;
55         Data_in <= 32'h00000000;
56         ImmSel1 <= `IMM_SEL_U;
57         MemtoReg <= `MEM2REG_ALU;
58         ALU_Control <= `ALU_OP_R2;
59         ALUSrc_B <= 1'b1;
60         Jump <= 1'b0;
61         Branch <= 1'b0;
62         RegWrite <= 1'b1;
63         sign <= 1'b1;
64         byte_n <= `WORD;
65         jump_choose <= `JUMP_PC_IMM;
66         #15;
67         // auipc x2, 0xA38
68         inst_field <= 32'h00A38117;
69         Data_in <= 32'h00000000;
70         ImmSel1 <= `IMM_SEL_U;
71         MemtoReg <= `MEM2REG_IMM_PC;
72         ALU_Control <= `ALU_OP_ADD;
73         ALUSrc_B <= 1'b1;
74         Jump <= 1'b0;
75         Branch <= 1'b0;
76         RegWrite <= 1'b1;
77         sign <= 1'b1;
78         byte_n <= `WORD;
79         jump_choose <= `JUMP_PC_IMM;
80         #10;
81
82         // add x3, x1, x2
83         inst_field <= 32'h002081B3;

```

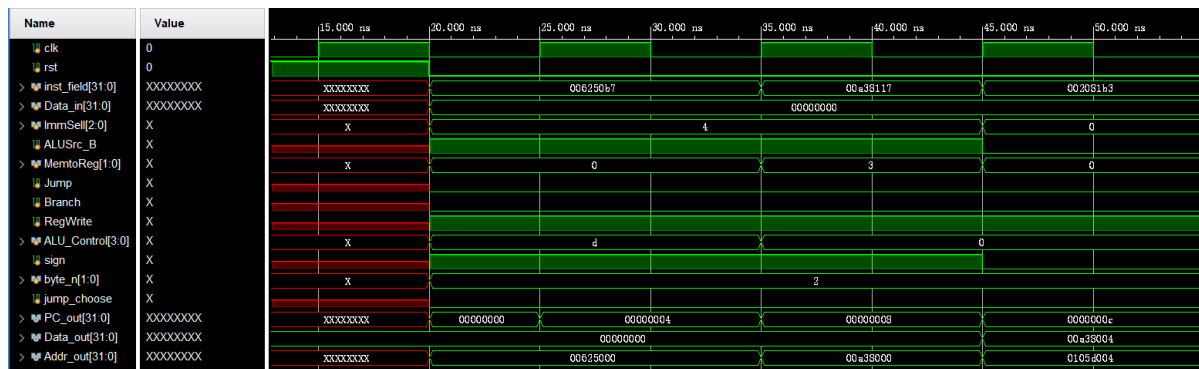


```

84     Data_in <= 32'h00000000;
85     ImmSel1 <= 3'b0;
86     MemtoReg <= `MEM2REG_ALU;
87     ALUSrc_B <= 1'b0;
88     Jump <= 1'b0;
89     Branch <= 1'b0;
90     RegWrite <= 1'b1;
91     sign <= 1'b0;
92     byte_n <= `WORD;
93     jump_choose <= `JUMP_PC_IMM;
94     ALU_Control <= `ALU_OP_ADD;
95     #10;
96
97     $finish;
98
99 end
100 // Clock generation
101 always #5 clk = ~clk; // Generate a clock with a period of 10ns
102
103 endmodule

```

以下为仿真图像:



该输入为 SCPU 仿真的前三条代码,仿真图像符合前表

controller 模块仿真

以下为仿真代码

```

1  `timescale 1ns / 1ps
2  module Controller_tb ();
3      wire [4:0] Opcode;
4      reg MIO_ready;
5      wire [7:0] Fun7;
6      wire [2:0] Fun3;
7      reg [31:0] inst_field;
8      wire CPU_MIO;
9      wire [3:0] wea;
10     wire [2:0] ImmSel1;
11     wire [1:0] MemtoReg;
12     wire [3:0] ALU_Control;
13     wire ALUSrc_B;
14     wire Jump;
15     wire Branch;
16     wire RegWrite;
17     wire sign;

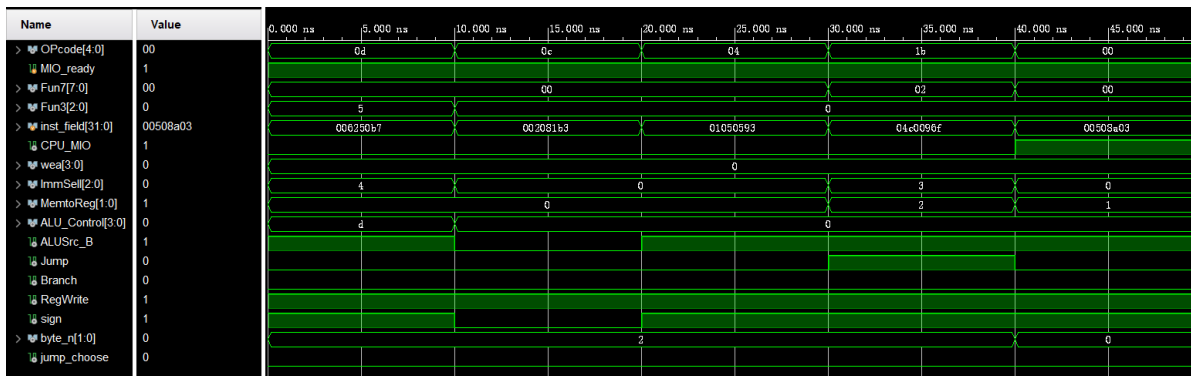
```

```

18  wire [1:0] byte_n;
19  wire jump_choose;
20  Controller uut (
21      .OPcode(OPcode),
22      .MIO_ready(MIO_ready),
23      .Fun7(Fun7),
24      .Fun3(Fun3),
25      .CPU_MIO(CPU_MIO),
26      .wea(wea),
27      .ImmSel1(ImmSel1),
28      .MemtoReg(MemtoReg),
29      .ALU_Control(ALU_Control),
30      .ALUSrc_B(ALUSrc_B),
31      .Jump(Jump),
32      .Branch(Branch),
33      .RegWrite(RegWrite),
34      .sign(sign),
35      .byte_n(byte_n),
36      .jump_choose(jump_choose)
37  );
38  assign OPcode = inst_field[6:2];
39  assign Fun7   = inst_field[31:25];
40  assign Fun3   = inst_field[14:12];
41  initial begin
42      // lui x1 1573
43      MIO_ready  = 1'b1;
44      inst_field = 32'h006250B7;
45      #10;
46      // add x3, x1, x2
47      inst_field = 32'h002081B3;
48      #10;
49      // addi x11, x10, 0x10
50      inst_field = 32'h01050593;
51      #10;
52      // jal x18, 76
53      inst_field = 32'h04C0096F;
54      #10;
55      // lb x20, 5(x1)
56      inst_field = 32'h00508A03;
57      #10;
58      $finish;
59  end
60  endmodule

```

以下为仿真图像:



仿真第一条指令为 lui ,可以看到输出 wea 为0不写入, ImmSel1 为4是U型指令, RegWrite 为1, MemtoReg 为0将 ALU 结果写入寄存器, ALUSrc_B 为1选取立即数作为第二个操作数, ALU_Control 为d表示 ALU 的操作为保留第二操作数输出, Jump 和 Branch 均为0表示不会跳转, 剩余信号不重要; 仿真第二条指令为 add ,可以看到输出 wea 为0不写入, RegWrite 为1, MemtoReg 为0将 ALU 结果写入寄存器, ALUSrc_B 为0选取寄存器作为第二个操作数, ALU_Control 为0表示 ALU 的操作为将第一第二操作数相加输出, Jump 和 Branch 均为0表示不会跳转, 剩余信号不重要; 仿真第三条指令为 add ,可以看到输出 wea 为0不写入, ImmSel1 为1是I型指令, RegWrite 为1, MemtoReg 为0将 ALU 结果写入寄存器, ALUSrc_B 为1选取立即数作为第二个操作数, ALU_Control 为0表示 ALU 的操作为将第一第二操作数相加输出, Jump 和 Branch 均为0表示不会跳转, 剩余信号不重要; 其余指令照前分析均符合预期。

实验结果与分析

可以看到x31为666说明通过验收代码.

浙江大学实验报告 Lab4-4

专业：计算机科学与技术 姓名：仇国智 学号：3220102181 日期：2024/4/10

课程名称: 计算机组成与设计 实验名称: 实现单周期 CPU-异常与中断 指导老师: 刘海风 成绩:

操作方法与实验步骤

编写源文件

在4-3的基础上增加了RV_INT, CSRRegs 模块.

CSRRegs 模块:

```

1  `include "Lab4.vh"
2  module CSRRegs (
3      input                clk,
4      input                rst,
5      input                [11:0] raddr,
6      input                [11:0] waddr,
7      input                [31:0] wdata,
8      input                csr_w,
9      input                [`CSR_CHANGE_MODE_WIDTH-1:0] csr_wsc_mode,
10     input                [31:0] mepc_bypass_in,
11     input                [31:0] mcause_bypass_in,
12     input                [31:0] mtval_bypass_in,
13     input                [31:0] mstatus_bypass_in,
14     output               [31:0] rdata,
15     output reg          [31:0] mepc,
16     output reg          [31:0] mcause,

```

```

17     output reg [          31:0] mtval,
18     output reg [          31:0] mtvec,
19     output reg [          31:0] mstatus
20 );
21 always @(posedge clk or posedge rst) begin
22     if (rst) begin
23         mepc <= 32'h0;
24         mcause <= 32'h0;
25         mtval <= 32'h0;
26         mtvec <= 32'h44;
27         mstatus <= 32'h8;
28     end else if (csr_w) begin
29         case (csr_wsc_mode)
30             `CSR_WRITE:
31                 case (waddr)
32                     `MEPC: mepc <= wdata;
33                     `MCAUSE: mcause <= wdata;
34                     `MTVAL: mtval <= wdata;
35                     `MTVEC: mtvec <= wdata;
36                     `MSTATUS: mstatus <= wdata;
37                 endcase
38             `CSR_OR:
39                 case (waddr)
40                     `MEPC: mepc <= mepc | wdata;
41                     `MCAUSE: mcause <= mcause | wdata;
42                     `MTVAL: mtval <= mtval | wdata;
43                     `MTVEC: mtvec <= mtvec | wdata;
44                     `MSTATUS: mstatus <= mstatus | wdata;
45                 endcase
46             `CSR_CLEAR:
47                 case (waddr)
48                     `MEPC: mepc <= mepc & !wdata;
49                     `MCAUSE: mcause <= mcause & !wdata;
50                     `MTVAL: mtval <= mtval & !wdata;
51                     `MTVEC: mtvec <= mtvec & !wdata;
52                     `MSTATUS: mstatus <= mstatus & !wdata;
53                 endcase
54             `CSR_BYPASS:
55                 begin
56                     mepc <= mepc_bypass_in;
57                     mcause <= mcause_bypass_in;
58                     mtval <= mtval_bypass_in;
59                     mstatus <= mstatus_bypass_in;
60                 end
61             endcase
62         end
63     end
64 assign rdata = (raddr == `MEPC) ? mepc :
65                 (raddr == `MCAUSE) ? mcause :
66                 (raddr == `MTVAL) ? mtval :
67                 (raddr == `MTVEC) ? mtvec :
68                 (raddr == `MSTATUS) ? mstatus :
69                 32'h0;
70 endmodule

```

该模块用于储存CSR寄存器的值,并且可以通过 `csr_wsc_mode` 来选择写入模式. `csr_wsc_mode` 的值有 `CSR_WRITE`, `CSR_OR`, `CSR_CLEAR`, `CSR_BYPASS` 四种,分别对应写入,设定,清除,和批量写入模式.批量写入时,会将 `mepc_bypass_in`, `mcause_bypass_in`, `mtval_bypass_in`, `mstatus_bypass_in` 的值写入到对应的寄存器中.

RV_INT 模块:

```
1  `include "Lab4.vh"
2  module RV_INT (
3      input clk,
4      input rst,
5      input INT, // 外部中断信号
6      input ecall, // ECALL 指令
7      input mret, // MRET 指令
8      input illegal_inst, // 非法指令信号
9      // input l_access_fault, // 数据访存不对齐
10     // input j_access_fault, // 跳转地址不对齐
11     input [31:0] PC, // PC
12     input [11:0] raddr,
13     input [11:0] waddr,
14     input [31:0] wdata,
15     input csr_w,
16     input wire [31:0] inst_field,
17     input [`CSR_CHANGE_MODE_WIDTH-1:0] csr_wsc_mode,
18     output trap_normal_change, // 用于指示trap和正常流程的转换
19     output [31:0] PC_change, // 用于指示PC流程的切换的PC
20     output [31:0] rdata, //CSR的值
21     `CSR_OUTPUTS
22 );
23 wire [31:0] mepc_bypass_in;
24 wire [31:0] mcause_bypass_in;
25 wire [31:0] mtval_bypass_in;
26 wire [31:0] mstatus_bypass_in;
27 wire [11:0] csr_wsc_mode_real;
28 wire csr_w_real;
29 CSRRegs U1 (
30     .clk(clk),
31     .rst(rst),
32     .raddr(raddr),
33     .waddr(waddr),
34     .wdata(wdata),
35     .csr_w(csr_w_real),
36     .csr_wsc_mode(csr_wsc_mode_real),
37     .mepc_bypass_in(mepc_bypass_in),
38     .mcause_bypass_in(mcause_bypass_in),
39     .mtval_bypass_in(mtval_bypass_in),
40     .mstatus_bypass_in(mstatus_bypass_in),
41     .rdata(rdata),
42     .mepc(mepc),
43     .mcause(mcause),
44     .mtval(mtval),
45     .mtvec(mtvec),
46     .mstatus(mstatus)
47 );
48 wire trap_start=mstatus[`MSTATUS_MIE]&&
(INT||ecall||illegal_inst);////l_access_fault||j_access_fault);
```

```

49     wire trap_exit = mret;
50     assign trap_normal_change = trap_exit || trap_start;
51     assign csr_wsc_mode_real = trap_start || trap_exit ? `CSR_BYPASS :
csr_wsc_mode;
52     assign csr_w_real = trap_start || trap_exit || csr_w;
53     assign PC_change = trap_start ? {mtvec[31:2], 2'b00} : trap_exit ? mepc :
PC;
54     assign mepc_bypass_in = trap_start ? PC : mepc;
55     assign mcause_bypass_in = trap_start ? (INT ? `MCAUSE_INT : ecall ?
`MCAUSE_ECALL : illegal_inst ? `MCAUSE_ILL_INST:32'h0) : mcause;
56     assign mtval_bypass_in = trap_start ? (illegal_inst ? inst_field : 32'h0)
: mtval;
57     assign mstatus_bypass_in = trap_start ?
58     {mstatus[31:`MSTATUS_MPIE+1], mstatus[`MSTATUS_MIE],
mstatus[`MSTATUS_MPIE-1:`MSTATUS_MIE+1], 1'b0, mstatus[`MSTATUS_MIE-1:0]}:
59     trap_exit?
60     {mstatus[31:`MSTATUS_MPIE+1], mstatus[`MSTATUS_MPIE],
mstatus[`MSTATUS_MPIE-1:`MSTATUS_MIE+1], 1'b1, mstatus[`MSTATUS_MIE-1:0]}:
61     mstatus;
62
63 endmodule

```

该模块用于处理异常和中断,当 mstatus 的 MIE 位为1时,并且有中断信号,或者是 ecall 指令,或者是非法指令时,会触发异常,并且批量写入CSR寄存器的值(此时写入 mstatus 的 MIE 位为 0,阻止 trap 处理过程中再次触发异常),同时跳转 PC 的值到 mtvec 中的值.当 mret 指令到来时,会从 mepc 中恢复PC的值,同时批量恢复CSR寄存器的值.注意到只有当中断发生和结束时,才会批量写入CSR寄存器的值同时跳转PC的值,而在其他情况下,只会按地址读取修改CSR寄存器的值.故需要 trap_normal_change 来指示非平凡状态,其余 trap 指令的执行与正常流程一致.

编写异常处理软件

```

1 00100093 addi x1,x0,1
2 00100113 addi x2,x0,1
3 001101B3 add x3,x2,x1
4 003100B3 add x1,x2,x3
5 00308133 add x2,x1,x3
6 001101B3 add x3,x2,x1
7 00000073 ecall
8 003100B3 add x1,x2,x3
9 00308133 add x2,x1,x3
10 001101B3 add x3,x2,x1
11 #INT
12 003100B3 add x1,x2,x3
13 00308133 add x2,x1,x3
14 001101B3 add x3,x2,x1
15 FFFFFFFF add x0,x0,x0 #ill inst
16 003100B3 add x1,x2,x3
17 00308133 add x2,x1,x3
18 001101B3 add x3,x2,x1
19 30002FF3 csrrs x31,0x300,x0 # MSTATUS
20 34102F73 csrrs x30,0x341,x0 # MEPC
21 34202EF3 csrrs x29,0x342,x0 # MCAUSE
22 34302E73 csrrs x28,0x343,x0 # MTVAL
23 30502DF3 csrrs x27,0x305,x0 # MTVEC

```

```

24 80000D37 li x26,0x8000000b
25 00BD0D13 li x26,0x8000000b
26 01DD0463 beq x26,x29,else
27 004F0F13 addi x30,x30,4 #MEPC=MEPC+4
28 341F1073 else: csrrw x0,0x341,x30 # MEPC=MEPC
29 30505073 cssrwi 0x,0x305,0
30 305D9073 csrrw x0,0x305,x27
31 342FD073 csrrsi x0,0x342,-1
32 342FF073 csrrci x0,0x342,-1
33 343F3D73 csrrc x26,0x343,x30
34 30200073 mret

```

这里要注意通过读出的 `mcause` 的值来判断异常的类型,以确定 `mepc` 是否需要加4.

编写仿真激励文件

此时已经将软件代码写入到了 ROM 中,下面为仿真激励模块:

```

1  `include "../sources_1/new/Lab4.vh"
2  module exception_tb ();
3      `CSR_DECLARATION;
4      reg clk = 1'b0;
5      reg rst;
6      wire [31:0] inst_in;
7      reg [31:0] Data_in;
8      reg INT;
9      wire MemRW;
10     wire [31:0] PC_out;
11     wire [31:0] Data_out;
12     wire [31:0] Addr_out;
13     wire [3:0] wea;
14     wire [`CSR_CHANGE_MODE_WIDTH-1:0] csr_wsc_mode;
15     wire CPU_MIO;
16     wire trap_normal_change;
17     wire mret;
18     wire ecall;
19     wire [31:0] `YOUR_REGS[31:0];
20     `RegFile_Regs_Declaration
21     SCPU uut (
22         .clk(clk),
23         .rst(rst),
24         .MIO_ready(MIO_ready),
25         .inst_in(inst_in),
26         .Data_in(Data_in),
27         .INT(INT),
28         .CPU_MIO(CPU_MIO),
29         .MemRW(MemRW),
30         .PC_out(PC_out),
31         .Data_out(Data_out),
32         .Addr_out(Addr_out),
33         .wea(wea),
34         .csr_wsc_mode(csr_wsc_mode),
35         .trap_normal_change(trap_normal_change),
36         .mret(mret),
37         .ecall(ecall),

```

```

38     `RegFile_Regs_Arguments
39     `CSR_ARGUMENTS
40 );
41 `RegFile_Regs_Assignments
42 U2 U2 (
43     .a (PC_out[11:2]),
44     .spo(inst_in)
45 );
46 always begin
47     #5 clk = ~clk;
48 end
49 initial begin
50     rst = 0;
51     INT=0;
52     Data_in=32'h00000000;
53     #10;
54     rst = 1;
55     #10;
56     rst = 0;
57     #270;
58     INT=1;
59     #20;
60     INT=0;
61 end
62 endmodule

```

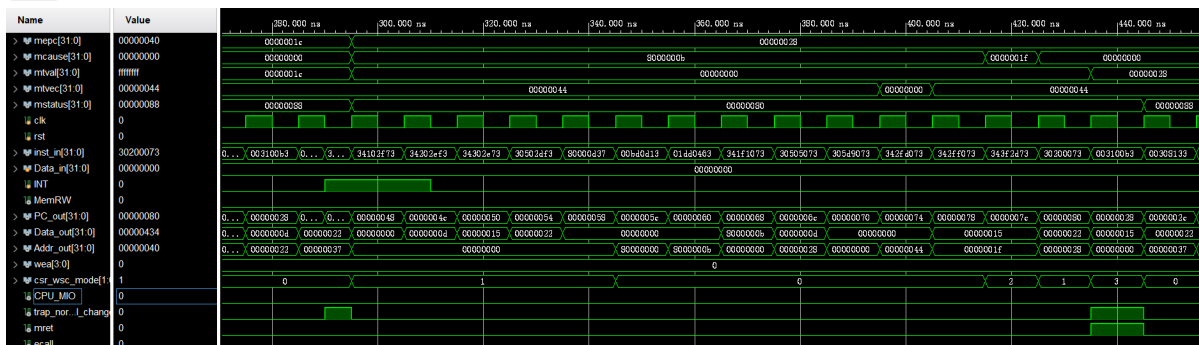
下面为仿真激励图像及其分析:

ecall 指令触发异常:



trap 的进入和退出实现了了 PC +4.在开始时, ecall 信号处于高位,对应的异常代码 8000000b 存入 mcause 中,并且 mstatus 的 MIE 位被清除,阻止再次触发异常.同时 mepc 被写入 PC 的值,跳转到 mtvec 中指向的值.在期间实现了 PC +4.在结束时, mepc 的值被写入 PC 的值,并且 mstatus 的 MIE 位被恢复,允许再次触发异常.

INT 信号触发异常:



分析同上,但是注意此时 PC 未进行递增,同时 INT 亮起超过一个周期,期间未发生二次 trap.

非法指令触发异常:


```

19     .mtvec(mtvec),\
20     .mstatus(mstatus)
21 // CSR DECLARATION
22 `define CSR_DECLARATION \
23     wire [31:0] mepc;\
24     wire [31:0] mcause;\
25     wire [31:0] mtval;\
26     wire [31:0] mtvec;\
27     wire [31:0] mstatus
28 //direct inst
29 `define ECALL 15'b00000000_000_11100
30 `define MRET 15'b0011000_000_11100
31 /*MCAUSE*/
32 `define MCAUSE_ECALL 32'h8000_0003
33 `define MCAUSE_INT 32'h8000_000b
34 `define MCAUSE_ILL_INST 32'h0000_0002
35 /*MASTATUS*/
36 `define MSTATUS_MIE 5'd3
37 `define MSTATUS_MPIE 5'd7
38 /*CSR*/
39 `define MSTATUS 12'h300
40 `define MEPC 12'h341
41 `define MCAUSE 12'h342
42 `define MTVAL 12'h343
43 `define MTVEC 12'h305
44
45 /*CSR change mode*/
46 `define CSR_CHANGE_MODE_WIDTH 2
47 `define CSR_WRITE 2'b00
48 `define CSR_OR 2'b01
49 `define CSR_CLEAR 2'b10
50 `define CSR_BYPASS 2'b11
51 /* wea */
52 `define WEA_READ 4'b0000
53 `define WEA_BYTE 4'b0001
54 `define WEA_HALF 4'b0011
55 `define WEA_WORD 4'b1111
56 /*JUMP CHOOSE*/
57 `define JUMP_PC_IMM 1'b0
58 `define JUMP_ALU 1'b1
59 /* Byte/Half/Word */
60 `define BYTE 2'b00
61 `define HALF 2'b01
62 `define WORD 2'b10
63 /* ALU Operation(Using in Lab1) */
64 `define ALU_OP_WIDTH 4
65
66 `define ALU_OP_ADD `ALU_OP_WIDTH'd0
67 `define ALU_OP_SUB `ALU_OP_WIDTH'd1
68 `define ALU_OP_SLL `ALU_OP_WIDTH'd2
69 `define ALU_OP_SLT `ALU_OP_WIDTH'd3
70 `define ALU_OP_SLTU `ALU_OP_WIDTH'd4
71 `define ALU_OP_XOR `ALU_OP_WIDTH'd5
72 `define ALU_OP_SRL `ALU_OP_WIDTH'd6
73 `define ALU_OP_SRA `ALU_OP_WIDTH'd7
74 `define ALU_OP_OR `ALU_OP_WIDTH'd8

```

```

75 `define ALU_OP_AND `ALU_OP_WIDTH'd9
76 `define ALU_OP_EQ `ALU_OP_WIDTH'd10
77 `define ALU_OP_SGE `ALU_OP_WIDTH'd11
78 `define ALU_OP_SGEU `ALU_OP_WIDTH'd12
79 `define ALU_OP_R2 `ALU_OP_WIDTH'd13
80 `define ALU_OP_R1 `ALU_OP_WIDTH'd14
81
82 /*-----*/
83
84 /* Inst decoding(Using in Lab4/5) */
85 /* Opcode(5-bits) */
86 // R-Type
87 `define OPCODE_ALU 5'b01100
88 // I-Type
89 `define OPCODE_ALU_IMM 5'b00100
90 `define OPCODE_LOAD 5'b00000
91 `define OPCODE_JALR 5'b11001
92 `define OPCODE_ENV 5'b11100
93 `define OPCODE_JALR 5'b11001
94 // S-Type
95 `define OPCODE_STORE 5'b01000
96 // B-Type
97 `define OPCODE_BRANCH 5'b11000
98 // J-Type
99 `define OPCODE_JAL 5'b11011
100 // U-Type
101 `define OPCODE_LUI 5'b01101
102 `define OPCODE_AUIPC 5'b00101
103
104 `define OPCODE_CSR 5'b11100
105 // not use
106 `define OPCODE_PASS 5'b00000
107
108 /* Func3(3-bits) */
109 // R-Type & I-Type(ALU)
110 // For R-Type, SUB if inst[30] else ADD
111 `define FUNC_ADD 3'd0
112 // Shift Left (Logical)
113 `define FUNC_SL 3'd1
114 `define FUNC_SLT 3'd2
115 `define FUNC_SLTU 3'd3
116 `define FUNC_XOR 3'd4
117 // Shift Right Arith if inst[30] else Logical
118 `define FUNC_SR 3'd5
119 `define FUNC_OR 3'd6
120 `define FUNC_AND 3'd7
121
122 // I-Type(Load) & S-Type
123 `define FUNC_BYTE 3'd0
124 `define FUNC_HALF 3'd1
125 `define FUNC_WORD 3'd2
126 `define FUNC_BYTE_UNSIGNED 3'd4
127 `define FUNC_HALF_UNSIGNED 3'd5
128
129 // B-Type
130 `define FUNC_EQ 3'd0

```

```

131 `define FUNC_NE 3'd1
132 `define FUNC_LT 3'd4
133 `define FUNC_GE 3'd5
134 `define FUNC_LTU 3'd6
135 `define FUNC_GEU 3'd7
136 /*-----*/
137
138 // CSR FUNC
139 `define FUNC_CSRRW 3'd1
140 `define FUNC_CSRRS 3'd2
141 `define FUNC_CSRRC 3'd3
142 `define FUNC_CSRRWI 3'd5
143 `define FUNC_CSRRSI 3'd6
144 `define FUNC_CSRRCI 3'd7
145
146 // JAR
147 `define FUNC_JALR 3'd0
148 /* ImmSel signals */
149 // NOTE: You may add terms in Lab4-3 to implement more inst.
150 `define IMM_SEL_WIDTH 3
151
152 `define IMM_SEL_I `IMM_SEL_WIDTH'd0
153 `define IMM_SEL_S `IMM_SEL_WIDTH'd1
154 `define IMM_SEL_B `IMM_SEL_WIDTH'd2
155 `define IMM_SEL_J `IMM_SEL_WIDTH'd3
156 `define IMM_SEL_U `IMM_SEL_WIDTH'd4
157 `define IMM_SEL_CSR `IMM_SEL_WIDTH'd6
158 /*-----*/
159
160 /* Mem2Reg signals */
161 // NOTE: You may add terms in Lab4-3 to implement more inst.
162 `define MEM2REG_WIDTH 3
163
164 `define MEM2REG_ALU `MEM2REG_WIDTH'd0
165 `define MEM2REG_MEM `MEM2REG_WIDTH'd1
166 `define MEM2REG_PC_PLUS `MEM2REG_WIDTH'd2
167 `define MEM2REG_IMM_PC `MEM2REG_WIDTH'd3
168 `define MEM2REG_CSR `MEM2REG_WIDTH'd4
169 // `define MEM2REG_IMM `MEM2REG_WIDTH'd4
170 /*-----*/
171
172 /*-----*/
173 /***** generated code *****/
174 /*-----*/
175
176 /* RegFiles Ports & debug signals */
177 /* NOTE:
178  * AFTER you change "... " in macro YOUR_REGS to the name of your reg-array
179  * in module Regs, such as regs,
180  * you need to *uncomment* the line "`define YOUR_REGS regs" below for
181  * using this set of macros
182  */
183 // `define YOUR_REGS ...
184 `define YOUR_REGS MY_REGS
185 `ifdef YOUR_REGS

```

```
185 `define RegFile_Regs_Outputs \  
186     output [31:0] Reg00, \  
187     output [31:0] Reg01, \  
188     output [31:0] Reg02, \  
189     output [31:0] Reg03, \  
190     output [31:0] Reg04, \  
191     output [31:0] Reg05, \  
192     output [31:0] Reg06, \  
193     output [31:0] Reg07, \  
194     output [31:0] Reg08, \  
195     output [31:0] Reg09, \  
196     output [31:0] Reg10, \  
197     output [31:0] Reg11, \  
198     output [31:0] Reg12, \  
199     output [31:0] Reg13, \  
200     output [31:0] Reg14, \  
201     output [31:0] Reg15, \  
202     output [31:0] Reg16, \  
203     output [31:0] Reg17, \  
204     output [31:0] Reg18, \  
205     output [31:0] Reg19, \  
206     output [31:0] Reg20, \  
207     output [31:0] Reg21, \  
208     output [31:0] Reg22, \  
209     output [31:0] Reg23, \  
210     output [31:0] Reg24, \  
211     output [31:0] Reg25, \  
212     output [31:0] Reg26, \  
213     output [31:0] Reg27, \  
214     output [31:0] Reg28, \  
215     output [31:0] Reg29, \  
216     output [31:0] Reg30, \  
217     output [31:0] Reg31, \  
218 \  
219 `define RegFile_Regs_Assignments \  
220     assign Reg00 = `YOUR_REGS[0]; \  
221     assign Reg01 = `YOUR_REGS[1]; \  
222     assign Reg02 = `YOUR_REGS[2]; \  
223     assign Reg03 = `YOUR_REGS[3]; \  
224     assign Reg04 = `YOUR_REGS[4]; \  
225     assign Reg05 = `YOUR_REGS[5]; \  
226     assign Reg06 = `YOUR_REGS[6]; \  
227     assign Reg07 = `YOUR_REGS[7]; \  
228     assign Reg08 = `YOUR_REGS[8]; \  
229     assign Reg09 = `YOUR_REGS[9]; \  
230     assign Reg10 = `YOUR_REGS[10]; \  
231     assign Reg11 = `YOUR_REGS[11]; \  
232     assign Reg12 = `YOUR_REGS[12]; \  
233     assign Reg13 = `YOUR_REGS[13]; \  
234     assign Reg14 = `YOUR_REGS[14]; \  
235     assign Reg15 = `YOUR_REGS[15]; \  
236     assign Reg16 = `YOUR_REGS[16]; \  
237     assign Reg17 = `YOUR_REGS[17]; \  
238     assign Reg18 = `YOUR_REGS[18]; \  
239     assign Reg19 = `YOUR_REGS[19]; \  
240     assign Reg20 = `YOUR_REGS[20]; \  

```

```

241     assign Reg21 = `YOUR_REGS[21]; \
242     assign Reg22 = `YOUR_REGS[22]; \
243     assign Reg23 = `YOUR_REGS[23]; \
244     assign Reg24 = `YOUR_REGS[24]; \
245     assign Reg25 = `YOUR_REGS[25]; \
246     assign Reg26 = `YOUR_REGS[26]; \
247     assign Reg27 = `YOUR_REGS[27]; \
248     assign Reg28 = `YOUR_REGS[28]; \
249     assign Reg29 = `YOUR_REGS[29]; \
250     assign Reg30 = `YOUR_REGS[30]; \
251     assign Reg31 = `YOUR_REGS[31];
252
253 `define RegFile_Regs_Arguments \
254     .Reg00(Reg00), \
255     .Reg01(Reg01), \
256     .Reg02(Reg02), \
257     .Reg03(Reg03), \
258     .Reg04(Reg04), \
259     .Reg05(Reg05), \
260     .Reg06(Reg06), \
261     .Reg07(Reg07), \
262     .Reg08(Reg08), \
263     .Reg09(Reg09), \
264     .Reg10(Reg10), \
265     .Reg11(Reg11), \
266     .Reg12(Reg12), \
267     .Reg13(Reg13), \
268     .Reg14(Reg14), \
269     .Reg15(Reg15), \
270     .Reg16(Reg16), \
271     .Reg17(Reg17), \
272     .Reg18(Reg18), \
273     .Reg19(Reg19), \
274     .Reg20(Reg20), \
275     .Reg21(Reg21), \
276     .Reg22(Reg22), \
277     .Reg23(Reg23), \
278     .Reg24(Reg24), \
279     .Reg25(Reg25), \
280     .Reg26(Reg26), \
281     .Reg27(Reg27), \
282     .Reg28(Reg28), \
283     .Reg29(Reg29), \
284     .Reg30(Reg30), \
285     .Reg31(Reg31),
286
287 `define RegFile_Regs_Declaration \
288     wire [31:0] Reg00; \
289     wire [31:0] Reg01; \
290     wire [31:0] Reg02; \
291     wire [31:0] Reg03; \
292     wire [31:0] Reg04; \
293     wire [31:0] Reg05; \
294     wire [31:0] Reg06; \
295     wire [31:0] Reg07; \
296     wire [31:0] Reg08; \

```

```
297     wire [31:0] Reg09; \
298     wire [31:0] Reg10; \
299     wire [31:0] Reg11; \
300     wire [31:0] Reg12; \
301     wire [31:0] Reg13; \
302     wire [31:0] Reg14; \
303     wire [31:0] Reg15; \
304     wire [31:0] Reg16; \
305     wire [31:0] Reg17; \
306     wire [31:0] Reg18; \
307     wire [31:0] Reg19; \
308     wire [31:0] Reg20; \
309     wire [31:0] Reg21; \
310     wire [31:0] Reg22; \
311     wire [31:0] Reg23; \
312     wire [31:0] Reg24; \
313     wire [31:0] Reg25; \
314     wire [31:0] Reg26; \
315     wire [31:0] Reg27; \
316     wire [31:0] Reg28; \
317     wire [31:0] Reg29; \
318     wire [31:0] Reg30; \
319     wire [31:0] Reg31;
320
321 `define VGA_RegFile_Inputs \
322     input [31:0] Reg00, \
323     input [31:0] Reg01, \
324     input [31:0] Reg02, \
325     input [31:0] Reg03, \
326     input [31:0] Reg04, \
327     input [31:0] Reg05, \
328     input [31:0] Reg06, \
329     input [31:0] Reg07, \
330     input [31:0] Reg08, \
331     input [31:0] Reg09, \
332     input [31:0] Reg10, \
333     input [31:0] Reg11, \
334     input [31:0] Reg12, \
335     input [31:0] Reg13, \
336     input [31:0] Reg14, \
337     input [31:0] Reg15, \
338     input [31:0] Reg16, \
339     input [31:0] Reg17, \
340     input [31:0] Reg18, \
341     input [31:0] Reg19, \
342     input [31:0] Reg20, \
343     input [31:0] Reg21, \
344     input [31:0] Reg22, \
345     input [31:0] Reg23, \
346     input [31:0] Reg24, \
347     input [31:0] Reg25, \
348     input [31:0] Reg26, \
349     input [31:0] Reg27, \
350     input [31:0] Reg28, \
351     input [31:0] Reg29, \
352     input [31:0] Reg30, \
```

```

353     input [31:0] Reg31,
354
355 `define VGA_RegFile_Arguments \
356     .x0 (Reg00), \
357     .ra (Reg01), \
358     .sp (Reg02), \
359     .gp (Reg03), \
360     .tp (Reg04), \
361     .t0 (Reg05), \
362     .t1 (Reg06), \
363     .t2 (Reg07), \
364     .s0 (Reg08), \
365     .s1 (Reg09), \
366     .a0 (Reg10), \
367     .a1 (Reg11), \
368     .a2 (Reg12), \
369     .a3 (Reg13), \
370     .a4 (Reg14), \
371     .a5 (Reg15), \
372     .a6 (Reg16), \
373     .a7 (Reg17), \
374     .s2 (Reg18), \
375     .s3 (Reg19), \
376     .s4 (Reg20), \
377     .s5 (Reg21), \
378     .s6 (Reg22), \
379     .s7 (Reg23), \
380     .s8 (Reg24), \
381     .s9 (Reg25), \
382     .s10(Reg26), \
383     .s11(Reg27), \
384     .t3 (Reg28), \
385     .t4 (Reg29), \
386     .t5 (Reg30), \
387     .t6 (Reg31),
388
389 `endif // YOUR_REGS
390
391 `define VGA_Debug_Signals_Inputs \
392     input [31:0] pc, \
393     input [31:0] inst, \
394     input [4:0] rs1, \
395     input [31:0] rs1_val, \
396     input [4:0] rs2, \
397     input [31:0] rs2_val, \
398     input [4:0] rd, \
399     input [31:0] reg_i_data, \
400     input reg_wen, \
401     input is_imm, \
402     input is_auipc, \
403     input is_lui, \
404     input [31:0] imm, \
405     input [31:0] a_val, \
406     input [31:0] b_val, \
407     input [3:0] alu_ctrl, \
408     input [2:0] cmp_ctrl, \

```



```

409     input [31:0] alu_res, \
410     input cmp_res, \
411     input is_branch, \
412     input is_jal, \
413     input is_jalr, \
414     input do_branch, \
415     input [31:0] pc_branch, \
416     input mem_wen, \
417     input mem_ren, \
418     input [31:0] dmem_o_data, \
419     input [31:0] dmem_i_data, \
420     input [31:0] dmem_addr,
421
422 `define VGA_Debug_Signals_Arguments \
423     .pc(pc), \
424     .inst(inst), \
425     .rs1(rs1), \
426     .rs1_val(rs1_val), \
427     .rs2(rs2), \
428     .rs2_val(rs2_val), \
429     .rd(rd), \
430     .reg_i_data(reg_i_data), \
431     .reg_wen(reg_wen), \
432     .is_imm(is_imm), \
433     .is_auipc(is_auipc), \
434     .is_lui(is_lui), \
435     .imm(imm), \
436     .a_val(a_val), \
437     .b_val(b_val), \
438     .alu_ctrl(alu_ctrl), \
439     .cmp_ctrl(cmp_ctrl), \
440     .alu_res(alu_res), \
441     .cmp_res(cmp_res), \
442     .is_branch(is_branch), \
443     .is_jal(is_jal), \
444     .is_jalr(is_jalr), \
445     .do_branch(do_branch), \
446     .pc_branch(pc_branch), \
447     .mem_wen(mem_wen), \
448     .mem_ren(mem_ren), \
449     .dmem_o_data(dmem_o_data), \
450     .dmem_i_data(dmem_i_data), \
451     .dmem_addr(dmem_addr),
452
453 `define VGA_Debug_Signals_Outputs \
454     output [31:0] pc, \
455     output [31:0] inst, \
456     output [4:0] rs1, \
457     output [31:0] rs1_val, \
458     output [4:0] rs2, \
459     output [31:0] rs2_val, \
460     output [4:0] rd, \
461     output [31:0] reg_i_data, \
462     output reg_wen, \
463     output is_imm, \
464     output is_auipc, \

```

```

465     output is_lui, \
466     output [31:0] imm, \
467     output [31:0] a_val, \
468     output [31:0] b_val, \
469     output [3:0] alu_ctrl, \
470     output [2:0] cmp_ctrl, \
471     output [31:0] alu_res, \
472     output cmp_res, \
473     output is_branch, \
474     output is_jal, \
475     output is_jalr, \
476     output do_branch, \
477     output [31:0] pc_branch, \
478     output mem_wen, \
479     output mem_ren, \
480     output [31:0] dmem_o_data, \
481     output [31:0] dmem_i_data, \
482     output [31:0] dmem_addr,

```

ALU 模块:

```

1  `timescale 1ns / 1ps
2  module ALU (
3      input  [31:0] A,
4      input  [31:0] B,
5      input  [ 3:0] ALU_operation,
6      output [31:0] res,
7      output          zero
8  );
9      wire signed [31:0] A_s = $signed(A);
10     wire signed [31:0] B_s = $signed(B);
11     wire [31:0] A_u = $unsigned(A);
12     wire [31:0] B_u = $unsigned(B);
13     wire [31:0] result0 = A_s + B_s;
14     wire [31:0] result1 = A_s - B_s;
15     wire [31:0] result2 = A << B[4:0];
16     wire [31:0] result3 = (A_s < B_s) ? 32'b1 : 32'b0;
17     wire [31:0] result4 = (A_u < B_u) ? 32'b1 : 32'b0;
18     wire [31:0] result5 = A ^ B;
19     wire [31:0] result6 = A >> B[4:0];
20     wire [31:0] result7 = A_s >>> B_s[4:0];
21     wire [31:0] result8 = A | B;
22     wire [31:0] result9 = A & B;
23     wire [31:0] result10 = ~|result1;
24     wire [31:0] result11 = ~|result3;
25     wire [31:0] result12 = ~|result4;
26     wire [31:0] result13 = B;
27     wire [31:0] result14 = A;
28     assign res = (ALU_operation==4'b0000)?result0:
29                 (ALU_operation==4'b0001)?result1:
30                 (ALU_operation==4'b0010)?result2:
31                 (ALU_operation==4'b0011)?result3:
32                 (ALU_operation==4'b0100)?result4:
33                 (ALU_operation==4'b0101)?result5:
34                 (ALU_operation==4'b0110)?result6:

```

```

35         (ALU_operation==4'b0111)?result7:
36         (ALU_operation==4'b1000)?result8:
37         (ALU_operation==4'b1001)?result9:
38         (ALU_operation==4'b1010)?result10:
39         (ALU_operation==4'b1011)?result11:
40         (ALU_operation==4'b1100)?result12:
41         (ALU_operation==4'b1101)?result13:
42         (ALU_operation==4'b1110)?result14:
43         32'b0;
44     assign zero = ~(|res) ? 1'b1 : 1'b0;
45 endmodule

```

Controler 模块

```

1  // RISC-V Controller
2  `include "Lab4.vh"
3  module Controller (
4      input wire [4:0] Opcode,
5      input wire MIO_ready,
6      input wire [6:0] Fun7,
7      input wire [2:0] Fun3,
8      output reg CPU_MIO,
9      output reg [3:0] wea,
10     output reg [`IMM_SEL_WIDTH-1:0] ImmSel1,
11     output reg [`MEM2REG_WIDTH-1:0] MemtoReg,
12     output reg [`ALU_OP_WIDTH-1:0] ALU_Control,
13     output reg ALUSrc_B,
14     output reg Jump,
15     output reg Branch,
16     output reg Regwrite,
17     output reg sign,
18     output reg [1:0] byte_n,
19     output reg jump_choose,
20     output reg illegal_inst,
21     output reg csr_w,
22     output reg [`CSR_CHANGE_MODE_WIDTH-1:0] csr_wsc_mode,
23     output ecall,
24     output mret
25 );
26 assign ecall = {Fun7,Fun3,Opcode} == `ECALL;
27 assign mret = {Fun7,Fun3,Opcode} == `MRET;
28 always @(*) begin
29     case (Opcode)
30         `OPCODE_ALU: begin
31             CPU_MIO <= 1'b0;
32             wea <= `WEA_READ;
33             ImmSel1 <= 3'b0;
34             MemtoReg <= `MEM2REG_ALU;
35             case (Fun3)
36                 `FUNC_ADD: ALU_Control <= Fun7[5] ? `ALU_OP_SUB : `ALU_OP_ADD;
37                 `FUNC_SL: ALU_Control <= `ALU_OP_SLL;
38                 `FUNC_SLT: ALU_Control <= `ALU_OP_SLT;
39                 `FUNC_SLTU: ALU_Control <= `ALU_OP_SLTU;
40                 `FUNC_XOR: ALU_Control <= `ALU_OP_XOR;
41                 `FUNC_OR: ALU_Control <= `ALU_OP_OR;

```

```

42     `FUNC_AND: ALU_Control <= `ALU_OP_AND;
43     `FUNC_SR: ALU_Control <= Fun7[5] ? `ALU_OP_SRA : `ALU_OP_SRL;
44     default: ALU_Control <= 4'b0;
45 endcase
46 ALUSrc_B <= 1'b0;
47 Jump <= 1'b0;
48 Branch <= 1'b0;
49 RegWrite <= 1'b1;
50 sign <= 1'b0;
51 byte_n <= `WORD;
52 jump_choose <= `JUMP_PC_IMM;
53 illegal_inst <= 1'b0;
54 csr_w <= 1'b0;
55 csr_wsc_mode <= `CSR_WRITE;
56 end
57 `OPCODE_ALU_IMM: begin
58     CPU_MIO <= 1'b0;
59     wea <= `WEA_READ;
60     ImmSel1 <= `IMM_SEL_I;
61     MemtoReg <= `MEM2REG_ALU;
62     case (Fun3)
63         `FUNC_ADD: ALU_Control <= `ALU_OP_ADD;
64         `FUNC_SL: ALU_Control <= `ALU_OP_SLL;
65         `FUNC_SLT: ALU_Control <= `ALU_OP_SLT;
66         `FUNC_SLTU: ALU_Control <= `ALU_OP_SLTU;
67         `FUNC_XOR: ALU_Control <= `ALU_OP_XOR;
68         `FUNC_OR: ALU_Control <= `ALU_OP_OR;
69         `FUNC_AND: ALU_Control <= `ALU_OP_AND;
70         `FUNC_SR: ALU_Control <= Fun7[5] ? `ALU_OP_SRA : `ALU_OP_SRL;
71         default: ALU_Control <= 4'b0;
72     endcase
73     ALUSrc_B <= 1'b1;
74     Jump <= 1'b0;
75     Branch <= 1'b0;
76     RegWrite <= 1'b1;
77     sign <= 1'b1;
78     byte_n <= `WORD;
79     jump_choose <= `JUMP_PC_IMM;
80     illegal_inst <= 1'b0;
81     csr_w <= 1'b0;
82     csr_wsc_mode <= `CSR_WRITE;
83 end
84 `OPCODE_LOAD: begin
85     CPU_MIO <= 1'b1;
86     wea <= `WEA_READ;
87     ImmSel1 <= `IMM_SEL_I;
88     MemtoReg <= `MEM2REG_MEM;
89     ALU_Control <= 4'b0;
90     ALUSrc_B <= 1'b1;
91     Jump <= 1'b0;
92     Branch <= 1'b0;
93     RegWrite <= 1'b1;
94     sign <= ~(Fun3 == `FUNC_BYTE_UNSIGNED || Fun3 ==
`FUNC_HALF_UNSIGNED);
95     case (Fun3)
96         `FUNC_BYTE, `FUNC_BYTE_UNSIGNED: byte_n <= `BYTE;

```

```

97         `FUNC_HALF, `FUNC_HALF_UNSIGNED: byte_n <= `HALF;
98         `FUNC_WORD: byte_n <= `WORD;
99         default: byte_n <= `WORD;
100     endcase
101     jump_choose <= `JUMP_PC_IMM;
102     illegal_inst <= 1'b0;
103     csr_w <= 1'b0;
104     csr_wsc_mode <= `CSR_WRITE;
105 end
106 `OPCODE_STORE: begin
107     CPU_MIO <= 1'b1;
108     case (Fun3)
109         `FUNC_BYTE: wea <= `WEA_BYTE;
110         `FUNC_HALF: wea <= `WEA_HALF;
111         `FUNC_WORD: wea <= `WEA_WORD;
112         default: wea <= `WEA_READ;
113     endcase
114     ImmSel1 <= `IMM_SEL_S;
115     MemtoReg <= `MEM2REG_MEM;
116     ALU_Control <= 4'b0;
117     ALUSrc_B <= 1'b1;
118     Jump <= 1'b0;
119     Branch <= 1'b0;
120     RegWrite <= 1'b0;
121     sign <= 1'b1;
122     byte_n <= `WORD;
123     jump_choose <= `JUMP_PC_IMM;
124     illegal_inst <= 1'b0;
125     csr_w <= 1'b0;
126     csr_wsc_mode <= `CSR_WRITE;
127 end
128 `OPCODE_BRANCH: begin
129     CPU_MIO <= 1'b0;
130     wea <= `WEA_READ;
131     ImmSel1 <= `IMM_SEL_B;
132     MemtoReg <= `MEM2REG_ALU;
133     case (Fun3)
134         `FUNC_EQ: ALU_Control <= `ALU_OP_SUB;
135         `FUNC_NE: ALU_Control <= `ALU_OP_EQ;
136         `FUNC_LT: ALU_Control <= `ALU_OP_SGE;
137         `FUNC_GE: ALU_Control <= `ALU_OP_SLT;
138         `FUNC_LTU: ALU_Control <= `ALU_OP_SGEU;
139         `FUNC_GEU: ALU_Control <= `ALU_OP_SLTU;
140         default: ALU_Control <= 4'b0;
141     endcase
142     ALUSrc_B <= 1'b0;
143     Jump <= 1'b0;
144     Branch <= 1'b1;
145     RegWrite <= 1'b0;
146     sign <= 1'b1;
147     byte_n <= `WORD;
148     jump_choose <= `JUMP_PC_IMM;
149     illegal_inst <= 1'b0;
150     csr_w <= 1'b0;
151     csr_wsc_mode <= `CSR_WRITE;
152 end

```

```

153     `OPCODE_JAL: begin
154         CPU_MIO <= 1'b0;
155         wea <= `WEA_READ;
156         ImmSel1 <= `IMM_SEL_J;
157         jump_choose <= `JUMP_PC_IMM;
158         MemtoReg <= `MEM2REG_PC_PLUS;
159         ALU_Control <= `ALU_OP_ADD;
160         ALUSrc_B <= 1'b1;
161         Jump <= 1'b1;
162         Branch <= 1'b0;
163         RegWrite <= 1'b1;
164         sign <= 1'b1;
165         byte_n <= `WORD;
166         illegal_inst <= 1'b0;
167         csr_w <= 1'b0;
168         csr_wsc_mode <= `CSR_WRITE;
169     end
170     `OPCODE_JALR: begin
171         CPU_MIO <= 1'b0;
172         wea <= `WEA_READ;
173         ImmSel1 <= `IMM_SEL_I;
174         MemtoReg <= `MEM2REG_PC_PLUS;
175         ALU_Control <= `ALU_OP_ADD;
176         ALUSrc_B <= 1'b1;
177         Jump <= 1'b1;
178         Branch <= 1'b0;
179         RegWrite <= 1'b1;
180         sign <= 1'b1;
181         byte_n <= `WORD;
182         jump_choose <= `JUMP_ALU;
183         illegal_inst <= 1'b0;
184         csr_w <= 1'b0;
185         csr_wsc_mode <= `CSR_WRITE;
186     end
187     `OPCODE_LUI: begin
188         CPU_MIO <= 1'b0;
189         wea <= `WEA_READ;
190         ImmSel1 <= `IMM_SEL_U;
191         MemtoReg <= `MEM2REG_ALU;
192         ALU_Control <= `ALU_OP_R2;
193         ALUSrc_B <= 1'b1;
194         Jump <= 1'b0;
195         Branch <= 1'b0;
196         RegWrite <= 1'b1;
197         sign <= 1'b1;
198         byte_n <= `WORD;
199         jump_choose <= `JUMP_PC_IMM;
200         illegal_inst <= 1'b0;
201         csr_w <= 1'b0;
202         csr_wsc_mode <= `CSR_WRITE;
203     end
204     `OPCODE_AUIPC: begin
205         CPU_MIO <= 1'b0;
206         wea <= `WEA_READ;
207         ImmSel1 <= `IMM_SEL_U;
208         MemtoReg <= `MEM2REG_IMM_PC;

```

```

209     ALU_Control <= `ALU_OP_ADD;
210     ALUSrc_B <= 1'b1;
211     Jump <= 1'b0;
212     Branch <= 1'b0;
213     RegWrite <= 1'b1;
214     sign <= 1'b1;
215     byte_n <= `WORD;
216     jump_choose <= `JUMP_PC_IMM;
217     illegal_inst <= 1'b0;
218     csr_w <= 1'b0;
219     csr_wsc_mode <= `CSR_WRITE;
220 end
221 `OPCODE_CSR: begin
222     CPU_MIO <= 1'b0;
223     wea <= `WEA_READ;
224     ImmSel1 <= `IMM_SEL_CSR;
225     MemtoReg <= `MEM2REG_CSR;
226     ALUSrc_B <= 1'b1;
227     RegWrite <= 1'b1;
228     case(Fun3)
229         `FUNC_CSRRW:
230             begin
231                 ALU_Control <= `ALU_OP_R1;
232                 csr_wsc_mode <= `CSR_WRITE;
233             end
234         `FUNC_CSRRS:
235             begin
236                 ALU_Control <= `ALU_OP_R1;
237                 csr_wsc_mode <= `CSR_OR;
238             end
239         `FUNC_CSRRC:
240             begin
241                 ALU_Control <= `ALU_OP_R1;
242                 csr_wsc_mode <= `CSR_CLEAR;
243             end
244         `FUNC_CSRRWI:
245             begin
246                 ALU_Control <= `ALU_OP_R2;
247                 csr_wsc_mode <= `CSR_WRITE;
248             end
249         `FUNC_CSRRSI:
250             begin
251                 ALU_Control <= `ALU_OP_R2;
252                 csr_wsc_mode <= `CSR_OR;
253             end
254         `FUNC_CSRRCI:
255             begin
256                 ALU_Control <= `ALU_OP_R2;
257                 csr_wsc_mode <= `CSR_CLEAR;
258             end
259         default:
260             begin
261                 ALU_Control <= `ALU_OP_R1;
262                 csr_wsc_mode <= `CSR_BYPASS;
263             end
264     endcase

```

```

265     Jump <= 1'b0;
266     Branch <= 1'b0;
267     sign <= 1'b0;
268     byte_n <= `WORD;
269     jump_choose <= `JUMP_PC_IMM;
270     illegal_inst <= 1'b0;
271     csr_w <= 1'b1;
272 end
273 default:
274 begin
275     CPU_MIO <= 1'b0;
276     wea <= `WEA_READ;
277     ImmSel1 <= 3'b0;
278     MemtoReg <= `MEM2REG_CSR;
279     ALU_Control <= 4'b0;
280     ALUSrc_B <= 1'b0;
281     Jump <= 1'b0;
282     Branch <= 1'b0;
283     RegWrite <= 1'b0;
284     sign <= 1'b0;
285     byte_n <= `WORD;
286     jump_choose <= `JUMP_PC_IMM;
287     illegal_inst <= 1;
288     csr_w <= 1'b1;
289     csr_wsc_mode <= `CSR_BYPASS;
290 end
291
292 endcase
293 end
294 endmodule

```

DataPath 模块

```

1
2 `include "Lab4.vh"
3 module DataPath (
4     `RegFile_Regs_Outputs
5     input wire clk,
6     input wire rst,
7     input wire [31:0] inst_field,
8     input wire [31:0] Data_in,
9     input wire [`IMM_SEL_WIDTH-1:0] ImmSel1,
10    input wire ALUSrc_B,
11    input wire [`MEM2REG_WIDTH-1:0] MemtoReg,
12    input wire Jump,
13    input wire Branch,
14    input wire RegWrite,
15    input wire [`ALU_OP_WIDTH-1:0] ALU_Control,
16    input wire sign,
17    input wire [1:0] byte_n,
18    input wire jump_choose,
19    input wire illegal_inst,
20    input wire csr_w,
21    input wire [`CSR_CHANGE_MODE_WIDTH-1:0] csr_wsc_mode,
22    input wire ecall,

```



```

23     input wire mret,
24     input wire INT,
25     output wire [31:0] PC_out,
26     output wire [31:0] Data_out,
27     output wire [31:0] Addr_out,
28     output wire trap_normal_change,
29     `CSR_OUTPUTS
30 );
31 reg [31:0] PC = 32'h4;
32 wire [31:0] imm;
33 ImmGen U1 (
34     .ImmSel1(ImmSel1),
35     .inst_field(inst_field),
36     .sign(1'b1),
37     .Imm(imm)
38 );
39 wire [31:0] Rs1_data, Rs2_data;
40 wire [4:0] Rs1_addr, Rs2_addr, W_addr;
41 wire [31:0] W_data;
42
43 wire [31:0] ALU_out;
44 wire [11:0] CSR_raddr = inst_field[31:20];
45 wire [31:0] CSR_wdata = ALU_out;
46 wire [31:0] CSR_waddr = inst_field[31:20];
47 wire [31:0] PC_change;
48 wire [31:0] CSR_rdata;
49
50 RV_INT U5 (
51     .clk(clk),
52     .rst(rst),
53     .INT(INT),
54     .ecall(ecall),
55     .mret(mret),
56     .illegal_inst(illegal_inst),
57     .PC(PC),
58     .raddr(CSR_raddr),
59     .waddr(CSR_waddr),
60     .wdata(CSR_wdata),
61     .csr_w(csr_w),
62     .csr_wsc_mode(csr_wsc_mode),
63     .trap_normal_change(trap_normal_change),
64     .PC_change(PC_change),
65     .rdata(CSR_rdata),
66     .inst_field(inst_field),
67     `CSR_ARGUMENTS
68 );
69 assign Rs1_addr = inst_field[19:15];
70 assign Rs2_addr = inst_field[24:20];
71 assign W_addr   = inst_field[11:7];
72 Regs U2 (
73     `RegFile_Regs_Arguments
74     .clk(clk),
75     .rst(rst),
76     .Rs1_addr(Rs1_addr),
77     .Rs2_addr(Rs2_addr),
78     .Wt_addr(W_addr),

```

```

79     .Wt_data(W_data),
80     .RegWrite(RegWrite & !trap_normal_change),
81     .Rs1_data(Rs1_data),
82     .Rs2_data(Rs2_data)
83 );
84 wire [31:0] adder_2;
85 wire zero;
86 ALU U3 (
87     .A(Rs1_data),
88     .B(adder_2),
89     .ALU_operation(ALU_Control),
90     .res(ALU_out),
91     .zero(zero)
92 );
93 assign Addr_out = ALU_out;
94 assign Data_out = Rs2_data;
95 wire [31:0] mem_out;
96 extend U4 (
97     .byte_n(byte_n),
98     .in(Data_in >> ((Addr_out % 4) << 3)),
99     .sign(sign),
100     .mem_data(mem_out)
101 );
102
103 assign adder_2 = ALUSrc_B ? imm : Rs2_data;
104
105 wire [31:0] PC_add_4 = PC + 4;
106 wire [31:0] PC_imm = imm + PC;
107 wire Branch_final = Branch & zero;
108 wire [31:0] PC_branch = Branch_final ? PC_imm : PC_add_4;
109 wire [31:0] PC_jump = Jump ? ((jump_choose == `JUMP_ALU) ? ALU_out :
PC_imm) : PC_branch;
110 assign PC_out = trap_normal_change ? PC_change : PC_jump;
111
112 always @(posedge clk or posedge rst) begin
113     if (rst) begin
114         PC <= 32'hFFFFFFFC;
115     end else begin
116         PC <= PC_out;
117     end
118 end
119
120 assign W_data=MemoReg==`MEM2REG_MEM?mem_out:
121         MemoReg==`MEM2REG_ALU?ALU_out:
122         MemoReg==`MEM2REG_PC_PLUS?PC_add_4:
123         MemoReg==`MEM2REG_IMM_PC?PC_imm:
124         MemoReg==`MEM2REG_CSR?CSR_rdata:32'b0;
125
126 endmodule

```

下板验证

注意这里的 PC 为正在执行指令的下一条指令

ecall 指令的退出与进入:

RV32I Single Cycle CPU

pc: 00000044 inst: 30002FF3

x0: 00000000	ra: 00000003	sp: 00000005	gp: 00000008	tp: 00000000
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: 00 rs2_val: 00000000

rd: 00 reg_i_data: 00000000 reg_wen: 0

is_imm: 0 is_auiopc: 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: 00000000

mem_wen: 0 mem_ren: 0

dmem_o_data: 00000000 dmem_i_data: 00000000 dmem_addr: 00000000

csr_wen: 0 csr_ind: 000 csr_ctrl: 0 csr_r_data: 00000000

mstatus: 00000008 mcause: 00000000 mepc: 00000000 mtval: 00000000

mtvec: 00000044 mie: 00000000 mip: 00000000

RV32I Single Cycle CPU

pc: 0000005C inst: 00BD0D13

x0: 00000000	ra: 00000003	sp: 00000005	gp: 00000008	tp: 00000000
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: 00000044	t3: 00000000	t4: 80000003
t5: 00000018	t6: 00000080			

rs1: 00 rs1_val: 00000000

rs2: 00 rs2_val: 00000000

rd: 00 reg_i_data: 00000000 reg_wen: 0

is_imm: 0 is_auiopc: 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: 00000000

mem_wen: 0 mem_ren: 0

dmem_o_data: 00000000 dmem_i_data: 00000000 dmem_addr: 00000000

csr_wen: 0 csr_ind: 000 csr_ctrl: 0 csr_r_data: 00000000

mstatus: 00000080 mcause: 80000003 mepc: 00000018 mtval: 00000000

mtvec: 00000044 mie: 00000000 mip: 00000000

RV32I Single Cycle CPU

pc: 00000020 inst: 00308133

x0: 00000000	ra: 00000003	sp: 00000005	gp: 00000008	tp: 00000000
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: 00000044	t3: 00000000	t4: 80000003
t5: 0000001C	t6: 00000080			

rs1: 00 rs1_val: 00000000

rs2: 00 rs2_val: 00000000

rd: 00 reg_i_data: 00000000 reg_wen: 0

is_imm: 0

is_auiopc: 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: 00000000

mem_wen: 0

mem_ren: 0

dmem_o_data: 00000000 dmem_i_data: 00000000 dmem_addr: 00000000

csr_wen: 0

csr_ind: 000

csr_ctrl: 0

csr_r_data: 00000000

mstatus: 00000088 mcause: 00000000

mepc: 0000001C mtval: 00000000

mtvec: 00000044 mie: 00000000 mip: 00000000

硬件中断的进入与退出:

RV32I Single Cycle CPU

pc: 00000044 inst: 30002FF3

x0: 00000000	ra: 0000000D	sp: 00000015	gp: 00000008	tp: 00000000
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: 00000044	t3: 00000000	t4: 80000003
t5: 0000001C	t6: 00000080			

rs1: 00 rs1_val: 00000000

rs2: 00 rs2_val: 00000000

rd: 00 reg_i_data: 00000000 reg_wen: 0

is_imm: 0

is_auiopc: 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: 00000000

mem_wen: 0

mem_ren: 0

dmem_o_data: 00000000 dmem_i_data: 00000000 dmem_addr: 00000000

csr_wen: 0

csr_ind: 000

csr_ctrl: 0

csr_r_data: 00000000

mstatus: 00000088 mcause: 00000000

mepc: 0000001C mtval: 00000000

mtvec: 00000044 mie: 00000000 mip: 00000000

RV32I Single Cycle CPU

pc: 0000005C inst: 00BD0D13

x0: 00000000	ra: 0000000D	sp: 00000015	gp: 00000008	tp: 00000000
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: 00000044	t3: 00000000	t4: 8000000B
t5: 00000024	t6: 00000080			

rs1: 00 rs1_val: 00000000

rs2: 00 rs2_val: 00000000

rd: 00 reg_i_data: 00000000 reg_wen: 0

is_imm: 0 is_auiopc: 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: 00000000

mem_wen: 0 mem_ren: 0

dmem_o_data: 00000000 dmem_i_data: 00000000 dmem_addr: 00000000

csr_wen: 0 csr_ind: 000 csr_ctrl: 0 csr_r_data: 00000000

mstatus: 00000080 mcause: 8000000B mepc: 00000024 mtval: 00000000

mtvec: 00000044 mie: 00000000 mip: 00000000

RV32I Single Cycle CPU

pc: 00000028 inst: 003100B3

x0: 00000000	ra: 0000000D	sp: 00000015	gp: 00000008	tp: 00000000
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: 00000044	t3: 00000000	t4: 8000000B
t5: 00000024	t6: 00000080			

rs1: 00 rs1_val: 00000000

rs2: 00 rs2_val: 00000000

rd: 00 reg_i_data: 00000000 reg_wen: 0

is_imm: 0 is_auiopc: 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: 00000000

mem_wen: 0 mem_ren: 0

dmem_o_data: 00000000 dmem_i_data: 00000000 dmem_addr: 00000000

csr_wen: 0 csr_ind: 000 csr_ctrl: 0 csr_r_data: 00000000

mstatus: 00000088 mcause: 00000000 mepc: 00000024 mtval: 00000000

mtvec: 00000044 mie: 00000000 mip: 00000000

非法指令中断的进入与退出,注意 `mtval` 正确保存了非法指令的值:

RV32I Single Cycle CPU

pc: 00000044 inst: 30002FF3

x0: 00000000	ra: 00000037	sp: 00000059	gp: 00000090	tp: 00000000
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: 00000044	t3: 00000000	t4: 8000000B
t5: 00000024	t6: 00000080			

rs1: 00 rs1_val: 00000000

rs2: 00 rs2_val: 00000000

rd: 00 reg_i_data: 00000000 reg_wen: 0

is_imm: 0 is_auiopc: 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: 00000000

mem_wen: 0 mem_ren: 0

dmem_o_data: 00000000 dmem_i_data: 00000000 dmem_addr: 00000000

csr_wen: 0 csr_ind: 000 csr_ctrl: 0 csr_r_data: 00000000

mstatus: 00000088 mcause: 00000000 mepc: 00000024 mtval: 00000000

mtvec: 00000044 mie: 00000000 mip: 00000000

RV32I Single Cycle CPU

pc: 0000005C inst: 00BD0D13

x0: 00000000	ra: 00000037	sp: 00000059	gp: 00000090	tp: 00000000
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: 00000044	t3: FFFFFFFF	t4: 00000002
t5: 00000034	t6: 00000080			

rs1: 00 rs1_val: 00000000

rs2: 00 rs2_val: 00000000

rd: 00 reg_i_data: 00000000 reg_wen: 0

is_imm: 0 is_auiopc: 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: 00000000

mem_wen: 0 mem_ren: 0

dmem_o_data: 00000000 dmem_i_data: 00000000 dmem_addr: 00000000

csr_wen: 0 csr_ind: 000 csr_ctrl: 0 csr_r_data: 00000000

mstatus: 00000080 mcause: 00000002 mepc: 00000034 mtval: FFFFFFFF

mtvec: 00000044 mie: 00000000 mip: 00000000


```

RV32I Single Cycle CPU

pc: 0000003C      inst: 00308133

x0: 00000000    ra: 00000037    sp: 00000059    gp: 00000090    tp: 00000000
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: FFFFFFFF    s11: 00000044    t3: FFFFFFFF    t4: 00000002
t5: 00000038    t6: 00000080

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

is_imm: 0        is_auiopc: 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: 00000000

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

csr_wen: 0        csr_ind: 000    csr_ctrl: 0        csr_r_data: 00000000
mstatus: 00000088    mcause: 00000000    mepc: 00000038    mtval: FFFFFFFC7
mtvec: 00000044    mie: 00000000    mip: 00000000

```

思考题

在涉及到一个大立即数的读入时，我们经常能想到使用 lui & addi 来实现，比如下面这段代码就将 0x22223333 赋给了 t0:

```

lui t0, 0x22223
addi t0, t0, 0x333

```

你是否能通过以下代码得到 0xDEADBEEF? 如果你觉得不能的话，先解释为什么不能，再修改代码中的一个字符，使得以下代码有效地得到 0xDEADBEEF。（如果你觉得可以的话，请重新学习 RISC-V ISA）

```

lui t1, 0xDEADB
addi t1, t1, 0xEEF

```

btw, 如果你把上边代码放到 Venus 上，会发现它给了你一个报错，不要理会它，它理解错了。

回答:

不行,因为立即数是有符号扩展的,EEF 会扩展为 0xFFFFFEEF,加上 0xDEADB000 得到 0xDEADAEEF,当立即数实际解释为负数时,会扩展为负数造成前五位产生一个退位,故无法得到 0xDEADBEEF.修改 0xDEADB 为 0xDEADC 即可得到 0xDEADBEEF.