同济大学计算机系

计算机组成原理实验报告

学　　号 2351753

姓　　名 黄保翔

专　　业 计算机科学与技术

授课老师 张冬冬

一、实验内容

通过了解 MIPS 架构中 CP0 中断异常的实现原理，Verilog 实现 54 条 CPU 指令中的 break,syscall,eret,mfc0,mtc0,实现 CP0 的异常处理。

二、模块建模

CP0 建模，接口定义如下

```
module CP0(
    input clk,
    input rst,
    input mfc0,              // CPU 指令 Mfc0
    input mtc0,              // CPU 指令 Mtc0
    input [31:0]pc,
    input [4:0] Rd,          // 指定 Cp0 寄存器
    input [31:0] wdata,      // 数据从 GP 寄存器到 CP0 寄存器
    input exception,
    input eret,              // 指令 ERET (Exception Return)
    input [4:0]cause,
    input intr,
    output [31:0] rdata,     //数据从 CP0 寄存器到 GP 寄存器
    output [31:0] status,
    output reg timer_int,
    output [31:0]exc_addr    // 异常起始地址
);
```

该模块主要满足以下功能

异常发生时保存当前指令的地址作为返回地址，响应异常时把 Status 寄存器的内容左移 5 位关中断， 执行中断处理程序时保存 Status 寄存器内容，中断返回时写回

代码：

```
module CP0(
 input clk,
 input rst,
 input mfc0, //CPU instruction is Mfc0
 input mtc0, //CPU instruction is Mtc0
 input [31:0] pc,
 input [4:0] addr, //Specifies CP0 register
 input [31:0] data, //Data from GP register to replace CP0 register
 input exception,
 input eret, //instruction is ERET(Exception Return)
 input [4:0] cause,
 output [31:0] rdata, //Data from CP0 register for GP register
```

```
 output [31:0] status,
 output [31:0] exc_addr //Address for PC output [31:0]exc_addr // Addre
ss for PC at the beginning of an exception
);
 reg [31:0] CP0_reg [0:31];
 assign status = CP0_reg[12];
 always @(posedge clk or posedge rst)
  begin
  if(rst)
  begin
   CP0_reg[0] <= 32'b0;
   CP0_reg[1] <= 32'b0;
   CP0_reg[2] <= 32'b0;
   CP0_reg[3] <= 32'b0;
   CP0_reg[4] <= 32'b0;
   CP0_reg[5] <= 32'b0;
   CP0_reg[6] <= 32'b0;
   CP0_reg[7] <= 32'b0;
   CP0_reg[8] <= 32'b0;
   CP0_reg[9] <= 32'b0;
   CP0_reg[10] <= 32'b0;
   CP0_reg[11] <= 32'b0;
   CP0_reg[12] <= 32'b0;
   CP0_reg[13] <= 32'b0;
   CP0_reg[14] <= 32'b0;
   CP0_reg[15] <= 32'b0;
   CP0_reg[16] <= 32'b0;
   CP0_reg[17] <= 32'b0;
   CP0_reg[18] <= 32'b0;
   CP0_reg[19] <= 32'b0;
   CP0_reg[20] <= 32'b0;
   CP0_reg[21] <= 32'b0;
   CP0_reg[22] <= 32'b0;
   CP0_reg[23] <= 32'b0;
   CP0_reg[24] <= 32'b0;
   CP0_reg[25] <= 32'b0;
   CP0_reg[26] <= 32'b0;
   CP0_reg[27] <= 32'b0;
   CP0_reg[28] <= 32'b0;
   CP0_reg[29] <= 32'b0;
   CP0_reg[30] <= 32'b0;
   CP0_reg[31] <= 32'b0;
  end
  else
```

```
  begin
   if(mtc0)
   begin
    CP0_reg[addr] <= data;
   end
   else if(exception) //SYSCALL,BREAK,TEQ
   begin
    CP0_reg[12] <= (status << 5); //status
    CP0_reg[13] <= {24'b0, cause, 2'b0}; //cause
    CP0_reg[14] <= pc; //EPC
   end
  else if(eret)
   begin
    CP0_reg[12] <= (status >> 5); //status
   end
  end
 end
  assign exc_addr = eret ? (CP0_reg[14] + 4) : 32'h00400004;
  assign rdata = mfc0 ? CP0_reg[addr] : 32'hzzzzzzzz;
endmodule
```

三、测试模块建模

Testbench 主要能满足以下测试

1. 测试 `mtc0` 写入和 `mfc0` 读出是否有效
2. 进行中断查看中断状态是否写入 EPC 和 CAUSE
3. 测试 ERET 指令，测试 status 是否恢复，以及 exc_addr 是否正确

```
module CP0_testbench;
  // Inputs
  reg clk;
  reg rst;
  reg mfc0;
  reg mtc0;
  reg [31:0] pc;
  reg [4:0] addr;
  reg [31:0] data;
  reg exception;
  reg eret;
  reg [4:0] cause;
  // Outputs
  wire [31:0] rdata;
  wire [31:0] status;
  wire [31:0] exc_addr;
```

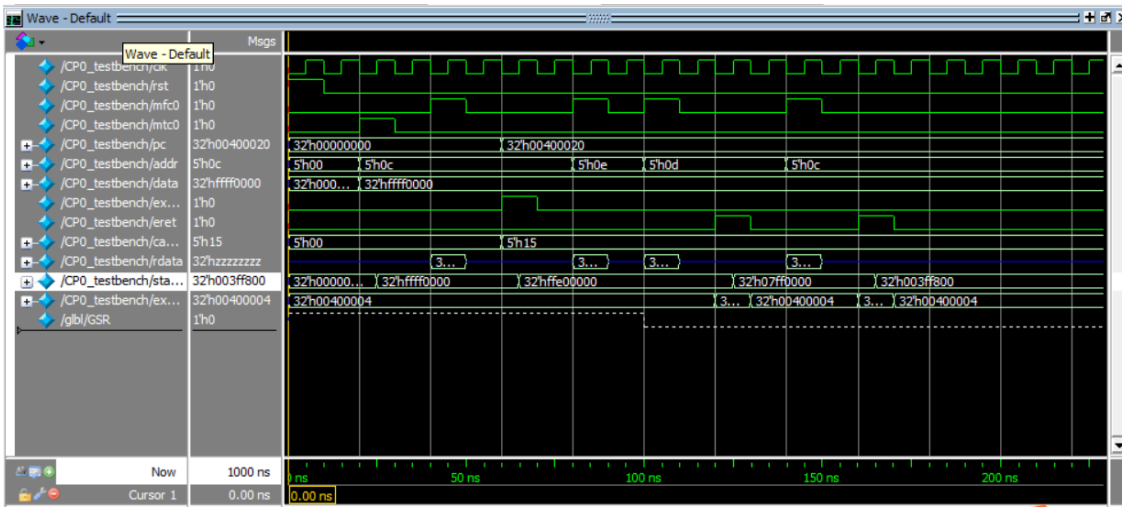
```
CP0 uut (
  .clk(clk),
  .rst(rst),
  .mfc0(mfc0),
  .mtc0(mtc0),
  .pc(pc),
  .addr(addr),
  .data(data),
  .exception(exception),
  .eret(eret),
  .cause(cause),
  .rdata(rdata),
  .status(status),
  .exc_addr(exc_addr)
);
// Clock generation
initial begin
  clk = 0;
  forever #5 clk = ~clk;
end
initial begin

  // 初始化
  rst = 1; mfc0 = 0; mtc0 = 0; exception = 0; eret = 0;
  addr = 0; data = 0; pc = 0; cause = 0;
  #10 rst = 0;
  //测试写入和读出是否有效
  // Write to CP0 register 12 (status) using MTC0
  #10 addr = 5'd12;
      data = 32'hffff_0000;
      mtc0 = 1;
  #10 mtc0 = 0;
  //测试读出
  // Read from CP0 register 12 using MFC0
  #10 mfc0 = 1;
  #10 $display("Read CP0[12] = 0x%08h (Expect: ffff0000)", rdata);
      mfc0 = 0;
  // 进行中断
  // Trigger an exception at PC = 0x00400020 with cause = 5'b10101
  #10 pc = 32'h00400020;
      cause = 5'b10101;
      exception = 1;
  #10 exception = 0;
  //查看中断状态是否写入
```

```
    // Check EPC and cause
    //读取 EPC
    #10 addr = 5'd14; mfc0 = 1; // EPC
    #10 $display("EPC = 0x%08h (Expect: 00400020)", rdata);
        mfc0 = 0;
    //读取 Status
    #10 addr = 5'd13; mfc0 = 1; // cause
    #10 $display("Cause = 0x%08h (Expect: 00000054)", rdata);
        mfc0 = 0;
    // ERET instruction (return from exception)
    #10 eret = 1;
    #10 eret = 0;
    // Check updated status
    #10 addr = 5'd12; mfc0 = 1;
    #10 $display("Status after ERET = 0x%08h", rdata);
        mfc0 = 0;
    // Check exc_addr output
    #10 eret = 1;
    #10; // 等待一个时钟周期，让 CP0 有机会处理 eret
    $display("exc_addr = 0x%08h (Expect: 00400024)", exc_addr);
    eret = 0;
  end
endmodule
```

四、实验结果

（该部分可截图说明，要求 logisim 逻辑验证图、modelsim 仿真波形图、以及下板后的实验结果贴图（实验步骤中没有下板要求的实验，不需要下板贴图））

波形图：



打印输出

```
# Read CP0[12] = 0xffff0000 (Expect: ffff0000)
# EPC = 0x00400020 (Expect: 00400020)
# Cause = 0x00000054 (Expect: 00000054)
# Status after ERET = 0x07ff0000
# exc_addr = 0x00400024 (Expect: 00400024)

VSIM 2>
```

CP0 正确写入

EPC 在未 eret 时为输入的 PC，eret 变为 1 时，输出 exc_addr 为 PC+4

在进行中断后（exception = 1）输出 status 和 Cause 均正确