Lab 2

李秋宇 3220103373

Design

需要实现的CSR有`mstatus`，`mtvec`，`mepc`，`mcause`，`mtval`，根据RISC-V手册，可得它们的CSR地址为

```Verilog
parameter MSTATUS = 12'h300;
parameter MTVEC   = 12'h305;
parameter MEPC    = 12'h341;
parameter MCAUSE  = 12'h342;
parameter MTVAL   = 12'h343;
parameter MIP     = 12'h344;
```

本实验中，模块`CSRRegs`内部完成地址映射，每次需要调用对应的寄存器就直接根据上述定义的常数作为地址传入进行访问

需要实现的异常中断分为：

- 访问错误`l_access_fault`和`s_access_fault`
- 环境调用`ecall_m`
- 非法指令`illegal_inst`
- 外部中断`interrupt`

记录异常类型：

```Verilog
parameter illegalInstructionType = 2'd0;
parameter ecallType = 2'd1;
parameter interruptType = 2'd2;
parameter mretType = 2'd3;
assign exceptionType =  (illegalInstructionType & {2{illegal_inst | l_access_fault | s_access_fault}}) |
                        (ecallType & {2{ecall_m}}) |
                        (interruptType & {2{interrupt & mstatus[3]}}) |
                        (mretType & {2{mret}});

wire handleTrap;
```

```
assign handleTrap = illegal_inst | l_access_fault | s_access_fault |
                    ecall_m | interrupt & mstatus[3] | mret | (state ≠
3'd0);
```

而处理访问错误的异常发生在MEM阶段，处理环境调用的异常和中断发生在WB阶段，处理非法指令的异常发生在ID阶段，这里涉及多个时钟周期，需要在多个时钟周期内完成对CSR的操作，因此需要设计一个时序逻辑用于在多个时钟周期内处理CSR

定义状态寄存器`state`用来实现这个过程

状态的转化为以下逻辑

Verilog

```
// State transition logic.
always @(posedge handleTrap or posedge clk or posedge rst) begin
    if (rst) begin
        state ≤ 3'd0;
    end
    else if (state == 3'd0 & handleTrap) begin
        if(exceptionType ≠ interruptType) begin
            state ≤ 3'd1;
        end
        else begin
            state ≤ 3'd7;
            reg_epc_cur ≤ epc_cur;
            reg_epc_next ≤ epc_next;
        end
        thisType ≤ exceptionType;
        if (l_access_fault | s_access_fault) begin
            mtval ≤ eaddr;
        end else if (illegal_inst) begin
            mtval ≤ einst;
        end
        EPC ≤ epc_cur;
    end else if(state == 3'd7) begin
        state ≤ 3'd1;
    end else if (state == 3'd1) begin
        state ≤ 3'd2;
    end else if (state == 3'd2) begin
        state ≤ 3'd3;
    end else if (state == 3'd3) begin
        state ≤ 3'd4;
    end else if (state == 3'd4) begin
        state ≤ 3'd0;
    end
end
```

其中

- 没有陷入需要处理时处于`state == 3'd0`，此时一切正常
- 发生陷入后，根据有无外部中断决定状态跳转
  - 如果没有外部中断，直接转状态1
  - 如果有外部中断，由于外部中断是异步信号，因此需要gap一个周期等待同步
- 在状态1完成后进入状态2
- 状态2后进入状态3
- 状态3后进入状态4
- 状态4后恢复状态0

发生陷入后，记录`mstatus`，把先前的`MIE`值保留到`MPIE`位中，然后在`MEPC`阶段保存好PC值,，然后跳转到`mtvec`保存的中断处理程序的位置进行中断处理，然后根据异常来源设置`mcause`，最后返回时从`mepc`返回，并且更新`mstatus`

Verilog

```
always @(negedge clk) begin
    if (rst) begin
        csr_w <= 0;
    end
    case (state)
        3'd0: begin // CSR instruction.
            csr_w     <= csr_rw_in;
            csr_raddr <= csr_rw_addr_in;
            csr_waddr <= csr_rw_addr_in;
            csr_wsc   <= csr_wsc_mode_in;
            csr_wdata <= csr_w_imm_mux ? {27'b0, csr_w_data_imm} : csr_w_data_reg;
        end
        3'd1: begin // MEPC.
            csr_w <= thisType != 2'd3;
            csr_raddr <= thisType == mretType ? MEPC : MTVEC;
            csr_waddr <= MEPC;
            csr_wdata <= thisType == interruptType? reg_epc_next:epc_cur;
            csr_wsc   <= 2'b01;
            // Registers for next states.
            cause <=  ecall_m ? 32'hb : (l_access_fault ? 32'h5 :
                        (s_access_fault ? 32'h7 : (illegal_inst ? 32'h2 : 32'hb)));
        end
        3'd2: begin // MCAUSE.
            csr_w <= thisType != 2'd3;
            csr_waddr <= MCAUSE;
            csr_wdata <= cause;
            csr_wsc   <= 2'b01;
        end
        3'd3: begin // MTVAL.
            csr_w     <= 1;
```

```
31              csr_waddr <= MTVAL;
32              csr_wdata <= mtval;
33              csr_wsc <= 2'b01;
34          end
35          3'd4: begin // MSTATUS.
36              csr_w     <= 1;
37              csr_waddr <= MSTATUS;
38              csr_wdata <= thisType == mretType ?
39                           {mstatus[31:8], 1'b0, mstatus[6:4], mstatus[7],
    mstatus[2:0]}:
40                           {mstatus[31:8], mstatus[3], mstatus[6:4], 1'b0,
    mstatus[2:0]};
41          end
42      endcase
43  end
```

其中，在状态1进行 `mepc` 记录，同时在这个状态存储了一些当前的信号用于接下俩周期使用；在状态2记录 `mcause` ，在状态3记录 `mtval` ，在状态4记录 `mstatus`

如果发生中断需要处理，此时对应也要刷掉流水线寄存器中的正常指令的数据

Verilog

```
1   // Flush registers.
2   assign reg_FD_flush = (state == 3'd1) || (state == 3'd7);
3   assign reg_DE_flush = (state == 3'd1) || (state == 3'd7);
4   assign reg_EM_flush = (state == 3'd1) || (state == 3'd7);
5   assign reg_MW_flush = (state == 3'd1) || (state == 3'd7);
6   assign RegWrite_cancel = (state == 3'd1) || (state == 3'd7);
7   assign MemWrite_cancel = 0;
8
9   assign PC_redirect = csr_r_data_out;
10  assign redirect_mux = (state == 3'd1) || (state == 3'd7);
```

Exercises

1

精确异常和非精确异常的区别是什么？

精确异常是指发生异常时CPU能够保存当前指令的状态，可以精准识别出哪个指令发生了异常，此时CPU的状态和上下文都能够被完整保存，以便处理完异常后能够正确返回到原来的程序状态，而非精确异常则与之相反

2

阅读测试代码，第一次导致 trap 的指令是哪条？trap 之后的指令做了什么？如果实现了 U mode，并以 U mode 从头开始执行测试指令，会出现什么新的异常？

- 第一次导致trap的指令为 `ecall`
- trap之后的指令进行了
  1. 保存发生异常的程序计数器到 `x25`
  2. 保存异常原因到 `x27`
  3. 保存机器状态到 `x28`
  4. 保存引发异常的非法指令或访问错误的地址信息到 `x29`
- 从头开始以U mode执行，会导致程序结果不一致，即无法从发生异常的上下文开始执行，导致程序结果不符合预期，同时无法使用CSR寄存器的对应操作

3

为什么异常要传到最后一段即 WB 段后，才送入异常处理模块？可不可以一旦在某一段流水线发现了异常就送入异常处理模块，如果可以请说明异常处理模块应该如何处理异常；如果不可以，请说明理由。

- 因为发生异常后，必须等待当前指令执行完成后才进入异常处理，而当前指令执行完成发生在WB阶段之后，WB之后才代表当前指令执行完成，如果提前送入可能导致指令无法完整执行而导致错误
- 如果在某阶段发现了异常就直接送入异常处理模块，那么此时异常处理模块需要分别保存当前的上下文，转移控制权，同时需要实现状态恢复，复杂性相对增加