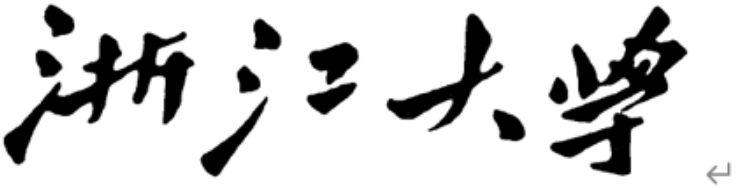

课程名称:	**计算机体系结构**
姓 名:	**杨吉祥**
学 院:	**计算机科学与技术学院**
系:	**竺可桢学院图灵班**
专 业:	**计算机科学与技术**
学 号:	**3230106222**
指导教师:	**常瑞**

一、设计思路

- CSRRegs.v

该部分进行对CSR寄存器的读写,根据`mstatus[3]`判断此时是否进入trap程序,如果为1表示未进入trap程序,那么此时就要判断是否发生异常或中断,如果发生异常或中断就要根据异常的情况去对不同的CSR寄存器进行不同的写操作。如果`mstatus[3]`为0表示已经进入trap程序,那这时候再发生中断的话,就不会因为中断而再次进入trap程序,而是要等该trap程序执行完后再进行处理。

`mstatus[3]`为`MIE`表示M模式全局中断使能,发生中断或异常进入trap程序后将`MIE`置为0,`mstatus[7]`为`MPIE`保存的前一个 M 模式全局中断使能,所以进入trap程序时将`MIE`的值赋给`MPIE`,`mstatus[12:11]`为`MPP`表示M 模式中断处理前的特权级别,`privilege`表示当前的特权级别,将`privilege`的值赋给`MPP`。执行`mret`指令结束trap程序时再将这些位恢复。

`mcause`存储了发生异常或中断的种类,根据不同的异常种类给`mcause`赋不同的值。

`mtval`当异常原因是内存访问越界时,`mtval`存储越界的地址,当异常原因是指令非法时,`mtval`存储非法指令。

执行CSR指令时,先根据指令求出实际要读写的地址,然后再根据CSR指令类型去求要写入的数据,需要注意如果rs1是0寄存器的话,这时候CSR寄存器就不能进行写操作。

```
`timescale 1ns/1ps
//0:mstatus 5:mtvec 9:mepc 10:mcause 11:mtval
module CSRRegs(
    input clk, rst,
    input INT,
    input ecall,
    input mret,
    input illegal_inst,
    input S_access_fault,
    input I_access_fault,
    input [31:0]addr,
    input is_csr,
    input [31:0]inst_in_MEM,inst_in_WB,
    input [31:0]PC_current,
    input [31:0]rs1_val,
    output[31:0] rdata,mstatus,mtvec,mcause,mtval,mepc,mscratch
);

reg [1:0] privilege;


wire [11:0]waddr = inst_in_MEM[31:20];
wire [11:0]raddr = inst_in_MEM[31:20];
wire raddr_valid = raddr[11:7] == 5'h6 && raddr[5:3] == 3'h0;
wire [3:0] raddr_map = (raddr[6] << 3) + raddr[2:0];
wire waddr_valid = waddr[11:7] == 5'h6 && waddr[5:3] == 3'h0;
wire [3:0] waddr_map = (waddr[6] << 3) + waddr[2:0];


reg [31:0]CSR[15:0];
always @(posedge clk or posedge rst)begin
        if(rst) begin
```

```
            privilege <= 2'b11;
            CSR[0]  <= 32'h1888;  // 0x300 mstatus
            CSR[1]  <= 0;  // 0x301 misa
            CSR[2]  <= 0;  // 0x302 medeleg
            CSR[3]  <= 0;  // 0x303 mideleg
            CSR[4]  <= 32'hfff;  // 0x304 mie
            CSR[5]  <= 0;  // 0x305 mtvec
            CSR[6]  <= 0;  // 0x306 mcounteren
            CSR[7]  <= 0;
            CSR[8]  <= 0;  // 0x340 mscratch
            CSR[9]  <= 0;  // 0x341 mepc
            CSR[10] <= 0;  // 0x342 mcause
            CSR[11] <= 0;  // 0x343 mtval
            CSR[12] <= 0;  // 0x344 mip
            CSR[13] <= 0;
            CSR[14] <= 0;
            CSR[15] <= 0;
        end
        else if(mstatus[3] == 1) begin
            if(INT)begin
                CSR[0][3]<=0;
                CSR[0][7]<=CSR[0][3];
                CSR[0][12:11]<=privilege; // 0x300 mstatus
                privilege<=2'b11;
                CSR[9]<=PC_current;  // 0x341 mepc
                CSR[10]<=32'h8000000B;  // 0x342 mcause
            end
            else if(ecall)begin
                CSR[0][3]<=0;
                CSR[0][7]<=CSR[0][3];
                CSR[0][12:11]<=privilege; // 0x300 mstatus
                privilege<=2'b11;
                CSR[9]<=PC_current;  // 0x341 mepc
                CSR[10]<=32'hB;     // 0x342 mcause
            end
            else if(illegal_inst)begin
                CSR[0][3]<=0;
                CSR[0][7]<=CSR[0][3];
                CSR[0][12:11]<=privilege; // 0x300 mstatus
                privilege<=2'b11;
                CSR[9]<=PC_current;  // 0x341 mepc
                CSR[10]<=32'h2;  // 0x342 mcause
                CSR[11]<=inst_in_WB;  // 0x343 mtval
            end
            else if(S_access_fault)begin
                CSR[0][3]<=0;
                CSR[0][7]<=CSR[0][3];
                CSR[0][12:11]<=privilege; // 0x300 mstatus
                privilege<=2'b11;
                CSR[9]<=PC_current;  // 0x341 mepc
                CSR[10]<=32'h7;  // 0x342 mcause
                CSR[11]<=addr;  // 0x343 mtval
            end
            else if(I_access_fault)begin
                CSR[0][3]<=0;
                CSR[0][7]<=CSR[0][3];
```

```
                CSR[0][12:11]<=privilege; // 0x300 mstatus
                privilege<=2'b11;
                CSR[9]<=PC_current;  // 0x341 mepc
                CSR[10]<=32'h5;  // 0x342 mcause
                CSR[11]<=addr;  // 0x343 mtval
            end
            else if(is_csr)begin
                if(inst_in_MEM[19:15]!=5'b0)begin
                    case(inst_in_MEM[14:12])
                    3'b001://CSRRW
                        CSR[waddr_map]<=rs1_val;
                    3'b010://CSRRS
                        CSR[waddr_map]<=CSR[waddr_map]|rs1_val;
                    3'b011://CSRRC
                        CSR[waddr_map]<=CSR[waddr_map]&(~rs1_val);
                    3'b101://CSRRWI
                        CSR[waddr_map]<={27'b0,inst_in_MEM[19:15]};
                    3'b110://CSRRSI
                        CSR[waddr_map]<=CSR[waddr_map]|
{27'b0,inst_in_MEM[19:15]};
                    3'b111://CSRRCI
                        CSR[waddr_map]<=CSR[waddr_map]&(~
{27'b0,inst_in_MEM[19:15]});
                    endcase
                end
            end
            else if(mret)begin
                CSR[0][3]<=CSR[0][7];
                CSR[0][7]<=1;
                CSR[0][12:11]<=2'b11; // 0x300 mstatus
                privilege<=CSR[0][12:11];
            end
        end
        else begin
            if(is_csr)begin
                if(inst_in_MEM[19:15]!=5'b0)begin
                    case(inst_in_MEM[14:12])
                    3'b001://CSRRW
                        CSR[waddr_map]<=rs1_val;
                    3'b010://CSRRS
                        CSR[waddr_map]<=CSR[waddr_map]|rs1_val;
                    3'b011://CSRRC
                        CSR[waddr_map]<=CSR[waddr_map]&(~rs1_val);
                    3'b101://CSRRWI
                        CSR[waddr_map]<={27'b0,inst_in_MEM[19:15]};
                    3'b110://CSRRSI
                        CSR[waddr_map]<=CSR[waddr_map]|
{27'b0,inst_in_MEM[19:15]};
                    3'b111://CSRRCI
                        CSR[waddr_map]<=CSR[waddr_map]&(~
{27'b0,inst_in_MEM[19:15]});
                    endcase
                end
            end
            else if(mret)begin
                CSR[0][3]<=1;
```

```
                CSR[0][7]<=CSR[0][3];
                CSR[0][12:11]<=privilege; // 0x300 mstatus
                privilege<=2'b11;
            end
        end
end


assign rdata=raddr_valid ? CSR[raddr_map] : 32'b0;
assign mstatus=CSR[0];
assign mtvec=CSR[5];
assign mcause=CSR[10];
assign mtval=CSR[11];
assign mepc=CSR[9];
assign mscratch=CSR[8];
endmodule
```

- 中间寄存器的stall信号

我的异常信号来源于WB阶段,当WB阶段检测出异常或者发生中断且此时中断使能为1时,这时候中间寄存器就要进行stall,将输出的指令变为`add x0,x0,x0`,即nop指令,而寄存器的读写发生在WB阶段,MEM阶段会对内存进行读写,所以将MEM阶段内存的写使能置为0,这样WB阶段前面那些阶段的指令都相当于没有执行。需要注意的是,当执行`mret`指令时也要进行stall,毕竟`mret`之后的指令也不能执行

```
wire
x=INT|ecall_out_MEM_WB|illegal_inst_out_MEM_WB|L_access_fault_out_MEM_WB|S_access
_fault_out_MEM_WB;

assign stall_IF_ID=stall||(x&&mstatus_MEM[3])||(mret_out_MEM_WB);
assign stall_ID_EX=(x&&mstatus_MEM[3])||(mret_out_MEM_WB);
assign stall_EX_MEM=(x&&mstatus_MEM[3])||(mret_out_MEM_WB);
assign stall_MEM_WB=(x&&mstatus_MEM[3])||(mret_out_MEM_WB);
```

- `mepc`的写入

对于异常，`mepc` 指向导致异常的指令;对于中断，`mepc` 指向中断处理后应该恢复执行的位置，且中断发生时 WB 阶段的指令需执行完毕。并且在trap处理程序中,如果是异常,`mepc`会加4,如果是中断,`mepc`不会加4。所以如果是中断的话,`mepc`应该来自此时MEM阶段的指令,如果是异常的话,`mepc`就来自于WB阶段的指令。

```
.PC_current(INT_MEM==1?PC_MEM:PC_WB),

CSR[9]<=PC_current;  // 0x341 mepc
```

- `MemWrite`和`RegWrite`的处理

当发生中断或异常或执行`mret`指令时,MEM阶段对内存的写使能应该置为0。发生中断时要执行完WB阶段的指令,而寄存器的写入是在WB阶段完成,所以发生中断时寄存器的写使能保持不变。发生异常时,只有`L_access_fault`的情况会对寄存器进行写,所以在这种情况下将寄存器写使能置为0,其他情况下寄存器写使能保持不变。

```
wire
x=INT|ecall_out_MEM_WB|illegal_inst_out_MEM_WB|L_access_fault_out_MEM_WB|S_access
_fault_out_MEM_WB;

assign MemWrite=(x||mret_out_MEM_WB)?4'b0000:MemWrite_out_MEM;

assign RegWrite_out_EX_MEM = (load_type_reg[1:0]!=0)&&(ALU_reg>127)?
0:RegWrite_reg;
```

二、思考题

1.

精确异常和非精确异常的主要区别在于异常发生时能否准确恢复程序的执行状态。

精确异常:发生异常时,能够准确地识别发生异常的指令,并且处理异常后,程序可以从该指令正确地继续执行。异常前的指令都已执行,异常后的指令都未执行。

非精确异常:发生异常时,无法准确识别是哪一条指令引发的,可能有多条指令已经部分执行,导致程序状态难以恢复。

2.

第一次导致trap的指令是`ecall`指令,trap之后的指令依次读取了mepc,mcause,mstatus,mtval的值,然后判断了mcause的最高位是否为1,判断不是1之后将`x26`的值加4后写入到`mepc`,最后用`mret`指令返回到异常指令的下一条指令。

3.

CSR寄存器只有在M Mode下才可以被访问,如果以U mode 从头开始执行测试指令,那么在`PC=0X2C`执行`csrrwi x1, mscratch, 0x10`指令时就会发生异常。

4.

当异常传入WB段后,才送入异常处理模块可以保证异常之前的指令都已经执行完,异常之后的指令都还未执行,从而实现精确异常,最后能准确地恢复程序的执行状态。如果在更早的阶段触发异常,可能导致部分指令已执行,部分未执行,导致非精确异常,使得恢复变得困难。我觉得可以在某一段流水线发现了异常就送入异常处理模块,只是可能需要多stall几个周期来保证异常之前的指令都已经执行完毕,从而实现精确异常。

三、感想

本次实验我还是在自己之前的框架上进行修改的,但是之前的UART没有显示相关CSR寄存器的值,导致我光是去理解之前的UART,然后去修改UART就花费了不少时间。我原以为之前单周期实现过中断会比较容易,但后来发现流水线的中断需要不断考虑写入的值来自于哪个阶段,还要考虑什么情况下要stall,最后还要连一堆的线。反正真的是麻烦了许多,还好最后也是成功完成了,希望自己再接再厉,顺利完成之后的实验。