计 算 机 体 系 结 构 （研 讨 课） 实 验 报 告

实验项目 prj4 小组编号 28 组员姓名 刘景平、张钰堃、付博宇

1 逻辑电路结构与仿真波形的截图及说明

1.1. exp12

1.1.1. 增加 CSR 模块

1. 需要为 CSR 模块单独写一个 verilog 文件，作为专用于 CSR 的寄存器堆，按照教材上的指导加入一系列输入输出信号, 并在头文件中加入每个 CSR 的寄存器号，每个 CSR 各部分的位置等信息：

1. 把所有的控制状态寄存器集中到一个模块中实现；
2. 模块接口分为用于指令访问的接口和与处理器核内部硬件电路逻辑直接交互的控制、状态信号接口两类；
3. 指令访问接口包含读使能（csr_re）[7]、寄存器号（csr_num）、寄存器读返回值（csr_rvalue）、写使能（csr_we）、写掩码（csr_wmask）和写数据（csr_wvalue）；
4. 与硬件电路逻辑直接交互的接口信号视需要各自独立定义，无须再统一编码，如送往预取指（pre-IF）流水级的异常处理入口地址 ex_entry、送往译码流水级的中断有效信号 has_int、来自写回流水级的 ertn 指令执行的有效信号 ertn_flush、来自写回流水级的异常处理触发信号 wb_ex 以及异常类型类型 wb_ecode、wb_esubcode 等。

图 1: 讲义上关于创建 CSR 模块的指导

```
`include "mycpu_head.vh"

module csr_reg(
    input                           clk,
    input                           reset,

    input [`WIDTH_CSR_NUM-1:0]       csr_num,           // Register num

    input                           csr_re,            // Read enable
    output              [31:0]      csr_rvalue,        // Read data
    output              [31:0]      ertn_pc,
    output              [31:0]      ex_entry,

    input                           csr_we,            // Write enable
    input               [31:0]      csr_wmask,         // Write mask
    input               [31:0]      csr_wvalue,        // Write data

    input                           wb_ex,             // Write-back le
    input               [31:0]      wb_pc,             // Exception PC
    input                           ertn_flush,        // ERTN instruct
    input               [5:0]       wb_ecode,          // Exception typ
    input               [8:0]       wb_esubcode,       // Exception typ
    input               [31:0]      wb_vaddr,
    input               [31:0]      coreid_in,

    output                          has_int,
    input               [7:0]       hw_int_in,
    input                           ipi_int_in
);
```

图 2: 创建 CSR 模块

```
`define CSR_ERA 14'h6
`define CSR_BADV 14'h7
`define CSR_EENTRY 14'hc
`define CSR_SAVE0 14'h30
`define CSR_SAVE1 14'h31
`define CSR_SAVE2 14'h32
`define CSR_SAVE3 14'h33
`define CSR_TID 14'h40
`define CSR_TCFG 14'h41
`define CSR_TVAL 14'h42
`define CSR_TICLR 14'h44


//CSR partition

//CSR_CRMD
`define CSR_CRMD_PLV 1:0
`define CSR_CRMD_IE 2:2
`define CSR_CRMD_DA 3:3
`define CSR_CRMD_PG 4:4
`define CSR_CRMD_DATF 6:5
`define CSR_CRMD_DATM 8:7
`define CSR_CRMD_ZERO 31:9

//CSR_PRMD
`define CSR_PRMD_PPLV 1:0
`define CSR_PRMD_PIE 2:2
`define CSR_PRMD_ZERO 31:3
```

图 3: 修改头文件

2. 根据讲义上的例子添加 CRMD、PRMD、ESTAT、ERA、EENTRY、SAVE0 3 八个控制状态寄存器。并按照指令集手册的要求添加关于 read value 的设置

```
reg [1:0] csr_prmd_pplv;
reg csr_prmd_pie;

always @(posedge clk)
    begin
        if(wb_ex)
            begin
                csr_prmd_pplv <= csr_crmd_plv;
                csr_prmd_pie  <= csr_crmd_ie;
            end
        else if(csr_we && csr_num == `CSR_PRMD)
            begin
                csr_prmd_pplv <= csr_wmask[`CSR_PRMD_PPLV] & csr_wvalue[`CSR_PRMD_PPLV]
                               | ~csr_wmask[`CSR_PRMD_PPLV] & csr_prmd_pplv;
                csr_prmd_pie  <= csr_wmask[`CSR_PRMD_PIE] & csr_wvalue[`CSR_PRMD_PIE]
                               | ~csr_wmask[`CSR_PRMD_PIE] & csr_prmd_pie;
            end
    end
```

图 4: PRMD 寄存器的设置

```
assign csr_tid_rvalue = csr_tid_tid;
assign csr_tcfg_rvalue = {csr_tcfg_initval, csr_tcfg_periodic, csr_tcfg_en};
assign csr_tval_rvalue = csr_tval;

assign csr_rvalue = {32{csr_num==`CSR_CRMD}} & csr_crmd_rvalue
                  | {32{csr_num==`CSR_PRMD}} & csr_prmd_rvalue
                  | {32{csr_num==`CSR_ECFG}} & csr_ecfg_rvalue
                  | {32{csr_num==`CSR_ESTAT}} & csr_estat_rvalue
                  | {32{csr_num==`CSR_ERA}} & csr_era_rvalue
                  | {32{csr_num==`CSR_BADV}} & csr_badv_rvalue
                  | {32{csr_num==`CSR_EENTRY}} & csr_eentry_rvalue
                  | {32{csr_num==`CSR_SAVE0}} & csr_save0_rvalue
                  | {32{csr_num==`CSR_SAVE1}} & csr_save1_rvalue
                  | {32{csr_num==`CSR_SAVE2}} & csr_save2_rvalue
                  | {32{csr_num==`CSR_SAVE3}} & csr_save3_rvalue
                  | {32{csr_num==`CSR_TID}} & csr_tid_rvalue
                  | {32{csr_num==`CSR_TCFG}} & csr_tcfg_rvalue
                  | {32{csr_num==`CSR_TVAL}} & csr_tval_rvalue;
```

图 5: 设置 rvalue

1.1.2. 修改五个流水线模块

1. 在 IF 中对 nextpc 进行修改，加入出现中断和例外，以及异常处理返回的情况

```
wire [31:0] next_pc; //nextpc from branch or sequence
assign next_pc = (has_int || wb_ex)? ex_entry : ertn_flush? ertn_pc :
    br_taken? br_target : seq_pc;
```

2. 在 ID 模块中加入对 CSR 读写指令有关的译码信号并增加和 CSR 有关的阻塞;

```
wire csr_crush;
assign csr_crush = (es_csr && (ex_crush1 || ex_crush2)) || (ms_csr &&
    (mem_crush1 || mem_crush2));
```

```
//inst[23:10] -- csr_num
wire [13:0] ds_csr_num;
assign ds_csr_num = inst[23:10];

wire ds_ex_syscall;
assign ds_ex_syscall = inst_syscall;

wire [14:0] ds_code;
assign ds_code = inst[14:0];

wire ds_csr;
assign ds_csr = inst_csrrd | inst_csrwr | inst_csrxchg;

wire ds_csr_write;
assign ds_csr_write = inst_csrwr || inst_csrxchg;

wire [31:0] ds_csr_wmask;
assign ds_csr_wmask = inst_csrxchg ? rj_value : 32'hffffffff;

wire ds_ertn_flush;
assign ds_ertn_flush = inst_ertn;
```

图 6: csr 读写译码

3. 在遇到 ertn 指令或者遇到中断和例外时，需要情况流水级间缓存：

```
reg [`WIDTH_DS_TO_ES_BUS-1:0] ds_to_es_bus_reg;
always @(posedge clk)
    begin
        if(reset)
            ds_to_es_bus_reg <= 0;
        else if(ertn_flush | has_int | wb_ex)
            ds_to_es_bus_reg <= 0;
        else if(ds_to_es_valid && es_allow_in)
            ds_to_es_bus_reg <= ds_to_es_bus;
        else if(ds_to_es_valid)
            ds_to_es_bus_reg <= ds_to_es_bus_reg;
            else
            ds_to_es_bus_reg <= 0;
    end
```

图 7: 清空流水级间缓存

4. 在最后一级 WB 级进行中断与异常的判断：

```
/*--------------------------link csr_reg---------------------*/
assign csr_num = ws_csr_num;
assign csr_re = 1'b1;
//input [31:0] csr_rvalue

assign csr_we = ws_csr_write;
assign csr_wvalue = ws_csr_wvalue;
assign csr_wmask = ws_csr_wmask;
assign ertn_flush = ws_ertn_flush;

assign wb_ex = ws_ex_syscall;    // assign wb_ex = ws_ex_syscall || ws_ex_xxx ||
assign wb_pc = ws_pc;

/*
 *deal with ecode and esubcode according to kind of ex
 *in task12, we just finish syscall
 */
assign wb_ecode = ws_ex_syscall ? 6'hb : 6'h0;          //syscall
assign wb_esubcode = ws_ex_syscall ? 9'h0 : 9'h0;       //syscall
```

图 8: 在最后一级判断中断与异常

1.2. exp13

1.2.1. 添加异常、中断支持

1. 根据讲义提供的思路，添加本次实验所要求的中断及异常的判定信号，以及对应的判定逻辑。

```
assign fs_exc_ADEF = inst_sram_en && (fetch_pc[1] | fetch_pc[0]);
```

```
assign    ds_exc_INE = ~(inst_add_w | inst_sub_w | inst_slt | inst_sltu
  | inst_nor | inst_and | inst_or | inst_xor | inst_slli_w
  | inst_srli_w | inst_srai_w | inst_addi_w | inst_ld_w | inst_st_w
  | inst_jirl | inst_b | inst_bl | inst_beq | inst_bne | inst_lu12i_w
  | inst_slti | inst_sltiu | inst_andi | inst_ori | inst_xori | inst_sll_w
  | inst_srl_w | inst_sra_w | inst_pcaddu12i | inst_mul_w | inst_mulh_w
  | inst_mulh_wu | inst_div_w | inst_div_wu | inst_mod_w | inst_mod_wu
  | inst_blt | inst_bge | inst_bltu | inst_bgeu | inst_st_b | inst_st_h
  | inst_ld_b | inst_ld_h | inst_ld_bu | inst_ld_hu
  | inst_csrrd | inst_csrwr | inst_csrxchg | inst_ertn | inst_syscall
  | inst_rdcntvl_w | inst_rdcntvh_w | inst_rdcntid | inst_break) && (ds_pc
     != 32'b0) ;


//exp13 ALE exception
  wire ld_st_w,ld_st_h;
  assign ld_st_w = es_st_op[0] | (es_ld_op[1:0] == 2'b0 & es_res_from_mem);
  assign ld_st_h = es_st_op[2] | es_ld_op[1];
  assign es_exc_ALE = ((ld_st_w & (es_unaligned_addr != 2'b0)) | (ld_st_h
     & (es_unaligned_addr[0]))) & es_valid;
  //exp13 int
  assign has_int = ((csr_estat_is[11:0] & csr_ecfg_lie[12:0]) != 12'b0)
     && (csr_crmd_ie == 1'b1);
```

根据讲义内容，异常相当于给异常流水线打上对应标签，因此需要将异常标签通过流水级间总线进行传递，这里以 EXE 级向 MEM 级传递为例

```
  //exp13
  assign es_to_ms_bus[173:173] = es_exc_ADEF;
  assign es_to_ms_bus[174:174] = es_exc_INE;
  assign es_to_ms_bus[175:175] = es_exc_ALE;
  assign es_to_ms_bus[176:176] = es_exc_break;
  assign es_to_ms_bus[177:177] = es_has_int;
  assign es_to_ms_bus[209:178] = es_vaddr;
```

2. 为了做到精确异常，需要实现异常时清空流水线，异常流水级后不能产生任何运行效果. 这里重点关注紧跟在异常流水级之后的 st 类指令。

```
//EX.v
  wire if_es_has_int;
  assign if_es_has_int = es_ex_syscall || es_ertn_flush || es_exc_ADEF ||
     es_exc_ALE || es_exc_INE || es_exc_break || es_has_int || wb_ex;
```

```
assign data_sram_en = 1'b1;
assign data_sram_wen = (es_mem_we && es_valid && ~has_int &&
    ~if_ms_has_int && ~wb_ex && ~if_es_has_int & ~ertn_flush &
    ~es_ertn_flush) ? w_strb : 4'b0000;
assign data_sram_addr = (es_mul_op != 0) ? {es_mul_result[31:2],2'b00} :
    {es_alu_result[31:2],2'b00};
assign data_sram_wdata = real_wdata;
```

1.2.2. 增加控制状态寄存器

讲义针对如何添加 ECFG，BADV，TID，TCFG，TVAL，TICLR 控制状态寄存器以及如何维护不同寄存器的各个域均有详细描述, 按照小组之前的代码框架对讲义上的实现方法进行微调，与 exp12 整体框架相同，具体代码也在书中多有提及，不再详细罗列。

1.2.3. 增加计时器相关指令

添加 rdcntvl.w,rdcntvh.w,rdcntid 指令

1. 添加上述指令相关的译码信号。

```
assign inst_rdcntid = op_31_26_d[6'h0] & op_25_22_d[4'h0] & op_21_20_d[2'h0]
    & op_19_15_d[5'h0] & (rk == 5'b11000) & (rd == 5'h0);
assign inst_rdcntvl_w = op_31_26_d[6'h0] & op_25_22_d[4'h0] &
    op_21_20_d[2'h0] & op_19_15_d[5'h0] & (rk == 5'b11000) & (rj == 5'h0);
assign inst_rdcntvh_w = op_31_26_d[6'h0] & op_25_22_d[4'h0] &
    op_21_20_d[2'h0] & op_19_15_d[5'h0] & (rk == 5'b11001) & (rj == 5'h0);
```

这里采用讲义建议，后两条指令在 EXE 阶段完成，rdcntid 指令在 ID 阶段和 csr 类指令共用一系列控制喜好，仅在 csr num 上做区分，与 csr 类指令一起在 WB 阶段完成,

```
//EX.v
assign es_cal_result = es_rdcntvl_w ? global_time_cnt[31:0] : es_rdcntvh_w ?
    global_time_cnt[63:32] :
    es_div_op[0] ? (es_div_op[2] ? es_div_result:es_mod_result ):
    ((es_mul_op != 0) ? es_mul_result : es_alu_result);

//ID.v
wire [13:0] ds_csr_num;
assign ds_csr_num = inst_rdcntid ? `CSR_TID :inst[23:10];
```

其余部分没有额外需要注意内容，不在赘述

2 实验过程中遇到的问题、对问题的思考过程及解决方法

2.1. exp12

在添加 ertn 指令过程中，因为 ertn 指令同样涉及类似于精确异常的操作，需要保证当 ertn 流水至 WB 级时 EX 级的 st 指令无法对内存产生影响。这里利用和异常类似的方法实现

```
assign data_sram_wen = (es_mem_we && es_valid && ~has_int &&
    ~if_ms_has_int && ~wb_ex && ~if_es_has_int & ~ertn_flush &
    ~es_ertn_flush) ? w_strb : 4'b0000;
```

2.2. exp13

1. 需要注意访存地址不对齐（ALE）的情况下，不能继续执行访存指令，需要直接进入异常

```
//WB.v
    assign ws_we = ws_gr_we && ws_valid && ~ws_exc_ALE;
```

而对于跳转至错误地址的跳转指令（ADEF），则需要先正常完成跳转指令，在下一步取值时判断异常。经过调试发现，ADEF 以 fetch pc 作为判断依据比以 next pc 作为判断依据更容易实现。因此这里采用前者。

```
//IF.v
    // ADEF exception
    wire fs_exc_ADEF;//pc not end with 2'b00
    assign fs_exc_ADEF = inst_sram_en && (fetch_pc[1] | fetch_pc[0]);
```

2. 在助教的帮助下，学会了利用金标文件调试。金标文件相较于汇编文件，能够更好的展现程序的运行顺序。但由于里面只包含运行结果的数据，需要辅助汇编文件一同产看，才能更好的发挥其作用

3 实验分工

张钰堃负责完成 exp12，刘景平负责完成 exp13。