实验 8 报告

学号：2017K8009922027 2017K8009929011
姓名：张磊、郭豪
箱子号：39

一、实验任务（10%）

任务：在实验 7 的基础上，增加 MTC0，MFC0，ERET，SYSCALL 指令，即增加 SYSCALL 例外支持，增加 CP0 寄存器 STATUS，CAUSE，EPC，运行功能测试通过，运行 func_lab8，要求通过仿真和上板验证。

二、实验设计（40%）

（一）总体设计思路

1. 对于 CP0 寄存器的设计，我选择把 CP0 寄存器封装成一个单独的模块，在写回级调用，这样代码看起开比较方便，也方便对 CP0 寄存器的管理；
2. SYSCALL 会触发例外（在译码级），此时给译码级附加一个产生例外的信号，留到写回级进行处理；
3. MFC0，MTC0，这两条指令可以复用之前的寄存器读写的数据通路；
4. ERET 指令在达到写回级时，需要将取指级的 NEXT_PC 换成 CP0_EPC 中的值，并且需要刷新流水线，我采取的操作是将各级的 fs_to 信号置为 0，这样可以确保之后的流水线暂时处于无效状态，直到取出 CP0_EPC 处的指令；
5. 如果出现与 MFC0 和 MTC0 有关的寄存器读相关，则将该流水级阻塞；
6. 写回级报例外时（ws_ex == 1'b1），也需要将各流水级刷新为无效状态，而且，如果写回级和访存级出现例外时，在执行阶段的写操作，如写 HI，LO 寄存器的操作，内存的写操作也将不能进行；

（二）总体设计图

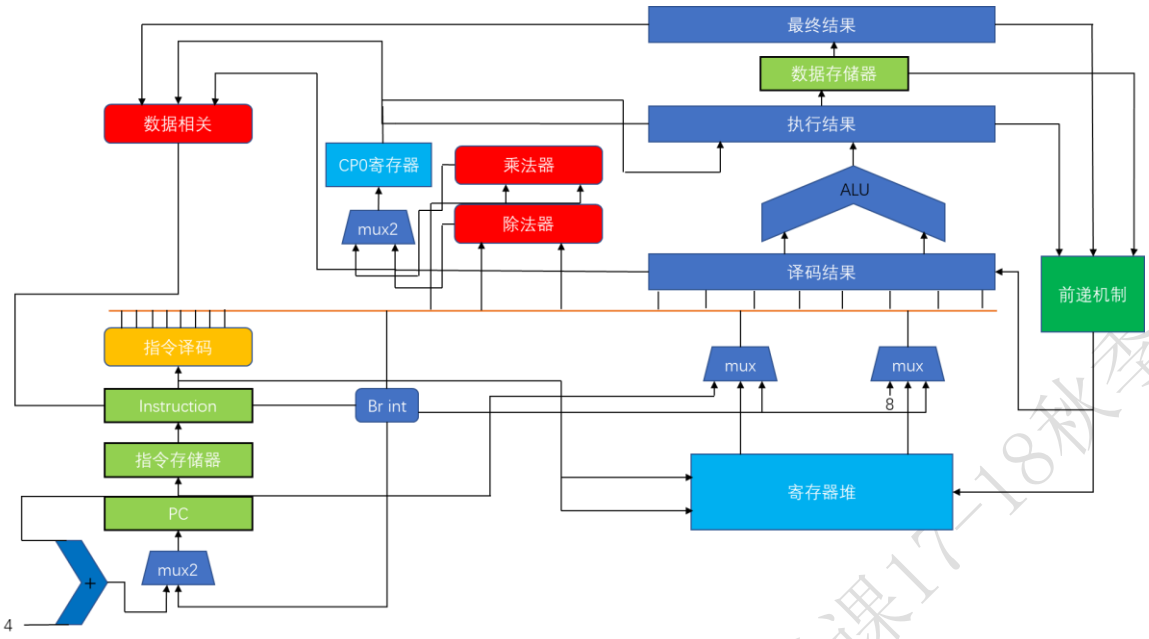


图 1 CPU 结构图

（三）重要模块 1 设计：CP0 模块；

```
module cp0(
    input         clk,
    input         rst,
    // EXCEPETION
    input         wb_ex,
    input         wb_bd,
    input  [4:0]  wb_excode,
    input  [31:0] wb_pc,
    input  [31:0] wb_badvaddr,
    input         eret_flush,
    input  [5:0]  ext_int_in,
    // READ PORT
    input  [ 7:0] cp0_addr,
    output [31:0] cp0_rdata,
    // WRITE PORT
    input         mtc0_we,
    input  [31:0] cp0_wdata,
    // OUTPUT PORT
    output [31:0] cp0_status_out,
    output [31:0] cp0_cause_out,
    output [31:0] cp0_epc_out,
    output [31:0] cp0_badvaddr_out,
    output [31:0] cp0_count_out,
    output [31:0] cp0_compare_out,
    output [31:0] cp0_config_out,
    output        count_eq_compare_out
    );
```

1、工作原理

整体上，在接口方面，仿照 regfile 进行设计，但多了 rst 信号接口，同时也提供读写操作的接口，由于同一时刻对 CP0 寄存器只能要么读，要么写，所以只需保留一个地址接口即可；

对 CP0 寄存的写操作与通用寄存器不同，这次需要按照不同的域进行写操作，以 CP0_STATUS 为例：

```
// CP0_STATUS++++++++++++++++++++++++++++++
reg [8:0]cp0_status_top;
always @(posedge clk)
begin
    if(rst)
        cp0_status_top <= 9'b0;
end

reg cp0_status_bev;
always @(posedge clk)
begin
    if (rst)
        cp0_status_bev = 1'b1;
end

reg [5:0]cp0_status_mid;
always @(posedge clk)
begin
    if(rst)
        cp0_status_mid <= 6'b0;
end

reg [7:0] cp0_status_im;
always @(posedge clk)
begin
    if (mtc0_we && cp0_addr == `CP0_STATUS_ADDR)
        cp0_status_im <= cp0_wdata[15:8];
end

reg [5:0] cp0_status_bot;
always @(posedge clk)
begin
    if(rst)
        cp0_status_bot <= 6'b0;
end
```

```
reg cp0_status_exl;
always @(posedge clk)
begin
    if(rst)
        cp0_status_exl <= 1'b0;
    else if(wb_ex)
        cp0_status_exl <= 1'b1;
    else if(eret_flush)
        cp0_status_exl <= 1'b0;
    else if(mtc0_we && cp0_addr == `CP0_STATUS_ADDR)
        cp0_status_exl <= cp0_wdata[1];
end

reg cp0_status_ie;
always @(posedge clk)
begin
    if(rst)
        cp0_status_ie <= 1'b0;
    else if(mtc0_we && cp0_addr == `CP0_STATUS_ADDR)
        cp0_status_ie <= cp0_wdata[0];
end
```

将 CP0_STATUS 的不同域分开进行写操作，但是在读取时，还是将整个 CP0 看做一个整体；

```
assign cp0_status = {
        cp0_status_top,     //31:23
        cp0_status_bev,     //22:22
        cp0_status_mid,     //21:16
        cp0_status_im,      //15:8
        cp0_status_bot,     //7:2
        cp0_status_exl,     //1:1
        cp0_status_ie       //0:0
};
```

2、功能描述

CP0 寄存器的设计；

（三）重要模块 2 设计：例外发生信号的传递

1、工作原理（以写回级到取指级为例）

由于在我们的设计中，所有的例外都是留到写回级统一处理，所以，当写回级报例外时，需要将之前的流水级刷掉（变成无效状态），因此在顶层模块增加从写回级到前面 4 个流水级的例外信号，

```
wb_stage wb_stage(
    .clk            (clk            ),
    .reset          (reset          ),
    //allowin
    .ws_allowin     (ws_allowin     ),
    //from ms
    .ms_to_ws_valid (ms_to_ws_valid ),
    .ms_to_ws_bus   (ms_to_ws_bus   ),
    //to rf: for write back
    .ws_to_rf_bus   (ws_to_rf_bus   ),
    //trace debug interface
    .debug_wb_pc      (debug_wb_pc      ),
    .debug_wb_rf_wen  (debug_wb_rf_wen  ),
    .debug_wb_rf_wnum (debug_wb_rf_wnum ),
    .debug_wb_rf_wdata(debug_wb_rf_wdata),
    //pipeline block
    .reg_dest_ws    (reg_dest_ws),
    .ws_value       (ws_value),
    .ws_mfc0        (ws_mfc0),
    //CP0
    .eret_flush     (eret_flush),
    .cp0_epc       (cp0_epc),
    .cp0_status    (cp0_status),
    .cp0_cause     (cp0_cause),
    .count_eq_compare  (count_eq_compare),
    .ws_ex_out  (ws_ex)
);
```

```
module if_stage(
    input                          clk            ,
    input                          reset          ,
    //allwoin
    input                          ds_allowin     ,
    //brbus
    input  [`BR_BUS_WD       -1:0] br_bus         ,
    //to ds
    output                         fs_to_ds_valid ,
    output [`FS_TO_DS_BUS_WD -1:0] fs_to_ds_bus   ,
    // inst sram interface
    output        inst_sram_en   ,
    output [ 3:0] inst_sram_wen  ,
    output [31:0] inst_sram_addr ,
    output [31:0] inst_sram_wdata,
    input  [31:0] inst_sram_rdata,

    //lab8
    input eret_flush,
    input [31:0] cp0_epc,
    input ds_ex_in,
    input es_ex_in,
    input ms_ex_in,
    input ws_ex_in
);
```

为达到，出现例外时，每一级流水都失效的效果，我们对每一级的 fs_to 信号做如下处理：当出现例外时，fs_to 信号置为 0，这样可以把之后的所有流水级多刷新为无效状态，而取指级依然可以正常取值指；

```
assign fs_allowin     = !fs_valid || fs_ready_go && ds_allowin;
assign fs_to_ds_valid =  fs_valid && fs_ready_go && !eret_flush && !ws_ex_in;
//assign fs_to_ds_valid =  fs_valid && fs_ready_go;
always @(posedge clk) begin
```

2、功能描述

写回级报例外时，刷新流水级；

（四）重要模块 4 设计：NEXT_PC 的生成

1、工作原理

原先的设计中，next_pc 的值只有发生跳转时才会出现需要改变为跳转目标的情况，否则都是 PC + 4，但是，在考虑例外的情况下，需要考虑在发生例外时，将 PC 修改为例外处理入口 0XBFC00380 的位置，并且，在例外退出，即 ERET 指令出现时，需要将 PC 切换为 CP0_EPC 中的值；

```
assign nextpc = ws_ex_in ? 32'hbfc00380:
                eret_flush ? cp0_epc:
                br_taken ? br_target:
                           seq_pc;
```

2、功能描述

根据不同的情况，改变 NEXT_PC 的值（包含例外和例外返回）。

三、实验过程（50%）

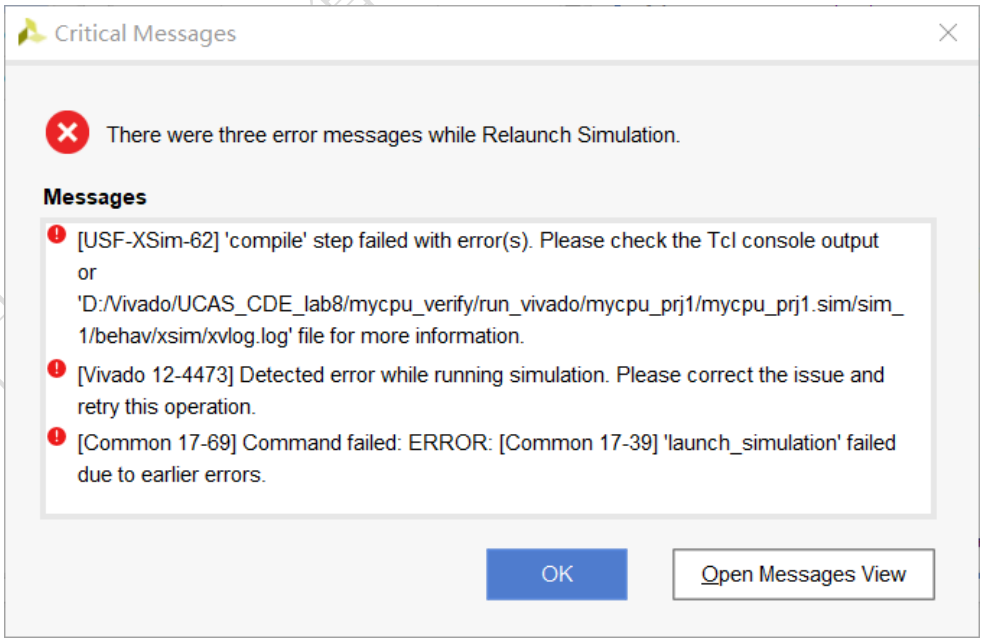
（一）实验流水账

1. 2019 年 10 月 27 日 11:00-22:00 完成 rtl 代码编写；
2. 2019 年 10 月 27 日 22:10-2019 年 10 月 28 日 03:00 完成前 2 个 bug 的修改；
3. 2019 年 10 月 28 日 9:00-2019 年 10 月 28 日 17:00 完成后两个 bug 的修改；
4. 2019 年 10 月 28 日 18:30-2019 年 10 月 28 日 19:00 完成上板验证；

（二）错误记录

1、错误 1：仿真报错

（1）错误现象



（2）分析定位过程

和我在上次实验中遇到的问题一样，一般这种情况都是由于端口补全，或者分号的缺失等问题；

（3）错误原因

在测试代码时，忘记恢复之前注释的端口

```
module if_stage(
    input                          clk           ,
    input                          reset         ,
    //allwoin
    input                          ds_allowin    ,
    //brbus
    input  [`BR_BUS_WD       -1:0] br_bus        ,
    //to ds
    output                         fs_to_ds_valid ,
    output [`FS_TO_DS_BUS_WD -1:0] fs_to_ds_bus   ,
    // inst sram interface
    output        inst_sram_en   ,
    output [ 3:0] inst_sram_wen  ,
    output [31:0] inst_sram_addr ,
    output [31:0] inst_sram_wdata,
    input  [31:0] inst_sram_rdata,

    //lab8
//    input eret_flush,
//    input [31:0] cp0_epc,
    input ds_ex_in,
    input es_ex_in,
    input ms_ex_in,
    input ws_ex_in
);
```

（4）修正效果

修正后，顺利开始仿真。

```
    //lab8
    input eret_flush,
    input [31:0] cp0_epc,
    input ds_ex_in,
    input es_ex_in,
    input ms_ex_in,
    input ws_ex_in
);
```

2、错误 2：进入死循环

（1）错误现象

```
----[ 585505 ns] Number 8'd19 Functional Test Point PASS!!!
       [ 592000 ns] Test is running, debug_wb_pc = 0xbfc2b3c4
       [ 602000 ns] Test is running, debug_wb_pc = 0xbfc2b3c4
       [ 612000 ns] Test is running, debug_wb_pc = 0xbfc2b3c4
       [ 622000 ns] Test is running, debug_wb_pc = 0xbfc2b3c4
       [ 632000 ns] Test is running, debug_wb_pc = 0xbfc2b3c4
```

（2）分析定位过程

由于第一次跑这里没问题，于是去找刚才修改过的地方；

```
assign fs_ex = (nextpc[1:0] == 2'b0) ? 1'b0 : 1'b1;
assign fs_ex = addr_error | ds_ex_in | es_ex_in | ms_ex_in | ws_ex_in;
```

图 11 错误一波形图

（3）错误原因

由于修改后没有把之前的代码注释掉，导致死循环；

（4）修正效果

修正后，顺利通过测试点。

3、错误 3： 阻塞导致取指失败

（1）错误现象

```
[1662847 ns] Error!!!
    reference: PC = 0xbfc00380, wb_rf_wnum = 0x1a, wb_rf_wdata = 0x00004000
    mycpu    : PC = 0xbfc98974, wb_rf_wnum = 0x09, wb_rf_wdata = 0x45000000
------------------------------------------------------------
```

（2）分析定位过程

由于在写回阶段的 PC 值错误，所以向前查找对应 PC，0XBFC00380 是例外入口，但是取值阶取到的却不是这个值，但写回级确实是 SYSCALL 指令，因此，一定是取指级出了问题。

（3）错误原因

最后，发现是由于 ds 即被阻塞，导致取指级没有把指令取出；

```
assign ds_ready_go = !(es_lw_valid && mfc0_stop | (es_lw_valid && ( (rs == reg_dest_es || rs == reg_dest_ms || rs == reg
                      rt == reg_dest_es || rt == reg_dest_ms ||rt == reg_dest_ws) )));
//assign ds_ready_go = 1'b1;
```

上图中的 es_lw_valid 信号出错；

（4）修正效果

修正后顺利向前进行。

4、错误 4：阻塞导致取指失败

（1）错误现象

这个问题的现象和上一个问题一模一样：

```
[1665597 ns] Error!!!
    reference: PC = 0xbfc00380, wb_rf_wnum = 0x1a, wb_rf_wdata = 0x00000036
    mycpu    : PC = 0xbfc98974, wb_rf_wnum = 0x09, wb_rf_wdata = 0x45000000
------------------------------------------------------------
```

（2）分析定位过程

由于这个问题的错误现象和上一个一模一样，所以肯定一定也是由于阻塞，导致取指级取指失败，所以重复上个问题的步骤查找，在执行级发现问题；

（3）错误原因

发生异常时，未取消流水级有效信号；

```
assign es_ready_go = es_inst_div ? div_done :
                     es_inst_divu ? divu_done :
                                    1'b1;
```

（4）修正效果

修正后，顺利通过所有测试。

```
assign es_ready_go = es_inst_div & !eret_flush & !ws_ex_in ? div_done :
                     es_inst_divu & !eret_flush & !ws_ex_in ? divu_done :
                                    1'b1;
```

四、实验总结（可选）

1. 这次实验思路比较简单，但是实现起来很复杂，而且对于例外有很多种不同的处理办法，在实现时比较容易搞糊涂；

2. 一定要先从整体上把握好了实验再开始写，否则会毫无头绪，写着写着就糊涂了；

3. 一定要早点开始，不然就没法好好睡觉了；