实验 8 报告

学号: 2017K8009929008、2017K8009929013
姓名:
箱子号: 72

一、实验任务

在五级流水 CPU 中添加例外处理逻辑，实现 cp0 寄存器，实现 syscall 例外支持。

二、实验设计

（一）总体设计思路

1. cp0 寄存器实现：

* 结构：不同功能位用不同的 reg 变量管理，各部分连接后的寄存器整体用 wire 变量管理

* 读：在 wb 阶段取得数据，传回 id 阶段完成

* 写：不同功能位的写控制分开，在 wb 阶段完成

2. 例外流程控制

* 例外信息传递：id 阶段识别例外指令，拉高例外信号，解析例外号等信息，放入 bus 往后传

* 阻塞信号：wb 阶段的例外信号传回 id，exe 和 mem，exe 阶段还需要来自 mem 阶段的例外信号（用于精确例外）

* 入口跳转：if 在 wb 阶段传来的例外信号为高时选择例外入口地址作为 nextpc

3. 精确例外

* 保证例外所在指令之前的指令都完成：id 阶段识别出的例外指令不阻塞后面阶段的指令执行

* 保证例外所在指令之后的指令未执行：

exe 阶段内存、HI/LO 寄存器的写和除法模块的调用：在后面阶段传回例外信号拉高时屏蔽

后面指令的阻塞：在 wb 阶段传来的例外信号为高时拉低 id，exe，mem 和 wb 阶段的 valid 信号

4. 结构框图：

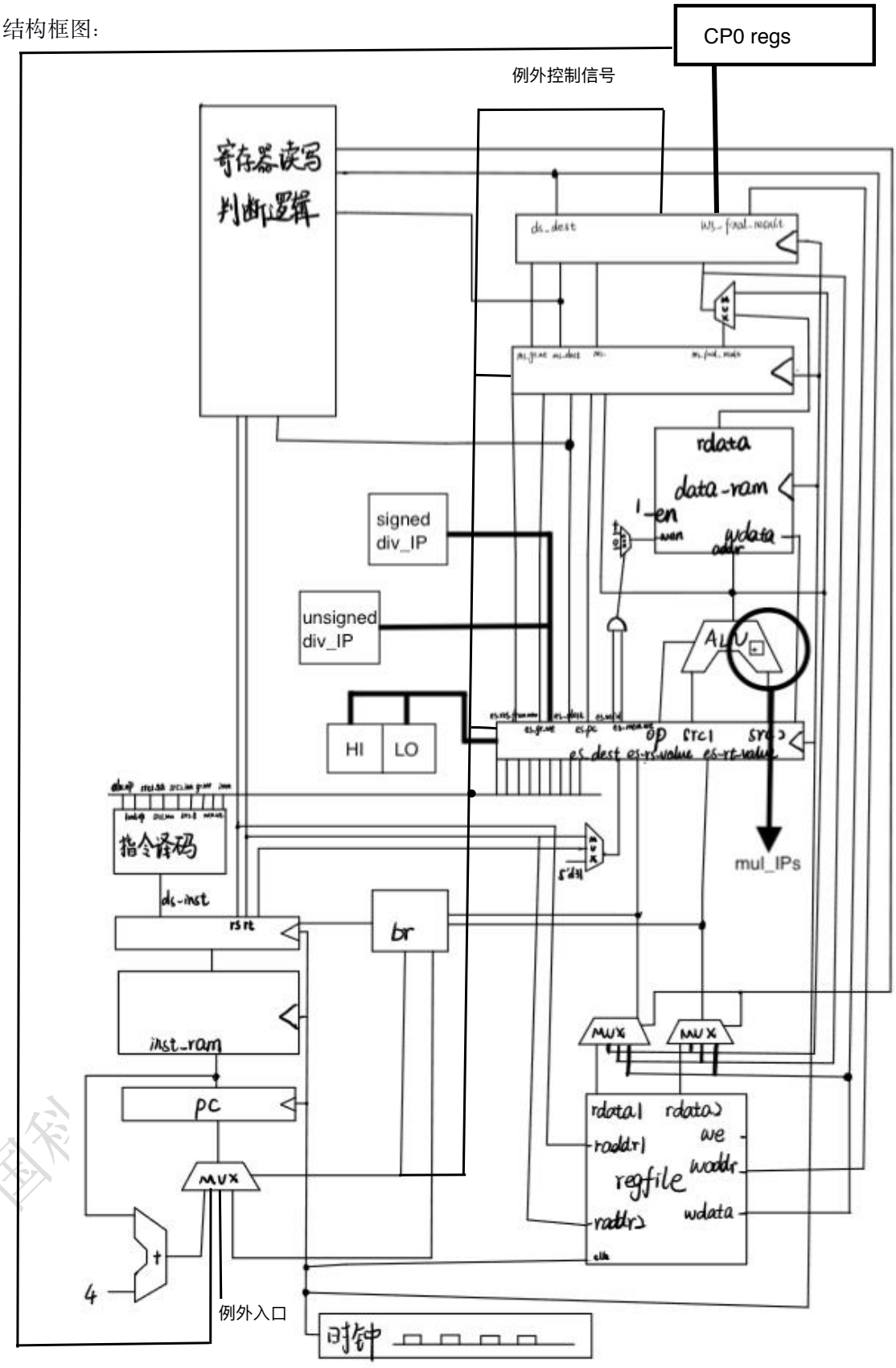

图 1-结构框图

（二）重要模块设计：ID 阶段：bus 中新增例外信息

1、代码

```
assign ds_to_es_bus = {
        inst_eret,    //175
        excode,       //174:170
        ex_op,        //169
        bd_op,        //168
        cp0_addr,     //167:160
        mtc0_op,    //159
        mfc0_op,    //158
        ...

assign    ex_op    = inst_syscall;
assign    excode   = inst_syscall? 5'h08 : 5'h00;
assign    cp0_addr = {rd, sel};
assign    mfc0_op   = inst_mfc0;
assign    mtc0_op = inst_mtc0;
assign    bd_op     = inst_beq |inst_bne |inst_bgez
                      |inst_bgtz|inst_blez|inst_bltz
                      |inst_bltzal|inst_bgezal;
```

图 2：示例代码：例外信息的生成

2、功能描述

* 例外指令判断与例外信息的生成与传递

（三）重要模块设计：精确例外：例外所在指令之后的指令处理

1、代码

```
assign write_reg     = (ws_to_es_ex==0 && ms_to_es_ex==0);

...  //HI/LO
else if(write_reg==1&&...)
begin
  HI <= ...;
  LO <= ...;
end
...

...  //除法IP调用信号
else if(write_reg==1&&...)
begin
  s_divisor_tvalid_reg <= 1'b1;
  s_dividend_tvalid_reg <= 1'b1;
end
...

...  //内存
assign data_sram_wen     = {4{write_reg&&...}} &...
```

图 3：示例代码：精确例外之 exe 阶段

```
assign ds_flush  = ws_to_rf_bus [43];

  ...
  else if (ds_flush)
    begin
      ds_valid <= 1'b0;
    end
  ...

assign es_flush  = ws_to_es_flush;

  ...
  else if (es_flush)
    begin
      es_valid <= 1'b0;
    end
  ...
```

图 4：示例代码：精确例外之阻塞

1、**功能描述**

* 在 wb 阶段传来的例外信号为高时拉低 id，exe，mem 和 wb 阶段的 valid 信号，阻塞还未进入的指令

* write_reg 信号在 mem 或 wb 阶段传来的例外信号为高时拉低，屏蔽 HI/LO、除法 IP 和内存的写

（四）重要模块设计：CP0 寄存器实现

2、代码

```
//status
wire [31:0] c0_status;
wire        c0_status_bev;
reg  [7:0]  c0_status_im;
reg         c0_status_exl;
reg         c0_status_ie;
assign c0_status = {9'b0          ,  //31:23
                    c0_status_bev,  //22:22
                    6'b0          ,  //21:16
                    c0_status_im ,  //15:8
                    6'b0          ,  //7:2
                    c0_status_exl,  //1:1
                    c0_status_ie    //0:0
                    };
//cause
wire [31:0] c0_cause;
reg         c0_cause_bd;
reg         c0_cause_ti;
reg  [7:0]  c0_cause_ip;
reg  [4:0]  c0_cause_excode;
assign c0_cause = {c0_cause_bd    ,//31:31
                   c0_cause_ti    ,//30:30
                   14'b0          ,//29:16
                   c0_cause_ip    ,//15:8
                   1'b0           ,//7:7
                   c0_cause_excode,//6:2
                   2'b0            //1:0
                   };
//EPC
reg  [31:0] c0_epc;
```

图 5：示例代码：3 个 cp0 寄存器

```
//status
assign c0_status_bev = 1'b1;
always @(posedge clk) begin
    if(mtc0_we && ws_cp0_addr==`cp0_status_addr)begin
        c0_status_im <= ws_final_result[15:8];
    end
end

always @(posedge clk)begin
    if(reset)begin
        c0_status_exl <= 1'b0;
    end
    else if(wb_ex)begin
        c0_status_exl <= 1'b1;
    end
    else if(eret_flush)begin
        c0_status_exl <= 1'b0;
    end
    else if(mtc0_we && ws_cp0_addr == `cp0_status_addr)begin
        c0_status_exl <= ws_final_result[1];
    end
end

always @(posedge clk) begin
    if(reset)begin
        c0_status_ie <= 1'b0;
    end
    else if(mtc0_we && ws_cp0_addr == `cp0_status_addr)begin
        c0_status_ie <= ws_final_result[0];
    end
end
```

图 6：示例代码：status 寄存器更新

```
//epc
always @(posedge clk)begin
    if(wb_ex && c0_status_exl==0)begin
        c0_epc <= ws_bd_op? (ws_pc - 3'h4) : ws_pc;
    end
    else if(mtc0_we && ws_cp0_addr == `cp0_epc_addr)begin
        c0_epc <= ws_final_result[31:0];
    end
end
```

图 7：示例代码：epc 寄存器更新

3、功能描述

* cp0 和 status 寄存器按功能分块定义与更新（更新逻辑类似，上面只列出了 status 的更新逻辑

* epc 整体管理，更新时若遇到延迟槽指令，应该设为当前 pc-4

三、实验过程

（一）实验流水账

周五下午：打开任务书，了解实验内容，小组交流分工

周五到周日：逻辑设计，代码实现，仿真调试，上板运行

周日晚上：小组交流代码实现和报告内容

周一上午：写报告

（二）debug 记录

1、错误 1：例外跳转时取指错误

错误现象：仿真报错：

```
[1659157 ns] Error!!!
    reference: PC = 0xbfc00380, wb_rf_wnum = 0x1a, wb_rf_wdata = 0x00004000
    mycpu    : PC = 0xbfc00384, wb_rf_wnum = 0x1b, wb_rf_wdata = 0x00000000
```

图 8：错误现象 1

错误查找过程：

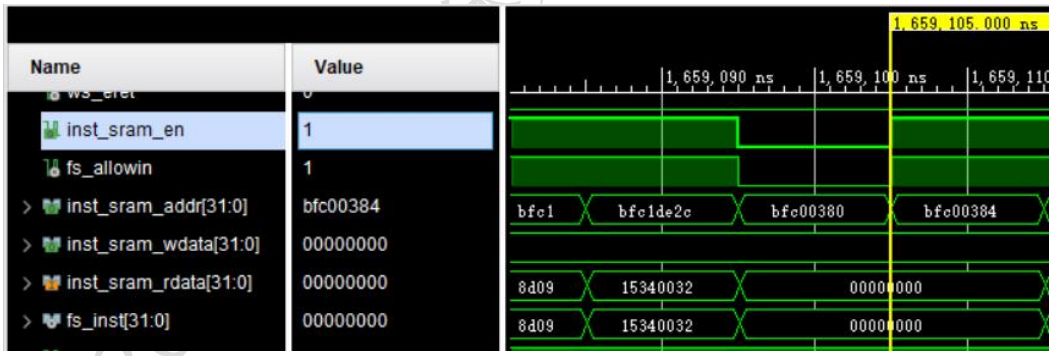

图 9：错误波形 1

发现 fs_pc 为例外入口的时候，指令 ram 的使能为 0，没有取出正确的指令；

修改方法：如下图所示，发生例外时可以从例外入口地址处正确取指：

```
assign inst_sram_en    = (to_fs_valid && fs_allowin) || fs_ex_op;
```

图 10：修正方法 1

2、错误 2：例外跳转时取指错误

错误现象：仿真报错：

```
[1661377 ns] Error!!!
    reference: PC = 0xbfc00380, wb_rf_wnum = 0x1a, wb_rf_wdata = 0x00000022
    mycpu    : PC = 0xbfc00380, wb_rf_wnum = 0x1a, wb_rf_wdata = 0x00000000
```

图 11：错误现象 2

错误查找过程:

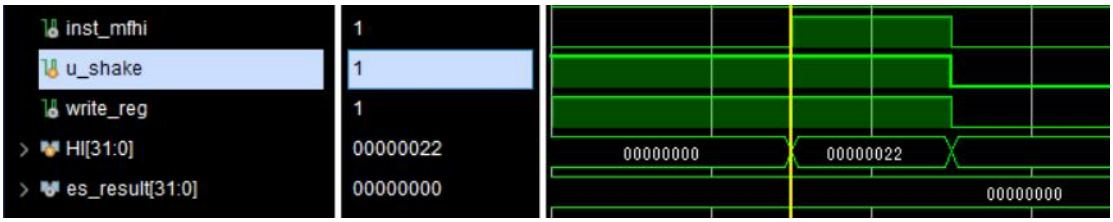

图 12：错误波形 2-1

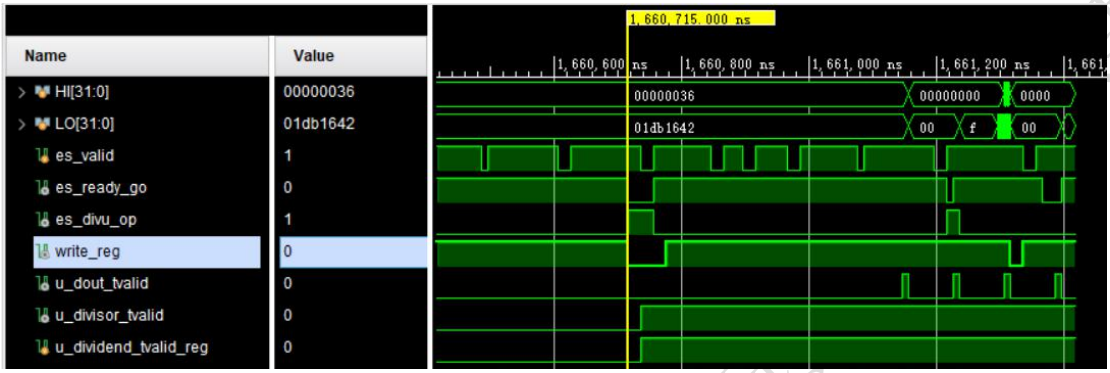

图 13：错误波形 2-2

如图错误波形 2-1 所示，发生例外的上一条指令为 mfhi，并且当 mfhi 位于译码阶段时，HI 寄存器内为正确结果，但是当 mfhi 到了 EXE 阶段，HI 寄存器改变；

再继续寻找有关写 HI 寄存器的操作，发现除法连续给出了多个计算结果，发现除法 IP 连续握手；

错误原因：如果 MEM，WB 阶段发生例外，EXE 阶段只考虑了不能写 HI，LO 和数据 ram，但是没有考虑到此时不能向除法 IP 中喂数据，除法握手以后会在后面某个时间连续输出；

修改方法：如下图所示，EXE 阶段看到 MEM，WB 是例外的时候，不能再进行除法握手

```
else if(es_div_op==1 && s_divisor_tready==0 && shake==0 && write_reg==1)begin
    s_divisor_tvalid_reg <= 1'b1;
    s_dividend_tvalid_reg <= 1'b1;
end
```

图 14：修正方法 2-1

```
else if(es_divu_op==1 && u_divisor_tready==0 && u_shake==0 && write_reg==1)begin
    u_divisor_tvalid_reg <= 1'b1;
    u_dividend_tvalid_reg <= 1'b1;
end
```

图 15：修正方法 2-2