# 实验 9 报告

学号：2017K80099290013　2017K8009929008
姓名：
箱子号：72

## 一、实验任务（10%）

1、 在流水线中添加例外和中断处理的方法；

2、 进一步加深对流水线结构的理解；

## 二、实验设计（40%）

### （一）总体设计思路

1、 增加 cp0 寄存器：

* 结构：在上次实验的基础上增加 c0_badvaddr，c0_count，c0_compare 等特权寄存器的实现

* 读写：根据特权寄存器的功能不同，在 WB 阶段完成相应的读，写操作；

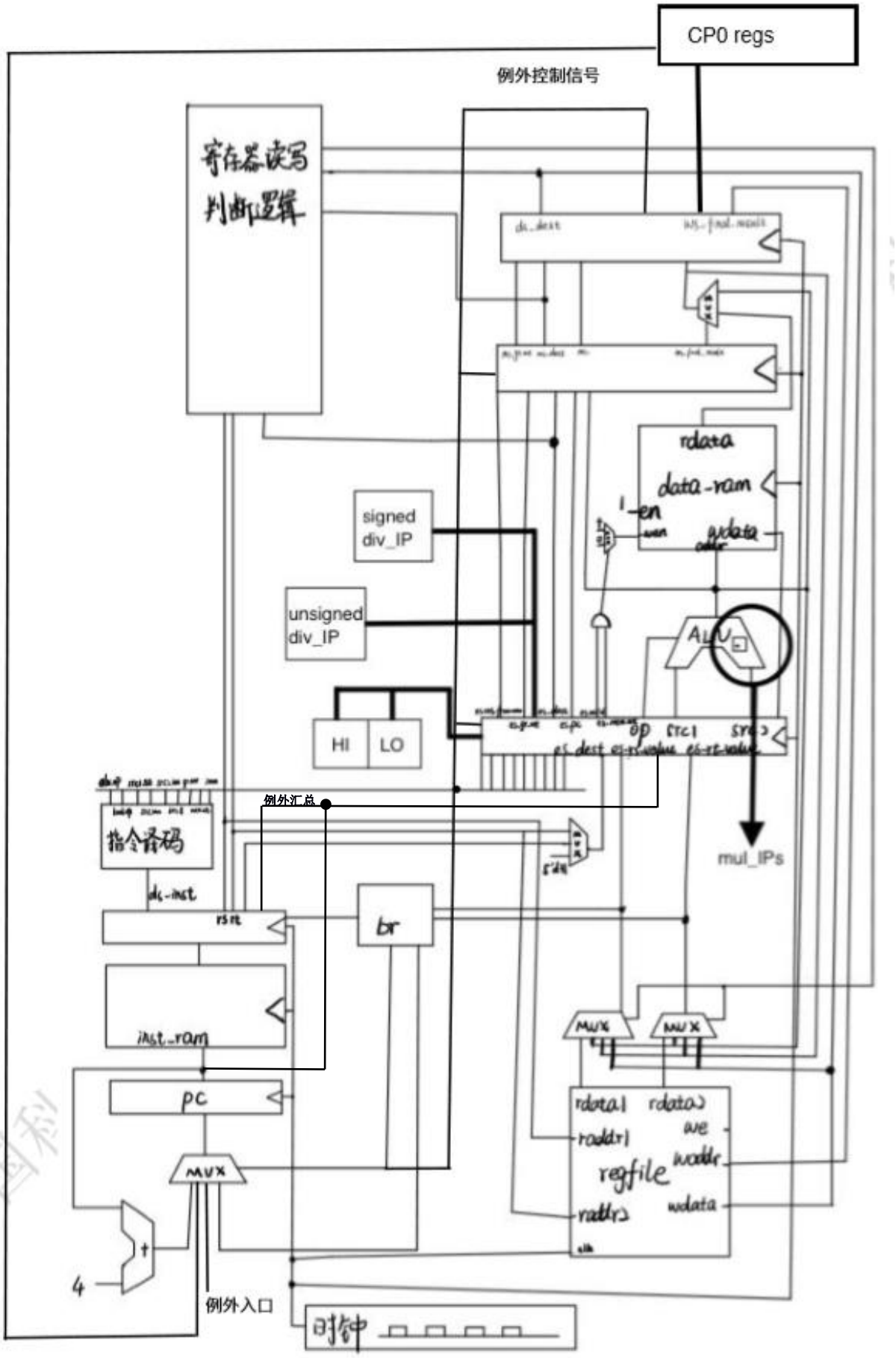
2、 中断，例外的实现：

* 例外信息传递：例外发生会发生在流水线的不同阶段，因此，从 IF 阶段开始，在向后的流水线中不断收集例外信息，并且根据优先级排序，一直到 WB 的汇聚点处，再根据例外种类开始进行例外处理；

* 时钟中断：当 c0_count 和 c0_compare 的值相等时触发时钟中断，c0_cause 寄存器的 ip 位发生改变记录时钟中断，在 ID 阶段接收时钟中断，然后流水至 WB 的汇聚点处进行处理；

五级流水的结构设计图如下：

国科大B6209计算机体系结构研讨课17-18秋季


CP0 regs
例外控制信号
寄存器读写
判断逻辑
rdata
data_ram
signed
div_IP
unsigned
div_IP
ALU
HI
LO
例外汇总
指令译码
mul_IPs
br
inst_ram
pc
MUX
MUX
MUX
regfile
4
例外入口
时钟

图一：流水线结构图

## （二）重要模块设计：IF 阶段：判断地址错例外

### 1、部分代码

```
wire fetch_error;
assign fetch_error = fs_pc[0]|fs_pc[1];


assign fs_to_ds_bus = {fs_pc,      //badvaddr
```

图二：IF 地址错例外判断逻辑

### 2、功能描述：

* 根据取出的 fs_pc 后两位是否同时为零，判断是否在 IF 阶段发生地址错例外；
* 将例外信息沿流水线传向 ID 阶段；

## （三）重要模块设计：ID 阶段：保留指令例外＋判断 break 例外＋时钟中断

### 1、部分代码：

```
assign ex_ri = ~(
/** cpu arithmetic **/  //14
    inst_add     |inst_addi    |inst_addiu   |inst_addu
   |inst_div     |inst_divu    |inst_mult    |inst_multu
   |inst_slt     |inst_slti    |inst_sltiu   |inst_sltu
   |inst_sub     |inst_subu
/** cpu branch and jump **/   //12
   |inst_beq     |inst_bgez    |inst_bgezal  |inst_bgtz    |inst_blez    |inst_bltz
   |inst_bltzal  |inst_bne     |inst_j       |inst_jal     |inst_jalr    |inst_jr
/** cpu instruction control **/
//   |inst_nop     |inst_ssnop
/** cpu load, store and memory control **/  //12
   |inst_lb      |inst_lbu     |inst_lh      |inst_lhu     |inst_lw      |inst_lwl
   |inst_lwr     |inst_sb      |inst_sh      |inst_sw      |inst_swl     |inst_swr
/** cpu logical **/                         //8
   |inst_and     |inst_andi    |inst_lui     |inst_nor     |inst_or      |inst_ori
   |inst_xor     |inst_xori
/** cpu insert/extract **/      //4
/** cpu move **/
   |inst_mfhi    |inst_mflo
   |inst_mthi    |inst_mtlo
/** cpu shift **/           //6
   |inst_sll     |inst_sllv    |inst_sra     |inst_srav    |inst_srl     |inst_srlv
/** cpu trap **/              //2
   |inst_break   |inst_syscall

/** privileged instructions **/      //3
   |inst_eret    |inst_mfc0    |inst_mtc0
/** ejtag **/
);
```

图三：保留指令例外判断

```
assign    id_excode   = has_int?5'h00
                      :(if_fetch_error?5'h04
                      :(ex_ri?5'h0a
                      :(inst_break? 5'h09
                      :(inst_syscall? 5'h08:5'h00))));
```

图四：ID 优先级排序

### 2、功能描述：

* 在 ID 阶段增加译码逻辑，判断是否发生 break 例外，保留指令例外；

* 从 cp0 寄存器处得到是否发生时钟中断的信号；
* 将现阶段的例外和中断进行优先级排序，并继续沿流水线传向 EXE 阶段

## （四）重要模块设计：EXE 阶段：判断地址错例外＋整型溢出例外

### 1、部分代码：

```
assign data_addr_error = ((lw_op|sw_op)&(data_sram_addr[1]|data_sram_addr[0]))
                       |((lh_op|lhu_op|sh_op)&(data_sram_addr[0]));
```

图五：EXE 地址错例外判断逻辑

```
assign overflow=((es_add|es_addi)&
                 ((~es_alu_src1[31]&~es_alu_src2[31]&es_alu_result[31])
                 |(es_alu_src1[31]&es_alu_src2[31]&~es_alu_result[31])))
               |(es_sub&
                 ((~es_alu_src1[31]&es_alu_src2[31]&es_alu_result[31])
                 |(es_alu_src1[31]&~es_alu_src2[31]&~es_alu_result[31])));
```

图六：整型溢出例外判断逻辑

```
assign es_excode = id_ex_op?id_excode
                 :(overflow?5'h0c
                 :(data_addr_error?(es_load_op?5'h04:5'h05)
                 :id_excode));
```

图七：EXE 优先级排序

### 2、功能描述：

* 判断访存指令是否存在地址错例外，若有，则不能向 Hi、LO、数据 ram，除法 IP 中写入数据；
* 如果是有符号计算指令，根据操作数和结果的符号位进行判断是否发生整型溢出例外；
* 结合之前的例外处理优先级，给出 EXE 阶段的例外处理优先级；

## （五）重要模块设计：WB 阶段：增加 cp0 寄存器及其读写逻辑

### 1、部分代码：

```
//badvaddr
always @(posedge clk) begin
    if (wb_ex && (ws_excode==5'h04 | ws_excode ==5'h05) && c0_status_exl==0)
        c0_badvaddr <= badvaddr;
end
```

图八：c0_badvaddr 寄存器

```
//count
always @(posedge clk) begin
    if (reset)
        tick <= 1'b0;
    else
        tick <= ~tick;  //两拍拉高一次
    if (mtc0_we && ws_cp0_addr==`CR_COUNT)
    begin
        c0_count <= ws_final_result;
        tick<=1'b0;
    end
    else if (tick)
        c0_count <= c0_count + 1'b1;  //两拍自增
end
```

图九：c0_count 寄存器

```
//compare
always @(posedge clk) begin
    if (mtc0_we && ws_cp0_addr==`CR_COMPARE)
        c0_compare <= ws_final_result;
end
```

图十：c0_compare 寄存器

```
//ti
always @(posedge clk)begin
    if(reset)
        c0_cause_ti <= 1'b0;
    else if (mtc0_we && ws_cp0_addr==`CR_COMPARE)
        c0_cause_ti <= 1'b0;
    else if (c0_count==c0_compare)
        c0_cause_ti <= 1'b1;
end
```

图十一：ti 寄存器

2、功能描述：

* 完成不同特权指令的读写功能；
* 将时钟中断的信号传向 id 阶段进行处理；

# 三、实验过程（50%）

## （一）实验流水账

周四下午：阅读任务书，更新环境，整理实验思路；

周五到周六：逻辑设计，代码实现，仿真调试，上板运行；

周六晚上：小组交流代码实现和报告内容；

周日下午：写报告；

## （二）错误记录

### 1、错误 1：取指阶段判断地址错例外时用了 nextpc

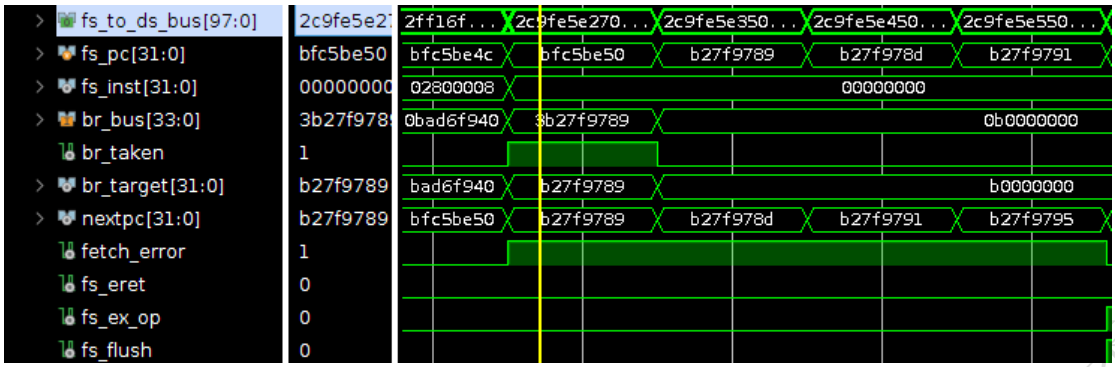
**错误现象**：仿真报错：

```
------------------------------------------------------------
[1725457 ns] Error!!!
    reference: PC = 0xbfc00544, wb_rf_wnum = 0x1a, wb_rf_wdata = 0xb27f9789
    mycpu    : PC = 0xbfc00544, wb_rf_wnum = 0x1a, wb_rf_wdata = 0xbfc5be4c
```

图十二：错误现象 1

**错误查找过程**：

国产龙芯2009H计算机体系结构研讨课1(-18秋季




图十三：错误波形 1

根据提示查看错误指令的波形，发现取指例外对应的 pc 保存成了上一条指令地址，而正确的应该是错误的地址，意识到应该用 fs_pc 判断是否有地址错误

**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：

```
wire fetch_error;
assign fetch_error = fs_pc[0]|fs_pc[1];
```

图十四：修正方法 1

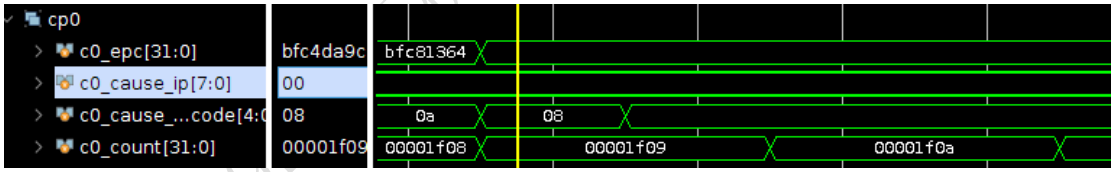
## 2、错误 2：cp0_excode 的更新逻辑中未考虑 cp0_status_exl

**错误现象**：仿真报错：excode 的值不符

```
------------------------------------------------------------
[1923657 ns] Error!!!
    reference: PC = 0xbfc00400, wb_rf_wnum = 0x1a, wb_rf_wdata = 0x80000020
    mycpu    : PC = 0xbfc00400, wb_rf_wnum = 0x1a, wb_rf_wdata = 0x80000010
------------------------------------------------------------
```

图十五：错误现象 2

**错误查找过程**：




图十六：错误波形 2

仿真报错从错误指令往前翻，查看上一条 syscall 时 excode 的记录情况，发现这个 excpde 在 syscall 的 excode 值更新一拍后就被修改

**修改方法**：如下图所示，将 excode 的更新判断逻辑中加入 cp0_status_exl

```
else if(wb_ex && c0_status_exl==0 )begin
    c0_cause_excode <= ws_excode;
end
```

图十七：修正方法 2

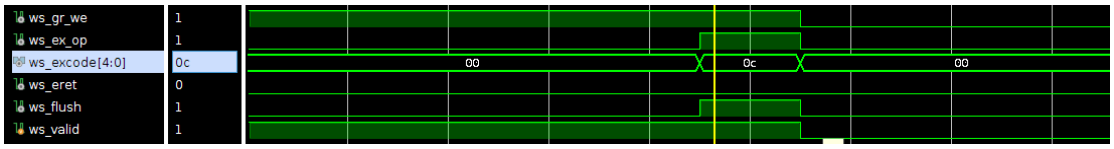
## 3、错误 3：wb 阶段发现中断时未完全拉低 regfile 的写使能信号

**错误现象**：仿真报错：

```
[1669557 ns] Error!!!
    reference: PC = 0xbfc00380, wb_rf_wnum = 0x1a, wb_rf_wdata = 0x0001001a
    mycpu    : PC = 0xbfc3ba4c, wb_rf_wnum = 0x02, wb_rf_wdata = 0x64c76d7c
------------------------------------------------------------
```

图十八：错误现象 3

**错误查找过程：**




图十九：错误波形 3

仿真报错处的指令因为数据错误导致跳转错误，往前查看发现是某次溢出异常时修改了 regfile，从波形图可以看出 ex_op 拉高是 gr_we 是 1，导致溢出错误结果写入了寄存器

**修改方法：**如下图所示，：ex_op 拉低 gr_we

```
assign ws_gr_we     = ms_gr_we&&ws_valid&&~ws_ex_op;
```

图二十：修正方法 3

国科大B6209H计算机体系结构研讨课17-18秋季