实验 4-2 报告

组员姓名 吴双、钟赟
学号 2016K8009937003、2016K8009915009

一、实验任务（10%）

(1) CPU 增加 BREAK 指令，也就是增加 break 例外支持；

(2) 增加地址错、整数溢出、保留指令例外支持；

(3) CPU 增加 CP0 寄存 BADVADDR/COUNT/COMPARE；

(4) CPU 增加时钟中断支持，时钟中断要求固定绑定在硬件中断 5 号上，也就是 CAUSE 对应的 IP7 上；

(5) CPU 增加 6 个硬件中断支持，编号为 0~5，对应 CAUSE 的 IP7~IP2；

(6) CPU 增加 2 个软件中断支持，对应 CAUSE 的 IP1~IP0；

(7) 运行 lab4.2 功能测试通过，总共有 94 个功能测试点。

二、实验设计（30%）

本次实验在实验 4-1 的基础上，增加了 3 个 cp0 寄存器 BADVADDR/COUNT/COMPARE，增加了地址错、整数溢出、保留指令、断点指令、中断例外的支持。

地址错例外可能在 3 种情况下发生：取指级取指令、访存级向内存内存数据、访存级读取内存数据，因此需要在取指级和访存级都进行地址错例外的判断。执行级的 ALU 会给出用于判断整数溢出例外的信号 Overflow。保留指令例外在译码级译码时进行判断。出现中断时，将中断标记在访存级。主要设计见以下模块：

3、cp0_regs 模块设计

CP0_regs 包括协寄存器，以及中断和例外的判断逻辑。CP0_regs 模块的端口如下：

表 1 cp0_regs 模块端口

名称	方向	作用
clk	input	时钟信号
resetn	input	复位信号
inst	input	用于判断 mfc0/mtc0/break/syscall/eret
pc	input	发生例外时存入 EPC，取指级发生地址错例外时存入 BADVADDR
data_sram_addr	input	访存级发生地址错例外时存入 BADVADDR
mtc0_value	input	mtc0 指令存入 cp0 寄存器的数据
hard_int	input	外界输入硬件中断信号

delay_slot	input	delay_slot = 1 表示当前访存级指令为延迟槽指令
ov_cmt	input	ov_cmt = 1 表示有整型溢出例外
ade_cmt	input	表示地址错例外，ade_cmt[2:0] = {取指地址错，访存读地址错，访存写地址错}
rsv_cmt	input	rsv_cmt = 1 表示出现保留指令
mfc0_value	output	mfc0 指令读出的协寄存器数据
excep_cmt	ouput	excep_cmt = 1 表示当前处理除中断之外的例外
int_cmt	ouput	int_cmt = 1 表示当前处理中断例外
eret_cmt	ouput	eret_cmt = 1 表示当前执行 eret 指令

包括以下几个寄存器：

表 3 cp0 寄存器

Name	badvaddr	count	compare	status	cause	epc
Number	8	9	11	12	13	14

支持的指令有 mfc0, mtc0, syscall, eret, break。

支持的例外如下：

表 4 支持的例外

Name	Int	AdEL	AdES	Sys	Bp	Ri	Ov
Exccode	0x00	0x04	0x05	0x08	0x09	0x0a	0x0c

Mfc0/Mtc0 指令处理：分析 rd 对应的是哪个寄存器，拉高对应寄存器的写使能信号，将数据写入寄存器。

通常例外处理流程：

(1) 当 cp0_status.exl 位为 0 时，更新 cp0_epc：如果指令不在延迟槽中，epc = pc ；否则 epc = pc - 4。

(2) 当 cp0_status.exl 位为 1 时，更新 cp0_cause.bd：如果指令在延迟槽中，cause.bd = 1’d1。

(3) cp0_status.exl 位置为 1。

(4) cp0_cause.exccode 按照讲义上例外处理的优先级，置为对应的例外编号。

(5) if 阶段的 pc 更新为 0xbfc00380，其他阶段的各寄存器清空。

中断例外处理：处理器响应中断（int_cmt = 1）的必要条件是：

(1) Status.IE = 1，表示全局中断使能开启。

(2) Status.EXL = 0，表示没有例外正在处理。

(3) 某个中断源产生中断且该中断源未被屏蔽。

中断例外的其余操作和通常例外相同。

Eret 处理流程：

(1) cp0_status.exl 位清零。

(2) if 阶段的 pc 更新为 epc 寄存器里的地址，其他阶段的各寄存器清空。

CPU 的整体结构如下：

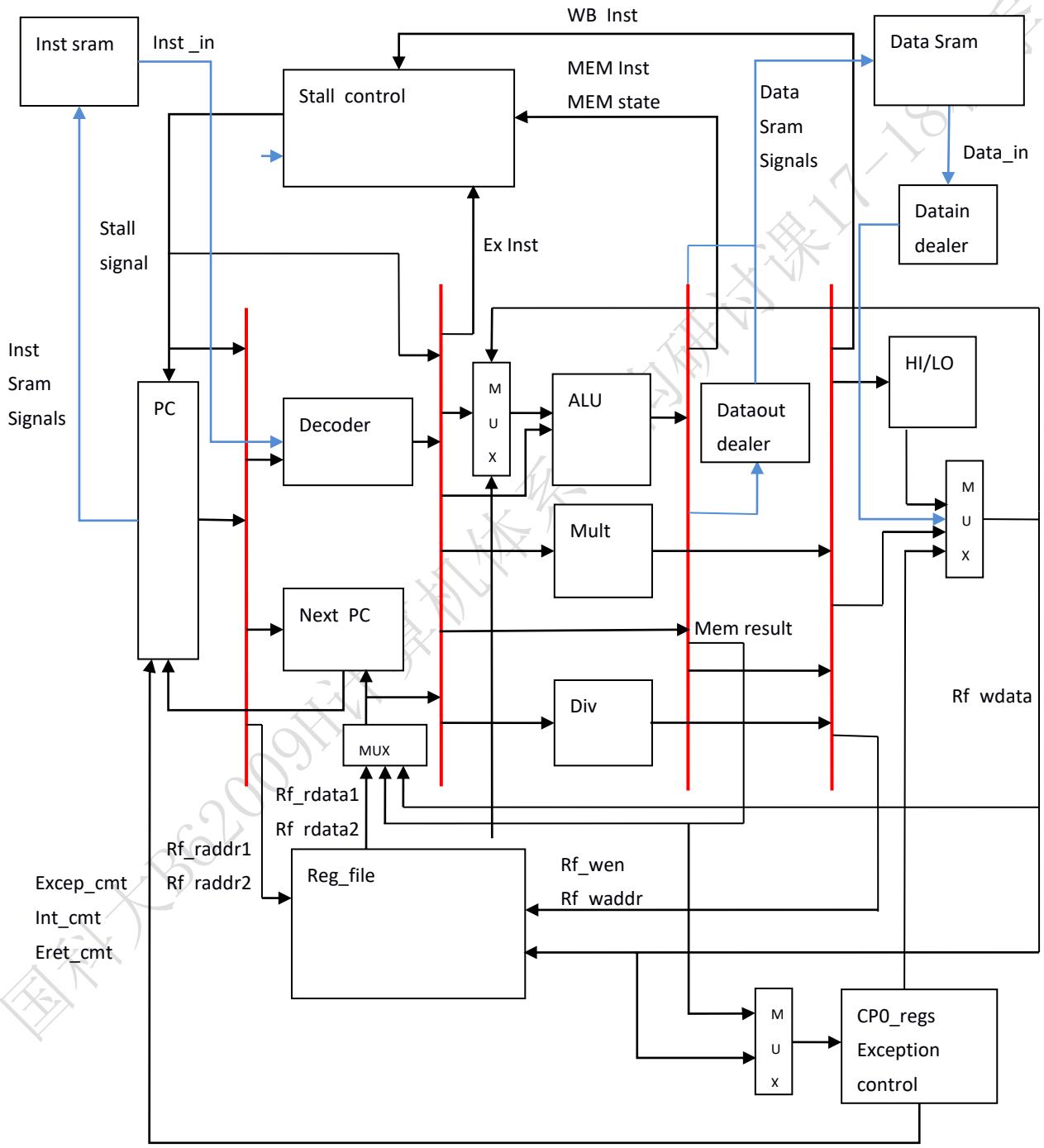

图 1 cpu 五级流水示意图

三、实验过程（60%）

（一）实验流水账

11 月 24 日 21:00 - 23:30 增加 cp0 寄存器，初步完成例外支持

11 月 26 日 00:30 – 01:30 完善中断处理

11 月 26 日 18:00 - 11 月 27 日 00:00 仿真验证 lab4.2 通过

11 月 27 日 09:30 - 12:30 上板验证通过，撰写实验报告

（二）错误记录

1、错误 1

（1）错误现象

在执行 mfc0 指令时，从 epc 内读取数据，写入寄存器堆的值比 reference 快一拍。

（2）分析定位过程

找到修改 epc 的时间点，发现这一时刻 mem 阶段的指令应该为延迟槽指令，但是用于表示延迟槽指令的 delay_slot 信号并没有置为 1，因此判断可能是延迟槽判断出错。

（3）错误原因

原先的延迟槽判断逻辑为：

a) id 阶段 PCDst 不为 0，说明有跳转或分支指令，此时 delay_slot_pc <= id_pc + 4；

b) if 阶段如果出现 if_start_pc == delay_slot_pc，则 if 阶段的 pc 对应的指令标记为延迟槽指令。

事实上，当 id 阶段为跳转或分支指令时，if 阶段已经是延迟槽指令。正确的判断是，如果是在 if 阶段标记延迟槽指令，在这个时刻就应该进行标记。但是 delay_slot_pc 在下一拍才会更新，如果 delay_slot_pc 之前存的值和 if_pc 不相同，这一时刻就不会标记延迟槽指令，从而导致之后处理例外时 epc 存入错误的值。

（4）修正效果

修改 id 阶段进行延迟槽指令的判断和标记，相关代码如下：

```
assign id_delay_slot = (!resetn) ? 1'd0 : (id_pc == delay_slot_pc);
always @ (posedge clk)  begin
    if (PCDst != 4'd0)    begin
        delay_slot_pc  <=  id_pc + 4;
    end
end
```

修正后，该处不再报错。

2、错误 2

（1）错误现象

测试软件中断时，CPU 跳转到 0xbfc00380 后陷入循环，if 阶段的 pc 永远为 0xbfc00380，流水线其他阶段寄存器一直为空值。

（2）分析定位过程

查看例外相关的信号，发现 int_cmt 信号一直为 1，判断响应中断的信号有问题。

（3）错误原因

int_cmt、exccode_int、int_req 原本的定义如下：

```
assign int_req      =  time_req || hw_req || sw_req;
assign exccode_int =  int_req && status_ie;
assign int_cmt      =  (!resetn) ? 1'd0 : exccode_int;
```

由于软件中断触发时，cause 寄存器中 ip 段对应位一直为 1，因此 sw_req 也一直保持为 1，此时 status_ie 位未经修改也一直为 1，由上述逻辑可以看出，int_cmt 也一直为 1，而当 int_cmt 为 1 时，CPU 会跳转到 0xbfc00380 并且刷新流水线。因此 CPU 会一直停留在 0xbfc00380 对应的指令上。

造成这个错误的原因是对于 3 个信号表示的意义混淆。3 个信号应该表示的意义为：

int_req：中断源触发中断请求；

exccode_int：中断未被屏蔽，CPU 响应例外，有待处理的中断例外（不一定在这个时刻处理）；

int_cmt：当前无正在处理的其他例外，该时刻处理的例外为中断例外。

（4）修正效果

修改 int_cmt 的赋值：

```
assign int_cmt = (!resetn) ? 1'd0 : (exccode_int && !status_exl);
```

该方法修改有效。

四、实验总结（可选）

本次实验