实验 4-2 报告

学号：2016K8009908007
姓名：薛峰

一、实验任务

实验目的：

(1) CPU 增加 CP0 寄存器 COUNT、COMPARE。

(2) 增加 BREAK 指令。

(3) CPU 增加地址错、整数溢出、保留指令例外以及时钟中断支持，其中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。

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

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

检验方法：

仿真和上板运行 func_lab4 通过；

二、实验设计

总体思路：

在 IF 阶段检测取值地址错例外，ID 阶段检测 syscall 和 break 指令以及保留指令例外，EX 阶段检测整型溢出例外以及读数据/写数据地址错误。并且将检测的结果传到 EX 阶段，最终在 EX 阶段提交例外。提交例外之后，需要将下一个时刻的 EX、ID 内的 valid 置 0（即清空流水线）。

具体实现：

对于 cp0_epc 寄存器：

首先，epc 寄存器是可写的，因此当 mtc0_wen_ep 为 1 时，需要将对应寄存器内的值写到 cp0_epc 中；其次，当例外提交的时候，需要将提交该例外的指令的 PC（即 PC_EX）写入 cp0_epc 寄存器中。但需要注意的是，如果该指令处于延迟槽内，需要将 PC_EX-4 存入到 cp0_epc 中。

对于 cp0_status 寄存器：

首先，因为 IM、EXL 和 IE 域是可写的，因此当 mtc0_wen_status 为 1 时，需要将对应寄存器内的值写到 cp0_status 中；另外，当例外提交的时候，如果 exl 域为 0（即此时处于正常级），则将 exl 改为 1；其次，若检测到 eret 指令，需要将 exl 域置为 0。

对于 cp0_cause 寄存器：

首先，因为 IP 域可写，因此当 mtc0_wen_cause 为 1 时，需要将对应寄存器内的值写到 cp0_cause 中；另外，

当例外提交时，需要将 ExcCode 域写对应的编码；另外，如果当前指令在延迟槽中，需要将 BD 域写 1，否则写 0；此外，当写 cp0_compare 寄存器时，需要将 TI 域置零，当 cp0_count == cp0_compare 时，需要将 TI 域置 1；其次，IP 域需要连上对应的中断。

对于 cp0_count 寄存器：

首先，因为 cp0_count 寄存器可写，因此当 mtc0_wen_count 为 1 时，需要将对应寄存器内的值写到 cp0_count 中；另外，每两个周期，count 寄存器需要加 1。

对于 cp0_compare 寄存器：

首先，因为 cp0_compare 寄存器可写，因此当 mtc0_wen_compare 为 1 时，需要将对应寄存器内的值写到 cp0_compare 中。

对于 cp0_BadVaddr 寄存器：

当取指错时，需要将 PC 写入 cp0_BadVaddr；当读数据/写数据时，需要将地址写入 cp0_BadVaddr。

三、实验过程

（一）实验流水账

时间	记录
11 月 23 日 18：10~22：30	一、阅读相关资料，了解处理各例外需要做的操作； 二、开始写 rtl 代码。
11 月 25 日 13：10~19：30 20：20~23：50	一、完成 rtl 代码； 二、进行仿真，但仿真未通过。
11 月 26 日 15：20~20：10	一、调波形，最终仿真通过； 二、上板验证通过。

（二）错误记录

1、错误 1

（1）错误现象

仿真的过程中控制台报错。

（2）分析定位过程

找到错误之处，发现在一条 syscall 指令之后有一条跳转指令，但是 syscall 之后的流水线并没清空，导致执行跳转指令，从而没能进入到例外处理程序中。

（3）错误原因

Lab4-1 中是在译码阶段提交例外，现在推迟到执行阶段，因此清空流水线处的逻辑也应该调整。

（4）修正效果

将清空流水线的逻辑多清空一级之后该处错误消失。

（5）归纳总结

该处错误是没有考虑好例外处理需要做的工作造成的。

2、错误 2

（1）错误现象

一条 mfc0 指令出错。

（2）分析定位过程

找到错误之处，发现取 badvaddr 数据出错，因此判断一定是在例外处理入口出写 badvaddr 寄存器出错。

（3）错误原因

跳入例外处理前的指令是一条跳转指令，跳转的地址没有四字节对齐，因此会触发例外，badvaddr 本应该存储要跳转到的错误 PC，但是我存了这条指令的 PC。

（4）修正效果

将 badvaddr 调正确之后该错误消失。

（5）归纳总结

没有搞清楚 badvaddr 需要存储的值。

3、错误 3

（1）错误现象

控制台报错。

（2）分析定位过程

找到错误之处，发现本应该进入到例外处理的地方，但我的代码并没有进入。在 test.s 中发现本应该进入 lhu 地址错误例外。

（3）错误原因

发现我在考虑取值地址错误的时候，没有考虑 lhu 这条指令。

（4）修正效果

在判断地址错的时候，加上 lhu 这条指令的情况发现错误消失。

（5）归纳总结

笔误。

4、错误 4

（1）错误现象

控制台报错。

（2）分析定位过程

找到错误之处，发现是一条转移指令，但是转移指令的跳转地址没有四字节对齐，因此会触发例外。

（3）错误原因

正常情况下该指令的延迟草需要执行，但是发现这条跳转指令之后就会立即清流水线。

（4）修正效果

后来询问同学之后发现跳转指令并不会触发例外，需要跳转到错误的 PC，之后根据 PC_ID 判断是否为地址错例外，这样延迟槽指令便能正确执行。这样修改过后，错误消失。

（5）归纳总结

没有正确理解地址错例外。

5、错误 5

（1）错误现象

控制台报错。

（2）分析定位过程

出错的指令是为了比较 cause 寄存器的 BD 域，该处错误发生在例外处理函数当中，先前查看波形，发现出发该例外的是在延迟槽中的指令。

（3）错误原因

该指令在延迟槽中，应该将 BD 域置为 1，但是我的代码当中并没有置 BD 域。

（4）修正效果

修改对 cause 寄存器的赋值，即当触发例外的指令在延迟槽当中时要将 BD 域置为 1。

（5）归纳总结

笔误，忘记对 BD 域进行复制。

6、错误 6

（1）错误现象

Lab4 的测试点都通过之后，开始仿真所有的功能测试点，但发现在第二个测试点的时候进入了例外处理入口。

（2）分析定位过程

查看波形发现报的例外是整型溢出例外，整型溢出例外只针对 ADD、ADDI 或 SUB 指令，但是功能测试点 2 检查的是 ADDU 指令，本不应该出现该例外。

（3）错误原因

在报整型溢出例外的时候需要检测，检测当前是不是 ADD、ADDI 或 SUB 指令。

（4）修正效果

增加对指令的检测之后，测试点 2 通过，并且最终所有测试点通过。

（5）归纳总结

笔误，忘记对指令进行检测。

四、实验总结

本次实验犯了一个非常睿智的错误。在进行仿真的时候还很奇怪为什么没有比对 trace，而是只在每个测试点的最后检测是否通过。于是前半段只能对照 132 的波形一个一个检查。后来发现不知道什么时候我将 testbench 改成了 132 的 testbench，因此没有对比 trace.....这导致了我前半段 debug 的效率非常低，所以这个实验花费了大量的时间。