实验四报告

2016K8009907007
黄熠华

一、实验任务（10%）

添加 MTC0、MFC0、ERET、SYSCALL 指令，硬件增添 CP0_STATUS、CP0_CAUSE、CP0_EPC 寄存器。运行功能测试通过。

二、实验设计（30%）

首先是设计 MTC0、MFC0、ERET、SYSCALL 指令，以上指令都需要对 CP0 寄存器进行修改或读取。如果在写回级对 CP0 寄存器进行写入，那么就要考虑复杂的数据相关问题。CP0 寄存器有 32 个，并且不同于通用寄存器，可以用 wire 的内容直接指定读写，而是需要用进行译码判断后决定读写的寄存器目标，这为数据前递更增添了工程上的复杂度。因此本次实验采用了在译码级读写 CP0 寄存器，将读取的 CP0 寄存器的值流水至后几级的方法。可以说是将 CP0 寄存器视作 32 位，来源有赖于指令种类的控制信号。如此一来，CP0 寄存器的读写在同一阶段，便不存在 CP0 寄存器相关的数据冲突。

CP0_STATUS、CP0_CAUSE 寄存器并不是每一位都对用户可写的，这点老师上课时强调过，我便按照老师上课时传授的风格实现了这两个寄存器。

需要注意的是，SYSCALL 是精确例外，需要保证 SYSCALL 前的指令都执行，之后的指令都取消。因此不同于跳转指令，SYSCALL 指令需要取消掉其延迟槽指令的执行。采取的办法是：在 ID_valid 的值允许更新时，判断当前译码指令是否是 SYSCALL 指令，如果是，那么在下个时钟上升沿将 ID_valid 的值赋值为 0，这样 SYSCALL 的下一条指令便失去了作用。同时 CP0_EPC 需要赋值为译码级的 PC，使得返回后 SYSCALL 的下一条指令会被重新执行。

三、实验过程（60%）

（一）实验流水账

11.18 20:00 开始写 Lab4 实验

11.19 03:00 调试结束，仿真通过

（二）错误记录

1、错误 1

（1）错误现象

例外发生后，PC 又跳至一处 insert_error 代码段。

（2）分析定位过程

发现 SYSCALL 后一条指令是 bne 指令，这正是跳转至 insert_error 代码段的指令。

（3）错误原因

发现忘记取消 SYSCALL 后的指令。

（4）修正效果

程序在该位置运行正确。

（5）归纳总结（可选）

2、错误 2

（1）错误现象

SYSCALL 指令写数据与 trace 不符。

（2）分析定位过程

发现与 trace 写数据相差了 4，便明白写数据用错了。

（3）错误原因

写 CP0_EPC 的数据应该为译码级的 PC 而不是当前取址级的 PC。

（4）修正效果

程序正确通过该处。

（5）归纳总结（可选）

四、实验总结（可选）

这次实验阶段相对简单，工作量合理，希望以后能经常遇到。