5.11.2　分支预测和预测错误处罚

在 3.6.6 节中通过实验证明，当分支预测逻辑不能正确预测一个分支是否要跳转的时候，条件分支可能会招致很大的预测错误处罚。既然我们已经学习到了一些关于处理器是如何工作的知识，就能理解这样的处罚是从哪里产生出来的了。

现代处理器的工作远超前于当前正在执行的指令，从内存读新指令，译码指令，以确定在什么操作数上执行什么操作。只要指令遵循的是一种简单的顺序，那么这种指令流水线化(instruction pipelining)就能很好地工作。当遇到分支的时候，处理器必须猜测分支该往哪个方向走。对于条件转移的情况，这意味着要预测是否会选择分支。对于像间接跳转(跳转到由一个跳转表条目指定的地址)或过程返回这样的指令，这意味着要预测目标地址。在这里，我们主要讨论条件分支。

在一个使用投机执行(speculative execution)的处理器中，处理器会开始执行预测的分支目标处的指令。它会避免修改任何实际的寄存器或内存位置，直到确定了实际的结果。如果预测正确，那么处理器就会“提交”投机执行的指令的结果，把它们存储到寄存器或内存。如果预测错误，处理器必须丢弃掉所有投机执行的结果，在正确的位置，重新开始取指令的过程。这样做会引起预测错误处罚，因为在产生有用的结果之前，必须重新填充指令流水线。

在 3.6.6 节中我们看到，最近的 x86 处理器(包含所有可以执行 x86-64 程序的处理器)有条件传送指令。在编译条件语句和表达式的时候，GCC 能产生使用这些指令的代码，而不是更传统的基于控制的条件转移的实现。翻译成条件传送的基本思想是计算出一个条件表达式或语句两个方向上的值，然后用条件传送选择期望的值。在 4.5.7 节中我们看到，条件传送指令可以被实现为普通指令流水线化处理的一部分。没有必要猜测条件是否满足，因此猜测错误也没有处罚。

那么一个 C 语言程序员怎么能够保证分支预测处罚不会阻碍程序的效率呢？对于参考机来说，预测错误处罚是 19 个时钟周期，赌注很高。对于这个问题没有简单的答案，但是下面的通用原则是可用的。

1. 不要过分关心可预测的分支

我们已经看到错误的分支预测的影响可能非常大，但是这并不意味着所有的程序分支都会减缓程序的执行。实际上，现代处理器中的分支预测逻辑非常善于辨别不同的分支指令的有规律的模式和长期的趋势。例如，在合并函数中结束循环的分支通常会被预测为选择分支，因此只在最后一次会导致预测错误处罚。

再来看另一个例子，当从 combine2 变化到 combine3 时，我们把函数 get_vec_element 从函数的内循环中拿了出来，考虑一下我们观察到的结果，如下所示：

函数	方法	整数		浮点数	
		+	*	+	*
combine2	移动 vec_length	7.02	9.03	9.02	11.03
combine3	直接数据访问	7.17	9.02	9.02	11.03

CPE 基本上没变，即使这个转变消除了每次迭代中用于检查向量索引是否在界限内的两个条件语句。对这个函数来说，这些检测总是确定索引是在界内的，所以是高度可预测的。

作为一种测试边界检查对性能影响的方法，考虑下面的合并代码，修改 combine4 的