C. 解释为什么这个优化保持了期望的行为，或者给出一个例子说明它产生了与使用较少优化的代码不同的结果。

使用了这最后的变换，至此，对于每个元素的计算，都只需要 1.25～5 个时钟周期。比起最开始采用优化时的 9～11 个周期，这是相当大的提高了。现在我们想看看是什么因素在制约着代码的性能，以及可以如何进一步提高。

5.7　理解现代处理器

到目前为止，我们运用的优化都不依赖于目标机器的任何特性。这些优化只是简单地降低了过程调用的开销，以及消除了一些重大的“妨碍优化的因素”，这些因素会给优化编译器造成困难。随着试图进一步提高性能，必须考虑利用处理器*微体系结构*的优化，也就是处理器用来执行指令的底层系统设计。要想充分提高性能，需要仔细分析程序，同时代码的生成也要针对目标处理器进行调整。尽管如此，我们还是能够运用一些基本的优化，在很大一类处理器上产生整体的性能提高。我们在这里公布的详细性能结果，对其他机器不一定有同样的效果，但是操作和优化的通用原则对各种各样的机器都适用。

为了理解改进性能的方法，我们需要理解现代处理器的微体系结构。由于大量的晶体管可以被集成到一块芯片上，现代微处理器采用了复杂的硬件，试图使程序性能最大化。带来的一个后果就是处理器的实际操作与通过观察机器级程序所察觉到的大相径庭。在代码级上，看上去似乎是一次执行一条指令，每条指令都包括从寄存器或内存取值，执行一个操作，并把结果存回到一个寄存器或内存位置。在实际的处理器中，是同时对多条指令求值的，这个现象称为*指令级并行*。在某些设计中，可以有 100 或更多条指令在处理中。采用一些精细的机制来确保这种并行执行的行为，正好能获得机器级程序要求的顺序语义模型的效果。现代微处理器取得的了不起的功绩之一是：它们采用复杂而奇异的微处理器结构，其中，多条指令可以并行地执行，同时又呈现出一种简单的顺序执行指令的表象。

虽然现代微处理器的详细设计超出了本书讲授的范围，对这些微处理器运行的原则有一般性的了解就足够能够理解它们如何实现指令级并行。我们会发现两种下界描述了程序的最大性能。当一系列操作必须按照严格顺序执行时，就会遇到*延迟界限*(latency bound)，因为在下一条指令开始之前，这条指令必须结束。当代码中的数据相关限制了处理器利用指令级并行的能力时，延迟界限能够限制程序性能。*吞吐量界限*(throughput bound)刻画了处理器功能单元的原始计算能力。这个界限是程序性能的终极限制。

5.7.1　整体操作

图 5-11 是现代微处理器的一个非常简单化的示意图。我们假想的处理器设计是不太严格地基于近期的 Intel 处理器的结构。这些处理器在工业界称为*超标量*(superscalar)，意思是它可以在每个时钟周期执行多个操作，而且是*乱序的*(out-of-order)，意思就是指令执行的顺序不一定要与它们在机器级程序中的顺序一致。整个设计有两个主要部分：*指令控制单元*(Instruction Control Unit，ICU)和*执行单元*(Execution Unit，EU)。前者负责从内存中读出指令序列，并根据这些指令序列生成一组针对程序数据的基本操作；而后者执行这些操作。和第 4 章中研究过的*按序*(in-order)流水线相比，乱序处理器需要更大、更复杂的硬件，但是它们能更好地达到更高的指令级并行度。