靠的方法是使用 GCC 支持的、操纵向量数据的 C 语言扩展。参见原书 546 页的网络旁注 OPT：SIMD，看看可以怎么做到这样。

3.12　小结

在本章中，我们窥视了 C 语言提供的抽象层下面的东西，以了解机器级编程。通过让编译器产生机器级程序的汇编代码表示，我们了解了编译器和它的优化能力，以及机器、数据类型和指令集。在第 5 章，我们会看到，当编写能有效映射到机器上的程序时，了解编译器的特性会有所帮助。我们还更完整地了解了程序如何将数据存储在不同的内存区域中。在第 12 章会看到许多这样的例子，应用程序员需要知道一个程序变量是在运行时栈中，是在某个动态分配的数据结构中，还是全局程序数据的一部分。理解程序如何映射到机器上，会让理解这些存储类型之间的区别容易一些。

机器级程序和它们的汇编代码表示，与 C 程序的差别很大。各种数据类型之间的差别很小。程序是以指令序列来表示的，每条指令都完成一个单独的操作。部分程序状态，如寄存器和运行时栈，对程序员来说是直接可见的。本书仅提供了低级操作来支持数据处理和程序控制。编译器必须使用多条指令来产生和操作各种数据结构，以及实现像条件、循环和过程这样的控制结构。我们讲述了 C 语言和如何编译它的许多不同方面。我们看到 C 语言中缺乏边界检查，使得许多程序容易出现缓冲区溢出。虽然最近的运行时系统提供了安全保护，而且编译器帮助使得程序更安全，但是这已经使许多系统容易受到恶意入侵者的攻击。

我们只分析了 C 到 x86-64 的映射，但是大多数内容对其他语言和机器组合来说也是类似的。例如，编译 C++ 与编译 C 就非常相似。实际上，C++ 的早期实现就只是简单地执行了从 C++ 到 C 的源到源的转换，并对结果运行 C 编译器，产生目标代码。C++ 的对象用结构来表示，类似于 C 的 `struct`。C++ 的方法是用指向实现方法的代码的指针来表示的。相比而言，Java 的实现方式完全不同。Java 的目标代码是一种特殊的二进制表示，称为 Java *字节代码*。这种代码可以看成是虚拟机的机器级程序。正如它的名字暗示的那样，这种机器并不是直接用硬件实现的，而是用软件解释器处理字节代码，模拟虚拟机的行为。另外，有一种称为*及时编译*(just-in-time compilation)的方法，动态地将字节代码序列翻译成机器指令。当代码要执行多次时(例如在循环中)，这种方法执行起来更快。用字节代码作为程序的低级表示，优点是相同的代码可以在许多不同的机器上执行，而在本章谈到的机器代码只能在 x86-64 机器上运行。

参考文献说明

Intel 和 AMD 提供了关于他们处理器的大量文档。包括从汇编语言程序员角度来看硬件的概貌[2, 50]，还包括每条指令的详细参考[3, 51]。读指令描述很复杂，因为 1)所有的文档都基于 Intel 汇编代码格式，2)由于不同的寻址和执行模式，每条指令都有多个变种，3)没有说明性示例。不过这些文档仍然是关于每条指令行为的权威参考。

组织 x86-64.org 负责定义运行在 Linux 系统上的 x86-64 代码的应用二进制接口(Applicatioin Binary Interface，ABI)[77]。这个接口描述了一些细节，包括过程链接、二进制代码文件和大量的为了让机器代码程序正确运行所需要的其他特性。

正如我们讨论过的那样，GCC 使用的 ATT 格式与 Intel 文档中使用的 Intel 格式和其他编译器(包括 Microsoft 编译器)使用的格式都很不相同。

Muchnick 的关于编译器设计的书[80]被认为是关于代码优化技术最全面的参考书。它涵盖了许多我们在此讨论过的技术，例如寄存器使用规则。

已经有很多文章是关于使用缓冲区溢出通过因特网来攻击系统的。Spafford 出版了关于 1988 年因特网蠕虫的详细分析[105]，而帮助阻止它传播的 MIT 团队的成员也出版了一些论著[35]。从那以后，大量的论文和项目提出了各种创建和阻止缓冲区溢出攻击的方法。Seacord 的书[97]提供了关于缓冲区溢出和其他一些对 C 编译器产生的代码进行攻击的丰富信息。

家庭作业

* 3.58　一个函数的原型为