第 3 章

C H A P T E R 3

程序的机器级表示

计算机执行机器代码，用字节序列编码低级的操作，包括处理数据、管理内存、读写存储设备上的数据，以及利用网络通信。编译器基于编程语言的规则、目标机器的指令集和操作系统遵循的惯例，经过一系列的阶段生成机器代码。GCC C 语言编译器以汇编代码的形式产生输出，汇编代码是机器代码的文本表示，给出程序中的每一条指令。然后 GCC 调用汇编器和链接器，根据汇编代码生成可执行的机器代码。在本章中，我们会近距离地观察机器代码，以及人类可读的表示——汇编代码。

当我们用高级语言编程的时候(例如 C 语言，Java 语言更是如此)，机器屏蔽了程序的细节，即机器级的实现。与此相反，当用汇编代码编程的时候(就像早期的计算)，程序员必须指定程序用来执行计算的低级指令。高级语言提供的抽象级别比较高，大多数时候，在这种抽象级别上工作效率会更高，也更可靠。编译器提供的类型检查能帮助我们发现许多程序错误，并能够保证按照一致的方式来引用和处理数据。通常情况下，使用现代的优化编译器产生的代码至少与一个熟练的汇编语言程序员手工编写的代码一样有效。最大的优点是，用高级语言编写的程序可以在很多不同的机器上编译和执行，而汇编代码则是与特定机器密切相关的。

那么为什么我们还要花时间学习机器代码呢？即使编译器承担了生成汇编代码的大部分工作，对于严谨的程序员来说，能够阅读和理解汇编代码仍是一项很重要的技能。以适当的命令行选项调用编译器，编译器就会产生一个以汇编代码形式表示的输出文件。通过阅读这些汇编代码，我们能够理解编译器的优化能力，并分析代码中隐含的低效率。就像我们将在第 5 章中体会到的那样，试图最大化一段关键代码性能的程序员，通常会尝试源代码的各种形式，每次编译并检查产生的汇编代码，从而了解程序将要运行的效率如何。此外，也有些时候，高级语言提供的抽象层会隐藏我们想要了解的程序的运行时行为。例如，第 12 章会讲到，用线程包写并发程序时，了解不同的线程是如何共享程序数据或保持数据私有的，以及准确知道如何在哪里访问共享数据，都是很重要的。这些信息在机器代码级是可见的。另外再举一个例子，程序遭受攻击(使得恶意软件侵扰系统)的许多方式中，都涉及程序存储运行时控制信息的方式的细节。许多攻击利用了系统程序中的漏洞重写信息，从而获得了系统的控制权。了解这些漏洞是如何出现的，以及如何防御它们，需要具备程序机器级表示的知识。程序员学习汇编代码的需求随着时间的推移也发生了变化，开始时要求程序员能直接用汇编语言编写程序，现在则要求他们能够阅读和理解编译器产生的代码。

在本章中，我们将详细学习一种特别的汇编语言，了解如何将 C 程序编译成这种形式的机器代码。阅读编译器产生的汇编代码，需要具备的技能不同于手工编写汇编代码。我们必须了解典型的编译器在将 C 程序结构变换成机器代码时所做的转换。相对于 C 代码表示的计算操作，优化编译器能够重新排列执行顺序，消除不必要的计算，用快速操作替换慢速操作，甚至将递归计算变换成迭代计算。源代码与对应的汇编代码的关系通常不太容易理解——就像要拼出的拼图与盒子上图片的设计有点不太一样。这是一种*逆向工程*(reverse engineering)——通过研究系统和逆向工作，来试图了解系统的创建过程。在这里，系统是一个机器产生的汇编语言程序，而不是由人设计的某个东西。这简化了逆向工程的任