始到结束需要长得多的时间，大约 20 个或者更多周期，但是处理器使用了非常多的聪明技巧来同时处理多达 100 条指令。在第 4 章中，我们会研究流水线(pipelining)的使用。在流水线中，将执行一条指令所需要的活动划分成不同的步骤，将处理器的硬件组织成一系列的阶段，每个阶段执行一个步骤。这些阶段可以并行地操作，用来处理不同指令的不同部分。我们会看到一个相当简单的硬件设计，它能够达到接近于一个时钟周期一条指令的执行速率。

如果处理器可以达到比一个周期一条指令更快的执行速率，就称之为*超标量*(superscalar)处理器。大多数现代处理器都支持超标量操作。第 5 章中，我们将描述超标量处理器的高级模型。应用程序员可以用这个模型来理解程序的性能。然后，他们就能写出拥有更高程度的指令级并行性的程序代码，因而也运行得更快。

3. 单指令、多数据并行

在最低层次上，许多现代处理器拥有特殊的硬件，允许一条指令产生多个可以并行执行的操作，这种方式称为*单指令、多数据*，即 SIMD 并行。例如，较新几代的 Intel 和 AMD 处理器都具有并行地对 8 对单精度浮点数(C 数据类型 `float`)做加法的指令。

提供这些 SIMD 指令多是为了提高处理影像、声音和视频数据应用的执行速度。虽然有些编译器会试图从 C 程序中自动抽取 SIMD 并行性，但是更可靠的方法是用编译器支持的特殊的向量数据类型来写程序，比如 GCC 就支持向量数据类型。作为对第 5 章中比较通用的程序优化描述的补充，我们在网络旁注 OPT:SIMD 中描述了这种编程方式。

1.9.3　计算机系统中抽象的重要性

*抽象*的使用是计算机科学中最为重要的概念之一。例如，为一组函数规定一个简单的应用程序接口(API)就是一个很好的编程习惯，程序员无须了解它内部的工作便可以使用这些代码。不同的编程语言提供不同形式和等级的抽象支持，例如 Java 类的声明和 C 语言的函数原型。

我们已经介绍了计算机系统中使用的几个抽象，如图 1-18 所示。在处理器里，*指令集架构*提供了对实际处理器硬件的抽象。使用这个抽象，机器代码程序表现得就好像运行在一个一次只执行一条指令的处理器上。底层的硬件远比抽象描述的要复杂精细，它并行地执行多条指令，但又总是与那个简单有序的模型保持一致。只要执行模型一样，不同的处理器实现也能执行同样的机器代码，而又提供不同的开销和性能。

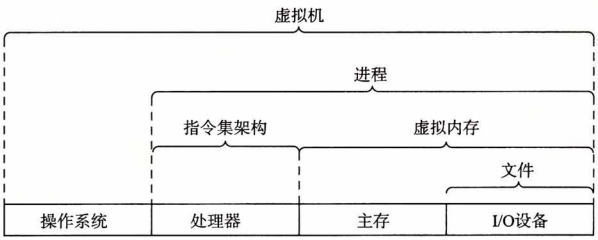

图 1-18　计算机系统提供的一些抽象。计算机系统中的一个重大主题就是提供不同层次的抽象表示，来隐藏实际实现的复杂性

在学习操作系统时，我们介绍了三个抽象：*文件*是对 I/O 设备的抽象，*虚拟内存*是对程序存储器的抽象，而*进程*是对一个正在运行的程序的抽象。我们再增加一个新的抽象：