- 理解链接时出现的错误。根据我们的经验，一些最令人困扰的程序错误往往都与链接器操作有关，尤其是当你试图构建大型的软件系统时。比如，链接器报告说它无法解析一个引用，这是什么意思？静态变量和全局变量的区别是什么？如果你在不同的 C 文件中定义了名字相同的两个全局变量会发生什么？静态库和动态库的区别是什么？我们在命令行上排列库的顺序有什么影响？最严重的是，为什么有些链接错误直到运行时才会出现？在第 7 章中，你将得到这些问题的答案。
- 避免安全漏洞。多年来，缓冲区溢出错误是造成大多数网络和 Internet 服务器上安全漏洞的主要原因。存在这些错误是因为很少有程序员能够理解需要限制从不受信任的源接收数据的数量和格式。学习安全编程的第一步就是理解数据和控制信息存储在程序栈上的方式会引起的后果。作为学习汇编语言的一部分，我们将在第 3 章中描述堆栈原理和缓冲区溢出错误。我们还将学习程序员、编译器和操作系统可以用来降低攻击威胁的方法。

## 1.4　处理器读并解释储存在内存中的指令

此刻，hello.c 源程序已经被编译系统翻译成了可执行目标文件 hello，并被存放在磁盘上。要想在 Unix 系统上运行该可执行文件，我们将它的文件名输入到称为 shell 的应用程序中：

```
linux> ./hello
hello, world
linux>
```

shell 是一个命令行解释器，它输出一个提示符，等待输入一个命令行，然后执行这个命令。如果该命令行的第一个单词不是一个内置的 shell 命令，那么 shell 就会假设这是一个可执行文件的名字，它将加载并运行这个文件。所以在此例中，shell 将加载并运行 hello 程序，然后等待程序终止。hello 程序在屏幕上输出它的消息，然后终止。shell 随后输出一个提示符，等待下一个输入的命令行。

### 1.4.1　系统的硬件组成

为了理解运行 hello 程序时发生了什么，我们需要了解一个典型系统的硬件组织，如图 1-4 所示。这张图是近期 Intel 系统产品族的模型，但是所有其他系统也有相同的外观和特性。现在不要担心这张图很复杂——我们将在本书分阶段对其进行详尽的介绍。

**1. 总线**

贯穿整个系统的是一组电子管道，称作总线，它携带信息字节并负责在各个部件间传递。通常总线被设计成传送定长的字节块，也就是字(word)。字中的字节数(即字长)是一个基本的系统参数，各个系统中都不尽相同。现在的大多数机器字长要么是 4 个字节(32 位)，要么是 8 个字节(64 位)。本书中，我们不对字长做任何固定的假设。相反，我们将在需要明确定义的上下文中具体说明一个“字”是多大。

**2. I/O 设备**

I/O(输入/输出)设备是系统与外部世界的联系通道。我们的示例系统包括四个 I/O 设备：作为用户输入的键盘和鼠标，作为用户输出的显示器，以及用于长期存储数据和程序的磁盘驱动器(简单地说就是磁盘)。最开始，可执行程序 hello 就存放在磁盘上。

每个 I/O 设备都通过一个控制器或适配器与 I/O 总线相连。控制器和适配器之间的区