第 4 章　虚拟机的安装和使用

和其他所有计算机语言一样，汇编语言程序设计具有很强的实践性，不实际上机操作，不思考，不能举一反三，是无法掌握它的。但是，当程序编译完成后，如何让处理器执行它呢？还有，如何才能知道执行的结果是不是正确呢？这都是非常重要的问题，要在本章里解决。本章的目标是：

1. 了解计算机的开机启动过程。只有这样，你才能知道我们应当把编译好的程序放到哪里才会被处理器执行到。
2. 了解硬盘的构造和作用。
3. 了解 VirtualBox 虚拟机的功能，下载和安装 VirtualBox 虚拟机软件，创建一台本书中要用到的虚拟机，学会往虚拟硬盘中写数据，学会 VirtualBox 虚拟机的使用方法。

4.1　计算机的启动过程

4.1.1　如何将编译好的程序提交给处理器

对于绝大多数编译好的程序来说，要想得到处理器的光顾，让它执行一下，必须借助于操作系统。就拿 Windows 来说，它为你显示每个程序的图标，允许你双击来运行它们。在内部你看不见的层面上，它必须给将要运行的程序分配空闲的内存空间，并在适当的时候将程序提交给处理器执行。

每种操作系统都对它所管理的程序提出了种种格式上的要求。比如，它要求编译好的程序必须在文件的开始部分包含编译日期，是针对哪种操作系统编译的，程序的版本，第一条指令从哪里开始，数据段从哪里开始、有多长，代码段从哪里开始、有多长，等等，Windows 甚至建议你在文件中包含至少一个用于显示的图标。如果你不按它的要求来，它也不会给你面子，并直截了当地弹出一个对话框，如图 4-1 所示，告诉你它不准备，也没办法将你的程序提交给处理器。

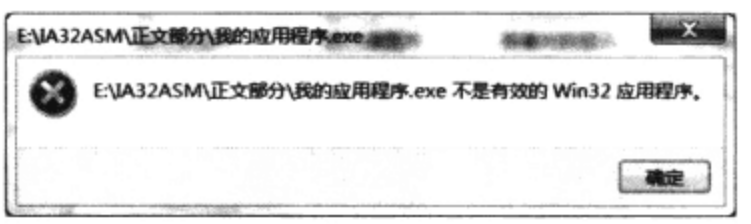

图 4-1　每种操作系统都会定义它自己的可执行文件格式

每种编译器都有能力针对不同的操作系统来生成不同格式的二进制文件，程序员所要做的，就是在源程序中加入一些相关的信息，比如指定每个段的开始和结束，并在编译时指定适当的参数。如果你对此感兴趣，可以阅读 NASM 文档。这是一个 PDF 文件，在安装 NASM 的时候，它也会被安装。

在特定的操作系统上开发软件肯定不是一件容易的事情。但换个角度考虑一下，操作系统也是一个需要在处理器上运行的软件，只不过比起一般的程序而言，体积更为庞大，功能更为复杂而已。如果我们能绕过它，或者代替它，让计算机一开机的时候直接执行我们自己的软件，岂不

更简单？

好，这个主意完全可行。那就让我们慢慢开始吧。

4.1.2　计算机的加电和复位

在处理器众多的引脚中，有一个是 RESET，用于接受复位信号。每当处理器加电，或者 RESET 引脚的电平由低变高时[①]，处理器都会执行一个硬件初始化，以及一个可选的内部自测试（Build-in Self-Test，BIST），然后将内部所有寄存器的内容初始到一个预置的状态。

比如，对于 Intel 8086 来说，复位将使代码段寄存器（CS）的内容为 0xFFFF，其他所有寄存器的内容都为 0x0000，包括指令指针寄存器（IP）。8086 之后的处理器并未延续这种设计，但毫无疑问，无论怎么设计，都是有目的的。

处理器的主要功能是取指令和执行指令，加电或者复位之后，它就会立刻尝试去做这样的工作。不过，在这个时候，内存中还没有任何有意义的指令和数据，它该怎么办呢？

在揭开谜底之前，我们先来看看内存的特点。

为了节约成本，并提高容量和集成度，在内存中，每个比特的存储都是靠一个极其微小的晶体管，外加一个同样极其微小的电容来完成的。可以想象，这样微小的电容，其泄漏电荷的速度当然也非常快。所以，个人计算机中使用的内存需要定期补充电荷，这称为刷新，所以这种存储器也称为动态随机访问存储器（Dynamic Random Access Memory，DRAM）。随机访问的意思是，访问任何一个内存单元的速度和它的位置（地址）无关。举个例子来说，从头至尾在一盘录音带上找某首歌曲，它越靠前，找到它所花的时间就越短。但内存就不一样，读写地址为 0x00001 的内存单元，和读写地址为 0xFFFF0 的内存单元，所需要的时间是一样的。

在内存刷新期间，处理器将无法访问它。这还不是最麻烦的，最麻烦的是，在它断电之后，所有保存的内容都会统统消失。所以，每当处理器加电之后，它无法从内存中取得任何指令。

4.1.3　基本输入输出系统

Intel 8086 可以访问 1MB 的内存空间，地址范围为 0x00000 到 0xFFFFF。出于各方面的考虑，计算机系统的设计者将这 1MB 的内存空间从物理上分为几个部分。

8086 有 20 根地址线，但并非全都用来访问 DRAM，也就是内存条。事实上，这些地址线经过分配，大部分用于访问 DRAM，剩余的部分给了只读存储器 ROM 和外围的板卡，如图 4-2 所示。

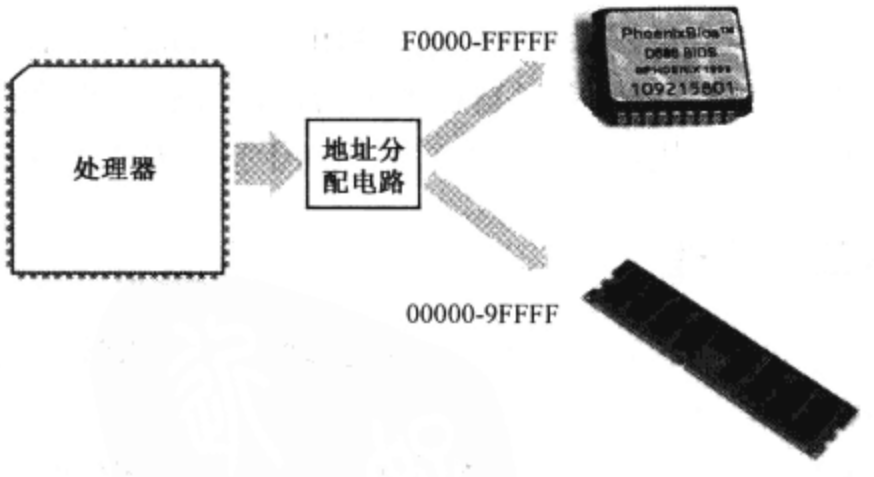

图 4-2　8086 系统的内存空间分配

① 比如，当你按下主机箱面板上的 RESET 按钮时，就会导致 RESET 引脚电平的变化，从而使计算机热启动。

与 DRAM 不同，只读存储器（Read Only Memory，ROM）不需要刷新，它的内容是预先写入的，即使掉电也不会消失，但也很难改变。这个特点很有用，比如，可以将一些程序指令固化在 ROM 中，使处理器在每次加电时都自动执行。处理器醒来后不能饿着，这是很重要的。

在以 Intel 8086 为处理器的系统中，ROM 占据着整个内存空间顶端的 64KB，物理地址范围是 0xF0000～0xFFFFF，里面固化了开机时要执行的指令；DRAM 占据着较低端的 640KB，地址范围是 0x00000～0x9FFFF；中间还有一部分，分给了其他外围设备，这个以后再说。因为 8086 加电或者复位时，CS=0xFFFF，IP=0x0000，所以，它取的第一条指令位于物理地址 0xFFFF0，正好位于 ROM 中，那里固化了开机时需要执行的指令。

处理器取指令执行的自然顺序是从内存的低地址往高低地址推进。如果从 0xFFFF0 开始执行，这个位置离 1MB 内存的顶端（物理地址 0xFFFFF）只有 16 个字节的长度，一旦 IP 寄存器的值超过 0x000F，比如 IP=0x0011，那么，它与 CS 一起形成的物理地址将因为溢出而变成 0x00001，这将回绕到 1MB 内存的最低端。

所以，ROM 中位于物理地址 0xFFFF0 的地方，通常是一个跳转指令，它通过改变 CS 和 IP 的内容，使处理器从 ROM 中的较低地址处开始取指令执行。在 NASM 汇编语言里，一个典型的跳转指令像这样：

```
        jmp 0xf000:0xe05b
```

在这里，“jmp”是跳转（jump）的简化形式；0xf000 是要跳转到的段地址，用来改变 CS 寄存器的内容；0xe05b 是目标代码段内的偏移地址，用来改变 IP 寄存器的内容。因此，目标位置的物理地址是 0xfe05b。一旦执行这条指令，处理器将开始从指定的“段: 偏移”处开始重新取指令执行。

到了本书第 5 章我们就能接触跳转指令了，现在，我们只需要知道，指令的执行并非总是顺序的，有时候不得不根据某些条件来选择执行哪些指令，不执行哪些指令。这个时候，跳转指令是很有用的。

这块 ROM 芯片中的内容包括很多部分，主要是进行硬件的诊断、检测和初始化。所谓初始化，就是让硬件处于一个正常的、默认的工作状态。最后，它还负责提供一套软件例程，让人们在不必了解硬件细节的情况下从外围设备（比如键盘）获取输入数据，或者向外围设备（比如显示器）输出数据。设备当然是很多的，所以这块 ROM 芯片只针对那些最基本的、对于使用计算机而言最重要的设备，而它所提供的软件例程，也只包含最基本、最常规的功能。正因为如此，这块芯片又叫基本输入输出系统（Base Input & Output System，BIOS）ROM。在读者缺乏基础知识的情况下讲述 ROM-BIOS 的工作只会越讲越糊涂，所以这些知识将会分散在各个章节里予以讲解。

ROM-BIOS 的容量是有限的，当它完成自己的使命后，最后所要做的，就是从辅助存储设备读取指令数据，然后转到那里开始执行。基本上，这相当于接力赛中的交接棒。

4.1.4 硬盘及其工作原理

历史上，有多种辅助存储设备，比如软盘、光盘、硬盘、U 盘等，相对于内存，它们就是人们常说的“外存”，即外存储器（设备）。

从软盘（Floppy Disk）启动计算机，这已经是过去的事了。软盘的尺寸比烟盒稍大一点，但是比较薄，采用塑料作为基片，上面是一层磁性物质，可以用来记录二进制位。这种塑料介质比较柔软，所以称为软盘。

在数据记录原理上和软盘很相似的设备是硬盘（Hard Disk，HDD），而且它们几乎是同一个时代的产物。但是，与软盘不同，硬盘是多盘片、密封、高转速的，采用铝合金作为基片，并在表

面涂上磁性物质来记录二进制位。这就使得它的盘片具有较高的硬度，故称为硬盘。

如图 4-3 所示，这是一块被拆开的硬盘，中间是用于记录数据的铝合金盘片，固定在中心的轴上，由一个高速旋转的马达驱动。附着在盘片表面的扁平锥状物，就是用于在盘片上读写数据的磁头。

图 4-3　一块被拆开密封盖的硬盘

为了进一步搞清楚硬盘的内部构造，图 4-4 给出了更为详细的图示。

硬盘可以只有一个盘片（这称为单碟），也可能有好几个盘片。但无论如何，它们都串在同一个轴上，由电动机带动着一起高速旋转。一般来说，转速可以达到每分钟 3600 转或者 7200 转，有的能达到一万多转，这个参数就是我们常说的“转/分钟”（Round Per Minute，RPM）。

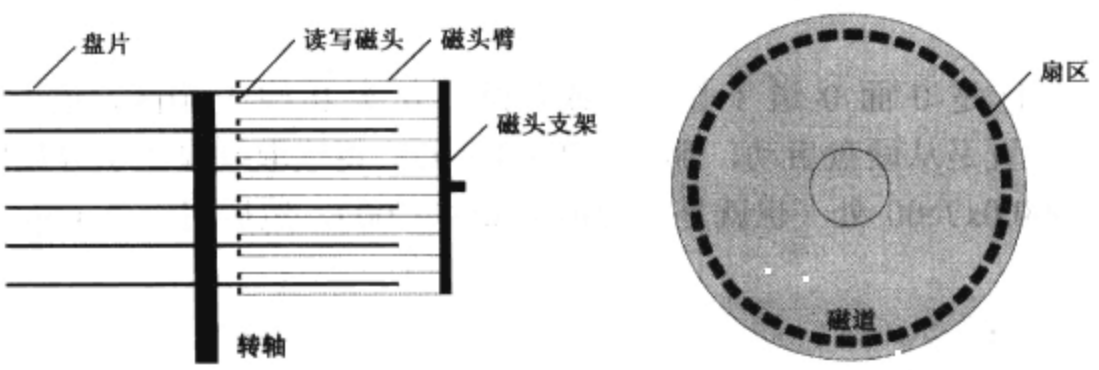

图 4-4　硬盘的结构示意图

每个盘片都有两个磁头（Head），上面一个，下面一个，所以经常用磁头来指代盘面。磁头都有编号，第 1 个盘片，上面的磁头编号为 0，下面的磁头编号为 1；第 2 个盘片，上面的磁头编号为 2，下面的磁头编号为 3，以此类推。

每个磁头不是单独移动的。相反，它们都通过磁头臂固定在同一个支架上，由步进电动机带动着一起在盘片的中心和边缘之间来回移动。也就是说，它们是同进退的。步进电动机由脉冲驱动，每次可以旋转一个固定的角度，即可以步进一次。

可以想象，当盘片高速旋转时，磁头每步进一次，都会从它所在的位置开始，绕着圆心“画”出一个看不见的圆圈，这就是磁道（Track）。磁道是数据记录的轨迹。因为所有磁头都是联动的，故每个盘面上的同一条磁道又可以形成一个虚拟的圆柱，称为柱面（Cylinder）。

磁道，或者柱面，也要编号。编号是从盘面最边缘的那条磁道开始，向着圆心的方向，从 0 开始编号。

柱面是一个用来优化数据读写的概念。初看起来，用硬盘来记录数据时，应该先将一个盘面填满后，再填写另一个盘面。实际上，移动磁头是一个机械动作，看似很快，但对处理器来说，却很漫长，这就是寻道时间。为了加速数据在硬盘上的读写，最好的办法就是尽量不移动磁头。这样，当 0 面的磁道不足以容纳要写入的数据时，应当把剩余的部分写在 1 面的同一磁道上。如果还写不下，那就继续把剩余的部分写在 2 面的同一磁道上。换句话说，在硬盘上，数据的访问是以柱面来组织的。

实际上，磁道还不是硬盘数据读写的最小单位，磁道还要进一步划分为扇区（Sector）。磁道很窄，也看不见，但在想象中，它仍呈带状，占有一定的宽度。将它划分许多分段之后，每一部分都呈扇形，这就是扇区的由来。

每条磁道能够划分为几个扇区，取决于磁盘的制造者，但通常为 63 个。而且，每个扇区都有

一个编号，与磁头和磁道不同，扇区的编号是从 1 开始的。

扇区与扇区之间以间隙（空白）间隔开来，每个扇区以扇区头开始，然后是 512 个字节的数据区。扇区头包含了每个扇区自己的信息，主要有本扇区的磁道号、磁头号和扇区号，用来供硬盘定位机构使用。现代的硬盘还会在扇区头部包括一个指示扇区是否健康的标志，以及用来替换该扇区的扇区地址。用于替换扇区的，是一些保留和隐藏的磁道。

4.1.5 一切从主引导扇区开始

尽管我们使用硬盘的历史很长，但它一直没能退出舞台，这主要是因为它总能通过不断提高自己的容量来打败那些竞争者。20 世纪 90 年代初，40MB 的硬盘算是常见的，能拥有 200MB 的硬盘很让人羡慕。看看现在，500GB 的硬盘也不算稀罕，而且价钱也很便宜。

前面说到，当 ROM-BIOS 完成自己的使命之前，最后要做的一件事是从外存储设备读取更多的指令来交给处理器执行。现实的情况是，绝大多数时候，对于 ROM-BIOS 来说，硬盘都是首选的外存储设备。

硬盘的第一个扇区是 0 面 0 道 1 扇区，或者说是 0 头 0 柱 1 扇区，这个扇区称为主引导扇区。如果计算机的设置是从硬盘启动，那么，ROM-BIOS 将读取硬盘主引导扇区的内容，将它加载到内存地址 0x0000:0x7c00 处（也就是物理地址 0x07C00），然后用一个 jmp 指令跳到那里接着执行：

```
jmp 0x0000:0x7c00
```

为什么偏偏是 0x7c00 这个地方？还不太清楚。反正当初定下这个方案的家伙已经被人说了很多坏话，我也就不准备再多说什么了。

通常，主引导扇区的功能是继续从硬盘的其他部分读取更多的内容加以执行。像 Windows 这样的操作系统，就是采用这种接力的方法一步一步把自己运行起来的。

说到这里，我们可以想象，如果我们把自己编译好的程序写到主引导扇区，不也能够让处理器执行吗？

对于这种想法，我有一个好消息和一个坏消息要告诉你。

好消息是，这是可以的，而且这几乎是在不依赖操作系统的情况下，让我们的程序可以执行的唯一方法。

不过，坏消息是，如果你改写了硬盘的主引导扇区，那么，Windows 和 Linux，以及任何你正在使用的操作系统都会瘫痪，无法启动了。

那么，我们该怎么办呢？答案是在你现有的计算机上，再虚拟出一台计算机来。

◆ **检测点 4.1**

1. 硬盘的磁头（盘面）是从数字（ ）开始编号的；每个盘面磁道是从数字（ ）开始编号的；每磁道/柱面上的扇区是从数字（ ）开始编号的，主引导扇区的位置是（ ）面（ ）道（ ）扇区；
2. 如果希望处理器从当前位置转移到物理地址 0xc5030 处开始执行，可以使用下面的哪些指令（可多选）：

   A. jmp 0xc000:0x5030　B. jmp 0xc500:0x0030

   C. jmp 0xc503:0x0000　D. jmp 0xbb00:0xa030

4.2 创建和使用虚拟机

4.2.1 别害怕，虚拟机是软件

对于第一次听说虚拟机（Virtual Machine，VM）的人来说，可能以为还要再花钱买一台计算机，这恐怕是他们最担心的。所谓虚拟机，就是在你的计算机上再虚拟出另一台计算机来。这台虚拟出来的计算机，和真正的计算机一样，可以启动，可以关闭，还可以安装操作系统、安装和运行各种各样的软件，或者访问网络。总之，你在真实的计算机上能做什么，在它里面一样可以那么做。使用虚拟机，你会发现，在 Windows 操作系统里，居然又可以拥有另一套 Windows。然而本质上，它只是运行在物理计算机上的一个软件程序。

如图 4-5 所示，整个大的背景，是 Windows 7 的桌面，它安装在一台真实的计算机上。图中的小窗口，正是虚拟机，运行的是 Windows Server 2003。像这样，我们就得到了两台“计算机”，而且它们都可以操作。

虚拟机仅仅是一个软件，运行在各种主流的操作系统上。它以自己运行的真实计算机为模板，虚拟出另一套处理器、内存和外围设备来。它的处理能力，完全来自于背后那台真实的计算机。

尤其重要的是，针对某种真实处理器所写的任何指令代码，通常都可以正确无误地在该处理器的虚拟机上执行。实际上，这也是虚拟机具有广泛应用价值的原因所在。

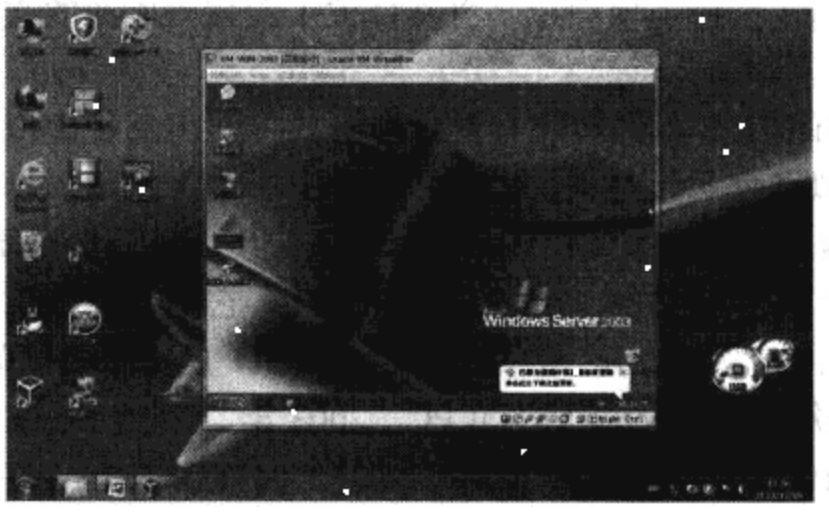

图 4-5 虚拟机的实例

在过去的若干年里，虚拟机得到了广泛应用。为了研制防病毒软件、测试最新的操作系统或者软件产品，软件公司通常需要多台用于做实验的计算机。采用虚拟机，就可以避免反复重装软件系统的麻烦，当这些软件系统崩溃时，崩溃的只是虚拟机，而真实的物理计算机丝毫不受影响。

利用虚拟机来教学，本书不是第一个，国内外都流行这种教学方式。虚拟机利用软件来模拟完整的计算机系统，无须添加任何新的设备，而且与主计算机系统是隔离的，在虚拟机上的任何操作都不会影响到物理计算机上的操作系统和软件，这对拥有大量计算机的培训机构来说，可以极大地节省维护上的成本。

4.2.2 下载和安装 Oracle VM VirtualBox

主流的虚拟机软件包括 VMWare、Virtual PC 和 VirtualBox 等，但只有 VirtualBox 是开源和

免费的。

要使用 VirtualBox，首先必须从网上下载并安装它。这里是它的主页：

```
https://www.virtualbox.org/
```

通过这个主页，你可以找到最新的版本并下载它。为了方便，下面给出下载页面的链接：

```
https://www.virtualbox.org/wiki/Downloads
```

本书的配书文件包中提供了关于如何下载、安装和配置 VirtualBox 软件的文档，有 WORD 和 PDF 两种版本，请选择使用。注意，要选择最新的版本下载，而且，由于该软件针对不同的操作系统平台开发，因此，要下载适用于 Windows 的安装程序。

VirtualBox 软件安装完毕之后 ，你需要创建，或者说“虚拟”出一台计算机来，并设置该“计算机”的相关参数，包括为它配备一块硬盘。有关的方法和步骤在配书文件包的教程中已有介绍，唯一的建议是选用本书为你准备的虚拟硬盘。

和真实的计算机一样，虚拟机也需要一个或几个辅助存储器（磁盘、光盘、U 盘等）才能工作。不过，为它配备的并非真正的盘片，而是一个特殊的文件，故称为虚拟盘。这样，当一个软件程序在虚拟机里读写硬盘或者光盘时，虚拟机将把它转换成对文件的操作，而软件程序还以为自己真的是在读写物理盘片。这样的一块磁盘，在需要的时候随时创建，不需要时可以随时删除，这真是非常神奇的磁盘。

前面你已经从网上下载了与本书配套的源码和工具，那是个压缩文件。解压之后，在源代码和工具文件夹里有一个现成的虚拟硬盘文件，文件名是 LEECHUNG.VHD，这是给你额外准备的，而且经过了测试，可以在你无法创建虚拟硬盘的时候派上用场。不管是你自己创建虚拟硬盘，还是选用这个现成的，都应当使虚拟硬盘文件位于源代码所在的文件夹，将来往该虚拟硬盘写数据时比较方便。

正如前面所说的，市面上有好几种流行的虚拟机软件，而每种虚拟机软件都企图制定自己的虚拟硬盘标准。因为虚拟硬盘实际是一个文件，所以，所谓虚拟硬盘标准，实际上就是该文件的格式。正是因为这样，虚拟硬盘类型说白了就是你准备采用哪家的虚拟硬盘文件格式。

因为虚拟硬盘实际上是一个文件，所以，通常来说，它的格式体现在它的文件扩展名上。比如上面的 LEECHUNG.VHD，采用的就是微软公司的 VHD 虚拟硬盘规范。VHD 规范最早起源于 Connectix 公司的虚拟机软件 Connectix Virtual PC，2003 年，微软公司收购了它并改名为 Microsoft Virtual PC。2006 年，微软公司正式发布了 VHD 虚拟硬盘格式规范。在本书配套的源代码和工具包里，有该规范的文档。

VDI 是 VirtualBox 自己的虚拟硬盘规范，VMDK 是 VMWare 的虚拟硬盘规范。采用哪个公司、哪个虚拟机软件的虚拟硬盘格式，对于普通的应用来说，这没什么关系，它们都能很好地工作。但是，对于本书和本书配套的工具来说，你必须选择“VHD（Virtual Hard Disk）”。具体原因，我们将在下一节讲述。

事实上，即使是 VHD，也分为两种类型：固定尺寸的和动态分配的。一个固定尺寸的 VHD，它对应的文件尺寸和该虚拟硬盘的容量是相同的，或者说是一次性分配够了的。比如，一个 2GB 的 VHD 虚拟硬盘，它对应的文件大小也是 2GB。注意，本书以及本书配套的工具仅支持固定尺寸的 VHD。

一旦完成了全部的准备工作，刚刚创建的虚拟机就会显示在 VirtualBox 控制台里，如图 4-6 所示，虚拟机的名字叫“LEARN-ASM”。基本上，你现在就可以单击控制台界面上的“开始”来启动这台虚拟机。但是，别忙，你的虚拟硬盘里还没有东西呢。

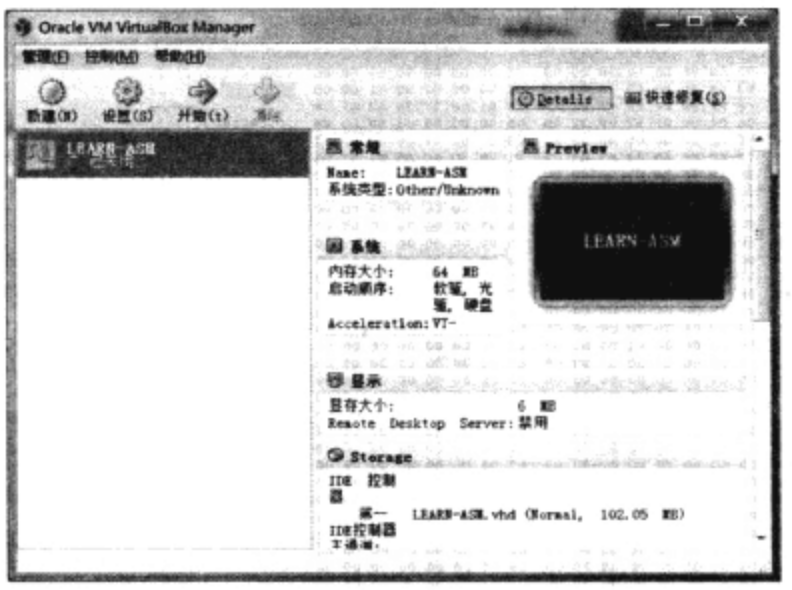

图 4-6 通过向导程序创建的 LEARN-ASM 虚拟机

4.2.3 虚拟硬盘简介

坦白地说，之所以要采用固定尺寸的 VHD 虚拟硬盘，是因为其简单性。我们知道，虚拟硬盘实际上是一个文件。固定尺寸的 VHD 虚拟硬盘是一个具有“.vhd”扩展名的文件，它仅包括两个部分，前面是数据区，用来模拟实际的硬盘空间，后面跟着一个 512 字节的结尾（2004 年前的规范里只有 511 字节）。

要访问硬盘，运行中的程序必须至少向硬盘控制器提供 4 个参数，分别是磁头号、磁道号、扇区号，以及访问意图（是读还是写）。

硬盘的读写是以扇区为最小单位的。所以，无论什么时候，要从硬盘读数据，或者向硬盘写数据，至少得是 1 个扇区。

你可能想，我只有 2 字节的数据，不足以填满一个扇区，怎么办呢？

这是你自己的事。你可以用无意义的废数字来填充，凑够一个扇区的长度，然后写入。读取的时候也是这样，你需要自己跟踪和把握从扇区里读到的数据，哪些是你真正想要的。换句话说，硬盘只是机械和电子的组合，它不会关心你都写了些什么。要是手机像人类一样智能，它一定会在坏人使用它的时候无法开机。

在 VHD 规范里，每个扇区是 512 字节。VHD 文件一开始的 512 字节，就对应着物理硬盘的 0 面 0 道 1 扇区。然后，VHD 文件的第二个 512 字节，对应着 0 面 0 道 2 扇区，后面的以此类推，一直对应到 0 面 0 道 *n* 扇区。这里，*n* 等于每磁道的扇区数。

再往后，因为硬盘的访问是按柱面进行的，所以，在 VHD 文件中，紧接着前面的数据块，下一个数据块对应的是 1 面 0 道 1 扇区，就这样一直往后排列，当把第一个柱面全部对应完后，再从第二个柱面开始对应。

如图 4-7 所示，为了标志一个文件是 VHD 格式的虚拟硬盘，并为使用它的虚拟机提供该硬盘的参数，在 VHD 文件的结尾，包含了 512 字节的格式信息。为了观察这些信息，我们使用了前面已经介绍过的配书工具 HexView。

如图 4-7 所示，文件尾信息是以一个字符串“conectix”开始的。这个标志用来告诉试图打开它的虚拟机，这的确是一个合法的 VHD 文件。该标志称为 VHD 创建者标识，就是说，该公司（conectix）创建了 VHD 文件格式的最初标准。

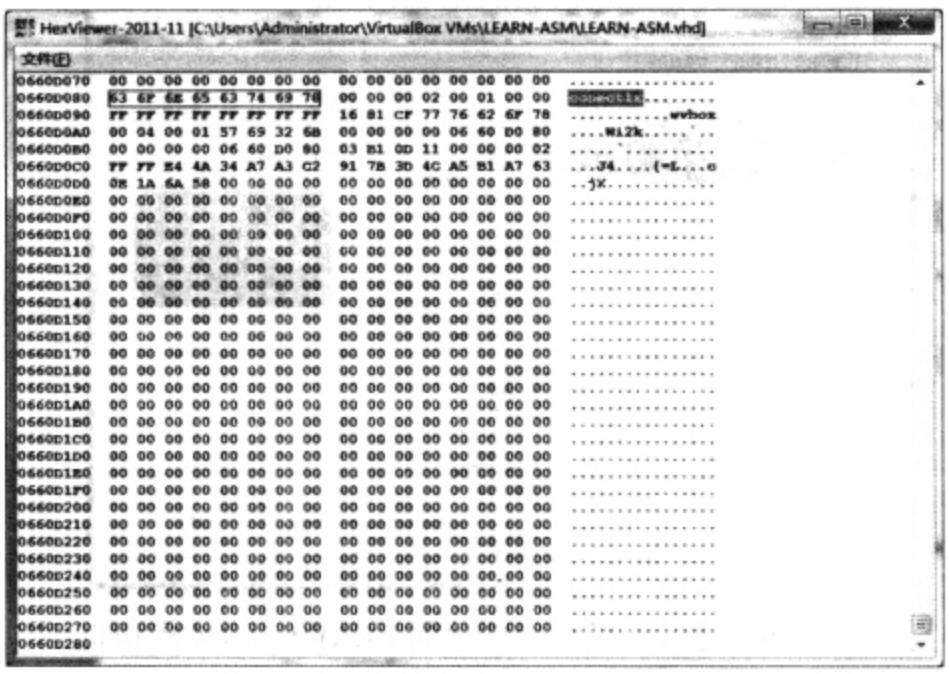

图 4-7　VHD 文件的格式信息

从这个标志开始，后面的数据包含了诸如文件的创建日期、VHD 的版本、创建该文件的应用程序名称和版本、创建该文件的应用程序所属的操作系统、该虚拟硬盘的参数（磁头数、每面磁道数、每磁道扇区数）、VHD 类型（固定尺寸还是动态增长）、虚拟硬盘容量等。

说到这里，也许你已经明白我为什么要在书中使用固定尺寸的 VHD。是的，因为它简单。为了学习汇编语言，我们不得不在硬盘上直接写入程序。因为 VHD 格式简单，所以我只花了很少的时间就开发了一个虚拟硬盘写入程序，作为配书工具让大家使用，这就是下一节将要介绍的 FixVhdWr。

至于为什么要使用 VirtualBox 虚拟机，是因为它支持 VHD，而且是免费的。先前版本的 VirtualBox 可以识别 VHD，但不支持创建新的 VHD，尽管微软公司很早就公开了 VHD 规范。好消息是现在的 VirtualBox 也可以创建 VHD 了。

4.2.4　练习使用 FixVhdWr 工具向虚拟硬盘写数据

通常，VHD 是由虚拟机 VirtualBox 使用的。应用程序像往常一样，直接针对硬盘进行操作，而在底层，虚拟机将这些硬件访问转化为对文件的读写。

为了在处理器加电或者复位之后能够执行我们写的程序，势必要将这些程序写到硬盘的主引导扇区里，也就是 0 面 0 道 1 扇区，即使是在虚拟机工作环境中，也是这样。

为了做到这一点，需要一个专门针对虚拟硬盘进行读写的工具。我自己写了一个，就在配书源代码和工具里，名叫 FixVhdWr。

FixVhdWr 只针对固定尺寸的 VHD。当它启动之后，首先需要选择要读写的 VHD 文件。如图 4-8 所示，一旦这是个合法的 VHD 文件，它将读取该文件的结尾，并显示该虚拟硬盘的信息。

注意，因为 FixVhdWr 只针对固定尺寸的 VHD，所以，如果它检测到该 VHD 是一个动态虚拟硬盘，则“下一步”按钮处于禁止状态。

第二步是选择要写入虚拟硬盘的数据文件。毕竟，在任何操作系统中，数据都是以文件的方式组织的，如图 4-9 所示。

最后一个界面，是执行写入操作，如图 4-10 所示，你应该选择第一种写入方式，即“LBA 连续直写模式”，并指定起始的逻辑扇区号。

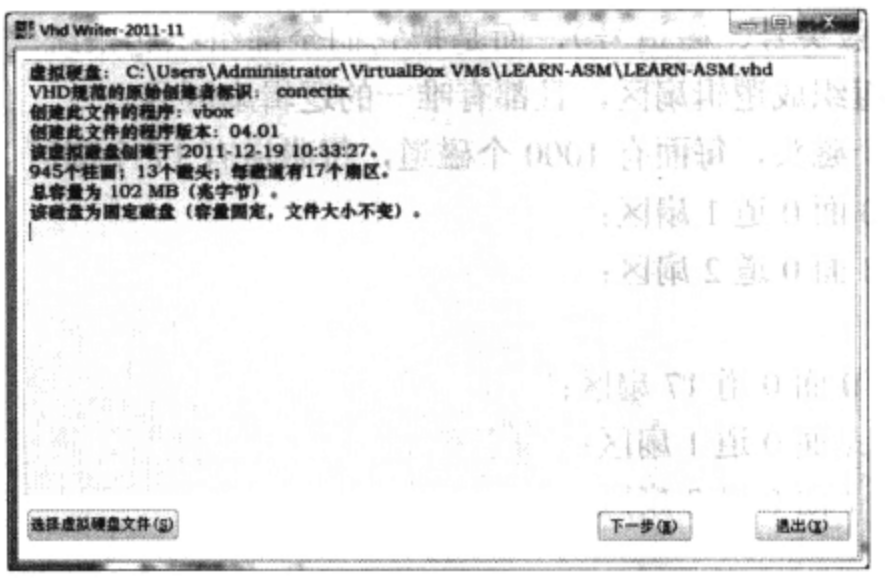

图 4-8 打开 VHD 文件并显示该虚拟硬盘的信息

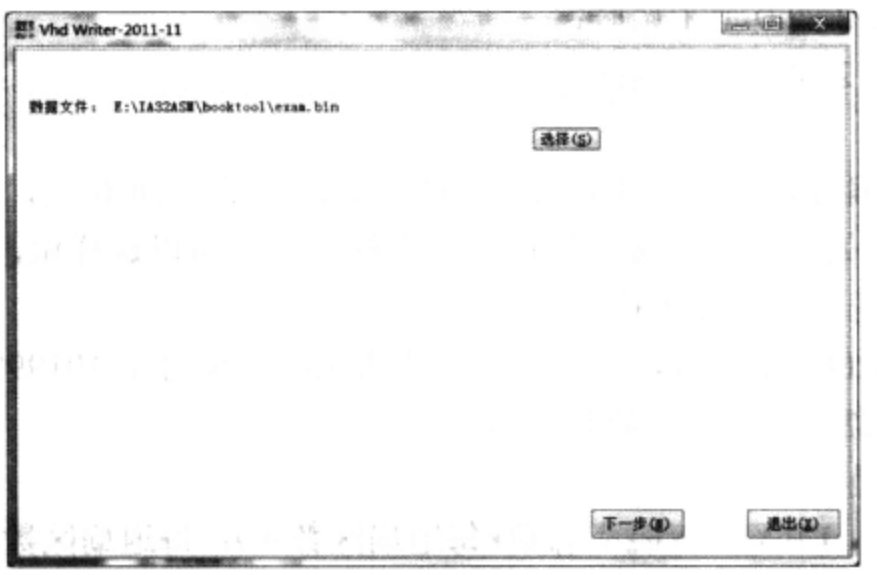

图 4-9 选择要写入虚拟硬盘的文件

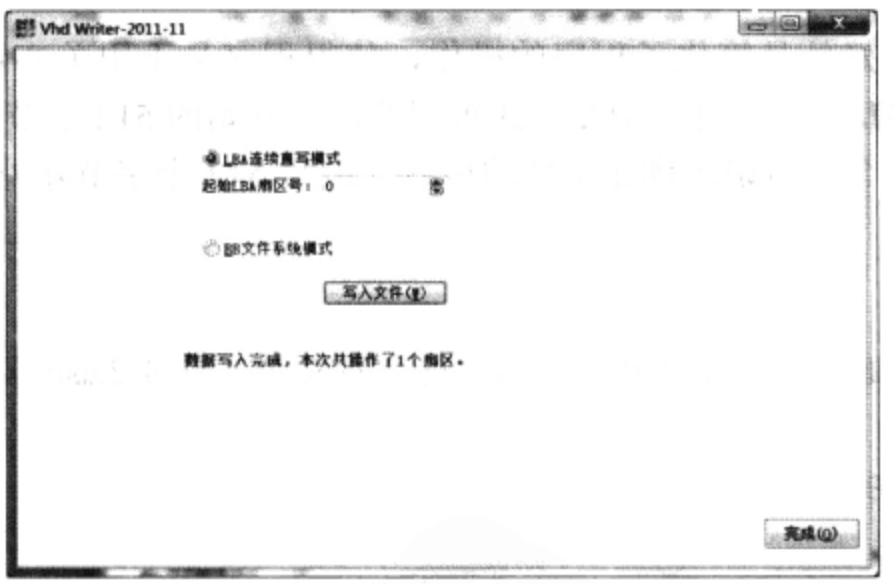

图 4-10 指定数据写入时的起始逻辑扇区号

通常，一个扇区的尺寸是 512 字节，可以看成一个数据块。所以，从这个意义上来说，硬盘是一个典型的块（Block）设备。

采用磁头、磁道和扇区这种模式来访问硬盘的方法称为 CHS 模式，但不是很方便。想想看，如果有一大堆数据要写，还得注意磁头号、磁道号和扇区号不要超过界限。所以，后来引入了逻辑块地址（Logical Block Address，LBA）的概念。现在市场上销售的硬盘，无论是哪个厂家生产的，都支持 LBA 模式。

LBA 模式是由硬盘控制器在硬件一级上提供支持，所以效率很高，兼容性很好。LBA 模式不

考虑扇区的物理位置（磁头号、磁道号），而是把它们全部组织起来统一编号。在这种编址方式下，原先的物理扇区被组织成逻辑扇区，且都有唯一的逻辑扇区号。

比如，某硬盘有 6 个磁头，每面有 1000 个磁道，每磁道有 17 个扇区。那么：

逻辑 0 扇区对应着 0 面 0 道 1 扇区；

逻辑 1 扇区对应着 0 面 0 道 2 扇区；

……

逻辑 16 扇区对应着 0 面 0 道 17 扇区；

逻辑 17 扇区对应着 1 面 0 道 1 扇区；

逻辑 18 扇区对应着 1 面 0 道 2 扇区；

……

逻辑 33 扇区对应着 1 面 0 道 17 扇区；

逻辑 34 扇区对应着 2 面 0 道 1 扇区；

逻辑 35 扇区对应着 2 面 0 道 2 扇区；

……

要注意到，扇区在编号时，是以柱面为单位的。即，先是 0 面 0 道，接着是 1 面 0 道，直到把所有盘面上的 0 磁道处理完，再接着处理下一个柱面。之所以这样做，是因为我们讲过，要加速硬盘的访问速度，最好是尽可能不移动磁头。

因为这里总共有 102000 个扇区，故最后一个逻辑扇区的编号是 101999，它对应着 5 面 999 道 17 扇区，这也是整个硬盘上最后一个物理扇区。

这里面的计算方法是：

$$\text{LBA} = C\times\text{磁头总数}\times\text{每道扇区数} + H\times\text{每道扇区数} + (S-1)$$

这里，LBA 是逻辑扇区号，C、H、S 是想求得逻辑扇区号的那个物理扇区所在的磁道、磁头和扇区号。

采用 LBA 模式的好处是简化了程序的操作，使得程序员不用关心数据在硬盘上的具体位置。对于本书来说，VHD 文件是按 LBA 方式组织的，一开始的 512 字节就是逻辑 0 扇区，然后是逻辑 1 扇区；最后一个逻辑扇区排在文件的最后（最后 512 个字节除外，那是 VHD 文件的标识部分）。

◆ **检测点 4.2**

1. 运行 NASMIDE 程序，输入以下汇编指令并保存为文件 4-2.asm（不要考虑这些指令的含义和功能）：

```
mov ax,0xb800
mov ds,ax
mov [0x00],'a'
mov [0x02],'s'
mov [0x04],'m'
jmp $
```

2. 将上面的 4-2.asm 文件编译，得到二进制文件 4-2.bin，并写入虚拟硬盘的主引导扇区。注意，该虚拟硬盘应当是 VirtualBox 虚拟机的启动硬盘。
3. 启动你的 VirtualBox 虚拟机。当虚拟机启动时，会像真实的计算机一样加载硬盘上的主引导扇区代码，并执行。此时，注意观察屏幕上都显示了什么内容。