1928 年，澳大利亚发明家弗里茨·佛勒玛（Fritz Phleumer）发明了一种磁记录设备，并为其申请了专利。此设备采用在生产香烟上的金属带时所用的技术，将铁粒子覆盖在很长的纸带上。不久以后，纸带被强度更高的醋酸盐纤维素取代，而一种更耐久、更知名的记录介质也从此诞生——卷轴式磁带，它被包装在塑料盒里，可以很方便地使用。对于记录和回放音乐及视频来说，卷轴式磁带无疑是很受欢迎的介质。

1950 年，雷明顿兰德公司（Remington Rand）发明了第一个用于记录计算机数字数据的商用磁带系统。当时，磁带的容量有限，一个 0.5 英寸的卷轴式磁带容量只有几兆字节。在早期家用计算机中，常见的盒式磁带录音机被人们用来保存信息。通过调用一些小的程序，可以将内存块中的内容保存到磁带上，以后再需要时还可以从磁带调入内存中。在第一代 IBM PC 上，有一个专为连接盒式磁带存储器而设计的接头。至今，磁带仍然是一种很通用的存储介质，对于那些想要长期保存的文档，磁带更是首要的选择。但是，磁盘并不是最理想的存储介质，想要快速地移动到磁盘的任一位置是不可能的，它只能顺序访问，频繁地快进和倒带会花费很多时间。

以几何学角度来看，磁盘是能够实现快速访问的介质。磁盘围绕其中心旋转，连到臂上的一个或多个磁头从磁盘外沿向中间移动，通过磁头可以快速访问磁盘上的任何区域。

在记录声音信息方面，磁盘实际上要早于磁带。早在 1956 年，IBM 公司就发明了世界上首款用来存储计算机数据的磁盘驱动器，称为 RAMAC（Random Access Method of Accounting and Control，计算与控制过程中的随机访问模式），它由 50 个金属盘片组成，直径 2 英尺，存储量为 5 MB。

自从磁盘被用作记录信息以来，它的体积越来越小而容量越来越大，习惯上将磁盘分为软盘（floppy disk）和硬盘（hard disk，或 fixed disk）。软盘是由单面覆盖磁性物质的塑料片组成，外面由厚纸板或塑料包装，起到保护作用（塑料包装主要是防止磁盘弯折，虽然现在的磁盘看起来不像以前的软盘那么松软，而且还有很多的区别，但软盘这个名字一直在用，延续至今）。在使用软盘的时候，必须将其插入到软盘驱动器，软盘驱动器是一个连接到计算机的部件，通过它可以读/写磁盘中的内容。早期的软盘直径为 8 英寸，第一代 IBM PC 使用的是 5.25 英寸的软盘，不过，现在最流行的是 3.5 英寸的软盘。软盘有很大的灵活性，可以实现在不同计算机之间传递数据。对商用软件来说，磁盘仍然是

一个不可或缺的发行媒介。

硬盘是由多个金属磁盘构成的，它永久驻留在驱动器里。相对于软盘来说，它的存取速度更快、存储量更大，唯一的缺点是硬盘本身是固定的，不能移动。

磁盘的表面被划分成许多同心圆，称为磁道（tracks），每个磁道又被划分成像圆饼切片形状的扇区（sectors），每个扇区可以存放一定数量的字节，通常为 512 字节。第一代 IBM PC 上使用的软盘只有一面可以存储信息，直径 5.25 英寸，被划分成 40 个磁道，每个磁道 8 个扇区，每个扇区存储 512 字节数据。通过计算，每个软盘可以存储 163,840 字节数据，即 160 KB。现今 PC 兼容机使用的软盘通常有两面，每面 80 个磁道，每个磁道 18 个扇区，每个扇区 512 字节，总的磁盘容量是 1,474,560 字节，即 1440 KB。

1983 年，IBM 在其 PC/XT 上率先使用了硬盘驱动器，当时硬盘的容量仅仅只有 10 MB。自此之后的 16 年里，硬盘的容量扩大了上百倍，而价格一直在下降。1999 年，20 GB（200 亿字节）容量的硬盘驱动器诞生了，而售价却不到 400 美元。

软盘和硬盘通常有它们自己的电气接口，除此之外，为了能和微处理器交互数据，这些电气接口与微处理器之间还需要有额外的接口与之相连。现在流行的硬盘驱动器标准接口有：小型计算机系统接口（Small Computer System Interface，SCSI）；增强的小型设备接口（Enhanced Small Device Interface，ESDI）和集成设备电气接口（Integrated Device Electronics，IDE）。这些接口都利用直接内存访问（direct memory access，DMA）技术来使用总线，DMA 可以不经过微处理器，实现数据在随机访问存储器和硬盘之间直接传送。这样的传送是以块为单位进行的，每次传输的块大小是磁盘扇区字节数的倍数，通常是 512 字节。

由于经常听到关于兆字节和吉字节之类的相关术语方面的谈论，让很多家用计算机的初学者感到困惑：到底随机访问存储器和磁盘存储器有什么区别？不过最近几年推出了一条分类规则，这让人们对术语的困惑减少了许多。这条规则规定：memory（内存）仅仅表示半导体随机访问存储器；storage（存储器）用来指任何的存储设备，通常包括软盘、硬盘和磁带。在本书中，尽量遵循这些规则，它的确能够给我们带来许多好处。

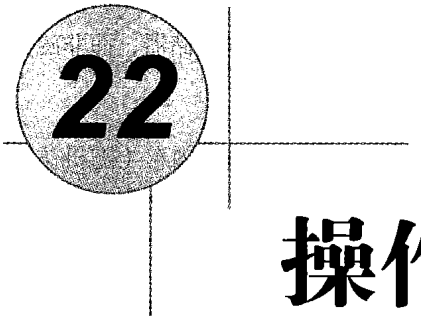
实际上，存储的信息是否易失，是随机访问存储器与磁介质存储器的主要区别。随机访问存储器是易失性存储设备，一旦掉电，存储内容将会消失；而磁介质存储器是永

久性存储设备，像软盘和磁盘，除非故意删除或写覆盖，否则数据将会一直保留不变。如果对微处理器的工作原理很了解，就会观察到随机访问存储器和磁介质存储器之间的显著区别，例如当微处理器发出一个地址信号，通常是寻址随机访问存储器，而非磁介质存储器。

微处理器不能直接从磁盘读取数据，需要将所需的数据从磁盘调入内存（随机访问存储器），然后它才能对其访问，当然这需要额外的步骤。微处理器还需要执行一段小程序，这段程序会访问磁盘，并将数据从磁盘调入内存。

关于随机访问存储器和磁介质存储器之间的差别，有个形象的比喻可以帮助我们加深理解：随机访问存储器就像办公桌的桌面，上面的任何东西都可以拿来直接使用；而磁介质存储器就像一个文件柜，里面的东西不能直接使用，如果想要使用放在文件柜里的某件东西，你需要站起来，走到文件柜前，查找需要的文件，然后带回桌面。如果桌面太拥挤，没有空间放置需要的文件，还需要把桌面上暂时不用的东西先放回到文件柜中。

这个比喻恰到好处，实际上，存储在磁盘上的数据的确是以文件（files）作为实体来存放的。存储和检索文件是操作系统（Operating System）很重要的一个功能，关于操作系统的相关知识，下一章将会进行专门介绍。



操作系统

一直以来，有一种想法在我们脑子里涌动：亲手去组装——在想象中进行虚拟组装也可以——一台近似完整的计算机。一块微处理器、一些随机访问存储器、一款键盘、一台视频显示器和一个磁盘驱动器是这台计算机所拥有的部件。当所有的硬件各就各位，我们激动地盯着计算机的开关，伸出手来给它上电，将这台计算机从沉睡中唤醒。或许上面描述的这一切会在你的脑海中产生一副维克多·弗兰肯斯坦[1]（Victor Frankenstein）组装怪物时的场景，甚至你或许还会想起老木偶匠盖比特（Geppetto）正在雕刻木偶匹诺曹（Pinocchio）的情形。

但我们还是漏了一些东西，既不是雷霆万钧的威力，也不是对着流星许下的纯洁愿望。让我们继续投入到工作中：运行这台新组装的计算机，请告诉我你眼前出现了什么?

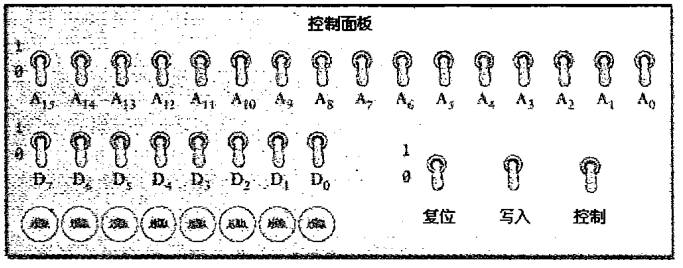
当阴极射线管加热之后，一串排列整齐——而又完全随机的——ASCII 码字符阵列出现在屏幕上。不出我们所料，掉电的时候，半导体存储器中的内容就会被全部清零；而

1 “弗兰肯斯坦”是英国著名小说家玛丽·雪莱（Mary Shelley）创作的同名小说中一个疯狂科学家的名字。小说中，弗兰肯斯坦用许多碎尸块拼接成一个“人”，并用闪电将其激活。玛丽·雪莱是英国著名浪漫主义诗人雪莱的妻子，被誉为科幻小说之母。

首次给它上电的时候，它将处于随机且不可预测的状态。同样，我们用来构建微处理器的所有 RAM 都包含随机的字节。如果可能，开机后微处理器会将这些随机字节解释为机器代码并执行。不用担心会因此发生什么糟糕的事情——计算机不会因此而坏掉——但是，计算机也无法完成任何有意义的工作。

这里我们漏掉的就是软件。当一个微处理器首次上电或复位时，它会从特定的内存地址开始执行机器代码。在英特尔的 8080 系统中，这个地址就是 0000h。在一台设计精良的计算机中，通过上电启动，将会有一条机器代码指令被载入到该内存地址中（一般情况下是一段程序的第一条指令）。

机器代码指令是怎么加载到那个内存地址中的呢？把软件安装到一台新设计的计算机的过程，可能是整个过程中最令人困惑的部分了，怎样理解它呢？让我们先从第 16 章所讲的一个控制面板入手吧，这个控制面板的功能是把字节写入随机访问存储器，之后还可将其读出。



与前面介绍过的有所不同，这个控制面板上设计了一个复位开关。复位开关与微处理器的复位输入相连接。一旦复位开关闭合（置 1），微处理器就会停止工作；当此开关断开后（置 0），微处理器就开始执行机器代码。

使用此控制面板的方法是：打开复位开关，微处理器复位并停止执行机器代码；打开控制开关，就会接收总线上的地址信号和数据信号。在该状态下，你可以通过开关 A_0 ~ A_{15} 来指定一个 16 位的存储器地址；通过灯泡 D_0 ~ D_7 的明灭组合来显示该存储器地址中的 8 位数据。那么怎样把一个新的字节写入到此地址中呢？首先通过开关 D_0 ~ D_7 来设置想要写入的字节，然后把写入开关先打开再关闭。当你已经完成向存储器中插入字节的工作后，关上控制及复位开关，微处理器就会执行程序。

上面这个过程展示了向这台我们刚刚打造出来的计算机输入第一条机器代码的步骤。

不言而喻，这是一个耗时耗力的过程。在这个过程中一些小错误是在所难免的。看看你的手指，或许已经磨出了水泡，你的大脑也感觉一片混乱，而这一切都是工作的代价。

但当你开始用视频显示器显示程序运行的结果时，到底是什么使这一切都变得简单、方便呢？上一章中我们讲到只显示字符的视频显示器，它有一个 1 KB 的随机访问存储器，可以存储 25 行、每行 40 个字符的 ASCII 码。程序将要显示的内容写入到此存储器中，其方法与向计算机中其他存储器中写入数据的方法一样。

尽管把程序的输出结果显示在视频显示器上看似简单，实则不然。例如，你编写了一个程序用来完成某个计算任务，如果计算结果是 4Bh，不能将这个值直接写入到视频显示器的内存中。如果犯了这样的错误，屏幕上显示的将是字母 K，因为此字母的对应 ASCII 码的值正是 4Bh。4Bh 由两个字符组成，其中 4 对应的 ASCII 码是 34h， B 对应的 ASCII 码是 42h，应该将这两个 ASCII 码写到视频显示器存储器上，才能在显示器上看到期望的数值。这里再强调一下，8 位二进制数可以表示两位十六进制数字，因此必须将每一位十六进制数字对应的 ASCII 码，写入到视频显示器的存储器中才能显示这个数。

这种转换可以通过编写小的程序来实现。下面是一段 8080 汇编程序，功能是把存储在累加器中的十六进制数（假设这个数介于 00h 与 0Fh 之间）的每一位转换成对应的 ASCII 码：

```
NibbleToAscii: CPI A, 0Ah  ; 检查是数字还是字母
               JC Number
               ADD A, 37h  ; 把 A~F 转换成 41h~46h
               RET
Number:        ADD A, 30h  ; 把数字 0~9 转换成 30h~39h
               RET
```

通过两次调用 NibbleToAscii，下面的程序实现了把累加器 A 中的一个字节转换成两个 ASCII 码对应的数字，分别存放在寄存器 B 和 C 中。

```
ByteToAscii:   PUSH PSW                ; 保存累加器 A
               RRC
               RRC
               RRC
               RRC                     ; 获取高半字节
               CALL NibbleToAscii      ; 转换成 ASCII 码
               MOV B, A                ; 把结果存入寄存器 B 中
               POP PSW                 ; 取出原始的 A
               AND A, 0Fh              ; 获取低半字节
               CALL NibbleToAscii      ; 转换成 ASCII 码
```

```
MOV C, A                    ; 把结果存入寄存器 C 中
RET
```

通过这些程序，可以把一个用十六进制表示的字节显示在视频显示器上。进一步来讲，如果想把它转换成十进制数，还需要做些别的工作。这个过程与把一个数从十六进制转换成十进制的过程（用 10 除几次）十分相似。

记住，到此为止实际上你还没有将汇编语言程序写入到内存。你需要把汇编语言写到纸上，将它们转换成机器代码后才写入到内存中。直到第 24 章我们一直都会采用这种“手工汇编”的方式。

控制面板的确不需要很多硬件支持，但它不便于使用，因为它有着最糟糕的输入/输出形式。我们甚至可以从头开始独立建造一台计算机，但仍然无法改变这样糟糕的输入/输出方式——通过按键输入 0 和 1——这的确让人尴尬。如何把控制面板去掉是要解决的首要问题。

实现按键来控制输入/输出的不二之选就是键盘。前面我们已经搭建了一个计算机键盘，每次有按键按下的时候，就会产生一个中断信号送至微处理器。计算机内有中断控制芯片，通过执行一条 RST 指令使得微处理器响应这次中断，例如 RST 1，微处理器执行这条指令，把当前程序计数器的值压入到堆栈中，然后跳转到地址 0008h 处。可以直接在这片地址空间上输入一些代码（使用控制面板），这些代码称为键盘处理程序（keyboard handler）。

为了使复位后微处理器能正常工作，微处理器在复位的时候需要执行一些代码，称为初始化代码（initialization code）。堆栈指针在运行初始化代码的时候会被设置，以保证堆栈处在内存的有效区域内。为了不让屏幕上显示随机字符，初始化代码还把视频显示器内存中的每个字节设置成十六进制数 20h，在 ASCII 码中这是一个空格符。此外，初始化代码还要把光标定位在第一行第一列的位置——OUT（Output）指令可以完成这一操作：光标在视频显示器上是以下画线的形式出现的——它可以显示出下一个要输入字符的位置。为了使微处理器能响应键盘中断，必须设置 EI 指令开中断，而 HLT 指令可以使微处理器停止工作。

上面讲述的就是初始化代码的作用。执行了 HLT 指令后，计算机则处于停机状态。为了把计算机从停机状态唤醒，只能通过控制面板的复位信号或者键盘的中断信号来实现。

键盘处理程序的规模要远大于初始化代码，这个程序才是真正响应键盘事件的代码段。

任何时候，只要键盘上的一个按键被按下，微处理器就会响应本次中断，并从初始化代码末尾的 HLT 语句跳转到键盘处理程序。键盘处理程序利用 IN（Input）指令用来检查是哪个按键被按下，并根据这个按键执行相关的操作（就是说，键盘处理程序对每个按键进行相应的处理），然后执行 RET（Return）指令以返回 HLT 语句，等待另一个键盘中断。

当你按下字母、数字或者是标点符号键的时候，键盘扫描程序就会启用键盘扫描码，并根据 Shift 键是按下与否确定相应的 ASCII 码。接下来我们要做的，就是要把这个 ASCII 码写到视频显示器的内存中，当然这不是随意的，而是要写在光标所在的位置。这样的一个过程，我们可以很形象地称之为按键到显示器的回显（echoing）。光标会随着字符的写入而移动，换言之，它总会出现在刚显示的字符后面的空格处。通过键盘，可以输入一串字符，然后把它们在屏幕上显示出来。

当按下回退键（相应的 ASCII 码值是 08h）时，最后写入视频显示器内存中的字符会被键盘处理程序删除（其实并不复杂——我们只要把空格符对应的 ASCII 码——20h 写入到那个内存位置处就行了）。在这个过程中，光标会移回一格。

通常我们在输入一行字符时，错误是难免的，这时就需要用退格键来改正错误，然后按下回车键。回车键并不难找到，键盘上标有“Enter”字样的按键就是。打字员在电动打字机上按下“Return”键表示已经完成一行文字的输入，同时也表明他们已经做好了输入下一行的准备，光标会指向下一行的开始。同样，在计算机中“Enter”键用来实现相同的功能，结束一行的输入并转到下一行。

当键盘处理程序对“Return”或“Enter”键（对应的 ASCII 码是 0Dh）进行处理时，它把视频显示器内存中的这一行文本解释为计算机的一条命令（command），换言之，键盘处理程序的任务是执行此命令。实际上，在键盘处理程序内含有一个命令处理程序（command processor），它可以解释如下三条命令：W 命令、D 命令和 R 命令。下面我们来深入地理解一下它们。

首先是 W 命令。它是以 W 开头的文本行，此命令用来把若干字节写入（Write）到内存中。例如输入到屏幕上的一行内容如下所示：

```
W 1020 35 4F 78 23 9B AC 67
```

运行这条命令，命令处理程序会从内存地址 1020h 处开始，把 35、4F 等十六进制表示的字节写入内存中。要完成这项工作，键盘处理程序需要把 ASCII 码转换成字节——前面讲过把字节转换成 ASCII 码，这里其实就是它的逆变换。

接下来是 D 命令。它是以 D 开头的文本行，此命令用来把内存中的一些字节显示（Display）出来。例如输入到屏幕上的一行内容如下所示：

```
D 1030
```

接收到命令后，命令处理程序会把从地址 1030h 开始的 11 个字节的内容显示出来（这里之所以说是 11 个字节，是因为在一个每行可以容纳 40 个字符的显示器上，除去显示命令与地址标识，后面能显示的也只有这么多了）。有了这条命令，就可以很方便地查看内存中的内容了。

最后是 R 命令。它是以 R 开头的文本行，表示运行（Run）。该命令的形式如下：

```
R 1000
```

执行此命令意味着“处理器会运行从地址 1000h 开始的一段程序”。首先命令处理程序把 1000h 存储在寄存器对 HL 中，接着执行指令 PCHL，这条指令的功能是，把 HL 所存储的值加载到程序计数器中，然后跳转到程序计数器指向的地址并运行程序。

键盘处理程序及命令处理程序简化了很多工作，可以说是计算机发展的一个里程碑。一旦使用了它，就无须理会那个令人难以忍受的控制面板了。我们不得不承认，使用键盘输入更简单、更快、更好，这是其他方法无法媲美的。

但是，你仍然没有摆脱之前的老问题，一旦关掉电源，你辛辛苦苦所输入的数据会全部消失。当然，你可以把所有新代码存到只读存储器（ROM）中。还记得上一章中，我们讲到的那个 ROM 芯片吗？它就包含了把 ASCII 码字符显示到视频显示器上所需的全部点阵模式。这里假定，这些数据在厂家制造芯片时已经配置好了，当然你也可以对 ROM 芯片进行编程。可编程只读存储器（Programmable Read-Only Memory，PROM）只能编程一次；而可擦除可编程只读存储器（Erasable Programmable Read-Only Memory，EPROM）可重复擦除和写入，该芯片的正面开有一个玻璃窗口，让紫外线透过这个孔照射内部芯片就可以擦除其中的内容了。

回忆前面曾讲过的内容，你一定会想起，RAM 板与一个 DIP 开关相连，有了这个开

关就可以设定 RAM 板的起始地址了。8080 系统在初始化时，其中一个 RAM 板的起始地址被设置为 0000h。但是如果有 ROM 的话，这个地址就会被其占用，而 RAM 板则转到更高的地址。

命令处理程序的使用极大地推动了计算机的发展，通过它不仅可以更快地向存储器输入数据，更重要的是，计算机变得可交互（interactive）了。当你通过键盘输入一些内容，计算机会立即做出响应，把你所输入的内容显示出来。

将命令处理程序存储到 ROM 后，就可以执行操作了：把内存中的数据写入到磁盘驱动器（可能是按照与磁盘的扇区大小一致的块的形式），然后再把数据读回到内存中。因为掉电的时候，RAM 中的内容会丢失，所以与 RAM 相比，把程序和数据存储到磁盘中会更安全（它们不会因为突然断电而丢失数据），就灵活性来说，也比存储到 ROM 中要好。

仅仅有上面的命令还是不够的，还需要向命令处理程序中添加新命令。例如，S 命令表示存储（Store）：

```
S  2080  2  15  3
```

运行这条命令后，在磁盘的第 2 面、第 15 道、第 3 扇区中将存放起始地址为 2080h 的内存块数据（被存放内存块的大小是由磁盘扇区的大小决定的）。类似地，还可以通过加载（Load）命令，把磁盘上相应扇区的内容写回到内存中，如下所示：

```
L  2080  2  15  3
```

当然，还要把存储的位置记录下来，这是必须要做的。你可以用手头上的纸和笔来完成。需要注意的是：你不能把位于某个地址处的代码又加载到内存的另外一个地址中，如果这样的话，程序将不能正常运行。具体来说，程序代码在内存中改变位置后，其跳转（Jump）和调用（Call）指令标识的依然是原来的地址，所以运行时会报错。也存在这样的情况，程序比磁盘的扇区大，这时就需要多个扇区来存放程序。而磁盘上某些扇区已被其他程序或数据占用了，而另外一些扇区是空闲的，可能在磁盘上找不到一块足够大的、连续的扇区来存放程序。

最后，所有的东西存储在磁盘的什么位置，都需要你手工地记录下来，这个工作挺多，也挺麻烦。出于这个原因，文件系统（file system）应运而生。

文件系统是磁盘存储的一种方法，就是把数据组织成文件（file）。简单地说，文件是

相关数据的集合，占用磁盘上一个或多个扇区。更重要的是，你可以为每个文件命名，这有助于记下文件里存放的内容。想象一下，磁盘是不是类似于一个文件柜，每个文件都有个小标签，标签上有文件的名称。

操作系统（operating system）是许多软件构成的庞大程序集合，文件系统就是其中的一部分。前面讲到的键盘处理程序和命令处理程序最终也能经过拓展，演变成为操作系统。那么操作系统到底能够做些什么、又是如何工作的呢？这里撇开操作系统漫长的发展演化过程，把重点放在刚才提出的问题上，目的就是试图让大家了解操作系统的工作机制。

如果你对操作系统的发展史有一定了解的话，你就会知道 CP/M（Control Program for Micros）是最重要的 8 位微处理器操作系统，它是 20 世纪 70 年代中期由加里·基尔代尔（Gary Kildall，生于 1942 年）专门为 Intel 8080 微处理器而开发的，加里后来成了 DRI（Digital Research Incorporated）公司的创始人。

CP/M 操作系统是存放在磁盘上的。单面、8 英寸的磁盘是早期的 CP/M 最常用的存储介质，它有 77 个磁道，每个磁道有 26 个扇区，每个扇区的大小是 128 个字节（总共算下来共有 256,256 个字节），CP/M 系统存放在磁盘最开始的两个磁道。在启动计算机时，需要把 CP/M 从磁盘调入到计算机的内存中，下面将介绍这一过程是如何进行的。

上面我们讲过，存放 CP/M 本身只占用 2 个磁道，那么剩下的 75 个磁道用来存储文件。CP/M 的文件系统固然简单，但两个最基本的要求还是可以满足的。首先，每个文件在磁盘中都有属于自己的名字，便于识别，这个名字也是存放在磁盘中的；实际上，文件以及读取这些文件需要的所有信息也是一起存储在磁盘中的。其次，文件存储在磁盘中不一定要占据连续的扇区空间，可以想象，由于大小不同的文件会被经常地创建和删除，磁盘上的可用空间就会很零碎。那么如何将文件存储在零碎的空间里呢？这主要得益于文件系统具有很强大的管理功能，它可以把一个大文件分散存储在不连续的磁盘上。

上面曾提到过剩下的 75 个磁道中的扇区用来存放文件，这些扇区按分配块（allocation blocks）进行分组。每个分配块中有 8 个扇区，总计 1024 个字节（每个扇区大小是 128 字节）。可以计算，在磁盘上共有 243 个分配块，编号为 0～242。

目录（directory）区占用最开始的两个分配块（编号为 0 和 1，总共 2048 字节）中。

目录区是磁盘中的一个非常重要的区域，磁盘文件中每个文件的名字和其他一些重要信息都存储在该区域，根据目录就能够很方便、高效地查找文件。存放目录也需要占用空间，磁盘上每个文件对应的目录项（directory entry）大小均为32字节，由于目录区大小为2048字节，所以这个磁盘上最多可以存放64（2048/32）个文件。

为了能够根据目录找到相应的文件，每一个32字节的目录项包含以下信息。

字　节	含　义
0	通常设为0
1～8	文件名
9～11	文件类型
12	文件扩展
13～14	保留（设置为0）
15	最后一块扇区数
16～31	磁盘存储表

在目录项中，第一个字节用来设置文件的共享属性，只有文件系统被两个或更多人同时共享时才设置此字节为1。在CP/M中，这个字节跟第13、14字节一样，通常设置为0。

CP/M中每个文件的文件名由两部分构成，第一部分称为文件名（filename），文件名最多由8个字符构成，目录项的第1～8字节用来存储文件名。第二部分称为文件类型（file type），最多由3个字符表示，这些字符存储在目录项的第9～11字节中。我们会经常遇到一些常见的标准文件类型，例如：TXT表示文本文件（此文件只包含ASCII码）；COM（command的简写）表示这个文件中存放的是8080机器码指令，或者说是一段程序。当命名一个文件时，通过一个点来隔开这两部分，如下所示：

MYLETTER.TXT

CALC.COM

人们习惯形象地称这种文件命名方式为8.3，就是说在点号隔开的前半部分最多有8个字母，后半部分最多有3个字母。

接下来介绍一下如何利用目录项查找文件。目录项中的第16～31字节是磁盘存储表，它能够标明文件所存放的分配块。假设磁盘存储表的前4项分别为14h、15h、07h、23h，

其余项全为 0，这表明文件占用了 4 个分配块的空间，大小为 4 KB，而实际上文件可能并没有用完 4 KB 空间，因为最后一个分配块往往只有部分扇区被使用。目录项的第 15 字节表明最后一个分配块到底用到了多少个 128 字节的扇区。

磁盘存储表的长度为 16 字节，最多可以容纳 16,384 字节（16 KB）的文件，如果文件的长度超过 16 KB，就需要用多个目录项来表示，这种方法称为扩展（extents）。如果一个大文件使用目录项扩展来，则将第一个目录项的第 12 字节设置为 0，第二个目录项的第 12 字节则设置为 1，依此类推。

文本文件对我们并不陌生，通常也称之为 ASCII 文件（ASCII file）、纯文本文件（text-only file）或纯 ASCII 文件（pure-ASCII file），当然还有其他一些类似的名称。ASCII 码（包括换行符和回车符）是文本文件中唯一包含的字符，文本文件通俗易懂，便于浏览。除了文本文件外，其余的文件称为二进制文件（binary file），在 CM/P 中，COM 文件存放的是二进制的 8080 机器码，它是二进制文件。

假设一个小文件中包含三个 16 位数，如：5A48h、78BFh 和 F510h。如果此文件是二进制文件，只要 6 字节就可以了。

```
48  5A  BF  78  10  F5
```

这是采用 Intel 格式来存储的，放在前面的是低位，放在后面的是高位。并不是所有的数据都是按照这种格式来存储的，例如为 Motorola 处理器编写的程序更倾向于按以下的方式来组织文件：

```
5A  48  78  BF  F5  10
```

假如上面的三个 16 位数用 ASCII 文本文件来存放，则文件中保存的数据如下所示：

```
35 41 34 38 68 0D 0A 37 38 42 46 68 0D 0A 46 35 31 30 68 0D 0A
```

上面的这些字节是数字和字母的 ASCII 码表示形式，用回车符（0Dh）和换行符（0Ah）来表示每一个数的结束。因为文本文件可以不通过解释相应的 ASCII 字节串来显示字符，而是将字符本身直接显示，所以显得更加方便，如下所示：

```
5A48h
78BFh
F510h
```

也可以用如下的形式来表示包含这三个数的 ASCII 文本文件：

```
32 33 31 31 32 0D 0A 33 30 39 3 31 0D 0A 36 32 37 33 36 0D 0A
```

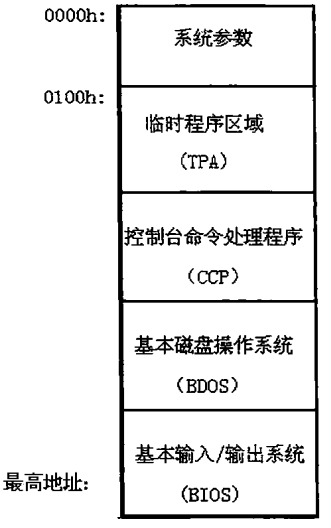
这是用十进制数的 ASCII 码形式来表示上述三个数，这两种表示形式是等价的，如下所示：

```
23112
30911
62736
```

显然，文本文件更易于人们阅读，同样，与十六进制相比，十进制更符合人们的习惯，没理由使用十六进制而拒绝十进制。

前面曾提到过，磁盘最开始的两个磁道存储 CP/M 系统本身，而 CP/M 在磁盘上是无法运行的，必须将其加载到内存里。只读存储器（ROM）在 CP/M 计算机中使用得并不多，只需要用它来存放一小段称为引导程序（bootstrap loader，操作系统的其余部分可以通这段代码的自举操作被高效地引导）的代码即可。开机启动时，磁盘上最开始的 128 字节的扇区内容，会首先由引导程序加载到内存并运行，这个扇区包含有特定的代码，可以把 CP/M 中的其余部分加载到内存中，整个过程称为操作系统的引导（booting）。

操作系统的引导过程完成后，随机存储器（RAM）的最高地址区域用来存放 CP/M，加载完 CP/M 后，整个内存空间的组织结构如下所示。



该图仅仅粗略地表示出了内存各构成部分，没有按比例刻画各部分所占内存的大小。控制台命令处理程序（Console Command Processor，CCP）、基本磁盘操作系统（Basic Disk Operating System，BDOS）和基本输入/输出系统（Basic Input/Output System，BIOS）是CP/M的三个组成部分，这三个部分只占用了6 KB大小的内存空间。在拥有64 KB内存空间的计算机中，大约58 KB被临时程序区（Transient Program Area，TPA）占用，但是这58 KB空间一开始时是空的。

控制台命令处理程序的功能和以前讨论过的命令处理程序是一样的。在这里，键盘和显示器组成了控制台（console）。控制台命令处理程序显示如下所示的命令提示符（prompt）：

```
A>
```

在命令提示符后面可以输入一些信息。大多数计算机可能有不止一个磁盘驱动器，第一个磁盘驱动器标为A，CP/M被装载到该驱动器中。在命令提示符后面敲入命令并按回车（Enter）键，控制台命令处理程序会执行该命令，然后将执行的结果显示到屏幕上。执行完命令后，命令提示符又会显示在屏幕上，等待下一次输入。

控制台命令处理程序只能识别一部分命令，其中最重要的命令是：

```
DIR
```

该命令用于显示磁盘的目录信息，换句话说，它列出了存储在磁盘上的所有文件。然而有时候需要查看具有特定名字和类型的文件，这时候就可以在命令中使用像“？”和“*”这样的特殊字符来限定。如果想显示当前目录下所有的文本文件，可以使用如下指令：

```
DIR  *.TXT
```

而

```
DIR  A???B.*
```

则显示所有文件名由5个字符构成，其中第一个字符是A，最后一个字符是B的文件。

如果想删除磁盘中的文件，要用到命令ERA，它是Erase的缩写，例如：

```
ERA  MYLETTER.TXT
```