CN43-1258/TP
ISSN 1007-130X

计算机工程与科学
COMPUTER ENGINEERING & SCIENCE

2006 年第 28 卷第 1 期
Vol. 28, No. 1, 2006

文章编号:1007-130X(2006)01-0107-03

引导扇区剖析与实例分析*
A Dissection of the Boot Sector

蔡山枫,周凤岐
CAI Shan-feng, ZHOU Feng-qi
(西北工业大学航天学院,陕西 西安 710072)
(School of Aeronautics, Northwestern Polytechnical University, Xi'an 710072, China)

摘 要:引导扇区实现了由裸机到操作系统的过渡。本文将介绍与引导扇区相关的软硬件知识,并提供了分析引导扇区的方法,最后以硬盘的 MBR 扇区为实例,剖析引导代码,解读引导过程。分析引导扇区从另一个角度提供了学习和实践计算机硬件和操作系统相关知识的途径。

Abstract: It is the boot sector that introduces OS into a bare computer. An analysis of the boot sector is provided in this article, and a dissection of the MBR bootstrap code is given. Dissecting the boot sector provides another means to studying the knowledge of both hardware and OS.

关键词:引导扇区;主引导记录(MBR);卷引导记录(VBR)
Key words: boot sector; MBR(Master Boot Record); VBR(Volume Boot Record)
中图分类号:TP316　　**文献标识码**:A

1 引言

扇区是磁盘(软盘或硬盘)访问的最小单位,通常包含 512 个字节,一些特殊位置处的扇区被写入引导代码,成为引导扇区。在计算机启动过程中,引导扇区被装入内存,并获得控制权,操作系统才得以装入。

引导扇区的主要功能是查找操作系统内核文件在磁盘上的扇区位置,将其读入内存,并移交控制权。由于分区的原因,硬盘有两种引导扇区:一是主引导扇区,又名 MBR(Master Boot Record,简称 MBR)扇区,通常是硬盘的物理第一个扇区;二是卷引导扇区,硬盘的每个分区都有卷引导扇区,又名 VBR(Volume Boot Record,简称 VBR)扇区,其位置由分区的大小和类型决定。

分区表位于 MBR 扇区,它记录了硬盘的分区情况。MBR 引导代码从分区表中找到活动分区(即可引导分区)的分区项,利用其中记录的扇区参数,将该分区的 VBR 扇区装入内存,并移交控制权。

在具体分析之前,有必要介绍一下相关的软硬件知识。

2 关于扇区

1.44M 软盘有 2 880 个扇区,而硬盘的扇区个数随容量的不同而不同,1G 的存储空间需要 2M 余个扇区。必须用一定的参数来指定扇区的位置,对于引导代码,可用的扇区参数有两种:CHS(Cylinder/Head/Sector,简称 CHS)参数和 LBA(Logical Block Address,简称 LBA)参数。

CHS 参数通过柱面(Cylinder)、磁头(Head)和扇区(Sector)这三个磁盘几何参数来指定扇区的位置。磁盘由若干个盘片(Platter)组成,盘片的两面各对应一个磁头;盘片被划分为若干圈磁道,不同盘片上处于同一位置的磁道,统称为柱面;磁道又由若干个扇区组成。比如,1.44M 软盘有 80 个柱面、2 个磁头,每个磁道有 18 个扇区。

引导代码通过调用 BIOS 中断函数[1]访问扇区。0x13 号 BIOS 中断的 0x02 号(CHS 读扇区)和 0x03 号(CHS 写扇区)以及 0x08 号(获取磁盘几何参数)子功能就提供了这样的接口。在调用时,将各个参数装入规定的寄存器,利用汇编指令[2]"INT 13H"就可完成相应的操作。由于寄存器位数的限制,CHS 扇区参数最多能访问约 7.8G 的磁盘空间[3]。

许多版本的 BIOS 对 0x13 号中断进行了扩充,提供了利用 LBA 参数访问扇区的函数接口:0x41 号(LBA 检测)、0x42 号(LBA 读扇区)和 0x43 号(LBA 写扇区)。LBA 参数用一个 32 位二进制数表示扇区的位置,能够访问 2

* 收稿日期:2004-09-23;修订日期:2004-11-19
作者简介:蔡山枫 (1979-),男,陕西岐山人,硕士生,研究方向为先进控制理论研究与应用;周凤岐,教授,博士生导师,研究方向为导航、制导与控制。
通讯地址:710072 陕西省西安市西北工业大学 250#信箱;Tel:13772132637;E-mail:BeannieCai@163.com
Address: School of Aeronautics, Northwestern Polytechnical University, Xi'an, Shaanxi 710072, P. R. China

000G左右的硬盘。在引导代码中，扇区的位置都是用LBA参数进行计算的，如果需要CHS参数，再进行转化。

软盘的引导扇区和硬盘的MBR扇区的CHS参数都是0/0/1，LBA参数为0，而VBR扇区的扇区参数只能从分区表中获得。

3 分析方法

为了分析方便，最好将引导扇区备份成磁盘文件。这里向大家推荐使用一个16位代码的命令行调试工具Debug。各个版本的DOS、Windows操作系统都带有Debug.exe程序。除了调试功能、文件操作功能外，Debug还可以访问VBR扇区，包括软盘的引导扇区。此外，Debug具有16位汇编代码编译功能，通过编写短小的汇编程序，调用0x13中断，也可以方便地读取硬盘的MBR扇区。

当计算机加电后，CPU工作在实模式方式下，仅有1M内存可供使用。ROM引导程序把引导扇区装入内存0x07C00处，然后CS和IP分别被设置为0x0000和0x7C00，这样引导扇区就获得了控制权。在1M内存空间中，0x00000～0x003FF存放中断向量表，0x00400～0x004FF存放BIOS数据，剩下的常规内存0x00500～0x09FFF可以由引导代码自由支配。

利用Debug的装入命令(L)将引导扇区装入内存，然后利用单步命令(T)或运行命令(G)可以对引导代码进行动态分析。在实际的引导过程中，引导代码的内存基地址是0x0000:0x7C00；然而，在用Debug调试时，绝不能使用这段内存，这是因为这段内存已经被DOS占用。除了0x00000～0x004FF(1.25K)的内存外，DOS又要占用几十K的内存空间，如果这些内存被随意修改，DOS将崩溃。一般来说，0x50000(320K)以后的常规内存总是空闲的。在下面的例子中，软盘的引导扇区被装入到内存0x5000:0x7C00处：

```
D:\>debug
-L 5000:7C00 0 0 1
-U 5000:7C00 1 3
5000:7C00  EB3C          JMP     7C3E
5000:7C02  90            NOP
-U 5000:7C3E
5000:7C3E  33C9          XOR     CX,CX
5000:7C40  8ED1          MOV     SS,CX
5000:7C42  BCFC7B        MOV     SP,7BFC
5000:7C45  16            PUSH    SS
5000:7C46  07            POP     ES
5000:7C47  BD7800        MOV     BP,0078
5000:7C4A  C57600        LDS     SI,[BP+00]
5000:7C4D  1E            PUSH    DS
5000:7C4E  56            PUSH    SI
5000:7C4F  16            PUSH    SS
5000:7C50  55            PUSH    BP
5000:7C51  BF2205        MOV     DI,0522
5000:7C54  897E00        MOV     [BP+00],DI
5000:7C57  894E02        MOV     [BP+02],CX
5000:7C5A  B10B          MOV     CL,0B
5000:7C5C  FC            CLD
5000:7C5D  F3            REPZ
5000:7C5E  A4            MOVSB
-
```

引导扇区被装入内存后，利用反汇编命令(U)，可以得到引导代码对应的汇编指令。由于段值不匹配，在动态分析代码时，需要利用"R"命令手动修改一些寄存器。例如，上面例子中跳至0x5000:0x7C3E处的代码将栈顶设置在0x0000:0x7BFC处，而此时的段址是0x5000，因此必须手动修改SS。后面的ES、DS也要做类似的修改。

Debug只能反汇编16位的8086指令，对于32位指令，Debug无法识别，可以利用其它反汇编调试工具，如Microsoft的CodeView或Borland的Turbo Debugger。

4 实例分析

下面以硬盘的MBR扇区为例，分析其中的引导代码：

```
0000:7C00 33C0          XOR     AX,AX
0000:7C02 8ED0          MOV     SS,AX
0000:7C04 BC007C        MOV     SP,7C00H
    ;栈顶设置在0x0000:0x7C00处。
0000:7C07 FB            STI             ;允许中断。
0000:7C08 50            PUSH    AX
0000:7C09 07            POP     ES      ;ES = 0。
0000:7C0A 50            PUSH    AX
0000:7C0B 1F            POP     DS      ;DS = 0。
```

在分析过程中，仅列出具有代表性的代码。为便于理解，代码中的一些变量和地址用表意的名称来代替。

任何引导代码在获得控制权后，首先必须做的工作就是设置堆栈及段寄存器。

然后，MBR引导代码"自举"到新的地址空间，腾出0x0000:0x7C00处的内存，预留给以后将被读入的VBR扇区，代码如下：

```
0000:7C0C FC        CLD
    ;设置字符串操作方向为递增。
0000:7C0D BE1B7C  MOV   SI, 7C1BH
    ;复制源地址，即跳转指令(RETF)后的代码地址
    ;0x7C1B。
0000:7C10 BF1B06  MOV   DI, 061BH
    ;这是复制的目的地址。
0000:7C13 50        PUSH   AX
    ;跳转地址的段址，此前AX=0。
0000:7C14 57        PUSH   DI
    ;跳转地址的偏移。
0000:7C15 B9E501  MOV   CX, 01E5H
    ;复制的字节数=扇区大小-已运行的代码大小，
    ;即0x200(512) -0x01B = 0x01E5。
0000:7C18 F3A4      REPZ MOVSB
    ;将DS:SI处的代码复制到ES:DI处。
0000:7C1A CB        RETF
    ;RETF指令从栈中弹处两个字，分别装入IP和CS，
    ;从而实现了长跳转。
```

接下来扫描分区表，寻找活动分区。分区表位于MBR扇区的第446(0x1BE)字节处，占用64个字节，共包含四个分区项，每个分区项纪录一个分区的信息。分区项的定义如下：

```
00  ActiveTag   DB 0或80H
    ;分区标志字节，0x80表示可引导分区，否则只能为0x00。
01  StartH      DB ?
    ;分区起始扇区(即VBR扇区)的Head参数。
02  StartCS     DW ?
    ;分区起始扇区(即VBR扇区)的Cylinder参数(10位)
    ;和Sector参数(6位)。
04  FSID        DB ?
    ;分区文件系统标志字节，常见的FSID有：
    ;0x0B—FAT32分区(采用CHS扇区参数)
    ;0x0C—FAT32分区(采用LBA扇区参数)
    ;0x0E—VFAT分区(同FAT32，采用LBA扇区参数)
    ;0x0F—扩展分区(一般采用LBA参数)
05  EndH        DB ?
    ;分区结束扇区的Head参数。
06  EndCS       DW ?
    ;分区结束扇区的Cylinder参数和Sector参数。
08  PreSecNum  DD ?
    ;分区起始扇区(即VBR扇区)的LBA参数。
0C  SectorNum  DD?
```

;分区包含的扇区总数。

利用 Debug 可以察看分区表，如图 1 所示。

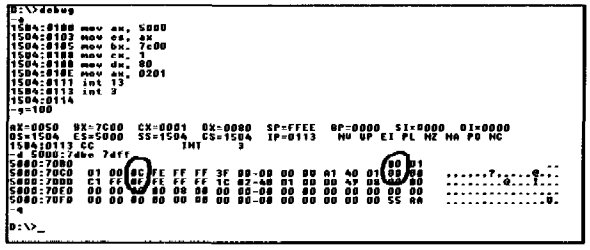

图 1　用 Debug 察看的分区表

通过调用 0x13 号中断的 0x02 号子功能，将 MBR 扇区装入 0x5000:0x7C00 处，那么分区表就位于 0x5000:0x7DBE。可以看到，这块硬盘只有两个分区。第一个分区是可引导分区(0x80)，采用 FAT32 文件系统(0x0C)，VBR 扇区的 LBA 参数是 63(0x0000003F)，CHS 参数是 1/0/1。第二个分区是扩展分区(0x0F)。

当 MBR 引导代码“自举”到 0x0000:0x061B 处后，分区表也被复制至 0x0000:0x07BE 处。接着，引导代码将依次检索分别位于 0x07BE、0x07CE、0x07DE、0x07EE 处的各个分区项的 ActiveTag，以确定活动分区，代码如下：

```
    0000:061B BEBE07   MOV   SI, 07BEH
        ;SI = 分区表地址。
    0000:061E B104     MOV   CL, 04H
        ;分区表包含 4 项分区记录，即总循环次数。
IsActive:
    0000:0620 382C     CMP   [SI], CH
        ;此时 CH 等于 0，将 ActiveTag 与 0 比较。
    0000:0622 7C09     JL    ActivePartFound
        ;找到活动分区，跳转。
    0000:0624 7515     JNZ   InvalidPartTbl
        ;若 ActiveTag 又不为 0，则分区表无效，跳转。
    0000:0626 83C610   ADD   SI, 10H
        ;调整 SI，指向下一个分区项的 ActiveTag。
    0000:0629 E2F5     LOOP  IsActive
        ;循环，察看下一个分区。
    0000:062B CD18     INT   18H
        ;如果所有的 ActiveTag 均为 0，表示没有活动分区，
        ;调用 0x18 号 BIOS 中断。通常这个中断给出屏幕提示
        ;后，将重新启动计算机。
ActivePartFound:
    0000:062D 8B14     MOV   DX, [SI]
        ;将 0x80 和 VBR 扇区的 Head 参数分别装入 DL
        ;和 DH，以便后面的 0x13 号 BIOS 中断调用。
    0000:062F 8BEE     MOV   BP, SI
        ;将分区项的地址保存在 BP，后面对该分区项的访问，
        ;都是以 BP 作为基准的。
```

如果分区表中含有无效项，引导代码会跳转到 InvalidPartTbl 处，然后利用 0x10 号视频中断给出屏幕提示信息，如“Invalid partition table”等。

通常情况下，硬盘总有一个活动分区。引导代码找到活动分区的分区项后，将 BP 设置为该活动分区项的偏移地址，然后察看[BP+4]处的文件系统标识字节 FSID，并按照如图 2 所示的流程读取 VBR 扇区。

VBR 扇区被装入内存后，MBR 引导代码还要检查扇区标志字 0xAA55，代码如下：

```
    0000:06AB 813EFE7D55AA   CMP   WORD PTR [7DFE],
        AA55H
        ;由于 VBR 扇区被装入 0x0000:0x7C00 处，故扇区标志
        字位于 0x0000:0x7DFE
    0000:06B1 745A   JZ   AA55Exist
        ;如果标志字存在，说明是引导扇区，跳转。
AA55Exist:
    0000:070D EB74   JMP   GoToVBR
```

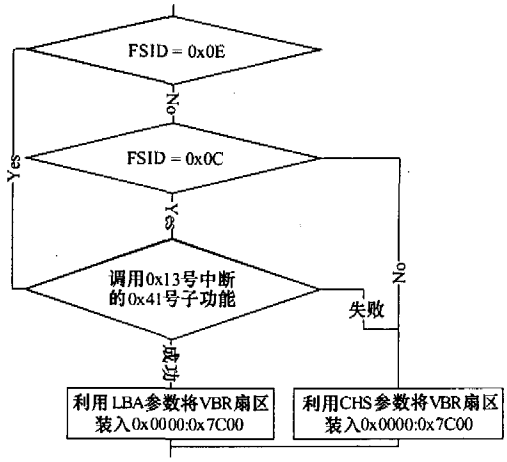

图 2　读取 VBR 扇区的流程

```
GoToVBR:
    0000:0783 8BFC   MOV   DI,SP
        ;SP = 0x7C00，故 DI = 0x7C00。
    0000:0785 1E   PUSH   DS          ;将段址 0x0000 压栈。
    0000:0786 57   PUSH   DI          ;将偏移 0x7C00 压栈。
    0000:0787 8BF5   MOV   SI,BP
        ;将 BP 指向的 Active 分区项偏移地址赋给 SI。
    0000:0789 CB   RETF               ;跳至 0x0000:0x7C00
```

至此，MBR 引导代码就将控制权移交给 VBR 扇区的引导代码，MBR 引导过程结束。

5　结束语

本文介绍了分析引导扇区的方法，并剖析了 MBR 引导扇区的引导过程。文中略去了某些中断函数调用的具体代码，感兴趣的读者可以按照本文提供的方法，结合参考文献自行分析。如果要分析 VBR 扇区(包括软盘的引导扇区)，还需掌握文件系统方面[2,3]的知识。总之，解读引导扇区，了解引导过程，从另一个角度提供了学习和实践计算机硬件以及操作系统相关知识的途径。

参考文献：

[1] Ralf Brown's Interrupt Jump Table[EB/OL]. http://www.ctyme.com/intr/int.htm,2004-08.

[2] Barry B Brey. 金惠华，曹庆华，李雅倩译. 80x86、奔腾机汇编语言程序设计[M]. 北京：电子工业出版社，1998.

[3] Scott Mueller. 前导工作室译. PC 升级与维护大全(第 12 版)[M]. 北京：机械工业出版社，2001.

[4] Dobiash Realms - FAT32 Structure Information[EB/OL]. http://home.teleport.com/~brainy,2004-08.

[5] NTFS.com NTFS File System General Information[EB/OL]. http://www.ntfs.com,2004-08.