目 录

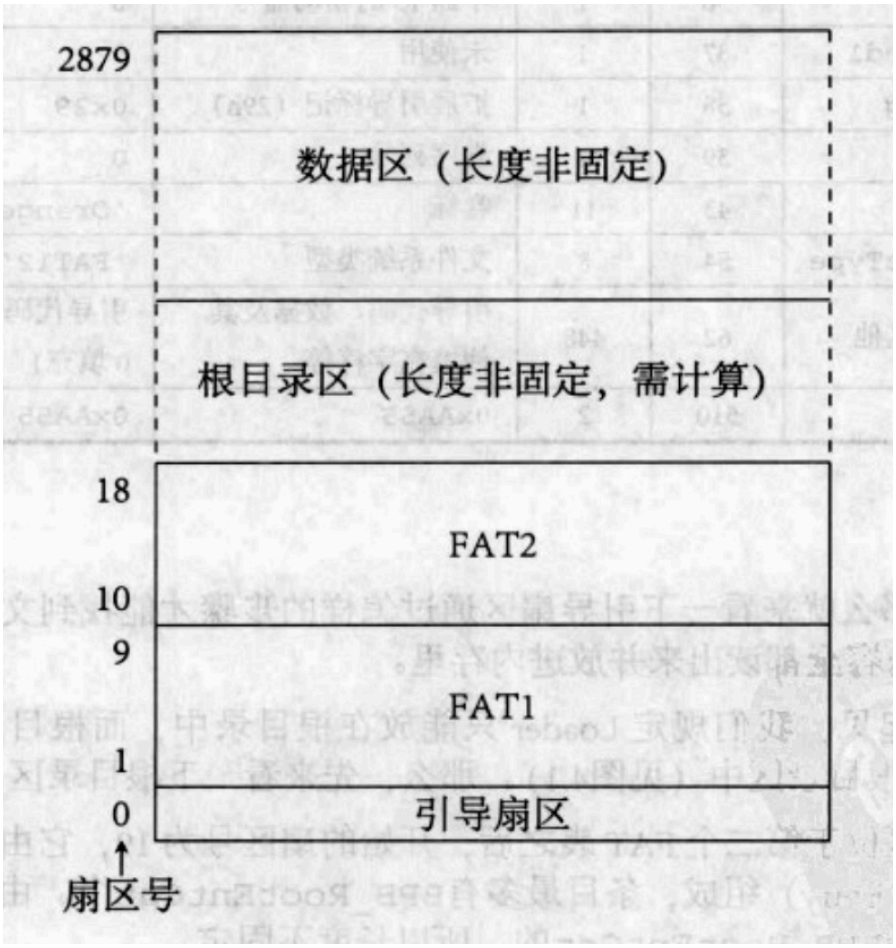
1 加载 Loader 入内存 2
1.1 FAT12 格式 2
1.1.1 根目录区条目 3
1.1.2 FAT 表 3
1.2 读软盘函数 4
1.3 寻找 Loader 5
1.4 计算 FAT 项的值 7
1.5 加载 Loader 入内存 8

1 加载 Loader 入内存

如果要加载一个 Loader 入内存，就需要读软盘，所以首先了解一下软盘的结构。

1.1 FAT12 格式

1.44MB 软盘上使用的是 FAT12 文件系统，软盘由引导扇区、FAT1、FAT2、根目录区和数据区组成。如下图所示：



我们想将 Loader 读入内存，首先要知道 Loader 在软盘中的位置。

Loader 一般存放在数据区中，而数据区信息存放在 FAT2 后面的根目录区中。

1.1.1 根目录区条目

根目录区开始的扇区号为 19，由若干个目录条目组成。每一个条目占用 32 字节，结构如下图所示：

名称	偏移	长度	描述
DIR_Name	0	0xB	文件名 8 字节，扩展名 3 字节
DIR_Attr	0xB	1	文件属性
保留位	0xC	10	保留位
DIR_WrtTime	0x16	2	最后一次写入时间
DIR_WrtDate	0x18	2	最后一次写入日期
DIR_FstClus	0x1A	2	此条目对应的开始簇号
DIR_FileSize	0x1C	4	文件大小

由上图可以看出，每个条目定义了文件的名称、属性大小、日期以及此条目对应的数据区中的开始簇号。因为一个簇号只包含一个扇区，所以条目记录着对应的扇区号。需要注意的是，数据区的第一个簇的簇号是 2。如果 DIR_FstClus 为 2，说明该文件的数据位于数据区的第一个簇。

随后还要算出根目录区所占的扇区数。一个扇区为 512 字节，一个条目为 32 字节，只要先算出根目录区总字节数，再除以 512 字节，就能得到根目录区所占的扇区数。如果不能整除，需要将扇区数加 1。

直到数据区开始扇区号，又知道文件在数据区中的开始簇号，就能得到总的扇区号。

1.1.2 FAT 表

文件有时候会大于 512 字节，所以会占用超过 512 字节，所以我们需要 FAT 表来找到所有的簇。

FAT 表有两个，FAT2 是 FAT1 表的备份。FAT1 的开始扇区号是 1，由 FAT 项组成。每个 FAT 项占 12 位，包含一个字节和另一个字节的一半。

当根目录区中条目记录的是数据区中开始的第 3 个簇，那么就对应第 3 个 FAT 项。如果这个 FAT 项值不为 0xFFF 而是 0x008，那么说明文件还未结束，文件的下一部分内容在数据区的第 8 个簇中。同时我们需要再找到第 8 个 FAT 项，依次查找下去，直到 FAT 项的值为 0xFFF，则代表着文件结束。

需要注意的是，在实际编写代码的过程中，要考虑一个 FAT 项跨越两个扇区的情况。

1.2 读软盘函数

如果要加载一个文件入内存，就需要读软盘，此时就需要用到 BIOS 中断 int 13h，它的用法如下所示：

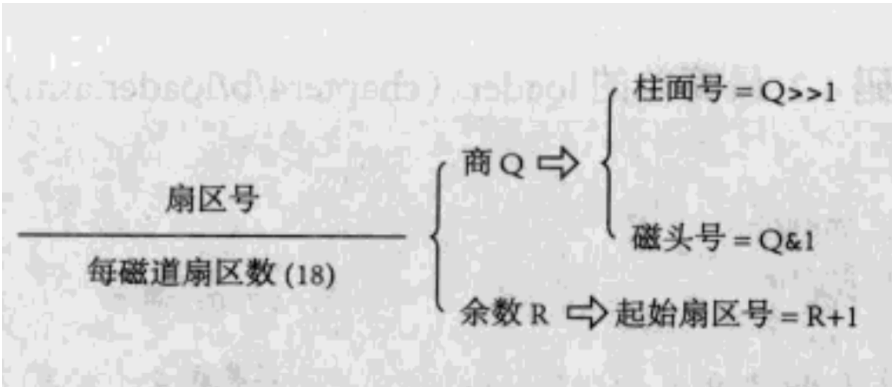
表 4.4 BIOS 中断 int 13h 的用法

中断号	寄存器		作用
13h	ah=00h	dl=驱动器号（0 表示 A 盘）	复位软驱
	ah=02h	al=要读扇区数	从磁盘将数据读入 es:bx 指向的缓冲区中
	ch=柱面（磁道）号	cl=起始扇区号	
	dh=磁头号	dl=驱动器号（0 表示 A 盘）	
	es:bx→数据缓冲区		

从上图可以看到，int 13h 需要用 al 指明要读的扇区数，ch 指定磁道号，cl 指定起始扇区号，dh 指定磁头号。然后用 es:bx 指定数据缓冲区。

对于 1.44MB 软盘而言，一共有两个磁头号 (磁头号 0 和 1)，每一面有 80 个磁道 (0 ～ 79)，每个磁道有 18 个扇区 (1 ～ 18)。需要注意的是，软盘两面的磁道号是交错排序的，磁头号为 0 的磁道号为偶数，磁头号为 1 的磁道号为奇数。

根据上面的已知条件，当我们拥有 Loader 所在的扇区号时，就可以算出磁头号、磁道号和起始扇区号。因为一个磁道有 18 个扇区，所以扇区号除以 18 就得到它所在的磁道号，同时余数就是起始扇区号。需要注意的是，因为起始扇区号是从 1 开始的，所以还需要加 1。而得到的磁道号也需要处理，因为这个磁道号是软盘的磁道号，不是软盘某一面上的磁道号。通过判断磁道号的奇偶就能得到磁头号，然后将总体的磁道号除以 2，就能得到软盘某一面上的磁道号。计算方法如下图所示：



根据上述的描述，我们就可以写一个读取软盘的函数了。函数使用堆栈暂时保存数据，所以还会建立一个堆栈。函数使用 ax 传递要读取的扇区号，使用 cl 传递要读取的扇区数。代码如下所示：

```
1    BaseOfStack  equ  07c00h  ; 相当于0～07c00h都可以当作堆栈
```

```
2
3       BPB_SecPerTrk DB 18
4
5       ; ...
6           ; 初始化堆栈
7           mov ax, cs
8           mov ds, ax
9           mov es, ax
10          mov ss, ax
11          mov sp, BaseOfStack
12
13      ReadSector:
14          push bp
15          mov bp, sp
16          ; 这里想用堆栈暂时保存要读取的扇区数
17          ; 但是因为push和pop默认处理的是字单元
18          ; 所以需要通过mov指令来另外处理
19          sub esp, 2
20          mov byte [bp-2], cl ; cl存放着要读取的扇区数
21          push bx             ; 保存bx
22
23          mov bl, [BPB_SecPerTrk]
24          div bl              ; ax存放着要读取的扇区号，bl为18，指令执行后ah存放余
                数，al存放商
25          inc ah              ; 得到起始扇区号
26          mov cl, ah          ; cl指定起始扇区号
27          mov dh, al
28          and dh, 1           ; 得到磁头号
29          shr al, 1           ; 得到扇区号
30          mov ch, al
31
32          pop                 ; 恢复bx
33          mov dl, [BS_DrvNum] ; 指定驱动器号
34      .GoOnReading:
35          mov ah, 2
36          mov al, byte [bp-2]
37          int 13h
38          jc .GoOnReading
39
40          add esp, 2
41          pop bp
42
43          ret
```

1.3 寻找 Loader

寻找 Loader 只需要遍历根目录区中的所有扇区，然后将每一个扇区加载入内存。然后从中寻找文件名为 Loader.bin 的条目，直到找到为止。代码如下：

```
1       BaseOfLoader equ 09000h
2       OffsetOfLoader equ 0100h
3       RootDirSectors equ 14               ; 根目录区占用的扇区数
4       SectorNoOfRootDirectory equ 19      ; 根目录区的开始扇区号
```

```
5
6       ; 首先复位软驱，通过int 13h中断实现
7       xor ah, ah
8       xor dl, dl
9       int 13h
10
11      mov word [wSectorNo], SectorNoOfRootDirectory
12
13  LABEL_SEARCH_IN_ROOT_DIR_BEGIN:
14      cmp word [wRootDirSizeForLoop], 0 ; 判断根目录区是否已经读完
15      jz LABEL_NO_LOADERBIN
16      dec word [wRootDirSizeForLoop]
17      mov ax, BaseOfLoader
18      mov es, ax
19      mov bx, OffsetOfLoader   ; es:bx指向数据缓冲区
20      mov ax, [wSectorNo]      ; wSectorNo存放着扇区号
21      mov cl, 1                ; cl存放着要读取的扇区数
22      call ReadSector
23
24      mov si, LoaderFileName
25      mov di, OffsetOfLoader
26      cld
27      mov dx, 10h
28
29  LABEL_SEARCH_FOR_LOADERBIN:
30      cmp dx, 0
31      jz LABEL_GOTO_NEXT_SECTOR_IN_ROOT_DIR
32      dec dx
33      mov cx, 11
34
35  LABEL_CMP_FILENAME:
36      cmp cx, 0
37      jz LABEL_FILENAME_FOUND
38      dev cx
39      lodsb         ; si加1
40      cmp al, byte [es:di]
41      jz LABEL_GO_ON
42      jmp LABEL_DIFFERENT
43
44  LABEL_GO_ON:
45      inc di        ; di加1
46      jmp LABEL_CMP_FILENAME
47
48  LABEL_DIFFERENT:
49      ; 跳到下一个目录条目
50      and di, 0FFE0h
51      and di, 20h
52      mov si, LoaderFileName
53      jmp LABEL_SEARCH_FOR_LOADERBIN
54
55  LABEL_GOTO_NEXT_SECTOR_IN_ROOT_DIR:
56      add wor [wSectorNo], 1
57      jmp LABEL_SEARCH_IN_ROOT_DIR_BEGIN
58
59  LABEL_NO_LOADERBIN:
60      mov dh, 2
```

```
61          call  DispStr
62
63      %ifdef _BOOT_DEBUG
64          mov ax, 4c00h
65          int 21h
66      %else
67          jmp $
68      %endif
69
70      LABEL_FILENAME_FOUND:
71          jmp $
72
73      wRootDirSizeForLoop dw RootDirSectors
74      wSectorNo dw 0
75      LoaderFileName db "LOADER  BIN",0    ; 最后加个0用于判断字符串是否结束
```

需要注意的是，我们上述代码还调用了 DispStr 函数。这个函数是自己定义的，通过 dh 传递函数参数，用于选择显示哪条信息。具体实现代码如下所示：

```
1       MessageLength equ 9
2       BootMessage db "Booting  "
3       Message1    db "Ready.   "
4       Message2    db "NO LOADER"
5
6       DispStr:
7           mov ax, MessageLength
8           mul dh
9           add ax, BootMessage
10          mov bp, ax
11          mov ax, ds
12          mov es, ax
13          mov cx, MessageLength
14          mov ax, 01301h
15          mob bx, 0007h
16          mov dl, 0
17          int 10h
18          ret
```

1.4 计算 FAT 项的值

在上一节中，我们找到了 Loader 在根目录区中的条目，然后就可以根据这个条目得到 Loader 在数据区中对应的开始簇号。同时我们还可以根据这个簇号得到 FAT 表中的 FAT 项，从而找到 Loader 占用的其余所有扇区。

在将 Loader 装入内存的 BaseOfLoader:OffsetOfLoader 之前，我们先写一个函数，用于求 FAT 项的值。函数使用 ax 传递扇区号，同时使用 ax 存放返回的 FAT 项的值。实现代码如下：

```
1       SectorNoOfFAT1 equ 1
2
3       GetFATEntry:
```

```
4         push es
5         push bx
6         push ax
7         mov ax, BaseOfLoader      ; BaseOfLoader值为0900h
8         sub ax, 0100h
9         mov es, ax                ; 改变es后，数据缓冲区为0800h:bx，FAT项将存入此处
10        pop ax
11
12        mov byte [bOdd], 0
13        mov bx, 3
14        mul bx
15        mob bx, 2
16        div bx            ; ax存放着FAT项的偏移字节，dx存放着余数
17        cmp dx, 0
18        jz LABEL_EVEN
19        mov byte [bOdd], 1
20
21    LABEL_EVEN:
22        xor dx, dx
23        mov bx, [BPB_SecPerSec]
24        div bx        ; ax存放着FAT项在FAT1中的开始扇区号，dx存放着FAT项在扇区内的偏
              移
25        push dx
26        mov bx, 0     ; bx为0，FAT项将被读入0800h:0处
27        add ax, SectorNoOfFAT1
28        mov cl, 2     ; 一次读两个扇区，防止FAT跨扇区
29        call ReadSector
30
31        pop dx
32        add bx, dx    ; bx存放着FAT项在扇区的偏移字节
33        mov ax, [es:bx]
34        cmp byte [bOdd], 1
35        jnz LABEL_EVEN_2
36        shr ax, 4
37    LABEL_EVEN_2:
38        and ax, 0FFFh
39
40    LABEL_GET_FAT_ENTRY_OK:
41        pop bx
42        pop es
43        ret
44
45    bOdd DB 0
```

1.5 加载 Loader 入内存

有了前面的准备，我们现在就可以开始直接加载 Loader 进入内存了，实现代码如下：

```
1     LABEL_FILENAME_FOUND:
2         mov ax, RootDirSectors
3         and di, 0FFE0h
4         add di, 01Ah
5         mov cx, word [es:di]      ; cx存放着FAT项的序号
```

```
6         push cx
7         add cx, ax                ; ax存放着RootDirSectors，也就是根目录区占用的扇区
              数
8         add cx, DeltaSectorNo     ; cx存放着Loader的扇区号，DeltaSectorNo存放着19
              -2=17，19是根目录区开始扇区号，2是因为数据区开始簇号为2
9         mov ax, BaseOfLoader
10        mov es, ax
11        mov bx, OffsetOfLoader
12        mov ax, cx
13
14    LABEL_GOON_LOADING_FILE:
15        ; 打印一个al存放的“.”
16        push ax
17        push bx
18        mov ah, 0Eh
19        mov al, '.'
20        mov bl, 0Fh
21        int 10h
22        pop bx
23        pop ax
24
25        mov cl, 1
26        call ReadSector
27        pop ax                    ; ax存放着FAT项的序号
28        call GetFATEntry
29        cmp ax, 0FFFh
30        jz LABEL_FILE_LOADED
31        push ax
32        mov dx, RootDirSectors
33        add ax, dx
34        add ax, DeltaSectorNo
35        add bx, [BPB_BytsPerSec]      ; 数据缓冲区指向下一个512字节
36        jmp LABEL_GOON_LOADING_FILE
37
38    LABEL_FILE_LOADED:
39        jmp $
```