目 录

1 实现引导扇区 2
1.1 引导扇区格式 2
1.2 加载 Loader 进入内存 2
1.2.1 寻找 Loader 的目录条目 3
1.2.2 寻找 FAT 项 5
1.2.3 加载 Loader 8
1.2.4 执行 Loader 模块的代码 8
1.3 引导扇区完整的实现代码 9

2 实现内核雏形 14

1 实现引导扇区

一个操作系统从开机到开始运行，需要经历“引导，加载内核进入内存，跳入保护模式，开始执行内核”。操作系统使用 Loader 模块来加载内核进入内存，并跳入保护模式。引导扇区负责将 Loader 加载入内存。

1.1 引导扇区格式

引导扇区是软盘的第 0 个扇区。我们把 Loader 模块复制到软盘上，然后引导扇区将找到并加载它。

引导扇区开头有一个很重要的数据结构，叫做 BPB。这个 BPB 数据结构使得软盘被操作系统识别。加上 BPB 数据结构之后，引导扇区的格式如下面的代码所示：

```
    jmp short LABEL_START
    nop

    BS_OEMName DB 'ForrestY' ; 生厂商名字
    BPB_BytsPerSec DW 512 ; 每扇区字节数
    BPB_SecPerClus DB 1 ; 每簇多少扇区
    BPB_RsvdSecCnt DB 1 ; Boot记录占用多少扇区
    BPB_NumFATS DB 2 ; 共有多少FAT表
    BPB_RootEntCnt DW 224 ; 根目录文件数最大值
    BPB_TotSec16 DW 2880 ; 逻辑扇区总数
    BPB_Media DB 0xF0 ; 媒体描述符
    BPB_FATSz16 DW 9 ; 每FAT扇区数
    BPB_SecPerTrk DW 18 ; 每磁道扇区数
    BPB_NumHeads DW 2 ; 磁头数
    BPB_HiddSec DD 0 ; 隐藏扇区数
    BPB_TotSec32 DD 0 ; 记录扇区数
    BS_DrvNum DB 0 ; 中断13的驱动器号
    BS_Reservedl DB 0 ; 未使用
    BS_BootSig DB 29h ; 扩展引导标记
    BS_VolID DD 0 ; 卷序列号
    BS_VolLab DB 'pengsida001' ; 卷标，必须11个字节
    BS_FileSysType DB 'FAT12' ; 文件系统类型

LABEL_START:
    ; ...
```

1.2 加载 Loader 进入内存

为了加载 Loader 文件，我们首先需要知道 Loader 模块所在的位置。我们假设 Loader 模块存放在根目录中，而根目录信息存放在根目录区中。根目录区从第 19 个扇区开始，由 BPB_RootEntCnt 个目录条目组成。

目录条目占用 32 字节，它的格式如下：

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

所以，我们只要 Loader 模块的目录条目，就可以根据 DIR_FstClus 的值找到 Loader 模块。

1.2.1 寻找 Loader 的目录条目

我们通过遍历根目录区来寻找 Loader 模块目录条目，代码如下：

```
    ; 根目录的第一个扇区号是19
    SectorNoOfRootDirectory equ 19
    ; 数据缓冲区的基地址
    BaseOfLoader equ 09000h
    ; 数据缓冲区的偏移地址
    OffsetOfLoader equ 0100h
    ; Loader模块的名字
    LoaderFileName db "LOADER BIN", 0

    ; wSectorNo地址单元存放着要读取的扇区号
    mov word [wSectorNo], SectorNoOfRootDirectory
LABEL_SEARCH_IN_ROOT_DIR_BEGIN:
    ; wRootDirSizeForLoop地址单元存放着扇区数
    cmp word [wRootDirSizeForLoop], 0
    jz LABEL_NO_LOADERBIN
    ; 每遍历一次，扇区数减一
    dec word [wRootDirSizeForLoop]
    ; 设置es:bx，指定数据缓冲区的地址
    mov ax, BaseOfLoader
    mov es, ax
    mov bx, OffsetOfLoader
    ; 设置要读的扇区号
    mov ax, [wSectorNo]
    ; 设置要读的扇区数
    mov cl, 1
    ; 读取一个扇区的内容
    call ReadSector

    mov si, LoaderFileName
    mov di, OffsetOfLoader
    cld
    ; dx代表着接下来的循环次数
    ; 一个扇区512字节，一个根目录条目32字节，所以需要循环16次
    mov dx, 10h
LABEL_SEARCH_FOR_LOADERBIN:
    cmp dx, 0
    jz LABEL_GOTO_NEXT_SECTOR_IN_ROOT_DIR
```

```
    dec dx
    ; 比较文件名称，文件名称为11个字节，所以比较11次
    mov cx, 11
LABEL_CMP_FILENAME:
    cmp cx, 0
    jz LABEL_FILENAME_FOUND
    dec cx
    ; 将LoaderFileName处的字节读入al
    lodsb
    ; 比较数据缓冲区中的字节
    cmp al, byte [es:di]
    jz LABEL_GO_ON
    jmp LABEL_DIFFERENT
LABEL_GO_ON:
    inc di
    jmp LABEL_CMP_FILENAME
LABEL_DIFFERENT:
    ; 让di指向下一个条目
    ; 每个目录条目为32字节
    and di, 0FFE0h
    add di, 20h
    mov si, LoaderFileName
    jmp LABEL_SEARCH_FOR_LOADERBIN
LABEL_GOTO_NEXT_SECTOR_IN_ROOT_DIR:
    ; 读取下一个扇区号
    add word [wSectorNo], 1
    jmp LABEL_SEARCH_IN_ROOT_DIR_BEGIN
LABEL_NO_LOADERBIN:
    mov dh, 2
    call DispStr
    jmp $
LABEL_FILENAME_FOUND:
    jmp $
```

上述代码中，用到了读取一个扇区内容的函数 ReadSector。需要读软盘的时候，要用到 BIOS 中断 int 13h。

当 ah=00h 时，int 13h 用于复位软驱，此时使用 dl 指定驱动器号。

当 ah=02h 时，int 13h 用于从磁盘将数据读入 es:bx 指向的缓冲区中。当读取错误时，CF 会被置一。此时需要设置如下的寄存器值：

```
    al = 要读扇区数
    ch = 磁道号
    cl = 起始扇区号
    dh = 磁头号
    dl = 驱动器号
    es:bx指定数据缓冲区
```

在代码中添加一个 ReadSector 函数，用于读取软盘。

ReadSector 函数将 al 和 cl 作为参数，al 是相对扇区号，cl 是要读取的扇区数。在函数中，程序根据相对扇区号获得磁道号、起始扇区号和磁头号。软盘中一个磁道有 18 个扇区，于是将相对扇区号除以 18，得到的商和余数分别是总磁道号和起始扇区号。软盘

中，因为有两面，所以分别要磁头号 0 和磁头号 1 标记。现在还需要确定的是在哪个磁头号的第几个磁道号。软盘结构中，不是先排完 0 磁头号再排 1 磁头号的，而是交错排列。总磁头号为偶数的位于磁头号 0，总磁头号为奇数的位于磁头号 1。所以只要判断总磁头号的奇偶就能得到磁头号。将总磁头号与 1 相与，为 0 的话磁头号就是 0，为 1 的话磁头号就是 1。然后将总磁头号除以 2，就能得到相对于磁头的起始磁道号。读取软盘扇区的代码如下；

```
ReadSector:
    push bp
    mov bp, sp
    sub esp, 2
    ; 处理int 13h所需要的参数
    ; cl存放着要读取的扇区数
    mov byte [bp-2], cl
    push bx
    ; bl存放着每个磁道上的扇区数
    mov bl, [BPB_SecPerTrk]
    ; ax/bl，商放在al中，余数放在ah中
    div bl
    ; 得到当前磁道中的起始扇区号
    inc ah
    ; 设置cl的值为起始扇区号
    mov cl, ah
    mov dh, al
    shr al, 1
    ; 设置ch的值为磁道号
    mov ch, al
    ; 设置dh的值为磁头号
    and dh, 1
    pop bx
    ; 设置dl的值为驱动器号
    mov dl, [BS_DrvNum]
.GoOnReading:
    mov ah, 2
    mov al, byte [bp-2]
    int 13h
    jc .GoOnReading

    add esp, 2
    pop bp
    ret
```

1.2.2 寻找 FAT 项

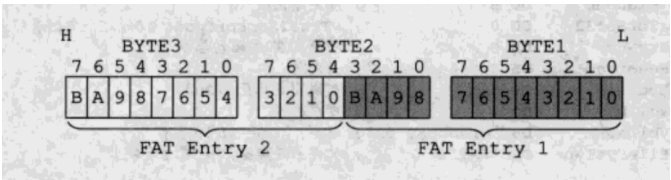
现在我们得到 Loader 模块的目录条目了，也就能得到 Loader 模块对应的开始簇号，也就能够得到 Loader.bin 的起始扇区号。通过这个扇区号，我们可以做到两件事：

- 把起始扇区装入内存。
- 通过这个扇区号找到 FAT 中的项，从而找到 Loader 占用的其他所有扇区。

在此先介绍一下软盘上的文件系统 FAT12，它由引导扇区、两个 FAT 表、根目录区和数据区组成，格式如下图:

我们之前获得的根目录条目就存放在根目录区中。现在我们拥有 Loader 模块的簇号，需要根据 FAT 表来找到 Loader 的所有簇。FAT 表由 FAT 项组成，每个 FAT 项长为 12 字节，FAT 项的值代表文件下一个簇号。如果 FAT 项的值大等于 0xFF8，那么代表当前簇是本文件的最后一个簇。如果 FAT 项的值为 0xFF7，表示它是一个坏簇。所以我们只要拥有起始簇号 n，就对应着 FAT 表中第 n 个 FAT 项，从而就能找到找到文件的所有簇号，也就找到了文件所占用的所有扇区。

FAT 项的格式如下:



实现代码如下:

```
1    bOdd db 0
2    SectorNoOfFAT1 equ 1
```

```
3     GetFATEntry:
4         push es
5         push bx
6         ; ax存放着起始扇区号，对应着第[ax]个FAT项
7         push ax
8         ; Loader模块在内存中的起始地址
9         mov ax, BaseOfLoader
10        ; 在Loader模块后面留出4k空间用于存放FAT项，作为数据缓冲区
11        sub ax, 0100h
12        mov es, ax
13        ; 恢复ax的值
14        pop ax
15        ; bOdd用于判断FAT项从第0位开始还是从第4位开始
16        mov byte [bOdd], 0
17        ; FAT项占用1.5个字节，所以ax先乘以3，再除以2
18        mov bx, 3
19        mul bx
20        mov bx, 2
21        ; 商放在ax中，余数放在dx中
22        div bx
23        ; 判断FAT项从第0位开始还是从第4位开始
24        ; ax为奇数时，FAT项从第4位开始。ax为偶数时，FAT项从第0位开始
25        cmp dx, 0
26        jz LABEL_EVEN
27        mov byte [bOdd], 1
28    LABEL_EVEN:
29        xor dx, dx
30        ; BPB_BytsPerSec是每个扇区占用的字节数
31        mov bx, [BPB_BytsPerSec]
32        ; ax存放着FAT项相对于FAT的扇区号，bx存放着FAT项在扇区中的偏移
33        div bx
34        push dx
35        mov bx, 0
36        ; ax加上FAT的扇区号，当前值为FAT项所在的扇区号
37        add ax, SectorNoOfFAT1
38        ; cl存放着要读取的扇区数
39        mov cl, 2
40        ; 读取FAT项所在的扇区，为了防止FAT项跨越两个扇区，所以一次读取两个扇区
41        call ReadSector
42        pop dx
43        add bx, dx
44        ; es是数据缓冲区基地址，bx是FAT项在扇区中的偏移
45        ; 现在ax中存放着FAT项的值
46        mov ax, [es:bx]
47        cmp byte [bOdd], 1
48        jnz LABEL_EVEN_2
49        ; 如果FAT从第4位开始，就将ax右移4位
50        shr ax, 4
51    LABEL_EVEN_2:
52        ; 如果FAT从第0位开始，就只保留ax的低12位
53        and ax, 0FFFh
54    LABEL_GET_FAT_ENTRY_OK:
55        pop bx
56        pop es
57        ret
```

1.2.3 加载 Loader

现在我们可以通过遍历根目录区来找到 Loader 模块对应的根目录条目。从根目录条目中找到相应的簇号，然后根据 FAT 表中的 FAT 项找到文件的下一个簇号。这里的簇号是相对于数据区的簇号。为了获得整个软盘中的簇号，需要加上根目录区的起始簇号，在加上根目录区簇的数量。因为数据区的簇号是从 2 开始的，所以还要减去 2。最后根据实际的簇号得到 Loader 模块所在的扇区号，然后将扇区号作为 ReadSector 函数的参数，读取相应扇区的数据。实现代码如下：

```
LABEL_FILENAME_FOUND:
    mov ax, RootDirSectors
    and di, 0FFE0h
    add di, 01Ah
    mov cx, word [es:di]
    push cx
    add cx, ax
    add cx, DeltaSectorNo
    mov ax, BaseOfLoader
    mov es, ax
    mov bx, OffsetOfLoader
    mov ax, cx
LABEL_GOON_LOADING_FILE:
    mov cl, 1
    call ReadSector
    pop ax
    call GetFATEntry
    ; 检查FAT项的值是否是0FFFh
    cmp ax, 0FFFh
    jz LABEL_FILE_LOADED
    push ax
    ; RootDirSectors是根目录扇区数
    mov dx, RootDirSectors
    add ax, dx
    ; 根目录开始扇区号为19，数据区第一个簇的簇号是2
    ; 为了正确求得FAT项对应的簇号，定义了DeltaSectorNo equ 17
    ; FAT项对应的簇号 = RootDirSectors + DeltaSectorNo + 起始簇号
    add ax, DeltaSectorNo
    ; es:bx指向数据缓冲区
    ; 读取一个扇区结束后，bx的值加512字节
    add bx, [BPB_BytsPerSec]
    jmp LABEL_GOON_LOADING_FILE
```

1.2.4 执行 Loader 模块的代码

前几个小结将 Loader 模块加载进了内存，放在 BaseOfLoader:OffsetOfLoader 处。现在只要将程序跳转到数据缓冲区的地址，就可以开始执行 Loader 模块的代码。实现代码如下：

```
jmp BaseOfLoader:OffsetOfLoader
```

回想本节一开始说的，一个操作系统从开机到开始运行，需要经历“引导，加载内核进入内存，跳入保护模式，开始执行内核”。现在我们只做到了引导，也就是将 Loader 模块加载进内存。Loader 要做的事情还有两件：

- 加载内核入内存。
- 跳入保护模式。

1.3 引导扇区完整的实现代码

前面小节虽然有贴代码，但是都是一些细节上的代码。现在感受一下完整的实现代码，以此对引导扇区整个的实现框架有清楚的认识。在阅读代码之前，先了解一下软盘上 FAT12 文件系统引导扇区的格式，从而对代码框架有更全面的认识。FAT12 引导扇区格式如下所示：

名称	偏移	长度	内容	Orange'S的值
BS_jmpBoot	0	3	一个短跳转指令	jmp LABEL_START nop
BS_OEMName	3	8	厂商名	'ForrestY'
BPB_BytsPerSec	11	2	每扇区字节数	0x200
BPB_SecPerClus	13	1	每簇扇区数	0x1
BPB_RsvdSecCnt	14	2	Boot 记录占用多少扇区	0x1
BPB_NumFATs	16	1	共有多少 FAT 表	0x2
BPB_RootEntCnt	17	2	根目录文件数最大值	0xE0
BPB_TotSec16	19	2	扇区总数	0xB40
BPB_Media	21	1	介质描述符	0xF0
BPB_FATSz16	22	2	每 FAT 扇区数	0x9
BPB_SecPerTrk	24	2	每磁道扇区数	0x12
BPB_NumHeads	26	2	磁头数（面数）	0x2
BPB_HiddSec	28	4	隐藏扇区数	0
BPB_TotSec32	32	4	如果BPB_TotSec16是0，由这个值记录扇区数	0
BS_DrvNum	36	1	中断 13 的驱动器号	0
BS_Reserved1	37	1	未使用	0
BS_BootSig	38	1	扩展引导标记（29h）	0x29
BS_VolID	39	4	卷序列号	0
BS_VolLab	43	11	卷标	'OrangeS0.02'
BS_FileSysType	54	8	文件系统类型	'FAT12'
引导代码及其他	62	448	引导代码、数据及其他填充字符等	引导代码（剩余空间被0填充）
结束标志	510	2	0xAA55	0xAA55

引导扇区实现代码如下：

```
org 07c00h

BaseOfStack equ 07c00h
BaseOfLoader equ 09000h
OffsetOfLoader equ 0100h

RootDirSectors equ 14 ; 根目录区占用了14扇区
SectorNoOfRootDirectory equ 19
SectorNoOfFAT1 equ 1
DeltaSectorNo equ 17

; FAT12引导扇区固有的头信息
; FAT12引导扇区格式已经在本节开头说明

; BS_jmpBoot，长度要求为3字节
; 因为jmp short LABEL_START指令是2字节，所以需要加个nop指令，使得长度为3字节
jmp short LABEL_START
nop

BS_OEM DB 'ForrestY'
BPB_BytsPerSec DW 512
BPB_SecPerClus DB 1
BPB_RsvdSecCnt DW 1
BPB_NumFATs DB 2
BPB_RootEntCnt DW 224
BPB_TotSec16 DW 2880
BPB_Media DB 0xF0
BPB_FATSz16 DW 9
BPB_SecPerTrk DW 18
BPB_NumHeads DW 2
BPB_HiddSec DD 0
BPB_TotSec32 DD 0
BS_DrvNum DB 0
BS_Reservd1 DB 0
BS_BootSig DB 29h
BS_VolID DD 0
BS_VolLab DB 'pengsida001'
BS_FileSysType DB 'FAT12   '

LABEL_START:
    ; 给每个段寄存器赋初值为代码段基址
    mov ax, cs
    mov ds, ax
    mov es, ax
    mov ss, ax
    ; 给堆栈指针赋初值，指向栈顶
    mov sp, BaseOfStack

    ;mov ax, 0600h
    ;mov bx, 0700h
    ;mov cx, 0
    ;mov dx, 0184fh
    ;int 10h

```

```
55      ; mov dh, 0
56      ; call DispStr
57
58      ; ah=00h时，int 13h的功能是复位软驱。使用dl来指定驱动器号
59      xor ah, ah
60      xor dl, dl
61      int 13h
62
63      ; 根目录区起始扇区号SectorNoOfRootDirector为19
64      ; 因为是要搜索根目录区中的根目录条目，所以要从头遍历
65      ; wSectorNo地址单元中存放着要读取的扇区号
66      mov word [wSectorNo], SectorNoOfRootDirectory
67  LABEL_SEARCH_IN_ROOT_DIR_BEGIN:
68      ; 检查是否已经将根目录区中的所有扇区都遍历完
69      cmp word [wRootDirSizeForLoop], 0
70      jz LABEL_NO_LOADERBIN
71      ; 将循环数减一
72      dec word [wRootDirSizeForLoop]
73      ; int 13h将扇区内容读入es:bx指定的数据缓冲区
74      mov ax, BaseOfLoader
75      mov es, ax
76      mov bx, OffsetOfLoader
77      ; ReadSector将ax和cl中的值作为参数
78      ; ReadSector将从第ax个扇区开始的cl个扇区读入ex:bx指定的数据缓冲区中
79      mov ax, [wSectorNo]
80      mov cl, 1
81      call ReadSector
82
83      ; ds存放着代码段段基址，si为LoaderFileName的偏移地址，ds:si就指向了代码段中定义
            的"LOADER  BIN"字符串
84      mov si, LoaderFileName
85      ; es存放着数据缓冲区的基地址，di为存放扇区数据处的偏移地址，es:di就指向了扇区数
            据
86      mov di, OffsetOfLoader
87      cld ; 从低位到高位
88      mov dx, 10h ; 一个扇区共有512字节，一个根目录条目为32字节，所以遍历一个扇区需要
            16次
89  LABEL_SEARCH_FOR_LOADERBIN:
90      cmp dx, 0 ; 检查循环是否结束
91      jz LABEL_GOTO_NEXT_SECTOR_IN_ROOT_DIR
92      dec dx
93      ; 文件名为11个字节。只要检查根目录条目前11个字节与文件名字符串是否相等就行了
94      mov cx, 11
95  LABEL_CMP_FILENAME:
96      cmp cx, 0 ; 检查循环是否结束
97      jz LABEL_FILENAME_FOUND
98      dec cx
99      lodsb ; 将ds:si指向的字节读入al，且将si的值自动加一
100     cmp al, byte [es:di] ; 比较两个地址处的字节是否相同
101     jz LABEL_GO_ON
102     jmp LABEL_DIFFERENT
103 LABEL_GO_ON:
104     inc di ; 将di的值加一，指向下一个字节
105     jmp LABEL_CMP_FILENAME
106 LABEL_DIFFERENT:
107     ; 如果文件名中11个字节有一个不相等，就跳过这个根目录条目
```

```
108     ; 这时候需要处理di和si
109     ; di最多加11，这时候di也只是改变低4位。所以让di的低4位与0000b相与
110     ; 因为根目录条目是32字节，所以di的值肯定是32的倍数，di的第4位肯定是0
111     ; 让di的第4位到第7位与1110b相与，让di的第4位变为0，让di指向本条目开头
112     ; 其实，因为di最多改变低4位，所以and di, 0FFF0h也能达到相同的效果
113     and di, 0FFE0h
114     ; di加32，指向下一个根目录条目
115     add di, 20h
116     ; 让si重新指向文件名字符串的开头
117     mov si, LoaderFileName
118     jmp LABEL_SEARCH_FOR_LOADERBIN
119 LABEL_GOTO_NEXT_SECTOR_IN_ROOT_DIR:
120     ; 当前扇区没有Loader模块，需要搜索下一个扇区的内容
121     ; 将要读取的扇区号加一
122     add word [wSectorNo], 1
123     jmp LABEL_SEARCH_IN_ROOT_DIR_BEGIN
124 LABEL_NO_LOADERBIN:
125     jmp $ ; 找不到Loader模块，程序陷入死循环
126 LABEL_FILENAME_FOUND:
127     mov ax, RootDirSectors
128     ; 让di指向根目录条目开头
129     and di, 0FFE0h
130     ; 让di指向此条目对应的开始簇号
131     add di, 01Ah
132     ; 将开始簇号存储在cx中
133     mov cx, word [es:di]
134     push cx ; 存储扇区在FAT表中的序号
135     ; 一个簇有一个扇区，所以簇号就是扇区号。cx目前存放着扇区号
136     ; 让cx与根目录区扇区数相加，并将结果存储在cx中
137     ; 根目录区起始扇区号为19，数据区的第一个簇的簇号是2
138     ; 所以cx加上19再减去2，就得到了相对扇区号
139     add cx, ax
140     add cx, DeltaSectorNo ; DeltaSectorNo equ 19
141     mov ax, BaseOfLoader
142     mov es, ax
143     mov bx, OffsetOfLoader
144     ; ax作为ReadSector函数的参数，存放着相对扇区号
145     mov ax, cx
146 LABEL_GOON_LOADING_FILE:
147     ; cl作为ReadSector函数的参数，存放着要读取的扇区数
148     mov cl, 1
149     call ReadSector
150     ; 将扇区在FAT表中的项号存储在ax中
151     pop ax
152     ; GetFATEntry返回后，ax存放着Loader模块下一个簇号
153     call GetFATEntry
154     ; 判断ax是否为0FFFh，如果是，说明当前扇区是Loader模块最后一个扇区
155     cmp ax, 0FFFh
156     jz LABEL_FILE_LOADED
157     push ax
158     mov dx, RootDirSectors
159     add ax, dx
160     add ax, DeltaSectorNo
161     ; 让bx指向数据缓冲区的下一个512个字节的开头，用于存放Loader模块的下一个扇区
162     add bx, [BPB_BytsPerSec]
163     jmp LABEL_GOON_LOADING_FILE
```

```
164 LABEL_FILE_LOADED:
165     ; 开始执行Loader模块的代码
166     jmp BaseOfLoader:OffsetOfLoader
167 
168 wRootDirSizeForLoop dw RootDirSectors ; 根目录区中的扇区数
169 wSectorNo dw 0 ; 用于存放要读取的扇区号
170 bOdd db 0 ; 判断FAT项是从字节中的第0位开始还是从第4位开始的
171 
172 LoaderFileName db "LOADER  BIN", 0 ; 文件名一定是11字节
173 
174 ReadSector:
175     push bp
176     mov bp, sp
177     sub esp, 2
178     ; bp-2地址单元中存放着要读取的扇区数
179     mov byte [bp-2], cl
180     push bx
181     ; BPB_SecPerTrk是每磁道的扇区数
182     mov bl, [BPB_SecPerTrk]
183     ; ax中存放着要读取的扇区号
184     ; ax/bl的商扇区所在的磁道号，存放在al中
185     ; ax/bl是要读取的扇区相对于当前磁道的起始扇区号，存放在ah中
186     div bl
187     ; 磁道的扇区号从1开始，所以要将ah的值加一
188     inc ah
189     ; 根据int 13h的要求，cl要存放相对于磁道的起始扇区号
190     ; dh存放磁头号。软盘中的磁头号不是0就是1。
191     ; 软盘的排列并不是按照我们所想象的"把0面先排完了再开始排1面“，而是交替排列的
192     ; 偶数的磁道号的磁头号为0，奇数的磁道号的磁头号为1
193     ; 所以将al中存放着总磁道号与1相与就可以得到磁头号
194     ; ch存放磁道号。原先al存放着软盘总的磁道号，而软盘有两面，所以需要除以2，得到相
            对于当前磁头的磁道号
195     mov cl, ah
196     mov dh, al
197     and dh, 1
198     shr al, 1
199     mov ch, al
200     pop bx
201     mov dl, [BS_DrvNum]
202 .GoOnReading:
203     ; ah指定int 13h的工作模式
204     mov ah, 2
205     ; al存放着要读取的扇区数
206     mov al, byte [bp-2]
207     int 13h
208     jc .GoOnReading ; 如果读取错误CF会被置为1，这里程序会重读，直到正确为止
209 
210     add esp, 2
211     pop bp
212     ret
213 
214 ; 将ax作为输入参数，指定FAT表中FAT项的序号
215 ; 将ax作为返回参数，将文件下一个簇号放在ax中
216 GetFATEntry:
217     push es
218     push bx
```

```
219     push ax
220     mov BaseOfLoader
221     sub ax, 0100h ; 在BaseOfLoader之前空出4k的空间用于存放FAT项所在的扇区
222     mov es, ax
223     pop ax
224     mov byte [bOdd], 0
225     mov bx, 3
226     mul bx
227     mov bx, 2
228     div bx
229     cmp dx, 0
230     jz LABEL_EVEN
231     mov byte [bOdd], 1
232 LABEL_EVEN:
233     xor dx, dx
234     mov bx, [BPB_BytsPerSec]
235     div bx
236     push dx
237     mov bx, 0
238     add ax, SectorNoOfFAT1
239     mov cl, 2
240     call ReadSector
241
242     pop dx
243     add dx, dx
244     mov ax, [es:bx]
245     cmp byte [bOdd], 1
246     jnz LABEL_EVEN_2
247     shr ax, 4
248 LABEL_EVEN_2:
249     and ax, 0FFFh
250 LABEL_GET_FAT_ENTRY_OK:
251     pop bx
252     pop es
253     ret
```

2 实现内核雏形

```
void merge(int a[], int n, int b[], int m, int *c)
{
    int index_a = 0;
    int index_b = 0;
    int index_c = 0;
    while(index_a < n && index_b < m)
    {
        if(a[index_a] < b[index_b])
            c[index_c++] = a[index_a++];
        else
            c[index_c++] = b[index_b++];
    }
    while(index_a < n)
        c[index_c++] = a[index_a++];
    while(index_b < m)
        c[index_c++] = b[index_b++];
}
```