第一部分　简　介

一. 硬盘结构简介

1. 硬盘参数释疑

到目前为止, 人们常说的硬盘参数还是古老的 CHS (Cylinder/Head/Sector)参数. 那么为什么要使用这些参数, 它们的意义是什么?它们的取值范围是什么?

很久以前, 硬盘的容量还非常小的时候, 人们采用与软盘类似的结构生产硬盘. 也就是硬盘盘片的每一条磁道都具有相同的扇区数. 由此产生了所谓的 3D 参数 (Disk Geometry). 既磁头数(Heads), 柱面数(Cylinders), 扇区数(Sectors),以及相应的寻址方式.

其中:

磁头数(Heads) 表示硬盘总共有几个磁头,也就是有几面盘片, 最大为 255 (用 8 个二进制位存储);

柱面数(Cylinders) 表示硬盘每一面盘片上有几条磁道, 最大为 1023(用 10 个二进制位存储);

扇区数(Sectors) 表示每一条磁道上有几个扇区, 最大为 63 (用 6 个二进制位存储).

每个扇区一般是 512 个字节, 理论上讲这不是必须的, 但好象没有取别的值的.

所以磁盘最大容量为:

255 * 1023 * 63 * 512 / 1048576 = 8024 GB (1M = 1048576 Bytes)

或硬盘厂商常用的单位:

255 * 1023 * 63 * 512 / 1000000 = 8414 GB (1M = 1000000 Bytes)

在 CHS 寻址方式中, 磁头, 柱面, 扇区的取值范围分别为 0 到 Heads - 1, 0 到 Cylinders - 1, 1 到 Sectors (注意是从 1 开始).

2. 基本 Int 13H 调用简介

BIOS Int 13H 调用是 BIOS 提供的磁盘基本输入输出中断调用, 它可以完成磁盘(包括硬盘和软盘)的复位, 读写, 校验, 定位, 诊断, 格式化等功能.它使用的就是 CHS 寻址方式, 因此最大识能访问 8 GB 左右的硬盘 (本文中如不作特殊说明, 均以 1M = 1048576 字节为单位).

3. 现代硬盘结构简介

在老式硬盘中, 由于每个磁道的扇区数相等, 所以外道的记录密度要远低于内道, 因此会浪费很多磁盘空间 (与软盘一样). 为了解决这一问题, 进一步提高硬盘容量, 人们改用等密度结构生产硬盘. 也就是说, 外圈磁道的扇区比内圈磁道多. 采用这种结构后, 硬盘不再具有实际的 3D 参数, 寻址方式也改为线性寻址, 即以扇区为单位进行寻址.

为了与使用 3D 寻址的老软件兼容 (如使用 BIOS Int13H 接口的软件), 在硬盘控制器内部安装了一个地址翻译器, 由它负责将老式 3D 参数翻译成新的线性参数. 这也是为什么现在硬盘的 3D 参数可以有多种选择的原因 (不同的工作模式, 对应不同的 3D 参数, 如 LBA,

LARGE, NORMAL).

4. 扩展 Int 13H 简介

虽然现代硬盘都已经采用了线性寻址, 但是由于基本 Int 13H 的制约, 使用 BIOS Int 13H 接口的程序, 如 DOS 等还只能访问 8 G 以内的硬盘空间.为了打破这一限制, Microsoft 等几家公司制定了扩展 Int 13H 标准(Extended Int13H), 采用线性寻址方式存取硬盘, 所以突破了 8 G 的限制,而且还加入了对可拆卸介质 (如活动硬盘) 的支持.

二. Boot Sector 结构简介

1. Boot Sector 的组成

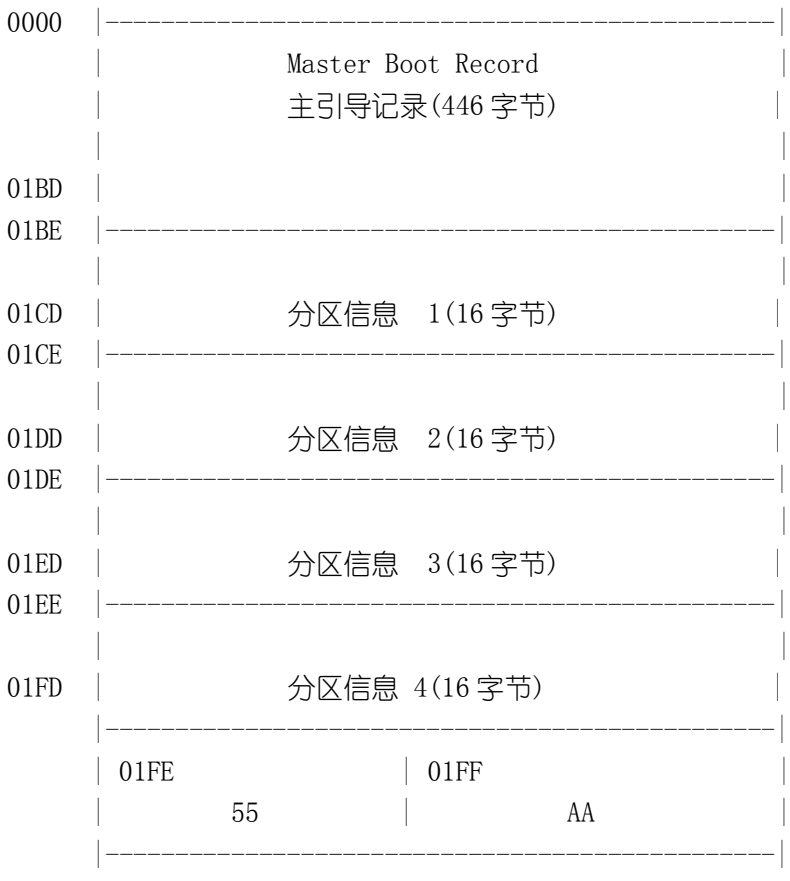
Boot Sector 也就是硬盘的第一个扇区, 它由 MBR (Master Boot Record),DPT (Disk Partition Table) 和 Boot Record ID 三部分组成.

MBR 又称作主引导记录占用 Boot Sector 的前 446 个字节 (0 to 0x1BD), 存放系统主引导程序 (它负责从活动分区中装载并运行系统引导程序).

DPT 即主分区表占用 64 个字节 (0x1BE to 0x1FD), 记录了磁盘的基本分区信息. 主分区表分为四个分区项, 每项 16 字节, 分别记录了每个主分区的信息(因此最多可以有四个主分区).

Boot Record ID 即引导区标记占用两个字节 (0x1FE and 0x1FF), 对于合法引导区, 它等于 0xAA55, 这是判别引导区是否合法的标志.

Boot Sector 的具体结构如下图所示 (参见 NightOwl 大侠的文章):



2. 分区表结构简介

分区表由四个分区项构成, 每一项的结构如下:

BYTE State : 分区状态, 0 = 未激活, 0x80 = 激活 (注意此项)
BYTE StartHead : 分区起始磁头号
WORD StartSC : 分区起始扇区和柱面号, 底字节的低 6 位为扇区号,
高 2 位为柱面号的第 9,10 位, 高字节为柱面号的低 8 位
BYTE Type : 分区类型, 如 0x0B = FAT32, 0x83 = Linux 等,
00 表示此项未用
BYTE EndHead : 分区结束磁头号
WORD EndSC : 分区结束扇区和柱面号, 定义同前
DWORD Relative : 在线性寻址方式下的分区相对扇区地址
(对于基本分区即为绝对地址)
DWORD Sectors : 分区大小 (总扇区数)

注意: 在 DOS / Windows 系统下, 基本分区必须以柱面为单位划分 (Sectors * Heads 个扇区), 如对于 CHS 为 764/255/63 的硬盘, 分区的最小尺寸为 255 * 63 * 512 / 1048576 = 7.844 MB.

3. 扩展分区简介

由于主分区表中只能分四个分区, 无法满足需求, 因此设计了一种扩展分区格式. 基本上说, 扩展分区的信息是以链表形式存放的, 但也有一些特别的地方.

首先, 主分区表中要有一个基本扩展分区项, 所有扩展分区都隶属于它,也就是说其他所有扩展分区的空间都必须包括在这个基本扩展分区中. 对于 DOS / Windows 来说, 扩展分区的类型为 0x05.

除基本扩展分区以外的其他所有扩展分区则以链表的形式级联存放, 后一个扩展分区的数据项记录在前一个扩展分区的分区表中, 但两个扩展分区的空间并不重叠.

扩展分区类似于一个完整的硬盘, 必须进一步分区才能使用. 但每个扩展分区中只能存在一个其他分区. 此分区在 DOS/Windows 环境中即为逻辑盘.因此每一个扩展分区的分区表 (同样存储在扩展分区的第一个扇区中)中最多只能有两个分区数据项(包括下一个扩展分区的数据项).

扩展分区和逻辑盘的示意图如下:

三. 系统启动过程简介

系统启动过程主要由一下几步组成(以硬盘启动为例):

1. 开机 :-)

2. BIOS 加电自检 (Power On Self Test -- POST)
   内存地址为 0ffff:0000
3. 将硬盘第一个扇区 (0 头 0 道 1 扇区, 也就是 Boot Sector)
   读入内存地址 0000:7c00 处.
4. 检查 (WORD) 0000:7dfe 是否等于 0xaa55, 若不等于
   则转去尝试其他启动介质, 如果没有其他启动介质则显示
   "No ROM BASIC" 然后死机.
5. 跳转到 0000:7c00 处执行 MBR 中的程序.
6. MBR 首先将自己复制到 0000:0600 处, 然后继续执行.
7. 在主分区表中搜索标志为活动的分区. 如果发现没有活动
   分区或有不止一个活动分区, 则转停止.
8. 将活动分区的第一个扇区读入内存地址 0000:7c00 处.
9. 检查 (WORD) 0000:7dfe 是否等于 0xaa55, 若不等于则

   显示 "Missing Operating System" 然后停止, 或尝试
   软盘启动.
10. 跳转到 0000:7c00 处继续执行特定系统的启动程序.
11. 启动系统 …

以上步骤中 2,3,4,5 步是由 BIOS 的引导程序完成. 6,7,8,9,10 步由 MBR 中的引导程序完成.

一般多系统引导程序 (如 SmartFDISK, BootStar, PQBoot 等)都是将标准主引导记录替换成自己的引导程序, 在运行系统启动程序之前让用户选择要启动的分区.而某些系统自带的多系统引导程序 (如 lilo, NT Loader 等)则可以将自己的引导程序放在系统所处分区的第一个扇区中, 在 Linux 中即为 SuperBlock (其实 SuperBlock 是两个扇区).

注: 以上各步骤中使用的是标准 MBR, 其他多系统引导程序的引导过程与此不同.

第二部分 技术资料

一. 简介

设计扩展 Int13H 接口的目的是为了扩展 BIOS 的功能, 使其支持多于 1024 柱面的硬盘, 以及可移动介质的琐定, 解锁及弹出等功能.

二. 数据结构

1. 数据类型约定

BYTE 1 字节整型 (8 位)
WORD 2 字节整型 (16 位)
DWORD 4 字节整型 (32 位)
QWORD 8 字节整型 (64 位)

2. 磁盘地址数据包 Disk Address Packet (DAP)

DAP 是基于绝对扇区地址的, 因此利用 DAP, Int13H 可以轻松地逾越 1024 柱面的限制, 因为它根本就不需要 CHS 的概念.

DAP 的结构如下:

```
struct DiskAddressPacket
{
    BYTE PacketSize;        // 数据包尺寸(16 字节)
    BYTE Reserved;          // ==0
    WORD BlockCount;        // 要传输的数据块个数(以扇区为单位)
    DWORD BufferAddr;       // 传输缓冲地址(segment:offset)
    QWORD BlockNum;         // 磁盘起始绝对块地址
};
```

PacketSize 保存了 DAP 结构的尺寸, 以便将来对其进行扩充. 在目前使用的扩展 Int13H 版本中 PacketSize 恒等于 16. 如果它小于 16, 扩展 Int13H 将返回错误码(AH=01, CF=1).

BlockCount 对于输入来说是需要传输的数据块总数, 对于输出来说是实际传输的数据块个数. BlockCount = 0 表示不传输任何数据块.

BufferAddr 是传输数据缓冲区的 32 位地址 (段地址:偏移量). 数据缓冲区必须位于常规内存以内(1M).

BlockNum 表示的是从磁盘开始算起的绝对块地址(以扇区为单位),与分区无关. 第一个块地址为 0. 一般来说, BlockNum 与 CHS 地址的关系是:

```
BlockNum = cylinder * NumberOfHeads +
           head * SectorsPerTrack +
           sector - 1;
```

其中 cylinder, head, sector 是 CHS 地址, NumberOfHeads 是磁盘的磁头数, SectorsPerTrack 是磁盘每磁道的扇区数.

也就是说 BlockNum 是沿着 扇区->磁道->柱面 的顺序记数的. 这一顺序是由磁盘控制器虚拟的, 磁盘表面数据块的实际排列顺序可能与此不同(如为了提高磁盘速度而设置的间隔因子将会打乱扇区的排列顺序).

3. 驱动器参数数据包 Drive Parameters Packet

驱动器参数数据包是在扩展 Int13H 的取得驱动器参数子功能调用中使用的数据包. 格式如下:

```
struct DriveParametersPacket
{
    WORD InfoSize;              // 数据包尺寸 (26 字节)
    WORD Flags;                 // 信息标志
    DWORD Cylinders;            // 磁盘柱面数
    DWORD Heads;                // 磁盘磁头数
    DWORD SectorsPerTrack;      // 每磁道扇区数
    QWORD Sectors;              // 磁盘总扇区数
    WORD SectorSize;            // 扇区尺寸 (以字节为单位)
};
```

信息标志用于返回磁盘的附加信息, 每一位的定义如下:

0 位:

0 = 可能发生 DMA 边界错误

1 = DMA 边界错误将被透明处理

如果这位置 1, 表示 BIOS 将自动处理 DMA 边界错误, 也就是说错误代码 09H 永远也不会出现.

1 位:

0 = 未提供 CHS 信息

1 = CHS 信息合法

如果块设备的传统 CHS 几何信息不适当的话, 该位将置 0.

2 位:

0 = 驱动器不可移动

1 = 驱动器可移动

3 位: 表示该驱动器是否支持写入时校验.

4 位:

0 = 驱动器不具备介质更换检测线

1 = 驱动器具备介质更换检测线

5 位:

0 = 驱动器不可锁定
1 = 驱动器可以锁定
要存取驱动器号大于 0x80 的可移动驱动器, 该位必须置 1
(某些驱动器号为 0 到 0x7F 的设备也需要置位)

6 位:
0 = CHS 值是当前存储介质的值 (仅对于可移动介质), 如果
驱动器中有存储介质, CHS 值将被返回.
1 = CHS 值是驱动器支持的最大值 (此时驱动器中没有介质).

7 - 15 位: 保留, 必须置 0.

三. 接口规范

1. 寄存器约定

在扩展 Int13H 调用中一般使用如下寄存器约定:

ds:di ==> 磁盘地址数据包(disk address packet)
dl ==> 驱动器号
ah ==> 功能代码 / 返回码

在基本 Int13H 调用中, 0 - 0x7F 之间的驱动器号代表可移动驱动器 0x80 - 0xFF 之间的驱动器号代表固定驱动器. 但在扩展 Int13H 调用中 0x80 - 0xFF 之间还包括一些新出现的可移动驱动器, 比如活动硬盘等. 这些驱动器支持先进的锁定,解锁等功能.

Ah 返回的错误码除了标准 Int13H 调用规定的基本错误码以外,又增加了以下错误码:

B0h 驱动器中的介质未被锁定

B1h 驱动器中的介质已经锁定

B2h 介质是可移动的

B3h 介质正在被使用

B4h 锁定记数溢出

B5h 合法的弹出请求失败

2. API 子集介绍

1.x 版的扩展 Int13H 调用中规定了两个主要的 API 子集.

第一个子集提供了访问大硬盘所必须的功能, 包括 检查扩展 In13H 是否存在(41h), 扩展读(42h), 扩展写(43h), 校验扇区(44h), 扩展定位(47h) 和 取得驱动器参数(48h).

第二个子集提供了对软件控制驱动器锁定和弹出的支持, 包括 检查扩展 Int13H 是否存在(41h), 锁定/解锁驱动器(45h), 弹出驱动器(46h),取得驱动器参数(48h), 取得扩展驱动器改变状态(49h), int 15h.

如果使用了调用规范中不支持的功能, BIOS 将返回错误码 ah = 01h, CF = 1.

3. API 详解

1) 检验扩展功能是否存在

入口:

AH = 41h
BX = 55AAh
DL = 驱动器号

返回:

CF = 0
AH = 扩展功能的主版本号
AL = 内部使用
BX = AA55h
CX = API 子集支持位图
CF = 1
AH = 错误码 01h, 无效命令

这个调用检验对特定的驱动器是否存在扩展功能. 如果进位标志置 1 则此驱动器不支持扩展功能. 如果进位标志为 0, 同时 BX = AA55h, 则存在扩展功能. 此时 CX 的 0 位表示是否支持第一个子集, 1 位表示是否支持第二个子集.

对于 1.x 版的扩展 Int13H 来说, 主版本号 AH = 1. AL 是副版本号,但这仅限于 BIOS 内部使用, 任何软件不得检查 AL 的值.

2) 扩展读

入口:

AH = 42h
DL = 驱动器号
DS:DI = 磁盘地址数据包(Disk Address Packet)

返回:

CF = 0, AH = 0 成功
CF = 1, AH = 错误码

这个调用将磁盘上的数据读入内存. 如果出现错误, DAP 的 BlockCount 项中则记录了出错前实际读取的数据块个数.

3) 扩展写

入口:

AH = 43h

AL

0 位 = 0 关闭写校验

1 打开写校验

1 - 7 位保留, 置 0

DL = 驱动器号

DS:DI = 磁盘地址数据包(DAP)

返回:

CF = 0, AH = 0 成功

CF = 1, AH = 错误码

这个调用将内存中的数据写入磁盘. 如果打开了写校验选项, 但 BIOS 不支持, 则会返回错误码 AH = 01h, CF = 1. 功能 48h 可以检测 BIOS 是否支持写校验.

如果出现错误, DAP 的 BlockCount 项中则记录了出错前实际写入的数据块个数.

4) 校验扇区

入口:

AH = 44h

DL = 驱动器号

DS:DI = 磁盘地址数据包(Disk Address Packet)

返回:

CF = 0, AH = 0 成功

CF = 1, AH = 错误码

这个调用校验磁盘数据, 但并不将数据读入内存.如果出现错误, DAP 的 BlockCount 项中则记录了出错前实际校验的数据块个数.

5) 锁定/解锁驱动器

入口:

AH = 45h

AL

= 0 锁定驱动器

= 1 驱动器解锁

= 02 返回锁定/解锁状态

= 03h-FFh - 保留

DL = 驱动器号

返回:

CF = 0, AH = 0 成功

CF = 1, AH = 错误码

这个调用用来缩定指定驱动器中的介质.

所有标号大于等于 0x80 的可移动驱动器必须支持这个功能. 如果在支持可移动驱动器控制功能子集的固定驱动器上使用这个功能调用, 将会成功返回.

驱动器必须支持最大 255 次锁定, 在所有锁定被解锁之前, 不能在物理上将驱动器解锁. 解锁一个未锁定的驱动器,将返回错误码 AH= B0h. 如果锁定一个已锁定了 255 次的驱动器, 将返回错误码 AH = B4h.

锁定一个没有介质的驱动器是合法的.

6) 弹出可移动驱动器中的介质

入口:

AH = 46h

AL = 0 保留

DL = 驱动器号

返回:

CF = 0, AH = 0 成功

CF = 1, AH = 错误码

这个调用用来弹出指定的可移动驱动器中的介质.

所有标号大于等于 0x80 的可移动驱动器必须支持这个功能. 如果在支持可移动驱动器控制功能子集的固定驱动器上使用这个功能调用, 将会返回错误码 AH = B2h (介质不可移动). 如果试图弹出一个被锁定的介质将返回错误码 AH = B1h (介质被锁定).

如果试图弹出一个没有介质的驱动器, 则返回错误码 Ah = 31h (驱动器中没有介质).

如果试图弹出一个未锁定的可移动驱动器中的介质, Int13h 会调用 Int15h(AH = 52h) 来检查弹出请求能否执行. 如果弹出请求被拒绝则返回错误码(同 Int15h). 如果弹出请求被接受,但出现了其他错误, 则返回错误码 AH = B5h.

7) 扩展定位

入口:

AH = 47h

DL = 驱动器号

DS:DI = 磁盘地址数据包(Disk Address Packet)

返回:

CF = 0, AH = 0 成功

CF = 1, AH = 错误码

这个调用将磁头定位到指定扇区.

8) 取得驱动器参数

入口:

AH = 48h

DL = 驱动器号

DS:DI = 返回数据缓冲区地址

返回:

CF = 0, AH = 0 成功

DS:DI 驱动器参数数据包地址, (参见前面的文章)

CF = 1, AH = 错误码

这个调用返回指定驱动器的参数.

9) 取得扩展驱动器介质更换检测线状态

入口:

AH = 49h

DL = 驱动器号

返回:

CF = 0, AH = 0 介质未更换

CF = 1, AH = 06h 介质可能已更换

这个调用返回指定驱动器的介质更换状态.

这个调用与 Int13h AH = 16h 子功能调用相同, 只是允许任何驱动器标号. 如果对一台支持可移动介质功能子集的固定驱动器使用此功能,则永远返回 CF = 0, AH = 0.

简单地将可移动介质锁定再解锁就可以激活检测线, 而无须真正更换介质.

10) Int 15h 可移动介质弹出支持

入口:

AH = 52h

DL = 驱动器号

返回:

CF = 0, AH = 0 弹出请求可能可以执行

CF = 1, AH = 错误码 B1h 或 B3h 弹出请求不能执行

这个调用是由 Int13h AH=46h 弹出介质功能调用内部使用的.

对硬盘进行操作的常用端口是 1f0h~1f7h 号端口，各端口含义如下：

端口号 读还是写 具体含义

1F0H 读/写 用来传送读/写的数据(其内容是正在传输的一个字节的数据)

1F1H 读 用来读取错误码

1F2H 读/写 用来放入要读写的扇区数量

1F3H 读/写 用来放入要读写的扇区号码

1F4H 读/写 用来存放读写柱面的低 8 位字节

1F5H 读/写 用来存放读写柱面的高 2 位字节(其高 6 位恒为 0)

1F6H 读/写 用来存放要读/写的磁盘号及磁头号

第 7 位 恒为 1

第 6 位 恒为 0

第 5 位 恒为 1
第 4 位 为 0 代表第一块硬盘、为 1 代表第二块硬盘
第 3~0 位 用来存放要读/写的磁头号
1f7H 读 用来存放读操作后的状态
第 7 位 控制器忙碌
第 6 位 磁盘驱动器准备好了
第 5 位 写入错误
第 4 位 搜索完成
第 3 位 为 1 时扇区缓冲区没有准备好
第 2 位 是否正确读取磁盘数据
第 1 位 磁盘每转一周将此位设为 1,
第 0 位 之前的命令因发生错误而结束
写 该位端口为命令端口,用来发出指定命令
为 50h 格式化磁道
为 20h 尝试读取扇区
为 21h 无须验证扇区是否准备好而直接读扇区
为 22h 尝试读取长扇区(用于早期的硬盘,每扇可能不是 512 字节,而是 128 字节到 1024 之间的值)
为 23h 无须验证扇区是否准备好而直接读长扇区
为 30h 尝试写扇区
为 31h 无须验证扇区是否准备好而直接写扇区
为 32h 尝试写长扇区
为 33h 无须验证扇区是否准备好而直接写长扇区

注:当然看完这个表你会发现，这种读写端口的方法其实是基于磁头、柱面、扇区的硬盘读写方法，不过大于 8G 的硬盘的读写方法也是通过端口 1F0H~1F7H 来实现的^_^