MBR 分区表构成

MBR 分区表以 80 为起始，以 55AA 为结束，共 64 个字节，分为 4 个分区表，一个可启动分区和三个不可启动分区。其结构如下：

分区表一：

第 1 个字节

80（HEX） *可启动分区

第 2~4 字节

01/01/00
（HEX）*开始磁头/开始扇区/开始柱面

换算一下，二进制表示就是。

0000 0001/00000001/0000 0000
（BIN）*开始磁头/开始扇区/开始柱面

以下将使用 16 进制表示，不再换算成二进制！

第五个字节

0C *分区类型 FAT32 NTFS 为 07，可自己变更

第 6~8 字节

FE/FF/FF *结束磁头/结束扇区/结束柱面

第 9~12 字节

00 00 00 3F *分区前的隐藏扇区

63 个隐藏扇区，此设计使 0 磁道使用 0 扇区，而 1 到 62 扇区不使用，减少读取，以此保护分区表，个人认为是个很不错的设计，不过很少有人知道这个。

第 13~16 字节

5B 24 40 01 *分区大小

20980827 个扇区
当然好要算上分区表的 63 个扇区
一共 20980889 个扇区

合 10GB 公式为 20980889*512/（1024^3）=10GB

----------我是一条分割线----------

分区表二：

第 17 个字节

00：不可启动分区

第 18~20 字节

开始磁头/开始扇区/开始柱面

第 21 字节

0F *分区类型为扩展分区

第 22~24 字节

结束磁头/结束扇区/结束柱面

第 25~28 字节

分区前的隐藏扇区

第 29~32 字节

分区大小

分区表三，分区表四，以此类推，不再解释。

至于分区表的修改 WinHex（DISKEDIT DOS 下）是个不错的软件，了解了这些，就可以自己重建分区表了。

另：MBR 是主分区表，误操作使 MBR 损坏时，DBR 或者 DBR 备份应该没损坏，我们可以通过搜索 55AA 来获得 DBR 信息（往前找就行了），来寻找 DBR 从而了解更多信息（主要是分区大小和开始结束的扇区，磁头，柱头），来帮助我们重建分区表。

此文乃看到一篇关于 CIH 磁盘修复的文章，后查了些书写出的，不能算原创，因为大部分资料都能查到，只不过很零散罢了，个人只是汇总了一下。在上次写的帖子里提到了这个方法 http://bbs.kafan.cn/thread-748156-1-1.html，当时懒得弄，现在有时间了，就写出来了。

MBR（Master Boot Record,另一说法为 Main Boot Record）,中文意为主引导区记录。

硬盘的 0 磁道的第一个扇区称为 MBR，它的大小是 512 字节，而这个区域可以分为三个部分。第一部分为 pre-boot 区（预启动区），占 446 字节；第二部分是 Partition table 区（分区表），占 64 个字节，硬盘中分区有多少以及每一分区的大小都记在其中。第三部分是 magic number，占 2 个字节，固定为 55AA。

他是不属于任何一个操作系统，也不能用操作系统提供的磁盘操作命令来读取它。但我们可以用 ROM-BIOS 中提供的 INT13H 的 2 号功能来读出该扇区的内容，也可用软件工具 Norton8.0 中的 DISKEDIT.EXE 来读取。

用 INT13H 的读磁盘扇区功能的调用参数如下：

入口参数：AH=2 （指定功能号）

AL=要读取的扇区数

DL=磁盘号（0、1-软盘；80、81-硬盘）

DH=磁头号

CL 高 2 位+CH=柱面号

CL 低 6 位=扇区号

CS:BX=存放读取数据的内存缓冲地址

出口参数：CS:BX=读取数据存放地址

错误信息：如果出错 CF=1 AH=错误代码

用 DEBUG 读取位于硬盘 0 柱面、0 磁头、1 扇区的操作如下：

A>DEBUG

-A 100

XXXX:XXXX MOV AX,0201 （用功能号 2 读 1 个扇区）

XXXX:XXXX MOV BX,1000 （把读出的数据放入缓冲区的地址为 CS:1000）

XXXX:XXXX MOV CX,0001 （读 0 柱面，1 扇区）

XXXX:XXXX MOV DX,0080 （指定第一物理盘的 0 磁头）

XXXX:XXXX INT 13

XXXX:XXXX INT 3

XXXX:XXXX （按回车键）

-G=100 （执行以上程序段）

-D 1000 11FF （显示 512 字节的 MBR 内容）

MBR 组成

一个扇区的硬盘主引导记录 MBR 由如图 6-15 所示的 4 个部分组成。

·主引导程序（偏移地址 0000H--0088H），它负责从活动分区中装载，并运行系统引导程序。

·出错信息数据区，偏移地址 0089H--00E1H 为出错信息，00E2H--01BDH 全为 0 字节。

·分区表（DPT,Disk Partition Table）含 4 个分区项，偏移地址 01BEH--01FDH,每个分区表项长 16 个字节，共 64 字节为分区项 1、分区项 2、分区项 3、分区项 4。

·结束标志字，偏移地址 01FE--01FF 的 2 个字节值为结束标志 55AA,如果该标志错误系统就不能启动。

物理第一扇 0 柱面，0 面，1 扇区是硬盘主引导记录扇 MBR,计算机启动时，首先就

读取该扇，读出硬盘分区表，从中选择三个主分区中唯一一个具有活动标记的分区，引导该分区上的操作系统。也就是说，无论有几个主分区（≤3），其中必须有一个分区是活动的。对硬盘进行分区，可以使用任何硬盘分区软件~

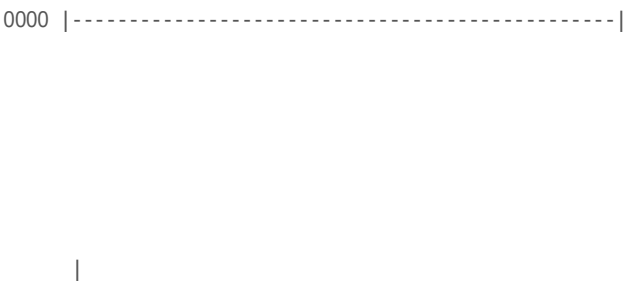
硬盘的前 512Byte，MBR 是前 446byte,447-510 是 DPT（Disk Partition Table）分区表.最后 2 位是奇偶校验，校验这个 MBR 和 DPT 是否完整.

请大家注意，备份的时候我们还是要备份前面 512Byte

MBR **知识点：**

引导扇区在每个分区里都存在，但是我们常说的*主引导扇区*是硬盘的第一物理扇区。它由两个部分组成：即主引导记录 MBR 和硬盘分区表 DPT。

在总共 512 字节的主引导分区里其中 MBR 占 446 个字节（偏移 0-- 偏移 1BDH)，DPT 占 64 个字节（偏移 1BEH--偏移 1FDH),最后两个字节“55，AA”（偏移 1FEH 偏移 1FFH)是分区的结束标志。大致的结构如下图：



```
|

          |

    | Main Boot Record |

    | |

    | |

    | 主引导记录(446 字节) |

    | |

    | |

    | |

    01BD | |

    01BE |-----------------------------------------------|

    | |

    01CD | 分区信息 1(16 字节) |

    01CE |-----------------------------------------------|

    | |

    01DD | 分区信息 2(16 字节) |

    01DE |-----------------------------------------------|

    | |

    01ED | 分区信息 3(16 字节) |

    01EE |-----------------------------------------------|

    | |
```

01FD | 分区信息 4(16 字节) |

|--|

| 01FE | 01FF |

| 55 | AA |

|--|

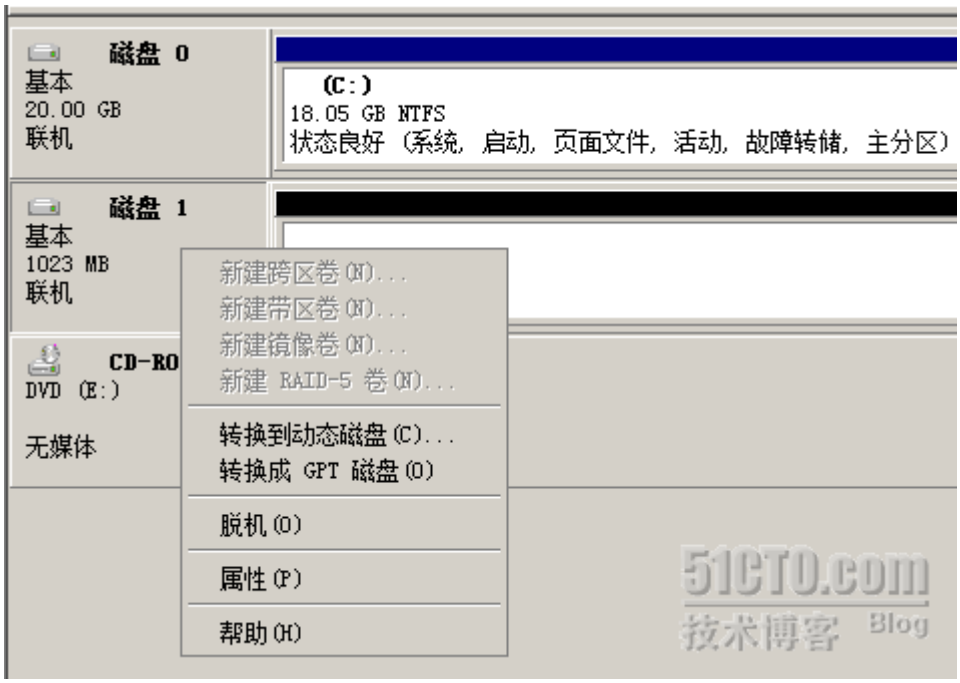
主引导记录中包含了硬盘的一系列参数和一段引导程序。引导程序主要是用来在系统硬件自检完后引导具有激活标志的分区上的操作系统。它执行到最后的是一条 **JMP** 指令跳到操作系统的引导程序去。这里往往是引导型病毒的注入点，也是各种多系统引导程序的注入点。但是由于引导程序本身完成的功能比较简单，所以我们可以完全地判断该引导程序的合法性（看 **JMP** 指令的合法性），因而也易于修复。象命令 **fdisk/mbr** 可以修复 **MBR** 和 **KV300** 这类软件可以查杀任意类型的引导型病毒，就是这个原因。

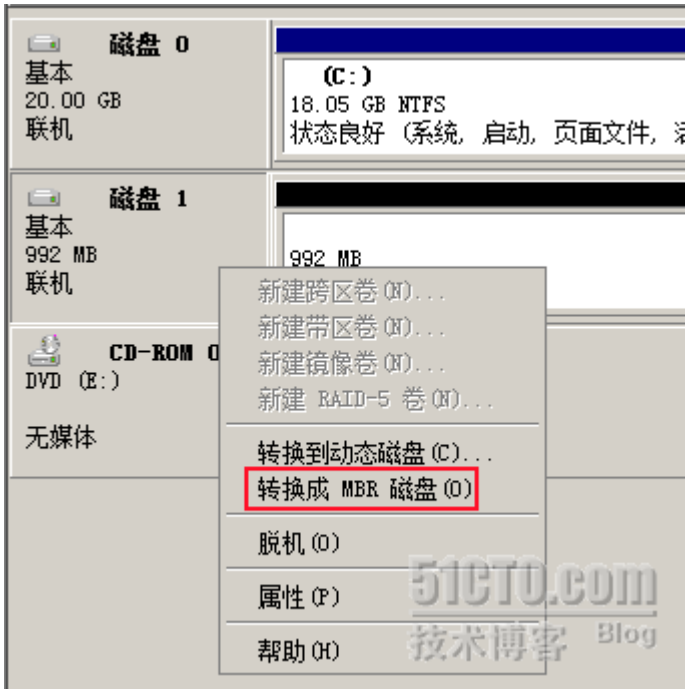
基本磁盘，动态磁盘，GPT 磁盘，MBR 磁盘介绍

2010-09-03 12:12:01

标签：电脑 IT 磁盘

Windows 磁盘管理器中，在磁盘标签处右击鼠标，随磁盘属性的不同会出现“转换到动态磁盘”，“转换到基本磁盘”“转换成 GPT 磁盘”，“转换成 MBR 磁盘”等选项。本文就此做简单介绍。部分资料参照网络上的资源。





基本磁盘与动态磁盘

磁盘的使用方式可以分为两类：一类是“基本磁盘”。基本磁盘非常常见，我们平时使用的磁盘类型基本上都是“基本磁盘”。“基本磁盘”受 26 个英文字母的限制，也就是说磁盘的盘符只能是 26 个英文字母中的

一个。因为A、B已经被软驱占用，实际上磁盘可用的盘符只有C～Z 24个。另外，在“基本磁盘”上只能建立四个主分区（注意是主分区，而不是扩展分区）；另一种磁盘类型是“动态磁盘”。“动态磁盘”不受26个英文字母的限制，它是用“卷”来命名的。“动态磁盘”的最大优点是可以将磁盘容量扩展到非邻近的磁盘空间。

动态硬盘，是指在磁盘管理器中将本地硬盘升级得来的。动态磁盘与基本磁盘相比，最大的不同就是不再采用以前的分区方式，而是叫做卷集（Volume），卷集分为简单卷、跨区卷、带区卷、镜像卷、RAID-5 卷。基本磁盘和动态磁盘相比，有以下区别：

1、卷集或分区数量。动态磁盘在一个硬盘上可创建的卷集个数没有限制。而基本磁盘在一个硬盘上只能分最多四个主分区。

2、磁盘空间管理。动态磁盘可以把不同磁盘的分区创建成一个卷集，并且这些分区可以是非邻接的，这样的大小就是几个磁盘分区的总大小。基本磁盘则不能跨硬盘分区并且要求分区必须是连续的空间，每个分区的容量最大只能是单个硬盘的最大容量，存取速度和单个硬盘相比也没有提升。

3、磁盘容量大小管理。动态磁盘允许我们在不重新启动机器的情况下调整动态磁盘大小，而且不会丢失和损坏已有的数据。而基本磁盘的分区一旦创建，就无法更改容量大小，除非借助于第三方磁盘工具软件，比如PQ Magic。

4、磁盘配置信息管理和容错。动态磁盘将磁盘配置信息放在磁盘中，如果是RAID容错系统会被复制到 其他动态磁盘上，这样可以利用RAID-1的容错功能，如果某个硬盘损坏，系统将自动调用另一个硬盘的数据，保持数据的完整性。而基本磁盘将配置信息存放在引导区，没有容错功能。

基本磁盘转换为动态磁盘可以直接进行，但是该过程是不可逆的。要想转回基本磁盘，只有把所有数据

全部拷出，然后删除硬盘所有分区后才能转回去。

GPT 磁盘与 MBR 磁盘

GPT（Globally Unique Identifier Partition Table Format）一种由基于 Itanium 计算机中的可扩展固件接口 (EFI) 使用的磁盘分区架构。与主启动记录 (MBR) 分区方法相比，GPT 具有更多的优点，因为它允许每个磁盘有多达 128 个分区，支持高达 18 千兆兆字节的卷大小，允许将主磁盘分区表和备份磁盘分区表用于冗余，还支持唯一的磁盘和分区 ID (GUID)。

与支持最大卷为 2 TB (terabytes) 并且每个磁盘最多有 4 个主分区（或 3 个主分区，1 个扩展分区和无限制的逻辑驱动器）的主启动记录 (MBR) 磁盘分区的样式相比，GUID 分区表 (GPT) 磁盘分区样式支持最大卷为 18 EB (exabytes) 并且每磁盘最多有 128 个分区。与 MBR 分区的磁盘不同，至关重要的平台操作数据位于分区，而不是位于非分区或隐藏扇区。另外，GPT 分区磁盘有多余的主要及备份分区表来提高分区数据结构的完整性。

在运行带有 Service Pack 1 (SP1) 的 Windows Server 2003 的基于 x86 的计算机和基于 x64 的计算机上，操作系统必须驻留在 MBR 磁盘上。其他的硬盘可以是 MBR 或 GPT。

在基于 Itanium 的计算机上，操作系统加载程序和启动分区必须驻留在 GPT 磁盘上。其他的硬盘可以是 MBR 或 GPT。

在单个动态磁盘组中既可以有 MBR，也可以有 GPT 磁盘。也使用将基本 GPT 和 MBR 磁盘的混合，但它们不是磁盘组的一部分。可以同时使用 MBR 和 GPT 磁盘来创建镜像卷、带区卷、跨区卷和 RAID-5 卷，但是 MBR 的

柱面对齐的限制可能会使得创建镜像卷有困难。通常可以将 MBR 的磁盘镜像到 GPT 磁盘上，从而避免柱面对齐的问题。可以将 MBR 磁盘转换为 GPT 磁盘，并且只有在磁盘为空的情况下，才可以将 GPT 磁盘转换为 MBR 磁盘。否则数据将发生丢失！！！

不能在可移动媒体，或者在与群集服务使用的共享 SCSI 或 Fibre Channel 总线连接的群集磁盘上使用 GPT 分区样式。

深度探讨 MBR 引导

2010-07-06 11:22:37

标签：探讨 深度 引导 MBR

http://popeyeywy.blog.51cto.com/745223/344738

在本文开始，笔者提出了 4 个疑问，这些问题将帮助我们更好的理解计算机启动引导的整个过程，让我们的思路更加清晰

1、全新硬盘 GHOST 克隆恢复，系统可以启动吗？

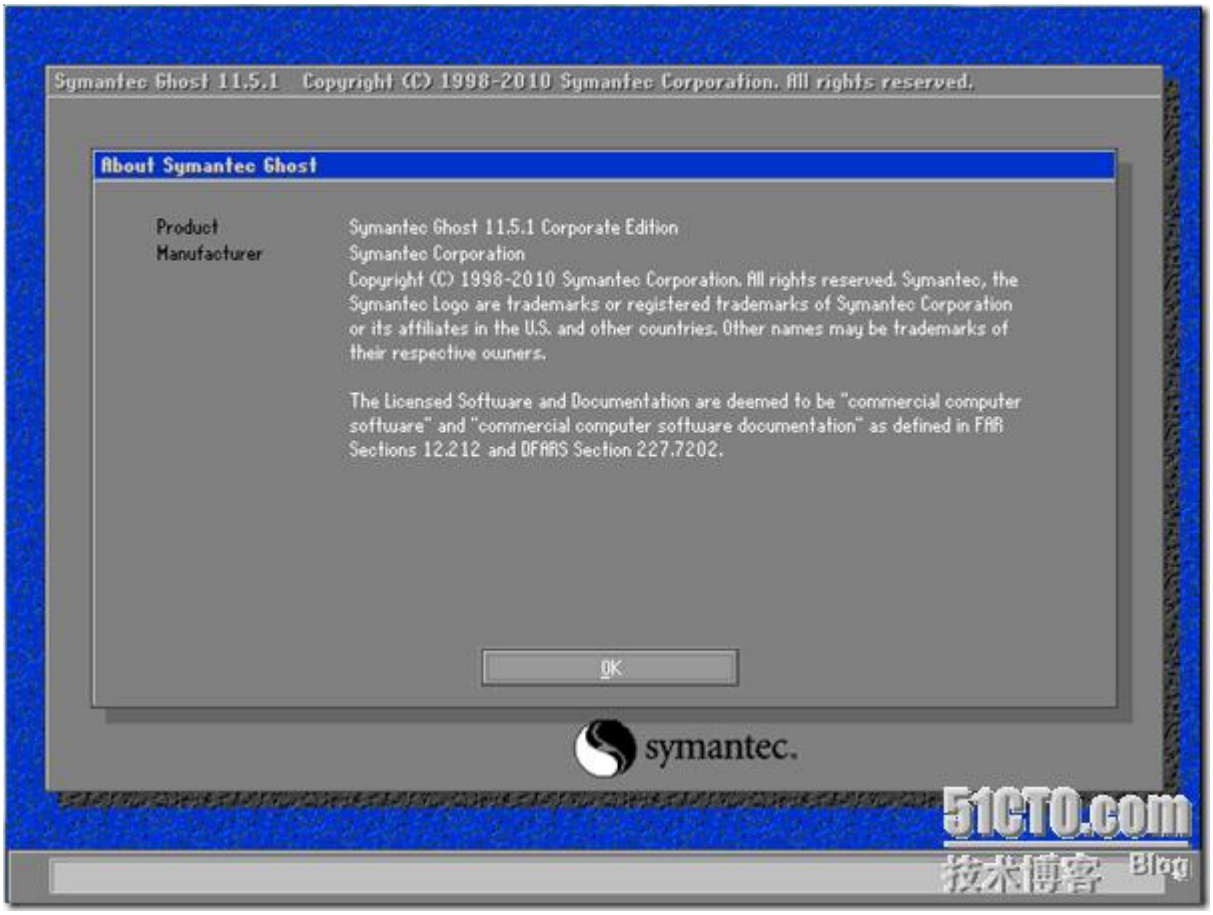
2、预装 XP 的操作系统的电脑,GHOST 克隆恢复系统可以启动吗？

3、预装 Vista 及以上级别的操作系统,GHOST 克隆恢复系统可以启动吗？

4、预装 Linux 操作系统的电脑，GHOST 克隆恢复可以启动吗？

这些问题其实都是关于 GHOST 的，大家都知道 GHOST 可以备份我们整个电脑的分区，甚至是整个硬盘，有些时候我们 GHOST 恢复过的计算机会发生不能启动的问题，为什么呢？这就是本文想和大家一起来探讨的重点

图：GHOST 11.5.1



这就是我们熟知的 GHOST 界面，在 GHOST 中常用的几个选项是

Partition to Image（将分区备份为 GHO 后缀的磁盘镜像）

Disk to Image（将磁盘备份为 GHO 后缀的磁盘镜像）

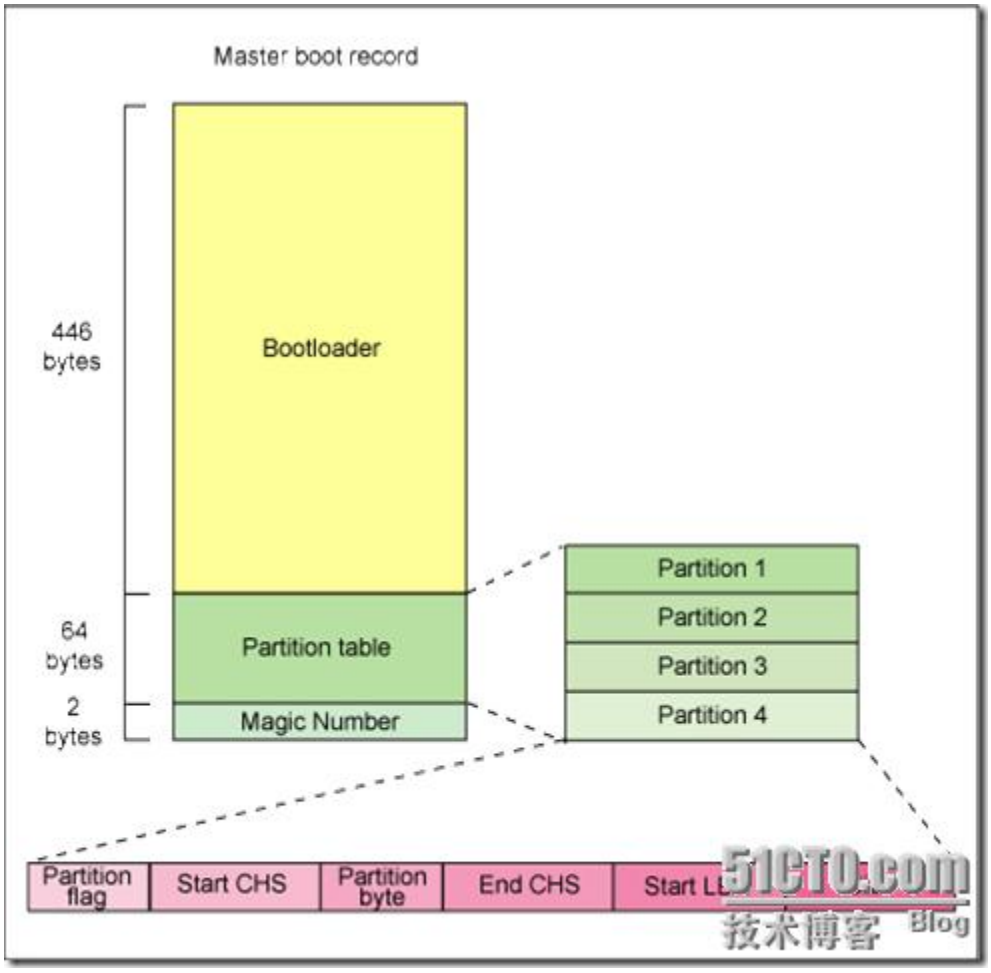
Partition from Image（将一个镜像恢复到一个分区）

Disk From Image（将一个镜像恢复到整个磁盘）

这些参数在平时可能我们用的比较多，特别是使用一些快速安装光盘的时候，例如深度技术快速安装，以及萝卜快速安装，之所以安装系统比较块是因为，制作这类系统盘的技术爱好者，已经将整个 XP 封装成了一个 GHO 文件，我们所需要做的就是插入光盘，选择一键安装即可

但是就像我们上方提出的问题一样，有些时候我们安装玩 GHOST 系统盘后发现操作系统启动不起来了？这是为什么呢？这里我们就要说说 MBR 了

Mbr（Mast Boot Record）主引导记录，这个主引导记录是存放在硬盘的 0 磁道中的，也就是硬盘的起始位置，MBR 的逻辑结构见下图



MBR 分为三部分

PART1、启动代码：也可以说是引导代码，这里面包含了各操作系统的引导信息，注意这里不存放引导文件，只存放引导代码，引导代码更具其特征来判断磁盘上存放的是什么操作系统，并且在分区表内找到激活的分区，来读取磁盘上的引导程序

我们知道已 Windows XP 为例，Windows XP 的引导程序主要有这三部分组成 NTLDR.EXE、BOOT.INI、NTDECT 引导代码的任务就是寻找到这些引导程序把控制权交给引导程序，引导程序来启动系统的内核，从来又内核来启动操作系统，在这里就不赘述了

PART2、分区表：分区表记录着我们磁盘的分区状况，一块磁盘只能有 4 个主分区，和我们在 XP 里看到的 C 盘、D 盘、E 盘、F 盘、G 盘不同的是，在一个物理磁盘上只能分 4 个主分区，在 XP 下我们看到的只能说是一个主分区或者说是几个逻辑驱动器，一般的习惯，我们会把 C 划分成主分区，其余的空间划分成扩展空间，在扩展空间中就可以创建 D\E\F\G\H\I 等等等等的逻辑驱动器

PART3、结束位

说了那么多关于 MBR 的，归根到底我们系统是否能正常启动取决于三个方面

第一、MBR 中的引导代码正确

第二、主分区被设置为活动

第三、引导程序正确

现在我们就回到上面的问题，先看看上述问题的实验结果

1、全新硬盘 GHOST 克隆恢复，系统可以启动吗？

笔者手头准备了一张深度技术 V9.0 的安装光盘，以光盘方式启动进入 GHOST 界面，笔者发现 GHOST 是不能够对分区进行还原的，因为全新硬盘并没有分区，所以全新硬盘只能使用 Disk From Image（从一个镜像恢复到整个硬盘）

恢复完毕后发现这个系统是可以正常启动的，笔者就疑惑了，难道 GHOST 可以写 MBR 信息吗？或者说深度技术的安装光盘是带有 MBR 信息的？

带着这个疑问笔者查看了 GHOST 的参数，发现 GHOST 其中的一个参数为-BI 这个参数是可以备份引导信息的，当然这一切只是推测，在后面的试验中笔者将更进一步的来描述 MBR 中的奥秘。

2、预装 XP 的操作系统的电脑,GHOST 克隆恢复系统可以启动吗？

预装 XP 的操作系统，在还原了深度 GHOST 盘后，能够正常启动，其实这也是在

笔者的预料之中，因为 XP 的引导信息一样，GHOST 默认是不会去写 MBR 中的信息，因为 GHOST 只是针对盘符中的数据来进行操作的，能够顺利还原

3、预装 Vista 及以上级别的操作系统,GHOST 克隆恢复系统可以启动吗？

预装 Vista 操作系统，可以顺利还原 XP 镜像，但是在启动时会蓝屏？同样是微软的引导代码为什么会蓝屏？笔者的疑惑开始不断增多？这是为什么？

4、预装 Linux 操作系统的电脑，GHOST 克隆恢复可以启动吗？

预装 Linux 操作系统，当还原 XP 镜像后，系统停留在 GRUB 界面，这个结果明显和 1 问题冲突，如果深度镜像写入了 MBR，那么系统应该是可以启动的

于是笔者 4 个实验做完还是一头雾水,MBR 中到底存放有哪些东西？如何查看 MBR 中的信息？对比 MBR 信息是否能找出其中的玄机呢？

DOS 分区体系的主引导记录扇区－MBR

2010-12-05 22:31:25

标签：MBR

使用 DOS 分区体系时，磁盘的第一个扇区－－也就是 0 号扇区被称为主引导记录扇区，也称为 MBR(主引导记录，Master Boot Recorder－－MBR)。当计算机启动并完成自检后，首先会寻找磁盘的 MBR 扇区并读取其中的引导记录，然后将系统控制权交给它。由此可见，如果 MBR 损坏，则后续的所有工作都无法继续进行。

1．MBR 数据结构

MBR 由 446 个字节的引导代码、64 个字节的主分区表及 2 个字节的签名值“55AA”组成。我们首先使用 Winhex 来看一下 MBR 扇区的内容，如图 2.11 所示（因为该磁盘尚未进行分区操作，所以分区表全部为空）。

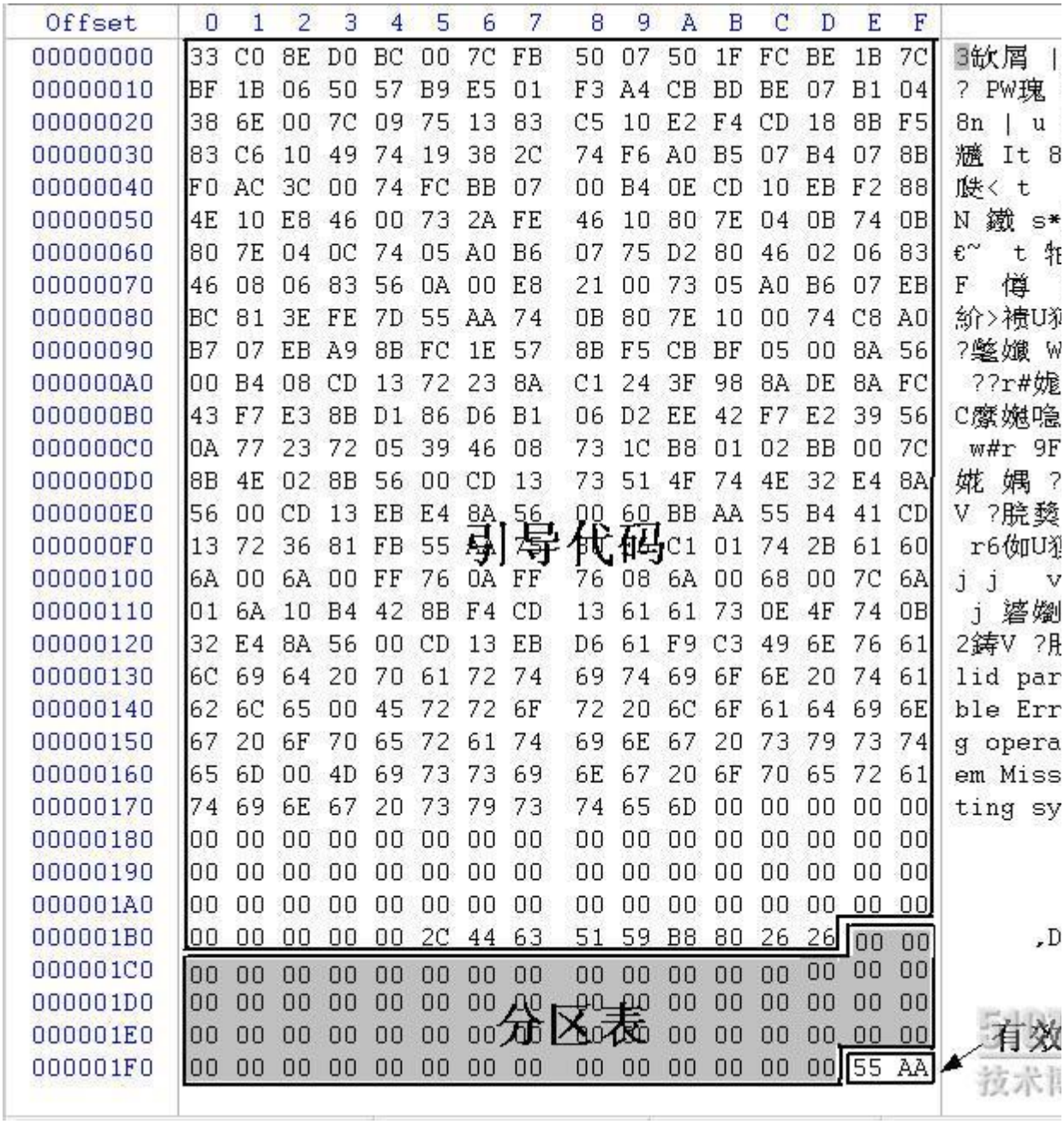


图 2.11　主引导记录扇区

可以看到，MBR 扇区由三大部分组成：

（1）引导代码。MBR 接管系统的控制权后，引导代码负责对其他代码信息进行检查（比如查看是否有“55AA”有效标记）并进一步引导系统。

（2）分区表。分区表负责描述磁盘内的分区情况。

（3）“55AA”有效标志。“55AA”标志做为有效标志以通知系统该 MBR 扇区是否有效，如果该标志丢失或损坏，磁盘将会显示为“未初始化”。

MBR 扇区的数据结构如表 2.1 所示。

表 2.1　MBR 扇区数据结构

字节偏移(十六进制)	字节数	描述

00～1BD	446	引导代码
1BE～1CD	16	分区表项 1
1CE～1DD	16	分区表项 2
1DE～1ED	16	分区表项 3
1EE～1FD	16	分区表项 4
1FE～1FF	2	签名值 (55AA)

具体含义解释如下：

1） 0x00～0x1BD：446 个字节，引导代码区域，包含一段指令，用以通知计算机如何访问分区表并定位操作系统的位置。

- 主引导代码是一段非常重要的代码，因为它是磁盘上最先装入内存并执行的代码。也正因为如此，很多引导型病毒把自己嵌入到主引导代码中，从而实现首先运行的目的。标准的 Microsoft 引导代码会在计算机启动完成自检并将控制权交给它后，读取分区表并根据分区表项的可引导标志判定哪个主分区是引导分区，找到这个分区后就查看并读取位于该分区第一个扇区的引导代码并进而启动操作系统，这部分代码会因操作系统的不同而不同。
- 利用引导代码可以实现多系统引导。很多用户需要在同一台计算机上安装超过一个的操作系统，这就需要使用多系统引导，以便能够在计算机启动时选择要进入的操作系统。多系统引导可以由两种方法实现。
  - 一种方法是，大多数用户会将 Windows 操作系统做为要安装的系统之一，Windows 可以在引导分区中设置一段代码，以允许用户选择要进入的操作系统。也就是说，MBR 中的主引导程序先加载 Windows 引导代码，然后由 Windows 引导代码再呈现给用户一个操作系统选择界面。
  - 另一种方法是改变 MBR 中的引导代码，修改后的引导代码会直接呈现给用户一个选项列表，由用户选择从哪个分区进行引导。这种方法一般会占用位于第一个分区之前的一部分未使用扇区存放程序代码。

2） 0x1BE～0x1FD：64 个字节，4 个分区表项，每个表项占用 16 个字节。每个表项描述一个 DOS 分区，最多可以描述 4 个主分区。

- 分区表项并没有顺序要求，也就是说，并不严格要求第一个分区表项对应物理位置的第一个分区，第二个表项对应第二个分区。
- 分区表也并不要求必须先使用第一个分区表项，然后依次使用后面的表项。操作系统在检索主分区表时，会完整地对四个分区表项进行完全检索，然后根据每个分区表项描述的物理位置定位分区，而不会以分区表项的先后顺序定位分区所处的先后位置关系。

图 2.12 描述了一个被划分成三个主分区的磁盘，前三个分区表项分别用来描述一个分区，最后一个分区表项未被使用。

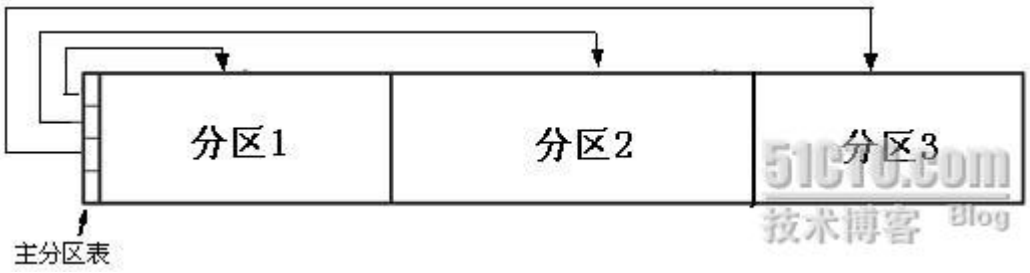
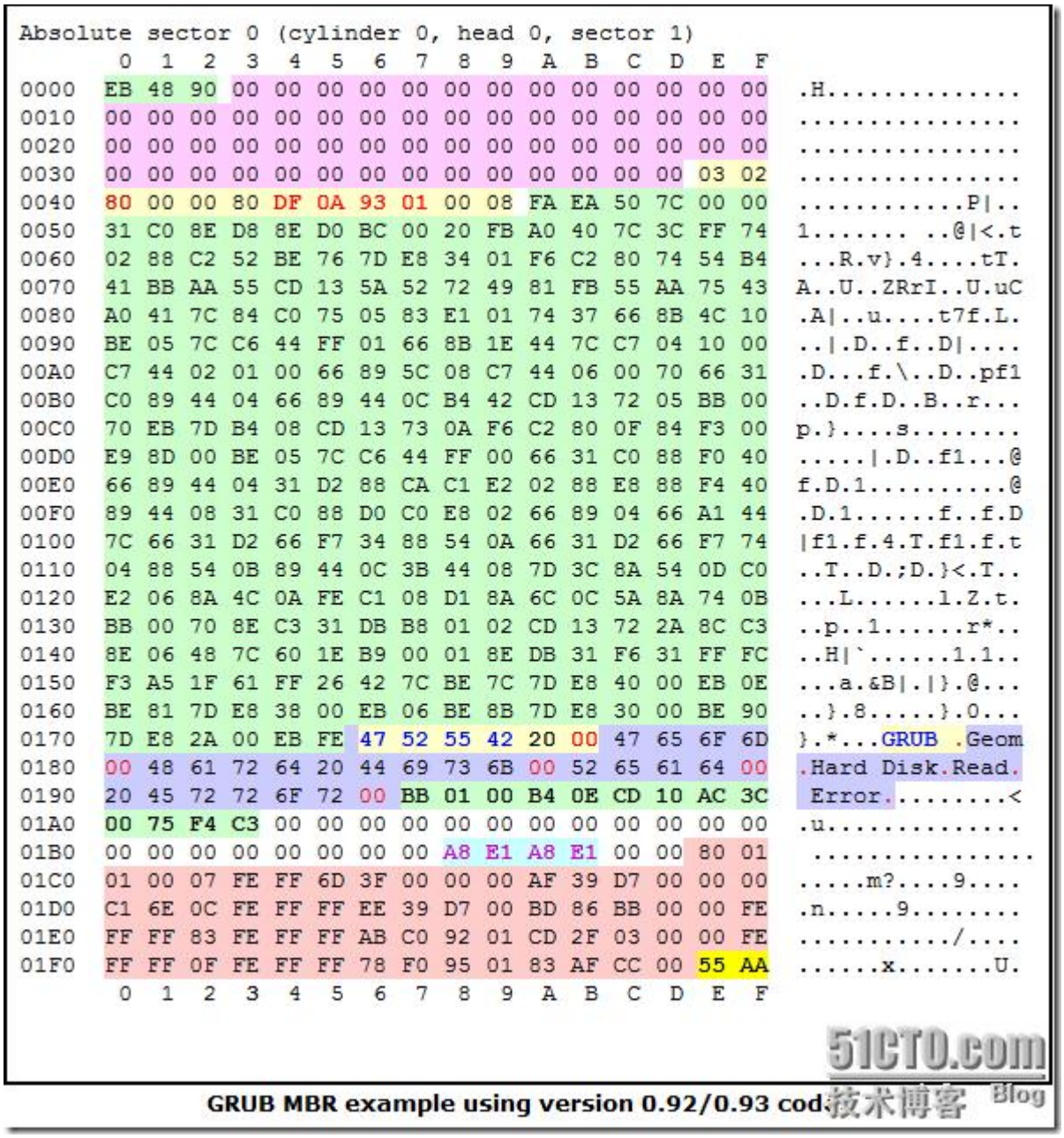


图 2.12　划分为三个主分区的磁盘

3） 0x1FE～0x1FF：2 个字节，有效结束标志“55AA”。如果没有这个签名值，操作系统会认为该磁盘没有被初始化，也就无法正常加载磁盘上的分区和解释数据。不过，只要分区和文件系统正常，某些数据恢复软件在没有这个签名值的情况下也可以正确的检测到分区表并正确地解释出所有正常的数据。



这是一张 MBR 表格，下面这些参数至关重要

在 LInux 环境下，备份 MBR 是一件相当简单的事情，只需要一条指令就搞定了。

dd if=/dev/sda of=~/MBR_`date +%F` bs=512 count=1 #需要 root 权限

这样，sda 的 MBR 就被写入用户目录下 MBR_2010-03-17 这样的文件中，执行 ls -l 可以看到，该文件大小为 512B。

MBR 损坏时，恢复也同样简单：

在 linux sruce 模式下　　dd if=~/MBR_* of=/dev/sda bs=512 count=1 #同样需要 root 权限

签：数据恢复 硬盘数据恢复 存储安全

说明：硬盘主引导记录独立于操作系统，但又和操作系统息息相关——很多时候它又是由

; 操作系统所提供的工具所生成（例外的情况是您使用了其他的分区工具，不过它又运行在

; 什么操作系统中呢？;()。

;

; 如果您安装了 Windows 98(我现在暂时不能接触 95 下的主引导记录，总不能用 95 重装我的

; 系统吧？)操作系统，那您机器上的主引导记录已经与以前的大有不同了，通过下面的分析

; 您一定能对 Windows 98 为什么要更改主引导记录有所了解——它已经开始支持扩展 Int13h

; 了！并且这个主引导记录的编程技巧更是我们应该学习的。

;

; 主引导记录包括代码、数据两部分。它在被 BIOS 中断 Int19h 装入内存后获得控制权。数据

; 部分最重要的当然是分区表了！彻底熟悉主引导记录，可以帮助我们了解系统的引导过程，

; 处理因主引导记录损坏所造成的无法引导故障，消除引导型计算机病毒，更使我们能通过

; 修改主引导记录完成我们希望的工作：如多重引导，系统加软锁等...

;

; BIOS 中断总是把主引导记录所在扇区（硬盘的 0 头 0 道 1 扇区）的内容（包括代码和数据）

; 装入内存 0000：7C00 起始的区域，然后检验该扇区内容的最后两个字节是不是“AA55”，

; 如果不是,那么对不起，Int19h 将不把控制权交给主引导记录；若是，则下面的主引导记录

; 才能获得了控制权了（Int19 通过跳转指令交转控制权）：

;

; 二进制形式的主引导记录：

0000:0600 33 C0 8E D0 BC 00 7C FB-50 07 50 1F FC BE 1B 7C 3.....|.P.P....|

0000:0610 BF 1B 06 50 57 B9 E5 01-F3 A4 CB BE BE 07 B1 04 ...PW...........

0000:0620 38 2C 7C 09 75 15 83 C6-10 E2 F5 CD 18 8B 14 8B 8,|.u...........

0000:0630 EE 83 C6 10 49 74 16 38-2C 74 F6 BE 10 07 4E AC ....It.8,t....N.

0000:0640 3C 00 74 FA BB 07 00 B4-0E CD 10 EB F2 89 46 25 <.t...........F%

0000:0650 96 8A 46 04 B4 06 3C 0E-74 11 B4 0B 3C 0C 74 05 ..F...<.t...<.t.

0000:0660 3A C4 75 2B 40 C6 46 25-06 75 24 BB AA 55 50 B4 :.u+@.F%.u$..UP.

0000:0670 41 CD 13 58 72 16 81 FB-55 AA 75 10 F6 C1 01 74 A..Xr...U.u....t

0000:0680 0B 8A E0 88 56 24 C7 06-A1 06 EB 1E 88 66 04 BF ....V$.......f..

0000:0690 0A 00 B8 01 02 8B DC 33-C9 83 FF 05 7F 03 8B 4E ....3.......N

0000:06A0 25 03 4E 02 CD 13 72 29-BE 2D 07 81 3E FE 7D 55 %.N...r).-..>.}U

0000:06B0 AA 74 5A 83 EF 05 7F DA-85 F6 75 83 BE 1A 07 EB .tZ.......u.....

0000:06C0 8A 98 91 52 99 03 46 08-13 56 0A E8 12 00 5A EB ...R..F..V....Z.

```
0000:06D0 D5 4F 74 E4 33 C0 CD 13-EB B8 00 00 80 49 12 00 .Ot.3........I..
0000:06E0 56 33 F6 56 56 52 50 06-53 51 BE 10 00 56 8B F4 V3.VVRP.SQ...V..
0000:06F0 50 52 B8 00 42 8A 56 24-CD 13 5A 58  64 10 72 PR..B.V$..ZX.d.r
0000:0700 0A 40 75 01 42 80 C7 02-E2 F7 F8 5E C3 EB 74 B7 .@u.B......^..t.
0000:0710 D6 C7 F8 B1 ED CE DE D0-A7 00 BC D3 D4 D8 B2 D9 ................
0000:0720 D7 F7 CF B5 CD B3 CA B1-B3 F6 B4 ED 00 4D 69 73 .............Mis
0000:0730 73 69 6E 67 20 6F 70 65-72 61 74 69 6E 67 20 73 sing operating s
0000:0740 79 73 74 65 6D 00 00 00-00 00 00 00 00 00 00 00 ystem...........
0000:0750 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0000:0760 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0000:0770 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0000:0780 00 00 00 8B FC 1E 57 8B-F5 CB 00 00 00 00 00 00 ......W.........
0000:0790 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0000:07A0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0000:07B0 00 00 00 00 00 00 00 00-86 D8 00 00 00 00 80 01 ................
0000:07C0 01 00 06 3F 3F FD 3F 00-00 00 41 A0 0F 00 00 00 ...??.?...A.....
0000:07D0 01 FE 05 3F FF FE 80 A0-0F 00 C0 4F 2F 00 00 00 ...?.......O/...
0000:07E0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0000:07F0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 55 AA ..............U.
;
; 反汇编结果
;
```

```
; 0000:7C00～0000:7C1A：初始化各个段寄存器、堆栈指针，最后将主引导记录在内存中搬家，腾出其所占内

; 存空间以供装入分区引导记录。

0000:7C00 33C0          XOR     AX,AX           ;AX 寄存器清 0

0000:7C02 8ED0          MOV     SS,AX           ;SS=0

0000:7C04 BC007C        MOV     SP,7C00         ;装填栈指针——SS:SP=0000:7C00

0000:7C07 FB            STI                     ;开中断(装填栈指针时为避免硬件中断引起栈混乱应关中断)

0000:7C08 50            PUSH    AX              ;

0000:7C09 07            POP     ES              ;装填附加数据段寄存器 ES=0

0000:7C0A 50            PUSH    AX              ;

0000:7C0B 1F            POP     DS              ;装填数据段寄存器 DS=0

0000:7C0C FC            CLD                     ;规定其后的串操作为正向串操作

0000:7C0D BE1B7C        MOV     SI,7C1B         ;源指针

0000:7C10 BF1B06        MOV     DI,061B         ;目的指针

0000:7C13 50            PUSH    AX              ;

0000:7C14 57            PUSH    DI              ;看看 0000:7C1A——构造一个跳转

0000:7C15 B9E501        MOV     CX,01E5         ;

0000:7C18 F3            REPZ                    ;

0000:7C19 A4            MOVSB                   ;0000:7C1B 起始的 CX 字节传送至 0000:061B 起始的区域

0000:7C1A CB            RETF                    ;跳转到 0000:061B(这是一种技巧跳转)

;

; 为直观起见，下面的地址按实际运行时的地址给出。

; 0000:061B～0000:062B：对分区表进行初步检验，一旦检测到某分区表项状态字节大于等于 80h，就通过（当
```

```
; 然，在此之前如果检测到某项分区表的状态字节小于 80h，就转错误处理。当然，如果四个分区项的状态字节
; 都为零，主引导记录就会调用 BIOS-ROM 的 INT 18h，显示"PRESS A KEY TO REBOOT"信息等待你的操作。
0000:061B BEBE07        MOV       SI,07BE           ;SI 指向第一个分区表项，这时 CX=0
0000:061E B104          MOV       CL,04             ;分区表共四个表项
0000:0620 382C          CMP       [SI],CH           ;
0000:0622 7C09          JL        062D              ;大于等于 80h 转[注意 JL 指令:(SF xor OF)=1 则转]
0000:0624 7515          JNZ       063B              ;不为 0 则[SI]一定小于 80h，只能转错误处理了!
0000:0626 83C610        ADD       SI,+10            ;为零则检查下一表项
0000:0629 E2F5          LOOP      0620              ;检查下一表项
0000:062B CD18          INT       18                ;四表项的状态字节都为 0，则系统只好调用 INT 18h 了!
;
; 0000:062D～0000:0639：检查剩余的分区表项——状态字节必须为零，否则显示错误信息“分区表无效”然
; 后当机! 拜托，微软搞错没有，怎么用中文提示信息？真 TM 傻得可爱!
; 这里还有个小 BUG，前面放行原则是只要状态字节大于等于 80h，那么如果这个字节是诸如 A0h、E5h 之类数值
; 呢？嘿嘿，这个引导记录统统认为是有效的可引导分区了!
0000:062D 8B14          MOV       DX,[SI]           ;为读分区引导记录做准备：磁头号→DH，驱动器号→DL
0000:062F 8BEE          MOV       BP,SI             ;SI→BP，保存可引导分区表项的指针
;
0000:0631 83C610        ADD       SI,+10            ;其余的分区表项还要检查检查的
0000:0634 49            DEC       CX                ;
0000:0635 7416          JZ        064D              ;CX=0 则检查顺利通过，转继续
0000:0637 382C          CMP       [SI],CH           ;
```

```
0000:0639 74F6          JZ      0631            ;为零，是合法表项，再查下一表项

;

; 0000:063B～0000:064B：执行错误处理——报告错误信息后当机

0000:063B BE1007        MOV     SI,0710         ;错误信息字符串偏移+1→SI

0000:063E 4E            DEC     SI              ;SI-1→SI

0000:063F AC            LODSB                   ;SI+1→SI

0000:0640 3C00          CMP     AL,00           ;

0000:0642 74FA          JZ      063E            ;AL=0 则表明一条错误信息显示完毕,系统陷入一个死循环

0000:0644 BB0700        MOV     BX,0007         ;字符方式显示

0000:0647 B40E          MOV     AH,0E           ;

0000:0649 CD10          INT     10              ;以写电传方式显示信息(只显示一个字符)

0000:064B EBF2          JMP     063F            ;显示下一个字符，直到遇到提示信息结束为止

;

; 0000:064D～0000:0662：判断可引导分区的分区类型，然后转相应处理程序。

0000:064D 894625        MOV     [BP+25],AX      ;BP=指向第一个可引导分区表项的指针，这时 AX=0000h

                                                ;使用长度最短的指令将[BP+25]起始的两个单元清零

                                                ;这两个单元将被用来存放中间变量

0000:0650 96            XCHG    SI,AX           ;此时 SI 清零的最佳指令选择(仅 1 字节)，将服务于 0000:06B8

0000:0651 8A4604        MOV     AL,[BP+04]      ;取分区类型(本例是“06”喽——FAT16 主 DOS 分区)

0000:0654 B406          MOV     AH,06           ;为扩展 INT 13h 无法使用做好更改分区类型的准备

0000:0656 3C0E          CMP     AL,0E           ;0Eh：需要用扩展 INT 13h 访问的 FAT16 主 DOS 分区

0000:0658 7411          JZ      066B            ;0Eh 类型的分区转 066Bh
```

```
0000:065A B40B          MOV     AH,0B           ;
0000:065C 3C0C          CMP     AL,0C           ;0Ch：需要用扩展 INT 13h 访问的 FAT32 分区
0000:065E 7405          JZ      0665            ;0Ch 类型的分区转 0665h 先行预处理
0000:0660 3AC4          CMP     AL,AH           ;0Bh：用传统 INT 13h 就可以访问的 FAT32 分区
0000:0662 752B          JNZ     068F            ;其他类型的分区转 068Fh
;
; 0000:0664～0000:06A1：根据分区类型和分区表表项内容进行读取分区引导记录前的处理工作
0000:0664 40            INC     AX              ;★★★0Bh 类型的分区由此开始处理，此条指令用意是清 ZF 位
0000:0665 C6462506      MOV     BYTE PTR [BP+25],06 ;★★★0Ch 类型的分区由此开始处理
                                                ;为什么取值 06，一时没有自圆我说的解释，请耐心几天吧。
0000:0669 7524          JNZ     068F            ;请注意上面指令对 ZF 位的影响：0Bh 类型分区转，0Ch 则不转
; 0000:066B～0000:068C 这段代码仅当分区类型是 0Ch、0Eh 才有获得执行的机会
0000:066B BBAA55        MOV     BX,55AA         ;★★★0Eh 类型的分区由此开始处理
0000:066E 50            PUSH    AX              ;
0000:066F B441          MOV     AH,41           ;扩展 INT 13h 功能，检测 BIOS 是否已经支持扩展 INT13h
0000:0671 CD13          INT     13              ;入口参数：BX=55AAh,DL=驱动器号,AH=41h
0000:0673 58            POP     AX              ;执行完恢复 AX 为 060Eh
0000:0674 7216          JB      068C            ;不支持则转
0000:0676 81FB55AA      CMP     BX,AA55         ;
0000:067A 7510          JNZ     068C            ;扩展 INT13h 不可用也转
0000:067C F6C101        TEST    CL,01           ;测试扩展盘访问是否被支持
0000:067F 740B          JZ      068C            ;不支持还转
```

```
; 因为扩展 INT13h 方式读盘与标准 INT13h 方式读盘有很大差别，所以 0000:0686 处指令修改其后的代码以保证按

; 照扩展读方式读分区引导扇区时能正确跳转到相应的处理程序中。

0000:0681 8AE0          MOV     AH,AL               ;分区类型→AH

0000:0683 885624        MOV     [BP+24],DL          ;保存驱动器号→[BP+24]

0000:0686 C706A106EB1E MOV      WORD PTR [06A1],1EEB ;修改 0000:06A1 处代码为"JMP 06C1"

0000:068C 886604        MOV     [BP+04],AH          ;注意：如果扩展 INT13h 不能使用则 A 改分区类型为 06，但如果

                                                    ;扩展 INT13h 能使用，则仍保持原分区类型不变

0000:068F BF0A00        MOV     DI,000A             ;★★★其它类型分区由此开始处理。此条指令初始化计数器

0000:0692 B80102        MOV     AX,0201             ;AH：读操作，AL：读取 1 个扇区的内容

0000:0695 8BDC          MOV     BX,SP               ;SP=7C00→BX，指定分区引导记录装入内存的位置偏移

0000:0697 33C9          XOR     CX,CX               ;CX 清零

0000:0699 83FF05        CMP     DI,+05              ;注意 5

0000:069C 7F03          JG      06A1                ;大于则转去读由分区表指定的分区引导扇区

0000:069E 8B4E25        MOV     CX,[BP+25]          ;小于则证明所读分区表指定的引导扇区无合法的引导记录，

                                                    ;改按？？？再读，毕竟多一种选择多一次机会嘛！;)

; 以下标有①②者请注意它们的地址都是一样的，就是说实际运行中只可能是二者之一，但为了分析之方便，我

; 把两者都列了出来以供对比，阅读时千万别看成是两条指令了啊！

①0000:06A1 034E02      ADD     CX,[BP+02]          ;获取分区引导扇区所在的柱面号和物理扇区号

②0000:06A1 EB1E          JMP     06C1              ;如果分区类型是 0Ch、0Eh 而且扩展读能使用则执行该指令

;

; 0000:06A4：将可引导分区的分区引导记录装入内存指定区域

; 入口参数：AH=功能号，02 为读盘操作；AL=一次读取的扇区数
```

```
;                   ES:BX=读入内存的起始地址

;                   CH=10 位柱面号的低 8 位；CL：高两位是 10 位柱面号的高两位，低 6 位是物理扇区号

;                   DH=磁头号；DL=驱动器号，最高位(即位 7)为 0 是软盘，为 1 是硬盘

0000:06A4 CD13          INT     13                  ;读分区引导记录到 0000:7C00 起始的区域

;

;

0000:06A6 7229          JB      06D1                ;不成功转

0000:06A8 BE2D07        MOV     SI,072D             ;错误信息字符串偏移→SI

0000:06AB 813EFE7D55AA CMP      WORD PTR [7DFE],AA55 ;分区引导记录合法吗?

0000:06B1 745A          JZ      070D                ;合法则转（这是主引导记录唯一的正常出口）

0000:06B3 83EF05        SUB     DI,+05              ;不合法则为换读其他扇区做准备

0000:06B6 7FDA          JG      0692                ;只有一次换读扇区的机会!

;

; 0000:06B8～0000:06BF：错误预处理

0000:06B8 85F6          TEST    SI,SI               ;测试 SI 值是否为 0，其意义在于确定该显示哪条信息

0000:06BA 7583          JNZ     063F                ;不为 0 则转错误处理，显示“Missing operating system”

0000:06BC BE1A07        MOV     SI,071A             ;错误信息字符串偏移→SI

0000:06BF EB8A          JMP     064B                ;转错误处理，显示“加载操作系统时出错”

;

; 0000:06C1～0000:06CF：整理扩展读所需入口参数，然后调用扩展读子程序

; 这段代码只有在以扩展读方式读取分区引导记录时才有机会获得执行

0000:06C1 98            CBW                         ;转换字节 AL 为字 AX，执行后，AX 中是一次要读的扇区数
```

```
0000:06C2 91              XCHG    CX,AX             ;AX→CX，CX→AX，执行后，CX 中是一次要读的扇区数

0000:06C3 52              PUSH    DX                  ;

0000:06C4 99              CWD                           ;将字 AX 转换为双字→DX,AX

0000:06C5 034608        ADD       AX,[BP+08]       ;

0000:06C8 13560A        ADC       DX,[BP+0A]       ;执行后，DX:AX=LBA 绝对物理扇区号

0000:06CB E81200        CALL    06E0                ;调用扩展读子程序

0000:06CE 5A              POP       DX                  ;

0000:06CF EBD5          JMP       06A6                ;

;

; 0000:06D1～0000:06D8 分区引导记录装入失败时的处理

0000:06D1 4F              DEC       DI                  ;计数器减 1

0000:06D2 74E4          JZ        06B8                ;五次读盘均未成功则转错误处理(注意这时 SI=0)

0000:06D4 33C0          XOR       AX,AX             ;置功能号

0000:06D6 CD13          INT       13                  ;复位磁盘系统

0000:06D8 EBB8          JMP       0692                ;再读

;

;

0000:06DA 00 00 80 49 12 00 ...I..

;

; 0000:06E0～0000:070C：使用扩展 INT 13h 功能读取分区引导记录的子程序

; 调用时，SP=7BFE。这段程序利用压栈寄存器方式构造了一个磁盘地址包，请注意体会。另外，0000:06FC 处

; 的一条指令就释放了几乎全部由本段程序占用的栈空间，构思之巧妙，绝对需要我们学习！
```

```
; 所以，分析该段程序，一个重点应放在栈的变化上。

0000:06E0 56              PUSH    SI                  ;保存 SI——注意，这次压栈并不构造磁盘地址包

0000:06E1 33F6            XOR     SI,SI               ;清零

0000:06E3 56              PUSH    SI                  ;

0000:06E4 56              PUSH    SI                  ;

0000:06E5 52              PUSH    DX                  ;

0000:06E6 50              PUSH    AX                  ;以上四条指令压栈的是扇区 LBA 号码*2

0000:06E7 06              PUSH    ES                  ;压栈内存目标缓冲区首址段址

0000:06E8 53              PUSH    BX                  ;压栈内存目标缓冲区首址偏移

0000:06E9 51              PUSH    CX                  ;压栈所读扇区数

0000:06EA BE1000          MOV     SI,0010             ;注意 SI 的高 8 位对应着磁盘地址包的保留字节，必须为 0

0000:06ED 56              PUSH    SI                  ;压栈磁盘地址包包长，执行完本条指令一个包已经构造完毕

0000:06EE 8BF4            MOV     SI,SP               ;规定磁盘地址包偏移指针，这时 SP=7BEA

0000:06F0 50              PUSH    AX                  ;保存 AX

0000:06F1 52              PUSH    DX                  ;保存 DX

0000:06F2 B80042          MOV     AX,4200             ;置扩展读功能号

0000:06F5 8A5624          MOV     DL,[BP+24]          ;取驱动器号，参照 0000:0683

; 入口参数：AH=功能号，02 为读盘操作；DL=驱动器号

;                 DS:SI=16 字节磁盘地址包——第 0 字节：包长度(固定为 10h)；第 1 字节：保留，必须为 0；

;                 第 2、3 字节：所读扇区数；第 4～5 字节：内存目标缓冲区首址偏移；

;                 第 6～7 字节：内存目标缓冲区首址段址； 第 8～15 字节：扇区 LBA 号码

; 出口参数：成功则 AH=0；错误则 AH=错误代码
```

```
0000:06F8 CD13          INT     13                  ;执行扩展读操作
0000:06FA 5A            POP     DX                  ;
0000:06FB 58            POP     AX                  ;
0000:06FC 8D6410        LEA     SP,[SI+10]      ;7BEA+10h=7BFA→SP（注意是取偏移而不是取单元内容）
0000:06FF 720A          JB      070B                ;扩展读不成功转
0000:0701 40            INC     AX                  ;
0000:0702 7501          JNZ     0705                ;
0000:0704 42            INC     DX                  ;AX 加 1 溢出时（比如 0FFFFh+1）DX 才加 1
0000:0705 80C702        ADD     BH,02               ;调整 BX，使偏移量增加 512 字节（刚好一扇区）
0000:0708 E2F7          LOOP    0701                ;0701～0708 一段代码暂未明白其真实意图!
0000:070A F8            CLC                         ;
0000:070B 5E            POP     SI                  ;
0000:070C C3            RET                         ;
;
; 0000:070D：中继跳转
0000:070D EB74          JMP     0783                ;
;
; 070F～0745 是错误信息！果然是中文 Windows98 生成的主引导记录，所以我要特别
; “感谢”微软这个傻 B，真难为它竟然用中文表述前两个信息！可惜真需显示的时
; 候鬼才能看懂是什么呢！！！我 K!——耍弄我们耶！？
; 070F～0718：“分区表无效”中文信息
; 071A～072B：“加载操作系统时出错”中文信息
```

```
; 072D～0744：“Missing operating system”英文信息

0000:070F B7 .

0000:0710 D6 C7 F8 B1 ED CE DE D0-A7 00 BC D3 D4 D8 B2 D9 ................

0000:0720 D7 F7 CF B5 CD B3 CA B1-B3 F6 B4 ED 00 4D 69 73 .............Mis

0000:0730 73 69 6E 67 20 6F 70 65-72 61 74 69 6E 67 20 73 sing operating s

0000:0740 79 73 74 65 6D 00 00 00-00 00 00 00 00 00 00 00 system..........

0000:0750 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

0000:0760 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

0000:0770 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

0000:0780 00 00 00 ...

;

; 0000:0783～0000:0789：控制权移交

0000:0783 8BFC          MOV     DI,SP           ;

0000:0785 1E            PUSH    DS              ;

0000:0786 57            PUSH    DI              ;构造一个跳转地址

0000:0787 8BF5          MOV     SI,BP           ;

0000:0789 CB            RETF                    ;交控制权给分区引导记录（0000:7C00）

;

;

0000:078A 00 00 00 00 00 00 ......

0000:0790 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

0000:07A0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
```

;

; 07B8～07BB 四个字节的内容用于什么呢？（不同机器此四字节均不同）

; 07BE～07FD 为分区表，内含四个分区表项(每表项 10h 字节)

0000:07B0 00 00 00 00 00 00 00 00-86 D8 00 00 00 00 80 01 ....

0000:07C0 01 00 06 3F 3F FD 3F 00-00 00 41 A0 0F 00 00 00 ...??.?...A.....

0000:07D0 01 FE 05 3F FF FE 80 A0-0F 00 C0 4F 2F 00 00 00 ...?.......O/...

0000:07E0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ....

0000:07F0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 55 AA ....U.

*1：因为物理扇区号总是从 1 排列而起

*2：由此可见，就是使用 LBA 扩展读的功能，主引导记录却限制了分区引导扇区必须在 LBA 绝对物理扇区

0FFFFFFFFh 之前才有可能从该分区引导系统！

文章作者 北亚数据恢复公司 文章来源 [url]http://www.bysjhf.cn[/url]

在分析 MBR 的结构之前，先有来看看计算机的引导顺序(System Boot Sequence)

Step 1. 内部电源打开，初始化，等待一小段时间用来产生稳定的电流。如果主板芯片和 CPU 收到了不符合规定的电流，将自动产生一个 RESET 信号。在主板没有收到电源的 Power Good 信号之前，重复步骤 1。

Step 2. 执行 BIOS 中 0FFF0h 处的代码。这里只有一条 JMP 指令，将跳转到真正的 BIOS 启动程序处。

Step 3. BIOS 开始加电自检(Power-On Self Test，POST)，如果出现错误，启动停止。成功的话执行 INT 19h(SYSTEM - BOOTSTRAP LOADER)

Step 4. BIOS 开始寻找显卡，找到的话将执行显卡的 BIOS。接着显卡初始化，将显示一段显卡信息，我们开机

看到的第一屏就是它。

Step 5. BIOS 开始执行所有其他设备的 BIOS，包括软驱，硬盘等。

Step 6. BIOS 显示启动信息

Step 7. BIOS 开始额外的检测。一般有内存检测，如果内存有问题，将显示错误消息。

Step 8. BIOS 探测所有的硬件，将显示如硬盘/光区信息等

Step 9. BIOS 给出一个已知硬件的列表

Step 10. BIOS 按照设置的驱动器顺序找驱动器，如果驱动器存在的话继续找启动扇区，软驱/硬盘的启动扇区都在 0 柱 0 头 1 扇区(cylinder 0, head 0, sector 1)

Step 11. 将启动扇区读到内存 0000:7c00 处，接着 INT 19h 开始执行 0000:7c00 处代码

Step 12. 如果找不到驱动器，系统显示错误信息并停止。通常是"No boot device"或"NO ROM BASIC -SYSTEM HALTED"

上面是冷启动的过程，热启动将从步骤 8 开始

磁盘的启动扇区就是主引导记录(Master Boot Record)，包括 0 柱 0 头 1 扇区的 512 个字节，它的任务是完成 BIOS 到操作系统的交接。

MBR 的大体结构:

偏移 内容

0000 MBR 程序代码

01BE 分区表

01FE 结束标志

分区表结构

BYTE

1 如果是引导分区，就是 80H，如果不是，就是 00H

2-4 是该分区的起始扇区号

5 标志字节，比如 05 表示扩展分区

6-8 该分区的终止扇区号

9-12 该分区已使用的扇区数

13-16 该分区总共占用的扇区数

这是从我的硬盘上提取的 MBR（硬盘是 Maxtor 的金钻 20G，netfay 的电脑早过时了:P），不同型号的硬盘 MBR 稍有不同，不过功能都是一样的

0000 33 C0 8E D0 BC 00 7C FB-50 07 50 1F FC BE 1B 7C 3.....|.P.P....|

```
0010 BF 1B 06 50 57 B9 E5 01-F3 A4 CB BE BE 07 B1 04 ...PW...........
0020 38 2C 7C 09 75 15 83 C6-10 E2 F5 CD 18 8B 14 8B 8,|.u...........
0030 EE 83 C6 10 49 74 16 38-2C 74 F6 BE 10 07 4E AC ....It.8,t....N.
0040 3C 00 74 FA BB 07 00 B4-0E CD 10 EB F2 89 46 25 <.t...........F%
0050 96 8A 46 04 B4 06 3C 0E-74 11 B4 0B 3C 0C 74 05 ..F...<.t...<.t.
0060 3A C4 75 2B 40 C6 46 25-06 75 24 BB AA 55 50 B4 :.u+@.F%.u$..UP.
0070 41 CD 13 58 72 16 81 FB-55 AA 75 10 F6 C1 01 74 A..Xr...U.u....t
0080 0B 8A E0 88 56 24 C7 06-A1 06 EB 1E 88 66 04 BF ....V$.......f..
0090 0A 00 B8 01 02 8B DC 33-C9 83 FF 05 7F 03 8B 4E .......3.......N
00A0 25 03 4E 02 CD 13 72 29-BE 59 07 81 3E FE 7D 55 %.N...r).Y..>.}U
00B0 AA 74 5A 83 EF 05 7F DA-85 F6 75 83 BE 2E 07 EB .tZ.......u.....
00C0 8A 98 91 52 99 03 46 08-13 56 0A E8 12 00 5A EB ...R..F..V....Z.
00D0 D5 4F 74 E4 33 C0 CD 13-EB B8 00 00 80 08 10 16 .Ot.3...........
00E0 56 33 F6 56 56 52 50 06-53 51 BE 10 00 56 8B F4 V3.VVRP.SQ...V..
00F0 50 52 B8 00 42 8A 56 24-CD 13 5A 58 8D 64 10 72 PR..B.V$..ZX.d.r
0100 0A 40 75 01 42 80 C7 02-E2 F7 F8 5E C3 EB 74 B7 .@u.B......^..t.
0110 D6 C7 F8 B1 ED CE DE D0-A7 A1 A3 B0 B2 D7 B0 B3 ................
0120 CC D0 F2 CE DE B7 A8 BC-CC D0 F8 A1 A3 00 BC D3 ................
0130 D4 D8 B2 D9 D7 F7 CF B5-CD B3 CA B1 B3 F6 CF D6 ................
0140 B4 ED CE F3 A1 A3 B0 B2-D7 B0 B3 CC D0 F2 CE DE ................
0150 B7 A8 BC CC D0 F8 A1 A3-00 C8 B1 C9 D9 B2 D9 D7 ................
0160 F7 CF B5 CD B3 00 00 00-00 00 00 00 00 00 00 00 ................
```

```
0170 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0180 00 00 00 8B FC 1E 57 8B-F5 CB 00 00 00 00 00 00 ......W.........
0190 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
01A0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
01B0 00 00 00 00 00 2C 44 63-B5 D7 B5 D7 00 00 80 01 .....,Dc........
01C0 01 00 0B FE 7F FD 3F 00-00 00 3F 04 7D 00 00 00 ......?...?.}...
01D0 41 FE 0C FE FF FF 7E 04-7D 00 7D 9B E5 01 00 00 A.....~.}.}.....
01E0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
01F0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 55 AA ..............U.
```

由于程序代码从 0000:7C00 开始，下面看反编译的结果(经过修改)

```
7C00 33C0 XOR AX,AX ;AX=0
7C02 8ED0 MOV SS,AX ;SS=0
7C04 BC007C MOV SP,7C00 ;SP=7C00
7C07 FB STI ;中断允许
7C08 50 PUSH AX
7C09 07 POP ES ;ES=0
7C0A 50 PUSH AX
7C0B 1F POP DS ;DS=0
7C0C FC CLD ;字符串操作方向：从低到高
7C0D BE1B7C MOV SI,7C1B ;源地址 DS:SI=0000:7C1B
```

```
7C10 BF1B06 MOV DI,061B ;目的地址 ES:DI=0000:061B

7C13 50 PUSH AX

7C14 57 PUSH DI

7C15 B9E501 MOV CX,01E5 ;共 1E5h 个字节

7C18 F3 REPZ

7C19 A4 MOVSB ;将 MBR 从 0000:7C00 移动到 0000:0600

7C1A CB RETF ;跳转到 0000:061B 处


PARTITION_SEARCH_LOOP:


061B BEBE07 MOV SI,07BE ;SI 指向分区表的开始

061E B104 MOV CL,04 ;循环 4 次，硬盘最多 4 个主分区

0620 382C CMP [SI],CH

0622 7C09 JL ACTIVE_PARTITION_FOUND

;分区是活动分区

0624 7515 JNZ INVALID_PARTITION_TABLE

;无效的分区表

0626 83C610 ADD SI,+10 ;每个分区占用 16 个字节，SI 指向下一个分区

0629 E2F5 LOOP PARTITION_SEARCH_LOOP


062B CD18 INT 18 ;分区表搜索完，无活动分区，INT 18h=DISKLESS BOOT HOOK
```

```
ACTIVE_PARTITON_FOUND:

062D 8B14 MOV DX,[SI] ;下面的搜索保证只存在一个活动分区，否则分区表无效
062F 8BEE MOV BP,SI ;找到的引导分区标志和开始地址分别存入 DX,BP

ONLY_ONE_ACTIVE_PARTITON_SEARCH_LOOP:

0631 83C610 ADD SI,+10
0634 49 DEC CX
0635 7416 JZ GOOD_PARTITION_TABLE ;搜索完毕，剩下的分区中无活动分区，分区表正常
0637 382C CMP [SI],CH
0639 74F6 JZ ONLY_ONE_ACTIVE_PARTITON_SEARCH_LOOP ;如果还有活动分区则继续向下执行

INVALID_PARTITION_TABLE:

063B BE1007 MOV SI,0710 ;SI 指向要显示的错误信息处

HANG_MACHINE_LOOP:

063E 4E DEC SI

DISPLAY_ERROR_MESSAGE_LOOP:
```

```
063F AC LODSB

0640 3C00 CMP AL,00

0642 74FA JZ HANG_MACHINE_LOOP

;到字符串尾时进入死循环，停止运行

0644 BB0700 MOV BX,0007

0647 B40E MOV AH,0E

0649 CD10 INT 10 ;显示错误信息


DISPLAY_ERROR_MESSAGE_LOOP_ALIAS:


064B EBF2 JMP DISPLAY_ERROR_MESSAGE_LOOP


GOOD_PARTITION_TABLE:


064D 894625 MOV [BP+25],AX

;tmpvar=BP+25 处清零，作为临时变量

0650 96 XCHG SI,AX ;SI=0

0651 8A4604 MOV AL,[BP+04]

;读分区类型入 AL

0654 B406 MOV AH,06

0656 3C0E CMP AL,0E ;类型 WIN95: DOS 16-bit FAT, LBA-mapped
```

```
0658 7411 JZ TYPE_WIN95_DOS_16BIT_FAT_LBA

065A B40B MOV AH,0B

065C 3C0C CMP AL,0C ;类型 WIN95 OSR2 32-bit FAT, LBA-mapped

065E 7405 JZ TYPE_WIN95_OSR2_32BIT_FAT_LBA

0660 3AC4 CMP AL,AH ;类型 WIN95 OSR2 32-bit FAT

0662 752B JNZ TYPE_DEFAULT

0664 40 INC AX ;AX=0B0C

TYPE_WIN95_OSR2_32BIT_FAT_LBA:

0665 C6462506 MOV BYTE PTR [BP+25],06

;tmpvar=06

0669 7524 JNZ TYPE_DEFAULT

;这里有点问题，这个转移应该肯定不成立？

TYPE_WIN95_DOS_16BIT_FAT_LBA:

066B BBAA55 MOV BX,55AA

066E 50 PUSH AX

066F B441 MOV AH,41
```

```
0671 CD13 INT 13 ;int 13h扩展功能的检测，IBM/MS INT 13 Extensions - INSTALLATION CHECK

0673 58 POP AX

0674 7216 JB INT13H_EXTENSION_UNSUPPORTED

;CF=1 - 不支持 int 13h 扩展功能

0676 81FB55AA CMP BX,AA55 ;BX 不为 AA55 - 不支持 int 13h 扩展功能

067A 7510 JNZ INT13H_EXTENSION_UNSUPPORTED

067C F6C101 TEST CL,01 ;CL 不为 1 - 不支持 int 13h 扩展功能

067F 740B JZ INT13H_EXTENSION_UNSUPPORTED

0681 8AE0 MOV AH,AL ;AH=0E

0683 885624 MOV [BP+24],DL

;tmpvar=DL,引导分区标志

0686 C706A106EB1E MOV WORD PTR [06A1],1EEB

;改 06A1 处指令为 PUSH DS; JMP NEW_LOCATION_1



INT13H_EXTENSION_UNSUPPORTED:



068C 886604 MOV [BP+04],AH

;如果支持的话置分区类型为 0E(类型 WIN95: DOS 16-bit FAT, LBA-mapped)

;否则为 06(类型 DOS 3.31+ 16-bit FAT over 32M)



TYPE_DEFAULT:
```

```
068F BF0A00 MOV DI,000A

READ_SECTOR_LOOP:

0692 B80102 MOV AX,0201
0695 8BDC MOV BX,SP ;BX 设置为 7C00
0697 33C9 XOR CX,CX ;CX=0
0699 83FF05 CMP DI,+05
069C 7F03 JG NEW_LOCATION_0

069E 8B4E25 MOV CX,[BP+25]

NEW_LOCATION_0:

06A1 034E02 ADD CX,[BP+02]
06A4 CD13 INT 13 ;将活动分区的起始扇区读到 0000:7C00

NEW_LOCATION_1:

06A6 7229 JB READ_SECTOR_ERROR
;CF=1 - 错误
06A8 BE5907 MOV SI,0759
```

```
06AB 813EFE7D55AA CMP WORD PTR [7DFE],AA55
;扇区结束标志是否正确?
06B1 745A JZ READ_SECTOR_SUCCEEDED
;正确
06B3 83EF05 SUB DI,+05 ;DI=DI-5
06B6 7FDA JG READ_SECTOR_LOOP

06B8 85F6 TEST SI,SI
06BA 7583 JNZ DISPLAY_ERROR_MESSAGE_LOOP:
;显示错误信息：缺少操作系统

06BC BE2E07 MOV SI,072E
06BF EB8A JMP DISPLAY_ERROR_MESSAGE_LOOP_ALIAS
;显示错误信息：加载操作系统时发生错误。

070D EB74 JMP CONTINUE_KOAD_OS

0783 8BFC MOV DI,SP
;DI=7C00
0785 1E PUSH DS
0786 57 PUSH DI
```

0787 8BF5 MOV SI,BP

0789 CB RETF ;转到执行 0000:7C00 处的语句，即操作系统的引导程序

80 代码：我们看到有一个 80 代码，80 代表着第一个被设置为激活的分区，当分区被设为激活那么分区的最前面就会加上 16 进制数据 80，MBR 就更具这个 80 来判断是从哪个主分区来进行启动

55 AA 代码：在表格的最后看到 55 AA 这两个 16 进制代码，这表示，引导代码正常，分区表正常可以正常启动

11B8-11BB 代码：XP 的引导代码

11BE-11FF 代码：Linux 引导代码

这是一张例图,在计算机中我们是如何查看 MBR 代码的呢？启动到纯 DOS 环境下，使用 DEBUG 命令汇编一段小代码

```
A&gt;DEBUG
-A 100
XXXX:0100 MOV AX, 0201
XXXX:0103 MOV BX, 1000
XXXX:0106 MOV CX, 0001
XXXX:0109 MOV DX, 0080
XXXX:010C INT 13
XXXX:010E INT 3
XXXX:010F
-G=100
-D 11BE 11FF （显示分区表数据）
```

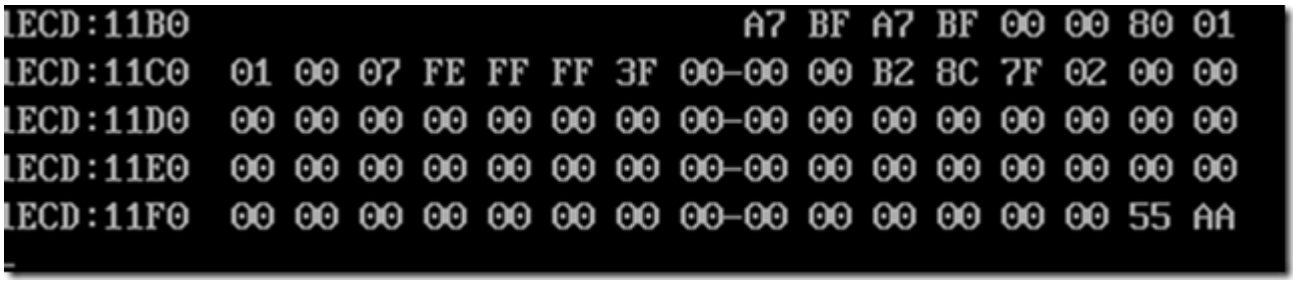
首先我们看第各个试验的 MBR 代码对比图

1、全新硬盘 GHOST 克隆恢复

全新硬盘克隆前

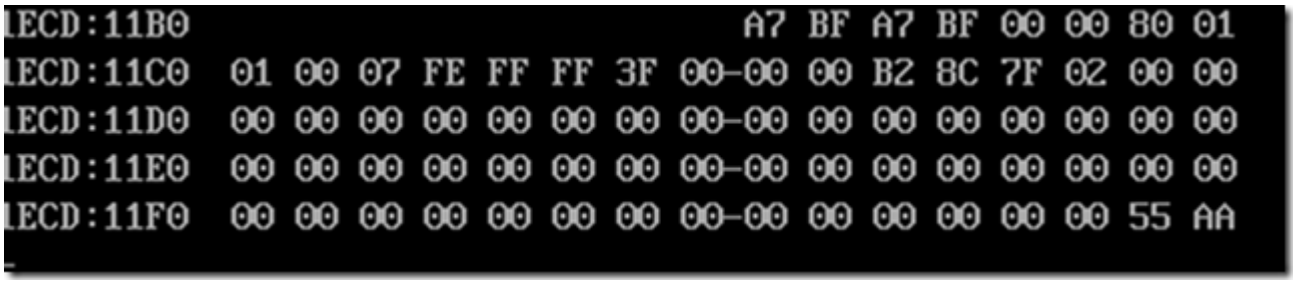
MBR 全部为 0

克隆后

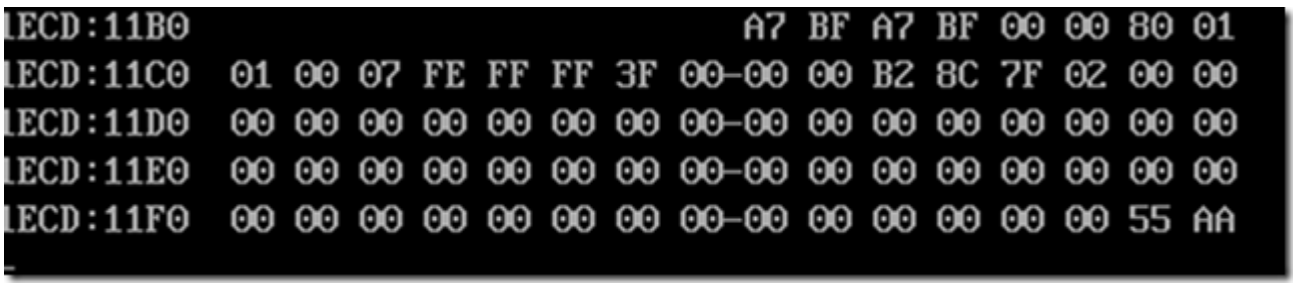


2、预装 XP 的操作系统的电脑

原始 XP



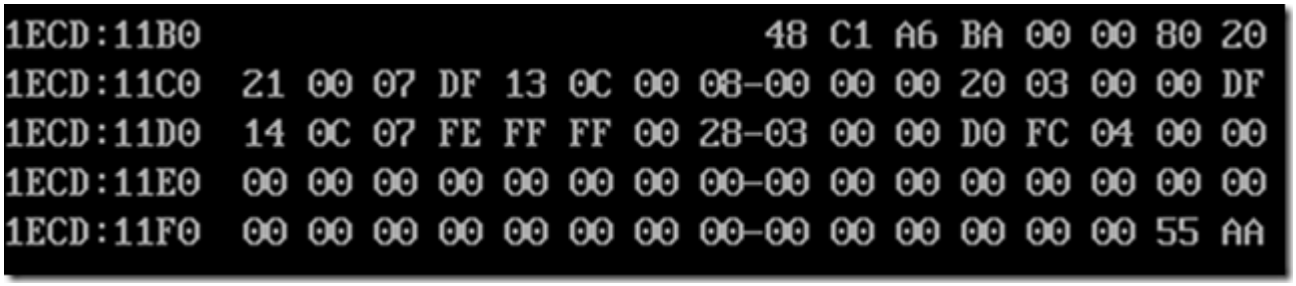
克隆后 XP



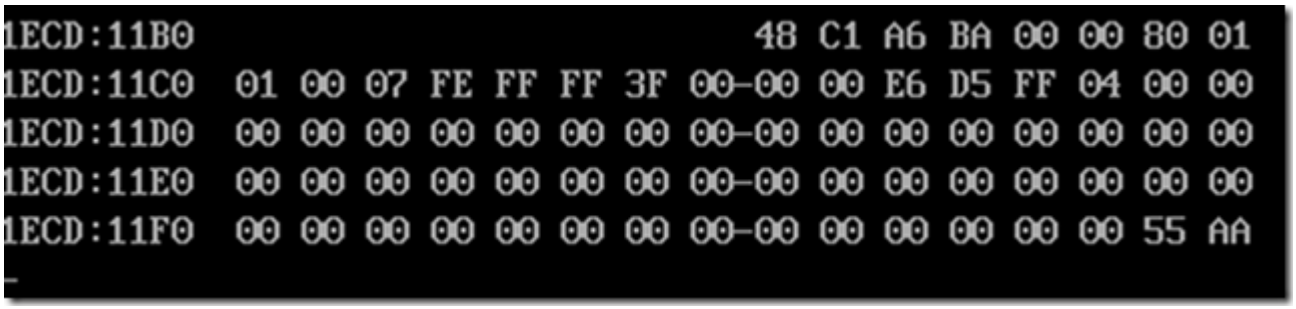
我们发现 MBR 数据完全没有发生变化，所以 XP 正常启动

3、预装 Vista 及以上级别的操作系统

原始 Vista



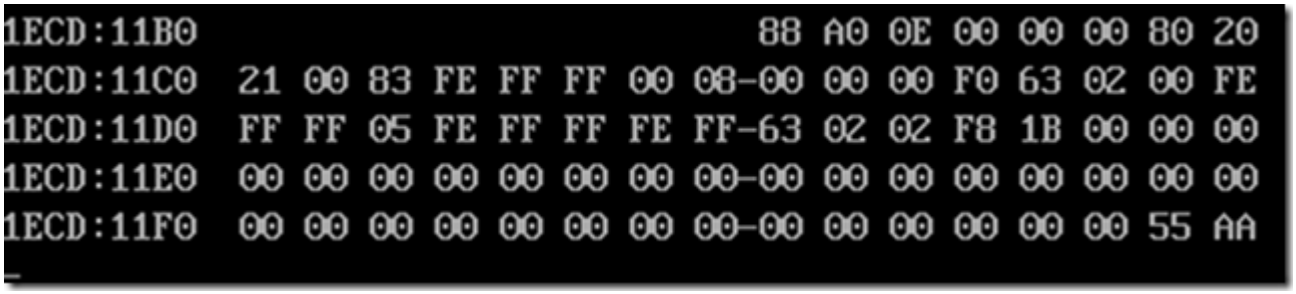
克隆后 XP



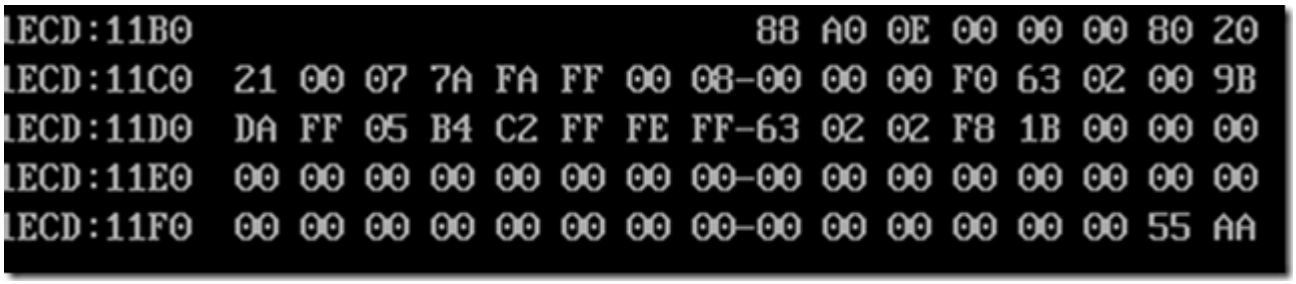
我们发现 48 C1 A6 BA 部分没有发生改变，但是后面 80 20 部分发生了改变，
这也就是为什么 Vista 系统还原深度镜像出现蓝屏的主要原因

4、预装 Linux 操作系统的电脑

原始 GRUB



覆盖后 XP



88 A0 0E 00 这段代码是不能引导 XP 的，在 GHOST 的过程当中，不会写入这段引导信息，所以 XP 无法引导

总结：

GHOST 更具深度版本的 GHO 文件内容自动重写了
1B8h - 1BBh 区域