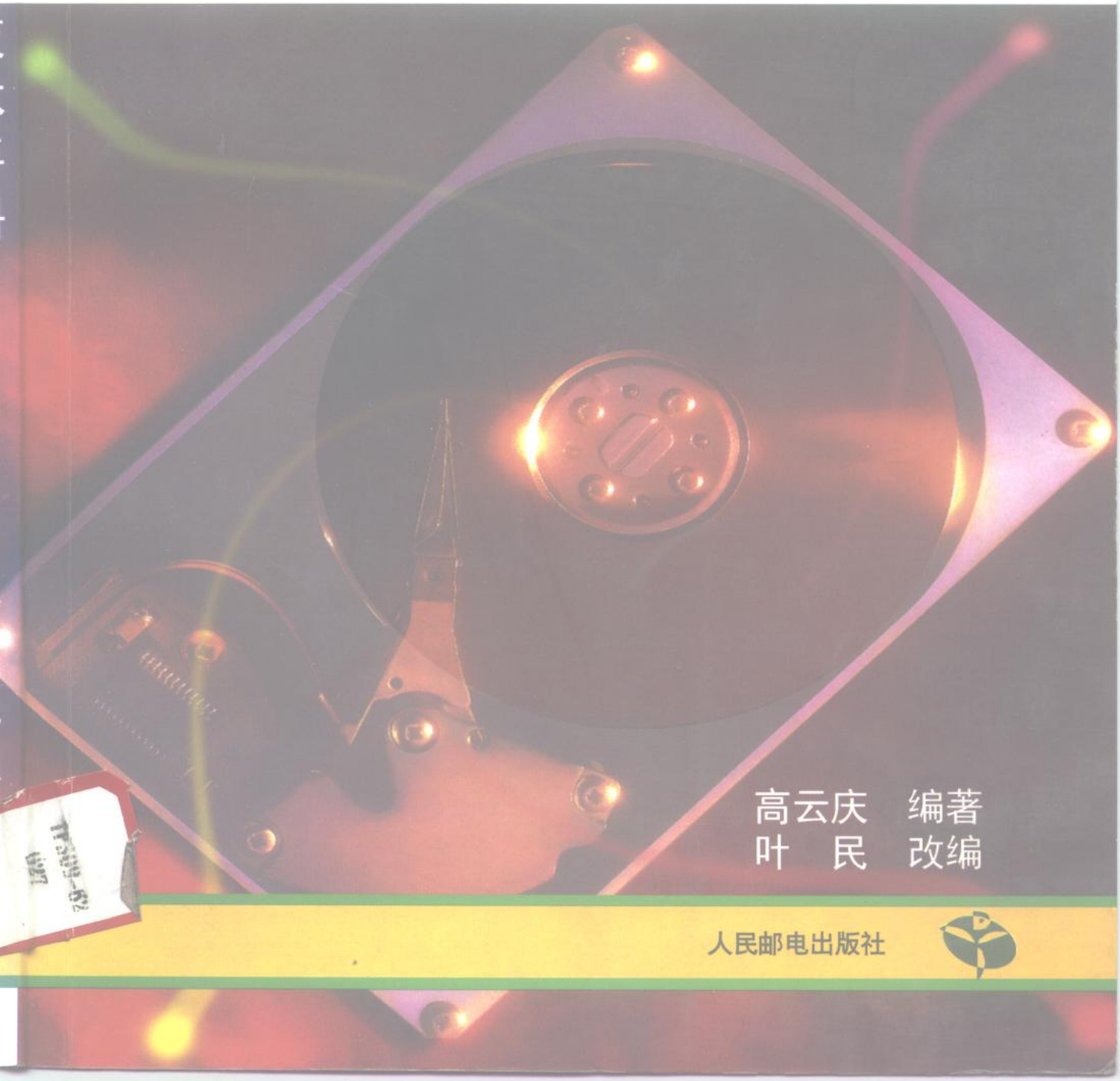
GOTOP

计算机技术入门提高精通系列丛书

硬盘保护技术手册



高云庆　编著
叶　民　改编

人民邮电出版社

计算机技术入门提高精通系列丛书

硬盘保护技术手册

高云庆　编著
叶　民　改编

人民邮电出版社

计算机技术入门提高精通系列丛书

硬盘保护技术手册

高云庆 编著

叶 民 改编

责任编辑 万东旭

*

人民邮电出版社出版发行

北京朝阳门内南竹杆胡同111号

北京密云春雷印刷厂印刷

新华书店总店科技发行所经销

*

开本：787×1092 1/16 1996年3月 第 一 版

印张：20 1996年3月 北京第1次印刷

字数：467千字 印数：1—5 000 册

ISBN 7-115-05999-3/TP·271

图字：01-95-198号

定价：30.00元

内容提要

硬盘是计算机用户保存数据的主要设备，如何避免硬盘受到他人恶意访问和计算机病毒的侵入，是所有用户关心的问题。本书全面剖析了从裸机上电启动到操作系统引导的全过程，介绍了在DOS环境下系统保护的各种方法。

本书分为三部分：第一部分基础篇，主要介绍保护程序设计的技巧、MASTER BOOT扇区、BOOT SECTOR的结构；第二部分提高篇，介绍扇区程序的编写，针对保护程序的每一具体功能分别做了详细讲解；第三部分研究篇，对于硬盘保护方法进行了总结、归纳，并介绍了两个典型硬盘锁程序。通过本书，读者不仅可以了解计算机基本原理，还可以掌握DOS环境下硬盘保护的各种方法，书中列举的大量编程技巧可供读者参考。

本书适用于广大计算机用户，及大、中专院校计算机专业师生阅读。

本书原版书名为《硬碟保护技术手册》，1995年3月出版。

出版说明

在计算机技术飞速发展的今天，为了进一步向全社会普及计算机知识，提高计算机应用人员的技术水平，使计算机在各个领域发挥更大作用，也为了促进海峡两岸计算机技术图书的交流，台湾碁峰资讯股份有限公司对我社独家授权陆续组织出版该公司的部分计算机技术书籍。这些书基本覆盖了当前最常用的各类计算机软、硬件技术，并紧随世界上计算机技术的飞速发展，不断有所更新。在写作特点上，这些书内容深入浅出、实用性强，在台湾很受读者欢迎。

在组织出版过程中，我们请有关专家在尊重原著的前提下，进行了改编，并对有关图文进行了核对和精心制作。

由于海峡两岸在计算机技术名词的称谓上差异较大，改编者依照有关规定和大陆习惯用法进行了统一整理。

对原书文字叙述中由于海峡两岸不同的语言习惯而造成的差异，我们的处理原则是只要不会造成读者理解上的歧义，一般没做改动，以尊重原著写作风格。另外改编时对原书的一些差错及疏漏之处做了订正。

由于本书改编和出版时间紧张，如有差错和疏漏，敬请读者指正。

人民邮电出版社

1995.9

目 录

基础篇

提高篇

研究篇

基 础 篇

第一章 程序编写基础

1—1 本书名词解释

为避免读者对本书中的用词有所疑惑，造成阅读的困扰，在此就本书中的用词做一简单说明：

主引导扇区（MASTER BOOT SECTOR）：硬盘的第 1 扇区，位于硬盘第 0 道第 0 面第 1 扇区，它是 IBM PC 读硬盘时所最先读取的扇区，包含了一小段装入程序（LOADER）及一个硬盘分区表（PARTITION TABLE）。这个扇区有些人称之为 MBR（MASTER BOOT RECORD），也有人因为这个扇区中包含分区表（PARTITION TABLE），称之为分区扇区。在本书中一律称之为 MASTER BOOT（主引导扇区），在以后的各章中所谓 MASTER BOOT 就是指这个扇区。在“基础篇”中我们会对这个扇区做深入的探讨，它也是“提高篇”的程序基础。

因为主引导扇区是独立在 DOS 之外用来装入操作系统的扇区（所使用的操作系统可以不是 DOS，可能为其它的操作系统，如 XENIX），所以不能用 DOS 的扇区读取命令来读取它，在 DEBUG 中也没有相对应的命令来读取它，所以必须使用 BIOS 的 INT 13H 功能调用编写一个程序去读取它。在本书的“基础篇”中会详加解释的。

引导扇区（BOOT SECTOR）：指的是软盘的第 1 扇区以及硬盘 DOS 分区开始的第 1 扇区。许多人会把这个扇区和硬盘的 MASTER BOOT 混淆。引导扇区（BOOT SECTOR）指的是逻辑驱动器的第 1 个扇区，假如一个硬盘分割成 C、D 两个逻辑驱动器，那么在 C 盘及 D 盘中各会有一个引导扇区（虽然 D 盘不可以用来启动，但

是它还是会有这个扇区),然而硬盘却只能有一个主引导扇区 (MASTER BOOT)。

简而言之,在硬盘中,MASTER BOOT 只能有一个,而 BOOT SECTOR 则依逻辑驱动器的数目而定。假如一台硬盘分割成 C、D、E 三个逻辑驱动器,那么它们就分别会有一个 BOOT SECTOR。换句话说, MASTER BOOT 的数目和物理驱动器数目相同,而 BOOT SECTOR 的数目则和逻辑驱动器数目相同。软盘则只有 BOOT SECTOR 而没有 MASTER BOOT。

引导扇区包含用来描述这个逻辑驱动器的 BPB (BIOS PARAMETER BLOCK——BIOS 参数块),以及用来装入 IO. SYS (MS—DOS) 或 IBMIO. SYS(PC—DOS) 的装入程序。

由于引导扇区是定义在 DOS 层次的,因此一般的 DOS 工具程序,如 DEBUG 及 PCTOOLS 都可以访问到这个扇区。

1—2 汇编语言程序编写说明

1. 程序的编译

本书所有程序都可以在 MASM 及 TASM 中编译,作者大部分是使用 TASM 作为编译工具,它们两者的汇编及链接方式如下:

(1) 以 XLOCK. ASM 说明扇区程序的编译法 (见第四章程序)。

· TASM

```
TASM XLOCK
TLINK XLOCK
EXE2BIN XLOCK——产生 XLOCK. BIN
BINTODAT XLOCK. BIN XLOCK. OUT——转换成汇编语言格式数据文件
```

· MASM

```
MASM XLOCK;
LINK XLOCK;
EXE2BIN XLOCK——产生 XLOCK. BIN
BINTODAT XLOCK. BIN XLOCK. OUT——转换成汇编语言格式数据文件
```

其中 BINTODAT. EXE 为作者自行编写的工具程序,用来将二进制文件转换成汇编语言格式的数据文件,因为 XLOCK. ASM 为一扇区程序,不能在 DOS 下执行,必须将它写入 MASTER BOOT 中才可以在启动的时候执行。为了程序编写方便,在本书中都是将它转换成汇编语言的数据文件,然后再写一个装入程序。至于 BINTODAT. EXE 的源程序及详细用法请见附录。

(2) 以 XLOCKLD. ASM 说明 COM 执行文件的编译法 (见第四章程序)。

· TASM

```
TASM    XLOCKLD
TLINK/t XLOCKLD —— t 必须为小写
```

· MASM

```
MASM XLOCKLD;
LINK XLOCKLD;
EXE2BIN XLOCKLD.EXE XLOCKLD.COM
```

如果读者没有 EXE2BIN 这个程序（TASM 不提供），也可以用 DEBUG 程序自行做转换的工作，其用法如下：

```
C>DEBUG XLOCK.EXE
-N XLOCK.BIN —— 取文件名为 XLOCK.BIN
-R CX
200          —— 设置文件大小（DEBUG 以 BX：CX 表文件大小）
-W CS：0     —— 由 CS：0 开始写入
-Q           —— 离开
```

（3）由于以上操作频繁，因此作者建议各位读者可以写一个批处理文件完成以上的工作。作者提供一个简单的批处理文件：

· RUNBIN.BAT

```
TASM  %1
TLINK  %1
EXE2BIN  %1
BINTODAT  %1.BIN %1.OUT
```

· RUNCOM.BAT

```
TASM    %1
TLINK/t  %1
```

如此一来前面的过程我们只要改成下面的用法即可：

```
RUNBIN XLOCK
RUNCOM XLOCKLD
```

使用 MASM 的读者只要将上面的 TASM 改成 MASM，TLINK 改成 LINK 即可正常执行了。

2. 汇编语言的特殊技巧

在本书中使用了少部分在汇编语言中鲜为人知的技巧，我们在此一一说明，读者也可以体会汇编语言的威力：

（1）控制权切换的方法。

汇编语言的控制权切换指令有下面几个：

- JMP 指令群（JMP，JNZ，JE……）
- CALL 指令
- INT 指令

其中 JMP 及 CALL 指令大多使用相对地址，也就是说链接程序在碰到 JMP 及 CALL 指令时必须将转移的相对地址算出，然后才给定它的值。（JMP 及 CALL 可以使用绝对地址，稍后再加说明，此处先说明一般使用符号地址做转移的情形）。

我们举一个简单的例子来说明：

程序 TEST.ASM

```
    .MODEL TINY
    .CODE
    ORG  100H
BEGIN:
    PUSH  CS
    POP   DS
    JMP   @1
@1:
    JMP   @2
    NOP
    NOP
@2:
    MOV   AH,4CH
    INT   21H
    END   BEGIN
```

显示如下：

```
C > DEBUG
-n test.com
-l
-u 100 110
3809:0100 0E        PUSH    CS
3809:0101 1F        POP     DS
3809:0102 EB01      JMP     0105 ←— 机器码 EB 01
3809:0104 90        NOP
3809:0105 EB03      JMP     010A ←— 机器码 EB 03
3809:0107 90        NOP
3809:0108 90        NOP
3809:0109 90        NOP
3809:010A B44C      MOV     AH,4C
3809:010C CD21      INT     21
3809:010E E9EEFE    JMP     EEFE
-q
```

TEST.ASM 是一个简单的程序，用来查看 JMP 指令所编译出来的机器码。我们由 DEBUG 的结果可以看出第一个转移指令 JMP @1 所编译出的机器码是 EB 01，而第二个转移指令 JMP @2 所编译出来的机器码是 EB 03。由这里我们可以很简单地看出，JMP 指令的机器码是 EB，而后面接的数字则是程序指针至新地址所必须加上的位移值。

同样地，读者可以用同样的方式来对 CALL 指令做检查，其结果应该和上面类似。

现在让读者思考一个问题：如果要使程序转移到固定地址时，应该如何来完成？我们将问题简单化，如果要让程序转移至 0：600 的地址，应该怎么做才可以呢？

前面说过，因为 JMP 指令是使用相对地址，因此在不同的地方使用 JMP 指令转移到同一地址会产生不同的机器码，因为其相对地址不同；就好像虽然同样是乘飞机到北京，但是由上海到北京和由广州到北京的票价就不同一样。我们写一个程序来验证这个说法：

程序 TEST2.ASM

```
    .MODEL TINY
    .CODE
    ORG     100H
BEGIN:
    PUSH    CS
    POP     DS
SHANGHAI:
    JMP     BEIJING
    NOP
    NOP
GUANGZHOU:
    JMP     BEIJING
    NOP
    NOP
BEIJING:
    MOV     AH,4CH
    INT     21H
    END     BEGIN
```

显示如下：

```
C > DEBUG
-n test2.com
-l
-u 100 110
3809:0100 0E        PUSH    CS
3809:0101 1F        POP     DS
3809:0102 EB08      JMP     010C
3809:0104 90        NOP
3809:0105 90        NOP
3809:0106 90        NOP
3809:0107 EB03      JMP     010C
3809:0109 90        NOP
3809:010A 90        NOP
3809:010B 90        NOP
3809:010C B44C      MOV     AH,4C
3809:010E CD21      INT     21
-q
```

同样是 JMP 10C，但是一个机器码为 EB 08，另一个为 EB 03，因为位移不同。