

C-64

SOFTWARE  
**PROTECTION**  
REVEALED



Dell M. & Robert H. Taylor

56

1038

**C-64**

**SOFTWARE**  
**PROTECTION**  
**REVEALED**

**Dell M. Taylor**

**Robert H. Taylor**



**Portland Oregon**



PET, C-64, and Commodore 64 are trademarks of Commodore Business Machines.

Radio Shack and TRS-80 are trademarks of the Tandy Corporation.

Published by Value Soft Inc. 9513 S.W. Barbur Blvd. Suite 56, Portland, Oregon 97219

Copyright 1985 by Value Soft Inc.

All rights reserved. No part of this manual may be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photo copy, recording or otherwise, without written permission from Value Soft Inc. No liability is assumed with the respect to the use of the information contained herein. While every precaution has been taken in the preparation of this book, the authors assume no responsibility for the errors or omissions. Neither is any liability assumed for its use or damages resulting from the use of the information contained herein.



## Commodore

### Trivia

Commodore Business Machines began as a typewriter repair business.

The first Commodore computer was designed for the Radio Shack company. After evaluation, Radio Shack decided to design and manufacture the TRS-80.

Commodore's first attempt at the personal computer, was known as the 'PET'. The name was based on the then popular 'Pet Rock'.

The PET later became known as the short for Personal Electronic Transactor.

PET was also said to stand for Peddle's Ego Trip. The head engineer at the time was Chuck Peddle.

Just stop and think, if Radio Shack had decided to market the Commodore, you would now be using a TRS-64.

# Contents

## CHAPTER ONE

INTRODUCTION	2
<hr/>	
Software Law	5
Copyright	5
Patent	5
Trademark	5
Trade Secret	6
Limiting Liability	6
Software Protection	6
Archival Copies	7

## CHAPTER TWO

WHAT IS SOFTWARE PROTECTION ?	12
<hr/>	
Hardware Protection	13
No Protection	14
The Program	14
The Disk	15
Disk Copy Programs	16

## CHAPTER THREE

1541 DISK DRIVE	20
<hr/>	

## CHAPTER FOUR

SUPERMON & MATH	26
<hr/>	
Supermon Registers	27
Supermon Commands	27
Hex, Decimal, & Binary	30
Conversion Table, Hex, Dec, Binary	32

## CHAPTER FIVE

### READ A GOOD DISK LATELY ? 38

---

Pointer To Next Sector Of DIR	41
DOS Version Used To Write Disk	41
BAM Map	42
Name Of Disk	43
I.D. Number Of The Disk	43
DOS Version & Format Type	44

## CHAPTER SIX

### TRACK 18... THE HEAD LIBRARIAN 46

---

Finding The Next Block Of The DIR	48
File Entry	48
Type Of File	49
First Track & Sector Of A File	50
Name Of File	50
Length Of File	50

## CHAPTER SEVEN

### BASIC PROGRAM PROTECTION 54

---

Compiled Programs	54
PoKs & PeEKs	55
No Line Numbers	56
Cassette Buffer	57
Move Start Of Basic	57
Auto Load & Run	58
REM's...	58
REM's... Listings	59
REM's... No List, No Print	60
REM's... Missing Code	61
REM's... Invisible Lines Numbers	61
REM's... Duplicate Line Numbers	62
REM's... Mixed Line Numbers	63
REM's... Clear The Screen	63
Change File Type	63

CHAPTER EIGHT

ALL KINDS OF TRICKS	66
<hr/>	
Write Protected Disk	67
Scratch Protected File	68
False Blocks Free	68
Directory Changes	68
False I.D.	69
Unusual Load & Save	69
Misleading File Names	70

CHAPTER NINE

STRANGE DISK DRIVE NOISE	74
<hr/>	
What Is A Disk Error ?	76

CHAPTER TEN

ADVANCED PROTECTION METHODS	82
<hr/>	
The Kernal	82
Kernal ROM	85
Super Line Numbers	88

CHAPTER ELEVEN

THE NEWCOMERS	94
<hr/>	
Non-Standard Sectors	96
Extra Sector	96
Density Changes	97
Guard Band	98
Half Tracks	98
Nibble Back-up	99
Nibble Counting	99
Tracks 35 To 40	99



APPENDIX A

PUBLIC DOMAIN SOFTWARE COLLECTION	102
-----------------------------------	-----

---

APPENDIX B

GLOSSARY	114
----------	-----

---

APPENDIX C

1541 RAM	120
----------	-----

---

APPENDIX D

HELP CHARTS	134
-------------	-----

---

Hex/Decimal/Binary/GCR	134
Bit Values	134
Hex To Decimal Conversion	135
True ASCII Conversion	136
1541 Disk Drive Commands	140
DOS Wedge Commands	140
1541 Utility Instruction Set	141



**Chapter**

**One**

# INTRODUCTION



The Commodore 64 has only been around a short time, but lots has happened in that time frame. I remember a wait in line to buy mine, it was the last 64 they had in stock..... As I moved quickly away from the counter, I could see a long line of eager eyes behind me. Then the other eager customers heard the shy clerk announce, "sorry folks, we are all out until our next shipment a week from Thursday". A wave of quiet fell over the crowd as all eyes turned towards me. I hurried to the car with the feeling that if I didn't leave quickly, someone might grab the box right out of my hands. I think I could have held an auction right on the spot and made a profit.

It wasn't long before a flood of software began to appear in the shops, new companies emerged every day. The race was on, everyone wanted to make their million dollars, and a few did. Many of the early programs were written in BASIC with little thought given to the subject of software protection. The people that bought the programs knew little about programming, everything seemed to be going pretty smooth.

Next to appear on the horizon, were the users groups. The individuals began to learn the secrets of their Commodore 64's. Groups of twenty, thirty, or even fifty computerists got together on Saturday afternoons to share ideas and learn from each other. People began trading software, it was all so innocent, just trading with a friend. No one had ever heard the word 'Piracy' linked with software. The word was associated with the days of sea captains and sailing ships. I don't know where the term got it's start, but in the software manufacturing circles, it was a popular topic. It's like the chicken and the egg as to which came first, piracy or protection, but soon there were both.

The protection schemes were simple, usually not much more than preventing the loading and saving of a program directly from memory. POKES and PEEKS disabled the RUN/STOP and RESTORE keys, listings vanished before your eyes, and REM statements took on a new meaning. The birth of the copy programs to make archival copies opened a new world to the user. Software being high priced and of questionable quality, tempted users to pool their financial resources and share programs. Nothing was thought of the practice, until it became apparent, that it had grown into a regular habit for many.

Software publishers soon began stepping up protection methods in their software. Archival copies (and a few illegal ones) were easy to make, too easy, thought the authors of the programs. This in turn opened up a new field of software, the copy program. A new protection scheme made the birth of a new copy program inevitable. The cat started chasing it's tail, never quite able to catch it, but not far behind.

The term piracy has become an everyday word in the field of software. Why much controversy over such a small word? Those six letters can end a friendship and raise more mis-information than any other I know of. Why??? If you buy a book or a record album and make a copy, very little is said, in fact usually nothing. Then why if a piece of software is copied, is it called piracy? Technically the terminology is right. If you intend to sell or give away copies, it is legally a criminal act. Copying for your own personal back-up or archival purpose is your legal right. To loan an original to a friend is okay. Think about the local library for a moment, if you cannot loan your software, then all the libraries in the country, are practicing an illegal act? I have never heard anyone question the right to have a library card and borrow books. Who knows the amount of material copied, for that matter who cares.

There is one other consumer right that seems to stay behind closed doors. The right to revise a program (you have purchased) to meet your needs. Often, software could increase your productivity, if just one little feature could be added to the original code. You have that right! All that is stopping you is the protection that has been placed on the disk by the author. The author has prevented you from exercising your legal rights.

Don't get the wrong idea, I don't condone piracy and never will. The person that copies and sells or gives away copies of a program is very wrong and breaking the law. But you as the buyer of the package also have rights. The right to protect your investment by making a back-up to store in a safe place, accidents do happen. This brings us to the subject of this book, software program protection.

The software manufacture has two primary methods to protect his products from unauthorized copying. First is offered by the laws as passed by the Congress of the United States. Second is copy protection that the programmer adds to prevent the practice of illegal duplication. Usually a combination of the two is used.

## **SOFTWARE LAW**

Legal means to protect software has been provided by law. Often it is next to impossible to enforce, but the field is new and it takes time to modify existing codes. What follows is a short overview of the four popular legal protections available to the author of software.

### **Copyright**

A copyright protects the expression of an idea, not the idea itself. It is not what the program does, but how it does it that really counts. This must be an original work by the author and not in the public domain. A copyright is born as the program is written. The author has up to five years to apply for a written legal copyright.

### **Patent**

A patent protects the idea itself. Sounds great, but don't get too excited. Patents are very slow to get ( maybe years ), plus very expensive. There is a good chance your program could be obsolete before you are done.

### **Trademark**

A trademark protects only the name of the program and not the software itself.

## **Tradeseecret**

Your program is a trade secret just as long as you can keep it a secret and no longer. Great while you work on the project, but not at time of sale.

## **Limiting Liability**

The author has an obligation to live up to the claims as stated on the package, or implied by the seller. By the means of a disclaimer, the author can protect himself from an unhappy customer. In many states a disclaimer must be placed in a visible location, without the need to open the package in order to be valid.

If you are an author, please consult an attorney before you begin your endeavor, it could save a lot of time and heart ache in the end.

## **SOFTWARE PROTECTION**

Software protection comes in many levels of sophistication. Some very simple, others unusual, and some are down right eccentric. A cartridge requires special hardware to copy, that in it self is protection. Not all users are willing to invest in an expander board just to pirate a program. Tape is very limited as to protection schemes, plus not sold in great numbers. A diskette is another story, there are more ways to protect a diskette, than I care to count. The use of auto boots, errors checking routines, extra sectors, side sectors, half tracks, just to name a few of the popular types, begins to make the pirate work for that free copy. Disk protection can only be compared to solving the cross-word puzzle in the Sunday paper, some of the words are easy and others are more complicated.

## ARCHIVAL COPIES

Disks and tapes are susceptible to every imaginable accident, a back-up should be made as soon after purchase as is possible. These are known as archival copies. It is your right as a buyer to make a copy to protect your investment. You the buyer of software also have rights provided for by law as follows:

Computer Software Copyright Act of 1980, (17 USC 101): "A computer program is a set of statements or instructions to be used directly or indirectly in a computer to bring about a certain result." This was the first time computer software was defined by U.S. copyright law.

Also added to the section was the following: "Notwithstanding the provisions of section 106, it is not an infringement for the owner of a copy of a computer program to make, or authorize the making of, another copy or adaptation of that computer program provided (1) that the new copy or adaptation is created as an essential step in the utilization of the computer program in conjunction with a machine and that it is used in no other manner, or (2) that the new copy or adaptation is for archival purposes only and that all archival copies are destroyed in the event that continued possession of the computer program ceases to be rightful."

"Any exact copies prepared in accordance with the provisions of this section may be leased, sold, or otherwise transferred, along with the copy from which the copies were prepared, only as part of the lease, sale or other transfer of all rights in the program. Adaptations so prepared may be transferred only with the authorization of the copyright holder."



How strange it is, I can buy a book to read, when finished, I can take it to a used book store without asking the author's permission and trade or sell the manual. Why is software treated any differently. This makes absolutely no sense at all. You buy the product, but it is not yours to do as you please. From the time a child is born, it is taught to share with friends. That is except software, this could get confusing to say the least.

Now comes the controversy. Sometimes the manufacture provides a back-up (if you send for it in the mail) and sometimes not. Here you are in the middle of a writing a letter, using your favorite word processor, the program crashes. Who knows why, maybe a bad disk or just a speck of dust. It seems a little far fetched to expect the user to mail order for a replacement, just to finish a letter. Then there is the matter of the additional fee, you have paid for it once and it should be yours. The law states you can make a back-up, but the author adds protection to your disk to circumvent your legal rights. Now is this fair?

Authors, if you are wanting to learn about protection, bear with me, there are two sides to this story. At this point I'm not talking about pirates, just an honest user trying to write one complete letter. He paid his money and expects to be able to do the job.

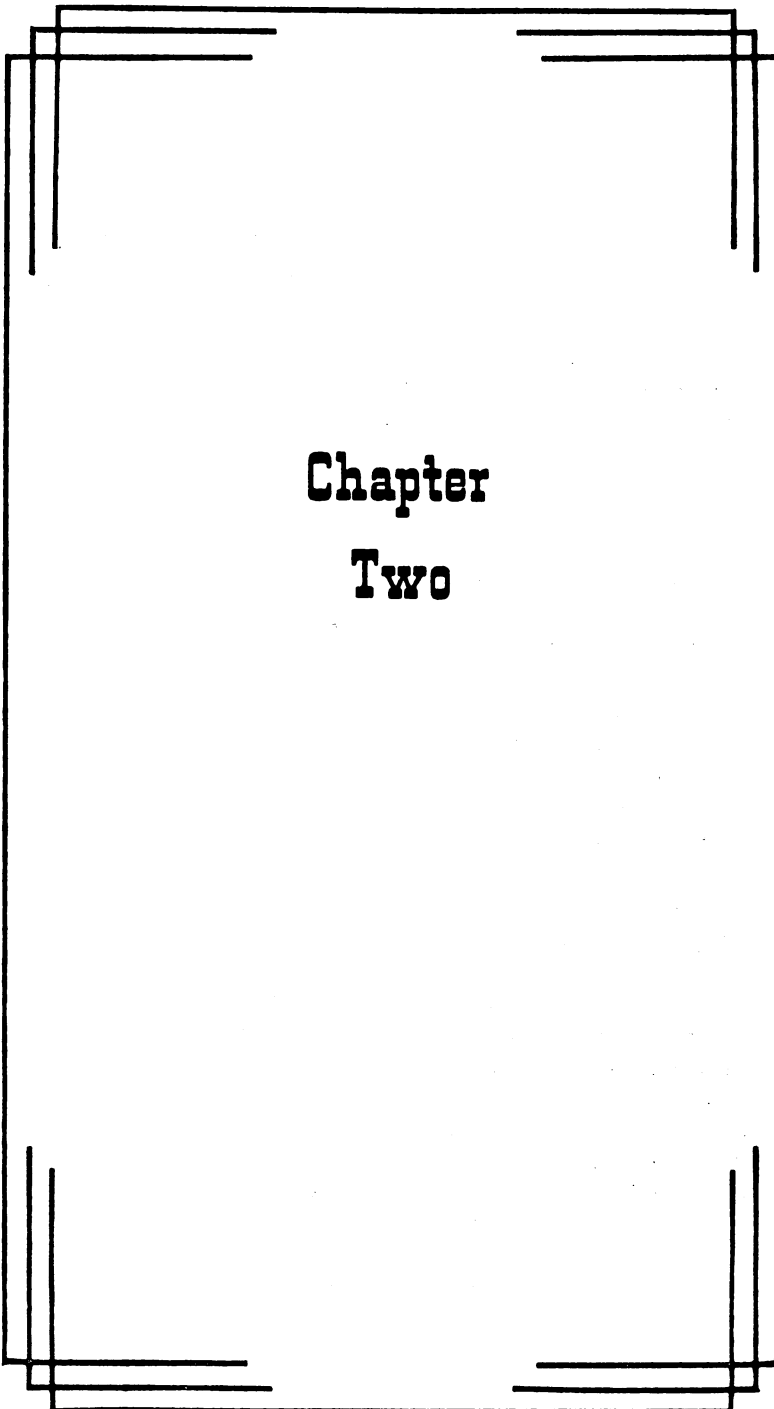
By this time the user gets angry, and why not, the disk is his and it's not usable. The next time he buys software, if he does, this won't happen. He will find a way to back-up the program before it is used, or maybe even get a pirated copy that is already broken and can be copied.

Software that has been broken has a lot going for it. It loads quicker, takes less disk space, does not rattle the heads of the disk drive. Above all a back-up copy can be made. A user has an investment in a disk drive. The wear and tear of the heads being beat out of alignment by protected software is an unneeded expense. It would seem that the manufacture is encouraging piracy, by forcing the user to break a program in order to obtain a back-up copy and to protect the investment in equipment.

There is one other step the user can take. On today's software market there are many copy programs that will make an exact copy of the original, or so the author's claim. Remember I said exact copy of the original, including the errors that make the heads of the disk drive beat against the end stop. This is one of the causes for drives getting out of alignment. The errors have been placed there by the programmer as a means of program protection, without consideration of the users equipment. In simple english the programmer is more interested in protecting the software from pirates than about your computer equipment. I personally know of one program that damaged the drives of five of my friends. Now I ask you, is that fair practice on the part of the author? Remember, it could happen to you.

<-- Protection Revealed -->

-----  
NOTES  
-----



**Chapter  
Two**

## WHAT IS PROTECTION ?

Program protection refers to various methods that software programmers use to stop a person from making illegal copies of programs. The only problem with this idea, is it also stops the buyer from making a legal back-up. It must be remembered that both the author and the user have legal rights. The author expects to make a decent wage for his time and the user wants a usable product for his money. Somewhere along the line a happy medium must be met, but until that happens there will be a need for books like this one. The whole subject of software protection seems to be clouded with intrigue, keeping information of how to protect or un-protect, a deep dark secret. The methods discussed in this book will help you to protect your own software and to make a back-up of the programs you have purchased.



The diskette affords a wider variety of protection schemes than any other medium. First is the legal means of protection and second is the disk itself. In this chapter we will begin to explore some of the various

methods used by the professional programmer. This book will be releasing information that is known by few, of which those in the know, hold to be a very deep dark secret. No attempt will be made to cover all the ways of protecting a disk, as new ones emerge every single day.

There are four major methods of protecting disk software from illegal duplication.

- 1 - Hardware
- 2 - No protection
- 3 - The program
- 4 - The disk

## **HARDWARE PROTECTION**

Some software manufactures use no protection on the disk, but add a device known as a dongle. The dongle is a hardware apparatus that must be plugged into a computer port in order for the program to run properly. On the Commodore 64 it may be located on the RS-232 user port, game cartridge port, joystick ports, or the cassette port. As you can see the author has quite a few options for placement.

Inside the dongle can be found anything from a piece of wire to very sophisticated electronic components. Often the dongle supplies information to the computer, to be used by the program. If the interaction is not complete, the program will crash. Inside the dongle may be found frequency generators, pulse multipliers, timers, just to name a few of the surprises in store for the curious mind.

With this form of protection both the buyer and the author are protected. The purchaser may make unlimited copies for

back-up purposes, but without the addition of the dongle, the program is useless. This sounds great, but remember nothing is ever fool proof. Eventually the dishonest person finds a way around anything, but at least the thief will have to work for it.

## NO PROTECTION

To say no protection can be one of the best methods available, must be confusing, but believe me, it can be. Think of some of the fancy manuals with colored pages and elaborate art work that come with good business packages. Colored pages and intricate drawings, often will not reproduce well. Adventure games can be useless without the directions and charts. When a pirate finds something won't photo copy, few are willing to re-type a lengthy documentation. Many of the flight type games ask for authentication codes that have been supplied in the manual. Most pirates overlook the need to copy printed matter, that costs time and money.

Serial numbers placed in out of the way spots, offer unique possibilities for the author. Just think, if the program uses part of the serial number for calculations or setting up the screen and an un-suspecting pirate removes it, things might never run right again. If someone were to pirate a tax package, it could get interesting around April 15th when all the figures in his tax return come out a few digits off.

## THE PROGRAM

Software protection within the the program itself comes in many varying degrees. From the very simple BASIC tricks to the super protected machine language techniques, all can be bad news to the user. The level of

protection is left to the programmer and possibly his mood of the day. The directory may be hidden from the user or modified so that it can not be listed. POKes and peekS can disable the keyboard or make the program auto-start. Add enough of these tricks to a program and it will slow down the best of the pirates.

Many programmers feel to compile their software is the ideal way of protection. This does discourage most pirates. It is kind to your disk drive (no heads rattling) and that means a lot to the average user. If you are an author looking to this book for help in protecting software that you wish to market, let me warn you, there are now de-compilers available. It would appear that nothing is foolproof.

## THE DISK

The disk is where Pandora's box opens up and the real fun protection begins. The drive can be classified both as a tool and a toy when a good programmer takes command. Can you believe a drive making musical sounds? It can be done, believe me, mine can play a whole song.

Many different types of errors may be intentionally placed on the disk. As the program runs it will check for them. If the right error is found in the right place the program will run fine. But if a copy of the original has been made and the errors are not found on the duplicate disk the program will crash. Another popular method of protection is to make the original disk on a different type of drive. It can be run on the 1541, but not copied. Auto-loaders execute as soon as they are loaded, thus preventing the user from making a copy. Add non listable directories,



extra sectors, and side sectors to the growing list. This sample is just the beginning of the disk story. In the pages that follow the plot will thicken as many other methods un-fold.

## COPY PROGRAMS

As fast as new forms of protection come on the market so do new copy programs. All a person has to do is read a popular current computing magazine to learn what is available. Who would have though less than a year ago, that it would be possible to copy virtually any commercial program in less than three minutes. The latest marvel can copy a full disk in 38 seconds, using two disk drives. Different kinds of protection schemes, use different copy programs. The following claim is from an add in one of the leading periodicals on the news stand today: " treats disk errors, extra sectors, re-numbered tracks and other protection schemes exactly the same as ordinary data". This kind of technology is enough to keep the software manufacture awake nights. The program sounds perfect, that is until the next new protection scheme comes along. The level of expertise in program protection becomes more sophisticated everyday. We may soon reach a level of protection that may not be overcome by the home user.

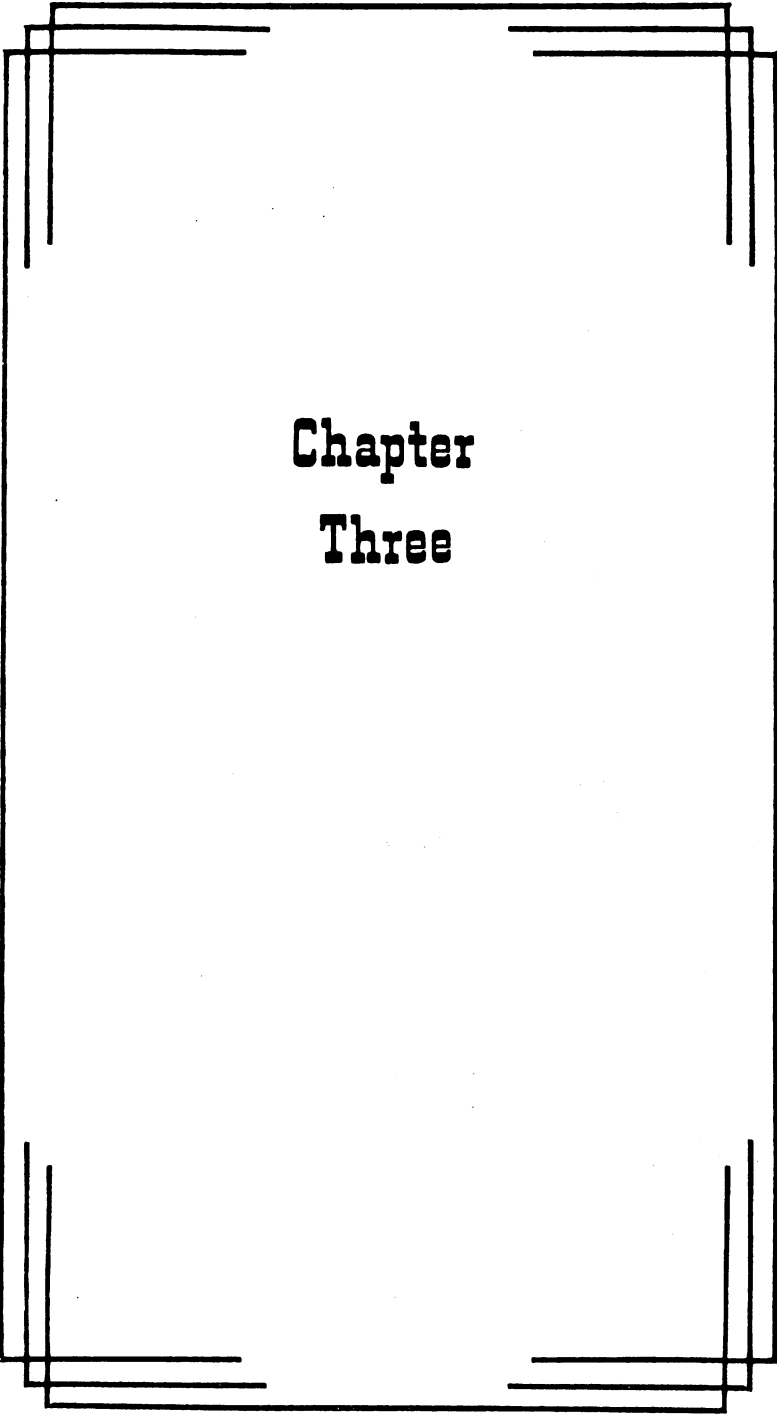
It would begin to appear that the industry and the customer are both going in circles. One adds a new protection scheme and the other dreams up a new way to de-protect. Have you ever watched a small kitten chase it's own tail? Sounds familiar doesn't it.

This book is designed to allow the user or the author (which ever the case may be) to take control of their software. Another phrase I often see applies here, "PATIENCE IS A

1541". Take your time, relax, read carefully, and you can be in command of your software. Remember above all else, it is the right of the author to protect his lively hood, but it is also the right of the user to make an archival copy and to modify code to suit personal needs.

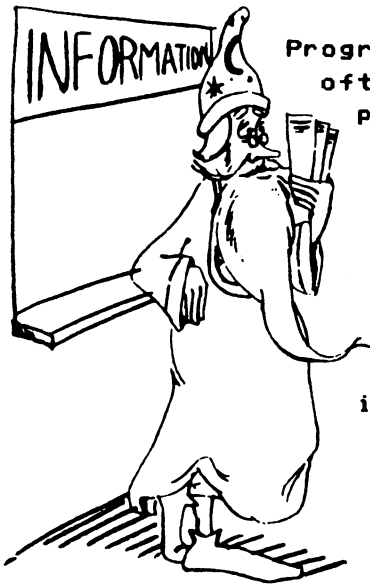
<-- Protection Revealed -->

-----  
NOTES  
-----



**Chapter  
Three**

## THE 1541 DISK DRIVE



Program protection methods are often thought of as mystical professional trade secrets. A million questions come to mind, the first time you try to save a protected program. Why can't it be listed? What kind of program is it? Why is my new disk drive growling at me? The list of questions grow longer each time you try to look at the disk. The disk drive and the diskette are the key elements to the the mystery, so that is where we will begin. If you find any

ny of the terms used, new to you, please refer to Appendix B. A small glossary of words used in this book have been included.

The 1541 Disk Drive has a complete operating system within itself, a total of 16K of ROM and 2K of RAM. It uses a 6502 microprocessor chip, just as is in the Commodore 64. The drive does it's own processing without using any memory from the computer, this makes it a smart device.

The 1541 disk drive stores 174,848 bytes of information per diskette. When a diskette is purchased, it is generic and may be used with many makes of computer. For a diskette to

be used with the Commodore 1541 it must first be formatted for compatibility with the drive.

EXAMPLE: OPEN 15,8,15  
PRINT#15,"N:name,id"

The OPEN command allows the user to communicate with the disk drive or another peripheral device. The 15 that follows the OPEN command is what channel to open, the 8 is the device number, the last 15 is a secondary address. The PRINT #15 in line two, sends data to a device other than the screen, in this case the disk drive. The letter 'N' stands for new, this erases the entire disk, adds timing and block markers, plus creates the directory and BAM. The name is for the user to insert a name of their choice. It may not be more than 16 characters in length. If the name is less than 16 characters, the computer will pad with blank spaces. The ID is two characters in length. This ID will be written to the directory plus each block on the disk.

As the diskette is formatted, it is divided into 35 tracks. Track one is the outermost track, 35 is found in the center of the disk. Each track is then divided into from 17 to 21 sectors, this varies as to where the track is located on the diskette. The chart below shows how the tracks are arranged.

DISK CONFIGURATION

TRACK NUMBERS	SECTORS PER TRACK	BYTES PER TRACK
1 to 17	21	5376
18 to 24	19	4864
25 to 30	18	4608
31 to 35	17	4352

There are 664 blocks of memory available to the programmer on each diskette. There is actually 683 blocks of memory on each diskette, but 19 blocks are used for housekeeping. Each block contains approximately 256 bytes. The total number of bytes per disk is 253,459. The information is stored in rings on the face of the disk, very much like the surface of a phonograph recording.

During the process of formatting the disk, many changes transpire. A directory is placed on track 18 to record information about the files placed on the disk. This is much like an index to a book, and the key to all information placed on the diskette. First you must provide the disk with a name (not more than 16 characters). The name is usually an aide as to what kind of programs are stored on the diskette. This name means nothing to the computer or drive, it is for your reference only. Next you must choose an ID, it will be written on each block or sector on the disk. The drive repeatedly checks for this ID, it must be unique. If you have two disks with the same ID and try to copy files from one disk to the other, the drive will become confused. The drive will not realize there actually are two disks and not know where to write the file you wish to save.

To better understand a diskette, let's examine the directory of the Public Domain disk that came with this book.

```
LOAD "$",8
```

Next type LIST. The directory is read from left to right, here is how it will appear on your screen:

```
0 "PROGRAMS FIN      " PF 2A
5  "MENU"           PRG
1  "-----"       PRG
32 "EDIT T & S"     PRG
1  "-----"       PRG
7  "VIEW BAM"      PRG
1  "-----"       PRG
3  "NEW DISK TITLE" PRG
1  "-----"       PRG
5  "CHANGE DISK ID" PRG
1  "-----"       PRG
3  "TRACK ID READER" PRG
1  "-----"       PRG
5  "TRACE FILE LINKS" PRG
1  "-----"       PRG
```

The example shows only a part of the directory that you will see if you execute a LIST command. Starting on the first line, the 0 is the number of the drive that was accessed. Next is the diskette name, "PROGRAMS FIN ". If the name was less than 16 characters in length, blank spaces have been added by the computer, as filler. After the name is the ID and finally the last two figures are the DOS version. The DOS tells the disk drive what type of disk drive was formatted on.

The numbers in the far left column, tell you how many blocks of data each file contains. Now using the figure above (256 bytes per block) you can determine how many K of memory the program contains.

EXAMPLE: 256 X number of blocks = bytes per file.

In the center column, are the names of the individual files contained on the disk. The three letters on the far right hand side, denote the type of files.

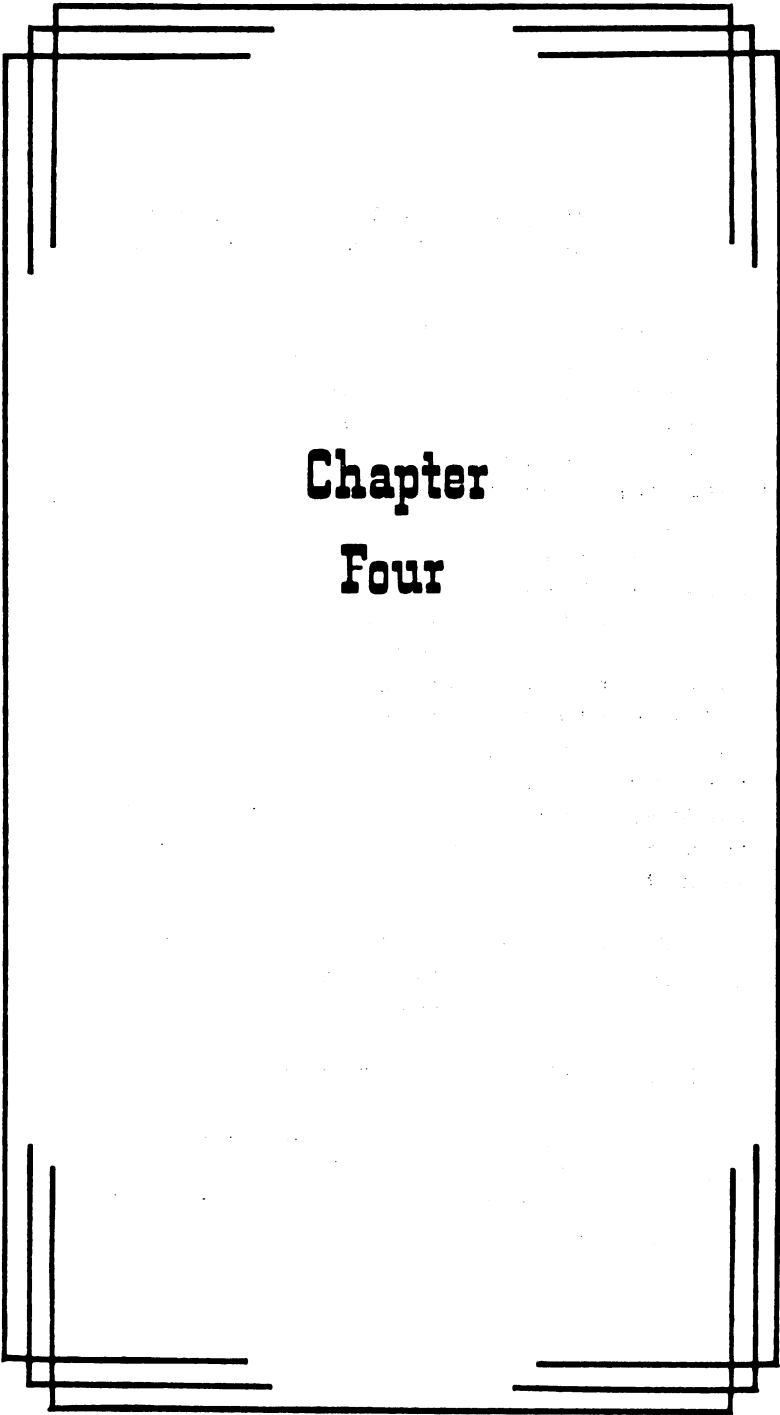


EXAMPLE:

PRG....Program files  
SEQ....Sequential files  
REL....Relative files  
USR....User file

The floppy diskette is made of thin flexible plastic and should be considered very fragile. Always store them in a dust free enclosure container. Never touch the actual disk through the openings in the jacket. A finger print is death to data stored on the disk. Never allow them near anything that is magnetic, this can erase all data. Remember two of the most common items in the home, the television and the telephone, have magnetic fields. I only mention this because many people have learned the hard way, resulting in countless lost dollars, plus endless hours of valuable time. When not writing to the disk, always keep a write-protect label over the notch in the side. I admit I have made the sad mistake more than once, and lost all data from several disks by this mistake.

Patience Is A 1541



**Chapter**  
**Four**

## SUPERMON & MATH

Supermon is a monitor that allows the programmer to write code in machine language. The writer may explore and change the computer's memory and registers. A monitor will disassemble (convert) machine language into assembly language. Assembly language has labels that are abbreviated english, and are much more readable by the user, than machine language. Supermon is an invaluable utility program when exploring or writing protection methods.



The program Supermon is on the Public Domain disk included with this book. Supermon should now be entered into memory.

```
LOAD "SUPERMON.C",8 <return>
```

When Supermon has finished loading from the disk, type RUN. The monitor identifies itself by printing B\* and the contents of the 6510 registers to the screen.

```
EXAMPLE:  B*   PC   SR AC XR YR SP  
          .; 37FE  31 40 E6 00 F6
```

## Supermon Registers

---

PC...PROGRAM COUNTER: keeps track of the address within the program that is to be processed next.

-----

SR...STATUS REGISTER: contains indicators of conditions that have occurred as the result of math or logical processing.

-----

AC...ACCUMULATOR: contains the results of math or logical operations, used to transfer data during input or output operations.

-----

XR...X-REGISTER: the index register is used for differing ways to access memory.

-----

YR...Y-REGISTER: the index register is used for differing ways to access memory.

-----

SP...STACK POINTER: contains the pointer to the top of the stack (work memory).

-----

IRQ..INTERRUPT VECTOR: is the address of the routine that gains control when a program interrupt occurs.

---

## Supermon Commands

---

.G = GO RUN

Go to the address in the PC register and begin RUN code. All registers will be replaced with the displayed values.

EXAMPLE: .G 1000

Go to address 1000 and begin running code.

---

.L = LOAD FROM TAPE OR DISK

EXAMPLE: .L "FILE NAME",XX

The XX, is a two digit device address (01 = cassette, 08 = disk). This command leaves BASIC pointers unchanged.

---

S = SAVE TO TAPE OR DISK

EXAMPLE: .S "NAME",XX,ZZZZ,YYYY

The XX is a two-digit device address (01=cassette, 08=disk), ZZZZ is the starting address and YYYY is the ending address of the program. A one (1) must always be added to YYYY.

---

M = MEMORY DISPLAY

EXAMPLE: .M XXXX ZZZZ

The XXXX's are the 4-digit starting address and the ZZZZ's are the 4-digit ending address. Enter both addresses as hexadecimal numbers. The contents of the memory locations XXXX thru ZZZZ will be displayed to the screen monitor.

---

D = DISASSEMBLE

EXAMPLE: .D XXXX ZZZZ

The contents of memory from XXXX (starting address) to ZZZZ (ending address) may be displayed with this command.

X = EXIT TO BASIC

Return to basic ready mode. The stack value saved when entered will be restored. Care should be taken that this value is the same as when the monitor was entered. A CLR in basic will fix any stack problems.

---

## Supermon Commands

G = GO RUN  
L = LOAD FROM TAPE OR DISK  
M = MEMORY DISPLAY  
S = SAVE TO TAPE OR DISK  
X = EXIT TO BASIC

### SUPERMON ADDITIONAL INSTRUCTIONS

A = SIMPLE ASSEMBLER  
D = DISASSEMBLER  
F = FILL MEMORY  
H = HUNT MEMORY  
I = SINGLE INSTRUCTION  
P = PRINTING DISASSEMBLER  
T = TRANSFER MEMORY

---



## HEX, DECIMAL, & BINARY

A person can program a long time, without understanding hexadecimal, decimal, or binary. But comes a time, in every user's life when questions begin to arise. Frankly, I found my basic math enough to survive, for forty plus years. That is to say, until the computer bug hit hard. Inadequacy does not even properly describe the feeling I had, when confronted with higher math terminology. How could one have missed so much in school. The computer books all seem to assume, that user's are full fledged mathematicians. Just because the person that wrote the book is well versed in the arts of math, doesn't mean everyone else stayed awake during class.

The rest of this chapter is devoted to the subject of hexadecimal, decimal, and binary. It will not be a math course, I leave that to those qualified to teach.

**BINARY...**This is the native language of the computer. A single unit of memory is known as a bit. Bit is an abbreviation for binary digit, a unit of information equal to one binary decision. Bits are placed in groups of eight to form one byte (8 bits = 1 byte). The only numerals used are 0 (off) and 1 (on), each represents one small piece of information. As these are the only two digits ever used in binary, the system is known as a base 2. This is a good language for the computer, but it becomes inconvenient for most programmers. Hexadecimal is better understood by we humans.

**DECIMAL...**Decimal is a base 10, numbers 0 through 9 are used. Decimal is used when programming in BASIC for POKES, PEEKS, SYS and USR commands.

HEXADECIMAL...Hexadecimal is a base 16, numbers 0 through 16 are used. The first ten digits are represented by 0 through 9, and the last six digits are represented by A,B,C,D,E, and F. A dollar sign is usually placed before a hexadecimal number for easy identification. This form of numbering is often referred to as a short-hand for binary numbers, or patterns. The system saves time and reduces errors which might otherwise be caused by, a confusion of ones and noughts.

### CONVERTING BINARY TO DECIMAL OR HEXADECIMAL

Binary may be converted to hexadecimal or decimal in groups of four. The size of the numbers which can be represented using the hexadecimal, decimal and binary-coded decimal numbering systems, is limited to 255 for each byte.

The following table illustrates how useful hexadecimal can be as a shorthand method of writing binary numbers.

BINARY	DECIMAL(base 10)	HEXADECIMAL(base 16)
0000	0	\$0
0001	1	\$1
0010	2	\$2
0011	3	\$3
0100	4	\$4
0101	5	\$5
0110	6	\$6
0111	7	\$7
1000	8	\$8
1001	9	\$9
1010	10	\$A
1011	11	\$B
1100	12	\$C
1101	13	\$D
1110	14	\$E
1111	15	\$F



CONVERSION TABLE HEX, DECIMAL, BINARY

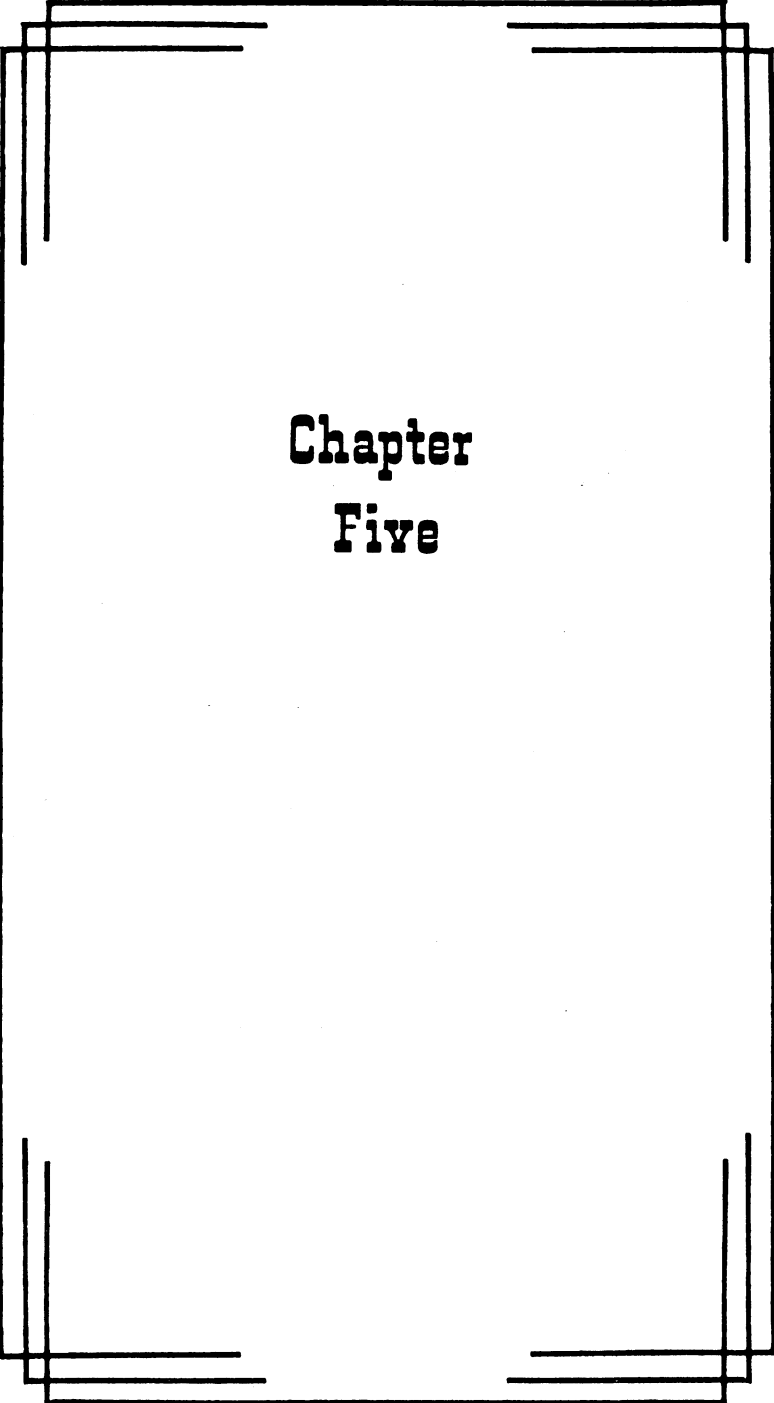
DEC	HEX	BINARY	DEC	HEX	BINARY
0	\$00	00000000	38	\$26	00100110
1	\$01	00000001	39	\$27	00100111
2	\$02	00000010	40	\$28	00101000
3	\$03	00000011	41	\$29	00101001
4	\$04	00000100	42	\$2A	00101010
5	\$05	00000101	43	\$2B	00101011
6	\$06	00000110	44	\$2C	00101100
7	\$07	00000111	45	\$2D	00101101
8	\$08	00001000	46	\$2E	00101110
9	\$09	00001001	47	\$2F	00101111
10	\$0A	00001010	48	\$30	00110001
11	\$0B	00001011	49	\$31	00110001
12	\$0C	00001100	50	\$32	00110010
13	\$0D	00001101	51	\$33	00110011
14	\$0E	00001110	52	\$34	00110100
15	\$0F	00001111	53	\$35	00110101
16	\$10	00010000	54	\$36	00110110
17	\$11	00010001	55	\$37	00110111
18	\$12	00010010	56	\$38	00111000
19	\$13	00010011	57	\$39	00111001
20	\$14	00010100	58	\$3A	00111010
21	\$15	00010101	59	\$3B	00111011
22	\$16	00010110	60	\$3C	00111100
23	\$17	00010111	61	\$3D	00111101
24	\$18	00011000	62	\$3E	00111110
25	\$19	00011001	63	\$3F	00111111
26	\$1A	00011010	64	\$40	01000000
27	\$1B	00011011	65	\$41	01000001
28	\$1C	00011100	66	\$42	01000010
29	\$1D	00011101	67	\$43	01000011
30	\$1E	00011110	68	\$44	01000100
31	\$1F	00011111	69	\$45	01000101
32	\$20	00100000	70	\$46	01000110
33	\$21	00100001	71	\$47	01000111
34	\$22	00100010	72	\$48	01001000
35	\$23	00100011	73	\$49	01001001
36	\$24	00100100	74	\$4A	01001010
37	\$25	00100101	75	\$4B	01001011

DEC	HEX	BINARY	DEC	HEX	BINARY
76	\$4C	01001100	116	\$74	01110100
77	\$4D	01001101	117	\$75	01110101
78	\$4E	01001110	118	\$76	01110110
79	\$4F	01001111	119	\$77	01110111
80	\$50	01010000	120	\$78	01111000
81	\$51	01010001	121	\$79	01111001
82	\$52	01010010	122	\$7A	01111010
83	\$53	01010011	123	\$7B	01111011
84	\$54	01010100	124	\$7C	01111100
85	\$55	01010101	125	\$7D	01111101
86	\$56	01010110	126	\$7E	01111110
87	\$57	01010111	127	\$7F	01111111
88	\$58	01011000	128	\$80	10000000
89	\$59	01011001	129	\$81	10000001
90	\$5A	01011010	130	\$82	10000010
91	\$5B	01011011	131	\$83	10000011
92	\$5C	01011100	132	\$84	10000100
93	\$5D	01011101	133	\$85	10000101
94	\$5E	01011110	134	\$86	10000110
95	\$5F	01011111	135	\$87	10000111
96	\$60	01100000	136	\$88	10001000
97	\$61	01100001	137	\$89	10001001
98	\$62	01100010	138	\$8A	10001010
99	\$63	01100011	139	\$8B	10001011
100	\$64	01100100	140	\$8C	10001100
101	\$65	01100101	141	\$8D	10001101
102	\$66	01100110	142	\$8E	10001110
103	\$67	01100111	143	\$8F	10001111
104	\$68	01101000	144	\$90	10010000
105	\$69	01101001	145	\$91	10010001
106	\$6A	01101010	146	\$92	10010010
107	\$6B	01101011	147	\$93	10010011
108	\$6C	01101100	148	\$94	10010100
109	\$6D	01101101	149	\$95	10010101
110	\$6E	01101110	150	\$96	10010110
111	\$6F	01101111	151	\$97	10010111
112	\$70	01110000	152	\$98	10011000
113	\$71	01110001	153	\$99	10011001
114	\$72	01110010	154	\$9A	10011010
115	\$73	01110011	155	\$9B	10011011

DEC	HEX	BINARY	DEC	HEX	BINARY
156	\$9C	10011100	196	\$C4	11000100
157	\$9D	10011101	197	\$C5	11000101
158	\$9E	10011110	198	\$C6	11000110
159	\$9F	10011111	199	\$C7	11000111
160	\$A0	10100000	200	\$C8	11001000
161	\$A1	10100001	201	\$C9	11001001
162	\$A2	10100010	202	\$CA	11001010
163	\$A3	10100011	203	\$CB	11001011
164	\$A4	10100100	204	\$CC	11001100
165	\$A5	10100101	205	\$CD	11001101
166	\$A6	10100110	206	\$CE	11001110
167	\$A7	10000111	207	\$CF	11001111
168	\$A8	10101000	208	\$D0	11010000
169	\$A9	10101001	209	\$D1	11010001
170	\$AA	10101010	210	\$D2	11010010
171	\$AB	10101011	211	\$D3	11010011
172	\$AC	10101100	212	\$D4	11010100
173	\$AD	10101101	213	\$D5	11010101
174	\$AE	10101110	214	\$D6	11010110
175	\$AF	10101111	215	\$D7	11010111
176	\$B0	10110000	216	\$D8	11011000
177	\$B1	10110001	217	\$D9	11011001
178	\$B2	10110010	218	\$DA	11011010
179	\$B3	10110011	219	\$DB	11011011
180	\$B4	10110100	220	\$DC	11011100
181	\$B5	10110101	221	\$DD	11011101
182	\$B6	10110110	222	\$DE	11011110
183	\$B7	10110111	223	\$DF	11011111
184	\$B8	10111000	224	\$E0	11100000
185	\$B9	10111001	225	\$E1	11100001
186	\$BA	10111010	226	\$E2	11100010
187	\$BB	10111011	227	\$E3	11100011
188	\$BC	10111100	228	\$E4	11100100
189	\$BD	10111101	229	\$E5	11100101
190	\$BE	10111110	230	\$E6	11100110
191	\$BF	10111111	231	\$E7	11100111
192	\$C0	11000000	232	\$E8	11101000
193	\$C1	11000001	233	\$E9	11101001
194	\$C2	11000010	234	\$EA	11101010
195	\$C3	11000011	235	\$EB	11101011

DEC	HEX	BINARY	DEC	HEX	BINARY
236	\$EC	11101100	246	\$F6	11110110
237	\$ED	11101101	247	\$F7	11110111
238	\$EE	11101110	248	\$F8	11111000
239	\$EF	11101111	249	\$F9	11111001
240	\$F0	11110000	250	\$FA	11111010
241	\$F1	11110001	251	\$FB	11111011
242	\$F2	11110010	252	\$FC	11111100
243	\$F3	11110011	253	\$FD	11111101
244	\$F4	11110100	254	\$FE	11111110
245	\$F5	11110101	255	\$FF	11111111

-----  
NOTES  
-----



**Chapter  
Five**

## READ A GOOD DISK LATELY ?



The easy approach to making an archival copy, would be a full disk copy program. Swap a disk in the drive, three times (single disk drive), four minutes, finished, one archival copy. This sounds good, but in reality no single program can copy all types of the types of protection. Many advertisements imply that it is possible, but regardless of the optimism, it can't be done.

As you learn more about the diskette and the drive, you will understand the problems that arise in the development of this type of software.

As a user, you are caught in the spiral of new protection methods, followed by new products to break them. The same applies to the author, they protect software and someone else sells a program to break it. I am beginning to wonder who really makes the profit, the author of the program, or the author of the copy program? It soon becomes like a cat after its own tale.

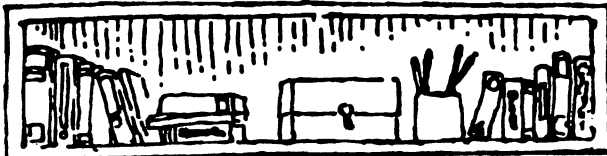
The only solution left, is to learn how to protect, or un-protect a disk yourself. In order to fully understand the process involved, you must first learn to read a disk, and know what you are seeing. On the disk

provided with this book, are tools to obtain information and modify a disk. The collection covers a full range of subjects pertaining to disk protection. Appendix A contains a description of each program, plus the instructions as to how to run them.

Track 18,0 is very special to the 1541 disk drive. This is where a vast amount of information is stored and read by the drive. This track is known as the BAM or the BLOCK ALLOCATION MAP. For the rest of our discussion, it shall be referred to as the BAM. Things that may be found in the BAM are:

1. Pointer to the next sector of directory.
2. DOS version used to write the disk.
3. BAM map, shows number of blocks used.
4. The name of the disk.
5. ID number of the disk.
6. DOS version and format type.

As you can see, the BAM is the beginning to the secrets of a diskette. Numerous changes can be made to this area, all in the name of program protection. Figure A, on page ~~37~~<sup>46</sup>, is the BAM of the disk included with this book. All numbers are in hexadecimal. Below the the BAM example (figure B) is a grid that will be used to identify particular areas of the BAM in the following text.





TRACK 18

SECTOR 0

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	12014100	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	0
1	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	1
2	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	0C555715	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	2
3	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	3
4	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	4
5	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	5
6	00000000	0FFE303	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	6
7	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	11FFFF01	7
8	11FFFF01	11FFFF01	11FFFF01	11FFFF01	11FFFF01	11FFFF01	11FFFF01	11FFFF01	11FFFF01	11FFFF01	11FFFF01	11FFFF01	11FFFF01	11FFFF01	11FFFF01	11FFFF01	8
9	50524F47	52414D53	2046494E	A0A0A0A0	A0A0A0A0	A0A0A0A0	A0A0A0A0	A0A0A0A0	A0A0A0A0	A0A0A0A0	A0A0A0A0	A0A0A0A0	A0A0A0A0	A0A0A0A0	A0A0A0A0	A0A0A0A0	9
A	A0A05046	A03241A0	A0A0A000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	A
B	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	B
C	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	C
D	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	D
E	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	E
F	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	F

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

## Next Sector

The example shown below is square number one of the grid, a part of the BAM, track 18, sector 0.

	0	1	2	3
0	12	01	41	00
1	15	FF	FF	1F
2	15	FF	FF	1F
3	00	00	00	00

The bytes of information are arranged from left to right.

ALL ARE IN HEXIDECIMAL

Starting at the upper left, row one, is the number twelve, this equals 18 in decimal. This is the pointer to the next track of the file. The second byte in row one, 01 equals 1 in decimal. From the first two bytes in this file, we can determined that the next block of the file is located at track 18, sector 1.

## DOS Version

The example shown below is square number one of the grid, a part of the BAM, track 18, sector 0.

	0	1	2	3
0	12	01	41	00
1	15	FF	FF	1F
2	15	FF	FF	1F
3	00	00	00	00

The next byte (third) is the number 41 or in decimal 65. This byte is also equivalent to the letter 'A' in ASCII. It must read 'A' as it is checked every

time a disk is read. If this is not the letter 'A', the error message " 73,DOS MISMATCH,18,00" is generated. If the 'A' has been changed to the letter 'E' (hexidcimal 45, decimal 69) you will not be able to write information to the disk. The computer has been fooled to think that the disk was formatted on a drive not compatable with the 1541. This is a very popular form of protection. The fourth byte is not used and usually reads 00.

# Bam Map

The example shown below is square number two of the grid, a part of the BAM, track 18, sector 0.

The next 140 bytes contain the Block Allocation Map (BAM). Each group of four bytes contain the BAM of one track. Before looking at the entire BAM, let's examine one square of the grid (figure \*\*).

	4	5	6	7
0	15	FF	FF	1F
1	15	FF	FF	1F
2	15	FF	FF	1F
3	00	00	00	00

The first row is the BAM for track number one.

The 15 (21 decimal) equals the blocks free in this track. The last three

bytes represent sectors

free. There are a total of 21 sectors available in track one. The three bytes contain a bit map for the track. Bytes FF, FF, and F1 contain 21 1's (in binary). When a sector is used the disk drive automatically tells the DOS to change the proper 1 to a 0. This is known as allocating a block. The BAM is updated each time you save or scratch a program.

Now we are ready to examine the entire BAM. Test yourself by reading track 23.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	12014100	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	0
1	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	1
2	15FFFF1F	15FFFF1F	15FFFF1F	15FFFF1F	0C555715	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	2
3	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	3
4	00000000	00000000	00000000	00000000	0C6CDB06	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	4
5	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	5
6	00000000	0FFE303	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	6
7	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	12FFFF03	7

S	11FFFF01	11FFFF01	11FFFF01	11FFFF01	S
9	50524F47	52414D53	2046494E	A0A0A0A0	9
A	A0A05046	A03241A0	A0A0A000	00000000	A
B	00000000	00000000	00000000	00000000	B
C	00000000	00000000	00000000	00000000	C
D	00000000	00000000	00000000	00000000	D
E	00000000	00000000	00000000	00000000	E
F	00000000	00000000	00000000	00000000	F

## Disk Name

When formatting a disk, you enter a name. For you this may be up to 16 characters in length, but for the computer it must be 16 bytes long. The disk drive will automatically fill any unused spaces with shifted spaces (A0) as fillers. Bytes 144 to 159 contain the disk name as shown below. Row nine, squares nine, ten, eleven, and twelve of the grid are the locations of bytes 144 to 159.

9 50524F47 52414D53 2046494E A0A0A0A0

Notice at the end of the name four A0's have been added as fillers. → (60)

## Disk I.D.

Bytes 162 and 163 are reserved for a two character disk ID. The ID is entered by the user when formatting a disk. This same ID is placed in the header of each block of the disk, plus this location in the BAM. The example below is from the grid, row A, square nine.

A A0A05046 A03241A0 A0A0A000 00000000

NOTE: bytes 160, 161, and 164 are shifted spaces (A0) ← (60)

↳ CHR# 160

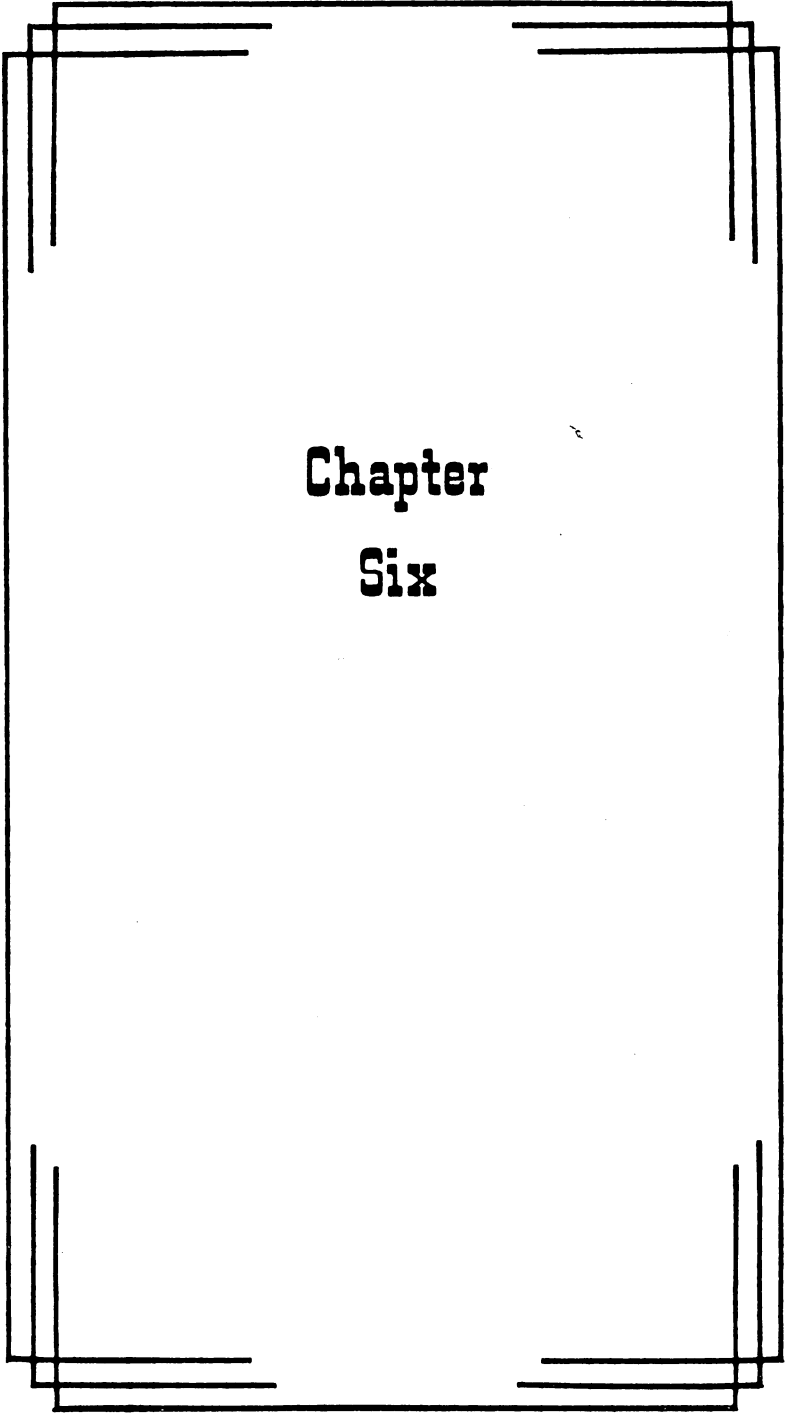
## DOS & Format Type

Bytes 165 and 166 contain the ASCII equivalent of 2A, this represents the DOS version of the 1541. This same DOS is also used by the 2031 and 4040 Commodore disk drives. The following example is from the grid, row A, square ten.

A A03241A0

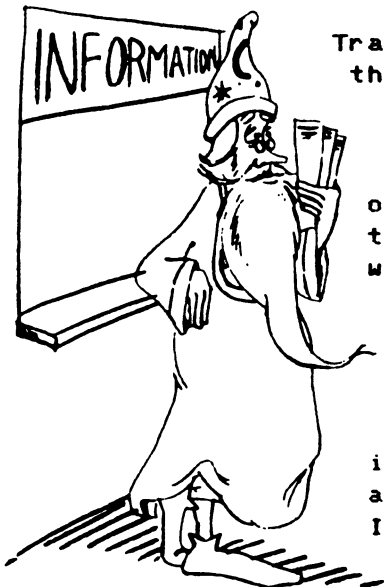
NOTE: bytes 167 to 170 are shifted spaces (A0).

Bytes 171 to 255 are not normally used. BUT the user may use this area for machine language or other messages. This is one of those great little hiding spots.



**Chapter  
Six**

## TRACK 18...HEAD LIBRARIAN



Track 18 is the librarian of the diskette. The names of the files are stored here very much like a book on a shelf. This location is often the key to disk protection. Learn to read it well, it hold many secrets of the disk.

The directory is made up during the formatting of a diskette. All of the information on track 18 is a part of the directory. It could be compared to an index or table of contents of a book. Start-

ing on track 18, sector 1, is the list of actual files contained on the disk. The maximum number of individual file entries is 144. All of the following information is included in the directory:

1. Next block of the directory file.
2. File entry.
3. Type of file.
4. Beginning track & sector of file.
5. Name of file.
6. Length of file.

The 1541 disk drive is really a very fast and efficient librarian.

TRACK :18

SECTOR :01

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	12048211	0A4D454E	55A0A0A0	A0A0A0A0	0											
1	A0A0A0A0	A0000000	00000000	00000500	1											
2	00008211	0B2D2D2D	2D2D2D2D	2D2D2D2D	2											
3	2D2D2D2D	2D000000	00000000	00000100	3											
4	00008211	0C454449	54205420	262053A0	4											
5	A0A0A0A0	A0000000	00000000	00002000	5											
6	00008213	0A2D2D2D	2D2D2D2D	2D2D2D2D	6											
7	2D2D2D2D	2D000000	00000000	00000100	7											
8	00008213	0B564945	57204241	4DA0A0A0	8											
9	A0A0A0A0	A0000000	00000000	00000700	9											
A	00008213	0F2D2D2D	2D2D2D2D	2D2D2D2D	A											
B	2D2D2D2D	2D000000	00000000	00000100	B											
C	00008213	104E4557	20444953	4B205449	C											
D	544C45A0	A0000000	00000000	00000300	D											
E	00008213	122D2D2D	2D2D2D2D	2D2D2D2D	E											
F	2D2D2D2D	2D000000	00000000	00000100	F											

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	.	.	.	.	M	E	N	U								
1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
4	.	.	.	.	E	D	I	T	T							
5	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
6	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
7	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
8	.	.	.	.	V	I	E	W	B	A	M					
9	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
A	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
B	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
C	.	.	.	.	N	E	W									
D	T	L	E	.	.	.	.	.	.	.	.	.	.	.	.	.
E	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
F	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.



Figure C (page 44) is track 18, sector 1 in hexadecimal. Figure D (page 44) is the same track and sector in ASCII. The same grid system as shown on page 37 (figure B) will be used in all of the following examples.

## Next Block of DIR

The example shown below is square number one of the grid, a part of the BAM, track 18, sector 1.

0	12048211
1	A0A0A0A0
2	00008211
3	20202020

The first two bytes of track 18, sector 1, are the pointers to the track and sector of the next block of the file. In the example row one, the first two

bytes are \$12 and \$04 (hexadecimal). These must be changed to decimal to find the location we are after. Byte one (\$12) equals 18 and byte two (\$04) equals 4, so the next block of the file will be found at track 18, sector 4. If a file is only one block in length, byte one would read \$00, byte two \$FF. Just remember, byte one is the track and byte two is the sector.

## File Entry

The names, type, location, and length of the individual files on a disk are listed on track 18 as follows:

Bytes	2 thru	31:	file number	1
Bytes	34 thru	63:	file number	2
Bytes	66 thru	95:	file number	3
Bytes	98 thru	127:	file number	4
Bytes	130 thru	159:	file number	5
Bytes	162 thru	191:	file number	6
Bytes	194 thru	223:	file number	7
Bytes	226 thru	255:	file number	8

As the directory grows and more room is needed, files titles are placed on other sectors in the following order:

- 1.....4.....7.....10....13....16
- 2.....5.....8.....11....14....17
- 3.....6.....9.....12....15....18

The order of the bytes is the same on each sector, byte 0 is the next track, byte 1 is the sector. When the last sector of the directory is reached, byte 0 will read \$00 and byte one \$FF denoting the end.

## File Type

The example shown below is line zero of Figure C (page 44).

```
      0 1 2 3 4 5 6 7 8 9 A B C D E F
ROW 0 12048211 0A4D454E 55A0A0A0 A0A0A0A0
```

Byte two tells us what type of file is stored on the disk. The following is a list of the different types of files that might be found in this location.

FILE TYPE	DIR/APPEARS	HEX	ASCII
Scratched	not shown	\$00	0
Deleted (DEL)	DEL	\$80	128
Sequential (SEQ)	SEQ	\$81	129
Program (PRG)	PRG	\$82	130
User (USR)	USR	\$83	131
Relative (REL)	REL	\$84	132
Open DEL	not shown	\$00	0
Open SEQ	*SEQ	\$01	1
Open PRG	*PRG	\$02	2
Open USR	*USR	\$03	3
Open REL	not apply	\$04	4
DEL @ replacement	DEL	\$A0	160
SEQ @ replacement	SEQ	\$A1	161
PRG @ replacement	PRG	\$A2	162
USR @ replacement	USR	\$A3	163

REL @ replacement	not apply	\$A4	164
Locked DEL	DEL<	\$C0	192
Locked SEQ	SEQ<	\$C1	193
Locked PRG	PRG<	\$C2	194
Locked USR	USR<	\$C3	195
Locked REL	REL<	\$C4	196

## 1st. Track & Sector

Again look at line zero of Figure C (page 44), it is listed below:

```
      0 1 2 3 4 5 6 7 8 9 A B C D E F
ROW 0 | 12048211 0A4D454E 55A0A0A0 A0A0A0A0
```

Bytes three and four, identify the track and sector of the first block of information in the program named 'Menu'. Hexadecimal \$11 equals 17 decimal. Hexadecimal \$0A equals 10 decimal. The program begins at track 17, sector 10.

## File Name

Row zero shown below, is listed in hexadecimal. Bytes five through twenty, contain the name of the first file on the diskette. The title must be sixteen characters in length. If it is less than sixteen characters, the disk drive will automatically fill the unused spaces with shifted spaces. These spaces are fillers only, and do not effect the title of the file.

```
      0 1 2 3 4 5 6 7 8 9 A B C D E F
ROW 0 | 12048211 0A4D454E 55A0A0A0 A0A0A0A0
```

## File Length

Row zero shown below, is listed in hexadecimal. Bytes thirty and thirty-one contain a count of the number of blocks used by a file. The formula to find the size of

file is: first convert bytes 30 and 31 to decimal, byte 30 + byte 31 \* 256 = file size. In the case of the file below:

\$08 hex = 8 decimal    \$00 hex = 0 decimal

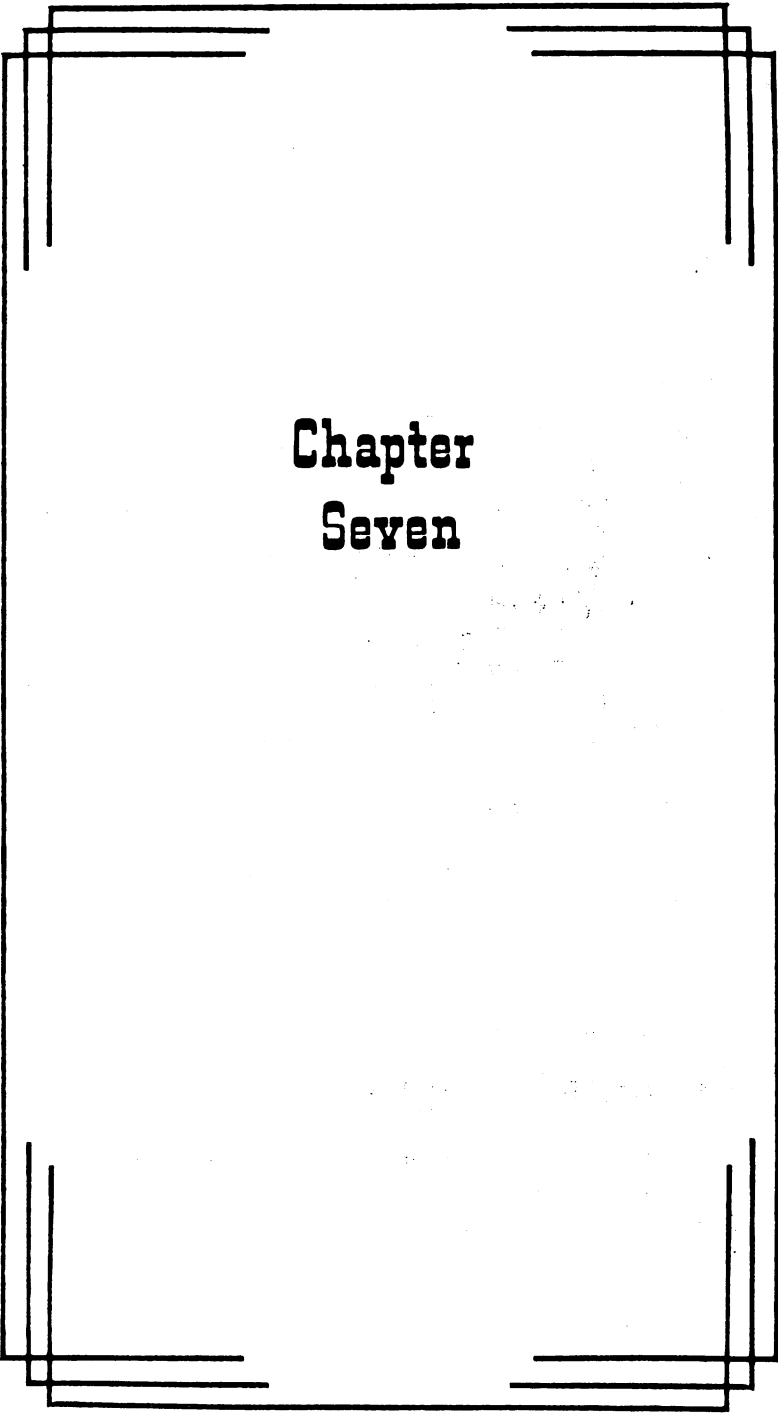
$$8 + 0 * 256 = 8$$

The file is 8 blocks long

As you may have noticed, bytes have been skipped, some are used and some of them are not. Bytes twenty-one thru twenty-three are for relative files only. Bytes twenty-four thru twenty-seven are un-used. Bytes twenty-eight and twenty-nine are temporary storage during a SAVE and REPLACE. All file entries will have the same format. Learn it once and you can read them all.

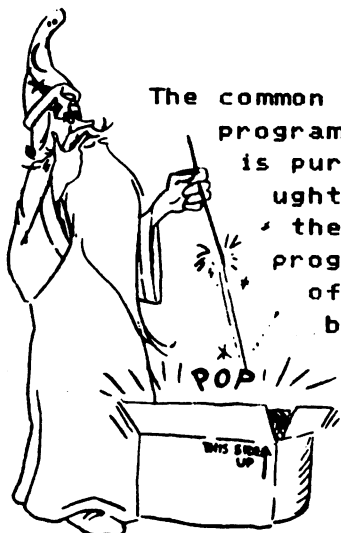
To some, these facts might not be as interesting as reading a good book, but like a book you can obtain a lot of good information from track 18.

-----  
NOTES  
-----



**Chapter  
Seven**

# BASIC PROGRAM PROTECTION



The common misconception, that a basic program can not be well protected, is pure fantasy. My personal thoughts on the matter, are just the opposite. Give me a basic program, I can confuse the best of the hackers. The code will become such a complex mess, most will cry in despair. The time involved to solve the puzzle, is too long to justify the end results, not to mention, the damage to their sanity.

A program written in BASIC may be protected in several ways. Often more than one scheme, at a time, is used to confuse the pirate. Frankly, the more the better. To mislead or puzzle, is the name of the game. The following ideas do not stop the user from making a legal back-up, but do add confusion when someone tries to alter code.

## Compiled Programs

The first old standby that comes to mind is compiling. Compiling a BASIC program is usually a reliable way to discourage beginning hackers. When compiled, the BASIC code is converted to pseudo code, commonly referred to as P code. To the novice this appears as garbage. Once discovered most will divert their efforts to something else. But there are

exceptions to that rule, the de-compiler. A de-compiler brings a compiled program back to its original state, just as if compiling had never taken place. At this time, de-compiler's are not available for all brands of compiler. I feel certain that they will soon be on the market. Still, not everyone will have the proper software for the job. Combined with a few other tricks compiling can still slow down the pirate.

## Pokes & Peeks

Most BASIC programs are commonly protected by POKES and PEEKs. These effect the program once it has been run. What follows is a list of accepted POKES and what they do. The list is a start, but not all that may be found. Curious programmer's are finding un-documented locations all the time.

POKE 1,0.....To disable the operating system.  
Default value: 1,1

POKE 198,0....To clear the keyboard buffer.  
Default: 198,0

POKE 649,0....To disable the keyboard.  
Default value: 649,10

POKE 650,0....To disable all keys, not cursor.  
Default value: 650,128

POKE 650,64...To disable repeat of all keys.  
Default value: 650,128

POKE 774,0....To vanish LISTing, leave line #.  
Default value: 774,26

POKE 774,141..To vanish LISTing completely.  
Default value: 774,26

POKE 775,168..To stop user from LISTing.Prints  
?UNDEF'D STATEMENT ERROR  
Default value: 775,167

POKE 775,171..To cause the computer to crash  
if a LIST attempt is made.  
Default value: 775,167

POKE 775,200..To disable LIST.  
Default value: 775,167



- POKE 788,52...To disable the STOP Key and TI\$.  
Default value: 788,49
- POKE 792,193..To disable STOP/RESTORE.  
Default value: 792,71
- POKE 793,203..To disable RESTORE.  
Default value: 793,254
- POKE 808,225..To disable the RUN/STOP-RESTORE.  
Default value: 808,237
- POKE 808,230..To disable a proper LISTing.  
Default value: 808,237
- POKE 808,232..To stop LIST or SAVE, program  
may not be stopped during run.  
Default value: 808,237
- POKE 808,234..To disable the RUN/STOP-RESTORE  
Keys.  
Default value: 808,237
- POKE 808,239..To disable the RUN/STOP Key.  
Default value: 808,237
- POKE 818,32...To disable the SAVE function.  
Default value: 818,237
- POKE 819,246..To stop all attempts to SAVE. A  
Ready message will be displayed.  
to the screen.  
Default value: 819,245

## No Line Numbers

This poke is one slick trick. Once a program has been run, it makes all the line numbers vanish. This is one of the few tricks, that will also prevent the code, from being listed to the printer. The program will always run as normal. To reset, cause a ?SYNTAX ERROR from the keyboard.

```
EXAMPLE: 10 PRINT " HELLO"  
         20 PRINT " THIS IS A"  
         30 PRINT " COMMODORE"  
         40 PRINT " COMPUTER."  
         50 POKE 22,35
```

The poke in line number 50, must be the last line of your program. Try it, this is a

must in your bag of tricks. Anyone trying to alter code, can become hopelessly lost forever.

## Cassette Buffer

A program may poke values into the memory locations of the cassette buffer. A short machine language routine can be placed at this address. This is not normally protection, but it may be confusing to the beginner to see an SYS to this location. Machine language is used to speed up the operation of the BASIC program.

## Move Start of Basic

Moving the start of BASIC pointers is a method of hiding a BASIC program. Locations 43 and 44 tell BASIC where text has been stored. Type the following line to verify the location of the start of BASIC text:

```
10 PRINT PEEK (43) + 256 * PEEK (44)
```

If everything is normal and the pointer has not been changed, the result will be 2049. Location 2049 is where the BASIC program text is stored until needed.

Now lets move BASIC to a higher location in memory. Enter the following lines of code.

```
10 A = 4096: REM NEW LOCATION
20 B = A - 1: REM NEW LOCATION LESS ONE
30 C = INT (A/256): REM HIGH BYTE
40 D = A - C * 256: REM LOW BYTE
50 POKE B,0:POKE 43,D:POKE 44,C: CLR: NEW
```

To test that BASIC has moved to the new location of 4096, again enter the test line.

```
10 PRINT PEEK (43) + 256 * PEEK (44)
```

This is just a diversion, but it can add confusion and pain to the life of a pirate.

## Auto Load & Run

Programs with automatic load and run are not difficult to break, but do add another touch of protection. On the disk of Public Domain software is a program that creates a boot with these features. Load and run "BOOT.MAKER". Insert the disk in the drive that contains the program you wish a boot made for. Answer the prompts, enter the name you desire for the boot. Next enter the name of the program to be loaded by the boot. Press return after each prompt, the program will construct the desired boot on your disk. When the boot is loaded and run the main source code will automatically be loaded and run without further user action.

**WARNING:** The program 'BOOT.MAKER' writes to the disk. Be sure there are enough blocks free on the disk for the addition of the boot.

## Rems....

With REM statements, we get to have some fun and add creative surprises to our programs. This can be one of the most puzzling forms of protection for the BASIC programmer that exists. Did you know that you can prevent someone from easily listing your program with a REM statement? Most books tell you that REMs are for remarks or comments only, and are ignored at any other time. WRONG, WRONG, WRONG, the REM has many hidden secrets that are not usually documented.

We start our REM education with three golden rules. They are as follows:

1. Un-shifted characters appear as typed.

2. Shifted characters are converted to BASIC command tokens if the ASCII code for the character is equivalent to a BASIC command token.
3. Reverse fields are stripped from any character.

To illustrate the above rules, type in the following four lines of code, and then type LIST.

```
10 REM ABCDEF
20 REM ABCDEF (shifted characters)
30 REM [RVS] ABCDEF [OFF]
40 REM [RVS] ABCDEF [OFF] (shifted characters)
```

Line 10 demonstrates rule one. All characters look normal.

Line 20 has changed. This shows rule two.

Lines 30 and 40 look just like lines 10 and 20, because the reverse field was stripped, rule number three.

## Rems.... Listings

Now wouldn't it be nice to prevent someone from listing a program? With a REM it can be done. Strange effects happen when shifted letters are used with a REM. Type in the following small program and we will begin the experiment. Be sure to SAVE this code once you have entered it, we will be using this example repeatedly. Each time it is used, reload a fresh copy into memory. This will save a lot of time.

```
100 PRINT"HELLO, MY NAME IS"
110 PRINT"THE C-64, WHAT IS"
120 PRINT"YOUR NAME ";
130 INPUT N$
```

```
140 PRINT
150 PRINT"HI ";N$
```

List the program, everything look like always. Next add the following:

```
115 REM [SHIFTED] L
```

Again list the program, lines 120 to 150 are missing. The letter L is the only letter of the alphabet to have this strange effect. Who would expect to get a 'syntax error' when all they did was the LIST command. If this is the first line in a program, all code after it will disappear. Try adding a line 90 with REM shifted L and you will see this in action. Now the bad news. If you list the program to a printer it will list as normal, or someone could LIST line 100 and see the code. But never fear there more tricks with REMs to try.

## Rems... No List, No Print

Have you ever listed a program only to see a screen full of basic commands? This effect could make a person think twice before trying to alter your program. The special line of code must be the first line in your program. Before adding the line, finish the file and do all the de-bugging you are going to do. Next add the following line as the first line in the program. To test, add the line to the sample code we have been working with.

```
50 REM QWERTYUIOPZXCVBNMASDFGHJKL
```

Enter the REM, the rest of the line is entered while pressing the SHIFT. When listed, you'll see a screen full of basic commands and ?SYNTAX ERROR. The key to what makes this happen is the last character entered. On the C-64, the capital L has no command equivalent, the computer regards it as an error. When

listed to the printer, all that will print is a line of garbage. You can get the correct listing by listing from the line following the trick line, but not everyone will think of doing this.

## Rems... Missing Code

Now back to the same program you typed in above, delete line 50 and enter the following:

```
115 REM" [press return]
```

Next edit the line by returning the cursor to the space following the quote mark, press the 'RVS ON' key. Next press the letter 'T' four times. Now list the program. The REM and the quote mark are gone.

Each 'T' entered, deletes from your view, one character to the left. Just count the number of characters you wish to hide, and next enter the same number of T's. By adding more T's you could make the entire line disappear.

## Rems... Invisible Lines

This one will make the line with the REM and all lines that follow invisible. We will be re-typing line 115 of the sample code.

```
115 REM "[delete 1][RVS ON]TTTTTTT[shifted M][token for the color wanted]
```

The T's remove the REM and the line number. Each color is shown on the screen by a token. Choose the color that is the same as the screen color. Identify the token, and enter after the shifted M.

All lines entered after line 115 will be the same color as the screen, thus invisible.

BLACK.....SHIFTED P	ORANGE.....SHIFTED A
WHITE.....E	BROWN.....SHIFTED U
RED.....£	LIGHT RED.....X
CYAN.....SHIFTED *	DARK GRAY.....SHIFTED W
PURPLE...LOGO KEY -	MED. GRAY.....SHIFTED X
GREEN.....↑	LIGHT GREEN...SHIFTED Y
BLUE....LEFT ARROW	LIGHT BLUE....SHIFTED Z
YELLOW...SHIFTED ↑	LIGHT GRAY.....↑

## Rems... Bogus Lines

Now for some real confusion and fun. How would you like two line 120s? Impossible you say, not with a REM statement. A false line number may be created by adding a new line 115 to the sample code.

```
115 REM"[DELETE KEY] [RVS ON] TTTTTTTT [RVS  
OFF]  
120 GET ZZ$:IF ZZ$="" THEN 120
```

By deleting the last quote after the REM you are placed in edit mode, the reverse T's are back spacing over the code to the left. This leaves only the code to the right of the T's showing. Results, two line 120s, or a bogus line of code.

## Rems... Duplicate Line Numbers

We have been taught, that it is impossible to have two lines in a program with the same number. When the second number is entered, it will replace the first. Wrong, add one little trick and the rule will appear false. With a REM you can fool the user into thinking the impossible. Type the following:

```
115 REM" [delete] [REV ON] TTTTTTT20  
GET C$:IF C$ = "" THEN 120
```

Again the T's have been back spaced over the characters to the left, moving the number

20 next to the character one in the line number. See, you now have two lines numbered 120.

## Rems.... Mixed Line Numbers

Seeing is believing. I know this one sounds crazy but it works. How do you think a person trying to break into someone else's code would feel, if they found the line numbers mixed? Add the following line to your program.

```
145 REM" [delete] [REV ON] TTTTTTTT  
      [shifted] M [shifted] QQQQQQQ
```

## Rems.... Clear Screen

Now we will make all your code disappear in one flash. First you see it and then you don't. Add the following line.

```
155 REM "[delete] [RVS ON] [shifted] M  
      [shifted] S
```

If a person is imaginative, the rems can be used to do some incredible things. Experiment, I am sure you can create a hoard of tricky nasty code, that is virtually impossible to understand.

## Change File Type

A program is a PRG file, but it could be saved to the disk as SEQ or USR file, with the right commands. These same files can then be loaded back into memory and executed as PRG files.

To SAVE a program as a sequential file type the following:

```
SAVE"0:filename,S",8
```



To LOAD the same program back in as a program file type the following:

```
LOAD"0:filename,S",8
```

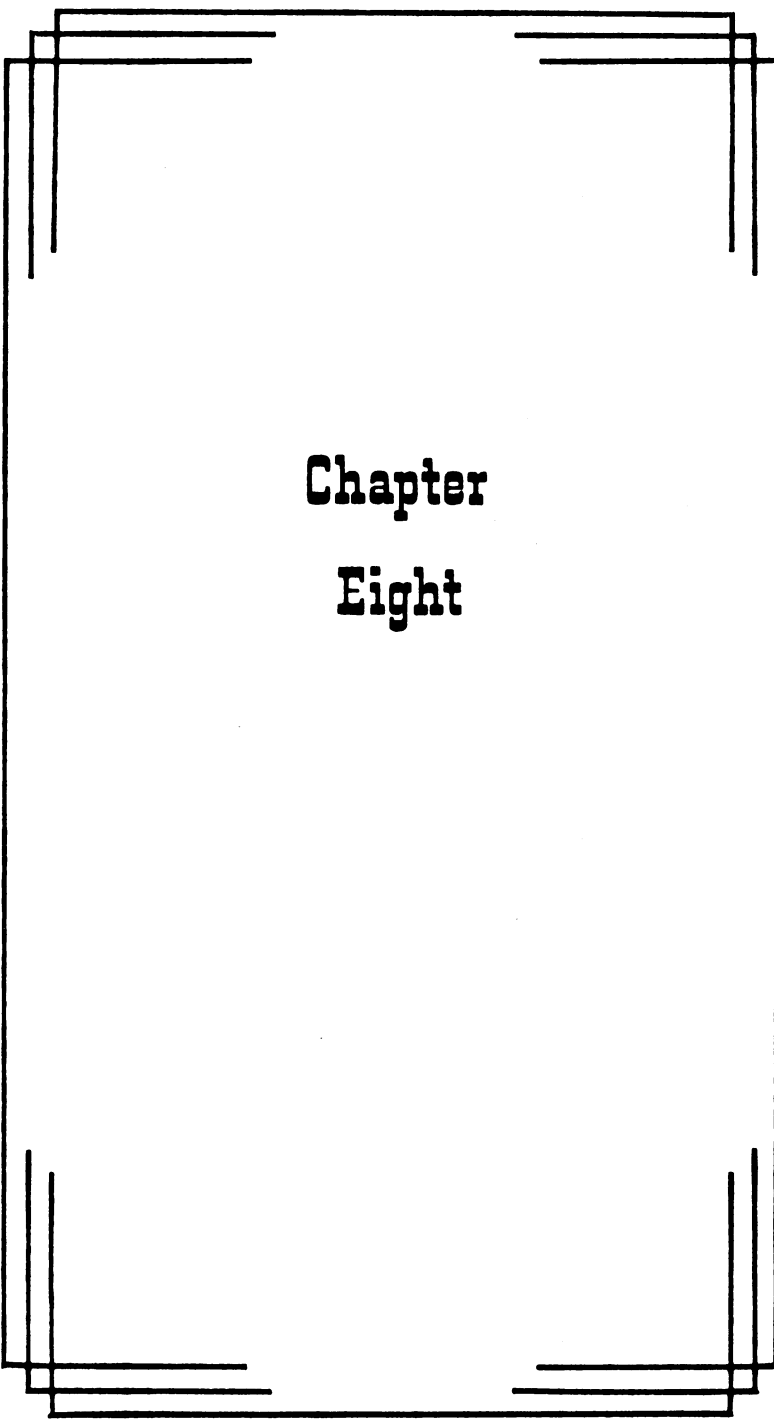
The file will appear in the directory as an SEQ file, but will really be PRG file. To convert to USR files substitute a 'u' for the 's' as shown below:

```
SAVE"0:filename,U",8
```

```
LOAD"0:filename,U",8
```

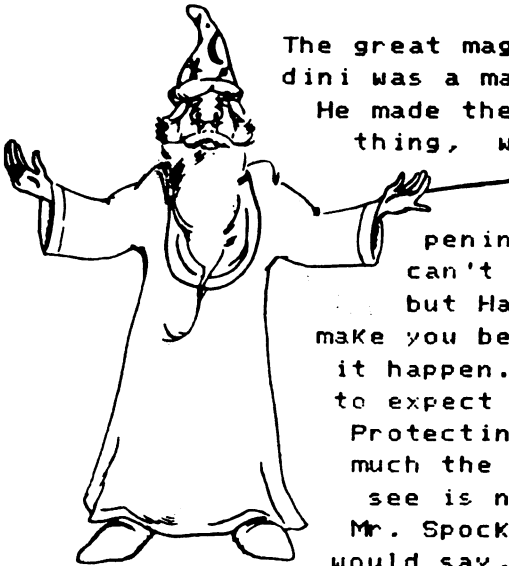
To use this bit of trickery, you might save the program as an SEQ file, and then construct a boot program with the code to load the file as a PRG file. When ever the directory is listed only the SEQ would show.

REMs and other tid-bits of trickery afford the author a wealth of confusion to aim at pirates. One of the technics alone is little protection. But use enough of them and the fun begins. Picture a program that does the following: big line numbers at the top of the code, mixed line numbers, duplicate line numbers, missing code, bogus code, add a few true REMs with GOSUBs going to them. If the code breaker determines the REMs are the problem, and deletes them, good-bye to the GOSUBs. The program will never run again. For even more fun, compile the whole mess. They will really have to work for this one. In chapters that follow, we will add some other tricks that can also be used with the REMs. It all adds up to some pretty good protection.



**Chapter**  
**Eight**

## ALL KINDS OF TRICKS



The great magician, Harry Houdini was a master at illusion.

He made the public think one thing, while in fact, some

thing quite the

reverse was hap-

pening. We all know you

can't see a lady in-half

but Harry Houdini could

make you believe you had seen

it happen. One had to learn

to expect the unexpected....

Protecting a program can be

much the same way, what you

see is not always fact. As

Mr. Spock of Star Trek fame

would say, "highly ill-log-

ical". The two would have been an odd pair in

computer programming. Mr. Spock with strict

logic and Houdini supreme in illusion.

In this chapter, we will take a look at more of the programmer's magic. Small slight of hand, concealed to most, can add confusion and pain to the lives of pirates everywhere. Software protection simply put, is to make your computer perform tricks that are not normally part of it's performance.

We now advance to the more sophisticated protection methods. To experiment with the following modifications, it is advised that you first make a back-up of the disk you are going to work with. If you should make a

mistake, nothing is lost that way. A copy of the Public Domain disk included with this book could be made with the program "4MINUT-V2" for the purpose of trial and error. Mistakes can happen to the best of us.

Some of the techniques, will require the use of the program 'Edit Track & Sector'. It can be found on the public domain disk, which accompanies this book. A list of the program's commands are included in Apendix A.

What follows are limited instructions and pertain only to the subject of changing a byte. Load and run 'Edit Track & Sector'. When prompted, insert the disk you wish to look at, in the drive. Then press return. To change a byte, press F2. In the lower left-hand corner of the screen, you will be ask the byte to change. Enter the byte number to be changed in decimal. Answer the next prompt in hexadecimal. Press return, the change will then appear on the screen. The prompts will re-appear so that you may make more changes if you like. Press 'f' to exit at any time.

When you have completed all the changes you wish to make, return to the main menu. To SAVE the modification to the disk, press the F4 key. Then press return to continue. 'ONE MOMENT' will appear on the screen while the command to SAVE is executed.

## **Write Protected Disk**

The third byte of track 18, sector 0, informs the computer what version of DOS was used to write the disk. Change this byte from the normal 'A' (ASCII) to an 'E' and the computer can be fooled to think the disk was formatted on another type of disk drive. Now the disk is write protected, you may read the disk, but know longer write to it.

The Commodore disk drives, models 4040 and the 2031 are often used to write disks for the 1541. Disks may be interchangeably read with either DOS, but a disk formatted on one version cannot be written upon with the other version because the format is different. New 1541's (made after July 1984) have been modified for write compatibility between diskettes formatted on either the 1541 or the 4040.

## Scratch Protected

Byte two of directory entries, tell us the type of file that is stored on the disk. These can be changed so that the file can no longer be scratched. The following values may be used to replace the original.

WANT	SHOWS AS	CHANGE TO
Locked DEL	DEL<	<sup>HEX</sup> \$C0 <sup>DEC</sup> (192)
Locked SEQ	SEQ<	\$C1 (193)
Locked PRG	PRG<	\$C2 (194)
LockedUSR	USR<	\$C3 (195)
Locked REL	REL<	\$C4 (196)

## False Blocks Free

The fourth byte of track 18, sector 0, tell us the number of blocks free on track 1. By changing this to FF, the drive can be fooled into thinking that track 1 has 255 blocks free. If this were changed for all the tracks, a VERY FALSE number of blocks free would be shown. Remember this is false information and the extra blocks may not be used.

## Directory Change

Bytes 144 thru 161 of track 18, sector 0, contain the name of the disk. By changing the

first six bytes to hexadecimal values: 14 14 14 14 14 14, the directory will no longer list to the monitor or the printer. This technique is standard practice by many of the software manufactures.

Another way to make the directory un-listable is modification of track 18, sector 1. Bytes 0 and 1 are the locations of the pointers to the next track and sector of the file. If this is the last block of the file, it will read 00 FF. By changing these to read 12 01 (decimal 18 01) the directory file will be placed in an endless loop. When you list the directory, the computer will prompt on the screen 'LOADING' for ever and ever. Just think, the command to copy could go on forever.

Bogus files may be added to the directory for more confusion. These are of no value, other than to befuddle the pirate.

## **False I.D.**

The bytes 162 and 163 of track 18, sector 0, contains the I.D. of the disk. Change these I.D. numbers to any characters you want. When the disk was formatted, uniform I.D. was placed on all sectors. Since only track 18, sector 0 has been changed, an error 29, ID MISMATCH will be created by this change.

## **Unusual Load & Save**

When hunting for protection in a program, the normal place to look is at PRG files. If a file were to be saved as an SEQ and then loaded back as an PRG, it could be unusual to say the least. Persons wishing to break into anothers code, seldom bother to look at sequential files. Here is how to do it:

To save as an SEQ type:

```
SAVE"0:FILENAME,S",8
```

To load the file as a PRG file:

```
LOAD"0:FILENAME,S",8
```

If you don't like 'SEQ' files, try using  
USR files for a change. All you have to do is  
substitute a 'U' for the 'S' in the above  
examples.

Convert one or two files of a multiple  
file program in this manner. Hide your major  
protection schemes in these converted files.  
Now the fun begins when hunting for the file  
with the protection.

## Misleading Names

File names are not always what they appear  
to be. Take a look at the following titles  
from a directory. How would you load one of  
the programs?

5	"MENU	PRG
32	"EDIT T & S	PRG
7	"VIEW BAM	PRG
11	"VIEW T & S	PRG
3	"NEW DISK TITLE	PRG
5	"CHANGE DISK ID	PRG
3	"TRACK ID READER	PRG
5	"TRACE FILE LINKS	PRG

This end result takes a different method  
of SAVE. Here is how to achieve the oddity.

```
SAVE" MENU",8
```

The space between the quote and the first  
letter of the filename is a shifted space. The  
only way the average user could load, is by

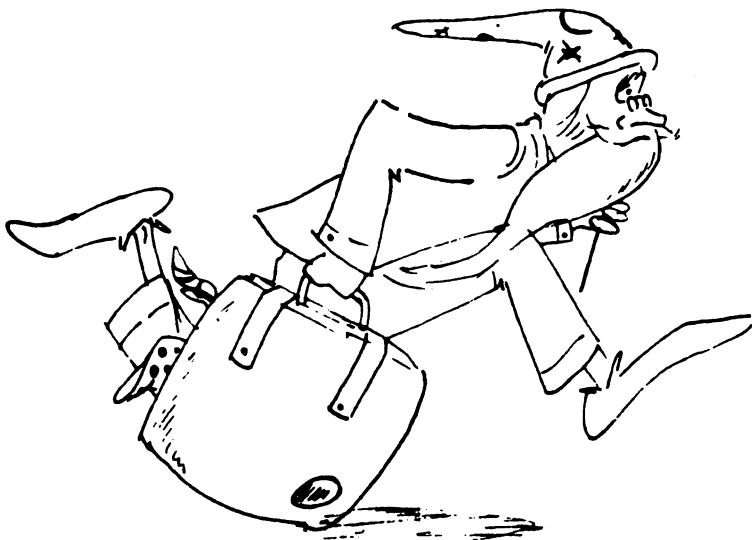
the first file on the disk:

LOAD"\*",8

If a person knows the trick, a file can be loaded in the same manner as it was saved. Don't forget the shifted space.

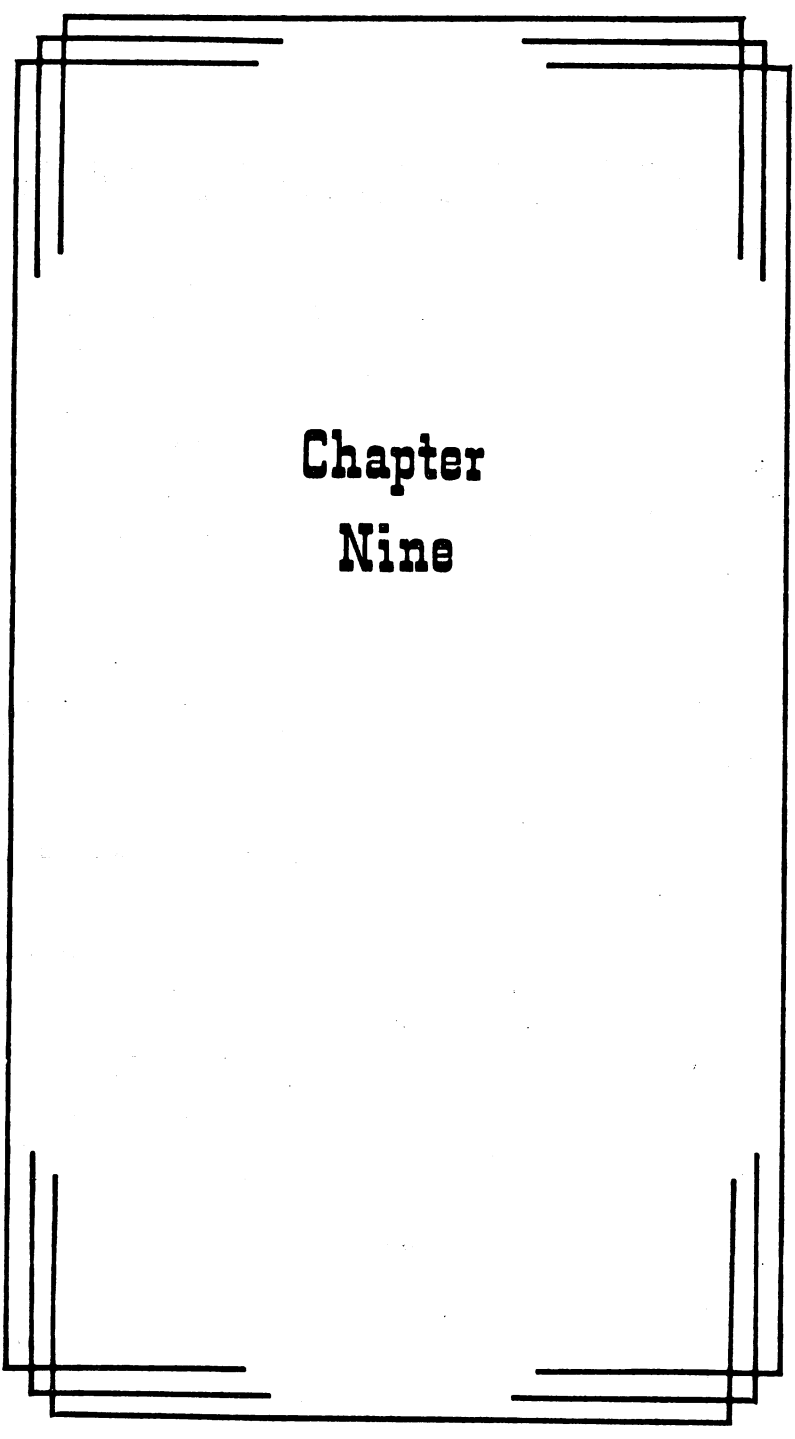
EXAMPLE: LOAD" MENU",8

This is just a beginning on the subject. Experiment, try new configurations of code. The creative mind and a disk drive are perfect partners and can be quite inventive. Pandora's box has just started to open in the field of software protection.





-----  
NOTES  
-----



**Chapter  
Nine**

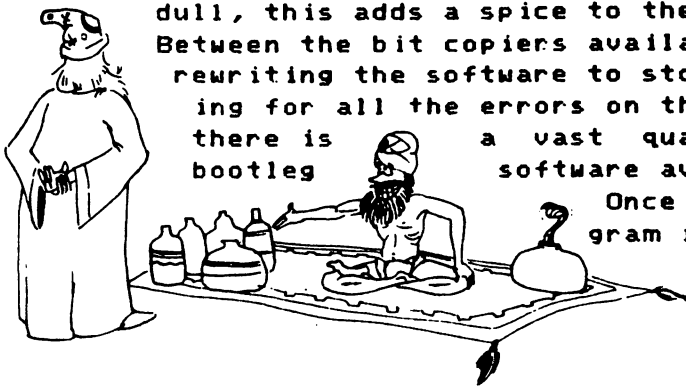
## STRANGE DISK DRIVE NOISE

Diskette errors for protection have both advantages and disadvantages. On the good side, they are cheap and easy to incorporate into your software package. As a program author, what more could you ask for, low cost fast protection. But as a user you are entitled to a back-up copy of the program you have just bought in good faith. You don't expect to have your disk drive rattle, grind, and run for ever, just because of a protection scheme used to guard against piracy.

The read errors and strange format tricks can cause your drive to virtually have a stroke every time a protected disk is used. This type of protection also slows down the load time while the drive hunts for errors and other hidden goodies lurking in every corner. The 1541 is slow enough without any futher interference.

For the pirate this type of protection presents a challenge, they seem to really enjoy the dare. Normal programming is often dull, this adds a spice to their life. Between the bit copiers available, and rewriting the software to stop checking for all the errors on the disk, there is a vast quantity of bootleg software available.

Once the program is broken it is often



better than the original. No more banging the heads of the disk drive. The files will load faster, and usually save disk space, by allowing more than one program per diskette.

As an author, you may be asking why bother with protection at all. It would begin to appear that anything can be copied or broken in to. This may be true, but the majority of users are basically honest and do respect the copyright laws. All they want is a back-up in the event the dog chews up their favorite game. The percentage of bad apples is very small.

The methods of diskette protection appears to be endless and new ones are invented everyday. The Block Allocation Map (BAM) may be modified. The directory (DIR) can be hidden from the user, or it may be modified to stop the user from even listing the directory. Errors, unformatted tracks, side sectors, modified headers, may all lurk on your new disk. There are more places to hide on a disk than I care to count.

When you load a program for the first time, listen and watch the disk drive. It can give the first signals that software protection does exist on the diskette. Is the red light flashing? Are strange sounds coming from the drive? Is it grinding and groaning as you have never heard before? Don't panic, it's only protection. This noise is generated as the disk drive attempts to read a bad block. The disk drive can not read the information in the header. When this happens the drive will re-position the read/write head. This in turn pounds the stepper motor cam against the end stop. The read/write head is attached to the cam. If a bad block is found, an error occurs and the read/write head bangs away. As panic sets in for the user, the drive repeats the

un-natural noise and the red light flashes while read/write head pounds against the end stop in an attempt to retrieve the information on the disk.

Every owner of a 1541 disk drive has heard the sad tales about the problems of alignment. Some say the reading and writing of bad blocks is a major contributor to the difficulty, others don't agree. I for one think this is a supporting reason for mis-alignment. Anything that sounds that bad, can't be good. Furthermore, if there is a possibility that a drive can be harmed, in even the slightest way, then why do these things? In all fairness to the user there are other methods of protection that are just as effective, without the chance of harming a disk drive. End of sermon.

## **What Is A Disk Error ?**

An error is a given sector or track on a disk that has been purposely corrupted by the author, to prevent an illegal copy of the program being made. The error acts as a password. When the program is run it will check the disk to see if the error is present. If the error is found, the program will execute in a normal manner. If the error is not present, the program will crash or go into an endless loop.

When attempting to remove an error, ALWAYS make a copy of the program before changing any code. This may be a file copy, as it dose not have to run, it is only for you to work with.

On the program disk that came with this book, you will find a program named "Error Checker". To load the program, type the following:

**LOAD"ERROR CHECKER",8**

When the program is RUN, it will ask if you would like to check selected tracks, or the entire disk. Insert the disk containing the program you would like to check for errors. Then press the proper function key for the desired choice. The errors will be displayed on the screen as the program runs. When it has finished, you will be ask if you would like a print-out of the results.

Each of the possible errors is described below.

---

**20 READ ERROR (HEADER BLOCK NOT FOUND)**

The disk controller is unable to locate the header of the requested block after 90 attempts. May be caused by an illegal block number, or the header has been destroyed.

---

**21 READ ERROR (NO SYNC CHARACTER)**

The disk controller could not detect a sync mark (10 or more consecutive bits) on a given track. May be caused by mis-alignment of the read/write head.

---

**22 READ ERROR (DATA BLOCK NOT PRESENT)**

The disk controller has been requested to read, or verify a data block, that when compared against a preset data block, has failed. This error message occurs in conjunction with the BLOCK commands and indicates an illegal track and/or block request.

---

**23 READ ERROR (CHECKSUM ERROR IN DATA BLOCK)**

This error message indicates that there is an error, in one or more of the data bytes. The data has been read into the DOS memory, but the checksum over the data is in error.

---

---

25 WRITE-VERIFY ERROR

The contents of the data just written to a sector, did not match the data in RAM when compared.

---

26 WRITE PROTECT ON

You have tried to write to a disk with the write protector tab in place. The tab must be removed.

---

27 READ ERROR (CHECKSUM ERROR IN HEADER BLOCK)

The controller has detected an error in the header of the requested data block. The block has not been read into the DOS memory.

---

29 READ ERROR (DISK ID MISMATCH)

The controller attempts to detect the sync mark of the next header after writing a data block. If the sync mark does not appear within a pre-determined time, the error message is generated. the error is caused by a bad diskette format ( the data extends into the next block), or by hardware failure.

---

73 DOS MISMATCH

An attempt was made to write to a diskette with a non-compatible format. This is a very common form of software protection.

---

Now you know what kind of error is on the disk. Next we have to find, where in the program, is the code that checks for the error. What follows is an example of the type of code you are looking for.

```
10 OPEN 15,8,15,"10:"
20 OPEN 5,8,5,"#"
30 PRINT#15,"B-R";5;0;33;3
50 IF A = 0 THEN 70
60 GOTO
70 SYS 64738
```

This is a generic sample that checks for any error. If line 50 were changed to read 'A = 21', then it would look for an error 21, or what ever the zero had been changed to. Some programmers substitute "U1" for the "B-R" in line 30, both will do the same thing. Once the code that checks for the error is found, it can be removed. Now the program can be moved as a file to another disk.

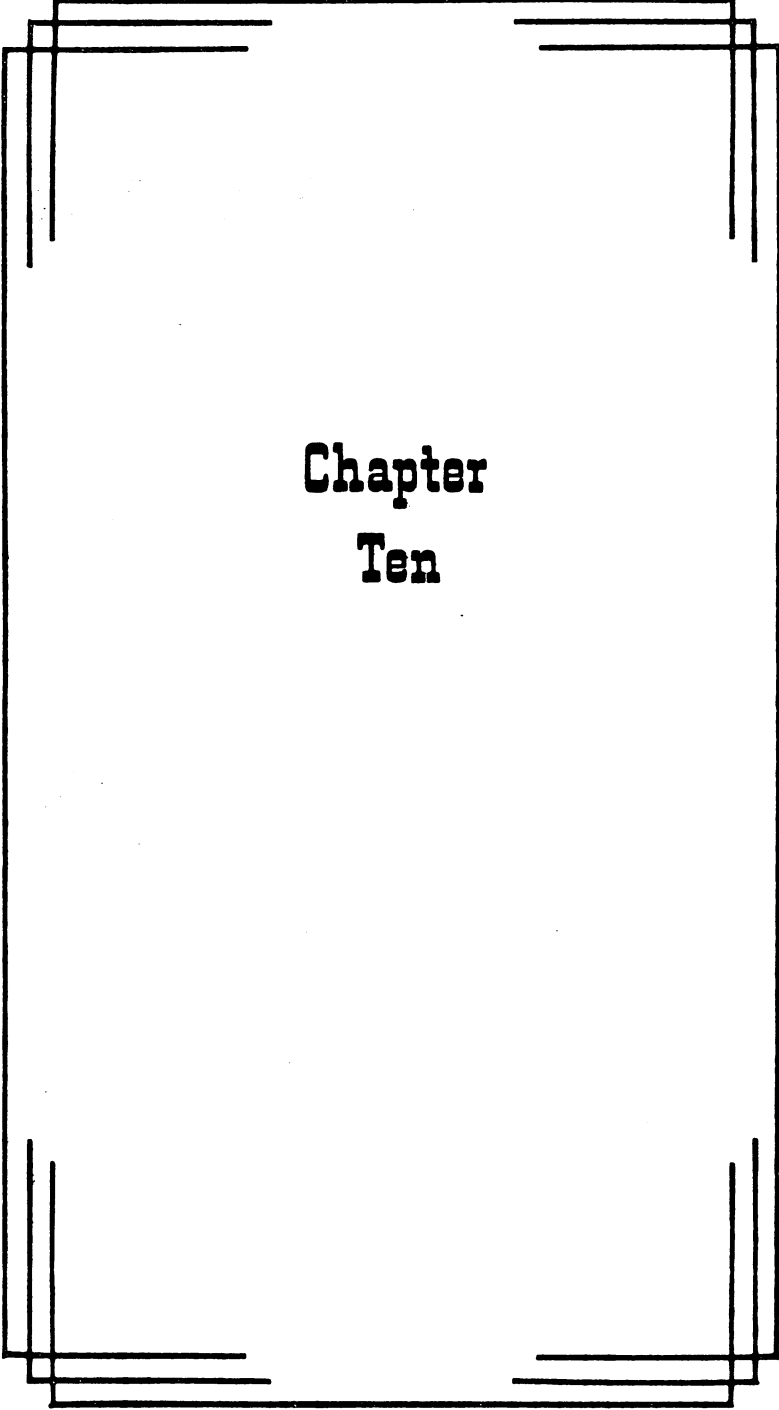
If you are working with machine language, the general outline is, the same, except ML will rely on KERNAL subroutines located at \$FFB1 to \$FFFF5. BASIC instructions and the ML routines will perform the same thing, check for an error. Here is a comparison of code.

BASIC	MACHINE LANGUAGE
OPEN	\$FFBA \$FFBD \$FFC0
PRINT#	\$FFD2 \$FFA8
"U1:5;0;1;0"	U1: 5 0 01 00
INPUT#	\$FFCF \$FFA5 \$FFE4
IF THEN	CMP BEQ
SYS 64738	JSR \$FCE2
CLOSE	\$FFC3 \$FFCC \$FFE7

The errors are what make the disk drive heads beat against the end stop. That is the noise that makes the user panic, and cry. Everytime I hear the sound I wonder, is this the time the heads are going to be knocked out of alignment? Once the error has been removed, those sounds are gone. The program can run faster, no more wasted time looking for errors. If size allows, it can be stored on the same disk with other files. Just think of all that disk space that has been made available. Authors, please think twice before using this type of protection. Be Kind to your following.



-----  
NOTES  
-----



**Chapter  
Ten**

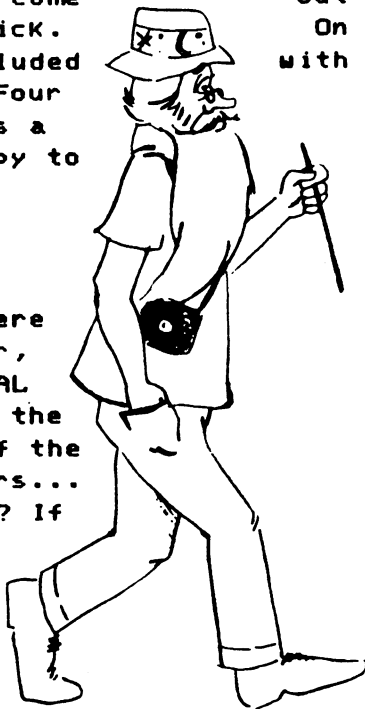
## ADVANCED PROTECTION

The longer an author programs, the more sophisticated the methods of program protection get. In this chapter we will begin to explore some of the more advanced procedures. If you have skipped the section on the use of the machine language monitor, it would be to your advantage to review it now.

For your own protection, it is advised that a back-up be made of the disk you are going to experiment with. Never play with your favorite programs, you can come out on the short end of the stick. On the Public Domain disk included with this book, is the program Four Minute Disk copier. This is a fast easy way to make a copy to work with.

### The KERNAL

The Commodore people were really thinking of the user, when they devised the KERNAL ROM. So far they have used the same ROM routines in all of the Commodore personal computers... What does this mean to you? If you were to up-grade to a larger computer of the same make, the ROM may be compatible.



The KERNAL ROM consists of 39 previously programmed subroutines. The prudent programmer makes use of these when ever possible. These routines are a jump table in ROM, that provides the link to the proper routine, no matter which Commodore computer you are using. The address of the jump stays the same, though the routine to which it jumps, may be moved to a different location. The routines were chosen by the developers, as the ones most often used by the programmer.

I know you are asking how does this relate to program protection? Read the following list, I think you will soon relate to the importance of the KERNAL ROM.

When a machine language program is going to access the disk drive, it must use KERNAL subroutines.

When one machine language program loads another, KERNAL subroutines are used.

When information is read from a user file, in machine language, KERNAL subroutines are used.

When a program tries to read an error on the disk, KERNAL subroutines are used to open the channel, tells the disk drive where to look for the error, etc.

When a program checks for a I.D. mismatch (error 29), KERNAL subroutines are used.

When a program has to access another device, KERNAL subroutines are used.

Do you begin to notice the fact, that the KERNAL subroutines control a lot of information? KERNAL ROM may be addressed from machine language or BASIC. In BASIC the SYS

command is used to address the proper subroutine. Find the KERNAL ROM routines and you have a good chance of finding the protection scheme.

Everyone has a different approach when looking for program protection. Here is the order I generally use to look for KERNAL subroutines or bad blocks.

1. Make a back-up copy to work with.
2. Run the original copy, time how long it takes for the program to load. KEEP NOTES.
3. Time how long it takes for the opening screen to appear.
4. Listen for unusual noise, head banging, etc. Try to determine in which file it occurred. Sounds are good clues as to bad blocks.
5. Watch the red light on the disk drive. The light will go off as a file is closed, and back on, when another is opened. Try to determine at what point in the this happens.
6. Print a directory of the original disk. If you can't, find out why not.

Now you have finished the first round of exploring. At this point it is generally possible to make an educated guess as to which files contains the protection. Next load the program Superman and examine the file. The 'H' command can be used to hunt for the KERNAL subroutines.

The following is a list of KERNAL ROM and what each location does.

# KERNAL ROM

HEX	DECIMAL	LABEL
\$FF81	65409	CINT
Initialize screen editor, such as, return VIC-II chip to normal after hi-res graphics.		
-----		
\$FF84	65412	IOINIT
Initialize (restore) all input/output devices to their normal condition.		
-----		
\$FF87	65415	RAMTAS
Initialize the computer when turned on.		
-----		
\$FF8A	65418	RESTOR
Restore default system and interrupt vectors.		
-----		
\$FF8D	65421	VECTOR
Read and set vectored input and output.		
-----		
\$FF90	65424	SETMSG
Controls print of error and control messages.		
-----		
\$FF93	65427	SECOND
Sends secondary address on serial bus after LISTEN routine.		
-----		
\$FF96	65430	TKSA
Send secondary address to device commanded to talk.		
-----		
\$FF99	65433	MEMTOP
Return or set address of top of memory.		
-----		
\$FF9C	65436	MEMBOT
Return and set address of bottom of mem. available		
-----		
\$FF9F	65439	SCNKEY
Scan Keyboard, returns ASCII of key pressed.		

HEX	DECIMAL	LABEL
\$FFA2	65442	SETTMO
Set IEEE bus time out flag.		
-----		
\$FFA5	65445	ACPTR
Reads a byte from the serial port.		
-----		
\$FFA8	65448	CIOUT
Output a byte to a device on serial IEEE bus.		
-----		
\$FFAB	65451	UNTLK
Stop talk on devices using IEEE serial bus.		
-----		
\$FFAE	65454	UNLSN
Stop LISTEN on devices using IEEE serial bus.		
-----		
\$FFB1	65457	LISTEN
Tell specific device on serial bus to listen		
-----		
\$FFB4	65460	TALK
Command device on IEEE serial bus to talk.		
-----		
\$FFB7	65463	READST
Returns current status of input/output device.		
-----		
\$FFBA	65466	SETLFS
Set up a logical file.		
-----		
\$FFBD	65469	SETNAM
Set up a file name.		
-----		
\$FFC0	65472	OPEN
Open a logical file.		
-----		
\$FFC3	65475	CLOSE
Close specific logical file.		
-----		
\$FFC6	65478	CHKIN
Open a channel for input, a logical file previously opened.		

HEX	DECIMAL	LABEL
\$FFC9	65481	CHKOUT
Open a channel for input, a logical file previously opened.		
-----		
\$FFCC	65484	CLRCHN
Close all input/output channels.		
-----		
\$FFCF	65487	CHRIN
Input a byte from the input channel.		
-----		
\$FFD2	65490	CHROUT
Output a byte from the channel opened for input.		
-----		
\$FFD5	65493	LOAD
Load or verify RAM from a device.		
-----		
\$FFD8	65496	SAVE
Save RAM to a device.		
-----		
\$FFDB	65499	SETTIM
Set the system clock (jiffy).		
-----		
\$FFDE	65502	ROTIM
Read system clock.		
-----		
\$FFE1	65505	STOP
Checks to see if the STOP key has been pressed		
-----		
\$FFE4	65508	GETIN
Get a character from keyboard or RS232 device.		
-----		
\$FFE7	65511	CLALL
Close all files channels and files now open.		
-----		
\$FFEA	65514	UDTIM
Update the system clock.		



HEX	DECIMAL	LABEL
\$FFED	65517	SCREEN
Returns format of the screen, columns & rows.		
-----		
\$FFF0	65520	PLOT
Get current, or set specified cursor position.		
-----		
\$FFF3	65523	IOBASE
Returns the address of the 6526 CIA.		

The more you study the KERNAL ROM, the more information you will find. Now on with looking for protection in your program. One of the strange facts about computers, is that they can read ASCII backwards as well as forwards. This is done by a few programmers to add some confusion. Be on the look out for this tid-bit.

## Super Line Numbers

Since the invention of the Commodore 64, we have been told that line numbers MUST fall into the range of 0 to 63999. The computer will not accept a larger number. WRONG, WRONG, the computer can be fooled. The secret to the trick is, the C-64 can not write or edit line numbers larger than 63999, but it can read them. With a little magic, it will accept line numbers up to 65535.

To test this tom-foolery yourself, enter the following program:

```
10 PRINT"[shifted clear-home]"
20 PRINT"HI MY IS THE C-64"
30 PRINT"WHAT IS YOUR NAME"
40 INPUTN$
50 PRINT
60 PRINT"HI ":N$
```

Save the test program to disk, as "SUPER LINES". Next load the program "SUPERMON.C" from the public domain disk that came with this book. Run the program. All numbers will be in hexadecimal. The following message will appear on the screen as the program completes loading:

SUPER 64-MON

JIM BUTTERFIELD

B\*

PC SR AC XR YR SP  
.:97FE 31 40 E6 00 F6

Enter: F 0801 1000 FF

*2049* This tells Supermon to fill memory from address 0801 to 1000 with the letters FF. This will make it easier for you to locate your program on the screen. Your file ends where the letters FF begin. *4096* *255*

Next enter: L "SUPER LINES",08

This command will load the program Super Lines into memory. As the loading takes place, the following message will appear on the screen:

SEARCHING FOR SUPER LINES  
LOADING

When a period appears on the screen, your program has completed loading.

Next enter: M 0800 0860 *2648* *2144*

Your program will now appear on the screen in hexadecimal. The file "SUPER LINES" ends where the letters FF begin.

<-- Protection Revealed -->

$LOC = \$0801 = 2,049$   
 0 1 2 3 4 5 6 7  
 .:0800 00 0A 08 0A 00 99 22 93  
 .:0808 22 00 28 08 14 00 99 22  
 .:0810 48 49 20 4D 59 20 4E 41  
 .:0818 4D 45 20 49 53 20 54 48  
 .:0820 45 20 43 2D 36 34 22 00  
 .:0828 41 08 1E 00 99 22 57 48  
 .:0830 41 54 20 49 53 20 59 4F  
 .:0838 55 52 20 4E 41 4D 45 22  
 .:0840 00 49 08 28 00 85 4E 24  
 .:0848 00 4F 08 32 00 99 00 5D  
 .:0850 08 3C 00 99 22 48 49 20  
 .:0858 22 3B 4E 24 00 00 00 FF  
 .:0860 FF FF FF FF FF FF FF FF

$LOC = \$085F = 2,143$

The first line contains information as to where the program resides in memory, the first line number in the program, and pointers to the second line number. Numbers are always read from left to right, in hexadecimal. In this example, the first byte is 800 (where the file resides in memory). Numbering progresses in this manner to the end of the file.

EXAMPLE:

800 801 802 803 804 805 806 807

0800 00 0A 08 0A 00 99 22 93

2048

0800 = the address in memory where the program resides. This is the start of BASIC.

800 = 00...This byte is always 0, the actual program starts in the next byte.

801 = 0A...low order byte, pointer to the address of the next line number in the file.

802 = 08...high order byte, pointer to the address of the next line number in the file.

803 = 0A...low order byte of the first line  
 10 POINTER = \$080A = 2,058

number in the file.

804 = 00...high order byte of the first line number in the file. 1ST LINE LOC = #000A = 10

805 = 99...<sup>153</sup>the start of the actual code.

To change line number 10 (this is <sup>255</sup> the first line number in the file), move the cursor to location 803. Type FF, move to location 804, type FF. Press return. Move the cursor down to the bottom of the screen, press return.

Enter: S "SUPER LINES 1",08,0801,085F <sup>2049</sup> <sup>2143</sup>

Line number <sup>#FFFF</sup>10 has now been changed to line number 65535. You can change all the line numbers in the file to the same or an entry other than FFFF.

To change line number 20, check locations 801 and 802 to find the location of the line in memory. Remember these are the pointers to the next line. The actual line number follows after the pointers. In this case the locations are 80C and 80D. Repeat the above process for as many lines as you wish to change.

Once line numbers have been changed in this manner, they may not be edited. All GOTO's, GOSUB's, and THEN commands, must go to a valid line number, not one of the super numbers.

<-- Protection Revealed -->

-----  
NOTES  
-----





**Chapter**

**Eleven**

## THE NEWCOMERS.....

The newer protection methods have produced a by-product, another round of computer buzz-words. Advertisements in magazines are abundant with pseudology to make their products sell. I find reading them an entertaining education. This latest evolution of copy protection techniques are like a window into the future. Can the programs really make an archival copy of the latest and the greatest?



The manufacture's claim their programs can copy just about any program on the market. If you read the advertisements for their software, one would think all they need do, is buy one. Making an archival back-up would never be a problem again. Just read a few of the following statements: new technological breakthrough, the ultimate bit by bit disk duplicator, copies identical syncs, no need to worry about extra sectors, half track nibbler. It all sounds great. What could possibly stop these marvels from making a copy of a protected program?

The first tip that copiers can not reproduce everything, was right in one of the

advertisements, "copies everything but it's self". Great, the program has better protection than the copier can handle. Another author claims "copies 99% of programs on the market". Strange how many of that one percent I find. I own and use twelve different copy programs, just to make archival back-ups. Each one copies only a few of the programs on the current market. To top it off, every four to five months the copy programs are obsolete. One of the popular companies has a cure for the situation. A registered owner may update at any time for only half price. How nice of them, it's a \$39.95 program, the half is still \$20.00.

Another of the popular copy programs, is sold in a rather deceptive manner. The buyer receives a disk containing an assortment of programs. They do not receive an instruction booklet. Now what good is the program, if you are not told how to operate it. To top off the situation, if you want to learn how to use about half of the programs on the disk, you must subscribe to a newsletter for an additional \$19.00.

To the authors of the world, take heed. Deceptive advertising, un-heard of buzz words, lack of support, plus high prices, all encourage the making of illegal copies of your programs.

The mystic surrounding software protection continues to increase as more programs appear on the market. Nothing adds to the situation more than the use of vague buzz-words. The rest of this chapter, is devoted to some of the new crop, now being used in books and advertisements. It's time that the few in the know, reveal to the user what they are saying. The real question in my mind is, just how practical are these methods for the average



user? I even wonder if the 1541 is capable of doing some of the wonderous new tricks.

## Non-standard Sectors

The sectors are normally written in numeric order from 0 to the maximum for the given track. Sectors written or duplicated in any other order, are considered non-standard. The program then checks for the non-standard sectors, if not in the order written, crash.

## Extra Sector

An extra sector can be added to tracks 18 thru 24. The following example illustrates the normal size of a sector.

	BITS
HEADER SYNC	40
HEADER BLOCK	80
HEADER GAP	64
DATA SYNC	40
DATA BLOCK	2600
TAIL GAP	64
<hr/>	
TOTAL	2888 BITS

8 BITS = 1 BYTE

2888/8 = 361 BYTES PER SECTOR

STANDARD NUMBER OF BYTES PER TRACK	7142
- 361 BYTES/SECTOR X 19 SECTORS	- 6859
<hr/>	
= EXTRA BYTES PER TRACK	283

The extra bytes are divided among the tail gaps of the sectors. As an entire sector would be 361 bytes, this leaves us a little short for an complete additional sector. There are two possible ways to gain the extra sector,

gather bytes from the tail gaps or reduce the speed of the drive.

To gain the use of the sectors in the tail gaps, one would have to burn a new ROM chip. The new ROM would instruct the drive to format 20 sectors instead of the normal 19, on tracks 18 thru 24 (your choice as to which track). It would also divide the total bytes among the proper number of sectors, thus the extra sector has been created.

The other alternate is the reduction of the drive speed for the purpose of formatting the disk. Data is written at the normal speed, but the diskette is now moving slower. More bytes will be placed over less space. When the disk is written to, at normal speed, bytes are closer together. Tricky isn't it.

The above are very light over-views of the procedures to gain an extra sector. If you happen to have one of the older Commodore drives (2040 or 4040), their standard format was 20 sectors on tracks 18 thru 24, and are read compatible with the 1541. If you ever have the luck of finding a 4040, grab it quick, they have been know as 'the old reliables'.

## **GAP Bytes**

The normal formatted track contains eight gap bytes, these separate the header from the data block.

## **Density Changes**

The bit pattern at location \$1C00 in the 1541, controls the density, motor on-off, signal light on/off, write protect sensor, and cycling the read head in half-track increments.

DISK CONTROLLER 6522

\$1C00

---

Bit 0 - \$01	Bits 0 & 1 are cycled
Bit 1 - \$02	to step the head
Bit 2 - \$04	Motor on (1) or off (0)
Bit 3 - \$08	Drive active LED on/off
Bit 4 - \$10	Write protect sense
Bit 5 - \$20	Density select (0)
Bit 6 - \$40	Density select (1)
Bit 7 - \$80	SYNC detect line

---

By changing bits 5 and 6, the density may be changed to a non-standard density. Lowering the density has proven successful. At higher density than normal, the bits can bleed together and make the code garbage. Several different densities can be used on the same track. A copy program will read the bits at normal density and write garbage to the new disk.

## Guard Band

The guard band erases a blank area on either side of a track. The purpose of this blank area is to buffer information from one track to another. This buffer prevents information from bleeding to an adjacent track.

## Half Tracks

The stepper motor of the 1541 can stop, read and write between standard tracks. This is known as half-tracking or side sectoring. This form of protection is not copied by all copy programs. It has one draw-back, if you write on a half-track, it wipes out the tracks on either side. The same happens to the half-track when you write on the whole track beside it. Just remember, one or the other, but not both.

## **Nibble Back-up**

The Commodore 1541 reads whole bytes, while a nibble is half a byte, that is, if you want to be technically correct. There are many copy programs on the market that claim to be nibble copiers. Nibble is a great sounding buzz-word, plus they seem to work, so who cares.

## **Nibble Counting**

This type of protection counts the number of nibbles (bytes) between two given points. The results is then compared to a pre-determined value. If the figures do not match, the program will crash.

## **Tracks 35 to 40**

It is possible to read and write information on the 1541 to track 40. It may not be reliable past track 37, because of bleed over. The tracks from 35 to 37 are a good place for protection check-sums to be placed. Many of the current copiers do not go out that far. The 1541 seems to have another problem if forced past track 37. The head can become physically stuck. This can only be corrected by taking the drive apart, not everyone wants to be in this situation.

I hope this small selection of modern protection schemes, has wet your appetite to explore the 1541 at length. A complete listing of the RAM is in Appendix C. It is an interesting study. The 1541 is a powerful piece of hardware, and can become a toy in the right hands. As to the program protection of the future, who knows what direction it will take. All that is certain, is that the disk drive will be the key.

-----  
**NOTES**  
-----

**Appendix**

**A**

# PUBLIC DOMAIN SOFTWARE

The following collection of utility programs are all in the Public Domain. They have been included to aid you in the understanding of software protection. Feel free to share copies with your friends. The last program (New Operating System) is just in fun, so don't take it too seriously. I hope you find these as useful as we do.

---

## MENU

---

LOAD "MENU",8

This menu program has been added to make the operation of the disk user friendly. First load the file by the above command, or LOAD "\*",8. Press return when the ready prompt appears. Make your choice from the selection on the screen and press return. The program will be loaded and automatically run. If you do not wish to use this feature, load, following the instructions in the individual descriptions below.

If you would like to use the menu on other disks you might have, here is how to change it. The program is currently set up for a maximum of 24 files. You may change the titles by altering the names between quotes in the DATA statements, lines 118 thru 123. The last entry must be the word "END".

---

---

EDIT TRACK & SECTOR

---

LOAD"EDIT T & S",8

Load and run the program. When prompted, insert the disk you wish to look at, in the drive. Then press return. The following menu will be displayed on the monitor:

F1 = ASCII DUMP  
F3 = HEX DUMP  
F5 = NEXT SECTOR  
F7 = LAST SECTOR

F2 = CHANGE BYTE  
F4 = WRITE SECTOR  
F6 = HARDCOPY  
F8 = NEW SECTOR

↑ = EXIT PROGRAM  
H = HELP SCREEN

F1 = ASCII DUMP...this command will display the Commodore ASCII of the desired track and sector.

F3 = HEX DUMP...this command will display the desired track and sector in hexadecimal.

F5 = NEXT SECTOR....will display the next linked sector. When sectors are written by the 1541 disk drive, they are not placed in numeric order. This program finds the correct order. If the sector you are viewing is the last one, it will be repeated.

F7 = LAST SECTOR....allows the user to back up one linked sector.

F2 = CHANGE BYTE...in the lower left-hand corner of the screen, you will be ask the byte to change. Enter the byte number to be changed



in decimal. Answer the next prompt in hexadecimal. Press return, the change will then appear on the screen. The prompts will re-appear so that you may make more changes if you like. ↑ to exit at any time.

F4 = WRITE SECTOR...exit CHANGE BYTE (press ↑) to the main screen. Press F4, then press return to continue. 'ONE MOMENT' will appear on the screen while the write command is being executed.

F6 = HARDCOPY...press F6, the prompt will read, 'PRINTER ON (Y/N)'. Turn on your printer, if you have not already done so. Next you will be ask the format you wish, (A)SCII, (H)EX, or (B)OTH. Make your selection and the printer will begin to print. Press ↑ to exit to main screen.

F8 = NEW SECTOR...press F8, enter the track and sector, then press return.

↑ = EXIT PROGRAM...at the main screen, this command will exit the program to BASIC. At all other times the command ↑ will return you to the main screen.

H = HELP SCREEN...press H for the main menu.

---

## VIEW BAM

---

LOAD"VIEW BAM",8

Load and run the program 'View Bam'. Insert the disk you wish to view, in the disk drive. Then press return. The yellow dots are sectors that are not in use. The large blue dots are sectors that have been filled with data. Another disk may be done by answering the prompt with 'Y', or 'N' to exit to BASIC.

---

NEW DISK TITLE

---

LOAD"NEW DISK TITLE",8

This program will change the name of a disk. The old title is shown on the screen. The user is ask if they wish to change the name. If the answer is yes, enter the new name when prompted. Press RETURN, and the computer will complete the task.

WARNING: remove the write-protector in order that the disk drive may write the new name to the disk.

---

CHANGE DISK ID

---

LOAD"CHANGE DISK ID",8

This program will change the ID of a disk. The old ID is shown on the screen and the user is ask if they wish to change the ID. If the answer is yes, enter the new ID when prompted. Press RETURN, the disk drive will complete the job.

WARNING: only the ID shown in the DIRECTORY is changed, not on each sector. Remove the write-protector in order that the disk drive may write the new ID to the disk.

---

TRACK ID READER

---

LOAD"TRACK ID READER",8

This program will read the ID of selected tracks. Enter the number of the track you wish. Press return. Press any key to read another track ID.

---

TRACE FILE LINKS

---

LOAD"TRACE FILE LINKS",8

The program 'Trace File Links' traces the order that files have been saved to the disk. Enter the starting track, then press return. Enter the starting sector, then press return. The display will now appear on the monitor.

---

DISK DOCTOR PLUS

---

LOAD"DISK DOCTOR PLUS",8

Disk Doctor Plus is a track and sector editor. The program may be used to examine and modify sectors of code on the disk. It will display all code in decimal. Load and review Disk Doctor Instructions for an understanding of commands and how to use the program.

---

DISK DOCTOR INSTRUCTIONS

---

LOAD"DR. INSTRUCTIONS",8

This file contains the instructions and commands to Disk Doctor Plus. Load and review before using the above program.

---

HEX DUMP FILE

---

LOAD"HEX DUMP FILE",8

The program 'Hex Dump File' will display a selected file to the screen in hexadecimal. Press the CTRL key to slow down the display of code.

---

## HEX/DEC/BIN CONVERSION

---

LOAD "HEX/DEC/BIN",8

The program is menu driven, the following choices are available:

1. Hexadecimal to Decimal
2. Decimal to Hexadecimal
3. Hexadecimal to Binary
4. Binary to Hexidecimal
5. Decimal to Binary
6. Binary to Decimal
- \*. Exit Program

Enter the correct selection, then press return. Next enter the number you wish converted.

A very fast, easy to use program. This is a true work horse.

---

## SUPERMON

---

LOAD "SUPERMON.C",8

Supermon is a machine language monitor. The program may be used examine, modify, or create machine language in memory. Load and review Supermon Instructions for an understanding of how to use the program.

---

## SUPERMON INSTRUCTIONS

---

LOAD "SUPERMON INST.C",8

This file contains the instructions and commands to Supermon. It is suggested that the user review this program before loading Supermon.

---

**DISK LOGGER**

---

**LOAD"DISC LOGGER",8**

Disk Logger gives the user a directory of a disk that includes the following information for each file: beginning track and sector, block size, name, starting and ending address. The user is ask if they wish a print-out or screen viewing. If the answer is yes enter the date when prompted. The print out will include the above plus date, disk name, and disk ID.

---

**CROSS REFERENCE 64**

---

**LOAD"CROSS-REF64",8**

Cross reference identifies all variables that have been used in a file. Very handy when writing a program or to modify existing code. May be viewed on the screen or print a hard copy.

Enter the name of the program when prompted, press return. Choose either printer or screen for output, then press return.

---

**LOCK/UNLOCK DISK**

---

**LOAD"LOCK/UNLOCK DISK",8**

The following menu will appear on the screen when the program is run:

F1 = LOCK A DISK  
F3 = UNLOCK A DISK  
F5 = EXIT PROGRAM

The program allows the author to prevent a user from writing to the disk. Select from the above menu, then press return.

---

## AUTOMATIC BOOT MAKER

---

LOAD"BOOT MAKER",8

Boot maker will create an auto-load-run boot for your program. Load and run Automatic Boot Maker. Insert the disk with the program for which you wish to make a boot, into the disk drive. Enter the name of the boot when prompted, press return. Enter the name of the main program, press return. The program will now construct the desired boot and write it to the disk.

To load the new boot type:

LOAD "BOOT NAME",8,1.

WARNING: remove the write protector tab from the object disk, so that the new boot may be written.

---

## LOAD IT ANYPLACE

---

LOAD"LOAD ANYPLACE",8

This program will re-locate a program as it is saved, to a new location in memory. Load and run the program. Enter the name of the file to be re-located, press return. Enter the low order byte (hexadecimal) of the new location, press return. Enter the high order byte (hexadecimal) of the new location, press return.

EXAMPLE:

\$0800....08 = HIGH ORDER BYTE  
00 = LOW ORDER BYTE

---

---

SEQUENTIAL FILE PRINTER & READER

---

LOAD"SEQ READER",8

The SEQ Reader allows the user to load a sequential file into memory and read or print the contents. Load and run the program. The following menu will appear:

OPTIONS

F1 = DIRECTORY  
F3 = SCREEN ONLY  
F5 = HARD COPY  
F7 = KILL DISK FILE  
F8 = EXIT PROGRAM

F1 = DIRECTORY...a sub-menu will appear, the choices are: F1 = Screen only, F3 = Hardcopy, and F5 = Cancel Function. enter your choice. F5 will return you to the main menu.

F3 = SCREEN ONLY...enter the name of the file to view, then press return. Press any key to pause operation. If Pause, a sub-menu will appear as follows: F1 = Printer ON, F3 = Printer OFF, F5 = Abort Operation, F7 = Continue Unchanged.

F5 = HARD COPY...enter the name of the file to print, then press return. Press any key to pause the operation. If Pause, a sub-menu will appear as follows: F1 = Printer ON, F3 = Printer OFF, F5 = Abort Operation, F7 = Continue Unchanged.

F7 = KILL DISK FILE...enter the name of the file to be scratched, then press return. You will be ask 'DO YOU REALLY WISH TO KILL, THE XXXX FILE? (Y/N)'. Input 'Y' or 'N'. If 'N', you will be returned to the main menu.

F8 = EXIT PROGRAM...exit to BASIC

---

ERROR CHECKER

---

LOAD"ERROR CHECKER",8

Error Check may be used to examine selected tracks or an entire disk for errors. If an error is found the error channel will be read. The track, sector, and type of error will displayed on the screen. A hardcopy may also be printed.

Load and run the program. The following menu will appear on the screen:

F1 = FOR SELECTED TRACKS  
F3 = FOR COMPLETE DISK  
F5 = EXIT PROGRAM

F1 = FOR SELECTED TRACKS....Place the disk you wish to error check in the disk drive. Enter the desired track when prompted, press return. Enter the last track you wish to examine, press return. When complete, you will be ask if you would like a hardcopy. If you answer no to the question, you will be returned to the main menu.

F3 = FOR COMPLETE DISK....Place the disk you wish to error check in the disk drive. When you press F3, the error check will begin. When complete, you will be ask if you would like a hardcopy. If you answer no to the question, you will be returned to the main menu.

F5 = EXIT PROGRAM....This will exit you from the program and return you to BASIC mode.

---



---

## TEN SECOND FORMAT

---

LOAD"FFORMAT",8

Ten Second Format will format your disk in ten seconds. When the program has been run answer the prompts on the screen and press return. After the format is complete and the READY prompt is shown, type run if you wish to format another disk.

**WARNING:** This program will erase all data on the disk as it formats.

---

## FOUR MINUTE DISK COPIER

---

LOAD"4MINUT-V2",8,1

This program will copy an entire disk in approximately four minutes. Follow the prompts on the monitor.

**WARNING:** This program will first format the new disk, all data will be erased.

---

## NEW OPERATING SYSTEM

---

LOAD"NEW OP II",8

I thought you might like a good laugh after so much hard work. LOAD and RUN the New Operating System, nothing will appear different, next type a few of the normal BASIC commands such as LOAD or SAVE. Just what you needed a lippy computer, with answers to all your input. Next time one of your computer friends come over, try it out on them for a good laugh.

# Appendix

## B

## GLOSSARY

---

**ADDRESS....**A character or group of characters, that identifies a particular location in memory.

---

**ARCHIVAL COPY....**A legal copy of a file or complete disk, made by the owner of such, for the purpose of safe keeping, in the event the original is damaged or lost.

---

**ASSEMBLER....**A computer program with the capability to translate assembly language into machine language. ML is the native language of the computer.

---

**AUTO-BOOT....**A program, that upon loading into memory, loads another program, and runs that program automatically. Often this type of program is written in ML.

---

**BACK-UP....**To make a duplicate of a program or complete disk for the purpose of replacement, in the event of loss of the original.

---

**BAM....**Acronym for Block Allocation Map. This tells the user how many blocks of memory have been used, and the number remaining free for use.

---

**BASIC....**Acronym for Beginner's All-purpose Symbolic Instruction Code, pronounce as one word. The language most often used to program the Commodore 64.

---

---

**BINARY....**The native number system of the computer. The binary number system is a base 2 and contains only the numbers 0 and 1.

---

**BIT....**Acronym for Binary Digit (pronounce as one word). The smallest element of a binary number, eight bits equal one byte.

---

**BLOCK....**A group (bytes, words, or files) handled as a unit.

---

**BOOT....**A short routine commonly machine language, used as a loader for the system or a program.

---

**BYTE....**A byte consists of eight bits, which as a group represent one character.

---

**CHECKSUM....**To tabulate a group of digits, and check the results against a previously computed sum for accuracy.

---

**CHIP....**A slang or buzz word commonly used for intergrated circuit.

---

**COMPILER....**A program that acts as a translator of computer languages to machine language, for which the computer was designed.

---

**COPYRIGHT....**A legal term for a method of software protection, which is applied to prevent unauthorized copies being made. Illegal copies are punishable by law.

---

**CRASH....**To stop, malfunction, or lock-up a computer system. The cause may be hardware, software, or operator related.

---

**DIRECTORY....**A listing of the file names, block size, and types, found on a given disk or tape, also known as a catalog.

---

---

**D.O.S....**Abbreviation for Disk Operating System (pronounce as one word). A ROM chip within the disk drive that is pre-programmed with controls for the system.

---

**EEPROM....**Acronym for Electrically Erasable Programmable Read Only Memory. A computer chip that may be custom programmed. The chip retains memory when the power has been turned off. It may be erased electrically by an Eprom programmer and re-used.

---

**ENCRYPTION....**To substitute one unit of information for another according to a set rule. The units may be alpha or numeric.

---

**EPROM....**Acronym for Erasable Programmable Read Only Memory. A computer chip that may be custom programmed. The chip retains memory when the power has been turned off. It may be erased by exposing it to ultra-violet light and re-used.

---

**FILE....**To record information for a particular purpose and recognize as a unit. For computers the data file (program) is then stored to disk or tape, etc.. The primary types are: PRG, SEQ, REL, USR.

---

**FORMAT....**The manner in which a diskette is set up for use. When a disk is formatted, it is erased, named and given a new identification number.

---

**HEADER....**The header is the part of a sector that contains the disk I.D., checksum, sync marks, plus any other information needed by the disk drive.

---

**HEX....**Acronym for hexadecimal, a numeric system based on the number 16.

---

---

I/O....Acronym for Input and Output. Refers to communication between computer chips or peripherals. Example: one computer talking to another via a modem.

---

JUNK....A slang terminology commonly used to describe garbage code or otherwise unintelligible data.

---

K....Acronym for the word Kilo. In computer use, a formula of measurement of byte memory. A 1K memory storage has 1024 bytes; a 64K memory has 65,536 bytes.

---

MACHINE LANGUAGE....A binary language that is the final language all computers use. All other languages must be translated into binary code before the computer may execute them.

---

MEMORY....A means of holding information (programs and data) for future use. Memory may be in the computer itself or a device such as a diskette or cassette tape.

---

PUBLIC DOMAIN SOFTWARE....Software that is not copyrighted and is intended to be copied and exchanged without restraint.

---

RAM....Acronym for RANDOM Access Memory, pronounce as one word. This memory may be accessed and written to by the user.

---

RESET....A hardware interrupt of the computer operation.

---

ROM...Acronym for Read Only Memory, pronounce as one word. ROM within the computer itself is pre-programmed memory which holds the operating systems such as; BASIC language commands, math formulas, write to disk or tape, etc.

---

---

SECTOR....See track.

---

SMART PERIPHERAL....A peripheral containing a microprocessor within itself, such as the Commodore 1541 disk drive. The 1541 disk performs many of its functions from its own memory, thus not using bytes from the computer.

---

STORAGE....To place data in memory or save by means of tape, disk, etc., for later use.

---

TRACK....Refers to the manner in which data is organized on a diskette. The diskette is divided into tracks and each track is again divided into smaller sectors. Each track may be addressed directly and therefore may be quickly accessed.

---

UTILITY....Programs or hardware that perform system functions such as; copy files, listings, trace, debug, etc. Often referred to as tool kits.

---

WRITE-PROTECT....A method of preventing writing over stored data, such as the placement of a protective tab over the notch in the side of a diskette.

---

**Appendix  
C**



# 1541 RAM

HEX	LABEL	FUNCTION
\$0000	JOB5	Job Que: Buffer 0
\$0001		Buffer 1
\$0002		Buffer 2
\$0003		Buffer 3
\$0004		Buffer 4
\$0005		Buffer 5
\$0006	HDR5	Job headers: Buffer 0-low
\$0007		Buffer 0-high
\$0008		Buffer 1-low
\$0009		Buffer 1-high
\$000A		Buffer 2-low
\$000B		Buffer 2-high
\$000C		Buffer 3-low
\$000D		Buffer 3-high
\$000E		Buffer 4-low
\$000F		Buffer 4-high
\$0010		Buffer 5-low
\$0011		Buffer 5-high
\$0012	DSK ID	Master copy disk ID: Drive 0
\$0013		Drive 0
\$0014		UN-USED drive 1
\$0015		UN-USED drive 1
\$0016	HEADER	Image last header: ID byte 1
\$0017		ID byte 2
\$0018		Track
\$0019		Sector
\$001A		Checksum
\$001B	ACTJOB	Controllers active job
\$001C	WPSW	WP change flag: Drive 0
\$001D		UN-USED Drive 1

HEX	LABEL	FUNCTION
\$001E	LWPT	Last state of WP switch: Drive 0
\$001F		UN-USED Drive 1
\$0020	DRVST	Drive current status: Drive 0
\$0021		UN-USED
\$0022	DRVTRK	Drive track number: Drive 0
\$0023		UN-USED Drive 1
\$0024	STAB	Storage table GCR conversion
\$0025		Continued: storage table GCR
\$0026		Continued: storage table GCR
\$0027		Continued: storage table GCR
\$0028		Continued: storage table GCR
\$0029		Continued: storage table GCR
\$002A		Continued: storage table GCR
\$002B		Continued: storage table GCR
\$002C		Continued: storage table GCR
\$002D		Continued: storage table GCR
\$002E	SAVPNT	Temporary save pointer location
\$002F		
\$0030	BUFPNT	Pointer to active buffer
\$0031		Cont: pointer to active buffer.
\$0032	HDRPNT	Header pointer: track
\$0033		Header pointer: sector
\$0034	GCRPNT	GCR pointer
\$0035	GCRERR	Indicates GCR decode error
\$0036	BYTCNT	Byte counter GCR/binary conver.
\$0037	BITCNT	Bit counter
\$0038	BID	Data block ID character
\$0039	HBID	Header block ID character
\$003A	CHKSUM	Data or header checksum
\$003B	HINIB	UN-USED
\$003C	BYTE	UN-USED
\$003D	DRIVE	Drive number, \$00=1541
\$003E	CDRIVE	Current active drive number
\$003F	JOB#	Current job number
\$0040	TRACC	Track-internal storage location
\$0041	NXTJOB	Next job
\$0042	NXTRK	Next track to move head to
\$0043	SECTR	Sectors per track for formatting
\$0044	WORK	Temporary work storage location

HEX	LABEL	FUNCTION
\$0045	JOB	Temporary storage of job type
\$0046	CTRACK	UN-USED
\$0047	DBID	Data block ID
\$0048	ACLTIM	Timer for acceleration of head
\$0049	SAVSP	Temporary save of stack pointer
\$004A	STEPS	# steps to move head to track
\$004B	TMP	Temporary storage location
\$004C	CSECT	Current sector
\$004D	NEXTS	Next sector
\$004E	NXTBF	Pointer to next GCR source buffer
\$004F	NXTPNT	Ptr/next byte location in buffer
\$0050	GCRFLG	GCR/binary flag in active buffer
\$0051	FTNUM	Current format flag
\$0052	BTAB	Binary table:GCR/binary work area
\$0053		Continued: binary table
\$0054		Continued: binary table
\$0055		Continued: binary table
\$0056	GTAB	GCR table: GCR/binary work area
\$0057		Continued: GCR table
\$0058		Continued: GCR table
\$0059		Continued: GCR table
\$005A		Continued: GCR table
\$005B		Continued: GCR table
\$005C		Continued: GCR table
\$005D		Continued: GCR table
\$005E	AS	# of steps to used to accel. head
\$005F	AF	Acceleration factor
\$0060	ACLSTP	Steps to go before complete
\$0061	RSTEPS	Number of run steps left
\$0062	NXTST	Pointer to stepping rtn-\$FA05
\$0063		Continued from \$0062
\$0064	MINSTP	Minimum steps required to accel.
\$0065	VNMI	Indirect for NMI-\$EB22
\$0066		
\$0067	NMIFLG	NMI in progress flag
\$0068	AUTOFG	Auto drive initialization flag
\$0069	SECCNT	Sector increment for sequential
\$006A	RECCNT	Error recovery counter
\$006B	USRJMP	User jump table pointer-\$FFEA

HEX	LABEL	FUNCTION
\$006C		Continued from \$006B
\$006D	BMPNT	Bit map pointer
\$006E		Continued from \$006D
\$006F	T0	Temporary work space
\$0070	T1	Temporary work space
\$0071	T2	Temporary work space
\$0072	T3	Temporary work space
\$0073	T4	Temporary work space
\$0074		Temporary work space
\$0075	IP	Indirect pointer variable
\$0076		
\$0077	LSNADR	Listen address: device # + \$20
\$0078	TLKADR	Talker address: device # + \$40
\$0079	LSNACT	Active listener flag
\$007A	TLKACT	Active talker flag
\$007B	ADRSED	Addressed flag
\$007C	ATNPND	Attention pending flag
\$007D	ATNMOD	6502 in attention mode
\$007E	PRGTRK	Last program accessed
\$007F	DRVNUM	Current drive number
\$0080	TRACK	Current track number
\$0081	SECTOR	Current sector number
\$0082	LINDX	Logical index
\$0083	SA	Current secondary address
\$0084	ORGSA	Original secondary address
\$0085	DATA	Temporary data byte
\$0086	R0	Temporary work area
\$0087	R1	Temporary work area
\$0088	R2	Temporary work area
\$0089	R3	Temporary work area
\$008A	R4	Temporary work area
\$008B	RESULT	Result of multiply/divide rtns
\$008C		Result location
\$008D		Result location
\$008E		Result location
\$008F	ACCUM	Remainder of multiply/divide rtns
\$0090		Accumulator
\$0091		Accumulator
\$0092		Accumulator

HEX	LABEL	FUNCTION
\$0093		Accumulator
\$0094	DIRBUF	Pointer to directory buffer
\$0095		Continued from \$0094
\$0096	ICMD	IEEE command in:un-used
\$0097	MYPA	MY PA flag: UN-USED
\$0098	CONT	Serial bit counter
\$0099	BUFTAB	Buffer byte pointer: buffer 0 low
\$009A		:buffer 0 high
\$009B		: buffer 1 low
\$009C		:buffer 0 high
\$009D		: buffer 2 low
\$009E		:buffer 0 high
\$009F		: buffer 3 low
\$00A0		:buffer 0 high
\$00A1		: buffer 4 low
\$00A2		:buffer 0 high
\$00A3		: CMD buffer low
\$00A4		: CMD buffer high
\$00A5		: error buffer low
\$00A6		:error buffer high
\$00A7	BUFO	Inactive flags for buffers
\$00A8		Cont.: inactive flags for buffers
\$00A9		Cont.: inactive flags for buffers
\$00AA		Cont.: inactive flags for buffers
\$00AB		Cont.: inactive flags for buffers
\$00AC		Cont.: inactive flags for buffers
\$00AD		Cont.: inactive flags for buffers
\$00AE	BUF1	Active flags for buffers
\$00AF		Cont.: active flags for buffers
\$00B0		Cont.: active flags for buffers
\$00B1		Cont.: active flags for buffers
\$00B2		Cont.: active flags for buffers
\$00B3		Cont.: active flags for buffers
\$00B4		Cont.: active flags for buffers
\$00B5	RECL	Low record # find Relative file
\$00B6		Cont.: low record # find REL file
\$00B7		Cont.: low record # find REL file
\$00B8		Cont.: low record # find REL file
\$00B9		Cont.: low record # find REL file

HEX	LABEL	FUNCTION
\$00BA		Cont.: low record # find REL file
\$00BB	RECH	High record # to find REL file
\$00BC		Cont.: high rec. # find REL file
\$00BD		Cont.: high rec. # find REL file
\$00BE		Cont.: high rec. # find REL file
\$00BF		Cont.: high rec. # find REL file
\$00C0		Cont.: high rec. # find REL file
\$00C1	NR	Next record table
\$00C2		Cont.: next record table
\$00C3		Cont.: next record table
\$00C4		Cont.: next record table
\$00C5		Cont.: next record table
\$00C6		Cont.: next record table
\$00C7	RS	Relative record size table
\$00C8		Cont.: relative record size table
\$00C9		Cont.: relative record size table
\$00CA		Cont.: relative record size table
\$00CB		Cont.: relative record size table
\$00CC		Cont.: relative record size table
\$00CD	SS	Side sector table
\$00CE		Cont.: side sector table
\$00CF		Cont.: side sector table
\$00D0		Cont.: side sector table
\$00D1		Cont.: side sector table
\$00D2		Cont.: side sector table
\$00D3	FIPTR	File stream 1 pointer
\$00D4	RECTR	1st byte wanted from REL record
\$00D5	SSNUM	Side sector number of REL file
\$00D6	SSIND	Index to side sector
\$00D7	RELPTR	Ptr/1st byte wanted in REL file
\$00D8	ENTSEC	Sector of directory entries
\$00D9		Cont.: sector of dir. entries
\$00DA		Cont.: sector of dir. entries
\$00DB		Cont.: sector of dir. entries
\$00DC		Cont.: sector of dir. entries
\$00DD	ENTIND	Index of directory entries
\$00DE		Cont.: index of directory entries
\$00DF		Cont.: index of directory entries
\$00E0		Cont.: index of directory entries

HEX	LABEL	FUNCTION
\$00E1		Cont.: index of directory entries
\$00E2	FILDRV	Default flag, drive number
\$00E3		Cont.: default flag, drive number
\$00E4		Cont.: default flag, drive number
\$00E5		Cont.: default flag, drive number
\$00E6		Cont.: default flag, drive number
\$00E7	PATTYP	Pattern, renew, closed-flags, type
\$00E8		Cont.: pat., renew, close-flag, type
\$00E9		Cont.: pat., renew, close-flag, type
\$00EA		Cont.: pat., renew, close-flag, type
\$00EB		Cont.: pat., renew, close-flag, type
\$00EC	FILTYP	Channel file type
\$00ED		Cont.: channel file type
\$00EE		Cont.: channel file type
\$00EF		Cont.: channel file type
\$00F0		Cont.: channel file type
\$00F1		Cont.: channel file type
\$00F2	CHNRDY	Channel status
\$00F3		Continued: channel status
\$00F4		Continued: channel status
\$00F5		Continued: channel status
\$00F6		Continued: channel status
\$00F7		Continued: channel status
\$00F8	EOIFLG	Temporary EOI
\$00F9	JOBNUM	Current job number
\$00FA	LRUTBL	Least recently used buffer table
\$00FB		Cont.: least recent used buf. tab.
\$00FC		Cont.: least recent used buf. tab.
\$00FD		Cont.: least recent used buf. tab.
\$00FE		Cont.: least recent used buf. tab.
\$00FF	NODRV	No drive flag: drive 0
\$0100		drive 1 un-used
\$0101	DSKVER	DOS version from 18,0
\$0102		Cont.: DOS version from 18,0
\$0103	ZPEND	UN-USED

STACK AREA \$0104-\$01FF	
HEX LABEL	FUNCTION
\$0200-CMDBUF	Command buffer
\$0229	
\$022A CMDNUM	Command number
\$022B-LINTAB	Secondary address: index table
\$202D	
\$023E-CHNDAT	Channel data byte
\$0243	
\$0244-LSTCHR	Channel last character ptr.
\$0249	
\$024A TYPE	Active file type
\$024B STRSIZ	String size in command buffer
\$024C TEMPSA	Temporary secondary address
\$024D CMD	Temporary job command
\$024E LSTSEC	Last sector
\$024F BUFUSE	Buffer allocation
\$0251 MDIRTY	Bam dirty flag: drives 0 and 1
\$0252	Cont.: bam dirty flag
\$0253 ENTFND	Directory entry found flag
\$0254 DIRLST	Directory listing flag
\$0255 CMDWAT	Command waiting flag
\$0256 LINJSE	Logical index (lindx) use word
\$0257 LBUSED	Last buffer used
\$0258 REC	Record size
\$0259 TRKSS	Track of side sector
\$025A SECSS	Sector of side sector
\$025B LSTJOB	Last job
\$025C	Continued: last job
\$025D	Continued: last job
\$025E	Continued: last job
\$025F	Continued: last job
\$0260 DSEC	Sector of directory entry
\$0261	Cont.: sector of directory entry
\$0262	Cont.: sector of directory entry
\$0263	Cont.: sector of directory entry
\$0264	Cont.: sector of directory entry
\$0265	Cont.: sector of directory entry
\$0266 DIND	Index of directory entry



HEX LABEL	FUNCTION
\$0267	Cont.: index of directory entry
\$0268	Cont.: index of directory entry
\$0269	Cont.: index of directory entry
\$026A	Cont.: index of directory entry
\$026B	Cont.: index of directory entry
\$026C ERWORD	Error word for recovery
\$026D ERLED	Error led mask for flashing
\$026E PRGDRV	Last program drive
\$026F PRGSEC	Last program sector
\$0270 WL INDX	Write logical index
\$0271 RL INDX	Read logical file
\$0272 NBTEMP	Number blocks temporary
\$0273	Cont.: number blocks temporary
\$0274 CMDSIZ	Command string size
\$0275 CHAR	Character under parser
\$0276 LIMIT	Pointer limit in compare
\$0277 F1CNT	File stream 1 count
\$0278 F2CNT	File stream 2 count
\$0279 F2PTR	File stream 2 pointer
PARSER TABLES \$027A-\$0289	
\$027A F1LTBL	Filename pointer
\$027B	Continued: filename pointer
\$027C	Continued: filename pointer
\$027D	Continued: filename pointer
\$027E	Continued: filename pointer
\$027F	Continued: filename pointer
\$0280 F1LTRK	1st file link (track)
\$0281	Continued: 1st file link (track)
\$0282	Continued: 1st file link (track)
\$0283	Continued: 1st file link (track)
\$0284	Continued: 1st file link (track)
\$0285 F1LSEC	1st file link (sector)
\$0286	Continued: 1st file link (sector)
\$0287	Continued: 1st file link (sector)
\$0288	Continued: 1st file link (sector)
\$0289	Continued: 1st file link (sector)
\$028A PATFLG	Pattern presence flag

HEX	LABEL	FUNCTION
\$028B	IMAGE	File stream image
\$028C	DRVCNT	Number of drive searches
\$028D	DRVFLG	Drive search flag
\$028E	LSTDRV	Last drive without error
\$028F	FOUND	Found flag in directory searches
\$0290	DIRSEC	Directory sector
\$0291	DELSEC	Sector of 1st available entry
\$0292	DELIND	Index of 1st available entry
\$0293	LSTBUF	=0 if last block
\$0294	INDEX	Current index in buffer
\$0295	FILCNT	Counter of file entries
\$0296	TYPFLG	Match by type flag
\$0297	MODE	Active mode (r,w)
\$0298	JOBRTN	Job return flag
\$0299	EPTR	Pointer for recovery
\$029A	TOFF	Total track offset
\$029B	UBAM	Last bam update pointer
\$029C		Cont.: last bam update pointer
\$029D	TBAM	Track number of bam image
\$029E		Cont.: track number of bam image
\$02A1-BAM		BAM images
\$02B0		Continued: BAM images
<b>OUTPUT BUFFERS \$02B1-\$02F8</b>		
\$02B1-NAMBUF		Directory buffer
\$02D4		
\$02D5-ERRBUF		Error message buffer
\$02F8		
\$02F9 WBAM		Don't write bam flag
\$02FA NOBL		Blocks free low byte, drive 0 & 1
\$02FB		Cont.: block free low byte
\$02FC NOBH		Blocks free high byte, drive 0 & 1
\$02FD		Cont.: blocks free high byte
\$02FE PHASE		Phase offset
\$02FF		Continued: phase offset

DATA BUFFERS \$0300-\$06FF	
HEX LABEL	FUNCTION
\$0300-BUFF0	Buffer #0
\$03FF	Continued: Buffer #0
\$0400-BUFF1	Buffer #1
\$04FF	Continued: Buffer #1
\$0500-BUFF2	Buffer #2
\$05FF	Continued: Buffer #2
\$0600-BUFF3	Buffer #3
\$06FF	Continued: Buffer #3
\$0620 CNT	Error counter: decrements from 10
\$0620 FMTVAR	Format variable
\$0621 NUM	Number between sync & non-sync
\$0623 TRYS	Number of tries in verify
\$0624 TRAL	
\$0625	
\$0626 DTRCK	Distance to track
\$0627 REMDR	Remainder of size
\$0628 SECT	Sector number counter
SERIAL I/O 6522 \$1800-\$180F	
\$1800 PB	Data port b
\$1801 PA1	Data port a: UN-USED
\$1802 DORB1	Data direction register port b
\$1803 DDRA1	Data direction register port a
\$1804 T1LC1	Timer 1 low count
\$1805 T1HC1	Timer 1 high counter
\$1805 TIMER1	Timer 1 counter
\$1806 T1LL1	Timer 1 low latch
\$1807 T1HL1	Timer 1 high latch
\$1808 T2LC1	Timer 2 low counter
\$1809 T2HC1	Timer 2 high counter
\$180A SR1	Shift register
\$180B ACR1	Auxillary control register
\$180C PCR1	Peripheral control register
\$180D IFR1	Interrupt flag register
\$180E IER1	Interrupt enable register

DISK CONTROLLER 6522 \$1C00-\$1C0F	
HEX LABEL	FUNCTION
\$1C00 DSKCNT	Disk controller I/O control line bit 0: step head in bit 1: step head out bit 2: motor on (1), off (0) bit 3: drive active LED on/off bit 4: write protect sense bit 5: density select 0 bit 6: density select 1 bit 7: sync detect
\$1C01 DATA2	Data port a
\$1C02 DDRB2	Data direction for port b
\$1C03 DDRA2	Data direction for port a
\$1C04 T1LC2	Timer 1 low counter
\$1C05 T1HC2	Timer 1 high counter
\$1C06 T1LL2	Timer 1 low counter
\$1C07 T1HL2	Timer 1 high counter
\$1C08 T2LL2	Timer 2 low counter
\$1C09 T2LH2	Timer 2 high counter
\$1C0A SR2	Shift register
\$1C0B ACR2	Auxillary control register
\$1C0C PCR2	Peripheral control register
\$1C0D 1FR2	Interrupt flag register
\$1C0E 1ER2	Interrupt enable register

-----  
NOTES  
-----

**Appendix**

**D**

HEX	DECIMAL	BINARY	GCR
\$0000	0	0000	01010
\$0001	1	0001	01011
\$0002	2	0010	10010
\$0003	3	0011	10011
\$0004	4	0100	01110
\$0005	5	0101	01111
\$0006	6	0110	10110
\$0007	7	0111	10111
\$0008	8	1000	01001
\$0009	9	1001	11001
\$000A	10	1010	11010
\$000B	11	1011	11011
\$000C	12	1100	01101
\$000D	13	1101	11101
\$000E	14	1110	11110
\$000F	15	1111	10101

GCR is the method in which data is magnetically written to the disk. Encoding involves breaking up each 8 bit byte (binary) into two 4-bit nibbles. The 5-bit equivalent for each nibble can be found by looking at the above table. The resulting 10 bits are stored in two consecutive memory locations.

## Bit Values

BIT	DECIMAL	HEX	BIT	DECIMAL	HEX
0	1	\$0001	8	256	\$0100
1	2	\$0002	9	512	\$0200
2	3	\$0003	10	1024	\$0400
3	4	\$0004	11	2048	\$0800
4	5	\$0005	12	4096	\$1000
5	6	\$0006	13	8192	\$2000
6	7	\$0007	14	16384	\$4000
7	8	\$0008	15	32768	\$8000

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

# Hex To Decimal Conversion Chart



## ASCII Conversion Chart

				EVEN PARITY		ODD PARITY		
DEC	HEX	CBM	TRUE	DEC	HEX	DEC	HEX	EBC- DIC
0	00		NUL	0	00	128	80	00
1	01		SOH	129	81	1	01	01
2	02		STX	130	82	2	02	02
3	03		ETX	3	03	131	83	03
4	04		EOT	132	84	4	04	37
5	05		ENQ	5	05	133	85	20
6	06		ACK	6	06	134	86	2E
7	07		BEL	135	87	7	07	2F
8	08		BS	136	88	8	08	16
9	09		HT	9	09	137	89	05
10	0A		LF	10	0A	138	8A	25
11	0B		VT	139	8B	11	0B	0B
12	0C		FF	12	0C	140	8C	0C
13	0D		CR	141	8D	13	0D	0D
14	0E		SO	142	8E	14	0E	0E
15	0F		SI	15	0F	143	8F	0F
16	10		DLE	144	90	16	10	10
17	11		DC1	17	11	145	91	11
18	12		DC2	18	12	146	92	12
19	13		DC3	147	93	19	13	13
20	14		DC4	20	14	148	94	14
21	15		NAK	149	95	21	15	3D
22	16		SYN	150	96	22	16	32
23	17		ETB	23	17	151	97	26
24	18		CAN	24	18	152	98	18
25	19		EM	153	99	25	19	19
26	1A		SUB	154	9A	26	1A	3F
27	1B		ESC	27	1B	155	9B	27
28	1C		FS	156	9C	28	1C	22
29	1D		GS	29	1D	157	9D	
30	1E		RS	30	1E	158	9E	35
31	1F		US	159	9F	31	1F	
32	20			160	A0	32	20	40
33	21			33	21	161	A1	5A

<-- Protection Revealed -->

				EVEN PARITY		ODD PARITY		
DEC	HEX	CBM	TRUE	DEC	HEX	DEC	HEX	EBC- DIC
34	22	"	"	34	22	162	A2	7F
35	23	#	#	163	A3	35	23	7B
36	24	\$	\$	36	24	164	A4	5B
37	25	%	%	165	A5	37	25	6C
38	26	&	&	166	A6	38	26	50
39	27	'	'	39	27	167	A7	7D
40	28	<	<	40	28	168	A8	4D
41	29	>	>	169	A9	41	29	5D
42	2A	*	*	170	AA	42	2A	5C
43	2B	+	+	43	2B	171	AB	4E
44	2C	,	,	172	AC	44	2C	6B
45	2D	-	-	45	2D	173	AD	60
46	2E	.	.	46	2E	174	AE	4B
47	2F	/	/	175	AF	47	2F	61
48	30	0	0	48	30	176	B0	F0
49	31	1	1	177	B1	49	31	F1
50	32	2	2	178	B2	50	32	F2
51	33	3	3	51	33	179	B3	F3
52	34	4	4	180	B4	52	34	F4
53	35	5	5	53	35	181	B5	F5
54	36	6	6	54	36	182	B6	F6
55	37	7	7	183	B7	55	37	F7
56	38	8	8	184	B8	56	38	F8
57	39	9	9	57	39	185	B9	F9
58	3A	:	:	58	3A	186	BA	7A
59	3B	;	;	187	BB	59	3B	5E
60	3C	<	<	60	3C	188	BC	4C
61	3D	=	=	189	BD	61	3D	7E
62	3E	>	>	190	BE	62	3E	6E
63	3F	?	?	63	3F	191	BF	6F
64	40	@	@	192	C0	64	40	7C
65	41	a	A	65	41	193	C1	C1
66	42	b	B	66	42	194	C2	C2
67	43	c	C	195	C3	67	43	C3
68	44	d	D	68	44	196	C4	C4
69	45	e	E	197	C5	69	45	C5

				EVEN PARITY		ODD PARITY		
DEC	HEX	CBM	TRUE	DEC	HEX	DEC	HEX	EBC- DIC
70	46	f	F	198	C6	70	46	C6
71	47	g	G	71	47	199	C7	C7
72	48	h	H	72	48	200	C8	C8
73	49	i	I	201	C9	73	49	C9
74	4A	j	J	202	CA	74	4A	D1
75	4B	k	K	75	4B	203	CB	D2
76	4C	l	L	204	CC	76	4C	D3
77	4D	m	M	77	4D	205	CD	D4
78	4E	n	N	78	4E	206	CE	D5
79	4F	o	O	207	CF	79	4F	D6
80	50	p	P	80	50	208	D0	D7
81	51	q	Q	209	D1	81	51	D8
82	52	r	R	210	D2	82	52	D9
83	53	s	S	83	53	211	D3	E2
84	54	t	T	212	D4	84	54	E3
85	55	u	U	85	55	213	D5	E4
86	56	v	V	86	56	214	D6	E5
87	57	w	W	215	D7	87	57	E6
88	58	x	X	216	D8	88	58	E7
89	59	y	Y	89	59	217	D9	E8
90	5A	z	Z	90	5A	218	DA	E9
91	5B	[	[	219	DB	91	5B	NA
92	5C			92	5C	220	DC	E0
93	5D	]	]	221	DD	93	5D	NA
94	5E	†	†	222	DE	94	5E	NA
95	5F			95	5F	223	DF	6D
96	60			96	60	224	E0	79
97	61		a	225	E1	97	61	81
98	62		b	226	E2	98	62	82
99	63		c	99	63	227	E3	83
100	64		d	228	E4	100	64	84
101	65		e	101	65	229	E5	85
102	66		f	102	66	230	E6	86
103	67		g	231	E7	103	67	87
104	68		h	232	E8	104	68	88
105	69		i	105	69	233	E9	89

				EVEN PARITY		ODD PARITY		
DEC	HEX	CBM	TRUE	DEC	HEX	DEC	HEX	EBC- DIC
106	6A		j	106	6A	234	EA	91
107	6B		k	235	EB	107	6B	92
108	6C		l	108	6C	236	EC	93
109	6D		m	237	ED	109	6D	94
110	6E		n	238	EE	110	6E	95
111	6F		o	111	6F	239	EF	96
112	70		p	240	F0	112	70	97
113	71		q	113	71	241	F1	98
114	72		r	114	72	242	F2	99
115	73		s	243	F3	115	73	A2
116	74		t	116	74	244	F4	A3
117	75		u	245	F5	117	75	A4
118	76		v	246	F6	118	76	A5
119	77		w	119	77	247	F7	A6
120	78		x	120	78	248	F8	A7
121	79		y	249	F9	121	79	A8
122	7A		z	250	FA	122	7A	A9
123	7B			123	7B	251	FB	C0
124	7C			252	FC	124	7C	6A
125	7D			125	7D	253	FD	D0
126	7E			126	7E	254	FE	A1
127	7F		DEL	255	FF	127	7F	07

### 1541 DISK DRIVE COMMANDS

LOAD.....  
LOAD"name",8.....Load basic program  
LOAD"\*",8.....Load first program on directory  
LOAD"name",8,1.....Load ML PROGRAM  
LOAD"name",8,3.....Automatic load and run  
LOAD"\$",8.....Load the directory  
LOAD"A\$",8.....Load contents of a VARIABLE  
LOAD"T\*",8.....Load 1st file starting with T  
LOAD"Jan\*",8..Load 1st file starting with Jan  
SAVE"NAME",8...to save file in memory to disk  
SAVE"@:NAME",8...save new program, replace old  
VERIFY"NAME",8.....Compare file/one in memory  
OPEN 15,8,15..Open channel and command channel  
PRINT#15:CLOSE15.....To close channel  
PRINT#15,"N:name,ID".....New a disk (format)  
PRINT#15,"N:name".....Clear directory  
PRINT#15,"C:new file=old file"..duplicate file  
PRINT#15,"R:new name=old name"...rename a file  
PRINT#15,"S:name".....Erase (scratch) a file  
PRINT#15,"I".....drive to start-up condition  
PRINT#15,"V"....re-organize, collect scattered  
                  blocks & make space available  
ERROR CHANNEL.....To read type the following:

10 OPEN 15,8,15  
20 INPUT#15,A\$,B\$,C\$,D\$  
30 PRINT A\$,B\$,C\$,D\$  
40 CLOSE 15

### DOS WEDGE COMMANDS

@ or >.....To display the directory  
@ or >.....Disk status, reads error channel  
↑ filename.....Load and automatic run  
/ filename.....Load program  
% filename.....Load file at it's own address  
  filename.....Save file  
@ N0:disk name,ID.....To format a disk  
@ N0:disk name.....To rename the directory  
@ S0:name of file....To scratch or remove file  
@ V0.....To validate the disk  
@ R0:old name=@0:new name.....To rename a file

# 1541 Utility Instruction Set

< - Protection Revealed - >

COMMAND	FORMAT	DESCRIPTION
BLOCK-READ	...B-R...	...PRINT#15,"B-R:"channel/drive/track;sector
BLOCK-ALLOCATE	...B-A...	...PRINT#15,"B-A:"drive/track;sector
BLOCK-WRITE	...B-W...	...PRINT#15,"B-W:"channel/drive/track;sector
BLOCK-EXECUTE	...B-E...	...PRINT#15,"B-E:"channel/drive/track;sector
BUFFER-POINTER	...B-P...	...PRINT#15,"B-P:"channel;byte
BLOCK-FREE	...B-F...	...PRINT#15,"B-F:"drive/track;sector
MEMORY-WRITE*	...M-W...	...PRINT#15,"M-W:"CHR\$(address low)CHR\$(address high) CHR\$(#bytes)CHR\$(data)CHR\$(data)...
MEMORY-READ*	...M-R...	...PRINT#15,"M-R"CHR\$(address low)CHR\$(address high)
MEMORY-EXECUTE*	...M-E...	...PRINT#15,"M-E"CHR\$(address low)CHR\$(address high)
U1	UA	...Replacement for BLOCK-READ
U2	UB	...Replacement for BLOCK-WRITE
U3	UC	...Disk Processor JMP \$0500
U4	UD	...Disk Processor JMP \$0503
U5	UE	...Disk Processor JMP \$0506
U6	UF	...Disk Processor JMP \$0509
U7	UG	...Disk Processor JMP \$050C
U8	UH	...Disk Processor JMP \$050F
U9	UI	...Disk Processor JMP \$FFFA
U:	UJ	...Disk Processor JMP power-up vector

\*THESE INSTRUCTIONS MUST BE ABBREVIATED



# INTRODUCING

Protection Revealed.....  
Take command of your software. End the spiral of buying copy programs. Protection Revealed unravels the puzzle of software protection. For authors and users, become a master of illusion. Learn how to protect or un-protect your software. Includes a large section on successful BASIC program security.

Covers the latest evolution of copy protection techniques.

Disk Errors	De-compilers
Half Tracks	Gap Bytes
Extra Sector	Nibble Counting
Guard Band	DOS Protection
Compilers	Density Changes

And much, much more.

Reveals the mystical professional trade secrets of the Commodore 64 and the 1541 Disk Drive.

This is the protection book for everyone.

Disk included, containing 21 utility programs. All menu driven, easy to use.....