THE SOFTWARE PROTECTION HANDBOOK

— For the C-64 —

David Thom and Vic Numbers

A PUBLICATION OF
PSIDAC

THE SOFTWARE PROTECTION HANDBOOK For the C-64

DAVID THOM

and

VIC NUMBERS

COVER and ILLUSTRATION

ACE CAMPBELL

PUBLISHER'S NOTE

*** PROGRAM NOTES ***

Most of the Basic programs will require that you use CBM "shorthand" when typing to avoid excess memory use.

Programs have been written for the Commodore 1541 disk and 1525 printer. Many of the programs have been tested on a version of the MSD. The programs which "talk" to the drive controller such as Diskpicker and error analyzers will not work with MSD. The others may work.

THE SOFTWARE PROTECTION HANDBOOK

SPH-64 V1284 USD19.95

THIRD PRINTING

FOREWORD

The philosophy of this book embraces three main goals.

1. To provide broad view of protection from laws and ethics through tools and techniques.

2. To alert you, the consumer, of the "vicious circle" nature of protection/protection breakers, and the potential cost of getting caught in this spiral.

3. To remove some of the mystery surrounding the subject while providing a useful reference document.

*** CONTENTS ***

*** CONTENTS CONT. ***

```
DISK NAME = PSIPACK (C)1984 VBN 3RD ED.

TYPE   TRACK   SECTOR     NAME             BLKS

PRG     17       0        1PSIMAIN        " 2
PRG     17       1        2PSIMAIN        " 2
PRG     20      12       SUPERDIRECTORY  " 8
PRG     19      13       DISK-EDITOR     " 7
PRG     17      19       RELOCATE/LOADER " 2
PRG     15       2        DISKPICKER      " 20
PRG     19       9        T/S ANALYZER    " 10
PRG     19       0        FASTBACK        " 7
PRG     17      15       1DUPDAC         " 17
PRG     16       7        2DUPDAC         " 17
PRG     20       2        ADMACH          " 1
PRG     20       3        MACHRELO        " 1
PRG     20       4        3ROMULATOR      " 4
PRG     20       6        2ROMULATOR      " 4
PRG     19       8        ZMACH           " 1
PRG     20       9        ANALYMACH       " 1
PRG     17       3        SUPERMON64.V1   " 10
PRG     19       6        ERROR ANALYZER  " 7
PRG     17       6        20 NO HEADER    " 1
PRG     17       7        21 ERASE TRK    " 1
PRG     17      16       22 NO DATA      " 1
PRG     16       0        23 DATCHKSUM    " 1
PRG     16       1        SYNC WRITER     " 1
PRG     19       5        HELP            " 4
PRG     17       8        WRITE HDR       " 1
PRG     17       2        READ HDR        " 1
PRG     19      14       COPY HDR        " 1
PRG     19       7        CON HDR         " 1
PRG     19      15       1CON HDR        " 1
PRG     16       3        LINKSTER        " 5
PRG     14       9        TMACHRELO       " 1
PRG     14      15       TRELO           " 7
PRG     19       4        SYNC ONLY       " 1
PRG     19      16       1TRKFMT         " 1
PRG     17      17       TRACK CHECK     " 10
```

PREFACE

The writing of this book was undertaken as a result of the large number of inquiries and strong interest developed after the release of our first book "The Software Pirate's Handbook for the VIC-20" (SPH-20). This book will follow a similar format to the SPH-20 and, although each are complete books in their own right, this book is designed as a complement to SPH-20.

The SPH-64 will expand on both the philosophy and technique of duplicating software. In doing so we want to dispel many of the doubts and fears that people often have about copying software. We also want to be sure that the legal issues are understood so that we do not promote theft or black-marketing of software which clearly are crimes. At the same time we will show you how to protect your own investments by making back-up copies of a variety of "protected" forms of software.

This book will first cover our philosophy for copying, and specific legal considerations. In the following chapters we will outline technical theory and concepts with specific numbered procedures and the program listings at the end of applicable chapters. In many cases you will be able to go directly to the specific procedure for the type of copy you wish to make. However for a better understanding or in case of difficulty in making a copy, you may need to refer to the sections which give attention to theory and concepts. In this manner you should be able to gain an understanding of the kinds of protection that you are encountering so that you can deduce possible "countermeasures".

The ultimate purpose of this book is not to provide you with the "newest thing" in copy history, but rather, we will try to provide you with primary tools and knowledge of protection systems. These tools are not candied for appeal but are essential utilities which are themselves open and unprotected so that you can use, study, and adapt them to a continually changing market. Our experience has shown that "packaged" protection breaking software becomes obsolete as fast as it is written, leaving the consumer with yet another hole to pour money into. Adaptive programs and user skills are necessary to stay current. In light of this, we offer this book, not as the final word, but as what may be the first technical reference document available on the subject. Together with the programs included, this makes a powerful analytical package for defeating protected software.

We greatly hope that you find this book informative, understandable, and USEFUL!

The authors.

*** CHAPTER ONE ***

INTRODUCTION

PIRACY...an issue so clouded with fear, intrigue, and misinformation that you dare only to utter it in safe, familiar company. The original title of this book was to be "The Software Pirate's Handbook II", but had to be changed after several advertisers flatly refused to advertise a book of that title in spite of the fact that the book was never meant to encourage piracy in any form. The use of "Pirate" was intended as a light-hearted reference to any copying process, and to inspire a certain tendancy of humankind; the attraction to things mysterious or secret.

Unfortunately, there is a great deal of hocus-pocus and puffery being used to cloak the alleged brainchildren of this new market. This reletively new consumer product - software - is being hawked, with all the vigor and claims of mysterious powers,like the patent medicines of earlier years. This technocratic rhetoric is replete with all the rumored pitfalls and warnings of what may happen to you or your equipment if you try to exercise your documented legal right to copy the product. All of this has the detrimental end effect of deluding the consumer into purchasing overpriced, underperforming products. In our opinion this will most certainly undermine the strong beginning and curiosity the home microcomputer market now has. It is our goal to provide the public with some knowldge, tools, and attitudes that will be a start in changing this situation. I see two futures for home microcomputing, one with software and computers accepted and used by many, and the other with software and computers rejected as expensive "hyped" toys of a bygone fad.

At this point we must separate the difference between what is acceptable copying and what is Piracy. Quite simply, the intent of the user will decide the question. Copying for sale, distribution or other non-personal uses is Piracy. Copying for backup, archival, study, and other personal uses is not piracy. Loaning your original to another person for temporary use is not piracy. (If that ever were changed we may as well burn the libraries and return to the Dark Ages!) However copying an original you do not own is unethical.

We should start to analyze this whole issue of copying software by first classifying software amongst the products with which it belongs. This will also help to peel away some of the misinformation that surrounds the issue. To get very basic, software is a set of instructions which produce a desired effect on a physical system. Software is written with letters and numbers and can be embodied in many different forms such as; verbal, magnetic recording, paper, solid state and so on...just like this book or a piece of music or a recipe for example. The fact is that the examples that I have given are just as common or likely to be found in one of the formats listed as software itself, with the exception of solid state, which refers mainly to ROM memory. The reason that books and other typically printed materials are not commonly found in this format is that they inherently contain vastly greater amounts of data than the average program. Any one of these also often cost more to produce, take more expensive equipment, and more time to write than a piece of software. How strange it is then that software should sell for tens of times more money than these other like products!

I suppose that the real mystique of software, and the factor most capitalized on, is it's code-like nature. Few people are professional programmers and thus cannot really understand the mysterious language of the program. Some might say that all that is important is that the user be able to use the program and that the degree of secrecy or protection applied to the program is not a concern. We feel that this is a little like a homemakers magazine saying that you should only be able to eat the end result of a recipe, not be able to understand the words and numbers which define the end product! If you only wish to eat and care less about the ingredients, that is your perogative, but if you like to understand what you are eating, there really should be no great mystery about the ingredients.

When it comes to making a copy of a piece of software many people get cold feet. In selling the "Pirate's Handbook" for the Vic-20, we advertised "for archival use only". We recieved many promises, signed statements and so on from our customers that they would use it only for "archival" purposes. I am sorry that so many people have been bullied into thinking that they might be required to make such a statement in the first place. I have yet to meet a person who felt the need to sign an affadavit before running up to the local grocery store to make a copy of a magazine article, pages of a book, recipe or whatever. Probably the only situation even near in comparison would be the fear of attempting to photocopy a dollar bill to try in a bill changer. (it doesn't work!)

A recent manifestation of this biased thinking on software appears in a proposal being considered for state law in Louisiana. The law, if enacted, would mandate that the act of purchasing and subsequently opening the package of a piece of software, would inherently place the "opener" in a legal contract with the seller. The purchaser would be obligated to refrain from copying, distributing, and whatever else they throw in. I hope they have a lot of jail space in Louisiana!

One of my favorite analogies which may help you to put this whole ethical question in perspective is the recipe analogy. Consider the fact that a recipe like software is a set of written instructions which cause a desired effect on a physical system. Like software, it requires a sequence of steps to be performed in the correct order in real time. Both require specialized hardware to perform their respective functions. As with computer hardware, a complete cooking system easily costs thousands of dollars. Developing a unique recipe requires specialized skill as well as a great deal of time.

Some of the major differences are a result of the general perception of a recipe as an ordinary everyday commodity often given away and the perception of a program as a highly valuable and unfathomable product. It is easy to believe that a great deal more knowledge and effort goes into a program than a recipe. However there are a lot of professionally trained chefs who would argue otherwise! Judging from some of the "professional" software that I have seen marketed, I am quite sure that they are right!

If you can then, consider that these diverse yet similar copyrightable literary forms should be given equal treatment and respect when we think of copying.

Are womens clubs and church groups that exchange recipes (many of which are blatantly copied from the pages of magazines) commiting an act of piracy? Probably they are in the strictest sense, but who really cares!? It doesn't appear that their doing so has seriously affected the market. If taken case by case, each category of information market has a similar situation. For each of these markets there is some form of equipment and general knowledge of methods for making copies of the copyrighted product. Your right to own and know these facts has never really been disallowed. Otherwise, tape recorders, photocopy machines, and probably even cameras could have been outlawed due to their potential for illegal use in copying protected information. Each of these markets also face the real threat of blatant piracy by those who would profit by use of copy technology. The current misunderstanding of the ethical question of copying software is not a result of a factual difference in the nature of software but rather a simple difference in the typical perception of software versus more commonly understood information like a recipe.

Historically software has been a closely guarded secret of the company owning it. This results largely from the fact that prior to 1980 software had no legal form of protection. It did not fall under patent law or copyright law. Also most software was for larger businesses as home use was not that extensive. Naturally the market was limited and the price necessarily high. As a result companies were rightfully worried about being ripped-off by those who might copy their software and sell it without fear of punity. This unfortunatly, happened too often. More recently for example, Franklin computer copied many versions of Apple computer's operating system. The uncertanty of the new laws protecting software prevented Apple from stopping this apparently obvious case of piracy. Although the two companies finally came to an out of court settlement, the 3rd U.S. Circuit Court of Appeals in Philadelphia ruled that the software was protected under current copyright law. "Piracy" of this form could quite literally, bankrupt a company. The operating systems of a microcomputer are closely linked to the hardware design of a system. A company stands to lose the fantastic amount of investment that it takes to set up production and marketing if unscrupulous persons pirate these software systems to install in a competing computer. In effect, this creates a limited monopoly for the original manufacturer. If a competitor wishes to make a 100% compatible alternate, the operating system he uses must be identical in performance. For technical reasons, this is virtually impossible to do without copying the operating system! Although legal protection for ROM based operating systems has been upheld in court, the issue is long from being settled.

The heart of the copyright intent is that the specific expression of an idea is copyrightable but the idea is not. Ideas must be free of legal encumbrances. However, if the idea can only be expressed in one way without losing its meaning, then the ability to protect that idea becomes very limited. The same sort of thing is true when product names become so common as to be considered generic. Legal protection is lost. Thus I can talk about crescent wrenches and skill saws without worrying about the legal ramifications. There is not a clearly defined legal answer available for this problem with operating system software. The real answer may lie in the intent of the person making the copy.

From another viewpoint, an applications package which may take a month or so for a single person to write and an average investment to produce and market should not be priced as the basis for a lifelong income. This does not mean that it should be stolen and distributed according to some modern Robin Hoodian mentality. It does mean that the over-pricing of applications software creates a climate where such piracy is going to be common. If the price better accommodated the market, the tendancy of people to distribute copyrighted software would greatly diminish.

Protection of software is clearly not the answer. I would not even recommend that anyone buy software if the protection methods seem too strong. The ability to make backups and expect a reliable and compatible product are rights of a consumer. As later chapters will bear out, some of the more elaborate protection schemes can prove rather unreliable. Often they are obvious encumbrances to smooth loading and running of a program. Furthermore, in the case of disks, they often are not compatible with other brands of disk drives. The real clincher is that Commodore, or any other drive manufacturer, reserve the right to update their drive hardware/software at any time as needed. Protection schemes which operate beyond the defined specifications of the system would not necessarily work on an "updated" system! The drive manufacturer has no obligation to try to figure out what has been done beyond the original accepted specifications. If you don't believe this is a real problem, try talking to anyone who has bought "Commodore compatible" hardware.

The difficulty of backing up highly protected software places you in the vulnerable position of facing complete system crash and resulting data loss. In many cases I can think of it would be totally unacceptable to even run a program without having a backup accessible within minutes. Although failure of the storage medium is more common with tapes and disks, it is still possible with cartridges. The major problems with cartridges are that a lot of wear and tear is placed on the port by constantly changing cartridges. As anyone experienced in failure analysis could tell you, the weakest link in many electronic systems is the mechanical interface such as edge connectors. I have seen them become loose, bent, intermittent and who knows what else. One popular solution has been to use expander boards with switchable slots but these are quite expensive and still limit the total number of cartridges you may use without switching. Our solution is to use an external RAM memory which can be loaded from tape or disk and then emulate the original ROM cartridge. This way, the RAM unit is left in the computer and does not interfere with any other operation. When desired, the cartridge program which was previously saved on disk or tape can be loaded and reloaded as needed. Your cartridges can be put away safely and be used only to make backups. The obvious problem with this system is that people who don't own the cartridge could illegaly obtain copies of the tape or disk and use the program without purchasing it. We strongly discourage this. It is an illegal act of piracy. Chapter 7 gives specific details on this system.

The point of importance is that without a backup copy of each program you endanger yourself needlessly. Feeble attempts of software houses to remedy this by offering backups if your original fails, simply do not solve the problems. This is true even IF the backup policy is quick and promises a replacement within say 24hrs. What happens if the original company goes out of business? What if the user faces a deadline or is in the middle of a presentation? In some cases a delay of more than some minutes can be devastating. In our opinion there is only one rule and all should take heed: Only a fool runs a program without a backup but it is a fools fool who runs a program without a backup of the backup! For those really important programs we recommend that backups be stored in two different locations. In one case, one of our programs was recovered from a disk which had it's protective cover cut away and had been thrown in the trash - after being passed around to show how they were constructed! One of the first things you should do especially with a new disk or tape is to make a backup.

It is likely that you may hear about a "great" program but find out it is uncopiable or very hard to copy. We ask; are you willing to suffer the consequences of a failure with no backup? Is it then, really such a great program? The truth is that such a program rewritten with no protection is much more valuable and marketable. The irony is that selling a program with significant modification may not even BE a violation of copyrights. "Reverse engineering", an accepted industrial practice of distilling the underlying principals of a product and then marketing your own version, has always been a common source of "improved" products.

Another issue which creates a problem for those who purchase software is the lack of documentation of the program itself. Not all people want or care to try to modify a program to suit their application but for many the original program may be ill suited to their needs. These people have a legitimate need to be able to list and modify the program. Without adequate documentation, this is nearly impossible. Our philosophy is that if you buy our book we are happy to give you our listings and you may make any changes that suit your special wishes. We feel that all software should be sold with listings and documentation or that they should be available for a reasonable price. The ability to customize is a unique feature of software which should be capitalized on instead of hidden. Some products are naturally customizable some are not. The custom car parts industry has made a fortune on this very fact. Occasionally a potentially valuable piece of software is totally useless to a person because of the inability to modify it.

The consumer has a legal right to expect a product to perform the "normal" functions associated with that type of product. In legal terms this is known as the merchantability of a product. Imagine that you buy an "all purpose" fertilizer for flowers. After reading all directions and using it, your roses do great but all your other flowers die. Let's say that after talking to your friend, a chemist, you find that it would work on other flowers - IF you had specialized knowledge of the chemical compounds used and how they could be applied successfully to other flowers. A lawyer would probably tell you that you have legal grounds for recovery because the product is not merchantable as an all purpose fertilizer without sufficient instructions on how to use it successfully, and any "normal" application does more harm than good. In the microcomputer business, there are many programs sold with the implication that they will do many things. Only after purchasing might you find out that it would require specialized knowledge or even modification to make it work to reasonable expectations. Since backup copying of owned software is a legal right I wonder if a copy protected program itself violates the principal of merchantability. As long as we have that right it is reasonable to expect software to be copyable.

Time and again various industries have gone through this "secrets" game only to find in the end that a large number of the consumers have a right, need, and demand to know what's inside. Limiting or protecting this information can only limit the useability of the equipment and also limits the growth of add-ons which make the original product more useful. Software houses would certainly view this as opening the door wide to the pirates. Already terrified by loss of profits due to piracy they find more and more elaborate and costly protection to be the answer. Yet this is the very thing that makes the market so attractive to the pirates in the first place! Take away the protection and high price, and who would need a pirate anyway!?

In strict legal terms, software has been protected by U.S. copyright law since 1980. Anyone who makes a copy of software and gives it away or sells it is in direct violation of these laws and prosecution is available. Other information forms have survived very well with copyright law as the only means of protection. The pricing has been set by the accepted method of market determination and copying and exchanging amongst small groups or by friends has never seriously affected the market. The thought of a ladies sewing circle being sued for exchanging patterns seems ludicrous because it is. Large scale piracy does not occur as a result of grocery store photocopy machines because the original is priced low enough that it wouldn't be profitable. With the areas of legality clearly defined, no one should feel intimidated or afraid to make a copy of software for backup purposes. The existance of copying hardware, software, and information should be no more threatening than a photocopy machine or tape recorder and the knowledge of how to use them!

*** CHAPTER TWO ***

TOOLS

Without a doubt the C64 is a powerful computer and a very versatile tool. However, the first time that you try to save a protected program you might suddenly feel like a child given the task of decrypting a classified and encoded top-secret document. A myriad of questions immediately present themselves. Where do you start? What kind of program is it? What methods of protection have been applied? Is there any way to list it? Where does the program reside in memory? And so on... It is often difficult to decide which questions to ask in the first place!

It would be wonderful if someone made a super-copier that would copy any software with the ease that a photocopy machine will copy this page. Unfortunately, there is just no way that a single answer will cover the wide variety of possibilities available for copy protection. The only way to really be equipped for the task is to have various specialized tools. Most of the jobs cannot be accomplished with only one of the tools but will require skilled application of many different tools. The key here is skill. The most effective tool you can posses is a sound knowledge of how software can be protected so that you can make a meaningful analysis of each situation and apply the correct tools with the greatest amount of skill.

In this chapter we will concentrate on familiarizing you with the kinds of tools to use on three categories of software media. These are, cartridges, disks, and tapes. Within these categories we will define "levels" of protection as they apply. The descriptions of tools will aquaint you with the general applications of the tools but we will leave out the specifics of use until the later chapters, which will list the procedures in a very detailed fashion. If breaking protected software is all new to you, this chapter should give you a good general understanding of what you are up against and what tools are available to help you. This should also help you develop plans of attack when you come up against a protected program that you need to back-up.

--Cartridges--

Cartridges are unique in that they are a form of protection in themselves. That is to say that the very features of the cartridge are what protect the program that it contains. The fact that the computer is designed to autorun cartridges makes it difficult to break into and list the program or save it to another format. Since most people don't have PROM memory burning facilities, they cannot expect to make a "copy cartridge" from an original cartridge. Furthermore, the cost of PROM burners and circuit boards to implement a copy cartridge would make it expensive anyway.

Actually cartridges are not that prone to failure. The simple excuse of making a duplicate cartridge for back-up purposes really isn't realistic in view of PROM making costs. We have developed a method that can be used to transfer cartridge programs to tape or disk and then run in a RAM expander! This makes the process very cheap if applied for many cartridges. With this system it becomes desirable to have back-ups for all cartridges you own. After all, if the cost is a little disk or tape real-estate, why not have a copy? There are good reasons to want a tape or disk copy, such as avoiding the hassle of continuously plugging and unplugging cartridges when changing programs. Making a modification of a program residing in a cartridge can't be accomplished unless the program can be put into RAM and run in RAM. A single disk can contain a whole library of programs while the cartridge holds only one. Having drawers full of odd shapped cartridges is a real pain. If, for example, you need to transport a library of cartridges to work each day, you would need an extra lunchbucket! But with your cartridge library on disk, your software is very transportable.

Remember that all these limitations in the cartridge medium are mostly to make copying less likely. It would be much cheaper for a software house to sell all of it's software on either tape or disk. Magnetic media does not require a circuit board or specially manufactured circuits and enclosures. Cartridges exist largely because of the fear of piracy. As is often the case, those with legitimate needs are penalized by actions of those with illegitimate greed.

When the C64 is powered up one of the housekeeping chores that it does is to check to see if a cartridge is plugged in. Part of this is accomplished by two pins on the cartridge edge connector which are identified by the names GAME and EXROM. The job of these pins is to control the way the memory is configured on the C64 so that the computer will allow Basic, Kernal and ROM to be located as necessary. The initialization routines also check memory locations where a cartridge resides to see if an access code is found there. These codes then tell the computer that there indeed is a cartridge and they pass information to the computer so it knows where to start running the program from. A program doesn't need to start at the beginning of the cartridge memory to run. Most of the time the entry point is not at the beginning but further on in cartridge memory. Following is an Interrogate dump of the first several bytes of a typical ROM cartridge. Notice the ASCII (reversed) display which shows the access code CBM80.

The first four bytes before this code give the cold start and warm start addresses respectively.

```
                  CS  $8394    WS  $83A0
.:8000[illegible] 94 83 A0 83 C3 C2 CD 38
.:8008[illegible] 30 02 BB 5A 30 5F EE 3D
.:8010[illegible] A8 C0 C0 80 57 80 57 EE
.:8018[illegible] 57 40 01 C4 04 40 19 CA
.:8020[illegible] CA 4A 19 40 C4 04 01 C0
```

--ROMULATOR--

Romulator is a cartridge copy system that comes in three versions. One for the VIC-20, one for C64 tape systems and one for C64 disk systems. The C64 versions are identified as 2Romulator for disk and 3Romulator for tape. In this book we will refer to the C64 system as Romulator and leave the numbers out except when needed for specific procedures etc. Romulator consists of a special program which moves the contents of the cartridge then saves it to the magnetic medium being used. A special Romulator circuit card is used which allows changing the configuration of the GAME and EXROM lines thus preventing auto starting. This card is used with both the tape and disk versions of the C64 software. The circuit card also has a socket for an 8K or 16K RAM expander. The RAM expander is the key since the tape or disk program can be downloaded to the RAM. The RAM is made to look like a ROM cartridge by the use of a write enable line which prevents erasing of the program by software means. Finally, the RAM location can be changed so that it will reside anywhere commonly used by commercial cartridges.

The general process is to use the Romulator circuit card to determine the normal cartridge configuration, next set the switches to defeat the autorun and then to copy the contents of the cartridge to the selected magnetic medium. The Romulator card with RAM is left in the computer. It does not interfere with any other operation. The cartridge can be stored away safely. When desired to run the program, it is downloaded into the expander RAM which is then write protected and switched into the configuration for the cartridge being run. A system reset button on the Romulator card is then pressed to force a cold start. Since the computer will see the cartridge configuration and cartridge codes, it will run the program as though the cartridge itself were plugged in! This can be done with 8K or 16K cartridges. To date we have not found one that it will not work with. Chapter seven lists the exact procedures to follow as well as the programs and circuits needed.

This book has been written with the aid of Quick Brown Fox word processor which we run on a Romulator system. The cartridge is safely stored in a drawer. It is very nice to be free of the anxiety of having some unexpected "glitch" destroy the cartridge and put us temporarily out of business. Also gone is the fear that an overworked edge connector will become intermittent or fail altogether. If you decide to set up a Romulator system for your computer you will find it a valuable accessory.

--DISKS--

There are a wider variety of protection schemes in existence for disk than any other medium. It is understandable that software houses have developed so many forms of protection since an unprotected disk is so easy to copy. Many people would be tempted to avoid purchasing by copying from a friend, and a few might even try to make a profit by selling pirated copies at a much lower price. The protection methods we will cover will help you to protect your own disk software and to break protection when you need to make a back-up of one you have purchased. As a consumer, you should avoid purchasing "super protected" disks because of the limitations and insecurity they force upon the user. If the super protected program is the best thing around, you will have to weigh the disadvantages versus the quality of the software. You might perhaps settle with purchasing a second original if you need maximum protection from the eventual disk crash. When appropriate, we will spell out specific limitations and problems to normal function introduced by some forms of protection.

We will define "levels" of protection to make our discussion easier. These are not in line with any "standard level", but are merely to give us a yardstick in comparing different kinds of protection. This may also help you in classifying programs that you wish to copy so that you can select the most appropriate form of "attack".

--PROTECTION LEVELS--

Level 0.....Unprotected. Can be saved by loading and saving. Contains basic only or basic loaded machine language.

Level 1.....Contains auto-run feature and a STOP disable poke in program. Doesn't allow saving but can be direct copied or Relocate/loaded and saved. (see copy systems; Direct Duplicator-1 [DD-1] and Relocate/Loader)

Level 2.....Contains "bad sector" errors on disk which prevents most commercial disk duplicators from reading past. Creates "shuts off in the middle of copying" syndrome with many disk copy programs. May also contain L1 techniques. DD-1 will copy these.

Level 3.....Same as above except program will not run without error sectors being put back into the copy. Requires Error Maker program or modification of program to take out sections which "look" for bad sectors.

Level 4.....This is what we will call "advanced" error protection. It involves altering the parameters of the normal drive formatting and/or writing. This will produce symptoms such as extra tracks, specially encoded data or format info, modified headers, "data under errors", and so on. The so called "half tracking" is one of these non-standard writing techniques. This seems to be the direction of newer software and can create severe compatibility problems. Diskpicker can be used to both analyze and develop routines to break anything that fits into the altered DOS category, as that is its primary purpose.

Level 5.....Disk requires hardware module to operate. "Dongle" protected. Requires Dongle synthesizer or modification of program so that it does not look for dongle. This is a "valid" form of protection if it allows backup disks to be made. It will minimize "Piracy" while giving the owner crash protection.

This listing does not attempt to cover every possibility under the sun but it does cover the more common methods that we have encountered. For those levels which require modification of the program, you will need an understanding of how to disassemble programs to make the necessary changes. A full fluency in machine language is beyond the scope of this book and really is an aquired skill. We will try to define the process however to give you some chance. Your own interest in going further is up to you. Level 4 is probably the one that you should avoid purchasing in the first place If you can find a similar program of similar quality without the protection. Don't be fooled though, a highly protected program has no relation whatsoever to the quality of the program. I have seen public domain programs that are significantly better than expensive commercial versions!

Attacking level four protection can be done with Diskpicker. The error making and header modifying routines we give will get you started in this direction.

-- WHOLE DISK DUPLICATORS --

Direct Duplicator-1 was written by Vic Numbers and is listed as a part of this book. PSIDAC holds copyrights on this program. DD-1 overcomes many of the limitations of other "whole disk" copiers. This is especially true for protected disks containing errors which will stop some copiers. Other copiers may attempt to "second guess" where the errors might be and do those sections last, which does not often work. DD-1 does a sector by sector duplication of every track and sector on the disk. It will transfer the contents of bad sectors but cannot reproduce an error. Most duplicators that can get through the errors can't duplicate the errors so programs called Error Makers are needed. These programs and means of clearing errors will be described later. DD-1 has a version for single disk owners and a version for dual disk owners. The single disk version (1DUPDAC and 1PSIMAIN) will require swapping disks. The dual disk version (2DUPDAC and 2PSIMAIN) is essentially a hands free system. Both versions will print the type of error on the screen or optionally to the printer, if the original contains "error protection". This is a powerful feature that gives needed information if you have to reinstall errors or modify the copied program to make it run.

One very unusual feature is a "fast write" mode which tests each byte of data on a sector and skips over any sector which contains normal format data but no program data. This will in no way affect the validity of the copy but it can cut the time to make a duplicate almost in half. Another really handy feature is the ability to write more than one copy per original read. The idea is that once DD-1 has read a buffer full of data (150 blocks) that this data can be written to more than one disk. This saves the extra read which would be redundant since the buffer still contains the data until a new 150 block section is read. Thus making many copies of the same disk can be accomplished at a very fast rate. This is very handy for disks of programs that you have written and wish to distribute. Generally, if the original contains more than three programs and you wish to make several copies, you will save time by using the fast write mode and multiple copy features of DD-1. Since this system is written in machine language (DUPDAC) with a basic controller (PSIMAIN) speed and flexibility are natural.

DD-1 is simple to use and usually overcomes protection on the original. Since the whole process is very direct, you will avoid spending the time required to get an understanding of what is being done on the disk. It is best for the lower levels of protection.

There are several other noteworthy copier programs on the market which you might find especially suited to your needs. We will point out some of their advantages and limitations from our viewpoint. Clone Machine is the trade name of a set of programs released by Micro Ware, of Butler, New Jersey. It offers a whole disk backup program along with other programs similar to the variety we give you in chapter 6. We will cover the other types of tools later in this chapter. The Clone Machine disk duplicater program provides a "graphic" display indicating the reading and writing of sectors. The version we used locked up when an error was encountered. This meant writing down track and sector, then returning to menu options and then trying to continue the copy process from the point ended. We found this extremely cumbersome compared to the automatic error skipping features of DD-1. The whole menu process can sometimes be tedious since typically you will be doing the same thing over and over. It is tiresome having to keep telling it that you're using device 8 each time!

Clone machine uses a 120 block buffer which allows reasonable copy times for backup purposes. Uses for multiple copies or high speed are not supported, and copying an original with lots of protection errors would be frustrating. In most respects though, Clone Machine does what is says it will and can be useful. The Unguard error writer seemed to function well, it provides a simple direct way of writing errors back onto a disk. The problem will be mainly one of obsolecence as new forms of protection hit the shelves. Our biggest complaint would be the price. At $49.95 it is hard to justify owning for the value obtained.

Another good copier is Supercopy by Richvale Telecommunications. It is relatively fast and has a nice menu display. Once again, the program is locked up giving you little opportunity to modify it and keep it current with the changing forms of protection. We believe a copy program should be open so that as new forms of protection come along you can add routines to accomodate them.

There are several very new entries to the market (mid 1984) that you may wish to consider. The major features are that they provide fast copy times (4 minute average) and automatic error writing including the "current popular errors". The copy protection used by these programs does indicate their inevitable obsolescence however. In the mean time though they are certainly "state-of-the-art"! Among those we have tried or that have come well recommended are "DI-SECTOR" from Starpoint Software of Gazelle, California, "GEMINI 2.0" available from Computron Business Systems of Portland, Oregon, and "ULTRA COPY" from Ultrabyte of Dearborn, MI. There may be more good copiers hatching than there are good programs to copy!

We have included two programs which will greatly simplify and speed up copying disks with several errors. The programs are T/S Analyzer and Fastback. T/S is used to examine the disk sector by sector and log any errors found. The check is made out through track number 35. This log can be saved on another disk for later use or simply examined to see what you are up against. The primary purpose for the log is to tell Fastback which sectors to copy and which ones to skip. Any unused sector or a sector containing an error are skipped. This results in a very fast copy. The original time spent to make the error log is spread over the total number of backups that you make. Fastback need waste no time repeating the error checking on each copy as does DD-1 and most other copiers. Another note is that the logging routine which takes about 10 minutes can operate unattended unlike the copier programs themselves. After doing the Fastback you can go back with an error maker and replace the error sectors as needed, or remove the sections of the program that look for the errors.

There are several copy programs "floating" around that you may run into.We have commonly found that the whole disk copiers usually have no fast or multiple features and often are awkward in handling errors. Although DD-1 cannot solve every copy situation, we have tried to make it economical to own and easy to use. T/S Analyzer and Fastback are especially handy if you intend to make several backups. T/S Analyzer by itself is useful if you want to find out what kind of errors you are going to find on a particular disk. (See listings chapter 6)

Contrary to how it may sound, a Dumb copier is probably the most effective way to duplicate "smart" protected disks. The principle of dumb copying is that data is fed from one drive to another without going through the logic of the system. This is basically a dubbing process. The effectiveness of protection in the first place relies on the fact that your disk drive has a computer inside that decides which data from the diskette is good, which is bad, and which is ugly. The dumb copier could care less, it writes what it sees! We have not heard of any system of this type yet on the market for low end users. We are currently considering developing and marketing such a system. At this point a lot depends on the market interest.

-- OTHER TOOLS --

Superdirectory is what its name says it is. When you run into programs which cannot be copied by whole disk means and when you load them and they take off running... it's time for Superdirectory. As you know, the Load"$",8 tells you whats on the disk but it doesn't tell eveything! Superdirectory will tell you a

```
                    SUPERDIRECTORY

DISK NAME = PSIPACK V1 C(84)

TYPE    TRACK    SECTOR      NAME              BLKS      HEX.ADD     DEC.ADD

PRG      17        0         1PSIMAIN          2          0801       2049
PRG      17        1         2PSIMAIN          2          0801       2049
PRG      19        0         SUPERDIRECTORY    8          0801       2049
PRG      19        13        DISK-EDITOR       7          0801       2049
PRG      17        19        RELOCATE/LOADER   2          0801       2049
PRG      16        2         DISKPICKER        17         0801       2049
PRG      19        9         T/S ANALYZER      8          0801       2049
PRG      17        2         FASTBACK          7          0801       2049
PRG      17        15        1DUPDAC           17         C000       49152
PRG      16        7         2DUPDAC           17         C000       49152
PRG      20        2         ADMACH            1          C000       49152
PRG      20        3         MACHRELO          1          0348       840
PRG      20        4         3ROMULATOR        4          0A00       2560
PRG      20        6         2ROMULATOR        4          0A00       2560
PRG      19        8         ZMACH             1          C000       49152
PRG      20        9         ANALYMACH         1          C840       51264
DEL      20        12        MONITOR$8000      17         0101       257
PRG      19        4         ERROR ANALYZER    7          0801       2049
PRG      17        6         20 NO HEADER      1          3300       13056
PRG      17        7         21 ERASE TRK      1          3300       13056
PRG      17        16        22 NO DATA        1          3300       13056
PRG      16        0         23 DATCHKSUM      1          3300       13056
PRG      16        1         SYNC WRITER       1          3300       13056
PRG      17        8         HELP              4          0801       2049
PRG      19        6         WRITE HDR         1          3300       13056
PRG      19        5         READ HDR          1          4300       17152
PRG      19        14        COPY HDR          1          3300       13056
PRG      19        7         CON HDR           1          5300       21248
PRG      19        15        1CON HDR          1          5300       21248
PRG      16        3         LINKSTER          5          0801       2049
PRG      14        9         TMACHRELO         1          1F48       8008
PRG      14        15        TRELO             7          0801       2049
```

lot about whats on your disk so that you can decide how to handle each program and file.

On the opposite page is a sample printout of a Superdirectory listing. Note that as well as the name and number of blocks, you are also told the starting track and sector for each. Note also that DELeted programs are listed. Until they have actually been written over, they are still on the disk. This can be used for protection since one normally wouldn't even know they were present! Appendix D shows how to restore scratched programs. The most valuable feature of Superdirectory is the second section which lists each program by its starting track and sector and gives the hex and decimal equivalents of its starting address in the computer. This makes it easy to separate machine routines from basic routines and to locate autorun and other protection boot systems that load in normally below hex 0801. With this info you can often load and "pick" programs separately. Without knowing these addresses, you can load them but you don't know where to find them!

Error Analyzer has two primary functions. One is to do a quick track by track check for normal sync formatting. This is done out to track 44 so that as well as finding Erased tracks it will also tell you if anything has been put beyond the normal track ranges. This operation can be done in a few seconds and is a good idea on a new program disk to get an idea of what you are up against. The second mode is a sector by sector check which is similar but gives you a complete listing of errors by sector. Unlike T/S analyzer, this one does not make a log of errors for Fastback. The other difference is that it is a machine controlled read which operates OUTSIDE of the normal DOS. This improves its ability to "tolerate" the errors it finds without bumping and grinding the head in the process. The error listings can be printed if desired for a permanent record.

Relocate/Loader is another useful tool in "picking" programs. For example you will probably be able to load and pick any program addressed above hex 0801 but the ones starting lower than this may lock up the computer. The trick is to load them somewhere else and pick them. Once the "picking" is done, you can save them and change the address on the disk so that they load back to where they were supposed to. Somtimes rather than even picking a program, you may simply reload it so that it won't run or lock up. Then you can save the program - which defeats the purpose of the lockup anyway! The saved version of the relocated program will need two bytes changed on the disk so that it will then load back to its normal disk location and run normally. A 50K byte buffer is available for your relocated programs. All efforts were made to keep Relocate/loader itself small. There is also a tape version called Trelo which is described in the tape section of this chapter and chapter 5.

Disk-Editor is a program that will display any sector of the disk on the screen and allow you to change any bytes in that sector. This will give you the ability to change the address contained in the first track and sector of a program which tells where that program loads to in the computer. This is normally done on programs that have been saved with Relocate/loader techniques. Also you may find occasions when you would like to scramble some data on a particular sector, reactivate a deleted file or otherwise confound some location on a disk. Some other programs similar to this are called disk doctors and also have the ability to change data on the disk.

Error Makers are programs which can reproduce certain errors on a disk. This is usually accomplished by sending a machine language program to the computer in the disk which tells it to do something outside its normal operating paramaters. Often there will be compatibility problems when using these with disks other than the 1541 since the operating system programs used by other manufacturers are not identical to the 1541.

Diskpicker is a disk drive software development system which allows you to write, load, and execute programs directly in the disk drive memory. We have provided the more commom error routines which can be sent to the disk drive to write the errors as desired on a disk. The primary purpose of Diskpicker however is to give you a development system on which you can devise your own error writing, modified formating, encrypted data writing, routines and so on. As Diskpicker is designed, it lets you develop a routine such as an error writer, send it to the disk and execute it and then read the disk to see if it operated the way you wanted. Since the market will continually change with new errors and techniques for protection, Diskpicker gives you a way to develop and use new error routines as needed. Alternately, if you are not into machine programming, you may be able to find someone with an error making routine and you can then use Diskpicker to send it and execute it. Commercial error writing programs do not generally allow for this and are thus prematurely obsolete. The Diskpicker requires Monitor$8000 by Commodore to give it the monitor features although other monitors can be used by changing the auto load and monitor call locations. Other monitors should not be located at $C000 as this area is used by Diskpicker. Beyond that it is a unique program that gives you a chance to stay current in this volatile pastime.

Linkster is a simple basic program that gives you a display and printout of the tracks and sectors used by a program. It is very handy when you need to know exactly which tracks and sectors are being used by any given program.

There are some other tools which you may like to have that we have not included in our kit. Two notable tools are BAM view and Track and Sector display. These are readily available in the public domain and are furnished on the demo with your 1541 drive, thus we have made no effort to include our own version. The Bam view is nice to give you a more graphical display of what tracks and sectors have been used on a disk. The track and sector display is similar to what you get when you use our Disk-Editor. A block at a time can be displayed or printed. The data is in hex with the ASCII representation shown off to the side. This is nice as it allows you to see things like names and other "coherent" data. It is a passive program only, and unlike the Disk-Editor which allows you to change contents of these locations.

The next group of tools that you may need to use are the most powerful but also the hardest to use. They are the Editor/Assemblers. To the uninitiated, these allow you to write, view, change, save, and generally manipulate machine language programs. The secret that most programmers and pirates alike try to keep is that in order to become really proficient at breaking protected programs you need an understanding of machine language and the use of editor/assemblers. There are too many possibilities for any likelyhood of ever seeing an undefeatable protection breaker. Get an editor/assembler and start learning how to use it. You may never become a machine language programmer, but it will be a powerful tool even if your ability is limited.

There are many E/As that will do the job and feel free to use one that you like. If you have no previous experience we recommend two which Commodore sells on a disk called "Commodore 64 Macro Assembler Development System". The two programs are called "Monitor$8000" and "Monitor$C000" after the hex locations in which they reside. We recommend these partly because the entire range of memory can be accessed with these two. If a program happens to extend up into the area used by one of the assemblers, you can always use the other! Also the command structure is identical for each which makes switching quite painless. Also this assembler is readily available . You will need it to use Diskpicker. There are two major weaknesses which are nearly unforgivable in a commercial system though. The first is that they will not save the memory locations where they reside, so you can't copy them. We have included a range of memory locations in Appendix B which you should change in Monitor$8000 so that this problem can be eliminated. In this manner Monitor$8000 can be modified so that it will save any range in memory. The second problem is that page zero cannot be saved or restored from a monitor command and thus it is awkward to go back and forth between basic and assembler. We have included a utility routine "ZMACH" that can be loaded when using the monitors to do this. With these fixes you should find them quite usefull in attacking machine programs and various forms of protection.

The use of E/As and the details of machine language are complete subjects in themselves. Many good books exist on the subject and you should purchase one along with the Commodore 64 Reference guide. Two books, either of which will get you started on machine language are "Programming the 6502" by Lawrence Leventhal and "6502 Software Design" by Leo J. Scanlon. The 64 reference guide is especially important because of the Kernal explanations and the complete memory maps which can be a great aid in trying to figure out what a program is doing. The two 6502 machine language books give good expanations of machine language commands and simple examples of their use. Many of the routines that you will be disassembling involve jumps to Kernal routines along with specialized machine routines. By tracing the kernal jumps in the program you can often get a "skeleton" of it. By filling in with what the specialized machine programs are doing you can then learn what has been done.

A good calculator with Hex to Decimal conversions is invaluable to a serious machine programmer. The Sharp EL-510S is a good as well as inexpensive choice.

-- Summary --

The simplest thing to do when first trying to back up a disk program is to try to make a direct whole-disk copy. If this copy will not run as is, you can then go back to see what kind of errors showed up in duplication. If using our copier you will already have a printout of this info. If you are quite certain that the original makes heavy use of error protection, you should first use Error Analyzer to check the disk out and depending on what you find, perhaps run T/S Analyzer to get an error log. The error log should be saved on a disk reserved for this purpose. A Fastback can then be done, utilizing the error log. Next use an error maker to try to reproduce these kinds of errors in the copy. If this fails, use a program such as SuperDirectory to locate what kind of programs are on the disk and where they load to. Where possible load these programs and list or dissassemble them as appropriate for the kind of program involved. Locate the sections which look for the errors, extra tracks, etc. and take these sections out. When programs autorun preventing listing, use Relocate/Loader to load then dissassemble. For more ideas on use of these programs see the chapters which cover specific applications.

If you obtain versions of the kinds of programs we have discussed and become proficient at using them, you will be able to break many forms of protection. As with any complex job, it is an art which depends largely on aquired skill. In this book we do not hope to solve every problem for you but rather to give you a guide to follow and explain the major tools and their use. If and when software falls into line with the rest of the information market, these problems of exotic protection may dissappear.

-- Tapes --

Compared to disks and cartridges, tape protection is very easy to circumvent. The main reason for this is that the tape medium simply does not provide for any really sophisticated "lock outs". A magnetic tape recorder is a very "dumb" peripheral device compared to a disk recorder. It contains no microprocessor or logic control devices which need to be fooled or bypassed as does a disk drive. The serial nature of the tape does not allow for sophisticated file handling and the kinds of access code checking that disks may use. In fact, if a program of interest is available both on disk or tape, you may choose to purchase the tape and make your back-ups on disk! You can then enjoy the speed and versatility of the disk while avoiding the sophisticated forms of protection which the disk version may contain!

The most effective way of copying a taped program is to use the "Clone" method which we describe in chapter 5. The effectiveness of this system is due to the fact that it is an entirely "dumb copier". A dumb copier makes no attempt to interpret any data, it has no means to do so. Information coming from one datasette is rerouted via a special plug and fed into another datasette. One plays whatever is on the tape while the other records whatever is on the tape. The data on the backup is an exact match to the original complete with any protection and so on.

When using cloning for duplication you should be aware of the fact that a cloned copy is never quite as good as a "computer saved" copy. The amount of degradation is minimal and will not cause any problems if the clone is made from an original tape. If clones are made of clones, there will be a multiplication effect so that after about four or five generations the copy would be unuseable. You might think of this as sort of a problem of inbreeding. The best practice is to clone the original and use the clone, keeping the original stored safely away in case your clone is damaged. In this manner all clones will be first generation and will not exhibit mutant tendencies!

Another way to make tape copies is to try to save them directly. One noteworthy form of tape protection against this relies on an autorun routine in the program. These are loaded below hex $0801 in memory. The usual trick is to use a short autorun program which also disables the STOP key so that you cannot break and list the main program(s). Often there may be more than one program, each of which can be loaded and saved normally as long as the autorun section is not loaded. If the datasette runs and stops and runs again while loading a program, this is probably the case. The autorun section can be saved separately by using the tape version of Relocate/Loader, Trelo.

If you decide to try to copy tape programs by loading and saving, you will need to know whether they are basic or machine language and where the machine sections reside in memory. Chapter 3 details the tape buffer which can be used to extract the beginning and ending addresses of programs being loaded. In order to be effective at this kind of work we recommend that you add a "Load Data Audio" circuit to your computer. Tape programs contain a "header" section which contains the name and locations. By listening to the program load with the audio circuit, you can stop the recorder at the appropriate times and Peek at the tape buffer to find out the starting and ending locations. An editor/assembler is helpful in this also so that you can get a full hex display of the important buffer locations and also directly save the machine programs to the backup tape or disk.

This completes our discussion of tools for duplicating software. Although other tools and methods exist, this is a good sampling of the essentials. As with any craft, you will probably find yourself collecting a variety of specialized tools to solve special problems. As your skill with the simple tools increases, you will begin to understand the applications for the more advanced tools.

*** CHAPTER THREE ***

MAPS

For many applications of the computer we do not need to know much about the memory of the system. Usually we can be happy knowing that we will not run out of useable memory space. When it comes to duplicating programs however, working without a memory map is like trying to find a house in a strange city without a roadmap. "Getting into" a protected program will often require that we know something about where it normally resides in memory or how the computer is configured to run that program. This chapter will give you the maps and information about system configuration that you will need. This chapter should be used as a reference when you get into situations which require you to locate programs. Do not worry about memorizing this information, you will be better off just familiarizing yourself with what kind of information is here, then look it up when you need it.

We will cover five topics concerning memory as follows:

1. Normal configuration.
2. Software reconfiguration.
3. Hardware reconfiguration.
4. Special locations.
5. Disk memory.

NORMAL CONFIGURATION

Figure 3.1 shows the normal configuration for the C64. Note that the memory addresses are given both in hex and in decimal. In this configuration the areas shown shaded are RAM available to the user. For basic programs the memory must be contiguous. Thus the area from 2048 to 40960 (38911 bytes) is available for basic programs.

Basic ROM starts at $A000 and extends to $BFFF. That means that in normal configuration, this area is occupied by Basic and cannot be used otherwise. Another free RAM zone begins at $C000 but can only be used for machine programs or data storage kinds of jobs since it is not contiguous with the other free RAM. The area at $C000 is 4K (4096) bytes, extending to $CFFF. A basic program can poke values to the 4K RAM at $C000 thus increasing the available RAM without reconfiguring. Alternately, this RAM at $C000 could be used for machine subroutines which basic could call thus reducing the requirements for uninterupted memory space.

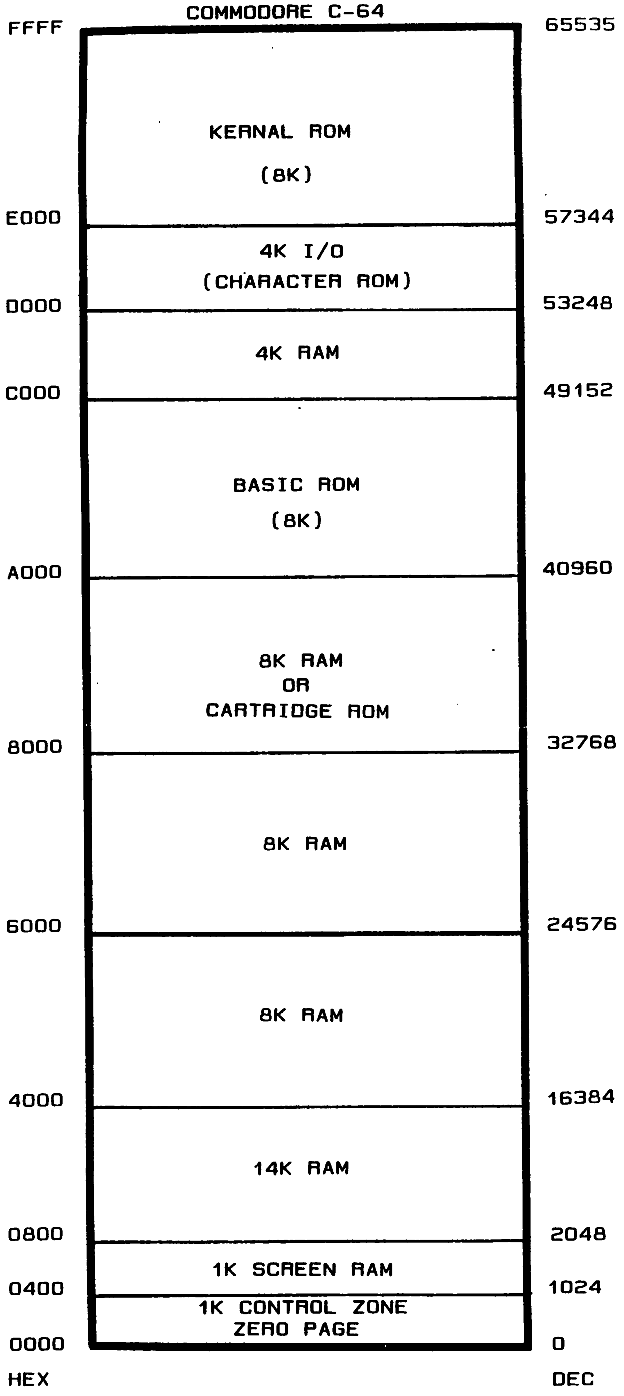
SOFTWARE RECONFIGURATION

One of the unique features of the C64 is the way its memory can be reconfigured. This is made possible because the C64 contains 64K of RAM located between hex $0000 and $FFFF. The ROM and I/O areas shown on figure 3.1 actually contain RAM "underneath" them. The 6510 microprocessor used in the C64 allows the programmer to switch memory blocks in or out. If the 6510 microprocessor chip had more than 16 address lines, the process of reconfiguring memory would not be necessary. The 16 address lines limit it to directly addressing 64K of memory. By using a special output port at location $0001, the 6510 "turns on or off" these memory blocks which are addressed "on top of each other". Actually, the ROM and I/O are normally "in", while the RAM underneath is accessed by switching the ROM "out". So Basic ROM is normally seen at $A000 and KERNAL ROM is normally seen at $E000. The I/O section hex $D000 to $DFFF has three possible memories to talk to. Normally it is I/O. By changing location 1 to a value such as decimal 51, the character ROM can be

FIG. 3.1

MEMORY MAP

COMMODORE C-64



switched into the D000 to DFFF locations. A value of decimal 48 will switch the D000 to DFFF RAM in.

In the case of BASIC and KERNAL areas, the underlying RAM can be written to at any time but can only be read if the ROM is switched out. Thus a basic program can poke values into these "hidden" RAM locations or a machine program can Store to these locations. To read the contents of this memory the ROM must be switched out by the accessing program. This means that the program that reads these locations cannot be BASIC because the BASIC ROM and/or KERNAL ROM will be "shut off" during the access time. For example, if the KERNAL area is "off" the machine routine must not access KERNAL routines during the time the hidden memory is being read. If Basic is off, the calling routine must not use any basic statements. One important note is that switching the KERNAL off also disables Basic. Also if using the I/O area, the program reading the underlying RAM must not contain any interrupts, keyboard or I/O calls, in addition to not using BASIC or KERNAL routines. Outside of these limitations, the memory reconfiguration is a good feature which allows the C64 to go far beyond the limitations of a normal 8/16 bit computer. Unfortunately, it also makes for quite a bit of confusion for many users and makes some forms of protection harder to analyze and break. In some cases, knowing the normal configuration for a program you are trying to break will be of paramount importance.

Three of the bits at location $0001 hex control the configuration of the ROM and I/O memory. Table 3.1 shows the hex, binary and decimal values used to reconfigure the port. Most often you would use a small machine routine to set the value at location $0001 for the configuration desired, then access the memory, finally resetting the value at $0001 to 37 before returning to basic or accessing normal I/O or KERNAL routines.

------ TABLE 3.1 ------

Value at Loc $0001			
HEX	BINARY	DEC	CONFIGURATION
37	110111	55	Normal (Map fig 3.1)
36	110110	54	BASIC out
35	110101	53	KERNAL & BASIC out
34	110100	52	BAS -KERN -I/O out (64K RAM)
33	110011	51	I/O out
32	110010	50	I/O & BASIC out
31	110001	49	I/O & KERNAL & BASIC out
30	110000	48	I/O-BAS -KERN out (64K RAM)

Although the values in table 3.1 are the "normal" state for the data in location $0001, there are other situations which may cause these values to appear different than the ones listed in table 3.1. The reason for this is that the value at location $0001 is the result of eight bits of binary information and only the last three of these bits actually control memory configuration. The other bits which go together to make up the value have to do with the cassette port. Simply storing one of the hex values listed in location $0001 will achieve the desired result but is not the "cleanest" in programming terms. A better method when you need to shut OFF one bit of eight, is to use the logical AND instruction. When you need to turn ON one bit without affecting the rest, the ORA (logical OR) instruction is best. Table 3.2 gives the pre-calculated values to AND or OR for changing memory configurations. Think of the AND as disabling certain ROMs while the OR (ORA machine) will reset to normal. A disable program for removing basic might look like this:

```
LDA #$FE
AND $01
STA  $01
RTS
```

A resetting routine:

```
LDA #$01
ORA $01
STA $01
RTS
```

------ TABLE 3.2 ------

Disable # (AND)		Enable # (OR)	Configuration
	HEX	HEX	
2-	FE	01	Basic out
3-	FD	02	KERN & BASIC out
4-	FC	03	KERN BAS I/O out
5-	FB	04	I/O out
6-	FA	05	I/O BAS out
7-	F9	06	I/O BAS KERN out
8-	F8	07	I/O BAS KERN out

HARDWARE RECONFIGURATION

(CARTRIDGES)

As you can see, the useable memory controlled by the addressing limitations of an 8/16 bit microprocessor, is pretty well filled up! The problem then arises of where to put cartridge programs. Leaving a certain area unused as the VIC 20 does, would mean a limitation of user RAM or a tradeoff in sophistication of the normal system. Rather than allow this, the Commodore designers included another memory reconfiguration scheme that could be controlled by the cartridge itself. The concept being that cartridge programs would occupy RAM locations starting at hex 8000 or hex A000, with a hardware system of switching out the normal memory in these locations. This still leaves a lot of RAM available if needed by the cartridge program and, in the case of A000 cartridges, uses BASIC area which is not often needed since most cartridges are machine language.

An interesting fact is that the cartridge program could again reconfigure memory once it starts operation. In many cases this would foil attempts to transfer the cartridge program to RAM inside the C64 and running it from there. Some cartridge programs can be operated from C64 memory without using the cartridge! This is not a very predictable method though as even a simple "write over" loop in the program could eliminate any possibility of running in RAM without write protection. This along with the chance that the program could reconfigure memory when it runs, convinced us that the only viable way of running cartridge programs without using the cartridge would be through the use of an external RAM with write protect capability, which essentially emulates ROM. Thus the Romulator system described in chapter seven provides a very reliable cartridge elimination scheme.

The hardware reconfiguration necessary for the cartridges involves two lines which are connected to the expansion port of your C64. These two lines are named GAME and EXROM and they are normally at logic one with no cartridge plugged in. In general, these lines are either individually or both grounded by the cartridge to control the memory configuration of the C64. Table 3.3 details the possible combinations of these lines and indicates what areas are made available. Actually there is no specific rule for exactly where in these areas a cartridge must start for any given line configuration. However unfortunate this might be, it has little effect on our ability to copy and operate the cartridge in external RAM such as the Romulator system does. All that is important is that you know the normal state of these lines with the particular cartridge in question. Chapter seven outlines a very simple non-destructive way to find out the normal starting location used by the cartridge ROM.

------ TABLE 3.3 ------

GAME	EXROM	MEMORY CONFIGURATION
1	1	NORMAL- no cartridge in.
0	1	$8000 & A000 & E000 available
1	0	Location $8000 available
0	0	BASIC out $8000 & $A000 available

Hardware reconfiguration will be needed if you are doing cartridge backups. The Romulator system provides you with the necessary hardware to accomplish this for the majority of 8K and 16K cartridges currently available.

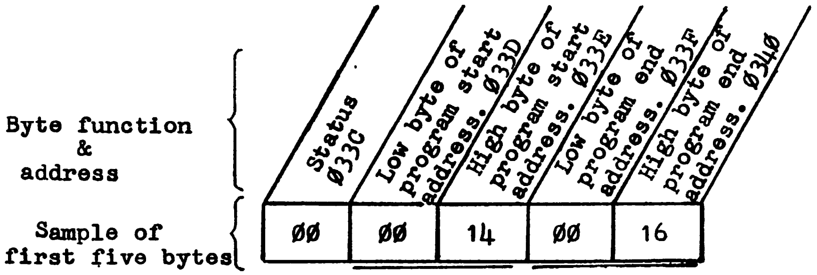
------ SPECIAL LOCATIONS ------

There are several areas in the C64 memory that are of particular concern to anyone breaking protected programs. These locations or areas of memory either contain information that you may need to duplicate a particular protected program, or they may be locations used for purposes of disabling the keyboard, auto-start routines and so on. One area of especial interest is the "control zone" which is indicated on figure 3.1. This zone which resides between $0000 and $0400 is mainly used by the operating system to store "housekeeping" data such as pointers, vectors, flags and so on. The "zero page" ($0000 to 00FF" is located in this zone. The memory maps in the Commodore 64 Programmer's Reference Guide spells out the function of every assigned location in the control zone. There are three fairly good size "free" areas within the control zone that you need to be aware of. The tape buffer $033C-$03FB is one and an unused area $02A7-$02FF is the other and $0100-$01FF the third. Although the tape buffer area and the $0100 area have other jobs, you will see them used for directly loaded routines.

The tape buffer has two characteristics of interest to us. One is that upon loading a tape header from a program, it will contain the starting and ending addresses of that program. Figure 3.2 shows the first few locations of the tape buffer and indicates what information the bytes located there contain. Note the way the starting and ending address of the program that has been loaded is determined. Second, the buffer area of the tape buffer is often used for a short control program or data storage. Programs in the buffer area can be written there from disk or tape or sometimes

FIGURE 3.2

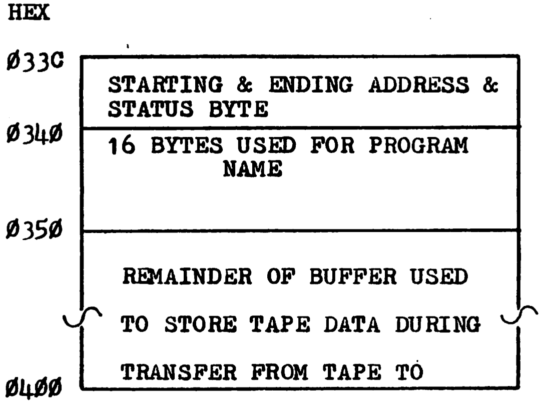
TAPE BUFFER



Byte function & address
Sample of first five bytes
Status Ø33C
Low byte of program start address. Ø33D
High byte of program start address. Ø33E
Low byte of program end address. Ø33F
High byte of program end address. Ø34Ø
ØØ
ØØ
14
ØØ
16

Identification of the first five locations of the tape buffer. (Ø33C - Ø34Ø) All values in Hex.

TAPE BUFFER MAP



HEX
Ø33C
Ø34Ø
Ø35Ø
Ø4ØØ
STARTING & ENDING ADDRESS & STATUS BYTE
16 BYTES USED FOR PROGRAM NAME
REMAINDER OF BUFFER USED TO STORE TAPE DATA DURING TRANSFER FROM TAPE TO

SAMPLE MONITOR DISPLAY OF DATA IN TAPE BUFFER

```
           ASCII                 HEX
.' 033C.....NAM 03 00 14 00 16 4E 41 4D
.' 0344E PSIDAC 45 20 50 53 49 44 41 43
.' 034C........ 00 00 00 00 00 00 00 00
.' 0354........ 00 00 00 00 00 00 00 00
.' 035C........ 00 00 00 00 00 00 00 00
.' 0364........ 00 00 00 00 00 00 00 00
.' 036C........ 00 00 00 00 00 00 00 00
```

placed there via a "poking" routine in another location. Since this area would only be affected by a tape load, it is protected from being destroyed by normal means such as resets etc. Also it is not obvious to the uninitiated and thus provides a small measure of secrecy. Routines here must be in machine language as it is not contiguous with user RAM. The size limits the extent of the program, but it is perfect for boot or protection routines and access codes.

The unused memory at $02A7-$02FF has similar applications. This is an insidious area because it borders on the BASIC vector locations. The significance of this is that if each of these five vector locations contain the starting address of a routine, any RUN/RESTORE attempt will force an automatic jump into the routine. In this manner, an attempt to break a program for listing or disassembly cannot be done from the keyboard. Most importantly though is that if these vector locations are loaded to from tape or disk, upon completion of the load the program they point to will run! This is the elusive method for AUTORUN! The trick to using autorun is that the program must be in the computer BEFORE these locations are loaded to. Since one of the purposes of autorun is to prevent listing of programs before the user RUNs them, that means that the program to be autorun must be loaded along with but before the vectors. The only area before the vectors big enough to contain a program is the $02A7 to $02FF area. Although not large this area is perfect to boot in a main program and run it. RUN/RESTORE will simply restart the program as long as the vectors are set! The Relocate/loader process in chapter six gives the procedure to get around this and save these routines or examine them. Appendix C lists an autorun routine that you can use with your own programs if you wish. Note locations specified for addresses and load options.

The zone at $0100 has sometimes been used for autorun boots. Most of the same ideas apply to it as the other areas we have already talked about. You should once again be alert for addresses shown up by Superdirectory which reside in this zone. The Relocate/loader is a perfect way to save these routines.

As usual, the possibilites for using or modifying certain locations in the control zone are limitless. Ultimately, you need the ability to find out what locations a program is using in the control zone and then analyzing the result for each particular case.

Some programs, especially basic will need to have the basic vector locations intact, that is the autostart method would interfere with the normal operation of a basic routine. When this case occurs, the autostart may be used as a loader but from the moment the basic program is accessed, the five vectors will need to be restored. There is a method which will prevent any basic program from being "Stopped" and listed which requires no special separate routines. That is to execute a POKE808,225 somewhere near the beginning of the program. This location is the KERNAL

STOP routine vector and poking 225 there prevents the computer from doing the STOP routine when the STOP key is pressed!

As a general rule you should be especially watchful of pokes or stores to any control page locations. Pay particular attention to any changes of vector addresses as they cause the computer to go to the wrong place when that routine or condition arises. This allows protectors to keep people from using "normal" means to look at their programs. For more information, you will need the 'Programmers Reference Guide or one of the other clones of this guide which fill most bookstore computer bookshelves. Probably the most fertile range to study and look for in programs are address locations $02A7 through $03FF. This range contains the five BASIC vectors and thirteen KERNAL vectors as well as the largest unused zone and the tape buffer.

The tools included in this book are just the beginning in breaking protection. The whole protection dilemma is dynamic in nature and even as we write this book, someone somewhere is bound to be devising a new and more diabolical scheme. Every time you buy another utility guaranteed to break "all forms of protection" you will eventually find something it won't work on. Information and understanding are the most important keys you can have to unlocking protection.

------ DISK MEMORY ------

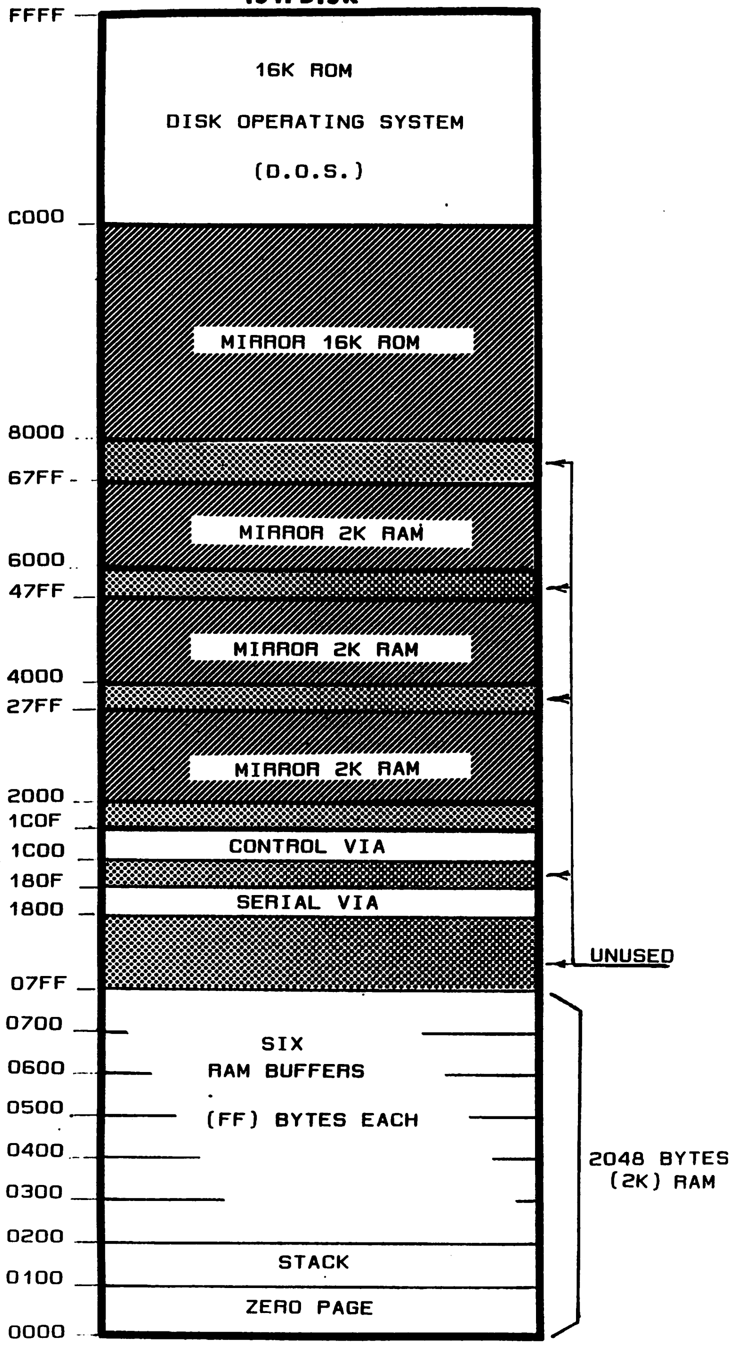
Perhaps one of the best kept secrets of the Commodore family of equipment is information concerning the disk drive. Most of the information around is a slightly fermented product of the grapevine... So and so said this... what'isname said that. The information and procedures being used with the disk range from gibberish to genius. What we have managed to distill from all this is somewhere between and although I know it is not genius, I hope you won't find it gibberish!

In this section we will give you a simplified disk map, figure 3.3 as well as point out a few of the areas of special interest. The VIAs which are used in the disk drive are detailed in figure 3.4. For diskette formatting, we will refer you to your User's Manual. Appendix F shows a GCR (group coded recording) map of a typical disk sector. You should also note the GCR header organization as it is slightly different from what the disk manual implies. The rest of this chapter will explain the areas shown on the maps in figure 3.3 and 3.4.

The ROM used in the disk is 16K and provides a unique operating system (DOS) in which the vast majority of all disk functions are accomplished by software. This is the hardware-software tradeoff which gives designers the choice of making fast and relatively expensive hardware intensive products versus slower, cheaper software intensive products. The Commodore directive was to produce an inexpensive

FIGURE 3.3

MEMORY MAP
1541 DISK



ALL ADDRESSES IN HEX
DOTTED AREA UNUSED

disk which inherently requires the use of software for as many functions as possible. Although much too long to detail here, you can get a complete disassembly of this ROM by using Diskpicker which is explained in more detail in chapter six. If you are serious about programming the disk, you should take the time to do this. Be sure to have lots of paper ready for your printer and a couple hours of time. 16K is a lot to transfer and print!

As a user, you can control the ROM only to the extent that you may use (JSR to) the routines contained there. However, those with the knowlege and equipment, could "burn" another set of ROMs with some of the routines specially modified. The purpose of this might then be to cause the disk to write in an abnormal manner which could not be reproduced by normal disk drives. If this is done in a manner that does not interfere with normal reading operations, it would be a very effective protection scheme. The users "normal" operating system could not copy or write in the same way as the manufacturers "abnormal" operating system. The only hope would be if the user, by analyzing the protection, could write a routine that could be fitted into an unused RAM buffer in the disk memory. In some cases, one could reproduce the way the disk was protected. This is one of the primary purposes of Diskpicker; to allow user machine language programs to be developed, tested, and operated in the disk RAM. The major problem in implementing such routines will be the limited RAM available. If the protection DOS routine is called by an earlier DOS routine, the users RAM routine would need both, as well as any others in the chain. The whole thing starts to become quite large. The reason is that the normal DOS is going to call routines within its address space, this does not always allow jumping out to RAM routines then back and so on. Only a modified ROM could provide this capability. There are still many routines you can use parts of from RAM calls however. As chapter six will show you, our error writing routines involve exactly this kind of process.

The ROM area from $C000 to $F24C serves mostly for the software oriented tasks of interpreting, manipulating data and so on. This you will probably not have a lot of reason to modify. The ROM from $F24D to $FFFF contains the hardware control routines. These routines control such things as selecting tracks and sectors, starting motors, selecting reading and writing and so on. Many of the error writing jobs can be accomplished by using these routines or modified versions of them. A source code listing, which you may be able to locate through a user group, can be an invaluable aid in using these routines.

The RAM used by the disk is 2K in size. The RAM provides the zero page ($0000-$00FF) which is required by 6502 based systems. Two pages $0100-$02FF are reserved for additional pointers, stack requirements, and so on. $025B-$02B0 which is in this area is used for RAM array and takes care of file handling. $02bl-

$02FF is for output buffer information such as error codes and directory. The rest of the RAM area is divided into 256 byte "buffers". There are five of these buffers which have the primary job of holding the data comming from and going to the disk. They are allocated as needed by the DOS. User programs can be located in these buffers and called by the user commands. The first 18 bytes of the buffer at $0500 are often used for a jump table to user routines.

By far the most important area of these RAM locations is the $0000-$0005 which is the job queue, and $0006-$0012 which provides the respective headers (track and sector) for the job in the queue. Note that there are two RAM locations for each job location.

In addition to the ROM and RAM, there are also two Versatile Interface adapters which are seen by the disk CPU as memory locations. By nature, a VIA occupies only 16 memory locations. Figure 3.4 shows what is found at each location of a VIA. The first VIA is normally "seen" at $1800-$180F. Its primary job is to control the serial bus. It is connected directly to the serial bus and has the job of taking data from the internal data bus and sending it out on the serial bus and vice versa. The second VIA is normally "seen" at $1C00-$1C0F and its job is to serve as hardware controller. It is connected to the circuits which drive the motors, sense write protect, and controls the read/write logic for the head. The hardware signals are transmitted to and from the VIA directly from the internal disk data

FIGURE 3.4
1541 DISK VIA MAP

00	I/O REGISTER B
01	I/O REGISTER A (WITH HANDSHAKE)
02	DATA DIRECTION REGISTER B (DDR-B)
03	DATA DIRECTION REGISTER A (DDR-A)
04	TIMER ONE LOW BYTE (CLRS INT ON READ)
05	TIMER ONE HIGH BYTE (CLRS INT ON WRITE)
06	TIMER ONE LOW BYTE TO LOAD
07	TIMER ONE HIGH BYTE TO LOAD
08	TIMER TWO LOW BYTE
09	TIMER TWO HIGH BYTE
0A	SHIFT REGISTER
0B	AUXILIARY CONTROL REGISTER (ACR)
0C	PERIPHERAL CONTROL REGISTER (PCR)
0D	INTERRUPT FLAG REGISTER (IFR)
0E	INTERRUPT ENABLE REGISTER (IER)
0F	I/O REGISTER A (WITHOUT HANDSHAKE)

The 1541 drive uses two VIAs. One is primarily used in communicating with the serial bus while the other is primarily used for hardware control. The VIA occupies 16 memory locations (00-0F). The specific function desired can be accessed at one of these locations as indicated. Note that either VIA can be READ from or WRITTEN to, thus a port would be an input if it is being READ and an output if it is being WRITTEN to. This is taken care of with the normal READ/WRITE control line from the microprocessor.

bus. As you can see from figure 3.4, among the other functions, the VIAs have directly addressable timers which are used as needed by the DOS.

One interesting fact about the 1541 drive is the "mirroring" of memory that is shown on the map figure 3.3. As shown, the devices which make up the disk memory can be seen at more than one location in the map. The ROM at two 16K areas, the RAM at four 2K locations, and each VIA at 256 16 byte locations (not shown). The reason for this is the way the disk hardware decodes memory. It was simply cheaper and easier to decode only the necessary address lines to place the memory in its designed locations. When only some of the address lines are decoded to define a memory block, that block will be "mirrored" at every other location defined by those lines. In order to get unique positions in memory, all address lines must be decoded. Mirroring causes no problems so long as two decoded blocks do not overlap each other. It can produce some confusion though if you are not aware of it and you "discover" what looks like important data at say $2003. What you would be seeing is actually the zero page data at $0003! A programmer could confuse those studying his code by accessing the same data with different addresses.

Other sources for information on the disk include the 1541 Maintenance Manual by Michael Peltier, which covers the system hardware if you need to make repairs. You will not find it much help for software applications though. The User's Manual that comes with the drive should be refered to for diagrams on how the diskette is configured with tracks and sectors. However note appendix F for the correct header format map. One final interesting note is that the drive can be forced beyond track 35 by software control. Chapter six covers this in detail.

*** CHAPTER FOUR ***

PROTECTION CONCEPTS

Software protection concepts have evolved from relatively simple schemes to very complex ones. In the rush to find ways to keep users out of programs, methods have been developed at an incredible pace. Each new form of protection then spawns a whole new set of breaker programs and copying techniques. The results of this vicious cycle takes its toll on programmers AND users. If you have been playing in this game for even a short time you have perhaps already bought some breaker products only to find that some new form of protection foils the methods it uses. So once again you are in the market for the latest breaker system and so on and so on.... We cannot offer a total solution to this nor I doubt, can anyone else. The ultimate answer for disks will probably take the form of a mechanical "dumb copier" which will not interpret any data but simply read and write an exact copy. Although it still wouldn't be a guarantee against certain forms of protection. This chapter will cover a variety of the techniques currently being used. Chapters five six and seven will then list specific procedures. Your ability to stay current with the ever changing techniques depends on your practice and motivation. We will start off with information about disk then go on to cartridge and finally tape protection methods.

There are really two things to consider when breaking protection. One is; do you just need a copy?, and two is do you need to modify the program (code) to customize it for your application? Where applicable we will try to include information to help you get listings of protected programs. Often you can learn quite a bit by studying listings of protected programs. In many cases, modifying the program so that the protection no longer exists will give you a much more valuable commodity. With your modified version, you won't have to play all the protection games if you need an additional backup copy!

------ DISK PROTECTION ------

Chapter two defined some arbitrary levels of protection. Here we will expand on this information giving you more detail on the protection in contrast with our earlier focus on tools available for breaking the various levels.

LEVEL 0.... Although this level is "unprotected" there are a few points that a beginner should be aware of. If the program is in Basic, it may involve more than one program which the user would need to copy separately. Since these are often chain loaded, it is important not to run the program. It should be loaded, saved to the backup then perhaps run to see if it loads any other programs. In a similar fashion, the program might access data files which would not be saved by the save command. Often the program might be in machine language and a simple save will not work since that is for basic only. Even though the machine routines are "unprotected", you may need to use something like Superdirectory to find the address range of the program so that you can save it with an Editor/Assembler. The E/A's require that you know the beginning and ending address of the program to save it. In general, with unprotected programs, if you wish to copy by Saving you need to know the details of what kind of program it is and what all it needs to operate correctly. The simplest method of duplicating such programs is to use a BAM type copier. Virtually any whole disk copier will also work.

LEVEL 1.... Programs in this category use some sort of autorun feature which prevents the user from making listings, and directly Saving the program. The added use of a STOP disable poke keeps the user from simply pressing the stop key to see whats in the program. On programs that ask for data from the keyboard or use a peripheral device, you can sometimes break into them by giving the computer some value out of range or unplugging the device the program is talking to. The idea is to force some kind of error which will return control to the user. At that point listings etc can be done. Usually the program will intercept the error codes and return control to the program, so you will need to be devious in the kinds of things you try. We were able to break into a well protected word processor and generate complete listings by unplugging the cassette connector while the program was writing a file. Other ways to get into such programs involve more coherent means such as relocate/loading which prevents autoruns as the data does not go into the vector table that the programmer wanted. Then by saving the program without the autorun or with a modified autorun, the disk can be edited so that when the copy is loaded it goes to the right place but no longer has the protection! The complete procedure for this is in chapter six. Once again if making a backup is your only concern and you do not need to get into the code, you can use BAM copiers or whole disk copiers. The advantage of relocate/loading is that in essence it gives you a new unprotected program which you can list and modify at will. The BAM and whole disk copiers will give you a "clone" of the original which will still contain the autorun and non-STOP features as originally written. The final choice depends on your needs.

LEVEL 2.... has been defined as including those programs which use disk errors to stop whole disk duplication. In this case, the errors will only stop duplicators which do not reset or "handle" errors. Although "old" as the protection methods go, there are several "whole disk" duplicator programs which will stop if the disk indicates that there is an error on the disk. The kind of error could be one of many possibilities but generally falls into the category of tampered headers or data on the disk. Regardless of the specific error, the duplicator stops because the disk drive tells the computer in essence that the diskette is defective. This of course is exactly what the protectors want to happen so that you will not get a useable copy. Most of the more recent entries on the duplicator market can handle this kind of error situation. There are various possibilities but the concept is to "skip over" any sector in which the disk drive sees an error. Some such as DD-1 will go ahead and transmit the data on the sector in question in many cases. Essentially it just ignores the errors that the disk drive says are there, and goes on to the next sector. This level of protection can be broken by error tolerant duplicators. Alternately, the relocate/load method can be used on this protection level. This level of protection usually includes the autorun and non-STOP features. Otherwise there would be nothing stopping the user from simply loading and saving the program. The program itself has no relation to the errors, they are just inserted on a blank track or sector to disable copiers which don't have error tolerant routines.

LEVEL 3.... This level is more typical of much current software. In this elaborate scheme, disks will have errors and autoruns as described earlier but to complicate matters, the program or a loader will check to see if the errors are present at the right locations on the disk. If the errors are not there, the program will not run, or it will crash. This is called error checking and if a program employs error checking you will need to either get rid of the error checking routines or you will need to put the same errors back on the backup disk as the original had (and in the correct places). One popular place to put the error checking routines is in a loader that runs after the autorun boot has started things up. The order is something like this: A very small autorun routine loads to the control zone resetting vectors and forcing itself to run. This autorun then boots in a loader program which is more sophisticated and handles the loading and error checking for the main disk program(s). If the loader doesn't find the proper errors in the correct locations it will not continue to load the main program or will cause a crash etc. In addition to checking for the errors, the loader program may be responsible for actually deciding which tracks and sectors to load and in what order. This way a main program can be saved and have its BAM erased. As long as the person who writes the loader knows which order the tracks and sectors need to be loaded. Interspersed amongst the valid data loads the loader can check predefined tracks and sectors for specific errors. By using a table of tracks to load and ones to error check, a very high degree of protection is obtained. Our disk analyzer is helpful both in making protected programs and backing up ones that have been protected in this manner. It will give you a sector by sector list of which ones contain data and which ones contain errors as well as what kind of error. When the main program has finally loaded correctly, it will run taking control of the system preventing breaks or normal methods of stopping. The main program may or may not perform further checks for errors on tracks and sectors depending on how it is designed. The simplest approach for these programs is to use an error analyzing routine and whole disk copier and then use an error writer to make errors on the backup - exactly like they were on the original. The major problem with this is keeping up to date on the errors that you can write onto your backup disk. As the state of the art progresses, you find that you need to be able to write an increasingly large number of different kinds of errors. The other method which is harder is to relocate/load and disassemble the loader routines and locate error checking sections. Again, the disk analyzer printout will help because it will tell you which sectors contain errors. Upon inspection of the loader you will know which sectors it is expecting to find errors in and spotting that data is easier. With the "Hunt" command of an editor/assembler you can systematically search through large amounts of code to locate occurances of likely command sequences. Practice and experience are the key to doing this successfully. In general you will be looking for a sequence of CMP #$XX (XX= hex error code value) followed by a BEQ or maybe BNE**.

The error writing process itself involes a fairly cumbersome system of writing special routines which are loaded into the disk drive RAM and executed by the disk drive microprocessor. The routines are basically mutant versions of the disks own formatting and data writing routines. In other words, if you have just saved a program which you wish to protect by these means (or made a backup which needs them), by running one of these mutated routines, you can cause something like one sector header to be formatted differently than the normal pattern for that disk. Other possibilities include erasing an entire track of data or completely wiping out a sector.

The level three will at some point check to see if errors are present before a run will be allowed. Alternately the errors might be checked within the program.

LEVEL 4.... This in many ways is just an extension of Level 3 but there is a sly twist. The errors involved in this level of protection include with the others, the trick of putting data beyond the normal 35 tracks defined in the original drive design, or writing data where none would be expected and can't be recovered by the DOS read functions. Both the extra tracks and the "hidden" data require modified DOS routines to be loaded. These DOS routines are machine programs that operate similar to the normal ones in the disk but they are designed to find the hidden data whereas the normal routines would simply return an error. The "extra track" protection could involve simply writing sync beyond track 35 or even go as far as putting data out there. There are other unexpected places to "hide" data such as between the last and first sector of any given track. This area is normally defined as a "gap", but data can be written and read here using modified DOS routines. Another trick is to erase all or part of a track and then to write data in the erased part. This data will not have the usual headers, so it must be read with a special routine. One way to find such data is to find the last valid sector using an error analyzer, then begin reading the data found AFTER this block. In this manner you can recover a couple of sectors worth of the hidden data. This process can be repeated with a change in the timing so that you see more and more of the data. Often a couple blocks worth is all you need. The diskpicker HDR read routine is useful for this purpose. Changing the sector number to an illegal value has a similar effect to these others. The disk may only expect to find perhaps 20 sectors on a given track but the protected disk will have some sectors with numbers higher than this. To recover this data, you would again go to a header reading routine. In essence, you will get the data First, then save it under a valid number. The last thing would be to change the copy header so that it looked like the original. Errors can be "spliced" into a copy disk by reading the bad part off the original and writing it verbatim to the copy. (See Diskpicker Procedure)

Once again though, in these processes the drive is being manipulated by mutant programs to perform these "unnatural acts". We discourage both the use and purchase of software with this form of protection as there are some valid questions as to the effects this can have on a drive. First of all, there are many 1541s, especially the earlier models, which seem to get out of align if the head is forced to "bump" excessively. (Bumping is when the head goes to the end and bumps several times) These programs only aggravate this situation as you may have already discovered. A good quality drive mechanism should be able to handle this without problems but this problem does exist and will occur on some drives. You will just have to live with it if you decide to purchase a program with such protection and it adversely affects your drive. In the least these programs will usually take their time loading with a undue share of bumping, chattering, blinking, and various other assorted paranormal responses.

Another dilemma with this level of protection is that it may not be compatible with drives of other manufacturers. How a drive loads and saves data is a function of the hardware and software design of the drive. When a drive is as software dependant as

Commodore's are, a 100% compatible drive would have to have the disk operating system copied verbatim. Although this has been done before with some computers, it is legally questionable. It is generally accepted that totally compatible means 99% compatible while most "compatible" equipment really comes in at a much lower percentage. What is reasonable to expect is that a compatible disk drive can save and load it's own programs and load commercial software that is recorded WITHIN ORIGINAL EQUIPMENT MANUFACTURERS SPECIFICATIONS. With level 4 software the protection is not within the design parameters of the drive so it is anybodies guess what it will and what it won't work on. It is something like an oil company putting in a additive to gasoline which rots out engine gasket material without ever bothering to check it out with car manufacturers! Thus programs, error writers and copiers that work at these levels will in general be uncompatible with drives other than the 1541. You simply cannot expect equipment manufacturers to be able to forsee the various ways programmers might try to "defile the DOS".

Making duplicates of this kind of software, if necessary, can be done like level three. The error maker will need to have the capability of writing data or errors beyond the normal tracks. The best approach for this level would be to remove the error checking from the loader or main program so that you won't have to worry about all the error checking problems every time you load the program. This is done virtually the same as the others, with relocate loads and dissassembly.

LEVEL 5.... This may be the most effective protection method in current use as well as the most valid. It works on the principle of a hardware "key" which is usually plugged in to the joystick port. As the program runs it can periodically check to see if the key is in and crash itself if it is not! The hardware key is effective for several reasons but the most noteworthy is that most people have no means of duplicating the piece of hardware. Even if they could it would be a time consuming task to first figure out what the key contains and then make a similar one. Removing the sections of the program that check for the key would be the best approach but certainly not easy. As opposed to the error checking which requires the disk to run and consume quite a bit of time, the hardware key can be validated in a matter of microseconds and has little or no degrading effect on the program. This is why error checking routines are often in a loader, so that they do not steal time from the main program. Many programs would be of little use if the protection system had to run the disk for a few seconds every so often in the middle of what you were trying to do. The hardware key does not suffer these limitations. It is possible to check for the valid key literally hundreds of times within the main body of the program, without noticeable time lag! If the checking is done during an interrupt cycle, the time taken is almost unmeasurable. Appendix H shows how an interrupt routine can be easily accessed from a main program.

The key itself could be as simple as a plug with certain pins shorted to ground which the computer would see as a specific value anytime it looked at the joystick port. Using just the five joystick switches allows 31 possible code values that could be hardwired into a small plug. This will be defined as a passive key. You can easily make one of these for your own protected programs! This would not be too hard to defeat though if the user realized that it was just a hardwired setup.

In more elaborate schemes, the protector can use a ROM chip which is addressed by a code sent on the lines and responds with a different value for each code. The circuit could send pulses of a certain frequency or in a certain pattern as the code. If desired the protector could use a complete microprocessor on a chip to respond with a very complex set of passkeys. There is simply no imagineable limit to what could be done.

Unfortunately we can offer no simple solution to the hardware key systems. If the key is the only protection, and the disk itself can be easily backed up, then you have little reason to worry. Your requirement for backup ability has been met while still protecting the seller from wanton distribution. We feel that it would be unethical for us to encourage reproduction of these keys. Thus in this respect, we feel key protection is valid as long as you can back up the magnetic medium.

One final word about compiled basic programs. The compiled program has been transformed into it's machine language equivalent, which among other things, makes it harder to analyze. Using protection on compiled programs makes a very effective means of protection.

In the future we hope to see prices of software come into line with value. New forms of protection are on the horizon but along with them comes the questions of compatibilty and effectivness against inspired pirating.

------ CARTRIDGE PROTECTION ------

Compared to disks, cartridges are quite simple. However due to the hardware necessary to make and use backups, the procedures are less well known. The real heart of cartridge protection is the autostart feature of the cartridge. Since it is running all the time, how do you disassemble or save it? The answer is just about as simple; do not let it autostart!

Understanding an doing this depends on realizing that the only time the cartridge will autostart is when the computer is first turned on or the coldstart or reset is initiated. If we disconnect certain lines on the cartridge and then turn on the computer the computer will not know the cartridge is there so the autostart will not occur. We can then switch the cartridge into certain areas of the computer and "look" at the contents or save them to tape or disk. An external RAM system which emulates ROM through the use of a write protect can later be loaded with the original cartridge code, switched into the correct location, then run as though it were the cartridge itself!

The most difficult obstacle to overcome with cartridges is the duplication of the hardware memory configuration used by the original. (See Chap. 3 Hardware Reconfiguration) Although the C64 is "filled" up with memory, the cartridges can be "switched" in place of some other memory device. Even though this is not protection in the true sense, it has the same effect. The Romulator system (Chap. 7) gives you a procedure which does not require you to know specifically "where" the cartridge is seen at. All that is required is that you determine the configuration of the GAME and EXROM lines which ultimately control what the computer does with memory and start up routines. We have encountered no other forms of protection with cartridges.

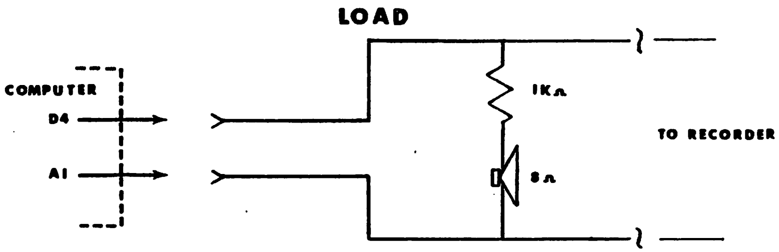
------ TAPE PROTECTION ------

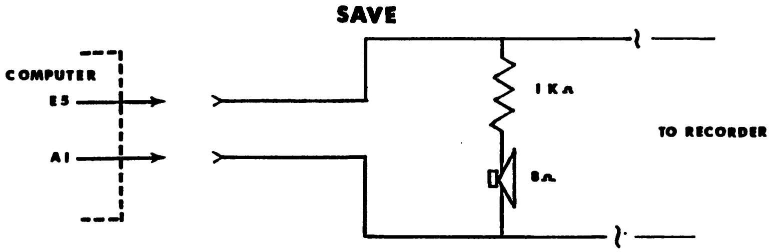
Tapes have severe limitations in the kinds of protection available compared to disks. As with the others, the autorun can and usually is used. The locations and method is really the same as the disk except that the autorun routine will boot in from the tape instead of the disk.

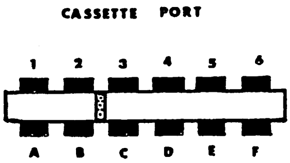
In the next chapter we will cover the clone system which will duplicate all tapes regardless of protection

FIGURE 4.1

LOAD DATA AUDIO & SAVE DATA AUDIO







PIN #	FUNCTION
A-1	Ground
B-2	+5 Volts
C-3	Cassette Motor
D-4	Cassette READ
E-5	Cassette WRITE
F-6	Cassette Switch

The LOAD data audio circuit provides an audio output during LOAD operations. This is useful in determining characteristics of pre-recorded program tapes. It also provides a simple way to align the tape head by "ear". (Chapter five - Head alignment procedure). Installation can be in the computer or on lines D-4 and A-1 where they enter the datasette. If you are using a Tapeworm, or similar interface, parts can be mounted on the interface unit itself. (Use earphone for speaker)

The SAVE data audio circuit is primarily for "Relocate Loading" which is detailed in chapter five. You may choose to wire two aligator clips to an earphone with a 1K ohm series resistor. In this manner, you can simply clip it across E-5 and A-1 when you are performing header changes . The SAVE data audio circuit provides audio only during the time that the computer is saving data to tape.

used. As with disks, often a tape program will load in several sections which each set up certain parameters of the program and all are needed to run. If a save is attempted after loading the program only part would be saved and it would be useless. To duplicate this by other than cloneing requires that you know when each section starts and stops so that you can stop the tape and put in you blank to make a copy. Also, if you try this, be alert for machine sections mixed between basic sections. You will need to use an editor/assembler for the machine sections.

Figure 4.1 shows the connections for an audio output which greatly helps in identifying separate load sections. A taped program consists of a tone leader followed by a short data burst (header) then another tone leader followed by a longer data burst which is the program. If you hope to copy taped programs by saving, you will find the circuits very helpful.

The one protection that tapes can use is the hardware key. However there is nothing to prevent you from making backup tapes, it will just keep you from distributing them or running on more than one computer. As with disks, this is not a real problem then for your own use. The one drawback is the reliance on the key itself, if it fries, you are out of business!

*** CHAPTER FIVE ***

TAPES

The very size of the C64 memory allows programmers to write some very powerful programs. As these programs require large amounts of data, they also tend to be quite slow if loaded from a datasette. As a result, tapes are not as popular as with the Vic 20. However, even if you .primarily use a disk drive, it is still handy to have a tape drive available. You may encounter a taped version of a program you wish to have. The cost of a datasette may prevent you from wanting to own one for this somewhat rare function. In answer to this we have developed a simple device which lets you avoid the expense of a datasette. Chapter five features TAPEWORM (tm) which is an inexpensive interface for standard recorders. If you do not own a datasette this circuit will allow you to add a tape drive for very little expense. If you already own a datasette and wish to "clone" tapes using audio type duplication, the Tapeworm will provide you with the means to add another drive as required for dubbing purposes. Another of the major differences is that if you wish to do audio work such as message playback or time lapse recording etc. the inherent Tapeworm cassette motor interface will make this possible. Audio work cannot be done at all with the datasette since it utilizes digital signal processing. Tapeworm makes no change to an audio recorder circuits. It is entirely an external device which turns digital computer data into audio signals for the recorder and the opposite function of converting the audio output of the tape recorder to a digital level signal for the computer. The theory of operation will give more technical detail about these aspects for those so inclined.

This chapter also features CLONEPLUG (tm) which is a simple plug that allows audio dubbing of digital tapes. In essence, Cloneplug is a dumb copier so virtually any tape can be duplicated. Although some have had success with doing this with two audio recorders, it is very unlikely that purely audio clones will work very well. The amount of signal degradation is severe if the data is not converted to digital format. An audio clone is even worse in successive generations which have been cloned from a previous clone. The clone method we will show you relies on digital signals or audio-conditioned digital signals so that cloning can be done over several generations without the inbreeding problems.

------ TAPEWORM ------

DESCRIPTION

TAPEWORM provides the proper interface circuitry between the Vic 20 and C64 computers and most standard cassette tape recorders.

FEATURES

- The TAPEWORM is an inexpensive and reliable alternative to the purchase of single use cassette data recorders such as the datasette.

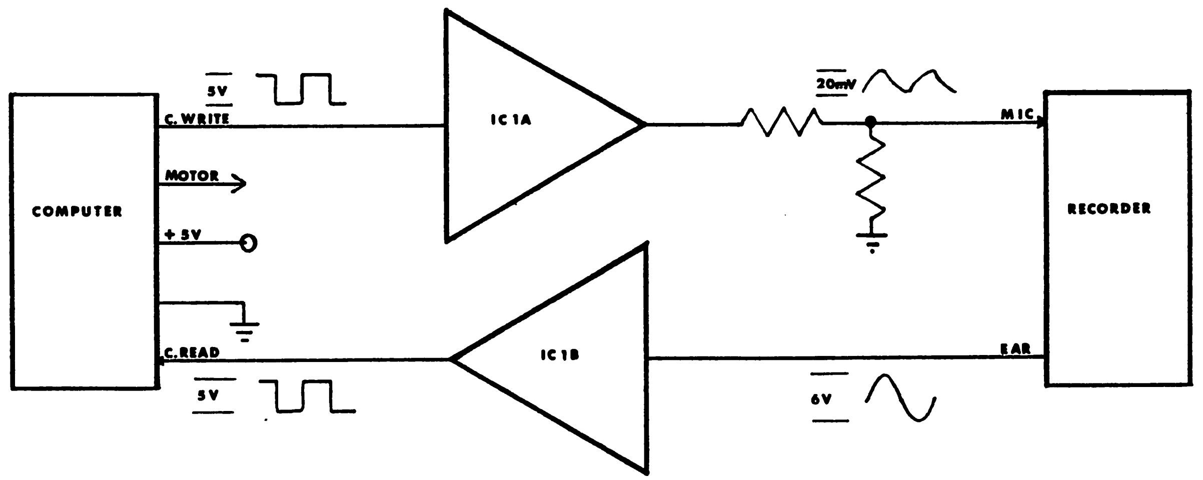
- When not being used with the computer, your cassette recorder can be used for normal recording applications. No changes or modifications need be made to your recorder.

- TAPEWORM allows the computer to control the cassette recorder to play and record voice/sound information under program command; i.e., telephone answering... security monitoring... slide show sound... time lapse recording... etc.

- TAPEWORM allows manual adjustment of the volume output level of the cassette recorder so that you have the

TAPEWORM BLOCK DIAGRAM

FIGURE 5.1



Computer output and input must be 5V square waves. When recording, the Tapeworm conditions the computer signal to feed the MIC input on the recorder. When playing, IC1B circuitry restores the digital level to the EAR signal from the cassette.

ability to compensate for tape quality variations.

- With an optional modification, the data can be heard during the load operations to aid in analyzing protection methods.
- A.C. adapters are not needed. The cassette and TAPEWORM obtain all power from the computer just like a datasette does.
- Used with CLONEPLUG, TAPEWORM facilitates tape duplication using standard recorders and or one datasette and one standard recorder. (fig. 5.7)
- Simple to hook up. Ear, Mic, an power plugs provide all cassette interface connections without modifying recorder.
- Recommended tape recorder: SANYO SLIM 1 or SLIM 2. Other tape recorders may work with TAPEWORM. Variations in record levels, fidelity, 6VDC connector polarity, voltages, etc. between manufacturers requires some technical discretion before making cassette recorder substitution.

TAPEWORM THEORY OF OPERATION

Refer to figure 5.1 for the block diagram showing function of Tapeworm. Note the cassette write and cassette read signals drawn at the left of the diagram by the computer block. As shown, both are five volt square waves. This is the normal digital signal the computer expects to "see". Cassette recorders on the other hand, do not like five volt square waves. They are much better suited to smoother waveforms like the sinewave shown at lower right. Furthermore, a microphone input on a cassette recorder expects to see a 10mV to 20mV (0.020V) signal, NOT 5 volts!

To accomplish this, the signal from the computer is fed through the input circuitry consisting of IC1A and the 1K/100 ohm voltage divider. IC1A functions as an integrator which rounds off the 5V square waves from the computer. The voltage divider then reduces the voltage level seen at the Mic input of the cassette deck to about 20mV. The sketches of the signals in figure 5.1 show the approximate shaping taking place.

For loading programs to the computer, the output of the tape recorder is a sinewave of about 6 volts. The output circuitry consisting of IC1B and Q1 must provide a clean squarewave of 5 volts to the computer. IC1B is designed as a high gain clipping amplifier and Q1 provides a fast risetime 5 volt square wave of the correct polarity. This is fed into the cassette read line of the cassette port on the computer.

As you can see from the schematic figure 5.2, the cassette switch line is always grounded. This causes the computer to always think the buttons of the cassette are depressed so that the computer will not print the "Press Play...." messages. This is done to eliminate the need for wiring inside the cassette deck to the switch which closes when the buttons are depressed. If you are technically able to determine the wiring on your particular deck and wish to do so, the circuit board has provision for this. Simply cut the ground link on line F6 and wire it to the switch in your deck. As this is awkward to do, we do not recommend it.

The Tapeworm board obtains operating power from the computer via the 5VDC output on pin B2. Six volt DC power is fed directly from the computer pin C3 to your cassette deck. There is enough power at this output to easily drive most modern 6VDC recorders. You should be careful of polarity for your recorder. (see note below)

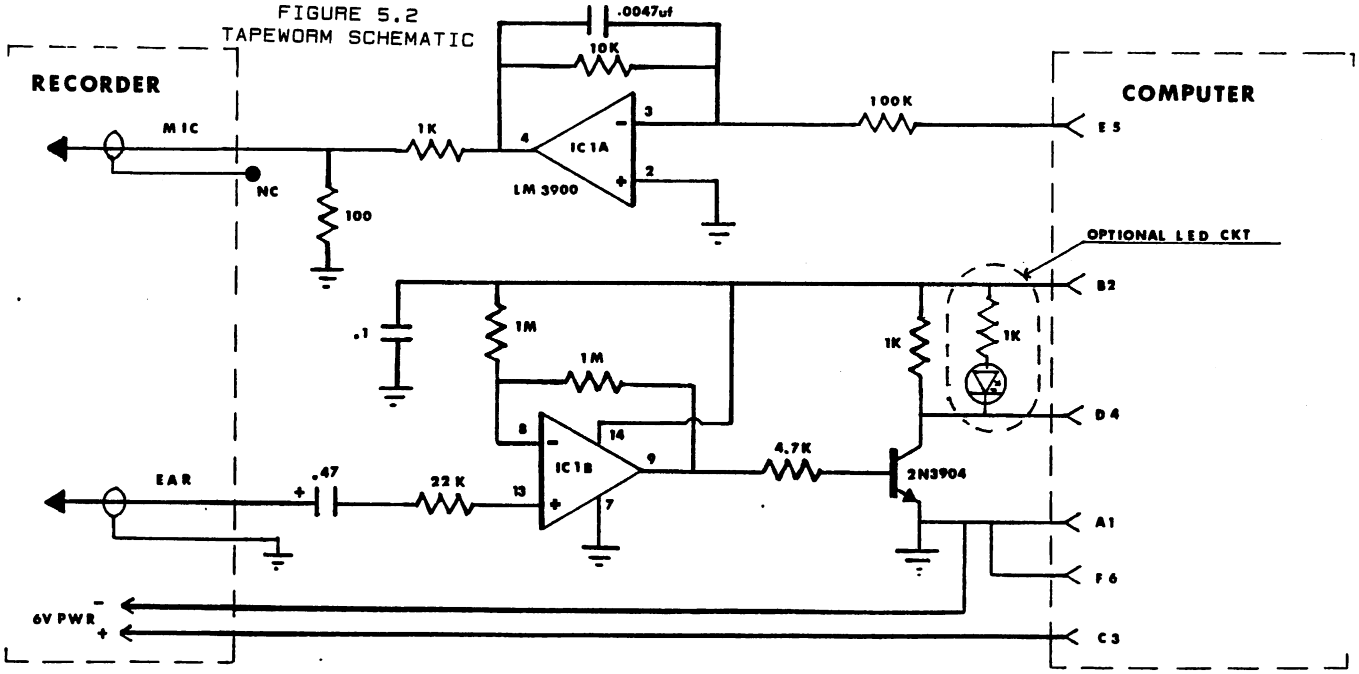
ASSEMBLY

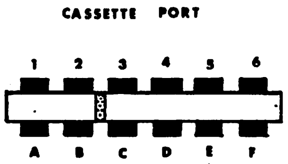
Figures 5.2-5.4 give the schematics and layouts for Tapeworm. Appendix D lists kits available from PSIDAC or you may use your own resources.

Before installing any parts, the circuit board should be cleaned with alcohol and scrubbing pad so that the copper is bright and shiny. All parts are installed from the BLANK side of the board with the leads protruding through the holes on the copper FOIL side of the board. Soldering should be done with radio grade rosin core solder and a small, clean soldering iron of 25 to 40 watts maximum. Be especially careful to orient the IC correctly. Use the pin 1 dot or notched end and the component layout for reference.

The edge connector is soldered directly to the PC

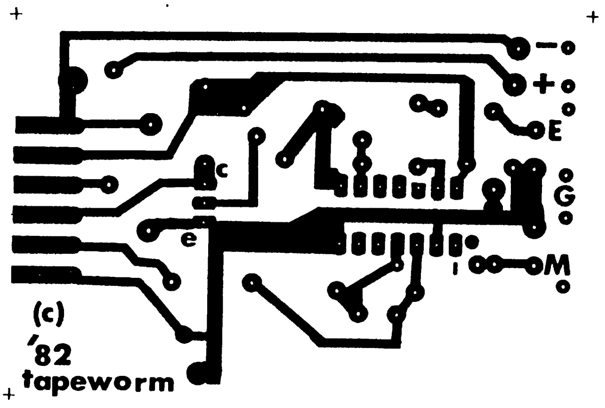
FIGURE 5.2
TAPEWORM SCHEMATIC





PIN #	FUNCTION
A-1	Ground
B-2	+5 Volts
C-3	Cassette Motor
D-4	Cassette READ
E-5	Cassette WRITE
F-6	Cassette Switch

FIGURE 5.3
PRINTED CIRCUIT LAYOUT

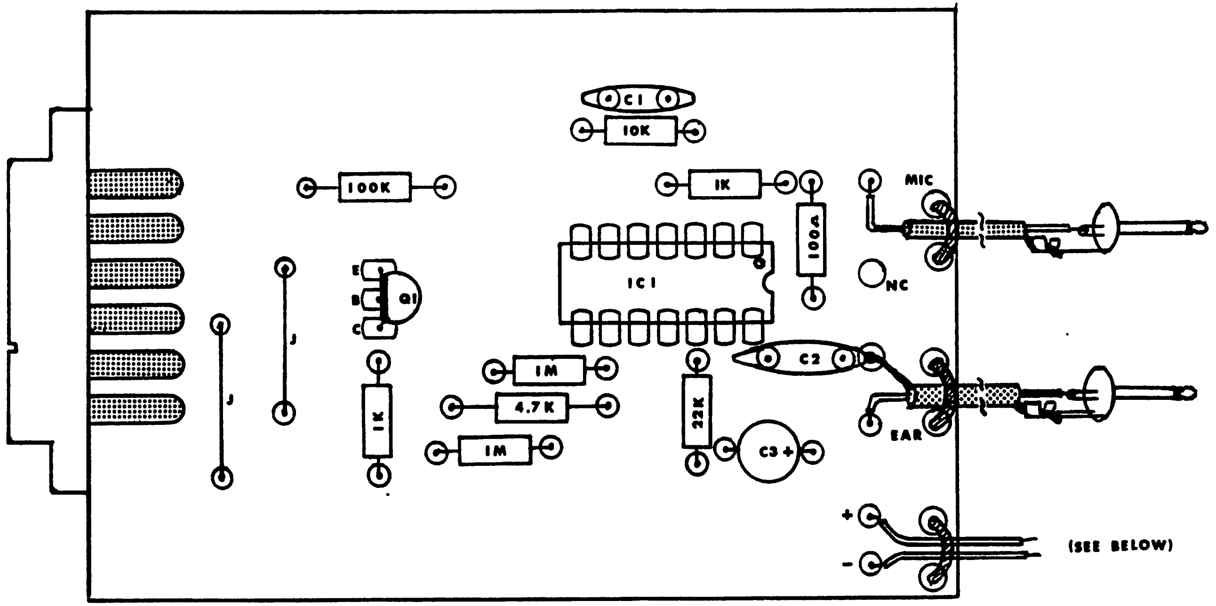


1:1 POSITIVE PC LAYOUT (FOIL SIDE SHOWN)

PT#	QTY	DESCRIPTION	RADIO SHACK EQUIV.
C1	1	.0047uf Disc Cap. 12V	272-130
C2	1	.1uf Disc Cap. 12V	272-135
C3	1	.47uf Electrolytic 16V	272-1417
IC1	1	LM3900 Quad OP Amp	276-1713
Q1	1	2N3904 NPN Transistor	276-2016
R1	1	100 ohm Resistor	271-1311
R2-3	2	1K ohm Resistor	271-1321
R4	1	4.7K ohm Resistor	271-1330
R5	1	10K ohm Resistor	271-1335
R6	1	22K ohm Resistor	271-1339
R7	1	100K ohm Resistor	271-1347
R8-9	2	1M ohm Resistor	271-1356
	(All resistors 1/4 Watt)		
EC1	1	6 pin .156" edge connector	PSIDAC #CONN*
P1-2	2	1/8" Mini phone plug	274-286
P3	1	DC plug (to match recrdr)	274-1551

Misc – Wire ties, solder, Mic. wire, etc. –

* For complete kit, see Appendix G-Price List.



NOTE: FOIL IS ON THE OPPOSITE SIDE OF PC BOARD.

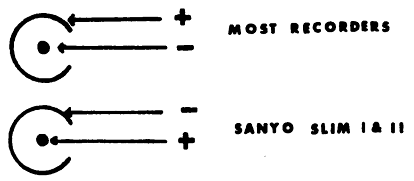


FIGURE 5.4
COMPONENT LAYOUT

★ DETERMINE CORRECT POLARITY ON YOUR RECORDER board traces. You should first make a small solder "puddle" on each of the end foil traces which are for the edge connector. By holding the edge connector in alignment and then heating it's terminal with your iron, you can "tack" the lead in place. Do the same with the other end. This will hold the connector firmly in place. If necessary, you can remelt and reposition the connector until it is perfectly aligned. Then solder the middle terminals down to the foil. Finally, go back and resolder the two end terminals to get a good shiny connection.

Note that the Mic ground is not soldered at the PC board end. It is soldered at the Plug end. This prevents "ground loop" interference while still maintaining shield properties of the cable. An optional LED circuit is shown in the schematic. You can add this by drilling extra holes in the PC board. The LED will light up when data is being loaded to the computer. This helps you determine whether data is present as well as being an aid in deducing the number of separate data loads the program must go through to run. Some prefer this to the audio modifications shown in chapter 4 and later in this chapter.

POLARITY

Be extra careful in wiring the audio plug. If you use any recorder other than the Sanyo recommended, you will need to determine the polarity. Most recorders have a negative center pin. Sanyo is opposite! See figure 5.4 inset. If your recorder has the positive lead connected to the Mic and Ear ground pins (Positive ground system) you will not be able to use Tapeworm. (Some Panasonics are wired this way)

If you make a mistake on the power connections, it will blow the computer fuse. It is highly unlikely to cause any other damage.

USE OF TAPEWORM

HOOK-UP

- Always plug the Tapeworm into the computer COMPONENT SIDE UP with the computer TURNED OFF!
- Make sure all cassette recorder switches are up or OFF before switching the computer on.
- Use high output, low noise tapes of good quality.
- Insert MIC and EAR plugs into the cassette jacks marked MIC and EAR.
- Insert Tapeworm plug marked DC6V into the cassette jack marked DC6V or 6V power.

OPERATION

- Turn on computer.
- You can advance or rewind tapes at this time.
- The cassette recorder volume should be set to about 3/4 of full volume. This setting may vary depending on tape quality and recorder used.
- The motor can be disabled by typing SAVE or S shift A

RETURN then hitting Run/Stop.

- The computer SAVE, LOAD, and VERIFY operations will now function in accordance with the computer instruction guide.

NOTE: The "PRESS PLAY ON TAPE" and "PRESS PLAY AND RECORD ON TAPE" messages will not be displayed when using the unmodified Tapeworm.

- REWIND of tape is best accomplished by placing computer in READY state and using the VERIFY command. This will turn on the motor voltage and give you control of the tape deck.

-Press RUN/STOP to disable manual control.

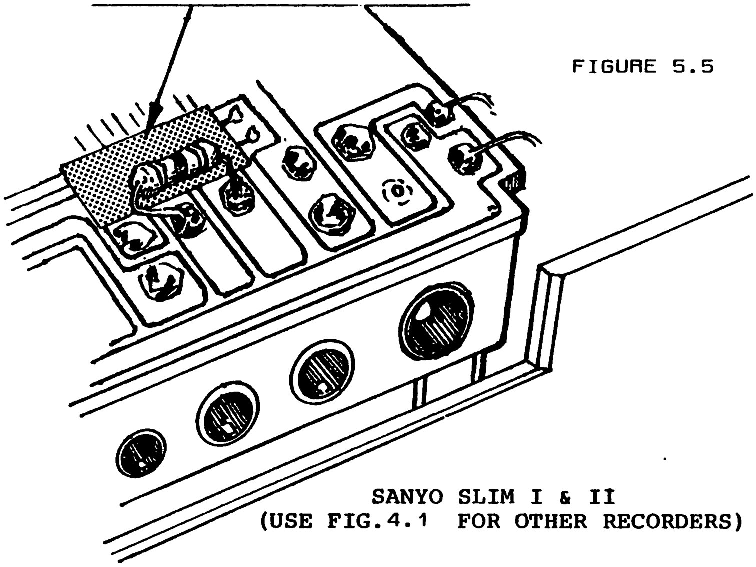
- We recommend that you always advance tape past the leader when performing SAVE operations so that no data is lost trying to record on leader.

OPTIONAL MODIFICATION

When the Tapeworm EAR plug is in the jack on your recorder, the speaker is shut off by a switch built into the jack. By jumping this switch with a resistor, a comfortable audio output will be obtained. This is a feature you may like. It allows you to hear when data is present on the tape. Figure 5.5 shows how to do this on a Sanyo Slim 1 or 2. For other recorders, use the information in figure 4.1.

USE A SMALL PIECE OF TAPE UNDER RESISTOR FOR INSULATION

FIGURE 5.5



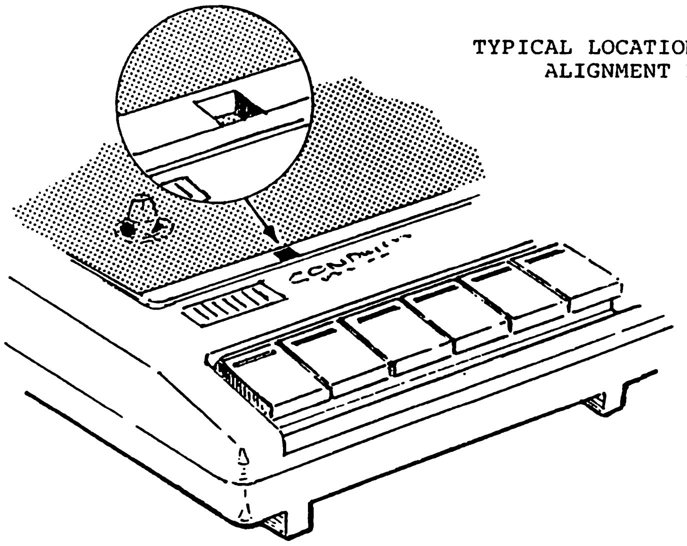
SANYO SLIM I & II
(USE FIG.4.1 FOR OTHER RECORDERS)

HEAD ALIGNMENT

Normally all tapes recorded on your recorder will be in alignment with the tape head and each other. However, tapes made on different recorders or some commercial tapes may not be aligned with your machine, resulting in difficult loading. If you are using an older tape recorder, you may need to align it with a commercially made tape before using it extensively. This will assure accurate loads of commercial tapes AND your own tapes. It is not a good idea to change the alignment once it is adjusted for normal tracking.. Otherwise you will end up with a mess of misaligned tapes, each requiring realignment to load properly. BE SURE IT IS NECESSARY BEFORE PROCEEDING WITH HEAD ALIGNMENT!

If you are using a datasette you will need to wire the "LOAD DATA AUDIO" circuit (fig. 4.1), or use an oscilloscope. If you have this circuit already or it's equivalent, simply ignore the reference to "Ear Plug" in the following procedure.

- Unplug "Ear" plug.
- Put in tape do not close cover.
- Locate alignment hole left of tape head. (see sketch)
- Set volume 1/4 to 1/2 of full - Press Play.
- Adjust screw for loudest output. **Do not turn far! A slight adjustment back and forth only!**



TYPICAL LOCATION OF HEAD ALIGNMENT HOLE

ADDITIONAL INFORMATION

Other brands of recorders usually work well with Tapeworm. However, if recorder voltage is different than 6VDC, you cannot use the power plug supplied. Instead, you can use the adapter or power source normally supplied with your recorder. This will require that you control your recorder MANUALLY since Tapworm normally controls the recorder through the 6VDC power plug.

Another inherent feature of using a standard recorder with Tapeworm is that it is still a normal recorder and as such can be used for audio work under control of your computer. By unplugging the Mic and Ear plugs and using only the 6VDC power plug, the computer can start and stop the recorder under program command. The most efficient way to do this is to use a line similar to this for turning motor ON:

```
100POKE1,PEEK(1)AND223
```

For turning motor OFF:

```
110POKE1,PEEK(1)OR32
```

CLONING

Fighting tape protection schemes can be a frustrating experience. In the least, it can become more time consuming than desireable especially if you just need to make a backup so that you can file the original for safekeeping. For these reasons we suggest that under all normal circumstances that you simply "Clone" the original using the cloneplug system we describe. As a dumb copier, the cloning method does not

FIGURE 5.6a

CLONE CIRCUITRY

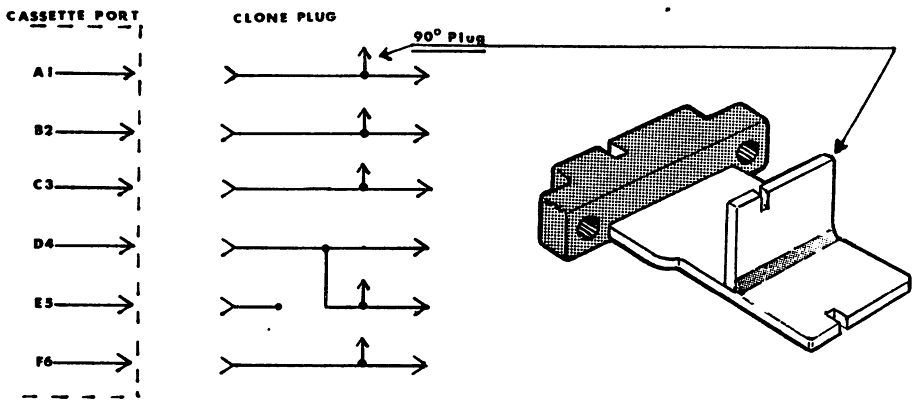


FIGURE 5.6b

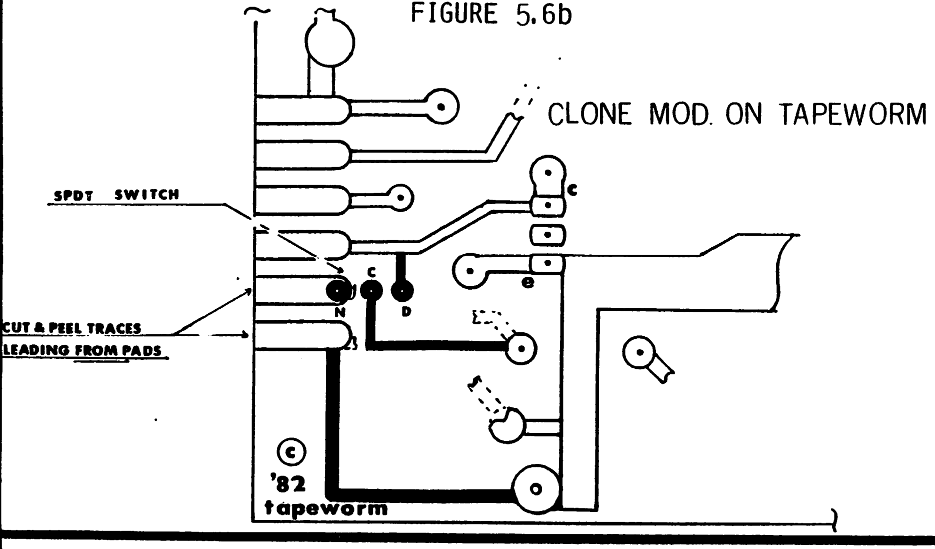


Figure 5.6a is the schematic of a clone plug. Note that if You are using one or more datasettes, that you will need the 90° plug. See figure 5.8 for PC layout.

Figure .6b shows how to modify a Tapeworm so that you will not need a clone plug. (You must use two standard recorders which work in the normal manner with Tapeworm to use this modification) You can either modify the artwork of figure 5.3 or remove traces from two lower pads and use jumpers for changes indicated in solid black. Clone is C-D, normal C-N.

depend on or vary with the type of protection used.

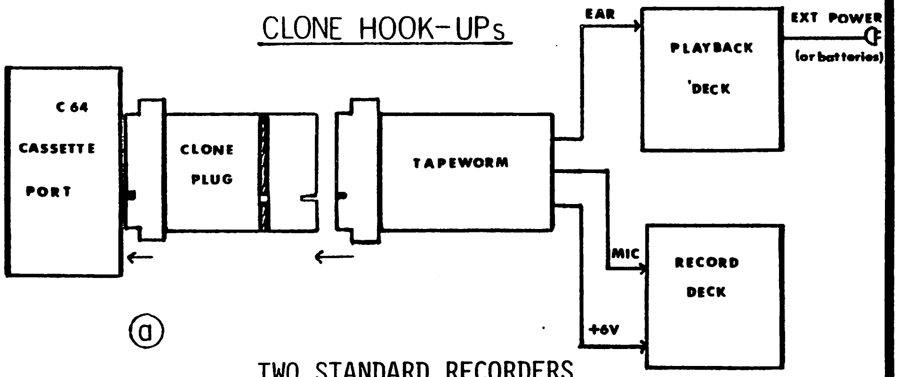
Although we have heard of some people obtaining useable copies with a straight audio to audio method and two tape recorders, we do not feel that it is a very good method. The reason lies in the fact that the signals used by the computer are not exactly the same as audio signals. The frequency requirements and tolerance to wow and flutter is not as great with these digital signals. As a result, the audio to audio method induces problems which are minor in audio recordings but seriously degrade digital data. If attempted at all, the audio method should use high quality reel to reel recorders and the copies should always be made from an original.

The Cloneplug system is a simple arrangement which uses the Tapeworm and/or datasette itself to restore the digital characteristics of the signals before sending them on to the recorder making the copy. This will normally allow good clones up to four or five "generations" removed from the original. However, when possible, best results are still obtained when the original is used for cloning.

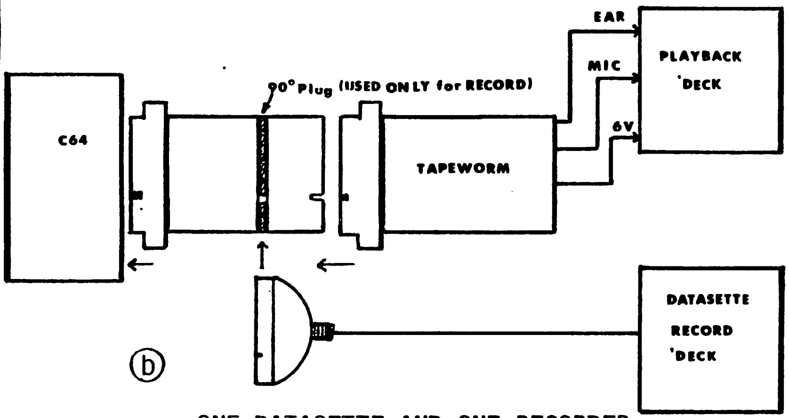
One major feature of the Cloneplug system is that you can use two standard recorders and one Tapeworm, or two datasettes and no Tapeworm, or a combination of a standard with Tapeworm and a datasette! Figure 5.7 shows the different hookups to match the equipment you have available. This way you should be able to use cloning with a minimum hardware aquisition.

FIGURE 5.7

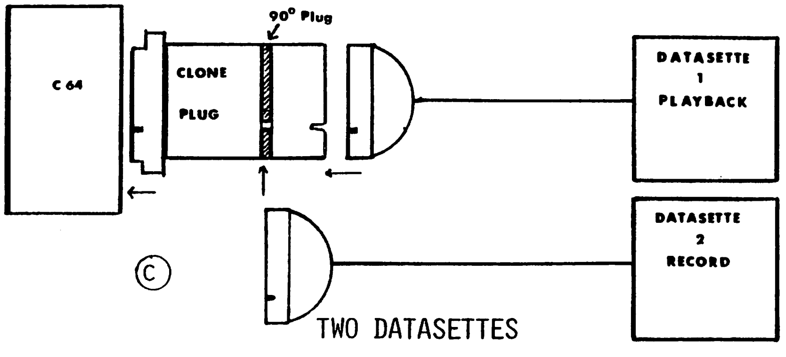
CLONE HOOK-UPs



(a) TWO STANDARD RECORDERS



(b) ONE DATASETTE AND ONE RECORDER



(c) TWO DATASETTES

CLONING PROCEDURE

Before cloning, be sure that the recorders that you are using are compatible with the Tapeworm and C64. If you are using datasettes only and no standard recorders, you will not need to worry about Tapeworm.

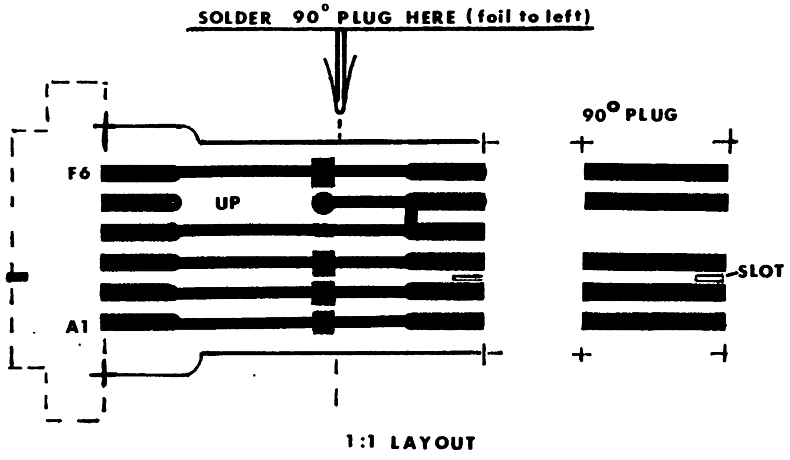
1. Determine the correct hookup from figure 5.7 based on the type of recorders you are using.
2. Place original tape in PLAYBACK DECK. NOTE: PLAYBACK DECK plugs into main (horizontal) edge of Cloneplug.
3. Place blank tape in RECORD DECK. NOTE: RECORD DECK, if using 5.7b or 5.7c, plugs into vertical (90 degree) plug of Cloneplug.
4. Make sure proper cassette buttons are set. Enter LOAD command on computer.
5. When done , verify clone by loading it.
6. Best results are obtained if you record the clone on the same deck as you normally use to load programs when not cloning.

SAVING TAPES

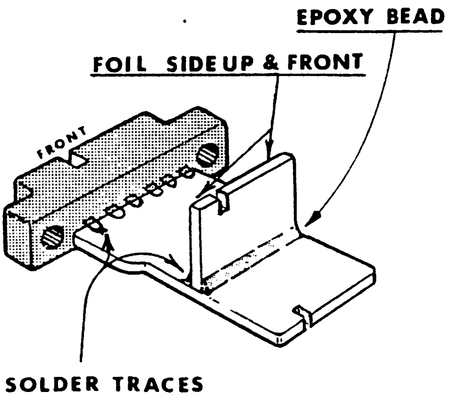
There are times when you may wish to Save a program via the computer instead of cloning. The biggest advantage to doing this is that the "Saved" program is equal in quality to an original tape.

FIGURE 5.8

SUPER CLONE PLUG PC BOARD



90° PLUG IS FOR RECORD ONLY (SEE FIG. 5.7)



Secondly, you do not have to use two recorders and play around with external hardware. The main disadvantage is that you need to have a good handle on the protection being used. Often this will mean simply to know how MANY sections a program loads in, where the sections load TO, and whether the program sections are MACHINE or BASIC. Following is a general outline of procedures to make direct computer saved copies of many taped programs.

Start by determining the number of separate loads taking place. This can be done by listening for the long tone leader followed by the short header burst. Another way is to watch the screen for the FOUND ..... interval which occurs after the header.

STOP the computer after each header load (before data load) and use an editor/assembler to read the tape buffer. Write down the starting and ending address given on each header. NOTE: DO NOT LOAD THE PROGRAM SECTIONS. Fast forward past these to the next header.

When done you should have a list that looks something like this:

```
1rst PRG. Starts $02A7 Ends $030B
2nd  PRG. Starts $0801 Ends $30FF
3rd  PRG. Starts $C000 Ends $C400
```

Remember that your list values will be in hex. Programs that load below $0801 probably have something to do with autorun and will require relocate/loading (see separate section this chapter) to load and save. Write R/L by each of these on your list. The programs above $0801 can be saved separately as machine or basic. If they are at $0801 you can load them separately and list them to see if they make sense as basic routines. If so, write BASIC beside them and when you save them for your final duplicate, they will be saved via the basic "SAVE" command. Assume the rest are machine language, they will be saved with the editor/assembler. Write Mach. beside each of these on your list.

To make the backup, you will need to save each program in order on your copy tape. Using your list, perform a LOAD"",1,1 and the appropriate save for each program. (Use TRELO and its special save procedure on those indicated R/L) If all goes well, your final tape will have an exact copy of the original programs. If you are so inclined, you may try to defeat the autorun feature all together so that your copy will be easier to duplicate for future backups.

MACHINE SAVES

If you are not familiar with editor assemblers, it may seem like a big chore to do machine saves. This is not true, it is a very simple process which you need to know. We recommend the use of Monitor$8000 and Monitor$C000 which Commodore sells on it's "Commodore 64 Macro Assembler Development System". (see chapter 2) When modified according to appendix B, any area in memory can be saved with the "S" (save) command as follows:

S"prg name",01,beginning hex address, ending hex address

NOTE:the 01 indicates tape drive. For disk use 08.

As you can see, this is a natural with the information provided by the tape buffer.

RELOCATE LOADER for TAPE

TRELO-TMACHRELO

The main purpose of the Relocate load (r/l) process is to force programs to load in the wrong place so that autorun and other "disabling" features will not operate. This allows you to "get into" the program and find out what it does. In some cases, we just want to be able to Save the program, which of course can't be done if it is running. The basis of the technique given here is that autorun routines which keep you from getting into programs are normally loaded into memory locations in the "control zone" before $0801.

The tape relocate/loader will shift the program up in memory to $2000. Thus a autorun which normally loads to $0100 will be loaded by the R/L to $2000. This relocating process will keep the program from running. The save is performed by TRELO as a machine save from $2000 to the "new end" of the program. TRELO does all the calculations for you and displays the original addresses so you can replace them in the tape header.

Since the relocated program SAVEs from an abnormal location, it will not yet load and run correctly. The trick is to EDIT the header data that tells the program where it will load to. In other words, your copy must have a different header put on so that it will load to the normal location instead of the relocated area. This procedure is called "header swapping" and can be tricky. It will require the use of the "Save Data" audio circuit of figure 4.1 or split second timing. If you use the timing method you only have about +/- 0.5 seconds to react. Also close attention needs to be paid to the EXACT start of the header on the tape. The process involves replacing the relocated copy's header with a "normal" one. The following step by step procedure will simplify this process for you.

RELOCATE/LOADER PROCEDURE

1. Load and run TRELO.
2. Follow prompts to make relocated copy. Before making the new header, study the following steps. They direct you through the process of putting a NEW header over the one with the relocated address. (TRELO keeps track of the correct addresses for you)
3. Connect "Save data audio" circuit or obtain accurate STOPWATCH. The audio method is more reliable. (With some monitors, you can turn up the volume and it will pick up the SAVE audio, eliminating the need for the

circuit or stopwatch. Check this by SAVING a program with your monitor volume at full) .

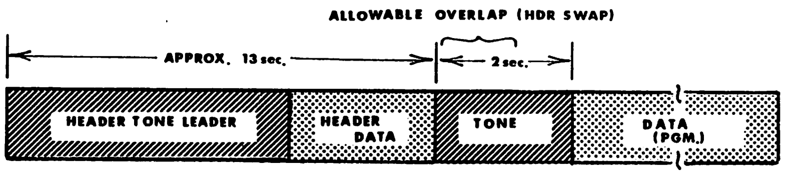
4. Set up relocated copy tape at the very beginning of FIRST tone leader. This positioning is critical if using timing.

5. Follow prompts. By listening to data, allow to save ONLY THE HEADER, then STOP tape recorder. (Or time for 13.1 seconds)

6. If all went well, your relocated program will have a "normal" header which will cause it to load and run as it should!

HINT: Practice listening to tape saves and loads to get used to the "sound" of headers and data loads. (see sketch) With a little practice, you will be able to accurately respond to the short break between. You must use the same name on the relocated copy and the doctored header so that the time for the header will not change. Be careful to start the tape in the same place when editing as when the relocate load was done. This will prevent it from recording over the data.

TAPE HEADER CONFIGURATION SKETCH



TAPE TO DISK TRELO PROCEDURE

1. Load TRELO and RUN.
2. Follow prompts for disk. WRITE DOWN ORIGINAL STARTING ADDRESS! Convert to HEX, break into HIGH byte and LOW byte. ie: if calculated value is $02A7 High byte =02 Low byte =A7.
3. When process is finished, use SUPERDIRECTORY to find the starting TRACK and SECTOR used by the disk to store your program.
4. Use DISK-EDITOR to change the starting ADDRESS on the Relocated copy back to what you calculated. (Original Address) The third location from the top left on Disk-Editor display is the LOW byte. Replace the THIRD BYTE TO YOUR CALCULATED LOW BYTE. The Fourth location is the HIGH byte. Change the FOURTH LOCATION TO YOUR CALCULATED HIGH BYTE. Hit RETURN and follow prompts to save data. (Chapter six has more info on DISK-EDITOR)
5. Relocated copy should now run normally. NOTE: NOT ALL PROGRAMS CAN BE SUCCESSFULLY COPIED WITH THIS PROCEDURE.

NOTE: If exchanging program medium from tape to disk, or vice versa, remember to change the device number in the booter section.

TRELO

```
1 IFA=0THENA=1:LOAD"TMACHRELO",8,1
2 POKE53280,11:POKE53281,11
3 PRINT"▯▮▘  TRELO/LOADER ▮PSIDAC (C)84 VBN▚":PRINT"▮▮":GOTO15
8 PRINT"▮PUT CPY TAPE IN DRIVE - SET FOR ▚ RECORD▀":PRINT"▮PRESS F7"
9 GETA$:IFA$<>"▮"THEN9
10 PRINT"▮SAVING ";NF$;" FROM ";BG;" TO ";EA:SYS8043
11 IFCK$="D"THENOPEN15,8,15:INPUT#15,A$,B$,C$,D$:PRINT"▮"A$,B$,C$,D$:CLOSE15
12 POKE8047,1:POKE8049,1:INPUT"▮▚MAKE NEW TAPE HEADER";AK$
13 IFAK$="Y"THEN100
14 IFCK$="D"THEN200:END
15 PRINT"▮PLACE ORIGINAL TAPE IN DRIVE":POKE1,PEEK(1)AND223
20 PRINT"▮PERFORM TAPE LOAD... WHEN FOUND MESSAGE"
21 PRINT"▮APPEARS, ▘RUN/STOP▚ THEN ENTER ▘RUN25▚":END
25 POKE52,20:POKE56,20:POKE53280,11:POKE53281,11
26 AH=PEEK(830):AL=PEEK(829):BH=PEEK(832):BL=PEEK(831)
27 OD=(PEEK(830)*256)+PEEK(829):OE=(PEEK(832)*256)+PEEK(831):BY=OE-OD
28 BG=32*256:POKE254,32:POKE253,0
30 PRINT"▯▮▘  TRELO/LOADER ▮PSIDAC (C)84 VBN▚"
32 POKE1,PEEK(1)AND223:PRINT"▮▚REWIND TAPE ▀PRESS F7 WHEN DONE"
33 GETA$:IFA$<>"▮"THEN33
34 POKE1,PEEK(1)OR32:PRINT"▯▚PRESS PLAY ▀                              "
35 INPUT"▮INPUT ▚EXACT▀ FILENAME";NF$:NL=LEN(NF$):POKE251,NL
40 SYS8008:SYS8103
45 PRINT"▯▯▯# OF BYTES="BY;"                                 ":EA=BG+BY
50 PRINT"▮ORIGINAL STARTING ADDRESS = ";OD:PRINT"▮ORIGINAL ENDING ADDRESS = ";OE
55 INPUT"▮SAVE ON T/D";CK$:IFCK$="T"THEN8
60 PRINT"▯PREP DISK TO COPY - F7 WHEN READY":POKE8047,8:POKE8049,8:GOTO9
100 POKE830,AH:POKE829,AL:POKE832,BH:POKE831,BL:POKE254,AH:POKE253,AL
105 POKE175,BH:POKE174,BL
115 PRINT"▯▮  *  ROUTINE TO REPLACE TAPE HEADER  *"
120 POKE1,PEEK(1)AND223:PRINT"▮   ▚REWIND TAPE TO EXACT BEGINNING▀"
125 PRINT"▮F7 WHEN READY!"
130 GETA$:IFA$<>"▮"THEN 130
140 PRINT"▯▯▯▚PRESS PLAY AND RECORD▀                    ":POKE1,PEEK(1)OR32
150 PRINT"▮▘TIME FOR ▚EXACTLY▘ 13.1 SECONDS▚.....":PRINT"▮THEN HIT ▚RUN/STOP▀"
160 PRINT"▮F7 WHEN READY"
170 GETA$:IFA$<>"▮"THEN170
180 SYS8043
200 PRINT"▮USE DISK-EDITOR TO RESET▘ LOAD TO◣ ADDRESS"
```

```
        TMACHRELO
1F48  20 95 1F JSR $1F95
1F4B  A9 01    LDA #$01
1F4D  A2 01    LDX #$01
1F4F  A0 00    LDY #$00
1F51  20 BA FF JSR $FFBA
1F54  A5 FB    LDA $FB
1F56  A2 41    LDX #$41
1F58  A0 03    LDY #$03
1F5A  20 BD FF JSR $FFBD
1F5D  A9 00    LDA #$00
1F5F  A6 FD    LDX $FD
1F61  A4 FE    LDY $FE
1F63  20 D5 FF JSR $FFD5
1F66  20 A0 1F JSR $1FA0
1F69  60       RTS
1F6A  00       BRK
1F6B  20 95 1F JSR $1F95
1F6E  A9 01    LDA #$01
1F70  A2 01    LDX #$01
1F72  A0 FF    LDY #$FF
1F74  20 BA FF JSR $FFBA
1F77  A5 FB    LDA $FB
1F79  A2 B1    LDX #$B1
1F7B  A0 02    LDY #$02
1F7D  20 BD FF JSR $FFBD
1F80  A5 FD    LDA $FD
1F82  85 FC    STA $FC
1F84  A5 FE    LDA $FE
1F86  85 FD    STA $FD
1F88  A9 FC    LDA #$FC
1F8A  A6 AE    LDX $AE
1F8C  A4 AF    LDY $AF
1F8E  20 D8 FF JSR $FFD8
1F91  20 A0 1F JSR $1FA0
1F94  60       RTS
1F95  A5 01    LDA $01
1F97  8D A7 02 STA $02A7
1F9A  29 FE    AND #$FE
1F9C  85 01    STA $01
1F9E  60       RTS
1F9F  00       BRK
1FA0  AD A7 02 LDA $02A7
1FA3  85 01    STA $01
1FA5  60       RTS
1FA6  00       BRK
1FA7  A6 FB    LDX $FB
1FA9  BD 40 03 LDA $0340,X
1FAC  9D B0 02 STA $02B0,X
1FAF  CA       DEX
1FB0  D0 F7    BNE $1FA9
1FB2  60       RTS
```

*** CHAPTER SIX ***

DISKS

If the point has not been. made clear enough yet, let me restate it here, at the beginning of the chapter you may have turned to first. There is not now, and may never be, a single software solution to the protection used on disks. Regardless of the optimism of advertisements in general, neither we nor anyone else can offer a 100% solution or method for protection breaking. The more you understand the disk drive and it's potential, the clearer this point becomes. What happens is that as a consumer you get caught in an endless vicious cycle of new protections and new products to break them. Everyone makes money at your expense! The true solutions lie in your ability to become a "detective" of sorts and to have tools that are adaptable to the situation. With this in mind, we make little attempt to supply overglossed, (and priced), declining term software. Instead we have developed some raw utilities which allow you to obtain information, look into and modify programs, override disk operating systems, and in general, to give you control of what should be done.

This chapter consists of three main sections. The first will cover each of the disk utilities in terms of what it is, generally what it does, and special operating details. Chapter two gives a shorter, less specific introduction to the various utilities for disk, tape, and cartridge. The second part of this chapter will give you some specific procedures to follow in attacking different types of problems. It will cover some of the common ways you would combine several utilities in a logical sequence to breaking a program. The third section gives you the actual listings of the programs and any pertinent loading details.

SECTION 1 - UTILITIES

SUPERDIRECTORY

Gives you an expanded directory of whats listed in the normal disk directory. The additional features include identification of the starting track and sector for each program and file, listing of deleted programs and files, and the starting address of each program given in hex and decimal. Although this may seem innocuous at first glance, it is really a key to determining what you may be up against on a given program. If for example, the program shows a starting address of less than $0801, it will affect the control zone and may be an autostart. Relocating will probably be needed if you wish to save or pick. The number of blocks used by the program will give you an idea of what its function may be. Again, a short routine loaded first is likely autorun or autoboot. If the program begins at 0801, it is probably basic and if loaded by itself, you will probably be able to list it. There is however, no reason that it cannot be machine language, so be ready to try disassembly also. Do not run a routine first, often the program can be listed before you run it but you may lose control after you run it. In addition, if the program is loaded separately, it may not run correctly as the other sections are often needed.

Programs starting in the $033C-$03FF range are usually machine routines or access keys which reside in the tape buffer. In these you should look for a meaningful dissassembly or ASCII code representing information.

Programs above $0801 will often be machine language routines or hex data used by the main program. Blocks in the $8000 and $C000 range are popular for machine programs and data.

Superdirectory utilizes the machine routine "ADMACH" in its operation. Loading and running Superdirectory will automatically load ADMACH. Use of Superdirectory is simple, after loading and running, it will give you the option of sending the output to the printer as well as the screen, then asking for the disk to be examined. We recommend that you print a Superdirectory for every disk you have, and make it one of the first things you do with new disks.

RELOCATE/LOADER

One of the disadvantages of using whole disk copiers is that an entire disk may be required when only a small portion of that disk is actually used. Many people like to combine similar programs on a disk so that they might have several games on one disk, several utilities on another, business packages on another, and so on. In order to "pack" a disk with different programs you usually need to use normal basic or machine SAVE commands. As you well know, commercial prgrams do not usually lend themselves to this operation.

The premise of Relocate/loader is that if a program is loaded to the WRONG PLACE, it will not take control of the system. Thus you are free to SAVE at will. The R/L has the SAVE routine as an inherent part of its operation. The most important thing to remember is that the relocated copy will always have a starting address of $0A00, regardless of where the original was supposed to reside. Thus the relocated copy cannot RUN until its starting address has been doctored. That is one of the primary purposes of Disk-Editor.

If you wish to pick the program, you can use the relocated version before you change the address back.

The disk version of Relocate/Loader uses a short basic program to control the machine relocating programs. The machine routines are loaded through the basic main program. "MACHRELO" is the name of these machine routines. One of the things done by the system is that basic is switched out during the relocating process. This along with the compact size of the basic main and machine routines allows us to squeeze in a 50K byte buffer. The size of this buffer should allow the use of R/L on the vast majority of "long" programs.

DISK-EDITOR

As one of the primary uses of Disk-Editor is to allow you to change the addresses contained on a disk, we will go into this next. Each sector of the disk contains header information and then a "block" of data which is 256 bytes in size (binary, actual GCR format is longer by 4:5 ratio, see Appendix F). Disk-Editor allows you to dump any block on the disk to examine it or to change any of the data contained in the block. On every block, the first two bytes of data tell which block is next on the disk. In this manner, the drive can "link" all of the blocks together which belong to any given program. Disk-Editor shows the links in green. On the first block of a program, the second and third bytes give the starting address (hex) of the program. These are the two bytes you need to change when "resetting" the address on a relocated program.

Once these addresses have been reset to their original values, the relocated program can be loaded an run as though it were the original! Remember that the two bytes you change are in hex and in the standard 6502 format of low byte first, high byte second. Also remember that they are contained on the starting sector of the program. These locations in the rest of the sectors contain program data.

There may be times when you will want to change some of the data within a particular sector of a disk. Disk-Editor gives you this option. It is simple to use, you postition the cursor over the bytes you wish to change, and type in the new values. When you have completed all the changes for a given sector, hitting the RETURN key cause the changes to be read and prompt you on whether you desire to actually write the changes onto the disk.

Disk-Editor is a basic program which loads its machine subroutines under the name ADMACH. As with the others, you must run the basic program to load the machine routines. If you get into a situation where you need to re-RUN the main program, you can always skip the first line which calls the machine load, as long as the machine program has not been wiped out in some manner. This is true of all of our basic/machine programs.

LINKSTER

Linkster is a basic program which gives you a complete list of sector links for a given program.

Linkster allows you to specify which track and sector you would like to begin with, then it will print a list of all the tracks and sectors used from that point in the program. You do not have to start linking from the beginning, it will just give you the ones from the point you choose. Linkster also counts the number of blocks from the point you started. The printout can be screen only or screen and printer.

Other than simply finding out where a program is stored on your disk, linkster can be used to find bad sectors when a program disk becomes defective and looses some data. If you have a program which loads for a bit and then stops and returns a disk error, use linkster to find out which sector the error is coming from. You can then concentrate on "fixing" the sector. If it is really bad you may be able to change the previous link so that the program "skips" the bad sector. In some cases you can list the program and replace what is missing by hand and then resave the "repaired" program!

ERROR ANALYZER

Error Analyzer checks an entire disk for errors by either a sector by sector check or by a track by track check. Error Analyzer checks tracks all the way to 44. This gives you the ability to find out if any level 4 protection has been used on the disk. The track by track check is primarily a sync locator which can indicate if sync has been wiped out on any normal tracks and if any has been written on the "extra" inner tracks above 35. Many of the protection methods such as wiping out a track, erasing sync, or writing to the extra tracks can be detected by this program. If any information or protection codes have been written to the "extra" tracks, this is the only program that will detect that, and do so in a short amount of time.

The track by track check only takes about 45 seconds and is a good idea as a primary test on new programs. The sector by sector test takes longer but is more thorough. When errors are found the program is designed so as to not "bump" the disk head which can be detrimental to alignment on some disk drives. Instead it uses a machine programming technique which will recheck each track or sector four times and then go on to the next. You will appreciate this feature on disks with lots of errors. It is quicker, cleaner, safer, and doesn't make your disk sound like its having a cow.

Error Analyzer has printer options on both checks. This will give you a convenient paper listing of the errors found. These printouts supplied by our programs should be used as a "map" as you work back through your copied program to get it operating. With current protection involving many errors in various locations, you will need these printouts to work your way through the "patching" chores on your copy disk.

T/S ANALYZER

T/S Analyzer has the primary purpose of producing a error log which can be stored as a file and passed along to Fastback. Although T/S does not have the advanced reading technique of Error analyzer, it does have several logging options and serves as a map maker for Fastback. If you plan to be doing a lot of copying, you should format a disk especially to hold your T/S error logs. In this manner, the log only need be done once when the disk to be copied is new. As time goes on, you will build a file of error logs and if you ever need to do another backup you can simply load Fastback and through it, the appropriate error log for the program in question. The output of T/S tells both what errors are encountered as well as which sectors contain data and which sectors are unused on the disk. This can be helpful to locate sectors which have data but are not called through the normal linking method. If a sector has been destroyed, linkster cannot find out what is beyond the bad sector. T/S can give you a clue as where to pick up from!

T/S loads its machine routines under the name "ANALYMACH". As before, this is done in the first line of the program. T/S gives you four options. First to analyze the Tracks and Sectors on the disk being tested. Once this is done you have the choice of (2) printing the log or (3) saving it to your log disk. The final option is to load an error log previously saved. You can then print that if you desire. This allows you to make your logs at one time and then later as the need arises or it becomes appropriate, you can make a printout. The number of errors counted is kept track of and printed out for your convenience.

FASTBACK

The error log from T/S, which Fastback asks for, relieves Fastback from having to bother with any empty or bad sectors on the disk. Only those with valid data are loaded into the buffer and subsequently saved on your copy disk. Thus Fastback is just about as fast as a copier can be. The operation is comparable to a BAM copier which only copies those sectors containing program. The difference is that it is not using an easily destroyed BAM but rather a verified list of which sectors have data and are copiable.

Once a Fastback copy has been made, you can use the error log printouts which indicate where changes, or errors need to be. Error writers or Diskpicker can be used to put errors on. Alternately you may choose to list/disassemble the program and try to modify the sections which look for the errors. There may be data in the sectors with errors that you will need to recover. Fastback loads its machine counterpart "ANALYMACH" in the first line of the program.

DD-1

DD-1 is the name of a group of four programs which provide direct duplication via whole disk copy. The programs are 1DUPDAC and its controller 1PSIMAIN which are for single disk copies. 2DUPDAC and its basic controller 2PSIMAIN are used for whole disk copies with two disk drives. Although not as advanced in technique as Fastback, you will find DD-1 is very handy to make an on the spot backup of a less protected disk. It can be quite fast if the original is only partially full and you make use of the fast copy features.

The programs are able to copy many of the lower level protection disks since they provide a direct track by track duplication which does not depend on BAMs, and it skips over errors. A 150 block buffer is used for a minimum number of swaps. Track and sector manipulations are handled by machine routines for maximum reading and writing speed. Error decoding allows simple operation and circumvents many error protection schemes.

The system also features an error display which can be sent to a printer for logging and later error writing or removal. DD-1 will skip sectors which contain errors designed to stop whole disk copiers. If the program requires the errors to be present you can use diskpicker or an error writer to replace them.

The "fast write" feature of DD-1 causes the program to skip over the writing of any sectors which contain only format data and no program data. For compatibility with other drives, this can be set to look for a normal format of "ONEs" or "ZEROs". The program checks the contents of the sector before writing it and if it contains only format data it will skip to the next.

The Multiple Copy Option allows making more than one copy per original - without rereading the original. This will allow you to read in a section of the original, then write it to several copies, and so on. This saves a great amount of time since the original is only read once.

After chosing either 1 or 2 disk drives, the corresponding DUPDAC is loaded. Then type NEW and load the basic PSIMAIN version that corresponds to the number of drives you are using. The options and procedure for DD-1 are given in the procedure part of this chapter (section 2).

DISKPICKER

Diskpicker is a machine language development system for the 1541 disk drive. It is designed to allow you to develop machine language routines in the C64 memory space and then transfer them to the disk drive. From there, the routines can be executed from a control menu in Diskpicker. The memory transfer features of diskpicker allow you to transfer any or all of the disk drive memory contents into the C64 memory space. It can then be examined, printed, modified an so on. Since Diskpicker uses the Commodore MONITOR$8000, you have a full featured editor assembler for accomplishing the various machine language tasks. Diskpicker also utilizes ZMACH which gives it the ability to switch effortlessly between its basic and machine functions.

You may wish to make a ROM listing of your disk drive ROM using diskpicker. The 16K size will make a rather large printout however. The disk controller routines that you will need the most reside from F24D to FFFF and thus makes a much shorter listing.

In addition to writing routines to be sent to the disk drive, you will also be able to load sectors from the disk drive and transfer them back to computer memory to see what is there. Diskpicker allows you to look at header images , and other format information that you normally couldn't see. Also modified headers can be created which will produce errors if read normally. With diskpicker, special machine language routines can be written to find and read data after these modified headers. This gives you the power to experiment with protection methods of your own as well as read out data that would otherwise be hidden.

The specific procedure for Diskpicker will be given in the next section but the menu options available are as follows:

1...Transfer disk memory to buffer (in C64)

2...Enable monitor (machine E/A)

3...Transfer (C64) buffer to disk memory space.

4...Direct execute user program.

5...Job Que execute user program.

6...Load Sector to disk buffer.

7...Initialize disk I/O.

8...Format Diskette.

ZMACH

Zmach is a short machine routine which you can load any time you are using an editor assembler, particularly Monitor$8000. Zmach provides you with a way to save and restore zero page as you go back and forth between basic and machine language. This is necessary to prevent "lock up" of the computer when eXiting from the editor assembler. When loaded, before going to the E/A, you SAVE ZERO PAGE by typing SYS49152 RETURN. Later, after you exit from the machine monitor, you type SYS49184 to RESTORE ZERO PAGE.

SECTION 2 -- PROCEDURES

We will start this section with general procedure to follow with any new disk and then we will cover remaining individual utility programs with special notes to make you aware of additional options that may not be obvious from a users standpoint.

1...Do a SUPERDIRECTORY listing of what is on your new disk. If you have a printer, make a hard copy for your records.

2...Run an ERROR ANALYZER TRACK check. This will identify use of error protection and extra tracks. Make hardcopy if errors show up. Next do a TRACK and SECTOR analysis, making hardcopy as needed.

3...Choose copy method. DD-1, FASTBACK or RELOCATE/LOAD. Go directly to procedure for method chosen.

DD-1 PROCEDURE

1...SELECT and LOAD "1DUPDAC",8,1 then NEW and LOAD "1PSIMAIN",8 for single drive users.

2...SELECT and LOAD "2DUPDAC",8,1 then NEW and LOAD "2PSIMAIN",8 for dual drives.

3...PRINTER option. List and change line 1 P=0 to P=1 to turn on printer. F7 must be pressed after each error printout unless last line of program is changed from SYS49903 to SYS49881. NOTE: if you use this option a lot you may wish to save the modified version for your own use. We have chosen this method of configuration over a menu to conserve memory for buffer space.

4...SET FAST COPY MODE. Run selected DD-1 program with disk to be copied but STOP after the first few sectors have been read on track one. ?PEEK(2561) and ?PEEK(2562). If both are 1 then POKE49747,1 and POKE49751,1. If both PEEKs were 0, then do nothing. 0 is default value for DD-1 FAST COPY.

5...SET MULTIPLE COPY option. Default makes one copy. POKE49174, [number of EXTRA copies desired]. Note that the value you poke will produce that many EXTRA write cycles. Thus poke 1 for 2 copies, 2 for 3 copies and so on. The default value of 0 produces one copy.

6...DUAL DISKS; device number 8 for original disk, and device number 9 for copy disk.

7...Run DD-1 program chosen. You must use previously formatted disks for copies since DD-1 has no format option. If you forget, just RUN/STOP and format disks in normal way. (Do not use wedge), then RUN again. Follow PROMPTS on screen.

8...Bell indicates program read or write is active. TRack, SECtor, and ERRor displays indicate current location and any errors encountered. You will have time to write the info down if an error was found. You must press F7 after error to continue with copy process.

FASTBACK PROCEDURE

1...LOAD and run T/S analyzer. Follow PROMPTS. Make error printout if desired. SAVE error log on disk reserved for that purpose.

2...LOAD and RUN FASTBACK. Follow PROMPTS.

3...After copy is done, use error log and error maker or Diskpicker to write errors back on copy disk.

4...As an alternate to error writing, try to disassemble original program and remove error checking routines. As these routines may be in the boot, you may need to use Relocate/Loader to get into these routines.

RELOCATE/LOADER PROCEDURE

1...LOAD and RUN RELOCATE/LOADER.

2...Follow PROMPTS and select program from original disk to be Relocate/loaded.

3...Type in EXACT program name when asked for it.

4...Save the Relocated copy. If you are trying to pick the program you can load Editor /Assemblers or other tools needed for picking. The relocated copy can be loaded as often as needed for study purposes. It will have a starting address of $0A00 or decimal 2560. You do not need to change the start address until you are done "picking". While picking, you can save the program as a normal machine language routine.

5...If relocating has been done to COPY and if you have completed any needed changes and now need to reset address, go to the DISK-EDITOR PROCEDURE.

DISK-EDITOR PROCEDURE

1...LOAD and RUN DISK-EDITOR.

2...Follow PROMPTs and select track and sector desired to edit. IF RESETTING ADDRESS, this will be the first track and sector for the program. If you do not know which is the first track and sector you will need to do a Superdirectory listing to find out.

3...Once desired sector has been loaded, you will have a display with the hex values of each byte in the sector. The first two are green and are the NEXT TRACK and SECTOR values IN HEX. (Convert these to decimal to find next track and sector and convert any desired values to hex before trying to replace current ones) For example if next track and sector is given as 17 10; the decimal value is 23 16. If you wish to change this link to. to say track 1 sector 12 you would type in 01 0C.

4...If you want to change the LOAD-TO address, it is given in the 3rd and 4th bytes. (Two following the green ones) Remember this only applies to the first sector of the program, all others contain data in this location. The values are in hex with the low byte first and the high byte second. If you are looking at a relocate/saved copy, you would see in this location these numbers: 00 0A Which means that the program will load to location $0A00. If you are resetting the address back to what is used on the original disk, look up the correct start address from the HEX.ADD column of your Superdirectory listing of that disk.

5...Once all desired changes have been made, follow the PROMPTS to either SAVE changed sector or get another one. SAVED sector will replace itself on the disk.

DISKPICKER PROCEDURE

The uses of Diskpicker go well beyond what a simple procedure can give you. As it is a system that allows you to develop programs to use in the disk, we will explain how to use the options and give you some example error routines which you can send to the disk. Beyond this, you will need fluency in machine language to be sucessful. Note that you must obtain a copy of MONITOR$8000 which should be saved on your PSIPACK disk. Other monitors may be used if you want to change the program lines that load and call MONITOR$8000. (SYS32768 is the call). Remember that other monitors cannot occupy the beginning of basic RAM or $C000 locations. You will need some free RAM to use as buffer space for developing routines and for storing information read from the disk. Typically a couple of K will be enough.

1...LOAD and RUN Diskpicker.

2...MENU OPTION 1 - Transfer disk memory to buffer.

The purpose of this option is to allow any valid locations in the disk drive to be transfered to the memory of the C64 from where you can dissassemble, study, and modify as desired. The monitor printer options will allow you to make printouts of this memory. Since the total RAM in the disk is only 2K in size, you have plenty of room in the 64 for this purpose. The requirements of Diskpicker do limit you to using memory between $3000 and $7FFF which is 20K.

Option one will ask for the starting and ending DISK locations you want to transfer to the C64. These must be in HEX! It will then ask you where you want the C64 buffer which will recieve the data to be. We STRONGLY recommend that you always use the same page offset. Thus if you want to transfer disk $0300-$03FF to the C64 you should use a buffer start such as 3300 or 4300 or 5300 etc. This will make disassembly more meaningful because all page addresses will be the same. NOTE: We define 00-FF as location address; 000-F00 as page address; and 0000-F000 as block address.

2...MENU OPTION 2 - Enable Monitor Mode.

This option puts you into the editor assembler. We use Monitor$8000 (by Commodore) for this purpose. The typical E/A features are available in this mode. For example, you can select a memory area and write a machine routine which can be transfered to the disk by option 3. When looking at information that you have loaded from a disk through disk memory, we recommend using the I (interrogate) command which will give you an ASCII as well as hex dump. This can be useful when looking for acess codes or particular info which would be meaningful in ASCII. Keep a copy of figure 3.3 (disk map) handy when using the monitor.

To EXIT the monitor mode, type G C020. This will return you to the main menu in basic mode. Your programs or data that you were working on will remain intact as long as you do not try to transfer something on top of them or erase them with a monitor command. This gives you the ability to keep many "images" of disk memory or machine routines in memory at one time.

3...Transfer Buffer to Disk Memory

This option allows you to send the data or program that you have in the C64 buffer space TO the disk drive memory. Remember that the disk uses $0000 to $02FF for system purposes and you will not normally transfer programs to these locations. The buffers in the disk, 0300, 0400, 0500, 0600, and 0700 are perfect for such uses.

4...Direct Execute User Program.

Assuming you have transferred a program to the disk drive, this option will allow you to cause that program to be executed! You will be asked for the address (hex) in the disk at which you want execution to begin. The proper commands will be sent by Diskpicker to cause that program to RUN in the disk memory. It is a good idea to make sure a scratch disk is in the drive the first few times you try a routine in case it backfires.

Direct Execute is primarily for routines which manipulate data rather than routines which control reading or writing to sectors. The reason for this is that direct execute does not provide for automatic track and sector preparation as does Job Que Execute.

5...Job Que Execute.

This option provides execution of your routine AS A PART OF one of the EXISTANT DISK ROUTINES. In other words, if you select lets say a Job Que WRITE, the drive will find the track and sector you have selected, and then execute your routine. Normally your routine would be designed to affect what might normally have been done through the disk routines. The job que functions save you the trouble of trying to get the motors going, find track and sector, etc.

The following list gives the direct execute commands:

128-READ -Reads in selected sector.
144-WRITE -Writes to selected sector.
160-VERIFY -Compares sector with one in memory.
176-SEEK TRACK -Locates specified track.
184-SEEK SEC -Locates specified sector.
192-BUMP -Runs head to stop and bumps (resets head)
208-JUMP -Jumps to specified memory location.
224-EXECUTE -Puts Track and Sector to be affected in Que, finds track and sector, loads the header there and goes to your machine routine. Note that your machine routine or its jump vector must start at the beginning of one of the five buffers, $0300 through $0700. Since data is read from the disk header, variables in the drive memory will be affected.

6...Load Sector To Disk Buffer

This option will ask you for the headers (track and sector) and will then load that sector to the disk memory. From there it can be transferred to C64 Memory for study. This option is useful to examine specific sectors.

7...Initialize disk I/O

This is essentially a RESET command which returns the disk to power up conditions. It does so without wiping out all of memory like a C64 RESET would do. Useful when you need to be sure the disk is clear for other operations. It is a good idea to use this command any time you have uncovered and error and wish to send new commands to the disk. This makes sure the disk is ready to receive the data properly.

8...Format Diskette.

This option allows to format a diskette without breaking out of program. This can be useful since you will typically be trying to do things which can "mess" up your practice scratch disk!

9...Resetting.

If the system locks up, use SYS49184 for a warm reset. In extreme cases you may need a reset button or turn off the computer. (See Appendix E). X is used to escape an input mode question.

ADVANCED TECHNIQUES

In making protection, there are some things which go beyond simply writing an error and having the program test the error by trying to load the sector. Error writing programs to date leave you with little other option. There are many things possible of which we will try to start you thinking about a few.

Consider first, how a disk sector or partial track could be erased. The disk would normally find this sector by its header, which can't be done if it has been erased. However, if you know which sector is bad, you can tell the disk to look for the sector in front of it, which is good. Once the preceeding good sector is found, the disk is programmed to wait for a certain period and write some data. This data then goes to the "nonexistant" sector. A similar process can be used to read this data. A quite effective means of protection since it is difficult to reproduce the exact parameters used to write the data! This variation can be used on the "extra" tracks beyond 35.

Other ways to protect include putting data in the GAP at the end of a track. If your copy maker is not aware that data is there it will not look there. With the machine routines we have included, the gap data can be read in a fashion similar to the above process.

Tracks and Sectors can be given illegal numbers by changing this data in the header. (Appendix F) The normal DOS will not accept out of range values. A machine routine used by the disk drive can do this. The diskette could also be entirely or partially formatted in an abnormal pattern which would only be recognized by a modified DOS routine.

Another devious means would be to write "encoded" sync pulses on an unused part of the disk. These sync pulses could be encoded by a means as simple as spacing. In other words, the time between consecutive sync pulses would have to be exactly according to a predetermined VARIABLE spacing, or the program checking them would abort. Breaking such a system would require a sophisticated analysis of the diskette which cannot be done through normal DOS routines.

These examples should give you a clearer idea of what can be done if you go outside the confines of the DOS and develop both reading and writing routines. It is hard enough to understand how the normal DOS routines function, let alone trying to figure out what someone has done beyond this. These methods by nature will not be compatible with other drives. It is also interesting to note that copier programs usually will not copy themselves... in other words they are obsoleted by their own manufacturers since they have developed protection that foils their OWN product!! If they can do it, so can anybody else. If a copy program cannot copy itself, you may rest certain that within a

VERY short time there will be many new programs out that it also cannot copy! If the market continues to develop in this direction, we will likely continue with our development of a disk "Dumb Copier". Such a system will put to rest any protection that does not involve external hardware or physical modification of the disk drive.

ERROR WRITING PROCEDURE

Following are some examples of how to send error writing routines to the disk via Diskpicker. In the program section of this chapter we have listings for all the error routines included. You may prefer to write a basic program which "sends" these routines to the disk and then executes them. We have chosen not to as such error writers are abundant already and become outdated as fast as they are sold. If you become familiar with the techniques of Diskpicker, you will be able to add and modify routines as YOU see fit.

IMPORTANT: Although every attempt was made to make these procedures compatible with all 1541s, there are apparently four revisions of the DOS ROM put out by Commodore. As explained earlier, this can lead to compatibility problems with many forms of protected software AS WELL AS error making procedures. This is why such protection is of questionable validity. We can not guarantee that all error making routines will work on all past and future versions.

SPECIAL INSTRUCTIONS AND CAUTIONS

NOTE:These techniques should not be attempted if you do not understand the underlying principles. Indiscriminate use could cause the disk head to stick which may require partial disassembly to correct!!

1. To Escape back to Main Menu, Input letter X for requested input value.

2. To escape from Monitor$8000, input G C020. (requires space btween G & C020)

3. If disk or computer locks up, (and if you have a reset button), press reset and enter SYS49184 to restore system.

4. Always Format disk to be written to with the same disk ID# as the original.

5. Never select tracks above 44 or disk head will bang against end stop and may get stuck. If this happens you will need to open disk case and gently push head back to center.

6. If head gets stuck, first try by using Initialize operation or call the Job Que Bump command, #192.

7. Always input valid header # and data for track & sector when using the Job Que.

ERROR 20 NO HEADER

Erases header from specified track and sector.

1. Load and run Diskpicker.

2. Select menu option 2. Set printer choice "N".

3. Input L"20 NO HEADER",08 After load, Input G C020.

4. Place diskette to be written on in drive.

5. Select menu option 7.

6. Select menu option 3.

7. Input disk start address 0300.

8. Input disk end address 034F.

9. Input Buffer address 3300.

10. After Data transfer to disk, select menu option 5.

11. Input Job choice 224. Select choice 1 (0300) for execute address. FOR ONE SECTOR SET MULTISECTOR='N'
FOR ALL SECTORS, SET MULTISECTOR='Y'

12.For header #1, input track # desired to write ON.

13.For header#2, input value ONE LESS than sector desired to write ON. (USE 0 FOR SECTOR #s 1)

* Note: If sector 0 is to be written to, select the highest sector number on this track. (see drive user manual). i.e.: Input a 20 if sector 0 on track 1 is being chosen.

14.REPEAT the above steps 8-11 for writing type 20 errors to other headers of your choice. * Intermixing read, write and load operations may write over the error routine being held in the disk memory buffer.

15.To test your errors, select option 5 (Job Que Execute).

16.Input job choice 128.

17.For header #1 , enter track #desired to read, and for header#2, enter the sector number to read.

18.If everything went right, you will see the proper error message and OP status code.

** For error routines 21 ERASE TRACK, 22 NO DATA, 23 DATCHKSUM, and SYNC WRITER, repeat the same instructions as given above EXCEPT select the EXACT track and sector #s you wish to write on. (No wrap around is needed on sector 0) The SYNC WRITER routine is usually used to write sync pulses to tracks from 35 thru 44. This will change the error returned when reading these areas from sync not found to HEADER NOT FOUND.

READ HEADER

This routine reads the GCR header from a disk and allows you to put it in the computer for analysis or modification. By using this routine and the HDR WRITE you can "SPLICE" bad headers from original disks on to your copy disks. In many cases this is more effective than trying to reproduce the header errors that have been used for protection.

1. Load and run Diskpicker.

2. Select Menu option 2 and printer choice "N".

3. Input L"READ HDR",08 after load, input L"COPY HDR",08

4. After load, input G C020

5. Place diskette to be read into drive.

6. Select menu option 7 then option 3.

7. Input disk start address 0300

8. Input disk end address 034F

9. Input buffer address 4300

10. After data transfer, select menu option 5.

11. Input Job choice 224 and select choice 1 for execute address of 0300. FOR ONE SECTOR, MULTISECTOR='N' FOR ALL SECTORS MULTISECTOR='Y'

12. For header #1, enter track header you wish to copy is on.

13. For header #2, enter sector value ONE LESS than sector value with header you want. ie: Input 0 if sector 1 was choice.
    ** For sectors 0, select last sector on that track!

14. Select menu option 1.

15. Input disk start address of 0400.

16. Input disk end address of 05FF.

17. Input buffer address of 6400.

18. After data transfer, select menu option 2 and desired printer option.

19. Use memory dump command (M) of monitor to display GCR image of the header and data now contained in computer memory locations from 6400 to 65FF. See Appendix F for explanation of GCR image. G C020 will return you to main menu.

COPY HEADER

Writes GCR image of header from disk buffer to object diskette. Useful for ERROR SPLICING.

1. Follow steps 1-19 for Read Hdr routine.

2. Insert disk to be written to in drive. (object disk)

3. Select menu option 3.

4. Input disk start address 0300.

5. Input disk end address 035F.

6. Input buffer address 3300.

7. After transfer of program to disk, select menu option 5.

8. Input Job Choice 224 and select choice 1 for execute address of 0300. FOR ONE SECTOR, MULTISECTOR='N' FOR ALL SECTORS MULTISECTOR='Y'

9. For header #1, enter the track header is to be written to.

10. For header#2, enter sector value ONE LESS than desired sector to be written to. ie: Input 0 if 1 is your choice.

* If choice is 0, use highest sector value on THAT track, as described in other procedures.

11. If everything went right, you can use READ HDR steps 5 thru 19 to look at and verify the new header just written!!

EXTRA NOTES

Remember, Diskpicker is designed to give you control of exactly what is written and sent to the disk. It is a disk development system especially for experimenting with and testing modified DOS routines. Do not confuse its intent with that of "turnkey" error writers which are simple to use but restrict you to what they have decided should be used for errors. Such programs are obsolete shortly after they are sold. With Diskpicker you can concentrate on collecting modified DOS routines which are by nature short and easily traded, and use Diskpicker to transfer and execute these routines. Dedicated error writers generally try to keep you from getting into the program let alone modify and update it.

Below are listed some protection possibilities to look for on original diskettes:

1. Missing or extra sync bytes. Use READ HDR to look for this.
2. Missing header or illegal header ID.
3. False header checksum.
4. Illegal or missing sectors.
5. False ID numbers.
6. Protection data in GAP area.
7. Missing or illegal data block ID byte.
8. Data block missing.
9. False data checksum.
10. Protection data in end of track GAP.

11.Protection data encoded using sync pulse combinations.
*Note: It is possible to achieve some of the above errors on tracks 36 thru 44 as these are readily available using the disk controller software.

Even though protection errors will cause loading problems under normal conditions, it is a fairly simple process to recover useable data from a bad sector. This is done by analyzing the error and writing a short machine program to recover the data. The trick is to sync up on a previous sector's sync pulse and count bytes to the location you wish to read from. When counting bytes, sync pulses show up as ONE byte even though several have occurred. A valid sync pulse does not always show up on a GCR sector read out as an "FF".

The general philosophy for precise error writing is to place the GCR data you wish to record in one of the disk memory buffers, find a valid sync pulse on a nearby preceeding sector, count bytes up to the area you wish to write to, switch the disk controller to its write mode and dump the buffer to the diskette. Switching from read to write is Best accomplished during the gap time when GCR 55's are being read.

The read/write techniques just described will allow a clever programmer to read or write desired information to/from any place on the diskette!! Good luck!!!

*** IMPORTANT: When switching diskettes during Diskpicker procedures, be sure to perform Initialization - Menu option seven. This will insure that the drive page zero memory locations are initialized to the current diskette identification parameters. THIS DOES NOT HAPPEN ON POWER UP!

CREATE HEADER ERRORS

1. For checksum error in header, load 1CON HDR through monitor mode of Diskpicker.

2. If checksum in header error is not desired, load CONHDR through monitor mode.

3. Load WRITE HDR through monitor mode.
   PUT OBJECT DISKETTE IN DRIVE.

4. Select 6 from main menu and enter track# and sector# desired.

5. Select choice 1 - (Transfer disk mem. to buffer)
   Start addr. 0000
   End addr. 00FF
   Buff. addr. 6000

6. Interogate memory locations 6000 -6030 form monitor mode. Refer to Appendix F "Important Disk Memory Locations" to identify byte functions.

7. Change locations to value desired to create errors. Disk ID #s might normally be changed.

8. The GCR image of this hdr can be seen in computer buffer locations 6024-602B. (Which come from disk memory 0024-002B) SELECT 3 FROM MAIN MENU START ADDR 0016 END ADDR 001B BUFF ADDR 6016

9. Select 3 from main menu.
   Start addr. 0300
   End addr. 031F
   Buff. addr. 5300

10. Select 4 from main menu. Entry addr. 0300

11. GCR image is now in disk memory locations 03E0-03E7, ready for transfer to the object disk. Make sure object disk is inserted in drive.

12. Select 3 from main menu.
    Start addr. 0300
    End addr. 036F
    Buff. addr. 3300

13. Select 5 from main menu. Job choice = 224. For execute start addr, select 1 (0300)

14. For Hdr #1, use track desired. For Hdr#2, use ONE LESS than desired. ("wraparound as in earlier procedures). ie: For sector 1 enter a 0.

15. To check errors, follow read header techniques.

1PSIMAIN

```
1 SYS49891:P=0
2 SYS49844
3 SYS49516:X=PEEK(49160):ONXGOTO4,3,5,10
4 T=PEEK(49156):S=PEEK(49157):PRINT#15,"U1:"2;0;T;S:GOSUB25:SYS49691:GOTO3
5 SYS49858:Z=0
6 SYS49939:X=PEEK(49161):ONXGOTO7,6,2,10,5
7 T=PEEK(49158):S=PEEK(49159):SYS49745:IFPEEK(49173)=1ANDZ=1THEN6
8 PRINT#15,"B-P:"2;0:SYS49718:PRINT#15,"U2:"2;0;T;S:GOSUB25:Z=1:GOTO6
10 SYS49872:END
25 PRINT"■■TRK."T"▮▮  ":PRINT"■SEC."S"▮ ":INPUT#15,A$.B$
26 PRINT"■ERR. "A$"TTTTTT":IFA$="00"THENRETURN
27 IFP=1THENOPEN4,4:PRINT#4,T,S,A$,B$:CLOSE4
28 SYS49903:RETURN
```

2PSIMAIN

```
1 SYS49778:P=0
2 SYS49816:M=0
3 SYS49516:X=PEEK(49160):ONXGOTO4,3,5,10
4 T=PEEK(49156):S=PEEK(49157):PRINT#15,"U1:"2;0;T;S:GOSUB25:SYS49691:GOTO3
5 SYS49832:M=1:Z=0
6 SYS49604:X=PEEK(49161):ONXGOTO7,6,2,10,5
7 T=PEEK(49158):S=PEEK(49159):SYS49745:IFPEEK(49173)=1ANDZ=1THEN6
8 PRINT#15,"B-P:"2;0:SYS49718:PRINT#15,"U2:"2;0;T;S:GOSUB25:Z=1:GOTO6
10 SYS49848:END
25 PRINT"■■TRK."T"▮▮  ":PRINT"■SEC."S"▮ ":INPUT#15,A$.B$
26 PRINT"■ERR. "A$"TTTTTT":IFA$="00"THENRETURN
27 IFP=1THENOPEN4,4:PRINT#4,T,S,A$,B$:CLOSE4
28 IFM=0THENSYS49867:RETURN
29 SYS49877:RETURN
```

```
DUPDAC DATA
:C008 00 00 0B 0B 11 F8 11 23
:C010 03 0F 23 08 09 00 00 00
      1DUPDAC
```

```
C018  A9 00     LDA #$00
C01A  85 FB     STA $FB
C01C  85 FD     STA $FD
C01E  A9 0A     LDA #$0A
C020  85 FC     STA $FC
C022  85 FE     STA $FE
C024  A0 00     LDY #$00
C026  A9 00     LDA #$00
C028  99 00 C0  STA $C000,Y
C02B  C8        INY
C02C  C0 0A     CPY #$0A
C02E  D0 F8     BNE $C028
C030  AD 0A C0  LDA $C00A
C033  8D 20 D0  STA $D020
C036  AD 0B C0  LDA $C00B
C039  8D 21 D0  STA $D021
C03C  AD 11 C0  LDA $C011
C03F  8D 18 D4  STA $D418
C042  AD 0C C0  LDA $C00C
C045  8D 05 D4  STA $D405
C048  AD 0D C0  LDA $C00D
C04B  8D 06 D4  STA $D406
C04E  AD 10 C0  LDA $C010
C051  8D 00 D4  STA $D400
C054  AD 0F C0  LDA $C00F
C057  8D 01 D4  STA $D401
C05A  60        RTS
C05B  00        BRK
C05C  00        BRK
C05D  00        BRK
C05E  A9 0F     LDA #$0F
C060  AE 13 C0  LDX $C013
C063  A0 0F     LDY #$0F
C065  20 BA FF  JSR $FFBA
C068  A9 00     LDA #$00
C06A  20 BD FF  JSR $FFBD
C06D  20 C0 FF  JSR $FFC0
C070  A9 02     LDA #$02
C072  AE 13 C0  LDX $C013
C075  A0 02     LDY #$02
C077  20 BA FF  JSR $FFBA
C07A  A9 01     LDA #$01
C07C  A2 12     LDX #$12
C07E  A0 C0     LDY #$C0
C080  20 BD FF  JSR $FFBD
C083  20 C0 FF  JSR $FFC0
C086  60        RTS
C087  00        BRK
C088  00        BRK
C089  00        BRK
C08A  00        BRK
C08B  A9 0A     LDA #$0A
C08D  85 FC     STA $FC
C08F  85 FE     STA $FE
C091  A9 02     LDA #$02
C093  20 C3 FF  JSR $FFC3
C096  A9 0F     LDA #$0F
C098  20 C3 FF  JSR $FFC3
C09B  60        RTS
C09C  00        BRK
C09D  00        BRK
C09E  00        BRK
C09F  00        BRK
C0A0  A9 0F     LDA #$0F
C0A2  AE 14 C0  LDX $C014
C0A5  A0 0F     LDY #$0F
C0A7  20 BA FF  JSR $FFBA
C0AA  A9 00     LDA #$00
C0AC  20 BD FF  JSR $FFBD
C0AF  20 C0 FF  JSR $FFC0
C0B2  A9 02     LDA #$02
C0B4  AE 14 C0  LDX $C014
C0B7  A0 02     LDY #$02
C0B9  20 BA FF  JSR $FFBA
C0BC  A9 01     LDA #$01
C0BE  A2 12     LDX #$12
C0C0  A0 C0     LDY #$C0
C0C2  20 BD FF  JSR $FFBD
C0C5  20 C0 FF  JSR $FFC0
C0C8  60        RTS
C0C9  00        BRK
C0CA  00        BRK
C0CB  00        BRK
C0CC  00        BRK
C0CD  18        CLC
C0CE  A2 00     LDX #$00
C0D0  BD F0 CD  LDA $CDF0,X
C0D3  C9 04     CMP #$04
C0D5  F0 06     BEQ $C0DD
C0D7  20 D2 FF  JSR $FFD2
C0DA  E8        INX
C0DB  D0 F3     BNE $C0D0
C0DD  60        RTS
C0DE  00        BRK
C0DF  00        BRK
C0E0  18        CLC
C0E1  A2 00     LDX #$00
C0E3  BD 8C CE  LDA $CE8C,X
C0E6  C9 04     CMP #$04
C0E8  F0 06     BEQ $C0F0
C0EA  20 D2 FF  JSR $FFD2
C0ED  E8        INX
C0EE  D0 F3     BNE $C0E3
C0F0  60        RTS
C0F1  18        CLC
C0F2  A2 00     LDX #$00
C0F4  BD E0 CE  LDA $CEE0,X
```

1DUPDAC -

```
C0F7   C9 04      CMP #$04
C0F9   F0 06      BEQ $C101
C0FB   20 D2 FF   JSR $FFD2
C0FE   E8         INX
C0FF   D0 F3      BNE $C0F4
C101   20 E4 FF   JSR $FFE4
C104   C9 88      CMP #$88
C106   D0 F9      BNE $C101
C108   60         RTS
C109   00         BRK
C10A   00         BRK
C10B   AE 0F C0   LDX $C00F
C10E   8E 04 D4   STX $D404
C111   CA         DEX
C112   8E 04 D4   STX $D404
C115   60         RTS
C116   00         BRK
C117   00         BRK
C118   A9 04      LDA #$04
C11A   8D 79 CE   STA $CE79
C11D   60         RTS
C11E   00         BRK
C11F   00         BRK
C120   18         CLC
C121   A2 00      LDX #$00
C123   BD B8 CE   LDA $CEB8,X
C126   C9 04      CMP #$04
C128   F0 06      BEQ $C130
C12A   20 D2 FF   JSR $FFD2
C12D   E8         INX
C12E   D0 F3      BNE $C123
C130   60         RTS
C131   00         BRK
C132   00         BRK
C133   18         CLC
C134   A2 00      LDX #$00
C136   BD F5 CE   LDA $CEF5,X
C139   C9 04      CMP #$04
C13B   F0 06      BEQ $C143
C13D   20 D2 FF   JSR $FFD2
C140   E8         INX
C141   D0 F3      BNE $C136
C143   60         RTS
C144   00         BRK
C145   00         BRK
C146   18         CLC
C147   A2 00      LDX #$00
C149   BD 13 CF   LDA $CF13,X
C14C   C9 04      CMP #$04
C14E   F0 06      BEQ $C156
C150   20 D2 FF   JSR $FFD2
C153   E8         INX
C154   D0 F3      BNE $C149
C156   60         RTS
C157   00         BRK
C158   00         BRK
C159   18         CLC
C15A   A2 00      LDX #$00
C15C   BD 32 CF   LDA $CF32,X
C15F   C9 04      CMP #$04
```

```
C161   F0 06      BEQ $C169
C163   20 D2 FF   JSR $FFD2
C166   E8         INX
C167   D0 F3      BNE $C15C
C169   60         RTS
C16A   00         BRK
C16B   00         BRK
C16C   18         CLC
C16D   AE 08 C0   LDX $C008
C170   E0 01      CPX #$01
C172   F0 38      BEQ $C1AC
C174   AC 00 C0   LDY $C000
C177   B9 76 CF   LDA $CF76,Y
C17A   C9 BB      CMP #$BB
C17C   D0 06      BNE $C184
C17E   A2 04      LDX #$04
C180   8E 08 C0   STX $C008
C183   60         RTS
C184   C9 FF      CMP #$FF
C186   D0 0A      BNE $C192
C188   A2 03      LDX #$03
C18A   8E 08 C0   STX $C008
C18D   C8         INY
C18E   8C 00 C0   STY $C000
C191   60         RTS
C192   A2 01      LDX #$01
C194   8E 08 C0   STX $C008
C197   8D 04 C0   STA $C004
C19A   C8         INY
C19B   B9 76 CF   LDA $CF76,Y
C19E   8D 05 C0   STA $C005
C1A1   C8         INY
C1A2   B9 76 CF   LDA $CF76,Y
C1A5   8D 02 C0   STA $C002
C1A8   C8         INY
C1A9   8C 00 C0   STY $C000
C1AC   AD 05 C0   LDA $C005
C1AF   CD 02 C0   CMP $C002
C1B2   D0 05      BNE $C1B9
C1B4   A2 02      LDX #$02
C1B6   8E 08 C0   STX $C008
C1B9   AE 0F C0   LDX $C00F
C1BC   20 0B C1   JSR $C10B
C1BF   60         RTS
C1C0   00         BRK
C1C1   00         BRK
C1C2   00         BRK
C1C3   00         BRK
C1C4   18         CLC
C1C5   AE 09 C0   LDX $C009
C1C8   E0 01      CPX #$01
C1CA   F0 37      BEQ $C203
C1CC   AC 01 C0   LDY $C001
C1CF   B9 76 CF   LDA $CF76,Y
C1D2   C9 FF      CMP #$FF
C1D4   D0 13      BNE $C1E9
C1D6   A2 03      LDX #$03
C1D8   C8         INY
C1D9   B9 76 CF   LDA $CF76,Y
C1DC   C9 BB      CMP #$BB
```

```
1DUPDAC --

C1DE   D0 02     BNE $C1E2
C1E0   A2 04     LDX #$04
C1E2   8E 09 C0  STX $C009
C1E5   8C 01 C0  STY $C001
C1E8   60        RTS
C1E9   A2 01     LDX #$01
C1EB   8E 09 C0  STX $C009
C1EE   8D 06 C0  STA $C006
C1F1   C8        INY
C1F2   B9 76 CF  LDA $CF76,Y
C1F5   8D 07 C0  STA $C007
C1F8   C8        INY
C1F9   B9 76 CF  LDA $CF76,Y
C1FC   8D 03 C0  STA $C003
C1FF   C8        INY
C200   8C 01 C0  STY $C001
C203   AD 07 C0  LDA $C007
C206   CD 03 C0  CMP $C003
C209   D0 05     BNE $C210
C20B   A2 02     LDX #$02
C20D   8E 09 C0  STX $C009
C210   AE 0F C0  LDX $C00F
C213   20 0B C1  JSR $C10B
C216   60        RTS
C217   00        BRK
C218   00        BRK
C219   00        BRK
C21A   00        BRK
C21B   18        CLC
C21C   A2 02     LDX #$02
C21E   20 C6 FF  JSR $FFC6
C221   A0 00     LDY #$00
C223   20 A5 FF  JSR $FFA5
C226   91 FB     STA ($FB),Y
C228   C8        INY
C229   D0 F8     BNE $C223
C22B   E6 FC     INC $FC
C22D   20 CC FF  JSR $FFCC
C230   EE 05 C0  INC $C005
C233   60        RTS
C234   00        BRK
C235   00        BRK
C236   18        CLC
C237   A2 02     LDX #$02
C239   20 C9 FF  JSR $FFC9
C23C   A0 00     LDY #$00
C23E   B1 FD     LDA ($FD),Y
C240   20 A8 FF  JSR $FFA8
C243   C8        INY
C244   D0 F8     BNE $C23E
C246   E6 FE     INC $FE
C248   20 CC FF  JSR $FFCC
C24B   EE 07 C0  INC $C007
C24E   60        RTS
C24F   00        BRK
C250   00        BRK
C251   18        CLC
C252   A0 00     LDY #$00
C254   B1 FD     LDA ($FD),Y
C256   C9 00     CMP #$00
C258   D0 0E     BNE $C268

C25A   C8        INY
C25B   D0 F7     BNE $C254
C25D   A2 01     LDX #$01
C25F   8E 15 C0  STX $C015
C262   EE 07 C0  INC $C007
C265   E6 FE     INC $FE
C267   60        RTS
C268   A2 00     LDX #$00
C26A   8E 15 C0  STX $C015
C26D   60        RTS
C26E   00        BRK
C26F   00        BRK
C270   00        BRK
C271   00        BRK
C272   20 CD C0  JSR $C0CD
C275   20 E0 C0  JSR $C0E0
C278   20 F1 C0  JSR $C0F1
C27B   20 0B C1  JSR $C10B
C27E   20 18 C1  JSR $C118
C281   20 CD C0  JSR $C0CD
C284   20 33 C1  JSR $C133
C287   60        RTS
C288   00        BRK
C289   00        BRK
C28A   00        BRK
C28B   00        BRK
C28C   20 CD C0  JSR $C0CD
C28F   20 20 C1  JSR $C120
C292   20 F1 C0  JSR $C0F1
C295   20 0B C1  JSR $C10B
C298   20 CD C0  JSR $C0CD
C29B   20 46 C1  JSR $C146
C29E   60        RTS
C29F   00        BRK
C2A0   00        BRK
C2A1   00        BRK
C2A2   00        BRK
C2A3   20 CD C0  JSR $C0CD
C2A6   20 59 C1  JSR $C159
C2A9   20 F1 C0  JSR $C0F1
C2AC   20 0B C1  JSR $C10B
C2AF   60        RTS
C2B0   00        BRK
C2B1   00        BRK
C2B2   00        BRK
C2B3   00        BRK
C2B4   20 8B C0  JSR $C08B
C2B7   20 72 C2  JSR $C272
C2BA   20 5E C0  JSR $C05E
C2BD   20 FB C2  JSR $C2FB
C2C0   60        RTS
C2C1   00        BRK
C2C2   20 8B C0  JSR $C08B
C2C5   20 8C C2  JSR $C28C
C2C8   20 5E C0  JSR $C05E
C2CB   20 03 C3  JSR $C303
C2CE   60        RTS
C2CF   00        BRK
C2D0   20 8B C0  JSR $C08B
```

```
         1DUPDAC ---
C2D3  20 A3 C2 JSR $C2A3
C2D6  60       RTS
C2D7  00       BRK
C2D8  00       BRK
C2D9  20 8B C0 JSR $C08B
C2DC  20 5E C0 JSR $C05E
C2DF  60       RTS
C2E0  00       BRK
C2E1  00       BRK
C2E2  00       BRK
C2E3  A9 0A    LDA #$0A
C2E5  85 34    STA $34
C2E7  85 38    STA $38
C2E9  20 18 C0 JSR $C018
C2EC  60       RTS
C2ED  00       BRK
C2EE  00       BRK
C2EF  20 8B C0 JSR $C08B
C2F2  20 01 C1 JSR $C101
C2F5  20 5E C0 JSR $C05E
C2F8  60       RTS
C2F9  00       BRK
C2FA  00       BRK
C2FB  A9 00    LDA #$00
C2FD  8D 17 C0 STA $C017
C300  60       RTS
C301  00       BRK
C302  00       BRK
C303  A0 00    LDY #$00
C305  B9 00 C0 LDA $C000,Y
C308  99 3C 03 STA $033C,Y
C30B  C8       INY
C30C  C0 09    CPY #$09
C30E  D0 F5    BNE $C305
C310  60       RTS
C311  00       BRK
C312  00       BRK
C313  20 C4 C1 JSR $C1C4
C316  18       CLC
C317  AD 17 C0 LDA $C017
C31A  CD 16 C0 CMP $C016
C31D  F0 24    BEQ $C343
C31F  AD 09 C0 LDA $C009
C322  C9 01    CMP #$01
C324  F0 1D    BEQ $C343
C326  C9 02    CMP #$02
C328  F0 19    BEQ $C343
C32A  EE 17 C0 INC $C017
C32D  A0 00    LDY #$00
C32F  B9 3C 03 LDA $033C,Y
C332  99 00 C0 STA $C000,Y
C335  C8       INY
C336  C0 09    CPY #$09
C338  D0 F5    BNE $C32F
C33A  A2 05    LDX #$05
C33C  8E 09 C0 STX $C009
C33F  A9 0A    LDA #$0A
C341  85 38    STA $38
C343  60       RTS
```

```
CDF0 93 20 20 20 20 20 20 20
CDF8 20 05 72 72 72 72 72 72
CE00 72 72 72 72 72 72 72 72
CE08 72 72 72 72 72 72 72 72
CE10 72 72 72 0D 20 20 20 20
CE18 20 20 20 20 05 43 2D 36
CE20 34 20 44 49 53 4B 20 42
CE28 41 43 4B 2D 55 50 20 31
CE30 44 55 50 44 41 43 0D 20
CE38 20 20 20 20 20 20 20 05
CE40 A3 A3 A3 A3 A3 A3 A3 A3
CE48 A3 A3 A3 A3 A3 A3 A3 A3
CE50 A3 A3 A3 A3 A3 A3 A3 A3
CE58 A3 0D 20 20 20 20 20 20
CE60 20 20 20 20 20 20 20 1E
CE68 28 43 29 20 50 53 49 44
CE70 41 43 20 9E 31 39 38 33
CE78 0D 04 9F 9F 42 59 20 56
CE80 49 43 20 4E 55 4D 42 45
CE88 52 53 0D 04 11 11 11 11
CE90 9E 2A 20 2A 20 49 4E 53
CE98 45 52 54 20 4F 52 49 47
CEA0 49 4E 41 4C 20 44 49 53
CEA8 4B 20 49 4E 20 44 52 49
CEB0 56 45 20 2A 20 2A 0D 04
CEB8 11 11 11 11 05 2A 20 2A
CEC0 20 49 4E 53 45 52 54 20
CEC8 43 4F 50 59 20 44 49 53
CED0 4B 20 49 4E 20 44 52 49
CED8 56 45 20 2A 20 2A 0D 04
CEE0 11 11 9F 2A 20 2A 20 50
CEE8 52 45 53 53 20 46 37 20
CEF0 2A 20 2A 0D 04 11 11 9E
CEF8 2A 20 2A 20 52 45 41 44
CF00 20 4D 4F 44 45 20 45 4E
CF08 41 42 4C 45 44 20 2A 20
CF10 2A 0D 04 11 11 05 2A 20
CF18 2A 20 57 52 49 54 45 20
CF20 4D 4F 44 45 20 45 4E 41
CF28 42 4C 45 44 20 2A 20 2A
CF30 0D 04 11 11 9F 9F 2A 20
CF38 2A 20 44 55 50 4C 49 43
CF40 41 54 49 4F 4E 20 43 4F
CF48 4D 50 4C 45 54 45 20 2A
CF50 20 2A 0D 11 11 2A 20 2A
CF58 20 52 45 4D 4F 56 45 20
CF60 44 49 53 43 20 2A 20 2A
CF68 0D 04 00 00 00 00 00 00
CF70 00 00 00 00 00 00 01 00
CF78 15 02 00 15 03 00 15 04
CF80 00 15 05 00 15 06 00 15
CF88 07 00 15 08 00 03 FF 08
CF90 03 15 09 00 15 0A 00 15
CF98 0B 00 15 0C 00 15 0D 00
CFA0 15 0E 00 15 0F 00 06 FF
CFA8 0F 06 15 10 00 15 11 00
CFB0 15 12 00 13 13 00 13 14
CFB8 00 13 15 00 13 16 00 11
CFC0 FF 16 11 13 17 00 13 18
CFC8 00 13 19 00 12 1A 00 12
CFD0 1B 00 12 1C 00 12 1D 00
CFD8 12 1E 00 12 1F 00 02 FF
CFE0 1F 02 11 20 00 11 21 00
CFE8 11 22 00 11 23 00 11 FF
CFF0 BB 00 00 00 00 00 00 00
CFF8 00 00 00 00 00 00 00 81
```

```
DUPDAC DATA
:C008 00 00 0B 0B 11 F8 11 23
:C010 03 0F 23 08 09 00 00 00
 C018   A9 00     LDA #$00
 C01A   85 FB     STA $FB
 C01C   85 FD     STA $FD
 C01E   A9 0A     LDA #$0A
 C020   85 FC     STA $FC
 C022   85 FE     STA $FE
 C024   A0 00     LDY #$00
 C026   A9 00     LDA #$00
 C028   99 00 C0  STA $C000,Y
 C02B   C8        INY
 C02C   C0 0A     CPY #$0A
 C02E   D0 F8     BNE $C028
 C030   AD 0A C0  LDA $C00A
 C033   8D 20 D0  STA $D020
 C036   AD 0B C0  LDA $C00B
 C039   8D 21 D0  STA $D021
 C03C   AD 11 C0  LDA $C011
 C03F   8D 18 D4  STA $D418
 C042   AD 0C C0  LDA $C00C
 C045   8D 05 D4  STA $D405
 C048   AD 0D C0  LDA $C00D
 C04B   8D 06 D4  STA $D406
 C04E   AD 10 C0  LDA $C010
 C051   8D 00 D4  STA $D400
 C054   AD 0F C0  LDA $C00F
 C057   8D 01 D4  STA $D401
 C05A   60        RTS
 C05B   00        BRK
 C05C   00        BRK
 C05D   00        BRK
 C05E   A9 0F     LDA #$0F
 C060   AE 13 C0  LDX $C013
 C063   A0 0F     LDY #$0F
 C065   20 BA FF  JSR $FFBA
 C068   A9 00     LDA #$00
 C06A   20 BD FF  JSR $FFBD
 C06D   20 C0 FF  JSR $FFC0
 C070   A9 02     LDA #$02
 C072   AE 13 C0  LDX $C013
 C075   A0 02     LDY #$02
 C077   20 BA FF  JSR $FFBA
 C07A   A9 01     LDA #$01
 C07C   A2 12     LDX #$12
 C07E   A0 C0     LDY #$C0
 C080   20 BD FF  JSR $FFBD
 C083   20 C0 FF  JSR $FFC0
 C086   60        RTS
 C087   00        BRK
 C088   00        BRK
 C089   00        BRK
 C08A   00        BRK
 C08B   A9 0A     LDA #$0A
 C08D   85 FC     STA $FC
 C08F   85 FE     STA $FE
```

```
2DUPDAC

C091   A9 02     LDA #$02
C093   20 C3 FF  JSR $FFC3
C096   A9 0F     LDA #$0F
C098   20 C3 FF  JSR $FFC3
C09B   60        RTS
C09C   00        BRK
C09D   00        BRK
C09E   00        BRK
C09F   00        BRK
C0A0   A9 0F     LDA #$0F
C0A2   AE 14 C0  LDX $C014
C0A5   A0 0F     LDY #$0F
C0A7   20 BA FF  JSR $FFBA
C0AA   A9 00     LDA #$00
C0AC   20 BD FF  JSR $FFBD
C0AF   20 C0 FF  JSR $FFC0
C0B2   A9 02     LDA #$02
C0B4   AE 14 C0  LDX $C014
C0B7   A0 02     LDY #$02
C0B9   20 BA FF  JSR $FFBA
C0BC   A9 01     LDA #$01
C0BE   A2 12     LDX #$12
C0C0   A0 C0     LDY #$C0
C0C2   20 BD FF  JSR $FFBD
C0C5   20 C0 FF  JSR $FFC0
C0C8   60        RTS
C0C9   00        BRK
C0CA   00        BRK
C0CB   00        BRK
C0CC   00        BRK
C0CD   18        CLC
C0CE   A2 00     LDX #$00
C0D0   BD F0 CD  LDA $CDF0,X
C0D3   C9 04     CMP #$04
C0D5   F0 06     BEQ $C0DD
C0D7   20 D2 FF  JSR $FFD2
C0DA   E8        INX
C0DB   D0 F3     BNE $C0D0
C0DD   60        RTS
C0DE   00        BRK
C0DF   00        BRK
C0E0   18        CLC
C0E1   A2 00     LDX #$00
C0E3   BD 8C CE  LDA $CE8C,X
C0E6   C9 04     CMP #$04
C0E8   F0 06     BEQ $C0F0
C0EA   20 D2 FF  JSR $FFD2
C0ED   E8        INX
C0EE   D0 F3     BNE $C0E3
C0F0   60        RTS
C0F1   18        CLC
C0F2   A2 00     LDX #$00
C0F4   BD E0 CE  LDA $CEE0,X
C0F7   C9 04     CMP #$04
```

```
C0F9   F0 06     BEQ $C101
C0FB   20 D2 FF  JSR $FFD2
C0FE   E8        INX
C0FF   D0 F3     BNE $C0F4
C101   20 E4 FF  JSR $FFE4
C104   C9 88     CMP #$88
C106   D0 F9     BNE $C101
C108   60        RTS
C109   00        BRK
C10A   00        BRK
C10B   AE 0F C0  LDX $C00F
C10E   8E 04 D4  STX $D404
C111   CA        DEX
C112   8E 04 D4  STX $D404
C115   60        RTS
C116   00        BRK
C117   00        BRK
C118   A9 04     LDA #$04
C11A   8D 79 CE  STA $CE79
C11D   60        RTS
C11E   00        BRK
C11F   00        BRK
C120   18        CLC
C121   A2 00     LDX #$00
C123   BD B8 CE  LDA $CEB8,X
C126   C9 04     CMP #$04
C128   F0 06     BEQ $C130
C12A   20 D2 FF  JSR $FFD2
C12D   E8        INX
C12E   D0 F3     BNE $C123
C130   60        RTS
C131   00        BRK
C132   00        BRK
C133   18        CLC
C134   A2 00     LDX #$00
C136   BD F5 CE  LDA $CEF5,X
C139   C9 04     CMP #$04
C13B   F0 06     BEQ $C143
C13D   20 D2 FF  JSR $FFD2
C140   E8        INX
C141   D0 F3     BNE $C136
C143   60        RTS
C144   00        BRK
C145   00        BRK
C146   18        CLC
C147   A2 00     LDX #$00
C149   BD 13 CF  LDA $CF13,X
C14C   C9 04     CMP #$04
C14E   F0 06     BEQ $C156
C150   20 D2 FF  JSR $FFD2
C153   E8        INX
C154   D0 F3     BNE $C149
C156   60        RTS
C157   00        BRK
C158   00        BRK
C159   18        CLC
C15A   A2 00     LDX #$00
C15C   BD 32 CF  LDA $CF32,X
C15F   C9 04     CMP #$04
C161   F0 06     BEQ $C169
C163   20 D2 FF  JSR $FFD2
C166   E8        INX
C167   D0 F3     BNE $C15C
C169   60        RTS
C16A   00        BRK
C16B   00        BRK
C16C   18        CLC
C16D   AE 08 C0  LDX $C008
C170   E0 01     CPX #$01
C172   F0 38     BEQ $C1AC
C174   AC 00 C0  LDY $C000
C177   B9 76 CF  LDA $CF76,Y
C17A   C9 BB     CMP #$BB
C17C   D0 06     BNE $C184
C17E   A2 04     LDX #$04
C180   8E 08 C0  STX $C008
C183   60        RTS
C184   C9 FF     CMP #$FF
C186   D0 0A     BNE $C192
C188   A2 03     LDX #$03
C18A   8E 08 C0  STX $C008
C18D   C8        INY
C18E   8C 00 C0  STY $C000
C191   60        RTS
C192   A2 01     LDX #$01
C194   8E 08 C0  STX $C008
C197   8D 04 C0  STA $C004
C19A   C8        INY
C19B   B9 76 CF  LDA $CF76,Y
C19E   8D 05 C0  STA $C005
C1A1   C8        INY
C1A2   B9 76 CF  LDA $CF76,Y
C1A5   8D 02 C0  STA $C002
C1A8   C8        INY
C1A9   8C 00 C0  STY $C000
C1AC   AD 05 C0  LDA $C005
C1AF   CD 02 C0  CMP $C002
C1B2   D0 05     BNE $C1B9
C1B4   A2 02     LDX #$02
C1B6   8E 08 C0  STX $C008
C1B9   AE 0F C0  LDX $C00F
C1BC   20 0B C1  JSR $C10B
C1BF   60        RTS
C1C0   00        BRK
C1C1   00        BRK
C1C2   00        BRK
C1C3   00        BRK
C1C4   18        CLC
```

2DUPDAC --

```
C1C5   AE 09 C0 LDX $C009
C1C8   E0 01    CPX #$01
C1CA   F0 37    BEQ $C203
C1CC   AC 01 C0 LDY $C001
C1CF   B9 76 CF LDA $CF76,Y
C1D2   C9 FF    CMP #$FF
C1D4   D0 13    BNE $C1E9
C1D6   A2 03    LDX #$03
C1D8   C8       INY
C1D9   B9 76 CF LDA $CF76,Y
C1DC   C9 BB    CMP #$BB
C1DE   D0 02    BNE $C1E2
C1E0   A2 04    LDX #$04
C1E2   8E 09 C0 STX $C009
C1E5   8C 01 C0 STY $C001
C1E8   60       RTS
C1E9   A2 01    LDX #$01
C1EB   8E 09 C0 STX $C009
C1EE   8D 06 C0 STA $C006
C1F1   C8       INY
C1F2   B9 76 CF LDA $CF76,Y
C1F5   8D 07 C0 STA $C007
C1F8   C8       INY
C1F9   B9 76 CF LDA $CF76,Y
C1FC   8D 03 C0 STA $C003
C1FF   C8       INY
C200   8C 01 C0 STY $C001
C203   AD 07 C0 LDA $C007
C206   CD 03 C0 CMP $C003
C209   D0 05    BNE $C210
C20B   A2 02    LDX #$02
C20D   8E 09 C0 STX $C009
C210   AE 0F C0 LDX $C00F
C213   20 0B C1 JSR $C10B
C216   60       RTS
C217   00       BRK
C218   00       BRK
C219   00       BRK
C21A   00       BRK
C21B   18       CLC
C21C   A2 02    LDX #$02
C21E   20 C6 FF JSR $FFC6
C221   A0 00    LDY #$00
C223   20 A5 FF JSR $FFA5
C226   91 FB    STA ($FB),Y
C228   C8       INY
C229   D0 F8    BNE $C223
C22B   E6 FC    INC $FC
C22D   20 CC FF JSR $FFCC
C230   EE 05 C0 INC $C005
C233   60       RTS
C234   00       BRK
C235   00       BRK
C236   18       CLC
C237   A2 02    LDX #$02
C239   20 C9 FF JSR $FFC9
C23C   A0 00    LDY #$00
C23E   B1 FD    LDA ($FD),Y
C240   20 A8 FF JSR $FFA8
C243   C8       INY
C244   D0 F8    BNE $C23E
C246   E6 FE    INC $FE
C248   20 CC FF JSR $FFCC
C24B   EE 07 C0 INC $C007
C24E   60       RTS
C24F   00       BRK
C250   00       BRK
C251   18       CLC
C252   A0 00    LDY #$00
C254   B1 FD    LDA ($FD),Y
C256   C9 00    CMP #$00
C258   D0 0E    BNE $C268
C25A   C8       INY
C25B   D0 F7    BNE $C254
C25D   A2 01    LDX #$01
C25F   8E 15 C0 STX $C015
C262   EE 07 C0 INC $C007
C265   E6 FE    INC $FE
C267   60       RTS
C268   A2 00    LDX #$00
C26A   8E 15 C0 STX $C015
C26D   60       RTS
C26E   00       BRK
C26F   00       BRK
C270   00       BRK
C271   00       BRK
C272   A9 0A    LDA #$0A
C274   85 34    STA $34
C276   85 38    STA $38
C278   20 18 C0 JSR $C018
C27B   20 CD C0 JSR $C0CD
C27E   20 E0 C0 JSR $C0E0
C281   20 F1 C0 JSR $C0F1
C284   20 0B C1 JSR $C10B
C287   20 18 C1 JSR $C118
C28A   20 CD C0 JSR $C0CD
C28D   20 20 C1 JSR $C120
C290   20 F1 C0 JSR $C0F1
C293   20 0B C1 JSR $C10B
C296   60       RTS
C297   00       BRK
C298   20 8B C0 JSR $C08B
C29B   20 CD C0 JSR $C0CD
C29E   20 33 C1 JSR $C133
C2A1   20 5E C0 JSR $C05E
```

2DUPDAC ---

```
C2A4          60 RTS
C2A5  00         BRK
C2A6  00         BRK
C2A7  00         BRK
C2A8  20 8B C0 JSR $C08B
C2AB  20 CD C0 JSR $C0CD
C2AE  20 46 C1 JSR $C146
C2B1  20 A0 C0 JSR $C0A0
C2B4  60       RTS
C2B5  00       BRK
C2B6  00       BRK
C2B7  00       BRK
C2B8  20 8B C0 JSR $C08B
C2BB  20 CD C0 JSR $C0CD
C2BE  20 59 C1 JSR $C159
C2C1  20 F1 C0 JSR $C0F1
C2C4  20 0B C1 JSR $C10B
C2C7  60       RTS
C2C8  00       BRK
C2C9  00       BRK
C2CA  00       BRK
C2CB  20 8B C0 JSR $C08B
C2CE  20 5E C0 JSR $C05E
C2D1  60       RTS
C2D2  00       BRK
C2D3  00       BRK
C2D4  00       BRK
C2D5  20 8B C0 JSR $C08B
C2D8  20 A0 C0 JSR $C0A0
C2DB  60       RTS
```

DATA TABLE

```
CDF0 93 20 20 20 20 20 20 20
CDF8 20 05 72 72 72 72 72 72
CE00 72 72 72 72 72 72 72 72
CE08 72 72 72 72 72 72 72 72
CE10 72 72 72 0D 20 20 20 20
CE18 20 20 20 20 05 43 2D 36
CE20 34 20 44 49 53 4B 20 42
CE28 41 43 4B 2D 55 50 20 32
CE30 44 55 50 44 41 43 0D 20
CE38 20 20 20 20 20 20 20 05
CE40 A3 A3 A3 A3 A3 A3 A3 A3
CE48 A3 A3 A3 A3 A3 A3 A3 A3
CE50 A3 A3 A3 A3 A3 A3 A3 A3
CE58 A3 0D 20 20 20 20 20 20
CE60 20 20 20 20 20 20 20 1E
CE68 28 43 29 20 50 53 49 44
CE70 41 43 20 9E 31 39 38 33
CE78 0D 11 9F 9F 42 59 20 56
CE80 49 43 20 4E 55 4D 42 45
CE88 52 53 0D 04 11 11 11 11
CE90 9E 2A 20 2A 20 49 4E 53
CE98 45 52 54 20 4F 52 49 47
CEA0 49 4E 41 4C 20 44 49 53
CEA8 4B 20 49 4E 20 44 52 49
CEB0 56 45 20 2A 20 2A 0D 04
CEB8 11 11 11 11 05 2A 20 2A
CEC0 20 49 4E 53 45 52 54 20
CEC8 43 4F 50 59 20 44 49 53
CED0 4B 20 49 4E 20 44 52 49
CED8 56 45 20 2A 20 2A 0D 04
CEE0 11 11 9F 2A 20 2A 20 50
CEE8 52 45 53 53 20 46 37 20
CEF0 2A 20 2A 0D 04 11 11 9E
CEF8 2A 20 2A 20 52 45 41 44
CF00 20 4D 4F 44 45 20 45 4E
CF08 41 42 4C 45 44 20 2A 20
CF10 2A 0D 04 11 11 05 2A 20
CF18 2A 20 57 52 49 54 45 20
CF20 4D 4F 44 45 20 45 4E 41
CF28 42 4C 45 44 20 2A 20 2A
CF30 0D 04 11 11 9F 9F 2A 20
CF38 2A 20 44 55 50 4C 49 43
CF40 41 54 49 4F 4E 20 43 4F
CF48 4D 50 4C 45 54 45 20 2A
CF50 20 2A 0D 11 11 2A 20 2A
CF58 20 52 45 4D 4F 56 45 20
CF60 44 49 53 43 20 2A 20 2A
CF68 0D 04 00 00 00 00 00 00
CF70 00 00 00 00 00 00 01 00
CF78 15 02 00 15 03 00 15 04
CF80 00 15 05 00 15 06 00 15
CF88 07 00 15 08 00 03 FF 08
CF90 03 15 09 00 15 0A 00 15
CF98 0B 00 15 0C 00 15 0D 00
CFA0 15 0E 00 15 0F 00 06 FF
CFA8 0F 06 15 10 00 15 11 00
CFB0 15 12 00 13 13 00 13 14
CFB8 00 13 15 00 13 16 00 11
CFC0 FF 16 11 13 17 00 13 18
CFC8 00 13 19 00 12 1A 00 12
CFD0 1B 00 12 1C 00 12 1D 00
CFD8 12 1E 00 12 1F 00 02 FF
CFE0 1F 02 11 20 00 11 21 00
CFE8 11 22 00 11 23 00 11 FF
CFF0 BB 00 00 00 00 00 00 00
CFF8 00 00 00 00 00 00 00 00
```

```
5 IFA=0THENA=1:LOAD"ADMACH",8,1
10 POKE53280,11:POKE53281,11:TR=18:SE=1:DIMA(300):DIMB(200)
15 HX$="0123456789ABCDEF"
18 P=0:INPUT"⊐≡PRINTER Y/N";YN$:IFYN$="Y"THENP=1
20 PRINT"⊐▮      SUPERDIRECTORY▮ (C)84 PSIDAC VBN≡"
25 IFP=1THENOPEN4,4:PRINT#4:PRINT#4,CHR$(16)"15SUPERDIRECTORY":PRINT#4:CLOSE4
30 PRINT"▮INSERT SOURCE DISK IN DRIVE"
35 INPUT"▮DISK NAME";DN$
40 IFPD=1THEN50
41 PRINT"▮▮PRESS F7"
42 GETA$:IFA$<>"▮"THEN42
45 IFP=1THENOPEN4,4:PRINT#4,"DISK NAME = "DN$:PRINT#4:CLOSE4
50 OPEN15,8,15:OPEN2,8,2,"#":PRINT#15,"U1:"2;0;TR;SE
60 SYS49152
70 CLOSE2:CLOSE15
80 FORRP=0TO255
100 A(RP)=PEEK(RP+52992)
120 NEXT
125 TR=A(0):SE=A(1)
130 PRINT"⊐▮TYPE TRACK SECTOR NAME                BLKS▮"
132 IFPD=1THENOPEN4,4:GOTO138
133 IFP<>1THEN136
134 OPEN4,4:PRINT#4,"TYPE   TRACK   SECTOR      NAME                 BLKS":PRINT#4
136 IFP=1THENPD=1
138 FORRP=2TO226STEP32:OK=1:FT$="    "
139 TK=A(RP+1):SK=A(RP+2):NB=A(RP+28):IFTK=0THENCLOSE4:GOTO200
140 IFA(RP)=0THENFT$="DEL":OK=1
145 IFA(RP)=130THENFT$="PRG":OK=1
150 IFA(RP)=132THENFT$="REL":OK=0
155 IFA(RP)=129THENFT$="SEQ":OK=0
160 PRINTFT$;
165 IFP=1THENPRINT#4,FT$;
172 PRINTTAB(5)TK;TAB(12)SK;
175 IFP=1THENPRINT#4,CHR$(16)"07"TK;CHR$(16)"16"SK;CHR$(16)"27";
178 IFOK=1THENB(C)=TK:B(C+1)=SK:C=C+2
180 PRINTTAB(18);:FORTX=3TO18:PRINTCHR$(A(RP+TX));:NEXT:PRINTTAB(34)NB
185 IFP=1THENFORTX=3TO18:PRINT#4,CHR$(A(RP+TX));:NEXT:PRINT#4,CHR$(16)"34"NB
190 NEXT:CLOSE4
200 IFA(0)<>0THENGOTO40
300 PRINT"▮PRESS F7▮"
305 GETA$:IFA$<>"▮"THEN305
310 PRINT"⊐TRACK   SECTOR   HEX.ADD      DEC.ADD":PRINT
315 IFP<>1THEN320
318 OPEN4,4:PRINT#4:PRINT#4,"TRACK  SECTOR  HEX.ADD     DEC.ADD":PRINT#4
319 CLOSE4
320 PC=1:FORRP=0TOC-2STEP2
330 TR=B(RP):SE=B(RP+1)
340 OPEN15,8,15:OPEN2,8,2,"#":PRINT#15,"U1:"2;0;TR;SE
350 SYS49152
360 CLOSE2:CLOSE15:PC=PC+1
364 IFP=1THEN370
365 IFPC<>18GOTO370
367 PRINT"▮PRESS F7▮"
368 GETA$:IFA$<>"▮"THEN368
369 PC=1:PRINT"⊐TRACK   SECTOR   HEX.ADD  DEC.ADD":PRINT
370 AL=PEEK(52994):AH=PEEK(52995):AT=256*AH+AL
375 GOSUB1000
380 PRINTTR;TAB(9)SE;TAB(18)D$;TAB(29)AT
382 IFP<>1THEN390
385 OPEN4,4
386 PRINT#4,TR;CHR$(16)"07"SE;CHR$(16)"16"D$;CHR$(16)"26"AT:CLOSE4
390 NEXT
400 PRINT"▮▮PRESS F7"
410 GETA$:IFA$<>"▮"THEN410
999 RUN10
1000 X=AT:D$=""
1005 D(1)=INT(X/4096):X=X-(D(1)*4096):D(2)=INT(X/256):X=X-(D(2)*256)
1010 D(3)=INT(X/16):D(4)=X-(D(3)*16)
1020 FORI=1TO4:D$=D$+MID$(HX$,(D(I)+1),1):NEXT
1060 RETURN
```

CHANGE THESE LINES FOR NON-COMMODORE PRINTERS

```
25 IFP=1THENOPEN4,4:PRINT#4:PRINT#4,TAB(15)"SUPERDIRECTORY":PRINT#4:CLOSE4
165 IFP=1THENPRINT#4,FT$;TAB(07)TK;
172 PRINTTAB(5)TK;TAB(12)SK;
175 IFP=1THENPRINT#4,TAB(18)SK;TAB(24);
178 IFOK=1THENB(C)=TK:B(C+1)=SK:C=C+2
180 PRINTTAB(18);:FORTX=3TO18:PRINTCHR$(A(RP+TX));:NEXT:PRINTTAB(34)NB
185 IFP=1THENFORTX=3TO18:PRINT#4,CHR$(A(RP+TX));:NEXT:PRINT#4,TAB(2)NB
386 PRINT#4,TR;TAB(4)SE;TAB(5)D$;TAB(7)AT:CLOSE4
```

ADMACH

```
C000  18        CLC
C001  A9 00     LDA #$00
C003  85 FB     STA $FB
C005  A9 CF     LDA #$CF
C007  85 FC     STA $FC
C009  A2 02     LDX #$02
C00B  20 C6 FF  JSR $FFC6
C00E  A0 00     LDY #$00
C010  20 A5 FF  JSR $FFA5
C013  91 FB     STA ($FB),Y
C015  C8        INY
C016  D0 F8     BNE $C010
C018  20 CC FF  JSR $FFCC
C01B  60        RTS
C01C  00        BRK
C01D  00        BRK
C01E  00        BRK
C01F  00        BRK
C020  18        CLC
C021  A9 00     LDA #$00
C023  85 FB     STA $FB
C025  A9 CF     LDA #$CF
C027  85 FC     STA $FC
C029  A2 02     LDX #$02
C02B  20 C9 FF  JSR $FFC9
C02E  A0 00     LDY #$00
C030  B1 FB     LDA ($FB),Y
C032  20 A8 FF  JSR $FFA8
C035  C8        INY
C036  D0 F8     BNE $C030
C038  20 CC FF  JSR $FFCC
C03B  60        RTS
C03C  00        BRK
C03D  00        BRK
C03E  C0 8D     CPY #$8D
```

DISK-EDITOR

```
10 IFA=0THENA=1:LOAD"ADMACH",8,1
20 POKE53280,11:POKE53281,11:DIMA(300):DIMA$(300):HX$="0123456789ABCDEF"
25 P=0:INPUT"⊐▤PRINTER Y/N";YN$:IFYN$="Y"THENP=1
30 PRINT"⊐▣     DISK-EDITOR▮ (C) PSIDAC VBN▤"
40 IFP=1THENOPEN4,4:PRINT#4:PRINT#4,CHR$(16)"13DISK-EDITOR":PRINT#4:CLOSE4
50 PRINT"▣ INSERT SOURCE DISK IN DRIVE":INPUT"▣ DISK NAME";DN$
80 IFP=1THENOPEN4,4:PRINT#4,"DISK NAME = "DN$:PRINT#4:CLOSE4
110 INPUT"▣ INPUT TRACK # ";TR:INPUT"▣ INPUT SECTOR # ";SE
120 OPEN15,8,15:OPEN2,8,2,"#":PRINT#15,"U1:"2;0;TR;SE:SYS49152:CLOSE2
125 INPUT#15,A$,B$,C$,D$:PRINT"▣ "A$,B$,C$,D$"▣":CLOSE15
130 FORRP=0TO255:A(RP)=PEEK(RP+52992):NEXT
140 PRINT"⊐";:RP=0:FORXA=0TO20:FORYA=1TO12:GOSUB1000:PRINTD$" ";:RP=RP+1
150 NEXT:PRINT"    ";:NEXT
160 FORRP=RPTO255:GOSUB1000:PRINTD$" ";:NEXT:PRINT
165 PRINT"TRACK "TR" SECTOR "SE:IFP=0THEN200
170 OPEN4,4:RP=0
175 FORXA=0TO20:FORYA=1TO12:GOSUB1000:PRINT#4,D$" ";:RP=RP+1
180 NEXT:PRINT#4," ":NEXT
190 FORRP=RPTO255:GOSUB1000:PRINT#4,D$" ";:NEXT:PRINT#4
195 PRINT#4,"TRACK "TR" SECTOR "SE:CLOSE4
200 FORCL=55296TO55296+5:POKECL,13:NEXT
205 SYS65487:PRINT"▣▣▣▣▣▣▣▣▣▣▣▣▣▣▣▣▣▣▣▣▣▣▣PLEASE WAIT▤";
210 SL=1024:BL=52992:FORBC=0TO255
220 X=PEEK(SL):IFX=32THENSL=SL+1:GOTO220
230 IFX>47THENX=(X-48)*16:GOTO260
240 IFX<7THENX=(X+9)*16:GOTO260
250 PRINT"⊐▣▣ ILLEGAL DATA FOUND":END
260 POKESL,32:SL=SL+1:Y=PEEK(SL)
270 IFY>47THENY=Y-48:GOTO300
280 IFY<7THENY=Y+9:GOTO300
290 GOTO250
300 NT=X+Y:POKEBL,NT:BL=BL+1:POKESL,32:SL=SL+1:NEXT
330 PRINT"⊐▣ READY DISK TO SAVE"
340 PRINT"▣ PRESS F7 TO SAVE":PRINT"▣ PRESS F1 TO NOT SAVE"
350 GETX$
360 IFX$="▮"THENPRINT"⊐":GOTO390
370 IFX$="■"THENPRINT"⊐":GOTO110
380 GOTO350
390 OPEN15,8,15:OPEN2,8,2,"#":PRINT#15,"B-P:"2;0
400 SYS49184:PRINT#15,"U2:"2;0;TR;SE:CLOSE2
410 INPUT#15,A$,B$,C$,D$:PRINT"▣ "A$,B$,C$,D$:CLOSE15
415 PRINT"▣ TO USE PGM. AGAIN PRESS F7"
420 GETX$:IFX$="▮"THENRUN20
440 GOTO420
999 END
1000 X=A(RP):D$=""
1010 D1=INT(X/16):D$=D$+MID$(HX$,D1+1,1)
1020 D2=INT(X-16*D1):D$=D$+MID$(HX$,D2+1,1):RETURN
```

FOR NON-COMMODORE PRINTERS CHANGE THESE LINES

```
40 IFP=1THENOPEN4,4:PRINT#4:PRINT#4,TAB(13)"DISK-EDITOR":PRINT#4:CLOSE4
```

ERROR ANALYZER

```
10 POKE53280,11:POKE53281,11:P=4:D=8:RT=1:NT=3
20 DIMER$(11):FORRP=1TO11:READER$(RP):NEXT
30 DATA"ALL OK","NO HEADER FOUND","NO SYNC FOUND","DATA BLOCK NOT FOUND"
40 DATA"CHECKSUM ERROR IN DATA","BYTE DECODING ERROR","WRITE VERIFY ERROR"
50 DATA"WRITE PROTECT ON","CHECK SUM ERROR IN HEADER","LONG DATA BLOCK"
60 DATA"DISK ID MISMATCH"
80 DATA 1,17,0,20,18,24,0,18,25,30,0,17,31,35,0,16,36,44,0,15
85 OPEN15,D,15,"I0":CLOSE15
90 PRINT"■■        ERROR ANALYZER■    PSIDAC(C)84 VBN"
100 PRINT"■■INSERT SOURCE DISK  IN DRIVE":PRINT"PRESS F7"
110 GETF7$:IFF7$<>"■"THEN110
115 PR=0:INPUT"■PRINTER Y/N";YN$:IFYN$="Y"THENPR=1
120 INPUT "■DISK NAME";DN$
122 PRINT"■SELECT CHOICE":PRINT"■(1)=LOG ERRORS"
123 PRINT"■(2)=LOG UNFORMATTED TRACKS■"
124 INPUTCH$:CH=VAL(CH$):PRINT"■■":ONCHGOTO128,300
125 PRINT"■":GOTO122
128 IFPR=1THENOPEN4,P:PRINT#4,"          ERROR LOG FOR "DN$:PRINT#4:CLOSE4
140 FORPP=1TO5:READFT,LT,FS,LS
150 FORTC=FTTOLT:FORSC=FSTOLS:OPEN15,D,15
160 T=0
170 PRINT#15,"M-W"CHR$(6)CHR$(0)CHR$(1)CHR$(TC)
180 PRINT#15,"M-W"CHR$(7)CHR$(0)CHR$(1)CHR$(SC)
190 PRINT#15,"M-W"CHR$(0)CHR$(0)CHR$(1)CHR$(128)
200 PRINT#15,"M-R"CHR$(0)CHR$(0):GET#15,A$:X=ASC(A$+CHR$(0)):IFX>127THEN200
202 IFX=1THEN210
205 T=T+1:IFT<>NTTHEN170
210 CLOSE15:PRINT"ER#"X;TAB(6)"TR#"TC;TAB(12)"SE#"SC:PRINTER$(X)"■"
220 IFX=1ORRT=1THEN1000
230 IFPR=0THEN265
235 OPEN4,P
250 PRINT#4,"ER#"X" TR# "TC;CHR$(16)"06SE# "SC;CHR$(16)"24"ER$(X):CLOSE4
260 GOTO1000
265 GOSUB2000
280 GOTO1000
300 IFPR=1THENOPEN4,P:PRINT#4,"UNFORMATTED TRACK LOG  FOR "DN$:PRINT#4:CLOSE4
310 FORTR=1TO44:OPEN15,D,15
320 PRINT#15,"M-W"CHR$(6)CHR$(0)CHR$(1)CHR$(TR)
330 PRINT#15,"M-W"CHR$(7)CHR$(0)CHR$(1)CHR$(0)
340 PRINT#15,"M-W"CHR$(0)CHR$(0)CHR$(1)CHR$(128)
350 PRINT#15,"M-R"CHR$(0)CHR$(0):GET#15,A$:X=ASC(A$+CHR$(0)):IFX>127THEN350
360 CLOSE15
365 FM$="SYNC FOUND":IFX=3THENFM$="UNFORMATTED"
375 PRINT"TRACK "TR;TAB(12)FM$
385 IFPR=1ANDX=3THEN OPEN4,P:PRINT#4,"TRACK "TR;CHR$(16)"12"FM$:CLOSE4
390 NEXT:GOSUB2000:GOTO85
999 END
1000 NEXT:NEXT:NEXT
1010 GOSUB2000:GOTO85
2000 PRINT"■PRESS F7 TO CONTINUE■"
2010 GETF7$:IFF7$<>"■"THEN2010
2020 RETURN
```

INSERT LINE 112 OPEN15,D,15,"I0":CLOSE15

CHANGE FOLLOWING LINES FOR NON-COMMODORE PRINTERS

```
250 PRINT#4,"ER#"X" TR# "TC;TAB(06)"SE# "SC;TAB(24)ER$(X):CLOSE4
385 IFPR=1ANDX=3THEN OPEN4,P:PRINT#4,"TRACK "TR;TAB(12)FM$:CLOSE4
```

NOTE: The following options for Error Analyzer are included for your benefit:
LINE 10 RT=0 for STOP on ERROR, or RT=1 to run through.
LINE 140 For non-extended (0-35) "formatted track check", Change PP=1 TO 5 to PP=1 TO 4.
LINE 310 Set TR=1TO44 for complete sync check. Set TR=1TO35 for non-extended sync check, or set TR=36TO44 to only check "extra" tracks for sync.
NOTE: IF DISK LOCKS UP BEYOND TRACK 35, OPEN DRIVE DOOR. Some diskette manufacturers certification methods may cause variations in what is "found" in the extra tracks.

LINKSTER

```
100 PRINT"┐        ╗LINKSTER ■(C) PSIDAC 1984 DT◣"
110 POKE53280,11:POKE53281,11:P=0:DR=8
120 INPUT"▒PRINTER ON Y/N";PR$
130 IFPR$="Y"THENP=1:INPUT"▒PRG NAME";NM$:GOTO140
135 IFPR$<>"N"THEN100
140 PRINT"┐        ╗LINKSTER ■(C) PSIDAC 1984 DT◣"
150 PRINT"▒ENTER STARTING TRACK & SECTOR"
160 INPUT"▒TRACK";TR:INPUT"▒SECTOR";SE:PRINT
180 IFP=1THENOPEN4,4:PRINT#4,NM$CHR$(13)"STARTING (TR SEC)-"TR;SE:CLOSE4
190 LN=1:CT=1
200 LN=LN+1:OPEN15,DR,15:OPEN3,DR,3,"#"
210 PRINT#15,"B-R";3;DR;TR;SE:GOTO600
220 PRINT#15,"B-P";3;0
230 GET#3,Q$:IFQ$=""THENQ$=CHR$(0)
240 TR=ASC(Q$):IFTR=0ORTR>35THEN320
250 GET#3,W$:IFW$=""THENW$=CHR$(0)
260 CLOSE3:CLOSE15:SE=ASC(W$)
270 IFP=1ANDLN=6THENOPEN4,4:PRINT#4," ":CLOSE4
280 IFLN=6THENPRINT:LN=1
290 PRINTTR"■"SE"-";
300 IFP=1THENOPEN4,4:PRINT#4,TR;SE" - ";:CLOSE4
310 CT=CT+1:GOTO200
320 PRINT" ENDING TR/SEC▒"
330 IFP=1THENOPEN4,4:PRINT#4," - ENDING TR/SEC"
340 IFP=1THENPRINT#4,CHR$(13)CT" BLOCKS USED":PRINT#4:CLOSE4
350 PRINTCT" BLOCKS USED   ▒"
360 PRINT"PRESS F7 TO RUN AGAIN":CLOSE15:CLOSE3
370 GETCK$:IFCK$<>"■"THEN370
380 RUN
600 INPUT#15,ER$,B$,C$,D$:IFER$="00"THENGOTO 220
610 PRINT:PRINT"╗ERROR CONDITION "ER$,B$" "C$" "D$"◣"
620 PRINT"▒CORRECT PROBLEM - PRESS F7"
630 GETCK$:IFCK$<>"■"THEN630
640 PRINT"┐                              ┬┬"
650 CLOSE3:CLOSE15:GOTO150

READY.
```

RELOCATE/LOADER

```
1 IFA=0THENA=1:LOAD"MACHRELO",8,1
2 POKE52,10:POKE56,10:POKE53280,11:POKE53281,11
3 PRINT"▯▮▮   RELO/LOADER ▮PSIDAC (C)84 VBN▮"
4 PRINT"▮PUT ORIG. DISK IN DRIVE"
5 INPUT"▮INPUT FILENAME";NF$:NL=LEN(NF$):POKE251,NL
6 FORLP=1TONL:POKE819+LP,ASC(MID$(NF$,LP,1)):NEXT:SYS840
7 EA=PEEK(175)*256+PEEK(174):PRINT"▮# BYTES ="EA-2560
8 PRINT"▮PUT CPY DISK IN DRIVE":PRINT"▮PRESS F7"
9 GETA$:IFA$<>"▮"THEN9
10 PRINT"▮SAVING "NF$" FROM 2560 TO"EA:SYS875:CLR
11 OPEN15,8,15:INPUT#15,A$,B$,C$,D$:PRINT"▮"A$,B$,C$,D$:CLOSE15
```

MACHRELO

```
0348  20 95 03 JSR $0395
034B  A9 08    LDA #$08
034D  A2 08    LDX #$08
034F  A0 00    LDY #$00
0351  20 BA FF JSR $FFBA
0354  A5 FB    LDA $FB
0356  A2 34    LDX #$34
0358  A0 03    LDY #$03
035A  20 BD FF JSR $FFBD
035D  A9 00    LDA #$00
035F  A2 00    LDX #$00
0361  A0 0A    LDY #$0A
0363  20 D5 FF JSR $FFD5
0366  20 A0 03 JSR $03A0
0369  60       RTS
036A  00       BRK
036B  20 95 03 JSR $0395
036E  A9 08    LDA #$08
0370  A2 08    LDX #$08
0372  A0 FF    LDY #$FF
0374  20 BA FF JSR $FFBA
0377  A5 FB    LDA $FB
0379  A2 34    LDX #$34
037B  A0 03    LDY #$03
037D  20 BD FF JSR $FFBD
0380  A9 00    LDA #$00
0382  85 FC    STA $FC
0384  A9 0A    LDA #$0A
0386  85 FD    STA $FD
0388  A9 FC    LDA #$FC
038A  A6 AE    LDX $AE
038C  A4 AF    LDY $AF
038E  20 D8 FF JSR $FFD8
0391  20 A0 03 JSR $03A0
0394  60       RTS
0395  A5 01    LDA $01
0397  8D A7 02 STA $02A7
039A  29 FE    AND #$FE
039C  85 01    STA $01
039E  60       RTS
039F  00       BRK
03A0  AD A7 02 LDA $02A7
03A3  85 01    STA $01
03A5  60       RTS
```

```
 10 IFA=0THENA=1:LOAD"ANALYMACH",8,1
 11 DATA"ALL OK","NO HEADER ","NO SYNC ","NO  DATA "
 12 DATA"DATA CHECK SUM","X","X"
 13 DATA"X","CHKSUM IN HDR","X"
 14 DATA"BAD ID"
 15 DIMER$(11):FORRP=1TO11:READER$(RP):NEXT
 18 POKE54296,15:POKE54277,17:POKE54278,248:POKE54272,3:POKE54273,35
 20 POKE53280,11:POKE53281,11:SL=49152:POKE51233,1:D=8:P=4
 30 DATA1,17,0,20,18,24,0,18,25,30,0,17,31,35,0,16
 40 GOSUB1000:OPEN15,D,15,"I0":CLOSE15
 50 PRINT"{SC}{YL}       T/S ANALYZER{LG}   PSIDAC(C)84 VBN"
 60 PRINT"{CD}{CD}{CY}                  * * * MENU * * *"
 70 PRINT"{CD}{WH}       (1)ANALYZE T/S DATA"
 75 PRINT"{CD}{WH}       (2)PRINT T/S STATUS LOG"
 80 PRINT"{CD}{WH}       (3)LOAD T/S STATUS LOG"
 85 PRINT"{CD}{WH}       (4)SAVE T/S STATUS LOG"
 90 INPUT"{CD}{YL}       SELECT CHOICE{WH}";CH$:CH=VAL(CH$)
 100 ONCHGOTO200,400,600,800:GOTO50
 200 PRINT"{SC}INSERT SOURCE DISK IN DRIVE":PRINT"PRESS F7"
 210 GETF7$:IFF7$<>"{F7}"THEN210
 215 INPUT"{CD}DISK NAME";DN$
 220 PRINT"{SC}{YL}T/S DATA ANALYZER ENABLED{WH}"
 230 FORPP=1TO4:READFT,LT,FS,LS
 240 FORTC=FTTOLT:FORSC=FSTOLS
 242 OPEN15,D,15:OPEN2,D,2,"#"
 245 PRINT#15,"M-W"CHR$(6)CHR$(0)CHR$(1)CHR$(TC)
 248 PRINT#15,"M-W"CHR$(7)CHR$(0)CHR$(1)CHR$(SC)
 250 PRINT#15,"M-W"CHR$(0)CHR$(0)CHR$(1)CHR$(128)
 255 PRINT#15,"M-R"CHR$(0)CHR$(0):GET#15,Q$:Q=ASC(Q$+CHR$(0)):IFQ>127THEN255
 258 IFQ<>1THEN280
 270 PRINT#15,"U1:"2;0;TC;SC
 275 SYS51264:SS=PEEK(51232):GOSUB1000
 280 EN=Q:NM$=ER$(Q):TR=TC:SE=SC
 300 IFQ<>1THENSS=Q:EC=EC+1:GOTO320
 315 NM$="DATA":IFSS=1THENNM$="UNUSED"
 320 POKESL,TC:SL=SL+1:POKESL,SC:SL=SL+1:POKESL,SS:SL=SL+1
 325 CLOSE2:GOSUB1005
 330 PRINT"TRK"TAB(3)TC;TAB(8)"SEC"TAB(11)SC;TAB(16)"CODE"TAB(18)SS;TAB(25)NM$
 335 NEXT:NEXT:NEXT
 340 PRINT"{CD}"EC"ERRORS COUNTED":FORTD=1TO3000:NEXT:GOTO40
 400 PRINT"{CD}PRINT MODE ENABLED"
 402 OPEN4,P:PRINT#4:PRINT#4," * * * DISK ANALYSIS * * *":PRINT#4
 405 PRINT#4," NAME = "DN$:PRINT#4
 410 PRINT#4," TRK   SEC   CODE   STATUS":PRINT#4
 415 FORBL=49152TO51198STEP3
 420 X=PEEK(BL):Y=PEEK(BL+1):Z=PEEK(BL+2)
 430 MS$="READ ERROR"
 440 IFZ=0THENMS$="DATA"
 450 IFZ=1THENMS$="UNUSED"
 460 PRINT#4,X;CHR$(16)"06"Y;CHR$(16)"13";CHR$(16)"12"Z;CHR$(16)"20"MS$
 470 NEXT:CLOSE4:GOTO40
 600 PRINT"{SC}LOAD MODE{CD}":PRINT"INSERT STATUS LOG DISK IN DRIVE":PRINT"PRESS F7"
 602 GETF7$:IFF7$<>"{F7}"THEN602
 605 INPUT"{SC}{YL}INPUT STATUS LOG NAME";NF$:DN$=NF$
 610 NL=LEN(NF$):POKE251,NL:FORLP=1TONL:POKE678+LP,ASC(MID$(NF$,LP,1))
 615 NEXT:SYS51308:OPEN15,D,15:INPUT#15,A$,B$,C$,D$:PRINTA$,B$,C$,D$
 620 GOSUB1005:FORTD=1TO3000:NEXT:GOTO40
 800 PRINT"{SC}SAVE MODE{CD}":PRINT"INSERT STATUS LOG DISK IN DRIVE":PRINT"PRESS F7"
 802 GETF7$:IFF7$<>"{F7}"THEN802
 805 INPUT"{SC}{YL}INPUT STATUS LOG NAME";NF$:DN$=NF$
 810 NL=LEN(NF$):POKE251,NL:FORLP=1TONL:POKE678+LP,ASC(MID$(NF$,LP,1))
 815 NEXT:SYS51335:OPEN15,D,15:INPUT#15,A$,B$,C$,D$:PRINTA$,B$,C$,D$
 820 GOSUB1005:FORTD=1TO3000:NEXT:GOTO40
 999 END
 1000 POKE54276,35:POKE54276,34:RETURN
 1005 CLOSE15:RETURN
```

CHANGE THIS LINE FOR NON-COMMODORE PRINTERS

```
   460 PRINT#4,X;TAB(02)Y;TAB(02)Z;TAB(03)MS$
```

NOTE: T/S Analyzer may indicate data on tracks containing only format data. This will be apparent on some partially full disks which indicate data in the first track. If you suspect this to be the case, POKE51233,0 (disk formatted with zeros) or POKE51233,1 (disk formatted with ones).

FASTBACK

```
10 IFA=0THENA=1:LOAD"ANALYMACH",8,1
15 POKE52,16:POKE56,16:POKE53281,11:POKE53280,11
20 POKE54296,15:POKE54277,17:POKE54278,248:POKE54272,3:POKE54273,35:D=8
30 JS=49152:JE=51198:BS=4096:BE=48988:RP=BS:WP=BS
35 SP$="                                        "
40 FORLP=1TO8:READIS(LP):NEXT:DATA 0,255,255,255,255,255,255,255
45 GOSUB1000:OPEN15,8,15,"I0":CLOSE15
50 PRINT"⊐ℼ      FASTBACK▮   PSIDAC(C)84 VBN"
60 PRINT"ⅅ≡INSERT STATUS LOG DISK IN DRIVE":GOSUB1050
70 INPUT"⊐INPUT STATUS LOG NAME";NF$:NL=LEN(NF$):POKE251,NL
75 FORLP=1TONL:POKE678+LP,ASC(MID$(NF$,LP,1)):NEXT:SYS51308
80 OPEN15,8,15:GOSUB1070:GOSUB1080
90 INPUT"⊐INPUT NUMBER OF COPIES";NC
100 PRINT"⊐INSERT SOURCE DISK IN DRIVE":GOSUB1050
102 PRINT"⊐ℼ          READ MODE ENABLEDⅅ≡"
105 IFJS=JETHEN400
110 SS=PEEK(JS+2):OK=0:FORCS=1TO8:IFSS=IS(CS)THENOK=1:NEXT
120 JS=JS+3:IFOK<>1THEN105
200 IFRP=BETHEN JS=JS-3:GOTO400
210 TR=PEEK(JS-3):SE=PEEK(JS-2):POKERP,TR:POKERP+1,SE:RP=RP+2
220 HI=INT(RP/256):LO=RP-(HI*256):POKE252,LO:POKE253,HI:GOSUB1090
240 GOSUB2000:PRINT#15,"U1:"2;0;TR;SE:SYS51376:GOSUB1000
250 RP=RP+256:GOSUB1070:GOTO105
400 FORCC=1TONC
410 PRINT"⊐INSERT COPY DISK IN DRIVE":GOSUB1050
415 PRINT"⊐◣   WRITE MODE ENABLEDⅅ≡"
420 IFWP=RPTHEN500
430 GOSUB2010:SYS(51452):TR=PEEK(679):SE=PEEK(680):WP=WP+2
440 GOSUB2010:LO=WP-(HI*256):POKE252,LO:POKE253,HI
460 GOSUB1090:GOSUB2000
470 PRINT#15,"B-P:"2;0:SYS51404:PRINT#15,"U2:"2;0;TR;SE:GOSUB1000
475 GOSUB1070:IFA$="00"THEN490
480 GOSUB1080:PRINT"⊐CLEAR ERROR THEN":GOSUB1050:PRINT"⊐":GOTO460
490 WP=WP+256:GOTO420
500 WP=BS:NEXT:IFJS<>JETHENRP=BS:GOTO100
510 PRINT"ℼ* * * DUPLICATION COMPLETE * * *":GOSUB1080:RUN15
999 END
1000 POKE54276,35:POKE54276,34:RETURN
1050 PRINT"PRESS F7"
1055 GETF7$:IFF7$<>"▮"THEN1055
1060 RETURN
1070 INPUT#15,A$,B$,C$,D$
1075 PRINT"ⅅ"SP$:PRINT"⊐"A$" "B$" "C$" "D$"TTT":CLOSE2:CLOSE15:OPEN15,8,15
1078 CLOSE15:RETURN
1080 FORTD=1TO3000:NEXT:RETURN
1090 PRINTSP$:PRINT"TTRACK"TAB(5)TR;TAB(10)"SECTOR"TAB(15)SE:RETURN
2000 OPEN15,D,15:OPEN2,D,2,"#":RETURN
2010 HI=INT(WP/256):LO=WP-(HI*256):POKE252,LO:POKE253,HI:RETURN
```

```
.,  C840  18        CLC
.,  C841  A9 00     LDA #$00
.,  C843  8D 20 C8  STA $C820
.,  C846  A2 02     LDX #$02
.,  C848  20 C6 FF  JSR $FFC6
.,  C84B  20 A5 FF  JSR $FFA5
.,  C84E  20 A5 FF  JSR $FFA5
.,  C851  A0 00     LDY #$00
.,  C853  20 A5 FF  JSR $FFA5
.,  C856  CD 21 C8  CMP $C821
.,  C859  D0 0A     BNE $C865
.,  C85B  C8        INY
.,  C85C  C0 FE     CPY #$FE
.,  C85E  D0 F3     BNE $C853
.,  C860  A9 01     LDA #$01
.,  C862  8D 20 C8  STA $C820
.,  C865  20 CC FF  JSR $FFCC
.,  C868  60        RTS
.,  C869  00        BRK
.,  C86A  00        BRK
.,  C86B  00        BRK
.,  C86C  A9 08     LDA #$08
.,  C86E  A2 08     LDX #$08
.,  C870  A0 FF     LDY #$FF
.,  C872  20 BA FF  JSR $FFBA
.,  C875  A5 FB     LDA $FB
.,  C877  A2 A7     LDX #$A7
.,  C879  A0 02     LDY #$02
.,  C87B  20 BD FF  JSR $FFBD
.,  C87E  A9 00     LDA #$00
.,  C880  20 D5 FF  JSR $FFD5
.,  C883  60        RTS
.,  C884  00        BRK
.,  C885  00        BRK
.,  C886  00        BRK
.,  C887  A9 08     LDA #$08
.,  C889  A2 08     LDX #$08
.,  C88B  A0 FF     LDY #$FF
.,  C88D  20 BA FF  JSR $FFBA
.,  C890  A5 FB     LDA $FB
.,  C892  A2 A7     LDX #$A7
.,  C894  A0 02     LDY #$02
.,  C896  20 BD FF  JSR $FFBD
.,  C899  A9 00     LDA #$00
.,  C89B  85 FB     STA $FB
.,  C89D  A9 C0     LDA #$C0
.,  C89F  85 FC     STA $FC
.,  C8A1  A9 FB     LDA #$FB
.,  C8A3  A2 02     LDX #$02
.,  C8A5  A0 C8     LDY #$C8
.,  C8A7  20 D8 FF  JSR $FFD8
.,  C8AA  60        RTS
.,  C8AB  00        BRK
.,  C8AC  00        BRK
.,  C8AD  00        BRK
.,  C8AE  00        BRK
.,  C8AF  00        BRK
.,  C8B0  20 E8 C8  JSR $C8E8
.,  C8B3  A2 02     LDX #$02
.
```

```
.,  C8B5  20 C6 FF  JSR $FFC6
.,  C8B8  A0 00     LDY #$00
.,  C8BA  20 A5 FF  JSR $FFA5
.,  C8BD  91 FC     STA ($FC),Y
.,  C8BF  C8        INY
.,  C8C0  D0 F8     BNE $C8BA
.,  C8C2  20 CC FF  JSR $FFCC
.,  C8C5  20 F4 C8  JSR $C8F4
.,  C8C8  60        RTS
.,  C8C9  00        BRK
.,  C8CA  00        BRK
.,  C8CB  00        BRK
.,  C8CC  20 E8 C8  JSR $C8E8
.,  C8CF  A2 02     LDX #$02
.,  C8D1  20 C9 FF  JSR $FFC9
.,  C8D4  A0 00     LDY #$00
.,  C8D6  B1 FC     LDA ($FC),Y
.,  C8D8  20 A8 FF  JSR $FFA8
.,  C8DB  C8        INY
.,  C8DC  D0 F8     BNE $C8D6
.,  C8DE  20 CC FF  JSR $FFCC
.,  C8E1  20 F4 C8  JSR $C8F4
.,  C8E4  60        RTS
.,  C8E5  00        BRK
.,  C8E6  00        BRK
.,  C8E7  00        BRK
.,  C8E8  A5 01     LDA $01
.,  C8EA  85 FE     STA $FE
.,  C8EC  29 FE     AND #$FE
.,  C8EE  85 01     STA $01
.,  C8F0  60        RTS
.,  C8F1  00        BRK
.,  C8F2  00        BRK
.,  C8F3  00        BRK
.,  C8F4  A5 FE     LDA $FE
.,  C8F6  85 01     STA $01
.,  C8F8  60        RTS
.,  C8F9  00        BRK
.,  C8FA  00        BRK
.,  C8FB  00        BRK
.,  C8FC  20 E8 C8  JSR $C8E8
.,  C8FF  A0 00     LDY #$00
.,  C901  B1 FC     LDA ($FC),Y
.,  C903  99 A7 02  STA $02A7,Y
.,  C906  C8        INY
.,  C907  B1 FC     LDA ($FC),Y
.,  C909  99 A7 02  STA $02A7,Y
.,  C90C  20 F4 C8  JSR $C8F4
.,  C90F  60        RTS
.,  C910  00        BRK
.
```

DISKPICKER pg1

```
10 IFA=0THENA=1:LOAD"MONITOR$8000",8,1
20 IFA=1THENA=2:LOAD"ZMACH",8,1
30 POKE52,31:POKE56,31
35 SYS49152:RUN40
40 POKE53280,11:POKE53281,11:CLR:DIMER$(11):GOSUB3000
45 PRINT"{SC}{YL}      DISKPICKER{LG} PSIDAC(C)84 VBN{WH}"
50 PRINT"{CD}             *** MENU ***"
60 PRINT"{CD}(1) TRANSFER DISK MEMORY TO BUFFER"
65 PRINT"{CD}(2) ENABLE MONITOR MODE"
70 PRINT"{CD}(3) TRANSFER BUFFER TO DISK MEMORY"
75 PRINT"{CD}(4) DIRECT EXECUTE USER PROGRAM"
78 PRINT"{CD}(5) JOB QUE EXECUTE USER PROGRAM"
80 PRINT"{CD}(6) LOAD SECTOR TO DISK BUFFER"
82 PRINT"{CD}(7) INITIALIZE DISK IO"
83 PRINT"{CD}(8) FORMAT DISKETTE"
84 PRINT"{CD}(9) POSITION READ/WRITE HEAD"
85 INPUT"{CD}SELECT CHOICE";CH$:CH=VAL(CH$)
88 IFCH>9THEN40
90 ONCH GOTO 100,300,400,500,600,700,800,900,805
95 GOTO40
100 PRINT"{SC}{CD}(1) TRANSFER DISK MEMORY TO BUFFER":GOSUB2000
105 INPUT"{CD}INPUT DISK START ADDRESS";X$:GOSUB1000:SA=X
110 INPUT"{CD}INPUT DISK END ADDRESS";X$:GOSUB1000:EA=X
120 INPUT"{CD}INPUT BUFFER ADDRESS";X$:GOSUB1000:BA=X:IFX<8192THENGOSUB2030
125 IFX>32767THENGOSUB2040
130 PRINT"{CD}DISK MEM. TO BUFF. TRANSFER IN PROCESS":PRINT"{CD}BYTE COUNT =";
140 NB=EA-SA:SH=INT(SA/256):SL=SA-(SH*256)
150 OPEN15,8,15:FORLP=0TONB
160 PRINT#15,"M-R"CHR$(SL)CHR$(SH):GET#15,A$:PN=ASC(A$+CHR$(0))
170 POKEBA,PN:BA=BA+1:SL=SL+1:IFSL=256THENSL=0:SH=SH+1
175 PRINTTAB(12)LP"{CU}":NEXT
180 CLOSE15:PRINT"{CD}{CD}END TRANSFER":FORTD=1TO3000:NEXT:RUN40
300 INPUT"{CD}PRINTER Y/N";YN$:IFYN$="Y"THENP=1
302 IFYN$<>"Y"ANDYN$<>"N"THENPRINT"{CU}{CU}{CU}":GOTO300
305 PRINT:IFP=1THENOPEN4,4:CMD4
310 SYS32768
320 END
400 PRINT"{SC}{CD}(3) TRANSFER BUFFER TO DISK MEMORY":GOSUB2000
405 INPUT"{CD}INPUT DISK START ADDRESS";X$:GOSUB1000:SA=X
410 INPUT"{CD}INPUT DISK END ADDRESS";X$:GOSUB1000:EA=X
420 INPUT"{CD}INPUT BUFFER ADDRESS";X$:GOSUB1000:BA=X:IFX<8192THENGOSUB2030
425 IFX>32767THENGOSUB2040
430 PRINT"{CD}BUFF. TO DISK MEM. TRANSFER IN PROCESS":PRINT"{CD}BYTE COUNT =";
440 NB=EA-SA:SH=INT(SA/256):SL=SA-(SH*256)
450 OPEN15,8,15:FORLP=0TONB
460 PN=PEEK(BA):PRINT#15,"M-W"CHR$(SL)CHR$(SH)CHR$(1)CHR$(PN)
470 BA=BA+1:SL=SL+1:IFSL=256THENSL=0:SH=SH+1
475 PRINTTAB(12)LP"{CU}":NEXT
480 CLOSE15:PRINT"{CD}{CD}END TRANSFER":FORTD=1TO3000:NEXT:RUN40
500 PRINT"{SC}{CD}(4) DIRECT EXECUTE USER PROGRAM":GOSUB2000
505 INPUT"{CD}INPUT DISK PROG. ENTRY ADDRESS";X$:GOSUB1000:SA=X
510 SH=INT(SA/256):SL=SA-(SH*256)
520 OPEN15,8,15:PRINT#15,"M-E"CHR$(SL)CHR$(SH):CLOSE15
530 PRINT"{CD}PROGRAM EXECUTION ENABLED":CLOSE15:FORTD=1TO3000:NEXT:RUN40
600 PRINT"{SC}{CD}(5) JOB QUE EXECUTE USER PROGRAM":GOSUB2000
605 PRINT"{CD}INPUT JOB CHOICE{CD}":PRINT"128=READ":PRINT"144=WRITE":PRINT"160=VERIFY
610 PRINT"176=SEEK":PRINT"184=SEC SEEK":PRINT"192=BUMP":PRINT"208=JUMP"
615 PRINT"224=EXECUTE":INPUT"{CD}CHOICE";CH$:CH=VAL(CH$):X$=CH$:GOSUB1100
617 IFCH=0THEN600:
618 IFCH<>224THEN640
620 PRINT"{SC}SELECT EXECUTE STARTING ADDRESS":PRINT"{CD}ADDRESSES AVAILABLE ARE:"
625 PRINT"{CD}(1)=0300":PRINT"(2)=0400":PRINT"(3)=0500":PRINT"(4)=0600"
628 PRINT"(5)=0700"
630 INPUT"{CD}CHOICE";CH$:X$=CH$:GOSUB1100
632 C1=VAL(CH$):ONC1GOTO640,640,640,640,640
635 GOTO620
640 GOSUB2000
642 OK$="N":INPUT"{CD}MULTI SECTOR Y/N";OK$
645 INPUT"{SC}{CD}INPUT HEADER #1 ";X$:TN=VAL(X$):IFX$="X"THENGOSUB1100
648 IFOK$="Y"THENFORSN=20TO0STEP-1:GOTO651
650 INPUT"{CD}INPUT HEADER #2 ";X$:SN=VAL(X$):IFX$="X"THENGOSUB1100
651 OPEN15,8,15
652 IFCH<>224THEN655
653 IFC1=1THENPRINT#15,"M-W"CHR$(63)CHR$(0)CHR$(1)CHR$(C1-1)
654 IFC1<>1THENPRINT#15,"M-W"CHR$(0)CHR$(3)CHR$(3)CHR$(76)CHR$(0)CHR$(C1+2)
655 PRINT#15,"M-W"CHR$(6)CHR$(0)CHR$(1)CHR$(TN)
660 PRINT#15,"M-W"CHR$(7)CHR$(0)CHR$(1)CHR$(SN)
665 PRINT#15,"M-W"CHR$(0)CHR$(0)CHR$(1)CHR$(CH)
670 PRINT"{CD}PROGRAM EXECUTION ENABLED"
675 PRINT#15,"M-R"CHR$(0)CHR$(0):GET#15,A$:OP=ASC(A$+CHR$(0)):IFOP>127THEN675
680 PRINT"{CD}TRK."TN" SEC."SN"    OP STATUS CODE = "OP
685 PRINT"{YL}{CD}"ER$(OP)"{WH}"
690 IFOK$="Y"THENCLOSE15:NEXT:GOSUB2000:RUN40
695 CLOSE15:GOSUB2000:RUN40
700 PRINT"{SC}{CD}(6) LOAD SECTOR TO DISK BUFFER":GOSUB2000
705 INPUT"{CD}INPUT TRACK # ";X$:TN=VAL(X$):GOSUB1100
710 INPUT"{CD}INPUT SECTOR # ";X$:SN=VAL(X$):GOSUB1100
715 OPEN15,8,15:OPEN2,8,2,"#"
```

```
720 PRINT#15,"U1:"2;0;TN;SN:INPUT#15,A$,B$,C$,D$:PRINT"{CD}"A$,B$,C$,D$
725 FORTD=1TO3000:NEXT
730 CLOSE2:CLOSE15:RUN40
800 OPEN15,8,15,"I0":CLOSE15:RUN40
805 PRINT"{SC}{CD}(9) POSITION READ/WRITE HEAD":TR$="18"
810 INPUT"{CD}     TRACK WANTED";TR$:TR=VAL(TR$):HT=TR-INT(TR):IFTR$="X"THENRUN40
811 IFTR>41THEN805
815 OPEN15,8,15,"I0":PRINT#15,"M-W"CHR$(0)CHR$(0)CHR$(1)CHR$(192):CLOSE15
817 FORTD=1TO5000:NEXT
820 OPEN15,8,15:PRINT#15,"M-W"CHR$(6)CHR$(0)CHR$(1)CHR$(TR)
830 PRINT#15,"M-W"CHR$(0)CHR$(0)CHR$(1)CHR$(176)
835 PRINT#15,"M-R"CHR$(0)CHR$(0)
840 GET#15,A$:X=ASC(A$+CHR$(0)):IFX>127THEN840
845 IFHT<>.5THENCLOSE15:RUN40
850 PRINT#15,"M-R"CHR$(0)CHR$(28):GET#15,X$:X=ASC(X$+CHR$(0))
855 BI=XAND3:BI=BI-1:BI=BIAND3:HP=(XAND252)ORBI
857 PRINT"{CD}   TRACK = "TR" HEAD PHASE = "BI
860 PRINT#15,"M-W"CHR$(0)CHR$(28)CHR$(1)CHR$(HP):CLOSE15:FORTD=1TO3000:NEXT
865 RUN40
900 PRINT"{SC}INSERT DISKETTE TO BE FORMATTED":GOSUB2000
905 INPUT"{CD}DISKETTE NAME";X$:NM$=X$:GOSUB1100
910 INPUT"{CD}DISKETTE ID";X$:ID$=X$:GOSUB1100:GOSUB2000
915 PRINT"{SC}FORMAT ENABLED PLEASE WAIT"
920 OPEN15,8,15:PRINT#15,"N0:"NM$;CHR$(44);ID$:INPUT#15,A$,B$,C$,D$
930 PRINT"{CD}"A$,B$,C$,D$:CLOSE15:FORTD=1TO3000:NEXT:RUN40
1000 IFX$="X"THENRUN40
1005 FORI=1TO4:D$(I)=MID$(X$,I,1):NEXT
1010 FORI=1TO4:D(I)=VAL(D$(I))
1020 IFD$(I)="A"THEND(I)=10
1025 IFD$(I)="B"THEND(I)=11
1030 IFD$(I)="C"THEND(I)=12
1035 IFD$(I)="D"THEND(I)=13
1040 IFD$(I)="E"THEND(I)=14
1045 IFD$(I)="F"THEND(I)=15
1050 NEXT
1060 D(1)=D(1)*4096:D(2)=D(2)*256:D(3)=D(3)*16:X=D(1)+D(2)+D(3)+D(4)
1065 RETURN
1100 IFX$<>"X"THENRETURN
1105 RUN40
2000 PRINT"{CD}CHECK DISK THEN PRESS F7"
2010 GETF7$:IFF7$<>"{F7}"THEN2010
2020 RETURN
2030 PRINT"{YL}{CD}BUFF ADDRESS TOO LOW(WH)":FORTD=1TO2000:NEXT:PRINT"{SC}":GOTO120:RETURN
2040 PRINT"{YL}{CD}BUFF ADDRESS TOO HIGH(WH)":FORTD=1TO2000:NEXT:PRINT"{SC}":GOTO120:RETURN
3000 FORRP=1TO11:READER$(RP):NEXT
3010 DATA"ALL OK","NO HEADER FOUND","NO SYNC FOUND","DATA BLOCK NOT FOUND"
3020 DATA"CHECKSUM ERROR IN DATA","BYTE DECODING ERROR","WRITE VERIFY ERROR"
3030 DATA"WRITE PROTECT ON","CHECKSUM ERROR IN HEADER","LONG DATA BLOCK"
3040 DATA"DISK ID MISMATCH"
3050 RESTORE:RETURN
```

ZMACH

```
.. C000  20 C3 FF JSR $FFC3
.. C003  AD 01 08 LDA $0801
.. C006  85 FD    STA $FD
.. C008  AD 02 08 LDA $0802
.. C00B  85 FE    STA $FE
.. C00D  A2 00    LDX #$00
.. C00F  B5 00    LDA $00,X
.. C011  9D 40 C0 STA $C040,X
.. C014  E8       INX
.. C015  D0 F8    BNE $C00F
.. C017  60       RTS
.. C018  00       BRK
.. C019  00       BRK
.. C01A  00       BRK
.. C01B  00       BRK
.. C01C  00       BRK
.. C01D  00       BRK
.. C01E  00       BRK
.. C01F  00       BRK
.. C020  20 C3 FF JSR $FFC3
.. C023  A2 00    LDX #$00
.. C025  BD 40 C0 LDA $C040,X
.. C028  95 00    STA $00,X
.. C02A  E8       INX
.. C02B  D0 F8    BNE $C025
.. C02D  A5 FD    LDA $FD
.. C02F  8D 01 08 STA $0801
.. C032  A5 FE    LDA $FE
.. C034  8D 02 08 STA $0802
.. C037  60       RTS
.
```

*NOTE: ZMACH can be used to recover any program after a crash and reset! Use following procedure:

1. First load "ZMACH",8,1
2. Type NEW [RETURN].
3. Load your program, type SYS49152 [RETURN]
4. To restore your program after a reset or new, type SYS49184 [RETURN]

20 NO HEADER

```
3300  A5 06     LDA $06
3302  85 08     STA $08
3304  A5 07     LDA $07
3306  85 09     STA $09
3308  A9 78     LDA #$78
330A  8D 01 00  STA $0001
330D  20 95 F3  JSR $F395
3310  20 10 F5  JSR $F510
3313  A0 00     LDY #$00
3315  B8        CLV
3316  50 FE     BVC $3316
3318  88        DEY
3319  D0 FA     BNE $3315
331B  20 56 F5  JSR $F556
331E  A9 FF     LDA #$FF
3320  8D 03 1C  STA $1C03
3323  AD 0C 1C  LDA $1C0C
3326  29 1F     AND #$1F
3328  09 C0     ORA #$C0
332A  8D 0C 1C  STA $1C0C
332D  A9 00     LDA #$00
332F  8D 01 1C  STA $1C01
3332  A0 00     LDY #$00
3334  B8        CLV
3335  50 FE     BVC $3335
3337  88        DEY
3338  D0 FA     BNE $3334
333A  20 00 FE  JSR $FE00
333D  A9 00     LDA #$00
333F  8D 01 00  STA $0001
3342  A0 00     LDY #$00
3344  20 95 F3  JSR $F395
3347  A9 01     LDA #$01
3349  4C 69 F9  JMP $F969
```

21 ERASE TRACK

```
3300  AD 0C 1C  LDA $1C0C
3303  29 1F     AND #$1F
3305  09 C0     ORA #$C0
3307  8D 0C 1C  STA $1C0C
330A  A9 FF     LDA #$FF
330C  8D 03 1C  STA $1C03
330F  A9 55     LDA #$55
3311  8D 01 1C  STA $1C01
3314  A2 28     LDX #$28
3316  A0 00     LDY #$00
3318  50 FE     BVC $3318
331A  B8        CLV
331B  88        DEY
331C  D0 FA     BNE $3318
331E  CA        DEX
331F  D0 F7     BNE $3318
3321  20 00 FE  JSR $FE00
3324  A9 01     LDA #$01
3326  4C 69 F9  JMP $F969
```

22 NO DATA

```
3300  A5 06     LDA $06
3302  85 08     STA $08
3304  A5 07     LDA $07
3306  85 09     STA $09
3308  A9 78     LDA #$78
330A  8D 01 00  STA $0001
330D  20 95 F3  JSR $F395
3310  20 10 F5  JSR $F510
3313  A9 FF     LDA #$FF
3315  8D 03 1C  STA $1C03
3318  AD 0C 1C  LDA $1C0C
331B  29 1F     AND #$1F
331D  09 C0     ORA #$C0
331F  8D 0C 1C  STA $1C0C
3322  A9 00     LDA #$00
3324  8D 01 1C  STA $1C01
3327  A0 00     LDY #$00
3329  B8        CLV
332A  50 FE     BVC $332A
332C  88        DEY
332D  D0 FA     BNE $3329
332F  20 00 FE  JSR $FE00
3332  A9 00     LDA #$00
3334  8D 01 00  STA $0001
3337  A9 01     LDA #$01
3339  4C 69 F9  JMP $F969
```

23 DATCHKSUM

```
3300  20 E9 F5  JSR $F5E9
3303  49 77     EOR #$77
3305  85 3A     STA $3A
3307  A9 90     LDA #$90
3309  85 00     STA $00
330B  4C 86 F5  JMP $F586
```

SYNC WRITER

```
3300  AD 0C 1C  LDA $1C0C
3303  29 1F     AND #$1F
3305  09 C0     ORA #$C0
3307  8D 0C 1C  STA $1C0C
330A  A9 FF     LDA #$FF
330C  8D 03 1C  STA $1C03
330F  A2 28     LDX #$28
3311  A0 00     LDY #$00
3313  A9 00     LDA #$00
3315  8D 03 1C  STA $1C03
3318  50 FE     BVC $3318
331A  B8        CLV
331B  88        DEY
331C  D0 FA     BNE $3318
331E  A0 04     LDY #$04
3320  A9 FF     LDA #$FF
3322  8D 03 1C  STA $1C03
3325  50 FE     BVC $3325
3327  B8        CLV
3328  88        DEY
3329  D0 FA     BNE $3325
332B  CA        DEX
332C  D0 E5     BNE $3313
332E  20 00 FE  JSR $FE00
3331  A9 01     LDA #$01
3333  4C 69 F9  JMP $F969
```

COPY HDR

```
3300  20 10 F5 JSR $F510
3303  A2 00    LDX #$00
3305  50 FE    BVC $3305
3307  B8       CLV
3308  CA       DEX
3309  D0 FA    BNE $3305
330B  A2 54    LDX #$54
330D  50 FE    BVC $330D
330F  B8       CLV
3310  CA       DEX
3311  D0 FA    BNE $330D
3313  A9 FF    LDA #$FF
3315  8D 03 1C STA $1C03
3318  AD 0C 1C LDA $1C0C
331B  29 1F    AND #$1F
331D  09 C0    ORA #$C0
331F  8D 0C 1C STA $1C0C
3322  A9 FF    LDA #$FF
3324  A2 05    LDX #$05
3326  8D 01 1C STA $1C01
3329  B8       CLV
332A  50 FE    BVC $332A
332C  B8       CLV
332D  CA       DEX
332E  D0 FA    BNE $332A
3330  A2 0F    LDX #$0F
3332  A0 00    LDY #$00
3334  B9 00 04 LDA $0400,Y
3337  50 FE    BVC $3337
3339  B8       CLV
333A  8D 01 1C STA $1C01
333D  C8       INY
333E  CA       DEX
333F  D0 F3    BNE $3334
3341  50 FE    BVC $3341
3343  B8       CLV
3344  AD 0C 1C LDA $1C0C
3347  09 E0    ORA #$E0
3349  8D 0C 1C STA $1C0C
334C  A9 00    LDA #$00
334E  8D 03 1C STA $1C03
3351  A9 00    LDA #$00
3353  85 01    STA $01
3355  A0 00    LDY #$00
3357  20 95 F3 JSR $F395
335A  A9 01    LDA #$01
335C  4C 69 F9 JMP $F969
```

CON HDR

```
5300  A9 00    LDA #$00
5302  45 16    EOR $16
5304  45 17    EOR $17
5306  45 18    EOR $18
5308  45 19    EOR $19
530A  85 1A    STA $1A
530C  20 34 F9 JSR $F934
530F  A0 08    LDY #$08
5311  A2 00    LDX #$00
5313  B5 24    LDA $24,X
5315  9D E0 03 STA $03E0,X
5318  E8       INX
5319  88       DEY
531A  D0 F7    BNE $5313
531C  60       RTS
```

1CON HDR

```
5300  A9 00    LDA #$00
5302  45 16    EOR $16
5304  45 17    EOR $17
5306  45 18    EOR $18
5308  45 19    EOR $19
530A  49 77    EOR #$77
530C  85 1A    STA $1A
530E  20 34 F9 JSR $F934
5311  A0 08    LDY #$08
5313  A2 00    LDX #$00
5315  B5 24    LDA $24,X
5317  9D E0 03 STA $03E0,X
531A  E8       INX
531B  88       DEY
531C  D0 F7    BNE $5315
531E  60       RTS
```

WRITE HDR

```
3300   20 10 F5 JSR $F510
3303   A2 00    LDX #$00
3305   50 FE    BVC $3305
3307   B8       CLV
3308   CA       DEX
3309   D0 FA    BNE $3305
330B   A2 54    LDX #$54
330D   50 FE    BVC $330D
330F   B8       CLV
3310   CA       DEX
3311   D0 FA    BNE $330D
3313   A9 FF    LDA #$FF
3315   8D 03 1C STA $1C03
3318   AD 0C 1C LDA $1C0C
331B   29 1F    AND #$1F
331D   09 C0    ORA #$C0
331F   8D 0C 1C STA $1C0C
3322   A9 FF    LDA #$FF
3324   A2 05    LDX #$05
3326   8D 01 1C STA $1C01
3329   B8       CLV
332A   50 FE    BVC $332A
332C   B8       CLV
332D   CA       DEX
332E   D0 FA    BNE $332A
3330   A2 08    LDX #$08
3332   A0 00    LDY #$00
3334   B9 E0 03 LDA $03E0,Y
3337   50 FE    BVC $3337
3339   B8       CLV
333A   8D 01 1C STA $1C01
333D   C8       INY
333E   CA       DEX
333F   D0 F3    BNE $3334
3341   A2 05    LDX #$05
3343   A9 55    LDA #$55
3345   50 FE    BVC $3345
3347   B8       CLV
3348   8D 01 1C STA $1C01
334B   CA       DEX
334C   D0 F7    BNE $3345
334E   50 FE    BVC $334E
3350   AD 0C 1C LDA $1C0C
3353   09 E0    ORA #$E0
3355   8D 0C 1C STA $1C0C
3358   A9 00    LDA #$00
335A   8D 03 1C STA $1C03
335D   A9 00    LDA #$00
335F   85 01    STA $01
3361   A0 00    LDY #$00
3363   20 95 F3 JSR $F395
3366   A9 01    LDA #$01
3368   4C 69 F9 JMP $F969
```

READ HDR

```
.
., 4300  20 10 F5 JSR $F510
., 4303  A0 00    LDY #$00
., 4305  B8       CLV
., 4306  50 FE    BVC $4306
., 4308  88       DEY
., 4309  D0 FA    BNE $4305
., 430B  20 56 F5 JSR $F556
., 430E  A0 00    LDY #$00
., 4310  50 FE    BVC $4310
., 4312  B8       CLV
., 4313  AD 01 1C LDA $1C01
., 4316  99 00 04 STA $0400,Y
., 4319  C8       INY
., 431A  D0 F4    BNE $4310
., 431C  A0 00    LDY #$00
., 431E  50 FE    BVC $431E
., 4320  B8       CLV
., 4321  AD 01 1C LDA $1C01
., 4324  99 00 05 STA $0500,Y
., 4327  C8       INY
., 4328  D0 F4    BNE $431E
., 432A  A9 00    LDA #$00
., 432C  85 01    STA $01
., 432E  A0 00    LDY #$00
., 4330  20 95 F3 JSR $F395
., 4333  A9 01    LDA #$01
., 4335  4C 69 F9 JMP $F969
., 4338  00       BRK
., 4339  00       BRK
., 433A  00       BRK
.
```

*** CHAPTER SEVEN ***

CARTRIDGES

This chapter is primarily a reference aid for building and using the ROMULATOR system. We will summarize the major points of cartridge duplication theory, but for a complete understanding, you will need to cover the material under the Cartridge headings in chapters 2, 3, and 4.

As discussed in the earlier sections, the autorun feature of a cartridge makes it impossible to "get into" via normal methods such as STOP RESTORE and so on. The autorun is a part of the power-up job of the C64, so defeating it requires the cartridge to be "invisible" during power-up. Once the system is running under user control, you are then able to get into the cartridge for saving, disassembling and whatever else you may want to do. This is one of the main tasks of the Romulator card.

The next area of concern with cartridges is the way they reconfigure memory. Two of the lines to the cartridge (GAME and EXROM) control the way the C64 organizes its internal RAM and ROM. There are four combinations which are shown in table 3.3. Once again, the Romulator circuit card overrides the cartridge and allows you to change the configuration. In this manner you can easily find out the normal configuration used by the cartridge.

The fact that a cartridge is a ROM based program affords it another kind of protection. That is that the program can "write" data to the ROM locations, but this will not change the data actually stored there. In other words, the program cannot "erase itself"! If you run a ROM based program in RAM, a write enable line is necessary. This enable can be turned OFF which in effect protects the RAM from write-over. It then works just like a ROM with the exception that it will lose data when power is removed. The Romulator card has a write enable switch as well as a socket into which you plug an 8K or 16K Vic-20 RAM expander card. These were chosen because they are readily available at a reasonable price. Many C64 owners started with a Vic-20 and still have expander cards. In the case of 16K duplication the card will also need block switching. (The Romulator Switch Card is illustrated in this chapter and is available for those with Commodore cartridges without the switching capability) If your

RAM already has the ability to switch either half of the 16K into block one or two or OFF, you will not need a switchcard. (Note that this reference is to the Vic-20 block one and two, not C64 RAM area)

It is possible to run some cartridge programs in the computer RAM. The general procedure is to find the normal location of the ROM program, load or transfer the contents of the ROM to its equivalent location in RAM. You must then determine the normal entry point of the program (sometimes the first two bytes of the program) and SYS to that location. You may wish to experiment with this procedure but we have not found it predictable enough to write about. The Romulator system on the other hand, has proven effective for every cartridge we have thus far encountered.

ROMULATOR HARDWARE
SYSTEM DESCRIPTION

The Romulator system consists of a special circuit card and a program which are used together to transfer the contents of ROM based cartridges to tape or disk. We will refer to the copies as "cartridge tapes" or "cartridge disks". To run the cartridge tapes or disks you must supply a 8K or 16K Vic-20 RAM expander (for 16K cartridges you must have 16K RAM with block switching). The Romulator with RAM plugged in need not be removed from the expansion port. It will not affect normal computer operations. Cartridge disks/tapes can be loaded and run at will. In addition, a cartridge program can be "switched" out and in without reloading, as long as power is not interrupted. Another handy feature when using a 16K RAM is that two 8K cartridge disks/tapes can be resident at one time allowing you to simply switch between them. An edge connector at the back allows cartridges to also be plugged in and operated without removing Romulator.

THEORY OF OPERATION

Figure 7.1 shows the Romulator schematic. Note that the data and address bus (A0-A12) pass uninterrupted to both the cartridge and RAM slots on the Romulator card. The GAME and EXROM lines however are intercepted by the card so that you have complete manual control of the system configuration. A Reset button is also provided for cold starts. Write enable is on board in case your RAM does not provide it.

The key to the system is the 74138 decoder and block switching system. We have chosen the $4000 block to serve as RAM buffer for cartridges since it allows other user programs below; and 16K worth of transfered ROM above. The Romulator program will transfer the contents of cartridges to $4000, and also saves them from there. A jumper rail is provided if you wish to modify this for your own reasons.

The ROML and ROMH lines are normally used by the computer to select the cartridge according to the location forced by EXROM and GAME lines. With these

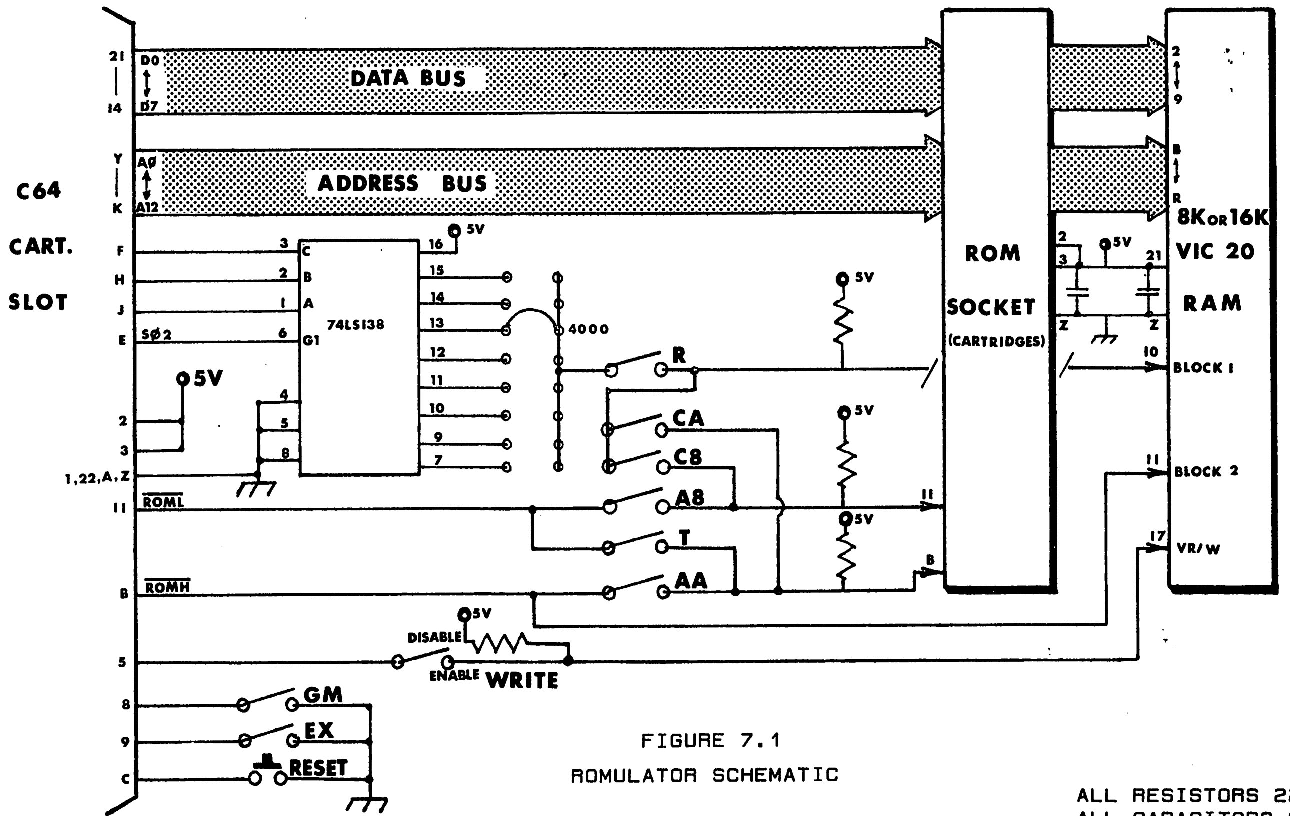


FIGURE 7.1
ROMULATOR SCHEMATIC

ALL RESISTORS 22K OHMS
ALL CAPACITORS 0.1 MFD

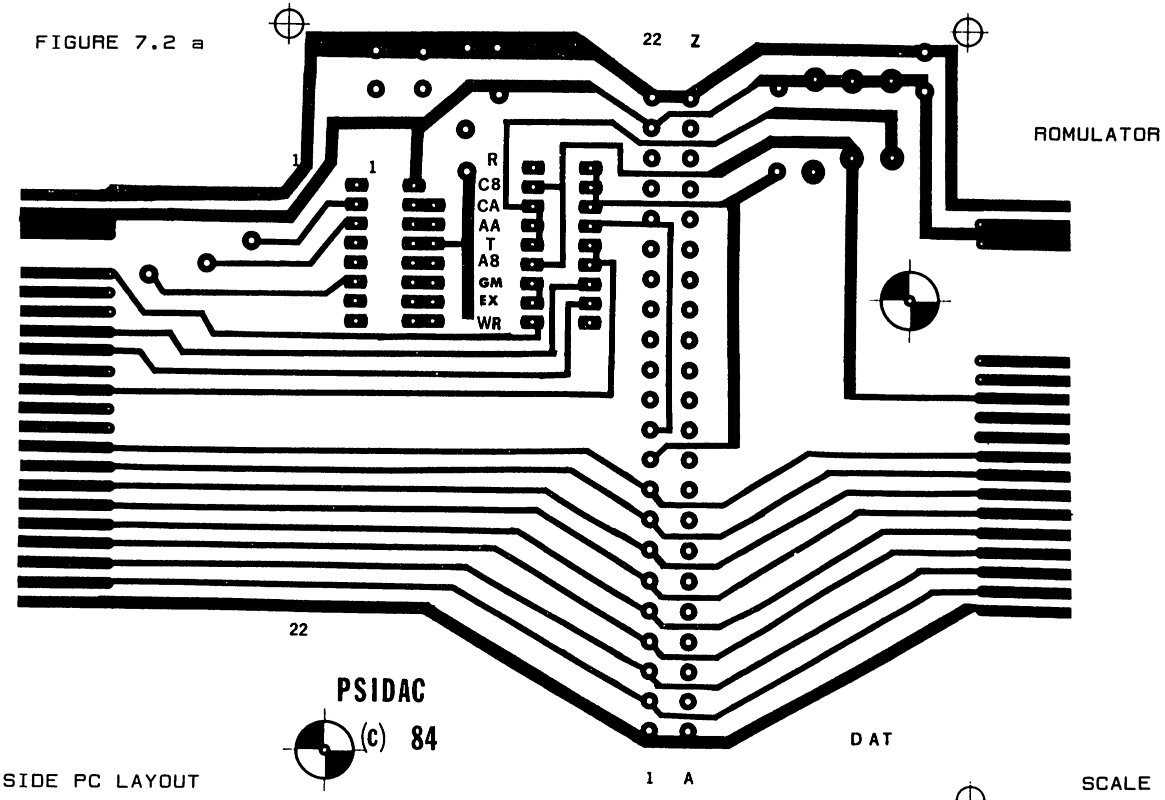
lines open and GAME and EXROM open, the computer cannot "see" the cartridge. To avoid conflicts with the basic ROM (ROMH), we will always connect the ROM to be copied to the ROML line. The T switch makes this transfer if the cartridge is normally selected by ROMH. By first turning on EXROM, reseting, then switching the cartridge into ROML, any ROM can be forced to appear at $8000 without running! The RAM need not be active at this time although it does not affect the operation if it is. The Romulator program then performs the transfer to $4000 and the Save. The process is repeated for 16K cartridges. For 16Ks, you will first save the ROML half then the ROMH half using the T switch.

The loading process requires the RAM to be situated at $4000 with the R switch, write Enable should be ON. A normal load is done, then if 16K the load is repeated with the other half of the RAM and other half of the cartridge disk. The RAM is then switched into the location used by that cartridge using the CA and AA switch or the C8 and A8 switches. (Switches ending in 'A for ROMH cartridges and '8 for ROMLs). Then by switching OFF write enable and setting GAME and EXROM, the RESET will start the cartridge program!

Though the theory may seem a little confusing, the step by step procedure given later will make the process quite simple.

ASSEMBLY

FIGURE 7.2 a

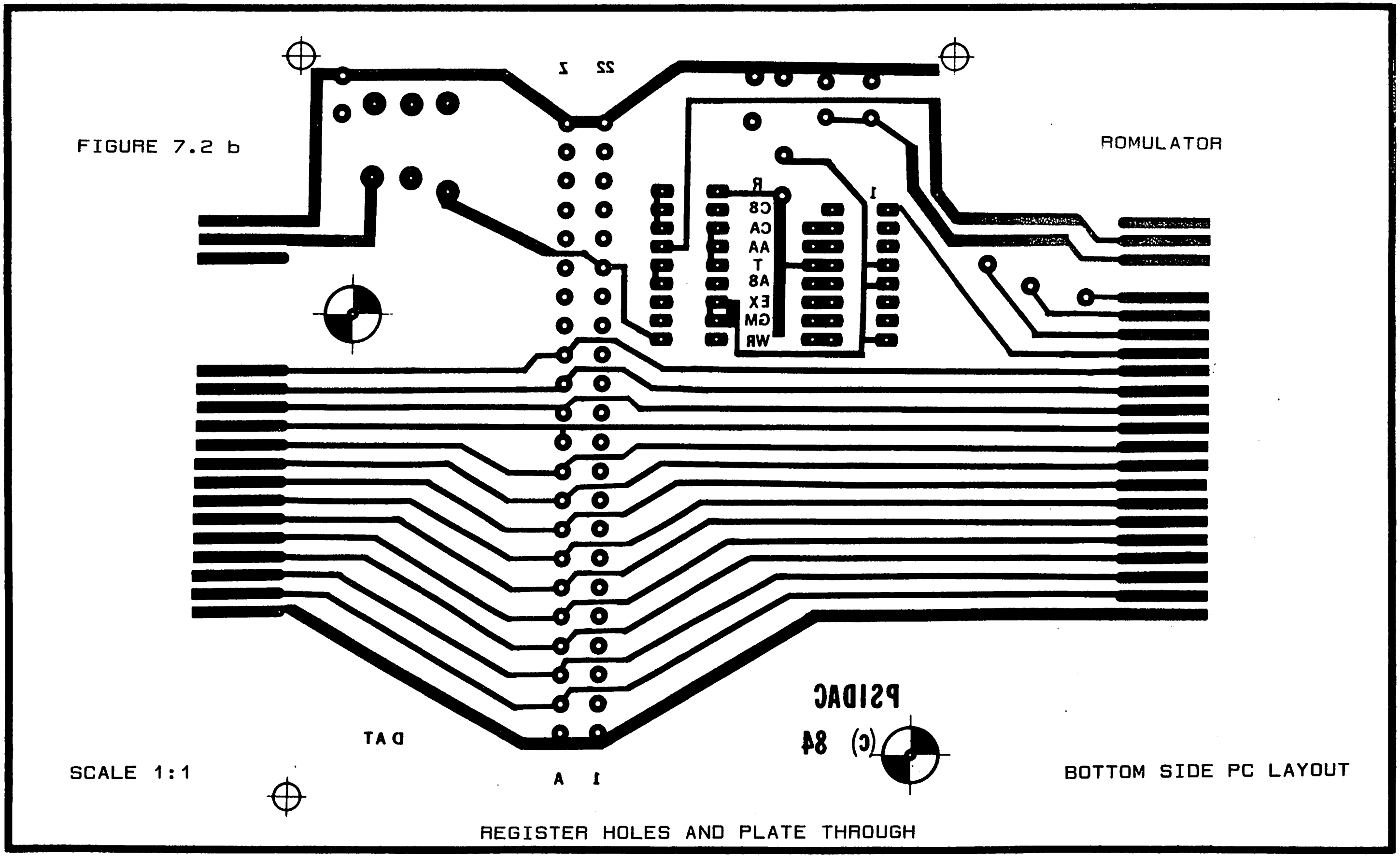


TOP SIDE PC LAYOUT

SCALE 1:1

NOTE: REGISTER HOLES AND PLATE THRU

FIGURE 7.2 b


ROMULATOR
BOTTOM SIDE PC LAYOUT
SCALE 1:1
REGISTER HOLES AND PLATE THROUGH
PSIDAC
(c) 84
DAT
A
1
Z
22
CB
CA
AA
T
8A
EX
GM
RW

We do not recommend that you try to build your own circuits unless you have had considerable experience doing so. The damage possible to your computer system through improperly built circuits would far outweigh the slight cost advantage you may obtain.

If you have the experience necessary you will find the layout straight-forward. The ROM socket is soldered directly to the pads at the end of the board. Note the correct orientation of the IC and switches. The RAM socket is mounted near the center of the board. The PC board is double sided, note the Top and Bottom side layouts. All through-hole feeds should be soldered top and bottom side or plated through. Be careful not to make solder bridges between adjacent traces on the PC board. See Appendix G for assembled unit availability.

16K RAM SWITCH MODIFICATION

If you own a Commodore 16K RAM and wish to be able to do 16K cartridges, you will need a RAM switch card. No changes are required if you are only interested in doing 8K cartridges and will be using an 8K expander addressed for block one. (Vic-20 block) Figures 7.3 and 7.4 show the schematic and PC layout for this. With the card installed in your RAM, you will have the ability to switch either 8K half of the 16K into block one or two. Figure 7.5 shows the positioning of the card in the Commodore RAM case. If you have another brand of

FIGURE 7.3
RAM SWITCHCARD

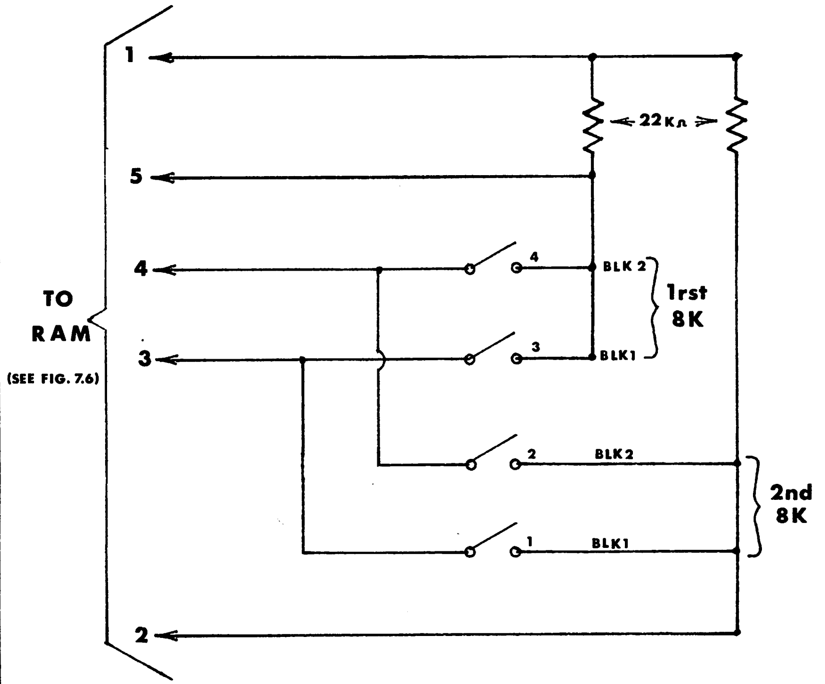
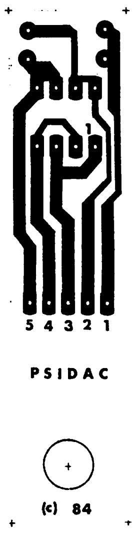
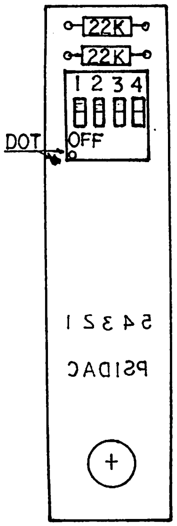


FIGURE 7.4 FOIL AND COMPONENT LAYOUTS (shown 1:1 size)

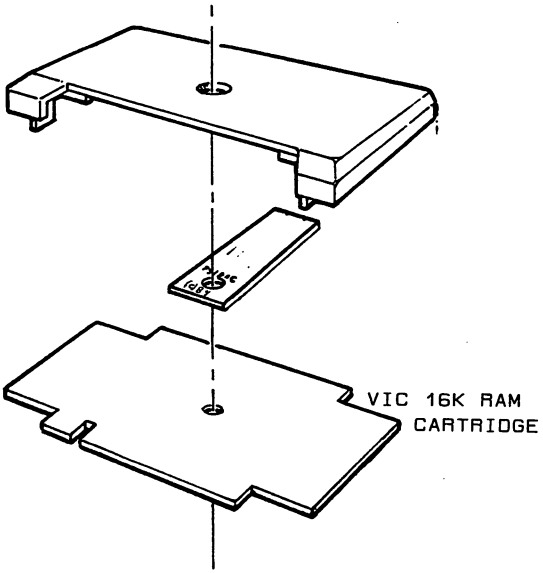




COMPONENT SIDE

FIGURE 7.5

SWITCH CARD OR ENTATION



RAM CARTRIDGE SHOWN FROM BOTTOM

RAM, you will have to have some other switching system.

16K RAM SWITCH INSTALLATION
FOR COMMODORE RAMS

1. Disassemble 16K case. (1 screw, 4 snaps)
2. Orient RAM SWITCH as shown in figure 7.5.
3. Use small piece of tape to hold board as shown in figure 7.6. You should be looking at the soldered side of the RAM and the foil side of the SWITCH CARD. The large hole in the SWITCH CARD should be aligned with the screw hole of the RAM.
4. Pretin SWITCH CARD numbered pads with SMALL puddle of solder.
5. Tack solder kynar wire from numbered pads on SWITCH CARD to like numbered pads on RAM. (1 to 1, 2 to 2, 3 to 3...)
6. Make certain jumpers on RAM at location indicated are cut. If switch is in this position, make sure all switches are OFF. (Some have 4 switches for half and a jumper for other half. All must be OPEN!
7. Double check all connections. Make certain that wire insulation is close enough to pad so inter-pad shorts cannot occur.

YOUR NEW BLOCK SELECT SWITCHES ARE:

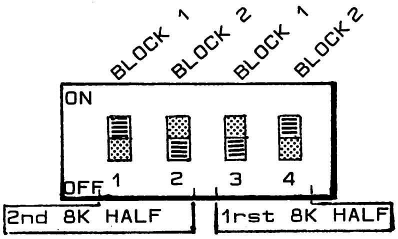
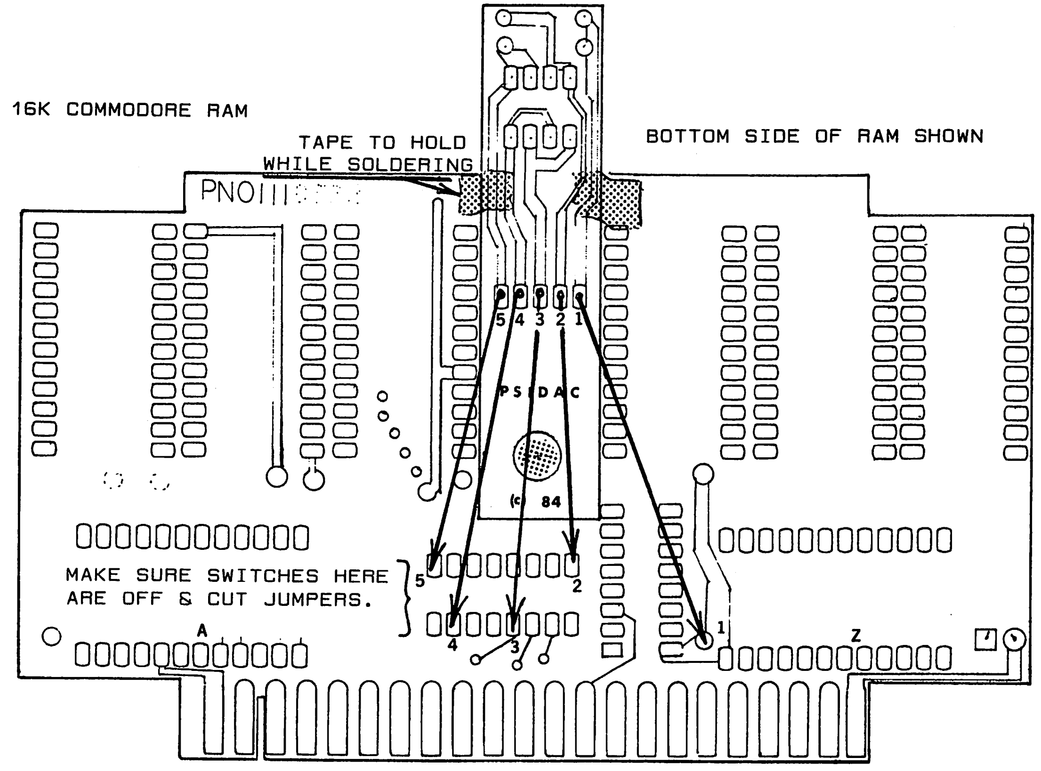


FIGURE 7.6
SWITCHCARD TO RAM WIRING



(8K RAMS DO NOT REQUIRE SWITCH)

ROMULATOR PROCEDURE 8K

The following instructions give the step by step procedure for using the Romulator system. We will refer all operations to disk. Just replace word "disk" with "tape" if you are using tape. Tape requires 3ROMULATOR program, disk uses 2ROMULATOR or, abbreviated, 2R*. Note that the Romulator circuit card with RAM may be left "permanently" plugged into computer. If you are not otherwise using expansion port it will not affect normal operation. All steps below assume it is already plugged in.

SAVING 8K CARTRIDGES

1. With power OFF, plug cartridge to copy into the Romulator cartridge socket. All Romulator switches should be OFF.
2. Turn ON computer. Using table 7.1, try each combination of GAME, EXROM, A8, and AA. (follow each by pressing RESET). One of the combinations should result in RUNNING of the cartridge. If not, it may be 16K, go to 16K save procedure. WRITE DOWN the combination used which made it RUN. (Such as A8-C8-GM-EX)
3. Turn OFF all switches. Turn ON EXROM and press RESET. 30719 Bytes Free message should appear.
4. Using information obtained in above steps, now turn ON switch A8 if you wrote down A8. Use switch T if you

wrote down AA.

5. LOAD appropriate Romulator program then perform SYS3291. Romulator title notice should appear. Follow prompts. NAME should be less than 16 characters ...Do NOT use quotes.

6. For additional backups repeat step 5 from SYS3291 only!

7. NOTE: A copy can be made from a copy by using editor assembler and loading cartridge disk then Saving $4000 to $6000. You may find this easier than using the ROMULATOR procedure to make backups of the disk at a later date.

LOADING-RUNNING CARTRIDGE DISKS/TAPES

1. Switch all Romulator Switches OFF. Turn ON switch R, and WRITE ENABLE. Switch RAM switch so RAM is in Block one. (When loading second half of 16K, use second half of RAM in block one...First half all OFF.)

2. Press RESET. 38911 message should appear. LOAD cartridge disk using "prg name",8,1 or tape with "prg name",1,1.

3. When done loading, Switch WRITE ENABLE OFF. (For 16K switch first half of RAM <u>out</u> of block one and switch second half <u>in</u> then repeat step 2. When <u>done</u>, Write Enable OFF, first half of RAM into block 1 second half into block 2 , also turn R OFF!)

4. Turn ON the switches you wrote down from SAVE procedure table 7.1. This information should be kept with the program. One convenient way is to include as part of

program name.

5. Press RESET button and program should RUN!

--TABLE 7.1--

TEST				WRITE DOWN
GM	EX	A8	AA	
ON	OFF	ON	OFF	A8-C8-GM
OFF	ON	ON	OFF	A8-C8-EX
ON	ON	ON	OFF	A8-C8-GM-EX
ON	OFF	OFF	ON	AA-CA-GM
OFF	ON	OFF	ON	AA-CA-EX
ON	ON	OFF	ON	AA-CA-GM-EX

16K SAVE PROCEDURE

The main difference between 16K and 8K procedure is that the 16K has to be handled in halves. You should think of your RAM as having a First half and a Second half. The ROM will be saved one half at a time. Each half is saved identically except that the T switch is used to place the second half of the ROM into position so Romulator can see and save it. These steps assume that you already know the cartridge is a 16K as discovered by the first few steps of the 8K procedure.

1. Use the first three patterns of TEST on table 7.1 to determine the RUN configuration. However, AA & A8 should both be on for each 16K test.
2. When you discover the correct pattern for RUNNING write down the configuration indicated.
3. Turn OFF all switches. Turn ON EXROM and press RESET.

30719 message should appear.

4. LOAD the appropriate version of Romulator (tape or disk). SYS3291 , Romulator title should appear. SWITCH A8 ON!
5. Follow prompts. For NAME use a 1 followed by program name to indicate 1rst half. Name must be less than 16 characters, do NOT use quotes.
6. When done turn OFF all switches and turn ON EXROM. Press RESET, 30719 message should appear. Turn ON T.
7. Type SYS3291. (You do not need to reload it as long as power wasn't interupted).
8. Again follow prompts. Use a 2 in front of name to indicate second half of program.
9. This completes 16K SAVE procedure. You now have the program in two halves called "1name" and "2name". The LOAD RUN procedure covers the method of running both 8K and 16K cartridge disks/tapes.

--- PROGRAMS ---

Following are the listings in assembly code (machine code is used for data tables). Note that there is a data table starting at $0A00 and extending to the beginning of the program area, $0BD6. We recommend the use of an editor assembler Memory command to type in the table and the Assemble command for the program. 2ROMULATOR is for disk saves and 3ROMULATOR is for tapes. PSIDAC supplies a complete disk of the programs in this book if you do not wish to type these by hand.

3ROM DATA TABLE

```
0A00 0B 0B 11 F8 11 23 03 0F
0A08 01 01 00 00 80 00 40 20
0A10 00 60 00 00 00 00 00 00
0A18 00 00 00 00 00 00 00 00
0A20 00 00 00 00 00 00 93 05
0A28 20 20 20 20 20 20 A4 A4
0A30 A4 A4 A4 A4 A4 A4 A4 A4
0A38 A4 A4 A4 A4 A4 A4 A4 A4
0A40 A4 A4 A4 A4 A4 A4 A4 A4
0A48 A4 A4 20 20 0D 20 20 20
0A50 20 20 20 74 9F 33 52 4F
0A58 4D 55 4C 41 54 4F 52 20
0A60 9E 28 43 29 1E 50 53 49
0A68 44 41 43 20 38 34 2D 56
0A70 4E 05 A7 20 0D 20 20 20
0A78 20 20 20 A3 A3 A3 A3 A3
0A80 A3 A3 A3 A3 A3 A3 A3 A3
0A88 A3 A3 A3 A3 A3 A3 A3 A3
0A90 A3 A3 A3 A3 A3 A3 A3 20
0A98 20 0D 20 20 20 20 A4 A4
0AA0 A4 A4 A4 A4 A4 A4 A4 A4
0AA8 A4 A4 A4 A4 A4 A4 A4 A4
0AB0 A4 A4 A4 A4 A4 A4 A4 A4
0AB8 A4 A4 A4 A4 A4 A4 20 0D
0AC0 04 20 20 20 20 2A 20 2A
0AC8 20 43 41 52 54 52 49 44
0AD0 47 45 20 42 41 43 4B 55
0AD8 50 20 50 52 4F 47 52 41
0AE0 4D 20 2A 20 2A 20 20 0D
0AE8 20 20 20 20 A3 A3 A3 A3
0AF0 A3 A3 A3 A3 A3 A3 A3 A3
0AF8 A3 A3 A3 A3 A3 A3 A3 A3
0B00 A3 A3 A3 A3 A3 A3 A3 A3
0B08 A3 A3 A3 A3 20 20 20 11
0B10 0D 20 20 20 20 20 28 31
0B18 29 20 52 45 41 44 59 20
0B20 54 41 50 45 20 55 4E 49
0B28 54 20 46 4F 52 20 53 41
0B30 56 45 20 20 20 11 0D 20
0B38 20 20 20 20 28 32 29 20
0B40 50 52 45 53 53 20 46 37
0B48 20 20 11 0D 04 04 20 20
0B50 20 20 20 28 33 29 20 49
0B58 4E 50 55 54 20 50 52 4F
0B60 47 52 41 4D 20 4E 41 4D
0B68 45 20 20 11 0D 20 20 20
0B70 20 20 28 34 29 20 04 04
0B78 04 04 04 11 20 20 20 20
0B80 20 28 35 29 20 4F 50 45
0B88 52 41 54 49 4F 4E 20 43
0B90 4F 4D 50 4C 45 54 45 20
0B98 52 45 4D 4F 56 45 20 20
0BA0 0D 20 20 20 20 20 20 20
0BA8 20 20 54 41 50 45 20 46
0BB0 52 4F 4D 20 55 4E 49 54
0BB8 20 11 20 20 0D 20 20 20
0BC0 20 20 28 36 29 20 50 52
0BC8 45 53 53 20 46 37 20 0D
0BD0 20 20 04 04 04 04 AD 0B
```

```
0BD6   AD 0B 0A LDA $0A0B
0BD9   85 FB    STA $FB
0BDB   AD 0C 0A LDA $0A0C
0BDE   85 FC    STA $FC
0BE0   AD 0D 0A LDA $0A0D
0BE3   85 FD    STA $FD
0BE5   AD 0E 0A LDA $0A0E
0BE8   85 FE    STA $FE
0BEA   AD 00 0A LDA $0A00
0BED   8D 20 D0 STA $D020
0BF0   AD 01 0A LDA $0A01
0BF3   8D 21 D0 STA $D021
0BF6   AD 07 0A LDA $0A07
0BF9   8D 18 D4 STA $D418
0BFC   AD 02 0A LDA $0A02
0BFF   8D 05 D4 STA $D405
0C02   AD 03 0A LDA $0A03
0C05   8D 06 D4 STA $D406
0C08   AD 06 0A LDA $0A06
0C0B   8D 00 D4 STA $D400
0C0E   AD 05 0A LDA $0A05
0C11   8D 01 D4 STA $D401
0C14   60       RTS
0C15   00       BRK
0C16   00       BRK
0C17   00       BRK
0C18   A2 00    LDX #$00
0C1A   BD 26 0A LDA $0A26,X
0C1D   C9 04    CMP #$04
0C1F   F0 06    BEQ $0C27
0C21   20 D2 FF JSR $FFD2
0C24   E8       INX
0C25   D0 F3    BNE $0C1A
0C27   60       RTS
0C28   00       BRK
0C29   00       BRK
0C2A   00       BRK
0C2B   A2 00    LDX #$00
0C2D   BD C1 0A LDA $0AC1,X
0C30   C9 04    CMP #$04
0C32   F0 06    BEQ $0C3A
0C34   20 D2 FF JSR $FFD2
0C37   E8       INX
0C38   D0 F3    BNE $0C2D
0C3A   60       RTS
0C3B   00       BRK
0C3C   00       BRK
0C3D   00       BRK
0C3E   A2 00    LDX #$00
0C40   BD 4E 0B LDA $0B4E,X
0C43   C9 04    CMP #$04
0C45   F0 06    BEQ $0C4D
0C47   20 D2 FF JSR $FFD2
0C4A   E8       INX
```

```
0C4B   D0 F3    BNE $0C40
0C4D   60       RTS
0C4E   00       BRK
0C4F   00       BRK
0C50   00       BRK
0C51   A2 00    LDX #$00
0C53   BD 7B 0B LDA $0B7B,X
0C56   C9 04    CMP #$04
0C58   F0 06    BEQ $0C60
0C5A   20 D2 FF JSR $FFD2
0C5D   E8       INX
0C5E   D0 F3    BNE $0C53
0C60   60       RTS
0C61   00       BRK
0C62   00       BRK
0C63   00       BRK
0C64   20 E4 FF JSR $FFE4
0C67   C9 88    CMP #$88
0C69   D0 F9    BNE $0C64
0C6B   60       RTS
0C6C   00       BRK
0C6D   00       BRK
0C6E   AE 05 0A LDX $0A05
0C71   8E 04 D4 STX $D404
0C74   CA       DEX
0C75   8E 04 D4 STX $D404
0C78   60       RTS
0C79   A2 00    LDX #$00
0C7B   A0 00    LDY #$00
0C7D   B1 FB    LDA ($FB),Y
0C7F   91 FD    STA ($FD),Y
0C81   C8       INY
0C82   C0 00    CPY #$00
0C84   D0 F7    BNE $0C7D
0C86   E6 FC    INC $FC
0C88   E6 FE    INC $FE
0C8A   E8       INX
0C8B   EC 0F 0A CPX $0A0F
0C8E   D0 EB    BNE $0C7B
0C90   60       RTS
0C91   00       BRK
0C92   00       BRK
0C93   00       BRK
0C94   A0 00    LDY #$00
0C96   20 CF FF JSR $FFCF
0C99   99 16 0A STA $0A16,Y
0C9C   C8       INY
0C9D   C9 0D    CMP #$0D
0C9F   D0 F5    BNE $0C96
0CA1   20 6E 0C JSR $0C6E
0CA4   8C 0A 0A STY $0A0A
0CA7   AD 08 0A LDA $0A08
0CAA   AE 09 0A LDX $0A09
0CAD   A0 FF    LDY #$FF
```

```
0CAF   20 BA FF JSR $FFBA
0CB2   AD 0A 0A LDA $0A0A
0CB5   A2 16    LDX #$16
0CB7   A0 0A    LDY #$0A
0CB9   20 BD FF JSR $FFBD
0CBC   AD 0D 0A LDA $0A0D
0CBF   85 FD    STA $FD
0CC1   AD 0E 0A LDA $0A0E
0CC4   85 FE    STA $FE
0CC6   A9 0D    LDA #$0D
0CC8   20 D2 FF JSR $FFD2
0CCB   A9 FD    LDA #$FD
0CCD   AE 10 0A LDX $0A10
0CD0   AC 11 0A LDY $0A11
0CD3   20 D8 FF JSR $FFD8
0CD6   60       RTS
0CD7   00       BRK
0CD8   00       BRK
0CD9   00       BRK
0CDA   00       BRK
0CDB   20 D6 0B JSR $0BD6
0CDE   20 18 0C JSR $0C18
0CE1   20 2B 0C JSR $0C2B
0CE4   20 64 0C JSR $0C64
0CE7   20 6E 0C JSR $0C6E
0CEA   20 79 0C JSR $0C79
0CED   20 3E 0C JSR $0C3E
0CF0   20 94 0C JSR $0C94
0CF3   20 51 0C JSR $0C51
0CF6   20 6E 0C JSR $0C6E
0CF9   20 64 0C JSR $0C64
0CFC   20 6E 0C JSR $0C6E
0CFF   60       RTS
```

2ROM. DATA TABLE

```
0A00 0B 0B 11 F8 11 23 03 0F
0A08 08 08 00 00 80 00 40 20
0A10 00 60 00 00 00 00 00 00
0A18 00 00 00 00 00 00 00 00
0A20 00 00 00 00 00 00 93 05
0A28 20 20 20 20 20 20 A4 A4
0A30 A4 A4 A4 A4 A4 A4 A4 A4
0A38 A4 A4 A4 A4 A4 A4 A4 A4
0A40 A4 A4 A4 A4 A4 A4 A4 A4
0A48 A4 A4 20 20 0D 20 20 20
0A50 20 20 20 74 9F 32 52 4F
0A58 4D 55 4C 41 54 4F 52 20
0A60 9E 28 43 29 1E 50 53 49
0A68 44 41 43 20 38 34 2D 56
0A70 4E 05 A7 20 0D 20 20 20
0A78 20 20 20 A3 A3 A3 A3 A3
0A80 A3 A3 A3 A3 A3 A3 A3 A3
0A88 A3 A3 A3 A3 A3 A3 A3 A3
0A90 A3 A3 A3 A3 A3 A3 A3 20
0A98 20 0D 20 20 20 20 A4 A4
0AA0 A4 A4 A4 A4 A4 A4 A4 A4
0AA8 A4 A4 A4 A4 A4 A4 A4 A4
0AB0 A4 A4 A4 A4 A4 A4 A4 A4
0AB8 A4 A4 A4 A4 A4 A4 20 0D
0AC0 04 20 20 20 20 2A 20 2A
0AC8 20 43 41 52 54 52 49 44
0AD0 47 45 20 42 41 43 4B 55
0AD8 50 20 50 52 4F 47 52 41
0AE0 4D 20 2A 20 2A 20 20 0D
0AE8 20 20 20 20 A3 A3 A3 A3

0AF0 A3 A3 A3 A3 A3 A3 A3 A3
0AF8 A3 A3 A3 A3 A3 A3 A3 A3
0B00 A3 A3 A3 A3 A3 A3 A3 A3
0B08 A3 A3 A3 A3 20 20 20 11
0B10 0D 20 20 20 20 20 28 31
0B18 29 20 49 4E 53 45 52 54
0B20 20 43 4F 50 59 20 44 49
0B28 53 4B 20 49 4E 20 44 52
0B30 49 56 45 20 20 11 0D 20
0B38 20 20 20 20 28 32 29 20
0B40 50 52 45 53 53 20 46 37
0B48 20 20 11 0D 04 04 20 20
0B50 20 20 20 28 33 29 20 49
0B58 4E 50 55 54 20 50 52 4F
0B60 47 52 41 4D 20 4E 41 4D
0B68 45 20 20 11 0D 20 20 20
0B70 20 20 28 34 29 20 04 04
0B78 04 04 04 11 20 20 20 20
0B80 20 28 35 29 20 4F 50 45
0B88 52 41 54 49 4F 4E 20 43
0B90 4F 4D 50 4C 45 54 45 20
0B98 52 45 4D 4F 56 45 20 20
0BA0 0D 20 20 20 20 20 20 20
0BA8 20 20 44 49 53 4B 20 46
0BB0 52 4F 4D 20 44 52 49 56
0BB8 45 11 20 20 0D 20 20 20
0BC0 20 20 28 36 29 20 50 52
0BC8 45 53 53 20 46 37 20 0D
0BD0 11 11 04 04 04 04 AD 0B
```

```
0BD6  AD 0B 0A LDA $0A0B
0BD9  85 FB    STA $FB
0BDB  AD 0C 0A LDA $0A0C
0BDE  85 FC    STA $FC
0BE0  AD 0D 0A LDA $0A0D
0BE3  85 FD    STA $FD
0BE5  AD 0E 0A LDA $0A0E
0BE8  85 FE    STA $FE
0BEA  AD 00 0A LDA $0A00
0BED  8D 20 D0 STA $D020
0BF0  AD 01 0A LDA $0A01
0BF3  8D 21 D0 STA $D021
0BF6  AD 07 0A LDA $0A07
0BF9  8D 18 D4 STA $D418
0BFC  AD 02 0A LDA $0A02
0BFF  8D 05 D4 STA $D405
0C02  AD 03 0A LDA $0A03
0C05  8D 06 D4 STA $D406
0C08  AD 06 0A LDA $0A06
0C0B  8D 00 D4 STA $D400
0C0E  AD 05 0A LDA $0A05
0C11  8D 01 D4 STA $D401
0C14  60       RTS
0C15  00       BRK
0C16  00       BRK
0C17  00       BRK
0C18  A2 00    LDX #$00
0C1A  BD 26 0A LDA $0A26,X
0C1D  C9 04    CMP #$04
0C1F  F0 06    BEQ $0C27
0C21  20 D2 FF JSR $FFD2
0C24  E8       INX
0C25  D0 F3    BNE $0C1A
0C27  60       RTS
0C28  00       BRK
0C29  00       BRK
0C2A  00       BRK
0C2B  A2 00    LDX #$00
0C2D  BD C1 0A LDA $0AC1,X
0C30  C9 04    CMP #$04
0C32  F0 06    BEQ $0C3A
0C34  20 D2 FF JSR $FFD2
0C37  E8       INX
0C38  D0 F3    BNE $0C2D
0C3A  60       RTS
0C3B  00       BRK
0C3C  00       BRK
0C3D  00       BRK
0C3E  A2 00    LDX #$00
0C40  BD 4E 0B LDA $0B4E,X
0C43  C9 04    CMP #$04
0C45  F0 06    BEQ $0C4D
0C47  20 D2 FF JSR $FFD2
0C4A  E8       INX
0C4B  D0 F3    BNE $0C40
0C4D  60       RTS
0C4E  00       BRK
0C4F  00       BRK
0C50  00       BRK
0C51  A2 00    LDX #$00
0C53  BD 7B 0B LDA $0B7B,X
0C56  C9 04    CMP #$04
0C58  F0 06    BEQ $0C60
0C5A  20 D2 FF JSR $FFD2
0C5D  E8       INX
0C5E  D0 F3    BNE $0C53
0C60  60       RTS
0C61  00       BRK
0C62  00       BRK
0C63  00       BRK
0C64  20 E4 FF JSR $FFE4
0C67  C9 88    CMP #$88
0C69  D0 F9    BNE $0C64
0C6B  60       RTS
0C6C  00       BRK
0C6D  00       BRK
0C6E  AE 05 0A LDX $0A05
0C71  8E 04 D4 STX $D404
0C74  CA       DEX
0C75  8E 04 D4 STX $D404
0C78  60       RTS
0C79  A2 00    LDX #$00
0C7B  A0 00    LDY #$00
0C7D  B1 FB    LDA ($FB),Y
0C7F  91 FD    STA ($FD),Y
0C81  C8       INY
0C82  C0 00    CPY #$00
0C84  D0 F7    BNE $0C7D
0C86  E6 FC    INC $FC
0C88  E6 FE    INC $FE
0C8A  E8       INX
0C8B  EC 0F 0A CPX $0A0F
0C8E  D0 EB    BNE $0C7B
0C90  60       RTS
0C91  00       BRK
0C92  00       BRK
0C93  00       BRK
0C94  A0 00    LDY #$00
0C96  20 CF FF JSR $FFCF
0C99  99 16 0A STA $0A16,Y
0C9C  C8       INY
0C9D  C9 0D    CMP #$0D
0C9F  D0 F5    BNE $0C96
0CA1  8C 0A 0A STY $0A0A
0CA4  AD 08 0A LDA $0A08
0CA7  AE 09 0A LDX $0A09
0CAA  A0 FF    LDY #$FF
0CAC  20 BA FF JSR $FFBA
```

```
0CAF   AD 0A 0A LDA $0A0A
0CB2   A2 16    LDX #$16
0CB4   A0 0A    LDY #$0A
0CB6   20 BD FF JSR $FFBD
0CB9   AD 0D 0A LDA $0A0D
0CBC   85 FD    STA $FD
0CBE   AD 0E 0A LDA $0A0E
0CC1   85 FE    STA $FE
0CC3   A9 FD    LDA #$FD
0CC5   AE 10 0A LDX $0A10
0CC8   AC 11 0A LDY $0A11
0CCB   20 D8 FF JSR $FFD8
0CCE   60       RTS
0CCF   20 6E 0C JSR $0C6E
0CD2   20 E4 FF JSR $FFE4
0CD5   C9 88    CMP #$88
0CD7   D0 F6    BNE $0CCF
0CD9   60       RTS
0CDA   00       BRK
0CDB   20 D6 0B JSR $0BD6
0CDE   20 18 0C JSR $0C18
0CE1   20 2B 0C JSR $0C2B
0CE4   20 64 0C JSR $0C64
0CE7   20 6E 0C JSR $0C6E
0CEA   20 79 0C JSR $0C79
0CED   20 3E 0C JSR $0C3E
0CF0   20 94 0C JSR $0C94
0CF3   20 51 0C JSR $0C51
0CF6   20 CF 0C JSR $0CCF
0CF9   60       RTS
```

APPENDIX A
HEX-CHR$ & SCREEN CODES CHART

CBM VALUE		CHR DISPLAYED		SCREEN CODE VALUE	
HEX	CHR$	MODE-UPR	MODE-LWR	HEX SCRN	DEC SCRN
00	0				
01	1				
02	2				
03	3				
04	4				
05	5	WHT			
06	6				
07	7				
08	8	SH/C OFF			
09	9	SH/C ON			
0A	10				
0B	11				
0C	12				
0D	13	RET			
0E	14	LWR/CS ON			
0F	15				
10	16				
11	17	C/DN			
12	18	RVS/ON			
13	19	CLR/HM			
14	20	INST/DEL			
15	21				
16	22				
17	23				
18	24				
19	25				
1A	26				
1B	27				
1C	28	RED			
1D	29	C/RT			
1E	30	GRN			
1F	31	BLU			
20	32	SPACE		20	32
21	33	!	!	21	33
22	34	"	"	22	34
23	35	#	#	23	35
24	36	$	$	24	36
25	37	%	%	25	37
26	38	&	&	26	38
27	39	'	'	27	39
28	40	(	(	28	40
29	41	)	)	29	41
2A	42	*	*	2A	42
2B	43	+	+	2B	43
2C	44	,	,	2C	44
2D	45	-	-	2D	45
2E	46	.	.	2E	46
2F	47	/	/	2F	47
30	48	0	0	30	48
31	49	1	1	31	49
32	50	2	2	32	50

33	51	3	3	33	51
34	52	4	4	34	52
35	53	5	5	35	53
36	54	6	6	36	54
37	55	7	7	37	55
38	56	8	8	38	56
39	57	9	9	39	57
3A	58	:	:	3A	58
3B	59	;	;	3B	59
3C	60	<	<	3C	60
3D	61	=	=	3D	61
3E	62	>	>	3E	62
3F	63	?	?	3F	63
40	64	@	@	00	0
41	65	A	a	01	1
42	66	B	b	02	2
43	67	C	c	03	3
44	68	D	d	04	4
45	69	E	e	05	5
46	70	F	f	06	6
47	71	G	g	07	7
48	72	H	h	08	8
49	73	I	i	09	9
4A	74	J	j	0A	10
4B	75	K	k	0B	11
4C	76	L	l	0C	12
4D	77	M	m	0D	13
4E	78	N	n	0E	14
4F	79	O	o	0F	15
50	80	P	p	10	16
51	81	Q	q	11	17
52	82	R	r	12	18
53	83	S	s	13	19
54	84	T	t	14	20
55	85	U	u	15	21
56	86	V	v	16	22
57	87	W	w	17	23
58	88	X	x	18	24
59	89	Y	y	19	25
5A	90	Z	z	1A	26
5B	91	[	[	1B	27
5C	92	£	£	1C	28
5D	93	]	]	1D	29
5E	94	↑	↑	1E	30
5F	95			1F	31
60	96	S/*	S/*	40	64
61	97	S/A	A	41	65
62	98	S/B	B	42	66
63	99	S/C	C	43	67
64	100	S/D	D	44	68
65	101	S/E	E	45	69
66	102	S/F	F	46	70
67	103	S/G	G	47	71
68	104	S/H	H	48	72
69	105	S/I	I	49	73
6A	106	S/J	J	4A	74
6B	107	S/K	K	4B	75
6C	108	S/L	L	4C	76
6D	109	S/M	M	4D	77
6E	110	S/N	N	4E	78

6F	111	S/O	O	4F	79
70	112	S/P	P	50	80
71	113	S/Q	Q	51	81
72	114	S/R	R	52	82
73	115	S/S	S	53	83
74	116	S/T	T	54	84
75	117	S/U	U	55	85
76	118	S/V	V	56	86
77	119	S/W	W	57	87
78	120	S/X	X	58	88
79	121	S/Y	Y	59	89
7A	122	S/Z	Z	5A	90
7B	123	S/+		5B	91
7C	124	C/-		5C	92
7D	125	S/-		5D	93
7E	126	Pi	Lg CKRBD	5E	94
7F	127	C/*	Diag Lns	5F	95
80	128				
81	129				
82	130				
83	131				
84	132				
85	133	f1			
86	134	f3			
87	135	f5			
88	136	f7			
89	137	f2			
8A	138	f4			
8B	139	f6			
8C	140	f8			
8D	141	SH/RET			
8E	142	UPR CS ON			
8F	143				
90	144	BLK			
91	145	CR/UP			
92	146	RVS ON			
93	147	CLR/HM			
94	148	INST/DEL			
95	149				
96	150				
97	151				
98	152				
99	153				
9A	154				
9B	155				
9C	156	PUR			
9D	157	CR/LFT			
9E	158	YEL			
9F	159	CYN			
A0	160	SPC		60	96
A1	161	C/K		61	97
A2	162	C/I		62	98
A3	163	C/T		63	99
A4	164	C/@		64	100
A5	165	C/G		65	101
A6	166	C/+		66	102
A7	167	C/M		67	103
A8	168	C/£		68	104
A9	169	S/£	Diag Lns	69	105
AA	170	C/N		6A	106

AB	171	C/Q		6B	107
AC	172	C/D		6C	108
AD	173	C/Z		6D	109
AE	174	C/S		6E	110
AF	175	C/P		6F	111
B0	176	C/A		70	112
B1	177	C/E		71	113
B2	178	C/R		72	114
B3	179	C/W		73	115
B4	180	C/H		74	116
B5	181	C/J		75	117
B6	182	C/L		76	118
B7	183	C/Y		77	119
B8	184	C/U		78	120
B9	185	C/O		79	121
BA	186	S/@	CK MARK	7A	122
BB	187	C/F		7B	123
BC	188	C/C		7C	124
BD	189	C/X		7D	125
BE	190	C/V		7E	126
BF	191	C/B		7F	127

NOTE: This table shows which characters will be displayed for any value of CBM ASCII code. The two left columns give the values in hex and decimal (dec used for CHR$). The 2nd and 3rd columns show what will be displayed or what control mode affected, depending whether the computer is in the UPPER CASE MODE (column 2), or LOWER CASE MODE (column 3). The last two columns give the hex and decimal values used for the SCREEN DISPLAY CODES. The decimal value is used to POKE directly to screen locations, while the hex is used if Storing to screen locations from a machine routine.. Note that S/ followed by a letter means the SHIFTED letter which is the right hand graphic symbol on your key. A C/ means the COMMODORE KEY which produces the left hand graphic symbol on your keyboard.

APPENDIX B

MONITOR USE WITH DISKPICKER

Other monitors can be substituted for MONITOR$8000 in Diskpicker by changing line 10 with alternate monitor's name, and by changing the SYS to the monitor's entry point in line 310 to that of the monitor you are using. Note that the alternate monitor should reside between $8000-$BFFF.

All Diskpicker functions except #2 - Monitor Mode can be used without a Monitor. This gives you the ability to load and execute error utilities etc. The procedure for this is:

1. Delete line 10. Change line 20 to read: IFA=0 THEN A=1 :LOAD "ZMACH",8,1
2. Press RUN/STOP - RESTORE then load "the desired utility program",8,1 then type NEW [RETURN]
3. SYS49184 [RETURN] Then follow normal Diskpicker procedure section for error writing routine.

HESMON- cartridge can be used in lieu of MONITOR$8000. However, you will lose the ability to easily recover from lockup conditions which normally require reset and ZMACH restore features. Use the following changes and procedure:

1. DELETE DISKPICKER LINE 10 and 20. Change line 35 to read: RUN40
2. Change line 310 to read SYS 36466
3. TO USE- Plug in HESMON, turn computer ON, EXIT HESMON with "XC" command.
4. Load "DISKPICKER",8 and modify as above then RUN.
5. Use RESTORE key to enter HESMON instead of Menu item 2! - To return from HESMON, type "X" command then RUN.

SUPERMON64.V1 -Is part of the Commodore 64 Software Bonus Pack and seems to be readily available in the public domain. It can be used in place of Monitor$8000. Modify Diskpicker as follows:

1. Load "SUPERMON64.V1",8 then RUN. Monitor prompt screen should appear. Use "X" command to EXIT to basic. Insert PSIPACK disk and LOAD "DISKPICKER",8
2. Delete line 10. Change line 20 IF A=0 THEN A=1:LOAD"ZMACH",8,1
3. Change line 310 to read: POKE53281,3 : SYS(PEEK(43)+256*PEEK(44)+127)
4. RUN Diskpicker as per procedure sections.
5. If you save the MODIFIED DISKPICKER, you need follow only step 1 in order to RUN Diskpicker with SUPERMON!!
6. If you are unable to locate SUPERMON or the COMMODORE BONUS PACK, we will supply a copy of SUPERMON for a handling fee of $5.00.

If your MONITOR$8000 will not save itself, change hex locations $86EC through $86F7 to EA (These are NOPs). MONITOR$8000 can be easily saved using RELOCATE/LOADER procedures.

APPENDIX C
AUTO-BOOTERS

Following are two samples of autorun booters. One is for booting machine routines and the other for basic. We have chosen 02A7-030C for the booter routines. Use an editor to assemble them in these locations. Remember that when you load the routine from the tape or disk it will call the program identified by the NAME bytes. To set the names and messages you wish to use, look up the HEX CBM ASCII value for the letter and control functions in appendix A.

The first part of the routine sends a name to the screen to indicate that the booter is operating. The routine for machine has more available space for this message. The basic routine simply sends the name that you are calling with the loader.

Next the standard Commodore format for setting logical file, device number, name, and loading the RAM is used. Note that the device number is listed as $08 (disk) tape is $01.

Finally, in the machine booter, it jumps to the starting address of the machine routine that was loaded. The basic routine sets necessary basic pointers and restores 0302-030B vectors before it runs.

It is a good idea to use the no-break Poke 808,255 in your basic program. You should play around with some simple test routines before you commit the loader to service.

To set the names and messages you wish to use, look up the HEX CBM ASCII value for the letter and control functions in appendix A.

APPENDIX C

BASIC AUTOBOOTER

```
.
.
.

.,  02A7  A2 00     LDX #$00        GET NAME AT
.,  02A9  BD F6 02  LDA $02F6,X     02F6
.,  02AC  20 D2 FF  JSR $FFD2       SEND TO
.,  02AF  E8        INX             SCREEN
.,  02B0  E0 04     CPX #$04
.,  02B2  D0 F5     BNE $02A9
.,  02B4  A9 10     LDA #$10        SET
.,  02B6  A2 08     LDX #$08        LFS
.,  02B8  A0 01     LDY #$01
.,  02BA  20 BA FF  JSR $FFBA
.,  02BD  A9 04     LDA #$04        NAME (AT
.,  02BF  A2 F6     LDX #$F6 -      02F6)
.,  02C1  A0 02     LDY #$02 -
.,  02C3  20 BD FF  JSR $FFBD
.,  02C6  A9 00     LDA #$00        LOAD
.,  02C8  85 9D     STA $9D         RAM
.,  02CA  20 D5 FF  JSR $FFD5
.,  02CD  86 2D     STX $2D
.,  02CF  84 2E     STY $2E
.,  02D1  A2 0C     LDX #$0C        RESET
.,  02D3  BD E9 02  LDA $02E9,X     VEC.
.,  02D6  9D FF 02  STA $02FF,X     EX.
.,  02D9  CA        DEX             BASIC
.,  02DA  D0 F7     BNE $02D3       PRG.
.,  02DC  A9 00     LDA #$00
.,  02DE  85 7A     STA $7A
.,  02E0  A9 08     LDA #$08
.,  02E2  85 7B     STA $7B
.,  02E4  20 60 A6  JSR $A660
.,  02E7  4C AE A7  JMP $A7AE
.,  02EA  8B
.,  02EB  E3
.,  02EC  83
.,  02ED  A4 7C
.,  02EF  A5 1A
.,  02F1  A7
.,  02F2  E4 A7
.,  02F4  86 AE
.
.          HEX DATA

.:02EA 8B E3 83 A4 7C A5 1A A7
.:02F2 E4 A7 86 AE 54 45 53 54
.:02FA 02 86 AE 00 00 00 36 34
.:0302 A7 02 A7 02 A7 02 A7 02
.:030A A7 02 00 00 00 00 4C 48

.'  02EA.C.$ %.'  8B E3 83 A4 7C A5 1A A7
.'  02F2D'..TEST E4 A7 86 AE 54 45 53 54
.'  02FA......64 02 86 AE 00 00 00 36 34
.'  0302'.'.'.'. A7 02 A7 02 A7 02 A7 02
.'  030A'.....LH A7 02 00 00 00 00 4C 48
```

APPENDIX C cont

MACHINE AUTOBOOTER

```
.,  02A7   A2 00      LDX #$00          SEND
.,  02A9   BD D0 02   LDA $02D0,X       SCREEN
.,  02AC   20 D2 FF   JSR $FFD2         MSG
.,  02AF   E8         INX
.,  02B0   E0 1E      CPX #$1E
.,  02B2   D0 F5      BNE $02A9
.,  02B4   A9 10      LDA #$10          LFS
.,  02B6   A2 08      LDX #$08
.,  02B8   A0 01      LDY #$01
.,  02BA   20 BA FF   JSR $FFBA
.,  02BD   A9 04      LDA #$04          NME
.,  02BF   A2 F0      LDX #$F0
.,  02C1   A0 02      LDY #$02
.,  02C3   20 BD FF   JSR $FFBD         LOAD
.,  02C6   A9 00      LDA #$00          MACH. PGM
.,  02C8   85 9D      STA $9D
.,  02CA   20 D5 FF   JSR $FFD5
.,  02CD   4C 00 45   JMP $4500      — EX MACH.
.
```

```
.
           MSG & NME  HEX
.:02D0 93 05 20 20 50 53 49 44
.:02D8 41 43 20 1E 41 55 54 02
.:02E0 42 4F 4F 54 20 9F 20 20
.:02E8 20 20 20 20 20 20 20 20
.:02F0 4D 41 43 48 00 00 00 00
.:02F8 00 00 00 00 00 00 00 00
.:0300 36 34 A7 02 A7 02 A7 02
.:0308 A7 02 A7 02 00 00 00 00
.
```

HEX DATA SHOWING ASCII CHRs

```
       ASCII
' 02D0..  PSID  93 05 20 20 50 53 49 44
' 02D8AC .AUT.  41 43 20 1E 41 55 54 02
' 02E0BOOT .    42 4F 4F 54 20 9F 20 20
' 02E8          20 20 20 20 20 20 20 20
' 02F0MACH....  4D 41 43 48 00 00 00 00
' 02F8........  00 00 00 00 00 00 00 00
' 030064'.'.'.  36 34 A7 02 A7 02 A7 02
' 0308'.'.....  A7 02 A7 02 00 00 00 00
```

SECTOR EXPLANATIONS

The following information shows typical directory and data sectors. The important bytes have been numbered and the list identifies the meaning of the numbered bytes. The first part is for the directory and the second part for program data sectors.
**Tells how to undelete a file.

```
   1  2  3  4  5  6  7  8  9  10 11 12 <-- BYTE #
   12 04 82 11 00 31 50 53 49 4D 41 49     15,16
13 4E A0 A0 A0 A0 A0 A0 A0 A0 00 00 00
14 00 00 00 00 00 00 02 00 00 00 82 11
   01 32 50 53 49 4D 41 49 4E A0 A0 A0
   A0 A0 A0 A0 A0 00 00 00 00 00 00 00
   00 00 02 00 00 00 82 13 00 53 55 50
   45 52 44 49 52 45 43 54 4F 52 59 A0
   A0 00 00 00 00 00 00 00 00 00 08 00
   00 00 82 13 0D 44 49 53 4B 2D 45 44
   49 54 4F 52 A0 A0 A0 A0 A0 00 00 00
   00 00 00 00 00 00 07 00 00 00 82 11
   13 52 45 4C 4F 43 41 54 45 2F 4C 4F
   41 44 45 52 A0 00 00 00 00 00 00 00
   00 00 02 00 00 00 82 10 02 44 49 53
   4B 50 49 43 4B 45 52 A0 A0 A0 A0 A0
   A0 00 00 00 00 00 00 00 00 00 11 00
   00 00 82 13 09 54 2F 53 20 41 4E 41
   4C 59 5A 45 52 A0 A0 A0 A0 00 00 00
   00 00 00 00 00 00 08 00 00 00 82 11
   02 46 41 53 54 42 41 43 4B A0 A0 A0
   A0 A0 A0 A0 A0 00 00 00 00 00 00 00
   00 00 07 00
   TRACK  18  SECTOR  1
   12 07 82 11 0F 31 44 55 50 44 41 43
   A0 A0 A0 A0 A0 A0 A0 A0 A0 00 00 00
   00 00 00 00 00 00 11 00 00 00 82 10
   07 32 44 55 50 44 41 43 A0 A0 A0 A0
   A0 A0 A0 A0 A0 00 00 00 00 00 00 00
   00 00 11 00 00 00 82 14 02 41 44 4D
   41 43 48 A0 A0 A0 A0 A0 A0 A0 A0 A0
   A0 00 00 00 00 00 00 00 00 00 01 00
   00 00 82 14 03 4D 41 43 48 52 45 4C
   4F A0 A0 A0 A0 A0 A0 A0 A0 00 00 00
   00 00 00 00 00 00 01 00 00 00 82 14
   04 33 52 4F 4D 55 4C 41 54 4F 52 A0
   A0 A0 A0 A0 A0 00 00 00 00 00 00 00
   00 00 04 00 00 00 82 14 06 32 52 4F
   4D 55 4C 41 54 4F 52 A0 A0 A0 A0 A0
   A0 00 00 00 00 00 00 00 00 00 04 00
   00 00 82 13 08 5A 4D 41 43 48 A0 A0
   A0 A0 A0 A0 A0 A0 A0 A0 A0 00 00 00
   00 00 00 00 00 00 01 00 00 00 82 14
   09 41 4E 41 4C 59 4D 41 43 48 A0 A0
   A0 A0 A0 A0 A0 00 00 00 00 00 00 00
   00 00 01 00
   TRACK  18  SECTOR  4
```

1. Next directory track-12HEX =18DEC.
2. Next sector-04HEX =04DEC.
3. Type of file-82HEX
   00=DELETED
   81=SEQUENTIAL
   82=PROGRAM
   84=RELATIVE
4. Disk File starting track- 11HEX =17DEC
5. Disk File starting sector- 00HEX =00DEC.
6. File NAME-1
7. -P
8. -S
9. -I
10. -M
11. -A
12. -I
13. -N
14. Ending characters -A0
15. Next Entry file type-82HEX
16. Next sector
   Data continues as above to the end of directory.

*File name data is the hex conversion of CHR$ codes.

** UNDELETE A FILE

You can undelete a file only if the data has not been written over. Any time something has been saved on a disk with a deleted file, some of the data has likely been written over. To undelete a file, change the file type entry back to its original number. ie:On a program file this byte should be changed to 82.

The following information explains how program sectors are configured.

```
 1  2  3  4  5  6  7
11 0A 01 08 10 08 01 00 9E 34 39 38
39 31 3A 50 B2 30 00 1B 08 02 00 9E
34 39 38 34 34 00 3D 08 03 00 9E 34
39 35 31 36 3A 58 B2 C2 28 34 39 31
36 30 29 3A 91 58 89 34 2C 33 2C 35
2C 31 30 00 76 08 04 00 54 B2 C2 28
34 39 31 35 36 29 3A 53 B2 C2 28 34
39 31 35 37 29 3A 98 31 35 2C 22 55
31 3A 22 32 3B 30 3B 54 3B 53 3A 8D
32 35 3A 9E 34 39 36 39 31 3A 89 33
00 85 08 05 00 9E 34 39 38 35 38 3A
5A B2 30 00 A9 08 06 00 9E 34 39 39
33 39 3A 58 B2 C2 28 34 39 31 36 31
29 3A 91 58 89 37 2C 36 2C 32 2C 31
30 2C 35 00 DC 08 07 00 54 B2 C2 28
34 39 31 35 38 29 3A 53 B2 C2 28 34
39 31 35 39 29 3A 9E 34 39 37 34 35
3A 8B C2 28 34 39 31 37 33 29 B2 31
AF 5A B2 31 A7 36 00 11 09 08 00 98
31 35 2C 22 42 2D 50 3A 22 32 3B 30
3A 9E 34 39 37 31 38 3A 98 31 35 2C
22 55 32 3A-8
TRACK  17  SECTOR  0
9-00 9E 22 32 3B 30 3B 54 3B 53 3A 8D
32 35 3A 5A B2 31 3A 89 36 00 1E 09
0A 00 9E 34 39 38 37 32 3A 80 00 4A
09 19 00 99 22 05 11 54 52 4B 2E 22
54 22 9D 20 20 22 3A 99 22 11 53 45
43 2E 22 53 22 9D 20 22 3A 84 31 35
2C 41 24 2C 42 24 00 6D 09 1A 00 99
22 11 45 52 52 2E 20 22 41 24 22 91
91 91 91 91 91 22 3A 8B 41 24 B2 22
30 30 22 A7 8E 00 8B 09 1B 00 8B 50
B2 31 A7 9F 34 2C 34 3A 98 34 2C 54
2C 53 2C 41 24 2C 42 24 3A A0 34 00
98 09 1C 00 9E 34 39 39 30 33 3A BE-10
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00
TRACK  17  SECTOR  10
```

1. Next track -11HEX =17DEC

2. Next sector -0AHEX=10DEC

3. Load TO address low byte=01

4. Load TO address high byte=08

*Thus load address =$0801 or 2049 DECIMAL.

5. Program DATA (in HEX)

6. " "

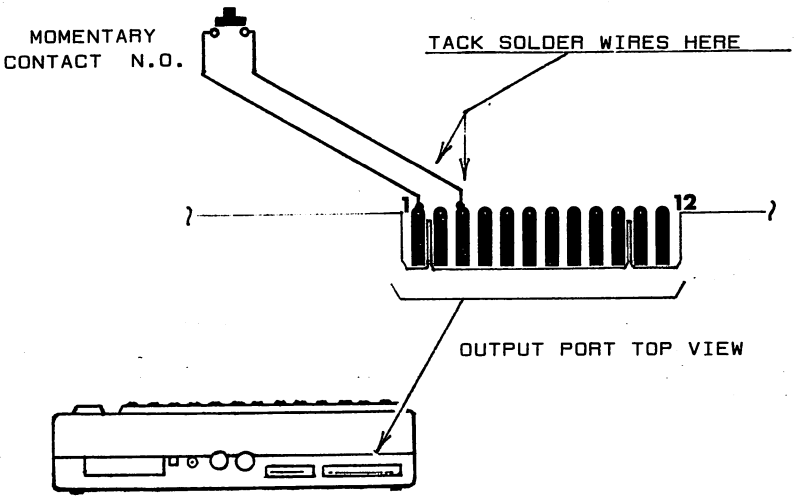
7. Continues to end (see next sector and explanation#9 and #10)

8. End byte of THIS sector. (part of data)

9. Indicates no next sector if 00, otherwise same as above except 4 & 5 would be data not addresses.

10. Last program byte.

APPENDIX E
RESET SWITCH



--INSTALLATION INSTRUCTIONS --

A Reset switch is useful for breaking out of lockup conditions without erasing RAM data. Tack solder two fine wires to OUTPUT PORT pins 1 and 3. To keep solder from flowing over the entire edge connector pad, you should put a piece of masking tape over the pads, exposing only the top end for soldering. The switch can be mounted at any convenient location where it is not likely to be accidentally pushed. If the location you use is not close to the port, loop the wires through the ferrite donut to preserve EMI shielding properties.

APPENDIX F
GCR INFORMATION

The normal binary information sent from the computer to the disk is the binary equivalent of the CBM ASCII. However when the information is to be stored on a diskette, the DOS converts the CBM ASCII to another format called Group Coded Recording (GCR). In this process, the standard eight bit code has two extra bits added to it. Thus the GCR equivalent of the ASCII is a 10 bit number. When you load this GCR data into RAM and view it, you are only seeing eight of the ten bits at a time, so it is more difficult to analyze the HEX GCR code. The other effect this has is that a 256 byte block actually takes about 320 8-bit locations on the disk.

Following we have listed a GCR Sector Map, an example printout of a GCR Sector Image, and a list of the Disk drive memory locations that hold GCR header images.

IMPORTANT DISK MEMORY LOCATIONS
HEADER IMAGES

```
LOC.     BINARY DATA
$0016    Disk ID HI
$0017    Disk ID LO
$0018    Track
$0019    Sector
$001A    Checksum
------ GCR DATA ------
$0024    Header block ID (HBID)
$0025    Checksum
$0026    Sector
$0027    Track
$0028
$0029    ID1
$002A    ID2
$002B
```

GCR BYTE MAP FOR A SECTOR

```
BEGIN SECTOR
FF    (5) Sync Bytes
FF
FF
52      Header block ID byte
XX      (7) Header bytes
XX      Checksum
XX      Sector
XX      Track
XX      ID1
XX      ID2
XX
55      (11) Gap bytes
55
55
55
55
55
55
55
55
55
55
FF      (5) Sync Bytes
FF
FF
FF
55      Data block ID Byte
XX      (324) Data block bytes
XX      DATA
≈
Checksum
0f
0f
55      (7) Gap bytes
55
55
55
55
55
55
Next Sector starts here.
```

*Note: The end of track Gap contains several "AA" GCR clear bytes.

GCR IMAGE EXAMPLE

```
.:6400 52 6E E5 2D 72 9A DF 65  HEADER
.:6408 55 55 55 55 55 55 55 55  HEADER GAP (55)
.:6410 55 55 25 7F 55 D7 25 39  SYNC  (FF  or  7F)   55=Data
.:6418 32 5A D4 A9 AD EA 7C DD                        block ID
.:6420 97 75 CB 76 5D ED 2B 4A  DATA
.:6428 D2 B4 AD 2B 4A D2 B4 A5
.:6430 29 4A 52 94 A5 29 4A 52
.:6438 94 A5 29 52 52 94 A5 29
.:6440 32 5A D4 B9 C9 EA 7C DD
.:6448 97 75 CB 76 5D ED 2B 4A
.:6450 D2 B4 AD 2B 4A D2 B4 A5
.:6458 29 4A 52 94 A5 29 4A 52
.:6460 94 A5 29 52 52 94 A5 29
.:6468 32 5C D4 A7 CD EF 7A 9C
.:6470 F7 C9 CE 76 5F 27 3D D3
.:6478 7B 9D 57 C9 F9 D2 B4 A5
.:6480 29 4A 52 94 A5 29 4A 52
.:6488 94 A5 29 49 52 94 A5 29
.:6490 32 5C D5 D7 39 D9 7C DD
.:6498 B9 75 CF 73 9D 97 B9 D5
.:64A0 7C B4 AD 2B 4A D2 B4 A5
.:64A8 29 4A 52 94 A5 29 4A 52
.:64B0 94 A5 29 57 52 94 A5 29
.:64B8 32 5A D7 37 C9 CF 73 5D
.:64C0 57 4D CB 7B 9C F9 55 CD
:.:64C8 75 5C B7 39 CF 7C B4 A5
.:64D0 29 4A 52 94 A5 29 4A 52
.:64D8 94 A5 29 52 52 94 A5 29
.:64E0 32 5A 95 27 39 D9 7C DD
.:64E8 B7 A9 D9 74 DD B7 3D F2
.:64F0 D2 B4 AD 2B 4A D2 B4 A5
.:64F8 29 4A 52 94 A5 29 4A 52  DATA
.:6500 94 A5 29 6B 52 94 A5 29
.:6508 32 5C D5 97 BA 55 7C E4
.:6510 A7 2D DE 72 DC D7 E5 FA
.:6518 73 DF 2D 2B 4A D2 B4 A5
.:6520 29 4A 52 94 A5 29 4A 52
.:6528 94 A5 29 49 52 94 A5 29
.:6530 32 5A D5 27 59 CB 7C DE
.:6538 E7 49 CB 74 DD BD 2B 4A
.:6540 D2 B4 AD 2B 4A D2 B4 A5
.:6548 29 4A 52 94 A5 29 4A 52
.:6550 94 A5 29 57 52 97 D5 29
.:6558 4C A7 55 55 55 55 55 55  DATA GAP
.:6560 55 55 55 55 55 55 FF 52  FF=SYNC   52=HDR BLOCK ID
.:6568 6F 75 49 72 9A DF 65 55
.:6570 55 55 55 55 55 55 55 55 - GAP
.:6578 55 55 FF 55 DD B5 2D 4B
.:6580 52 D4 B5 2D 4B 52 D4 B5
.:6588 2D 4B 52 D4 B5 2D 4B 52
.:6590 D4 B5 2D 4B 52 D4 B5 2D
.:6598 4B 52 D4 B5 2D 4B 52 D4
.:65A0 B5 2D 4B 52 D4 B5 2D 4B     FORMATTED ONLY
.:65A8 52 D4 B5 2D 4B 52 D4 B5
.:65B0 2D 4B 52 D4 B5 2D 4B 52
.:65B8 D4 B5 2D 4B 52 D4 B5 2D
.:65C0 4B 52 D4 B5 2D 4B 52 D4
.:65C8 B5 2D 4B 52 D4 B5 2D 4B
.:65D0 52 D4 B5 2D 4B 52 D4 B5
```

APPENDIX G

PRICE LIST
884-1284

******** C-64PRODUCTS ********

PART#	DESCRIPTION	PRICE(US)
SPH-64	Software Protection Handbook for C-64	$19.95
PPK-1	PSIPACK disk of SPH-64 programs. (Instructions in SPH64 only)	$16.95
SPH-64-D	SPH-64 book and PPK-1 disk combination.	$29.95
RMA-2	Romulator for C-64 (requires 8K or 16K Vic compatible RAM expander)	$39.95
RS-2	16KRAM switch. (see text) (kit)	$9.95
SC-1	Super Clone Plug kit. (VIC & 64)	$9.95
TK-1	Tapeworm Kit. All parts, PC & inst.(VIC & 64)	$19.95
CS-1	Reset switch Kit. (VIC & 64)	$6.95
	******** VIC & Misc PRODUCTS ********	
SPH-20	Software Protection Handbook for the VIC	$9.95
PT-1	Pirate's Tape of programs in SPH-20	$9.95
RMK-1	Romulator, VIC version (kit)	$19.95
PA-1	Protection Arsenal for VIC 20. Pirates Tape, Tapeworm Kit, Vic Romulator, Super Clone kit.	$49.95
SHIRT	"Software Pirates" T-Shirt White Skull and Crossbones on jet-black Shirt. (S-M-L)	$9.95
MON-1	"SUPERMON" Disk Can be used in place of Monitor$8000. See Appendix B.	$5.00

*Shipping-- add 10% to order total. ($5.00 maximum)

*Overseas Airmail-- add $6.00 Orders including SPH-64 or many items, add $10.00

*Canada-- allow for current exchange rate or obtain money order in US dollars.

** If not available at your local dealer, you may order directly from:

PSIDAC Products Div.
7326 N. Atlantic
Portland, OR 97217

Prices subject to change without notice.

APPENDIX H
INTERRUPTS

At roughly 1/60 second intervals, the C-64 takes a "time out" from the user program to do some of its own internal housekeeping. This is the "hardware interrupt" cycle which is a normal part of computer systems. Since it is automatic by nature, the user need not worry about it for normal purposes. The user CAN take advantage of it for his own routines. The limitation is that the routines should be short and in machine language.

A control zone vector at $0314 and $0315 holds the address that tells where the routines to be executed during the interrupt are. This value is normally $EA31. You can write a routine wherever convenient and change the value at $0314 and $0315 to point to your new routine. Remember that the 64 still needs to do its own routines; so at the END of your routine you must have a JMP $EA31 so that the normal housekeeping will get done.

THIRD EDITION ADDITIONS

The following procedures work with Diskpicker to accomplish additional error writing capabilities.

CREATE ERROR #27

Places error 27 on all sectors of desired track.

1. Load "1CON HDR",08 through monitor mode of Diskpicker.

2. Load "WRITE HDR",08 through monitor mode. Then G C020

3. Put object diskette in drive.

4. Select 6 from main menu and enter track # and sector # desired to be errored. *If sector desired is in doubt, use sector #1

5. Select 3 from main menu.
```
Start Addr  0300
End   Addr  031F
Buff  Addr  5300
```

6. Select 4 from main menu. Entry addr 0300.

7. Select 3 from main menu.
```
Start Addr  0300
End   Addr  036F
Buff  Addr  3300
```

8. Select 5 from main menu Job choice= 224. For execute start addr, select 1 =(0300)

9. For "Multi Sector Y/N", select Y.

10. For HDR#1, use track# to be errored.

TO CHECK ERROR 27

1. Select 5 from main menu. Job choice = 128.

2. For "Multi Sector Y/N", select Y.

3. For HDR#1 use track # just errored in above procedure.

* Sectors should read "Checksum Error In Header"

* For more 27 errors repeat starting at step 4.

CREATE ERROR 29

Places "29" error on all sectors of desired track.

1. Load "CON HDR" through monitor mode of Diskpicker.

2. Load "WRITE HDR",08 through monitor mode. Then type G C020 [RETURN]

3. Put object diskette in drive.

4. Select 6 from main menu and enter track# and sector# desired to error. *If in doubt, use sector #1.

5. Select 1 from main menu
```
Start Addr  0016
End   Addr  001B
Buff  Addr  6016
```

6. Dump memory locations 6016-601B from monitor mode.

7. Change memory locations $6016 and $6017 to the Hex value of the desired ID#s. *Refer to appendix F.

8. Select 3 from main menu
```
Start Addr  0016
End   Addr  001B
Buff  Addr  6016
```

9. Select 3 from main menu
```
Start Addr  0300
End   Addr  031F
Buff  Addr  5300
```

10. Select 4 from main menu, Entry addr 0300

11. Select 3 from main menu
```
Start Addr  0300
End   Addr  036F
Buff  Addr  3300
```

12. Select 5 from main menu. Job choice=224. For execute start addr select 1=(0300).

13. For Multi Sector Y/N, select Y.

14. For Hdr#1 use track# to be errored.

TO CHECK ERROR 29

1. Select 5 from main menu. Job choice = 128.

2. For Multi Sector Y/N, select Y.

3. For HDR#1 use track # just errored.
 * Sectors should read "DISK ID MISMATCH"
 * For more 29 Errors, repeat procedure from step 4.

ASSIGN ILLEGAL SECTOR #"

This procedure will substitute an illegal sector # onto the diskette in place of a legal one. The illegal sector can be read using the Job Que - menu option 5. Whenever a header not found error is encountered, make sure that illegal sectoring has not been used. This is done by loading a sector using the Job Que read function to see if the sector can be found in the illegal ranges specified in the table following step 7 of this procedure.

1. Load "CON HDR" through monitor mode of Diskpicker.

2. Load "WRITE HDR" through monitor mode.

3. Put object diskette in drive.

4. Select 6 from main menu and enter track# and sector # desired to be changed to illegal value.

5. Select 1 from main menu,
```
Start Addr  0016
End   Addr  001B
Buff  Addr  6016
```

6. View memory locations 6016-601B from monitor mode. Refer to appendix F GCR info to identify byte function.

7. Change data at memory location $6019 to hex value of sector # you want to substitute for sector # chosen in step 4. The range of acceptable illegal sectors are:

TRACKS	LEGAL SECTORS	ILLEGAL SECTORS	
-----	-----	DEC.	HEX
1-17	0-20	21-25	15-19
18-24	0-18	19-23	13-17
25-30	0-17	18-22	12-16
31-35	0-16	17-21	11-15

8. Select 3 from main menu
```
Start Addr  0016
End   Addr  001B
Buff  Addr  6016
```

9. Select 3 from main menu
```
Start Addr  0300
End   Addr  031F
Buff  Addr  5300
```

10. Select 4 from main menu. Entry Addr 0300.

11. Select 3 from main menu
```
Start Addr  0300
End   Addr  036F
Buff  Addr  3300
```

12. Select 5 from main menu. Job choice =224. For Execute Start Addr, select 1 (0300)

13. For Multi Sector Y/N, select 'N'.

14. For HDR #1, use track # selected in step 6.

15. For HDR #2 Use one less than sector # selected in step 6. ("Wraparound" as in earlier procedures... ie. for sector 1 enter a 0)

TO CHECK ILLEGAL SECTOR NUMBER

1. Select 5 from main menu. Job choice= 128.

2. For Multi Sector Y/N, select 'N'.

3. For HDR #1 use track # selected in step 6 above.

4. For HDR #2, use DECIMAL value of illegal sector number you selected in step 7.

5. Even though it is an illegal value, the DOS should find the sector and load the data into the disk buffer. The sector selected in step 4 is now re-assigned to an illegal value. This can be verified by repeating check procedure steps 1-3 with a 'Y' for Multi sector question, step 2. With a little imagination, this can be used for protecting your own programs quite effectively.

SYNC ONLY

This process will cover any desired track with GCR FF's for the purpose of locking up most current whole disk copy programs. By using this routine on several unused tracks, especially the lower numbered ones, you can help protect your own software.

1. Load and Run Diskpicker.

2. Select menu option 2, printer choice 'N'.

3. Input L"SYNC ONLY",08 to load machine routine. After load is done, Type G C020

4. Place diskette to be errored into drive.

5. Select menu option 7.

6. Select menu option 3.
```
Start Addr  0300
End   Addr  034F
Buff  Addr  3300
```

7. After data transfer to disk, select menu option 5.

8. Input Job choice 224. Select execute address 1 (0300).

9. For Multi sector Y/N, select 'N'.

10. For Header #1, input track # to be errored.

11. For header #2, input sector #0.

12. Repeat procedure steps 7 through 10 for additional sync only errors.

13. To test your errors select option 5. Job Que execute addr 128.

14. For Multi Sector Y/N, select 'N'.

15. For Header #1, enter track

with error.

16. For Header #2, enter sector #0.

17. If everything went right, the disk will lock up and continue to look for the sector you selected.

18. The only way to escape this error is to open the drive door until a sync not found error shows on the monitor.

ONE TRACK FORMAT

This routine will reformat a single track. Can be used to repair any whole track error such as 22, 27, 29, Sync only, Multi-No Header. This routine can also be used to create 29 errors on a track.

** You will need a reset switch as described to facilitate the use of this procedure.

1. Load "1TRKFMT",08 through the monitor mode of diskpicker. Then type G C020

2. Put object diskette in drive.

3. Select 7 from main menu.

4. Select 2 from main menu and change data in memory locations $3312 (ID HI) and $3313 (ID LOW) to ID#s you want. IDs different than the one the disk was formatted with (False ID) will reformat the track to produce a 29 error. Correct ID#s (same as disk formatted ID) will provide normal reformat. See appendix A column 1 for the HEX values to use for ID characters of your choice. ie. the hex codes for an ID of "VN"=56 4E. (56 is ID HI and 4E is ID LOW)
When done, enter G C020

5. Select menu option 3.
```
Start Addr  0300
End   Addr  0315
Buff  Addr  3300
```

6. Select menu option 5. Job choice = 224. For execute addr, select 1 (0300).

7. For Multi Sector Y/N, choose 'N'.

8. For HDR #1 use track desired to be reformatted.

9. For HDR #2 use sector 0.

10. Wait about 5 seconds then press computer reset. (non critical time).

11. Type in SYS49184 then [RETURN].

12. Select menu option 7, if disk LED lamp flashes, repeat this step.

TO CHECK

1. Select menu option 5. Job choice=128.

2. For Multi Sector Y/N, choose 'Y'.

3. For HDR #1, select track number just reformatted.

4. Refer to the error codes returned on your monitor which indicate result of your attempts.

DETERMINING ID NUMBERS

This process will allow you to find out the Hex value of ID#s on specific tracks and sectors of the disk.

1. Assuming Diskpicker is loaded, put object diskette in drive.

2. Select 7 from main menu.

3. Select 6 from main menu, input Track # desired, Input Sector # desired.

4. Select 1 from main menu.
```
Start Addr  0012
End   Addr  001F
Buff  Addr  6012
```

5. Select 2 from menu and interrogate memory locations 6012-6018

MEMORY LOCATION DEFINITIONS

DISK ADDR	BUFF ADDR
$0012	$6012= Disk ID HI
$0013	$6013= Disk ID LOW
$0016	$6016= Sector ID HI
$0017	$6017= Sector ID LOW
$0018	$6018= Track #
$0019	$6019= Sector #
$001A	$601A= HDR Checksum

TRACK CHECK

The track check program was designed to provide the following information for full and half tracks.

1. Disk master ID in Hex.

2. Actual track found in each physical track location.

3. Header ID#s in hex for each physical track location.

4. Status of error for each physical track location.

To operate; load and run "TRACK CHECK". Follow the screen prompts to provide your choice of options. P-Trk data returns the physical track called for. D-Trk returns the actual track found in any physical track location. In some forms of protection the actual track found will be different than the normal track number designed to be in that location. IDH and IDL return the actual header ID#s found in the physical track location. If you choose the half track option the data returned may or may not be reliable because normal track widths often overlap the half track areas. This program will not return valid data for illegal tracks having abnormal bit densities. If an abnormal bit density exists on a particular physical track location, it will show up as a header not found error or some other erratic non repeatable error.

The only commercial program diskettes we have found using half track or bit density errors have been on highly protected copy programs written to run on 1541 drives only. This makes these programs incompatible on other so called "Commodore compatible" drives. We have found some commercial disks with more than one track 35. Track Check will prove useful for most present day commercial programs.

FORMATTING ILLEGAL TRACKS

This procedure will format a new illegal track onto the diskette in place of an existing legal track. This process can also be accomplished in between tracks known as half tracks. For example: Track 34 could be reformatted as a track 35 resulting in two track 34s. Likewise, a physical location of 29.5 could be reformatted as a track 33 etc.

It is best not to reformat a physical track location with format data containing more sectors than will fit in that location. To reformat physical track 35 as a track 1 would present a problem because track 1 format data contains too many sectors to fit into the physical track 35 location. It is possible to have illegal ID#s on the illegally formatted tracks.

An effective protection method can be utilized by formatting higher numbered track format data onto lower physical track locations. This makes these tracks unreadable under normal circumstances because the data will be formatted at a lower than normal bit density rate for that physical track. Disk memory location $1C00 bits 5 & 6 control the bit density rate.

BIT DENSITY CONTROL TABLE

TRACK	BITS/SEC	$1C00 Bit 6	$1C00 Bit 5
1-17	307,692.31	1	1
18-24	285,714.29	1	0
25-30	266,666.67	0	1
31-35	250,000.00	0	0

To test this concept try the following procedure by formatting physical track 1 with format data for track 35. You will notice that you can find the headers etc. using the check procedure but you will not be able to read headers/data from track 1 using the T/S analyzer or Track Check programs as the data density is not normal in this location.

The theory of this method is to make the disk drive memory think that it is set for track 35 and the read write head over track 35 while in reality the read write head is over track 1! This method also forces an illegal bit density number into memory location $1C00. By combining illegal half tracks and illegal ID#s with illegal bit densities you can achieve a high level of software protection.

ILLEGAL TRACK FORMATTING

1. Load "1TRKFMT",08 through the monitor mode of Diskpicker. Then G C020.

2. Put object diskette in drive.

3. Select 6 from main menu and enter the track # you want to use for format data. Then enter sector 0. This sets up the DOS pointers and data variables in the drive memory.

4. Select menu option 1
   Start Addr 0000
   End Addr 00FF
   Buff Addr 6000
   This stores the Disk zero page in computer buffer.
5. Select menu option 9 Position read write head. Enter your track choice for the physical track position you wish to format with the format data determined in step 3.

6. Select menu option 3.

```
Start Addr  0000
End   Addr  00FF
Buff  Addr  6000
```
This step restores the drive page zero to contain data for "illegal" track selected in step 3.

7. Select 2 from main menu and change the data in memory locations $3312(ID HI) and $3313 (ID LOW) to the ID#s you want. Then enter G C020. (If in question about ID you want to use see section on determining ID numbers or use Track Check on original program if making backups.

8. Select menu option 3.
```
Start Addr  0300
End   Addr  0315
Buff  Addr  3300
```

9. Select menu option 5. Job Choice = 224. For execute Addr select 1=(0300)

10. For multi sector Y/N select N.

11. For Hdr#1 use the data track number selected in step 3.

12. For HDR#2 use sector 0.

13. Wait about 5 seconds then press computer reset.

14. Type in SYS 49184 [RETURN]

15. Select menu option 9. For track select track 18.

16. Select menu option 7. If disk LED lamp flashes repeat this step.

* EXAMPLE
If you chose track 32 for step 3 and track 34.5 for step 4, then you will have two track #32s, one in physical track position 32 and one in physical track postion 34.5.

TO CHECK

1. Select 6 from main menu and enter the track # that you chose in step 3 of the format procedure. Then enter sector 0.

2. Select menu option 1
```
Start Addr  0000
End   Addr  00FF
Buff  Addr  6000
```

3. Select menu option 9 Position read write head. Enter the physical track position you selected in step 4 of the format procedure.

4. Select menu option 3.
```
Start Addr 0000
End   Addr 00FF
Buff  Addr 6000
```

5. Select menu option 5. Job choice =128.

6. For Multi Sector Y/N Select Y.

7. For header #1 select the track # you used in step 1 of this check procedure.

8. Refer to the error codes returned on your monitor for the results of your attempt.

9. Select menu option 9. For track select track 18.

10. Select menu option 7. If disk LED lamp flashes, repeat this step.

SYNC ONLY

```
., 3300  AD 0C 1C LDA $1C0C
., 3303  29 1F    AND #$1F
., 3305  09 C0    ORA #$C0
., 3307  8D 0C 1C STA $1C0C
., 330A  A9 FF    LDA #$FF
., 330C  8D 03 1C STA $1C03
., 330F  A9 FF    LDA #$FF
., 3311  8D 01 1C STA $1C01
., 3314  A2 28    LDX #$28
., 3316  A0 00    LDY #$00
., 3318  50 FE    BVC $3318
., 331A  B8       CLV
., 331B  88       DEY
., 331C  D0 FA    BNE $3318
., 331E  CA       DEX
., 331F  D0 F7    BNE $3318
., 3321  20 00 FE JSR $FE00
., 3324  A9 01    LDA #$01
., 3326  4C 69 F9 JMP $F969
., 3329  00       BRK
., 332A  00       BRK
.
```

1TRKFMT

```
.

., 3300  A5 06    LDA $06
., 3302  85 51    STA $51
., 3304  AD 12 03 LDA $0312
., 3307  85 12    STA $12
., 3309  AD 13 03 LDA $0313
., 330C  85 13    STA $13
., 330E  4C C7 FA JMP $FAC7
., 3311  00       BRK
., 3312  EA       NOP
., 3313  EA       NOP
., 3314  00       BRK
., 3315  8D 03 1C STA $1C03
.
```

```
10 POKE53280,11:POKE53281,11:P=4:D=8:HX$="0123456789ABCDEF"
12 OPEN15,8,15,"I0":CLOSE15
15 PRINT"{SC}{YL}        TRACK CHECK{LG}  PSIDAC(C)84 VBN":FORTD=1TO2000:NEXT
20 CH$="NORM":PRINT"{SC}{WH}SELECT CHOICE":PRINT"{CD}(1) NORMAL TRACK CHECK"
30 PRINT"{CD}(2) HALF TRACK CHECK{CD}"
40 INPUTCH$:IFCH$="1"THENCH$="NORM"
50 IFCH$="2"THENCH$="HALF"
100 PRINT"{CD}{WH}INSERT SOURCE DISK  IN DRIVE":PRINT"PRESS F7"
110 GETF7$:IFF7$<>"{F7}"THEN110
115 PR=0:INPUT"{CD}PRINTER Y/N";YN$:IFYN$="Y"THENPR=1
120 INPUT "{CD}DISK NAME";DN$
125 PRINT"{CD}TRACK LOG FOR "DN$
130 IFPR=1THENOPEN4,P:PRINT#4,"     TRACK LOG FOR "DN$:PRINT#4:CLOSE4
140 OPEN15,8,15,"I0"
150 PRINT#15,"M-R"CHR$(18)CHR$(0):GET#15,MH$:MH=ASC(MH$+CHR$(0))
152 X=MH:GOSUB3000:MH$=D$
155 PRINT#15,"M-R"CHR$(19)CHR$(0):GET#15,ML$:ML=ASC(ML$+CHR$(0)):CLOSE15
158 X=ML:GOSUB3000:ML$=D$
160 PRINT"{CD}MASTER ID# = "MH$;ML$" IN HEX{CD}"
162 IFPR=1THENOPEN4,P:PRINT#4,"MASTER ID# = "MH$;ML$" IN HEX":PRINT#4
165 PRINT"{CD}P-TRK D-TRK  IDH IDL  STATUS{CD}"
170 IFPR=1THEN:PRINT#4,"P-TRK   D-TRK   IDH IDL  STATUS":PRINT#4:CLOSE4
172 IFCH$="HALF"THENGOSUB500
175 FORTR=1TO42
182 IFCH$="HALF"THENGOSUB200:GOSUB700
183 IFCH$="NORM"THENGOSUB200:GOSUB295
185 NEXT:IFCH$="HALF"THENGOSUB500
190 CLOSE15:RUN
200 OPEN15,D,15
205 PRINT#15,"M-W"CHR$(24)CHR$(0)CHR$(1)CHR$(0)
206 PRINT#15,"M-W"CHR$(22)CHR$(0)CHR$(1)CHR$(0)
207 PRINT#15,"M-W"CHR$(23)CHR$(0)CHR$(1)CHR$(0)
210 PRINT#15,"M-W"CHR$(6)CHR$(0)CHR$(1)CHR$(TR)
220 PRINT#15,"M-W"CHR$(7)CHR$(0)CHR$(1)CHR$(0)
230 PRINT#15,"M-W"CHR$(0)CHR$(0)CHR$(1)CHR$(176)
240 PRINT#15,"M-R"CHR$(0)CHR$(0)
250 GET#15,A$:X=ASC(A$+CHR$(0)):IFX>127THEN240
254 OP$=" ":IFX=3THENOP$="NO SYNC FOUND"
255 IFX=2THENOP$="NO HDR FOUND"
258 IFX=9THENOP$="HDR CHKSUM ERROR"
260 PRINT#15,"M-R"CHR$(24)CHR$(0):GET#15,CT$:CT=ASC(CT$+CHR$(0))
265 PRINT#15,"M-R"CHR$(22)CHR$(0):GET#15,IH$:IH=ASC(IH$+CHR$(0))
268 X=IH:GOSUB3000:IH$=D$
270 PRINT#15,"M-R"CHR$(23)CHR$(0):GET#15,IL$:IL=ASC(IL$+CHR$(0)) :CLOSE15
272 X=IL:GOSUB3000:IL$=D$
275 IFCT<>0ANDCT<>TRTHENOP$="ILLEGAL TRACK"
280 RETURN
295 PRINTTR;TAB(6);CT;TAB(13);IH$;TAB(17);IL$;TAB(22);OP$
300 IFIL=0ANDIH=0THEN308
305 IFIL<>MLORIH<>MHTHENPRINT"   * * * * * *  BAD ID# * * * * * *{CD}":OK=1
308 IFPR=1THENGOSUB1000
310 RETURN
500 OPEN15,D,15
510 PRINT#15,"M-R"CHR$(0)CHR$(28)
512 GET#15,X$:X=ASC(X$+CHR$(0))
515 BI=XAND3:BI=BI+1:BI=BIAND3:HP=(XAND252)ORBI
520 PRINT#15,"M-W"CHR$(0)CHR$(28)CHR$(1)CHR$(HP):CLOSE15
530 RETURN
700 PRINTTR"{CL}.5";TAB(6);CT;TAB(13);IH$;TAB(17);IL$;TAB(22);OP$
710 IFIL=0ANDIH=0THEN718
715 IFIL<>MLORIH<>MHTHENPRINT"   * * * * * *  BAD ID# * * * * * *{CD}":OK=1
718 IFPR=1THENGOSUB2000
720 RETURN
1000 OPEN4,P
1010 PRINT#4,TR;CHR$(16)"08"CT;CHR$(16)"14"IH$;CHR$(16)"18"IL$;CHR$(16)"24"OP$
1020 IFOK=1THENOK=0:PRINT#4," * * * * *  *  BAD ID# * * * * * *"
1030 CLOSE4:RETURN
2000 OPEN4,P
2010 PRINT#4,TR".5"CHR$(16)"08"CT;CHR$(16)"14"IH$;CHR$(16)"18"IL$;
2012 PRINT#4,CHR$(16)"24"OP$
2020 IFOK=1THENOK=0:PRINT#4," * * * * *  *  BAD ID# * * * * * *"
2030 CLOSE4:RETURN
3000 D$=" ":D1=INT(X/16):D$=D$+MID$(HX$,D1+1,1)
3010 D2=INT(X-16*D1):D$=D$+MID$(HX$,D2+1,1):RETURN
```

ABOUT THE AUTHORS

Vic Numbers and David Thom have been involved in a working association for over seven years. They formed PSIDAC as a partnership to develop and market electronic devices and software.

Vic Numbers has an extensive background in electronics. He has 15 years of experience in custom designed automatic test systems used for fault analysis at a Naval weapons testing facility.

David Thom has been involved with applications of microprocessors for electro-mechanical systems control as well as video games. Mr. Thom also served as Engineering Manager for Windmills International Diversified over a two year period.

Since their association, Numbers and Thom have been involved in numerous industrial and commercial projects. Thom and Numbers headed an engineering team in designing a microprocessor controller for a 150KW wind electrical generator. They have also cooperated in designing computerized advertising displays as well as producing and selling products for the VIC-20 and C-64.

After successfully marketing a handbook and kits for the VIC-20, Numbers and Thom decided to continue with a more serious and in depth study on the C-64. This book is the product of that goal.

THE
SOFTWARE
PROTECTION
HANDBOOK
For the C-64

CONTRARY TO POPULAR BELIEF, WHEN YOU BUY SOFTWARE, YOU OWN MORE THAN JUST THE LIGHT RAYS FROM YOUR TV!

IT IS TIME TO DEFROCK THE WIZARDS BY RELEASING INFORMATION ON THE "BLACK ARTS" OF PROTECTION. YOU NO LONGER NEED SEEK OUT THE CRUMBS OF INFORMATION DOLED OUT BY THE LOCAL WIZARD OR HINTED AT IN THE MAGAZINES.

THIS MANUAL WILL HELP YOU "BLOW THE LOCKS OFF" PROTECTED SOFTWARE! THE SOFTWARE PROTECTION HANDBOOK LETS YOU KNOW WHERE YOU STAND LEGALLY AND PROVIDES INFORMATION ON PROTECTION TECHNIQUES AND BREAKING FOR EACH CATEGORY OF SOFTWARE; DISKS, CARTRIDGES, AND TAPES. THIS MANUAL ALSO GIVES THE LISTINGS AND PROCEDURES FOR ELEVEN PROTECTION ANALYSIS AND BREAKING TOOLS.

THE REFERENCE VALUE ALONE WILL KEEP YOU TURNING BACK TO THIS BOOK FOR AS LONG AS YOU OWN YOUR COMPUTER. YOUR BIGGEST PROBLEM WILL BE IN KEEPING YOUR COMPUTING FRIENDS FROM BORROWING YOUR COPY OF THIS BOOK!!!

A PSIDAC PUBLICATION
7326 N. ATLANTIC
PORTLAND, OREGON
97217