Commodore 64: Quadra 64 – Write 4 Programs At Once!

The Transactor

November 1984

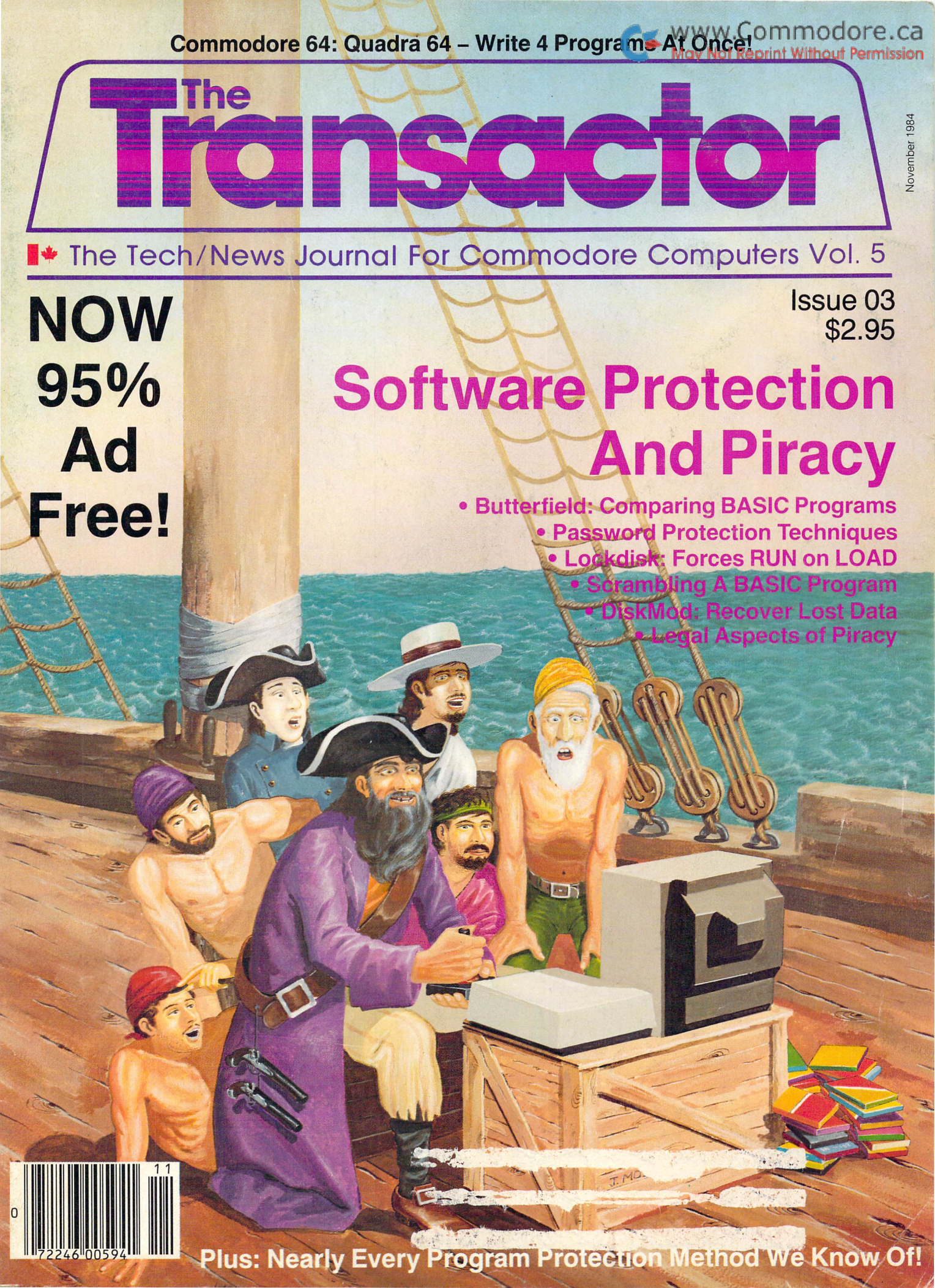
The Tech/News Journal For Commodore Computers Vol. 5

Issue 03
$2.95

NOW 95% Ad Free!

Software Protection And Piracy

- Butterfield: Comparing BASIC Programs
- Password Protection Techniques
- Lockdisk: Forces RUN on LOAD
- Scrambling A BASIC Program
- DiskMod: Recover Lost Data
- Legal Aspects of Piracy



0 72246 00594 11

Plus: Nearly Every Program Protection Method We Know Of!

INTRODUCING

THE PRO-LINE TEAM

PAL 64
The fastest and easiest to use assembler for the Commodore 64.† Pal 64 enables the user to perform assembly language programming using the standard MOS mnemonics. **$69.95**

POWER 64
Is an absolutely indispensible aid to the programmer using Commodore 64 BASIC. Power 64 turbo-charges resident BASIC with dozens of new super useful commands like MERGE, UNDO, TEST and DISK as well as all the old standbys such as RENUM and SEARCH & REPLACE. Includes MorePower 64. **$69.95**

TOOL BOX 64
Is the ultimate programmer's utility package. Includes Pal 64 assembler and Power 64 BASIC soup-up kit all together in one fully integrated and economical package. **$129.95**

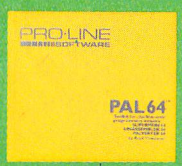
SPELLPRO 64
Is an easy to use spelling checker with a standard dictionary expandable to 25,000 words. SpellPro 64 quickly adapts itself to your personal vocabulary and business jargon allowing you to add and delete words to/from the dictionary, edit documents to correct unrecognized words and output lists of unrecognized words to printer or screen. SpellPro 64 was designed to work with the WordPro Series* and other wordprocessing programs using the WordPro file format. **$69.95**

NOW SHIPPING!!!

For Your Nearest Dealer Call (416) 273-6350

WP64
This brand new offering from the originators of the WordPro Series* brings professional wordprocessing to the Commodore 64 for the first time. Two years under development, WP64 features 100% proportional printing capability as well as 40/80 column display, automatic word wrap, two column printing, alternate paging for headers & footers, four way scrolling, extra text area and a brand new 'OOPS' buffer that magically brings back text deleted in error. All you ever dreamed of in a wordprocessor program, WP64 sets a new high standard for the software industry to meet. **$69.95**

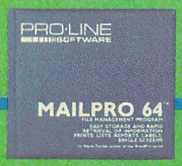
MAILPRO 64
A new generation of data organizer and list manager, MailPro 64 is the easiest of all to learn and use. Handles up to 4,000 records on one disk, prints multiple labels across, does minor text editing ie: setting up invoices. Best of all, MailPro 64 resides entirely within memory so you don't have to constantly juggle disks like you must with other data base managers for the Commodore 64. **$69.95**

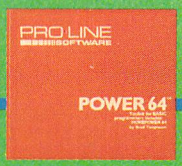


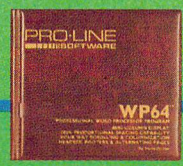
†Commodore 64 and Commodore are trademarks of Commodore Business Machines Inc.

*Presently marketed by Professional Software Inc.

Specifications subject to change without notice...







PRO·LINE SOFTWARE

(416) 273-6350
755 THE QUEENSWAY EAST, UNIT 8,
MISSISSAUGA, ONTARIO, CANADA, L4Y 4C5

Volume 5
Issue 03
Circulation 49,000

The Transactor

The Transactor
The Tech/News Journal For Commodore Computers

Managing Editor
Karl J. H. Hildon

Editor
Richard Evers

Technical Editor
Chris Zamara

Advertising Manager
Kelly M. George
416 826 1662

Art Director
John Mostacci

Subscriptions
Mandy Sedgwick

Contributing Writers
Don Bell
Michael Bertrand
Daniel Bingamon
Jim Butterfield
Gary Cobb
Elizabeth Deal
Domenic Defrancisco
G. Denis
Bob Drake
Mike Forani
Jeff Goebel
Dave Gzik
Phil Honsinger
Garry Kiziak
Scott Maclean
Mike Panning
Howy Parkins
Glen Pearce
Louis F. Sander
George Shirinian
Darren J. Spruyt
Colin Thompson
Mike Todd
Vikash Verma
James Whitewood
Chris Zamara

Production
Attic Typesetting Ltd.

Printing
Printed in Canada by
MacLean Hunter Printing

Program Listings In The Transactor

All programs listed in The Transactor will appear as they would on your screen in Upper/Lower case mode. To clarify two potential character mix-ups, zeroes will appear as '0' and the letter "o" will of course be in lower case. Secondly, the lower case L ('l') has a flat top as opposed to the number 1 which has an angled top.

Many programs will contain reverse video characters that represent cursor movements, colours, or function keys. These will also be shown exactly as they would appear on your screen, but they're listed here for reference. Also remember: CTRL-q within quotes is identical to a Cursor Down, et al.

Occasionally programs will contain lines that show consecutive spaces. Often the number of spaces you insert will not be critical to correct operation of the program. When it is, the required number of spaces will be shown. For example:

print" flush right" – would be shown as – print"[space10]flush right"

Cursor Characters For PET / CBM / VIC / 64

Down – q	Insert – T
Up – Q	Delete – t
Right – [ ]	Clear Scrn – S
Left – [Lft]	Home – s
RVS – r	STOP – c
RVS Off – R	

Colour Characters For VIC / 64

Black – P	Orange – A
White – e	Brown – U
Red – £	Lt. Red – V
Cyan – [Cyn]	Grey 1 – W
Purple – [Pur]	Grey 2 – X
Green – ↑	Lt. Green – Y
Blue – ←	Lt. Blue – Z
Yellow – [Yel]	Grey 3 – [Gr3]

Function Keys For VIC / 64

F1 – E	F5 – G
F2 – I	F6 – K
F3 – F	F7 – H
F4 – J	F8 – L

The Transactor is published bi-monthly by Transactor Publishing Inc., 500 Steeles Avenue, Milton, Ontario, L9T 3P7. Canadian Second Class mail registration number 6342. Second Class postage pending at Buffalo, NY, for U.S. subscribers. U.S. Postmasters: send address changes to The Transactor, 277 Linwood Avenue, Buffalo, NY, 14209, 716-884-0630.

Subscriptions:
Canada $15 Cdn. U.S.A. $15 US. All other $21 US.
Air Mail (Overseas only) $40 US. ($4.15 postage/issue)

Send all subscriptions to: The Transactor, Subscriptions Department, 500 Steeles Avenue, Milton, Ontario, Canada, L9T 3P7, 416 876 4741. From Toronto call 826 1662. Note: Subscriptions are handled at this address ONLY. Subscriptions sent to our Buffalo address (above) will be forwarded to Milton HQ.

Back Issues: $4.50 each. Order all back issues from Milton HQ.
SOLD OUT: The Best of The Transactor Volumes 1, 2 & 3, and Volume 4, Issues 4, 5, & 6 are no longer available.

Quantity Orders:

CompuLit
PO Box 352
Port Coquitlam, BC
V5C 4K6
604 464 1221

U.S.A. Distributor:

Capital Distributing co.

Capital Distributing
Charlton Building
Derby, CT
06418
(203) 735 3381

MICRON DISTRIBUTING

Micron Distributing
409 Queen Street West
Toronto, Ontario, M5V 2A5
(416) 593 9862
Dealer Inquiries ONLY:
1 800 268 9052
Subscription related inquiries
are handled ONLY at Milton HQ

Master Media
261 Wyecroft Road
Oakville, Ontario
L6J 5B4
(416) 842 1555

Editorial contributions are always welcome. Writers are encouraged to prepare material according to themes as shown in Editorial Schedule (see list near the end of this issue). Remuneration is $40 per printed page. Preferred media is 1541, 2031, 4040, 8050, or 8250 diskettes with WordPro, WordCraft, Superscript, or SEQ text files. Program listings over 20 lines should be provided on disk or tape. Manuscripts should be typewritten, double spaced, with special characters or formats clearly marked. Photos of authors or equipment, and illustrations will be included with articles depending on quality. Diskettes, tapes and/or photos will be returned on request.

From The Editor's Desk

Piracy: A Fact Of Life?

Piracy. It exists in just about any industry you can think of, in one form or another. In some circles it's far more rampant than we realize, mainly because it's been there for so long that we no longer notice.

Did you know airline employees and there immediate families can fly anywhere in the world at little or no charge? Same with most railways and bus companies. Add all these up and you wonder just how much of your "full-fare" ticket goes to subsidizing free rides. Nobody is stealing because nobody really loses anything. The president collects the same salary. And the average rider gladly pays the price because it's cheaper or faster than going by car. Fundamentally, it's the privilege of getting something for nothing for those closest to the operation.

Other industries too. Brewery employees get free beer. Park employees get free admission. Hotel workers stay rent free at any hotel in the chain. Bank employees get loans or mortgages much easier at less interest than most. A member of a policemans family rarely pays for a parking ticket or most other misdemeanors. How 'bout politicians? Talk about "easy street"; they don't even pay income tax!

The list goes on and on. But where does software fit in. Just who is getting programs for nothing that others pay for. Manufacturer employees will get the same privileges as those closest to any other industry. It's only natural. And the vendors for these items make special concessions too; they call it "on consignment" or "on loan for testing purposes". But most authors don't complain about these, even though every member of every family associated with the software business will get at least one free ride.

The real problem, in the eyes of the writers, lies beyond the immediate industry individuals. A package goes on the market, sells for a while, and eventually lands in the hands of a hacker; the computer hobbyist who sees protection against copying as challenging their abilities. More often than not they win. The program becomes a "collectors" item, not because it's hard to get but because they won't have to pay for it.

So how many "free rides" does the collector represent? Even if every member of the Toronto PET Users Group were to get a copy we're only up to 20,000. Commodore sold over 2,000,000 VICs! And C64 sales will easily pass that by the end of 1984, if they haven't already! Hobbyists represent only 10% of the market. The other 90% will never come in contact with the unprotected copy of software they need. Sure, there will be some, but not all 100% of the hobbyists will bag your programs either.

Piracy hit the games market worst, especially among students. As soon as one student got a copy, the whole school wasn't far behind. But the games market has almost totally crashed. You can't blame that on piracy. Fads will always come and go, and computer games were no exception.

Business software is where the money lies. If your idea has a market, pursue it. Those few copies that reach the collector "au gratis" just aren't worth complaining about. Spend your time finding new sales and quit wasting time battling pirates that wouldn't have bought your program anyways. Don't waste money either. Advertising business packages in technical computer magazines is like advertising frozen food to a connoisseur. Some will buy it but most will make their own. Market your programs in their market, not in their venue.

Software is no longer a "get rich quick" business. It used to be but not any more. A good programmer spends no more time on a new package than a good novelist spends on a fresh story, some books take years to complete. So if you price your software beyond its value, naturally it will become a candidate for a pirates thrill.

Publishers are one answer. Many book publishers are now adding software to their line. But just like books, some software isn't worth publishing. If you have a finely polished product, consider presenting it to a publisher. Most will guarantee a minimum royalty that is usually more than fair.

Don't let greed cloud your expectations. I admit, writing software requires talent and skill, but no more than some other occupations that pay considerably less. In this business, Return on Investment is a time versus profit ratio. Break it down to the hour and compare.

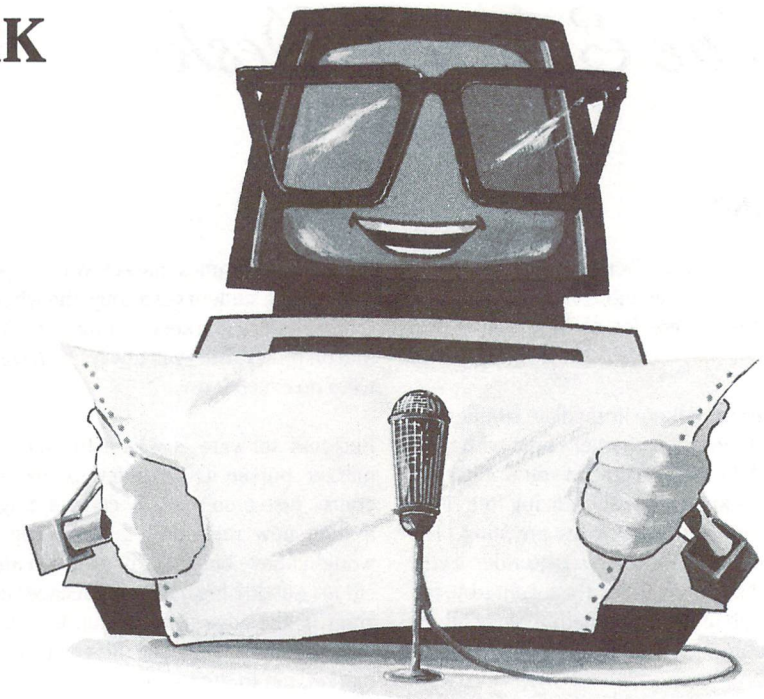
I don't condone piracy. Never have, never will. But there's no point wasting energy that could be put to more productive results.

There's nothing as constant as change. . . until next issue, I remain,

Karl J.H. Hildon
Managing Editor, The Transactor

Post Script: I would have announced Chris Zamara as our new Technical Editor, but there wasn't enough space.

News BRK



Transactor News

95% Ad Free!

You may have noticed our somewhat less than subtle message on the cover – perhaps an explanation is in order. Back when The Transactor began publishing, we anticipated an initial circulation of about 5,000. Our advertising rates were designed accordingly. When we hit 15,000 print run, the rates were doubled but are costs were up by 3 times. Now our print run is almost 50,000 and our costs have outgrown what we can feasibly charge for ad space. Although we appreciate the support our advertisers have extended us, to save money on our printing bill we've decided to reduce our available ad space to the cover and 1 or 2 pages inside the magazine.

Each person we've told of this policy asks, "How will you survive without that revenue". Well, to be quite honest, several of our advertisers have outstanding bills going back 6 months. Others have gone out of business, which makes it tough to collect anything. Our counter is, however, that without advertising we have a more desirable product which means we stand to sell more copies.

Some say that advertising is partly why they buy magazines, to see what's available out there. Most of what we were advertising is mentioned at least once here in News BRK, except for the stuff we feel isn't worthy of space. Besides, we didn't have *that* many advertisers – for a broader picture you might consider COMPUTE or BYTE or RUN. They have 50% ads or better, and they'll advertise just about anything. . . which leads to the next point.

We believe our readers are advancing to a stage of computing beyond playing games and/or buying packaged software that bears little resemblance to the package itself. We think most of our readers can write software that outclasses 90% of the stuff for sale. So anything less than fabulous will not appeal to Transactor readers.

Eventually we may re-introduce advertising, but at our discretion. Ads for products that don't meet the standards of our readers will not be accepted. In the meantime, we will print a couple of ads that contain information we feel is important enough to merit space. . . and we'll probably not charge for these. If this policy changes, our criteria will be so tough to meet that 5 pages should be more than enough to accommodate every ad.

Further, a smart advertiser should not contract for space in any magazine without seeing an audited circulation report. This information can be obtained through ABC, the Audit Bureau of Circulations. Presently we are not listed with ABC, but if we do start accepting ads again, we will be.

And finally, the magazine industry is not unlike any other. Competition is fierce. But entering the Indy 500 with a Rolls Royce is pointless. Instead we intend to open a league where polish, finesse, and attention to detail are more important. Does anyone care to join us?

Cover Date

Did anyone notice the date on the cover? Don't be alarmed – the Issue number will tell you whether you've missed any copies or not. The date is there only because second class mail regulations require it. The reason it's so far in advance is to give The Transactor a better shelf life at the newstand. Subscribers should get this issue early in August. But, in some places, this issue won't be on the newstand until the second week of September. No kidding. Since The Transactor is bi–monthly, a cover date less than November would mean retailers begin returning this issue before the end of October. Ideally, no cover date would give us better exposure time. But since that's not an option. . .

Expiry and Subscription Number

From now on, your subscription number and the issue with which your subscription expires will appear at the top of your mailing label. The most convenient way to renew your subscription is with the postage paid reply cards at the center of each issue. To speed the process at our end, please indicate that you are indeed a renewing subscriber and include your subscription number. Then add your name, address (don't forget your postal/zip code) and a charge card number (& exp. date) and drop it in the mail. If you can't use the card in your copy, perhaps you know someone who can.

If there is any disagreement with the Expiry Issue on your most recent label, please let us know ASAP so we can clear things up.

Micron Distributing

Access Computer Services is no longer distributing The Transactor. Retailers with standing orders from Access should contact Micron or us at Transactor HQ (see page 2). Micron specializes in computer magazines servicing most of the titles currently available. They also carry a huge selection of computer books, software, hardware, and accessories. In fact, about the only thing Micron doesn't have, you don't want anyways. Give them a call.

Reference Transactor Update

The Complete Commodore Encyclopedia, another name we're toying with, is well on its way. We'll probably decide which name we'll use at the last minute (or maybe a third?). Order information is still not available so don't send any money just yet. Although it won't be included as part of a subscription, a special price to subscribers is under consideration. Regular price will be anywhere from 12 to 25 dollars depending on final size (in pages) and whether we include the utilities diskette or not.

Commodore News

Jack Tramiel Buys Atari

Jack Tramiel, founder and former vice chairman of Commodore International bought Atari from Warner Communications on a promissory note for a reported 240 million dollars. Tramiel left Commodore last January for reasons ranging from lack of self confidence to major differences with other Board members. It would seem the former is rather unlikely in the light of this report.

Atari has been crippled by losses over the last 2 years, but with Jack behind the wheel, you should see new life before long. Plans were announced to inject over 70 million towards rebuilding the companies' foundation.

Commodore may well be feeling they shouldn't have let Jack slip away. Already 4 of Commodores' development staff have defected to Atari, no doubt due to Tramiel dynamics. Commodore has already filed suit against Tramiel and Atari for stealing trade secrets, a battle that could probably get great TV ratings. Ramifications may severely hinder new Atari plans. Temporary injunctions were already issued at the time of this writing. If permanent injunctions are brought down pending trial, and Atari wins, Commodore may find themselves staring down the business end of a double barreled countersuit for lost production time. This could be the computer industry legal mess of the century.

Commodore Supports Canadian Organizations With Promotion Proceeds

TORONTO — Commodore Business Machines Limited recently presented the Canadian Amateur Hockey Association and the Canadian Association for the Mentally Retarded with funds raised through a Commodore–sponsored promotion held in conjunction with the Canadian Motion Picture Distributors Association.

Over $35,000 was raised by the sale of The Commodore 1984 Movie Poster Calendar and was divided between the two organizations. Richard G. McIntyre, Commodore's National Sales Manager, presented cheques to Jim Gates, CAHA Director and Barry Wymant, nine of the Scitron–sponsored Commodore team and to Jacques Pelletier, CAMR Acting Executive Vice President and Jeff Oswin of the CAMR Toronto staff.

Commodore also supports the CAHA through its national Custom Team Uniform program which annually supplies sweaters and stockings to 500 CAHA teams across Canada. For more information, contact:

Mr. Richard Browne
Commodore Business Machines Limited
3370 Pharmacy Avenue
Agincourt, ON M1W 2K4
416 499–4292

New ROMs For Commodore 64, 1541 Disk

Commodore has released new upgrade ROMs for The 64 and 1541. Reasons for the new 64 ROM are unclear at this time, but reports have come in that it has problems reading cassette tape. This is not definite, so if your machine goes in for service, check this out as soon as you get it back. As you know, Commodore, like other electronics firms, simply replace the entire PC board for a better turn–around time. Your board is fixed and one day it too becomes a replacement.

The 1541 ROM was designed to make all 4040 and 1541 diskettes read AND write compatible. 1541 and 4040 were always read compatible, but interchangeable writing was ill advised. The only problem here is that now old 1541s won't be compatible with new 1541s. Chances are they've already been recalled. More next issue.

By the way, remember the problem with SAVE and Replace? Back in the early days of Commodore disk drives, SAVE with Replace was blamed for some of the most mysterious diskette failures. So far nobody has been able to deliberately recreate the problem. In fact, there has been no evidence to

suggest there ever was a problem. DOS source code has been checked thoroughly, but no changes have ever been made in this area. If you can supply a program or procedure to demonstrate how SAVE with Replace can ruin a disk, there are already 2 rewards up for grabs; a case of beer from Harry Broomhall in England (one of the leading authorities on disk drives) and a bottle of champaign from Transactor Publishing Inc. Good Luck!

Commodore Introduces The 8296

The new 8296 has 128K RAM (96K plus 32K or use of the User Port, but not both simultaneously), 18K ROM, detachable keyboard, tilt/swivel 80x25 screen, and an 8050 disk unit (1.05 MBytes) that may or may not be built-in. The computer is being packaged in B Series casings. (why not, that was about the only good thing going for that machine)

The package comes with software: PaperClip (with over 900 lines for text), CalcResult, and The Consultant database. Communications and system utilities are included too, but otherwise it will be virtually 100% compatible with any software for the 8032/96. No word on price. Contact your dealer or:

Mr. Richard McIntyre
Commodore Business Machines Limited
3370 Pharmacy Avenue
Agincourt, ON M1W 2K4
416 499-4292

Commodore 264 Now Called The +4, 16K Version Also Released

Remember the TED? Changed to the 264? Now it's called the +4. Same machine in all cases - same old Commodore too. Seems product name changes at Commodore would be an entire department by now.

Although Commodore has lots of VIC 20s warehoused, production of the classic colour pioneer has all but come to an end making way for the Commodore 16. The 16 is basically a 16K version of the +4 (64K) packed in a black VIC/64 shaped casing. Dubbed, "The Learning Machine", it's being aimed at the uninitiated computer user.

Like the +4, most of the I/O ports have been altered to different connectors making old peripherals incompatible. However, old tapes and diskettes should be readable from the new peripherals.

The best part is the price: $99 US for the machine itself, plus peripherals. Not bad considering all the features. 16 colours with brilliance and flashing attributes, extended BASIC, extended Monitor commands, graphics commands, editing commands, and the sound is still pretty good even without the SID chip.

Compared to earlier technology, the 16 and +4 will make learning much more enjoyable as awareness for odd system configurations will be virtually eliminated. For more, contact your dealer or Commodore. Available Fall '84.

General News

The Canadian Computer Museum Institute

The new Computer Museum will provide Toronto with an enhanced international high-tech profile, important for the development of an advanced-technology industrial base. The museum will be a place where people may turn to gain some computer literacy, and thus, an important educational resource for the region. The museum will add to Toronto's tourism industry. The museum will be a significant architectural development for Toronto.

Impact on the computer industry. The museum will provide a showcase for Canadian accomplishments in computers and a place where industry can highlight the evolution of ideas and advances which have led society into the computer age. The museum will cover the use and impact of computers in many facets of business, the arts and society. For more information, contact:

The Canadian Computer Museum Institute
212 King Street W. Suite 400
Toronto, ON M5H 1K5
416 593-5777

Copylock Protection For Software Duplication

An improved version of COPYLOCK (TM) protection technology is now available from Magtech. This new protection has totally defeated the two most formidable protection-breaking programs available today, COPY II PC and COPYWRITE.

Using the standard COPYLOCK protection, each software program is duplicated using a Metered Program Disc. These discs allow a finite number of duplicated copies to be produced during a duplication run. The standard COPYLOCK protection is currently available for the IBM PC. Dos 1.1 and 2.0 operating systems, although more versions are expected in the future.

Custom software protection is currently available for certain Commodore, Apple, IBM, and Atari systems. This type of protection can not be broken by protection-breaking programs and offers a unique protection method for software.

The duplication is done under clean conditions and each production run is heavily quality controlled for trouble free software performance.

Magtech is one of Canada's largest software duplication houses and offers bulk duplicating as well as packaging services. For further information please call or write:

Magtech Inc.
87 Telson Road
Markham, ON L3R 1E4
416 474-0170

MicroEd Home Library Donations Pass Half Million Dollar Mark

During the past three months, MicroEd, Incorporated, a Minneapolis-based publisher of educational software, has donated more than a half million dollars worth of its instructional programs to school systems interested in establishing free software lending libraries for the families they serve, according to MicroEd President Thorwald Esbensen.

"A persistent problem for educators and parent," observes Esbensen, "has been the development of effective communication channels between home and school. Confronted now with the growth of the home computer market, it behooves boards of education and their administrators to respond vigorously to the challenge of helping families make informed decisions with respect to the proper use of educational software that can effectively supplement the academic goals of their local school systems."

To encourage the development of an orderly and comprehensive plan for dealing with this problem, MicroEd has been donating up to ten thousand dollars worth of its educational programs to any elementary school library system that can meet designated standards for strengthening home/school cooperation in the area of computer-assisted instruction.

To date, more than fifty school systems across the county have qualified for a MicroEd grant, with further approvals pending.

No cutoff time has yet been established for grant proposals to be submitted to MicroEd. "We hope to be able to do this on a continuing basis," sys Esbensen. "We think it's an idea that makes it possible for everybody to be a winner in a worthy cause. The only limitation will be the extent to which we eventually find ourselves hard pressed to handle the production load. We'll just have to see how things work out in that regard."

Inquiries may be addressed to:

MicroEd Home Library Grant Project
PO Box 444005
Eden Prairie, MN 55344
612 944-8750

Events

Toronto International Software Show Offers Great Opportunity For "Computerphiles"

Toronto. . . The first Toronto International Software Show has been announced by the producers of Computer Fair.

Hunter Nichols Inc. recently impressed the industry with its enormous success at Computer Fair '84. The Toronto International Software Show is a natural progression and an answer to the problem of keeping up to date with the latest software products as they become available in the market.

In addition to hundreds of exhibits that will appeal to all computer users for home and business, there will be a seminar program running in conjunction with the Show.

In keeping with the high standard of previous Hunter Nichols shows and seminars, the Toronto International Software Show program is expected to give excellent insight into growing areas such as hardware compatibility, user interfaces and future trends.

Toronto International Software Show, International Centre, 6900 Airport Road (at Derry Rd.), Mississauga, Ontario.

Show Hours and Dates: Starts Thursday, September 20, 1984, until Sunday. Thursday and Friday, 10 am to 9 pm, Saturday and Sunday, 10 to 6.

Admission: Adults $6.00: Seniors/Students $5.00. Special rates for groups available from Show Management in advance. For more information:

Hunter Nichols Inc.
721 Progress Avenue
Scarborough, ON M1H 2W7
416 439-4140

World Of Commodore II

The second annual World Of Commodore Show is scheduled for November 29 thru December 2, 1984, at Toronto's International Centre. Unlike last year, this year's show will not be shared by the Home Entertainment Show.

Based on the success of last year's show, this one should be one of the world's best. So far, Britain's famed "Pet Show" has held top honours since it began. But only because it had no competition. Not any more.

Floor space is $11.50/Sq Ft. For more inforamtion, contact Hunter Nichols above.

Books And Magazines

INFO 64

A new Commodore information magazine has emerged for users of the 64. INFO 64 is published quarterly out of Auburn, WA. Almost all of the magazine is printed with a dot matrix printer, but use of reverse type, border lines, shading for 3D effects, and hi-res printer dumps give it a rather smart appearance. For more, contact:

INFO 64
PO Box 958
Auburn, WA 98071
206 833-6502

Commander Magazine Stops Publishing

Commander Magazine, a Commodore information monthly out of Tacoma, WA, has ceased publication. The company that owned Commander was recently acquired by Zif Davis,

the publisher responsible for Creative Computing among others. Zif Davis is substituting copies of Creative to fulfill subscription obligations at Commander.

Computer Book Centres Rack Up Profits

There's no denying it. Personal computing has taken the nation by storm! And it's a storm that's not about to abate. For you, the retailer, this means a thunderous market. Typical computer owners invest as much in computer-related books and book-software as in hardware.

Copp Clark Pitman Computer Book Centres come in all shapes and sizes. We have one just right for your retail business. Our spinner rack can hold up to 100 books — 20 different titles on four individually moving tiers. The books are easily accessible to your customers in a minimum amount of valuable floor space.

We can start you with a selection of titles from Howard W. Sams, Pitman, and Wadsworth Electronic Publishing tailored to your customers needs and guaranteed to keep them coming back for more.

Our Pitman Programming Pocket Guides have already sold over 250,000 copies internationally. The displayer is yours free with your purchase of a preselected group of Pitman Pocket Guides. For more information contact Copp Clark Pitman or your local book and software wholesaler.

Copp Clark Pitman Ltd.
495 Wellington St. West
Toronto, ON M5V 1E9
416 593-9911

The Commodore Diary 1985, by Jim Butterfield

A computer reference date book, from Copp Clark Pitman. This handy pocket-sized diary features a whole week per page spread in a clear, uncluttered format.

A convenient listing of all major computer shows across North America and Europe appears in the front matter of the diary.

Extensive reference material for the C64, VIC20, Pet/CBM Series, B-Series, the Commodore 264 and 364, written by Jim Butterfield is featured-memory layouts, screen codes, useful short programs, machine language instructional set, Kernal subroutines, and much more!

Special Feature: The Commodore Diary cover can easily be customized to suit your business and/or promotional needs.

For further information please feel free to contact: Gus Creces at Copp Clark Pitman Ltd. (see above)

Commodore Magazine Index

Altacom, Inc. is introducing PcDex and PcDex Quarterly, microcomputer magazine resource guides to Commodore 64, VIC-20, and PET/CBM. The only exclusively Commodore magazine index, PcDex provides fast, easy access to the often overwhelming amount of microcomputer magazine literature. Designed as six separate indexes—subject, title, program listings, software reviews, hardware reviews, and tables of contents—PcDex allows the serious home, business, or educational user to quickly locate specific items of interest, including articles, columns, letters, programs, and reviews. Special features include cross-referencing, program descriptions, updates and revisions, specific machine requirements, and suggestions for locating back issues.

PcDex indexes the 12 most popular Commodore and related general microcomputer magazines published between January 1982 and April 1984, with yearly updates planned to include the current three years. PcDex is intended both as a reference companion to a user's own magazine collection and as a reference to a broader base of magazine literature.

PcDex Quarterly follows the same format, but will be published four times a year with an annual compilation and will include any relevant new publications which may appear. PcDex Quarterly is available through subscription only for those who want to be up-to-date on current Commodore related publications.

PcDex is available at bookstores or directly from Altacom for $14.95; PcDex Quarterly is $17.95 for a one year subscription. Direct inquiries to:

Altacom, Inc.,
P.O. Box 19070
Alexandrea, VA 22314
703 683 1442.

Software News

MAILBOX 64

MAILBOX 64 – A revolutionary new Amateur Radio Teletype "Bulletin Board System" operating at 110 Baud ASCII for the Commodore 64 computer. MAILBOX 64 incorporates many of the features found in much more expensive systems and in some areas exceeds those systems. The program is written in BASIC and can be readily tailored by the average user. MAILBOX is an ideal choice for the budget conscious individual or club.

Provides 20 different user commands including:

OPEN	– opens the buffer to store text.
CLOSE	– closes the message buffer.
SAVE	– save the message buffer to the Sysop disk.
MSG	– transmits the contents of the message buffer.
PRINT	– prints the contents of the message buffer to the Sysop printer.
INFO	– transmits the SYSTEM file.
BEACON	– transmits the CQ beacon call.
LOG	– logs the user's call to the System disk.
GRAPHICS	– activates the graphics mode.

In addition to the user commands, there are 31 Sysop commands allowing for complete control and versatility for

the System Operator.

MAILBOX 64 includes a low-resolution Graphics Mode which allows the transmitting of Commodore graphics and colour to users with a Commodore 64 computer.

MAILBOX 64 interfaces via the User I/O Port of the Commodore 64 and is compatible with most popular Terminal Unit (TU) interfaces including the Kantronics and AEA. Requires the Commodore 64 computer, Disk Drive, a Printer at device #4, TU and your radio transceiver.

Available from RAK Electronics on disk for $49.95 plus $2.00 shipping and handling. Catalog order number is RM837.

RAK Electronics
Microcomputer Software
P.O. Box 1585
Orange Park, FL 32067
904 264-6777

Graphics Terminal Emulator For The 64

The standard method for transmitting data to graphics terminals is extremely clever and simple, but a Basic program for this on the Commodore 64 is too slow to keep up with a 300 baud rate. This is true even if the program uses machine language graphics subroutines. Because it is written totally in machine language, GRAPH-TERM 64 not only can display high-resolution graphs as they are transmitted but can download the plot files and replay them up to 20 times faster. It can also produce hard copies of the plots on the Commodore 1520 plotter.

GRAPH-TERM 64 is a terminal program which prints text and high resolution plots generated by a mainframe computer. It is thus of particular interest to scientists and engineers who use standard graphics programs which generate plot files in Tektronix format. While displaying the incoming data, the program also stores it in memory for subsequent transfer to disk or tape or to the Commodore 1520 Plotter. During a terminal session or afterwards, the information can be reviewed at high speed, slow motion or stop action. The high resolution screen of the Commodore 64 is limited to 320 x 200 pixels but on the plotter the resolution is 630 x 480. This is not much less than that of Tektronix 4010, 1024 x 780.

The most important program is SAVE/LOAD which stores and retrieves data downloaded into memory. For those who wish to generate their own plots, the program TEK-ENCODER shows how to encode plots in Tektronix format. Program TEKPLOTTER is a Basic program which produces hard copies of downloaded plots on the Commodore 1520 Plotter in the same way as the machine language menu option of GRAPH-TERM 64. It is included for those who are curious about how the programs work or who may want to extend them.

The subroutines include the usual primitive graphics routines for drawing lines and setting pixels and for shifting to high resolution mode and back. In addition there are routines for drawing simple hidden surface 3-D figures and general "ellipses" of arbitrary shape, orientation, position and number of sides. A novel aspect of these subroutines is that they correct for the fact that the pixels on the Commodore 64 are not square. The screen is assumed to be 780 x 1024 pixels, the same as a Tektronix 4010 graphics terminal. The program "GEOMETRY FUN" illustrates the use of these routines.

In summary, the machine language program and the Basic programs which use its subroutines allow you to:

1. View Tektronix format plots generated by a mainframe computer.
2. Download text or plot files.
3. Generate plot files on the Commodore 64.
4. Preview plots on the high resolution screen, then:
5. Create hard copies of the plots on the Commodore 1520 Plotter.

The price is $49.95 + $4.00 shipping and handling (U.S. funds). Foreign orders other than Canada add 20%. Visa and Master Card accepted. Please include expiration date and correct number. Dealer inquiries invited.

Bennett Software Company
3465 Yellowstone
Ann Arbor, MI 48105
313 665-4156

PRINT-MASTER For The Commodore 64

No programmer should be without PRINT-MASTER, the ultimate printer enhancement cartridge for the Commodore 64. It unites your computer and your Epson compatible printer as if they were made for each other. PRINT-MASTER can do hi-res mode or character mode screen dumps to the printer with a single keystroke. It allows printing of exact replicas of the full Commodore 64 character set, including graphics and control characters, at full printer speed. It can also printer user defined character sets.

PRINT-MASTER adds twelve BASIC commands that allow easy selection of most printer features such as emphasized, subscript, lines per inch, and skip over perf, from the keyboard or from a BASIC program. The BASIC OPEN command is enhanced to provide 12 new printing options. In addition, PRINT-MASTER provides a complete set of disk support commands (similar to DOS 5.1), to load and save programs, display the directory, send commands to the disk, and read the error channel.

For BASIC listings, PRINT-MASTER can expand control characters into mnemonics such as (HOME) and (BLU). It can format listings by printing each BASIC statement on a separate line, indenting FOR loops, and assuring that BASIC keywords and mnemonics are not broken at the end of a line. Full left and right margin control is included. With the special UNNEW command, you can recover a BASIC program in memory after a NEW command or system reset.

PRINT-MASTER works only with Epson compatible printers connected to the Commodore 64 by a serial-bus-to-arallel

printer interface. A version is also available with a built-in output port and cable for direct connection to the parallel port of the printer. PRINT-MASTER is a versatile tool that provides easy user control of the power of these dot matrix printers.

Price: $39.95. For more information:

IPS
10570 SW Walker Rd.
Beaverton, OR 97005

Antenna Reducer

A unique Amateur Radio antenna design program for the Commodore 64 computer. Allows the user to design a reduced size antenna in the frequency range of .5 to 15 MHZ. After inputting the desired frequency, the user can select an antenna design of 30, 40, 50, 60, 70, 80, 90 or 100% of full size. The user then selects the loading coil diameter of 1.5, 2, 2.5 or 3 inches using 4, 6, 8 or 10 turns per inch. The program then calculates the antenna measurements and graphically displays the antenna design.

Requires the Commodore 64 computer with Tape Datasette or Disk Drive. Available on Tape for $7.95 or Disk for $10.95 plus $2.00 shipping and handling. Catalog order number is WC836.

RAK Electronics
PO Box 1585
Orange Park, FL 32067
904 264-6777

EnTech Offers Software Demo Disks

Too often people spend forty or fifty on software only to be disappointed. To help take the guesswork out of buying software, EnTech has created the "Knock Your Socks Off For 5 Bucks" promotion.

In this deal, EnTech will send you a demo disk of any of the programs for 5 dollars. They come with a coupon for 5 dollars off the actual program. So even if you don't like the demo, you still keep the disk which might have cost you 5 dollars anyways.

The demos are available from EnTech dealers or directly from EnTech Software.

EnTech Revolutionizes Talking Software

EnTech Software of Studio City, California has introduced software that talks in a real human voice. For the first time, the Commodore 64 will be able to reproduce the intonations, the accents, and the character of real speech. EnTech's development is certain to revolutionize the software industry.

EnTech will be using this new speech process to enhance all of its current software programs. Talking versions of its popular "Space Math 64" educational game, music program "Studio 64", and business program "Management System 64" will be introduced at the Summer Consumer Electronics Show in Chicago, booth 6904. EnTech will also be producing a new line of talking educational programs.

According to EnTech's Chairman of the Board, Ray Soular, "Our innovation makes the computer more human. By talking in a human voice, the home computer will be able to teach foreign languages, help with spelling, tell stories, and do many things it couldn't do before."

To educate dealers about this new generation of talking software, EnTech is distributing a talking demonstration disk. Interested dealers can contact:

EnTech Software
P.O. Box 185
Sun Valley, CA 91353
818 768-6646

EnTech Introduces Data Protection Plan

EnTech Software has announced its new Data Protection Plan, the first comprehensive system for the protection and repair of customers' valuable data.

EnTech's "Management System 64" business program for the Commodore 64 now includes three new disk maintenance features as well as a data service warranty. The first is a disk backup program that repairs damaged sectors as it copies the data disk. If a single record is damaged, a second data repair program examines each individual record and corrects it. If the disk is totally damaged, a third feature completely reconstructs the disk, by first re-formating it and creating new files. Then it examines the damaged data and transfers it to the new files record by record. It will also inform the user of any incorrect records it finds and allow them to be repaired.

In addition to these new program features, EnTech has added a data service warranty. For a small fee, EnTech will repair any damaged data disk, or the disk will be returned with the money. This service is for all users who have sent in their EnTech warranty card.

People are already benefitting from this new warranty plan. Tom Lindgren of Kapri International, a company that uses "Management System 64", said, "These new features have already gotten us out a tough scrape. We used them to repair a disk damaged by a power failure."

EnTech plans to extend its data protection policy to all of its programs, including "Data Base 64", "Finance Calc 64", "Studio 64", "Recipe Keeper", and "Checkbook System". EnTech president Rick Bates said, "A business computer system is worthless if your data is accidentally destroyed. A complete business program should allow people to make mistakes."

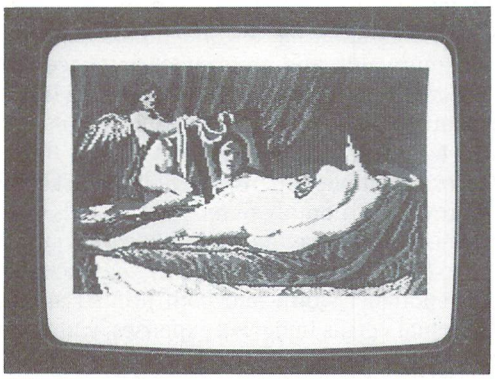
Computer Aided Design for the C-64

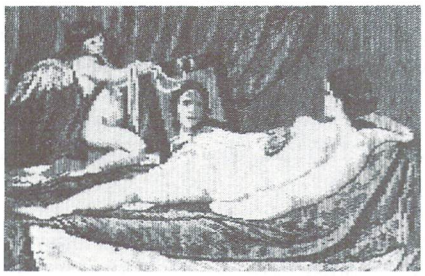
Kiwisoft Programs has created CADPIC for the C-64, combining two programs PAINTPIC and PRINTAPIC to give a

total package for true picture design. Cartoons, tapestry, hooked rugs, furniture design, game backgrounds, coloring books, advertisements, and original paintings are some of the uses for CADPIC.

PAINTPIC has full 16 color painting on a 160 x 200 dot screen. Over 65,000 Multicolor brushes and complex wallpaper patterns are supported. Many automatic tilted shapes may be drawn and filled. Block operations include save, copy, double, halve, 90o rotate, and mirror. Drawing and painting are done at the keyboard or with joystick.

PRINTAPIC provides a true proportion, five gray-scale, dot-graphic printout of PAINTPIC pictures on most printers. Character printers are also supported. Additional features include black and white plotter style print, color separation pictures, MICRONEYE photo conversion, "paint by numbers" outline prints, and hooked rug and tapestry design. PRINTAPIC includes the full color "Venus" by Velazquez.





No special hardware is required, although an MPS-801 graphics printer and the MICRONEYE are recommended. CADPIC sells on diskette for $79.95. PAINTPIC comes separately on disk for $39.95; PRINTAPIC for $44.95. The MICRONEYE camera is from Micron Technology, Boise, Idaho, for $295. Contact:

Kiwisoft Programs,
18003-L Skypark South,
Irvine, CA 92714
714 261-5114

New Weather Sensing Package

Designed for use with Commodore 64 and Vic 20 Computers, the new HAWS 8 Home Automatic Weather Station) from Vaisala combines a professional quality weather sensor with a creative software package that teaches, forecasts, and graphically displays weather. More than a toy or game, HAWS utilizes the same weather sensor used by weather services in 60 countries worldwide. In addition, HAWS represents the first personal computer application utilizing an external sensing device, allowing the user to interact and analyze input which is not contained in his computer or the software itself.

HAWS allows the user to monitor weather conditions inside or outside the home, as well as allowing the user to interact with the software program to help predict and cope with changing weather conditions. HAWS even allows the user to rate his/her forecasting performance against the local weatherman's predictions.

HAWS is an excellent educational tool for teaching meteorology concepts and for learning about weather, either in the home or in the classroom. In addition, HAWS can also be used to monitor and control indoor living space, greenhouses, and office environments, etc.

The HAWS package is riced at $199.95 and includes sensor, choice of cassette tape or floppy disk program, 15-foot cable with connector for the computer, and complete user manual. For more information including dealer inquiries, write or call:

Vaisala Inc.
2 Tower Office Park
Woburn, MA 01801
617 933-4500

Educational Administrative System

Distributed exclusively by Aurora Software Inc., the EAS - Educational Administrative System is a flexible administrative tool developed by experienced educators for use in schools. The three modules that make up the system - Data Base, Attendance and Timetabling - are currently available, with additional modules, including Marks Reporting and Report Card, to follow. Easy-to-use documentation accompanies the package in a sturdy manual format. Ongoing

updates for the manual will be provided as they become available.

The EAS has been successfully tested in schools of up to 2,500 students. In addition, plans for further testing the system are underway in Northern Ontario.

The Educational Administrative System utilizes the CBM 8032 (either the 8050 or 8250 drive format), with a high speed dot matrix printer. Is is anticipated that this system will also become available for other micros, including the government's Ontario Approved Microcomputer (ICON).

The Data Base, Attendance and Timetabling modules for the Educational Administrative System are now available at $250.00 each. All orders must be pre-paid. Ontario residents add 7% P.S.T. Contact:

Aurora Software Inc.,
PO Box 1394,
Haileybury, ON P0J 1K0
705 672-5517

NET WORTH

NET WORTH is a fast, powerful, versatile and easy to use program to manage, track and organize every aspect of a family's financial affairs. With NET WORTH, the home computerist can:

- Set up a budget – with as many as 350 categories – then compare actual income and expenses to that budget.
- Keep a record of every banking and credit card transaction and reconcile statements instantly. NET WORTH can handle up to ten bank accounts, and it prints checks too.
- Maintain an up-to-date record of personal "net worth" – a balance sheet of assets and liabilities.
- Record every tax deductible expenditure, for instant recall at tax time.
- Make money work harder by analyzing interest rates on savings plans and loans.
- Document household valuables, collectibles and important papers for insurance and other purposes, and record their locations as well.
- Display or print financial reports.
- Record stocks, bonds and other investment transactions.

NET WORTH is an easy way to make sense of the family's finances. Documentation is written in clear, simple language, without technical accounting terms. Special help functions are available on-screen at all times.

NET WORTH is a value-filled addition to the home software library. To dramatize its relevance, a Susan B. Anthony silver dollar and an excerpted edition of Sylvia Porter's "New Money Book For The 80s" are included in each package.

NET WORTH is available for the Commodore 64 at $79.95. Contact:

Scarborough Systems, Inc.
25 North Broadway
Tarrytown, NY 10591
212 986-1556

TOTL.MONEYMINDER For The C64

TOTL Software Inc., manufacturers of practical small business and home productivity software for the Commodore 64, have announced the scheduled release of their eighth software product for the Commodore 64. TOTL.MONEYMINDER is a complete home accounting system designed to complement other software already available from TOTL. It will be shipped to distributors and dealers on May 15, 1984, and will carry a suggested retail price of $39.95, consistent with the other low-cost programs available from TOTL.

TOTL.MONEYMINDER is a disk-oriented set of programs, compatible with most column expansion hardware, with advanced features to simplify use and increase flexibility, such as a configuration file for one-time setup for screen colors, column width, and any printer-interface combination. TOTL.MONEYMINDER will allow up to 110 user-defined accounts (for expenses, checking, income, charge accounts, etc.), with double entry transactions that can be spread across multiple expense accounts. TOTL.MONEYMINDER provides a monthly reminder of all bills due, and a powerful monthly budgeting capability. The system will record up to 8400 transactions per year, and includes capabilities to report net worth and both printed and graphic display of actual versus budgeted expenses, with reports on the status of each account.

Other important features include a built in database for long term assets and liability records, with investment return and amortization tools. Full checkbook balancing and statements are provided for, and a label file is easily generated to allow printing of address labels with TOTL.LABEL, and interaction with TOTL.TEXT and TOTL.INFOMASTER for even more versatility.

At year end a report is generated which shows all income and expenses, groups expenses into deductible and non-deductible, and groups deductible expenses into their deduction categories.

More information can be obtained by contacting:

Charles Palmer-McCarty
President, TOTL Software Inc.
1555 Third Ave.
Walnut Creek, CA 94596
415 943-877

WATCOM Pascal for the Commodore 64

Pascal is a widely respected language, particularly for teaching computer science because it encourages students to write readable, structured programs and to think about programming in a logical way.

WATCOM Pascal for the Commodore 64 is a full function

Pascal* conforming to both ANIS and ISO-draft standards and extended to support Commodore 64 features such as sprites, sound synthesizer and colour and bit-map graphics. WATCOM Pascal is already available on a number of micros and mainframes including IBM VM/SP CMS, IBM PC/DOS, DEC VAX/VMS, the Commodore SuperPet and the CEMCORP ICON.

• The one omission from the standard is that you cannot pass a procedure as a parameter.

- WATCOM Pascal is unique as an interactive interpreter. An interpreter is an extremely efficient tool in both program development and teaching because it gives the user immediate feedback and execution rather than waiting through the usual compile/link/execute steps.
- WATCOM string extensions to Pascal provide the ability to handle variable-length strings and improve normal string manipulation facilities.
- Extra input/output features have been included. Relative access files and extensions to the reset and rewrite commands allow system file names to be used.
- Built-in procedures and functions permit machine level interface including PEEK to examine memory, POKE to st into memory , SYSFUNC and SYSPROC to call machine language routines and ADDRESS to obtain a variable's machine address.
- The CASE statement has been extended with the ELSE clause to allow for cases which are not defined.
- WATCOM Pascal contains a function to generate random numbers; a feature not normally part of the language implementation.
- An interactive debugger provides immediate execution of Pascal statements, execution of a Pascal program one statement at a time and invocations of the debugger from any point in a running program.
- WATCOM Pascal has also provided a bit-mapped graphics capability, a significant expansion of the normal graphics modes available on the Commodore 64.
- SYSFUNC or SYSPROC commands can have parameters which are passed to the called program. In the case of SYSFUNC the machine-language routine may also return to the integer value.
- Function keys can be used to provide many useful editing operations including insertion, deletion, splitting and joining of entire lines.

WATCOM Pascal for the Commodore 64 is documented in one book which explains both the language and the editor. This book is designed to be either a self-teaching tool or a textbook for a course and could be used at the introductory level in high school or university.

The book includes a language primer with step-by-step examples, an advanced section on Pascal suitable for a second level course, a complete editor reference manual and finally the full syntax and semantics of the language.

WATCOM Pascal for the Commodore 64 is packaged as three separate components. A diskette containing the WATCOM Pascal interpreter, a cartridge containing the WATCOM Editor in 16K of ROM, and the textbook.

Waterloo Structured BASIC For The C64

Waterloo Structured BASIC extends the normal BASIC system on Commodore-64 machines to include Structured Programming statements. These extensions are necessary in order to teach proper programming methodology using the BASIC language. The Commodore-64 version is similar to the implementation used since 1980 with the Commodore PET.

There are several academic benefits to be derived from the Waterloo Structured BASIC system. Most notably, programs written with the system are more readable. This means:

- Programs are easier to write, debug and maintain
- The BASIC language is easier to teach and to learn
- Students learn important principles which can be applied with other programming languages.

Students learn proper programming discipline and style. They write Structured Programs, not the old-style "spaghetti code".

Structured Statements
Waterloo Structured BASIC extends the normal BASIC with new statements to control loops and selection with IF statements.

Procedures
It is important to be able to modularize a program in a meaningful way. This is accomplished in Waterloo Structured BASIC with procedures. A procedure is a group of BASIC statements which are given a meaningful name:

```
PROC name
    . . . BASIC statements
ENDPROC
```

The procedure can be invoked from anywhere in the program using a CALL statement (ie. CALL name)

Several commands are added for program development:

AUTO	- automatically generates line numbers as program text is added to a program.
DELETE	- used to remove a range of lines from a program.
RENUMBER	- renumbers the lines in a program including all references.

Waterloo Structured BASIC for the Commodore 64 is documented in one book which is designed to be either a self-teaching tool or a textbook for a course. The book is a tutorial and reference manual which assists the student in learning the modern concepts of Structured Programming and top-down design. Additional copies of the text may be purchased from WATCOM Publications (see below).

Waterloo Structured BASIC for the Commodore 64 has two separate components; A cartridge containing Waterloo Structured BASIC in 4k of ROM, and a book containing tutorial and reference material for Waterloo Structured BASIC. For additional texts or information, contact:

WATCOM Products Inc.
415 Phillip Street
Waterloo, ON N2L 3X2
519 886-3700
Telex 06-955458

Hardware News

SELECT-A-RAM - 64K for the Commodore VIC 20

Advanced Processor Systems introduces the SELECT-A-RAM, a 64K memory expansion cartridge for the Commodore VIC 20. The SELECT-A-RAM provides two expansion slots for program and game cartridges or additional memory expansion up to 192K. Decoding circuitry in the SELECT-A-RAM allows switching RAM and ROM in 8K blocks by inputs generated from the keyboard or by software command.

The SELECT-A-RAM plugs directly into the memory expansion slot on the VIC 20 and is powered by the VIC 20 supply. Other features include write protection, reset switch and optional external power.

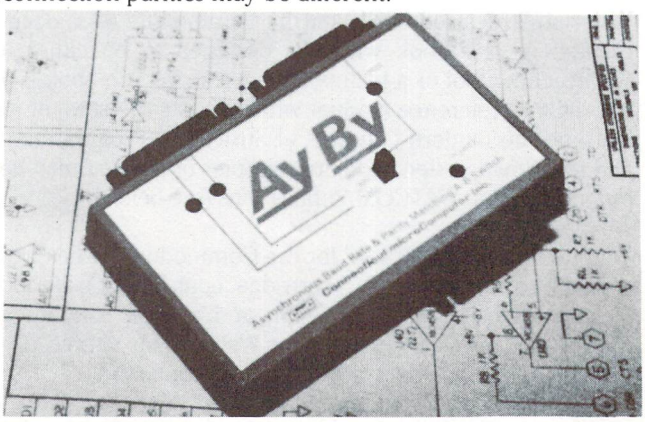
The use of high density dynamic RAMS with transparent refresh makes the SELECT-A-RAM the lowest cost per bit memory expansion product on the market today for the Commodore VIC 20.

The SELECT-A-RAM is covered by a one year warranty and a 15 day money back guarantee. The SELECT-A-RAM is priced at $169.00 in single unit quantities.

Paul G. Jones, Public Relations Director
Advanced Processor Systems
PO Box 43006
Austin, TX 78745-0001
512 441-3202

Asynchronous RS-232 Baud Rate & Parity Matching A-B Switch

Connecticut microComputer announces AyBy, an RS-232 A-B switch with baud rate and parity matching capability. Designed for both office and laboratory use, AyBy enables a user of a 9600 baud terminal to instantly switch from a 9600 baud connection to a 1200 or 300 baud connection. The connection parities may be different.



AyBy is equipped with three female DB-25S connectors: one for the 9600 baud terminal; and one for a 1200 or 300 baud connection, such as a modem. Parities for the 9600 baud and 1200/300 baud connections may be independently set.

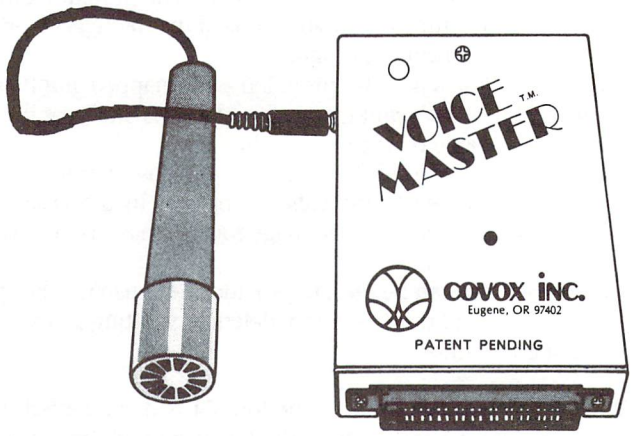
With a terminal set at 9600 baud, to change the connection from 9600 to 1200 or 300 requires merely moving a slide switch. An LED indicates the port and speed in use.

AyBy, in a tan and white high impact plastic case, sells for $300. Contact:

Connecticut microComputer Inc.
36 Del Mar Drive,
Brookfield, CT 06804
203 775-4595
Twx: 710:456-0052

Voice Master

Speech with high intelligibility and naturalness can be recorded for computer response using the Covox Voice Master. Requiring only about 400 bytes for the average word, up to 64 numbered words or phrases or other sounds can reside in memory for instant recall using simple BASIC commands. Complete vocabularies can be stored on disk or tape to extend the number of available words without limit. Voice Master makes it possible to devise practical applications for talking computers - from robot advisories to cockpit announcements to video games - at far less cost than ever before.



Based on zero crossings and amplitude information, the unique technique gives good results at bit rates under 7000 per second (user selectable). For some computers, the Voice Master device itself is needed only for recording; user created software then functions without the added hardware. Software which extends Voice Master capabilities to word recognition is also scheduled for release. Available for the C64 at $89.95 US. Contact:

Covox, Inc.
675 Conger Street
Eugene, OR 97402
503 342-1271
Tlx: 706017

Letters

Copywrites Rights: This letter is a request for a renewal of my faith in you. Over the past few years I have witnessed the Transactor bloom from a small Commodore newsletter to a handsome bi-monthly journal. What I ask of you is to give me your views on the status of programs published in your magazine.

The reason why I have written to you is because of a very recent issue of Compute magazine, May 1984, in which they state that only one person per purchased issue can use the programs contained within the pages of their magazine. On pages 13 and 14, a letter was printed from Gary Lee Crowell. In this letter, Mr. Crowell asked for Compute's views on the use of Compute's programs by users groups and libraries who have a subscription to their magazine. Computes answer was a quick and simple no. "You can only use the programs in an issue of Compute if you own a copy of that issue".

In my opinion, this stand is one of which makes them look pretty shallow, and also one that would be next to impossible to uphold. All this law of theirs proves is that they no longer care about the education of the general consensus, only the lining of their pockets by the increase in sales. I really wonder how many schools, users groups and libraries would keep their subscriptions if they realised how self serving Compute's policies are. How many teachers would use Compute's articles and programs to help teach their students if they realised that they were breaking the law to do so ?

Jim Butterfield has always been known to help promote greater education by his articles, lectures and programs. He has been seen on television, can be obtained on video tape, and can be read in numerous magazines. User group libraries throughout the world are filled with large quantities of Jim Butterfield's programs, released into public domain by Jim Butterfield himself. Now, considering that Mr. Butterfield is the Associate Editor Of Compute Magazine, why has Compute allowed Mr. Butterfield to release his programs into public domain after they have been printed in the sacred pages of their once great rag?

As far as I can tell, they would never dare cross Mr. Butterfield, for they would risk losing his contributions, and thus large volumes of sales. In the circle of Commodore users worldwide, Compute' is usually bought because of Mr. Butterfields articles, not because of their 50% volume of advertisements, nor their pages and pages of articles on computers other than Commodore, or even the small content of articles on Commodore computers. It is usually because of Mr. Butterfield and his words of wisdom. Beginners, intermediate and advanced programmers alike have always gained from Mr. Butterfields knowledge, as I have, until now. Compute has lost my support in their magazine, and with it I have lost future knowledge gained through the pages of Compute. But, as far as I can tell, Transactor, Commodore Magazine and a couple of others have supplied me with plenty of knowledge in the past, and it is quite evident that they will continue to do so in the future. Your magazine alone has been advancing so quickly over the past short while that it will more than make up for the space left by Compute.

Thank you for allowing me to take up room in the pages of your magazine. I know that your views are favourable on the status of the programs you print, but please confirm it for me. I would like my children to learn from the best sources available, but not if they have to break the law in order to do so. I hope that more people will think of this before they buy their next copy of Compute. Does their policy mean that if the father buys the magazine, the children are in default of the law if they use the programs contained within? A very stupid question, one which is only surpassed by a stupid policy.

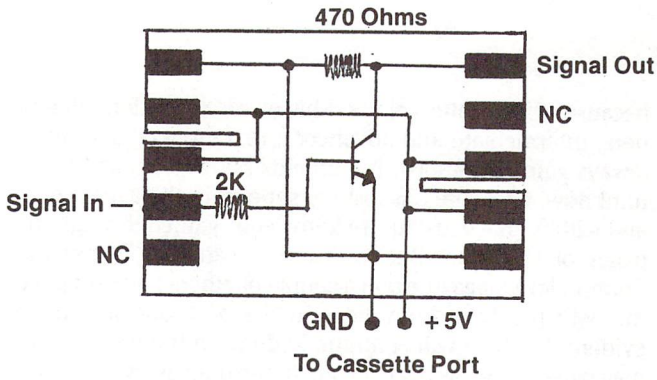
Edward C. James – Los Angeles California

When I read your letter I couldn't believe Compute would actually have the audacity to print such bilgewater. I found a copy of the May issue, and sure enough. Incredible. You're right though, I can't imagine how they would enforce that policy. Consider the writer. Does this mean he cannot pass out copies of his own program? And if he does, what then. Is Compute going to sue him? Or anybody else for that matter. If it weren't laughed out of court, the settlement wouldn't even cover lunch at the cafeteria. As for Jim's stuff, he holds a personal copyright on everything he releases for publication. Since he carries a little more clout than other writers, he can get away with it and still see his articles published. Anyone else who tries it may never see their article again. As for our stuff, rest assured that any copies of programs you enter are passed around with our blessing. In fact, 99% of all our articles are available for reprint in any other publication, provided the author is paid for it. Contact us anytime. We'll even be flattered that you express the desire to do so.

Take That Tape Worms! In response to R.D. Anderson's letter in Volume 5, Issue 01, I am quite sure that his difficulty is that a phase inversion is required between the originate and duplicate tapes. I had precisely the same problems and would have said the identical things about what I observed!

Enclosed is a connection diagram that works well for me which is used for duplicating tapes for the San Luis Obispo Commodore Group. The transistor is any handy NPN type; a 2N2222 works fine. The "interface" was made from scrap PC board.

Cliff Buttschardt, Morro Bay, CA



Originate Tape Recorder Side

Duplicate Tape Recorder Side

You might have noticed there is one pin missing from Cliff's interface. Pin 6 is the Cassette Button Sense line. It's used so the computer can detect if any buttons are depressed on the cassette unit. Since this won't be required for the intended operation, it's been left unserviced.

WordPro Quips: I Have two Commodore 8032 with 8050 disk drives and I use WordPro IV. I also have a telephone modem. Would your newsletter answer the following questions:
James L. Robinson, Jr., Tucson, AZ

1. Is there a way to exit from WordPro to BASIC without having to power down?
Yes. Control, Shifted 'Q'.

2. I am thinking about increasing the capacity of my 8032 by adding a 64K board. Will this affect the operations of WordPro? Will I be able to use the additional 64K for text space?
Adding 64K to your 8032 will not affect the operation of WordPro in any way, including the addition of extra text space. WordPro does not check to see if the extra 64K board is there so it will never even attempt to use it. There was a version that did use the extra memory. Instead of 2 text areas it had 5 but I don't know if it was ever released. Perhaps contact Pro-Line (see ad).

3. I may want to upgrade to WordPro IV Plus. Will I be able to use the files that have formats preceded by a checkmark rather than an @?
Confusing question. As far as I know, all later versions of WordPro use the checkmark to precede commands. Perhaps you saw some documentation that shows the @ sign. This is probably because the printer that created it wasn't capable of printing a checkmark. Your files should all be completely compatible.

4. Do you know of any other programs that work in conjunction with WordPro?
MailPro was written by the same author as WordPro and was designed to accept WordPro files for input. WordPro can produce output files that are acceptable by any other program that can input from a disk file. Telecommunication programs that can send text from disk are a good example.

5. I use both the NEC 5530 and a Mannesman Tally 8024 (also a 4032 Dot Matrix). The formatting for the NEC does not work the same as on the Mannesman Tally. Underlining in particular. Do you have any special instructions for the Mannesman Tally?
First of all, I'm not sure the Mannesman Tally will do underlining, but I could be wrong. The problem is, printer manufacturers are only beginning to get together on code values for printer feature controls. So what invokes one function on one printer may invoke something entirely different on the other. Late versions of WordPro allow you to define special characters in a command line. Perhaps with this you might be able to send control characters to the Mannesman that will make it behave. You'll require your Mannesman manual and some experimentation. M.Ed.

Joy Cursed: *We've received recently a couple of letters concerning "JoyCursor", a program published in the last issue of The T. One claimed that the program returned an ILLEGAL QUANTITY ERROR in line 170. Line 170 is the line that POKEs the values into memory. An illegal quantity could only occur in one of two places: the address or the value to be POKEd. If the address goes, for some reason, negative, or if it goes above 65535, an illegal quantity results. If the value to be POKEd goes negative or above 255, same thing. Check to make sure you have commas separating all the numbers in your DATA statements. If one is missing, you'll have problems. If it's missing between, say, a 1 and a 17, BASIC will READ a '117' and the loader will run out of data before reaching the end. If the comma is missing between, say, a 34 and 212, BASIC will try to POKE the location with 34212. No way. This is a potential hazard for any program containing DATA statements. If you run into problems at the line containing the POKE, check your commas for one missing.*

Also, make sure your joystick is plugged into the specified port. Even with no program in the machine, the joystick will cause sporadic activity on your screen. Try it! The reason? The chip that handles the joystick ports is the same one that services the keyboard. When signals are sent in through the port, the 64 thinks they are coming from the keyboard. Therefore a program is required to help the 64 determine the source of the activity. Such a program will check the high bit of the port register. If it's zero it's because a joystick is grounding that pin. Perhaps a complete article explaining this technique is in order.

Now with the Joycursor program in place, try it again. If you get the same wierdness, try the other port. M.Ed.

Bits and Pieces

Line Doo Daa

Our first screen blitz was submitted by Giovani Polese of Downsview, Ontario. The program shown is somewhat longer than it has to be – try PRINTA$ after running it once. We changed it to make it enterable from all keyboards (business keyboards don't have some of the graphic characters available). Try changing the '15' in line 30 to 14, 13, 12, etc., for different effects.

```
30 for j = 1 to 15 : read a
40 a$ = a$ + chr$(a) : next
50 print " SN " ;
60 print a$; : goto 60
70 data 164, 210, 198, 192, 195, 196, 197, 163
80 data 197, 196, 195, 192, 198, 210, 164
```

Colourtest

Colourtest is a simple little program that merely draws boxes in all the colours available so that you can adjust your TV/monitor for the best possible contrast. Like the program above, it too is much longer than it needs to be. For example, lines 115, 120 and 125 can be replaced by C$ = all the colour control characters except black, which is the background colour used for the test. This will also save you from entering the DATA statements. Lines 135 and 140 can be replaced by B$ = 1 cursor down and 5 cursor lefts.

```
100 print " S " : rem clear screen
110 poke 13*4096 + 33,0
115 for i = 1 to 15
120 read a : c$ = c$ + chr$(a)
125 next
130 s$ = "     ":rem 5 spaces
135 b$ = chr$(157)
140 b$ = " q " + b$ + b$ + b$ + b$ + b$
145 for i = 1 to 15
150 print mid$(c$,i,1);
155 print " s " ; : rem home
160 for j = 1 to i:print:next
170 printspc(i*2) " r " s$b$s$b$s$b$s$
175 for j = 1 to 300:next
180 next i
200 data   5,  28, 159, 156,  30,  31
210 data 158, 129, 149, 150, 151, 152
220 data 153, 154, 155
```

Would You Buy A Used Car From This Man?

Bytefinder

Have you ever needed to know what byte values are NOT present in a program or file you may be working on? The situation arises when you need a value to act as a terminator. If this same value exists elsewhere, the file will be terminated prematurely. The following program will show which values are not present in the 4K ROM block between $F000 and $FFFF. Quite simply, the program counts the occurence of byte values (line 120) by incrementing the approriate array element of U(. Naturally, all the values will lie between 0 and 255, hence DIM U(255). The elemnts of U(that remain zero indicate values that were not encountered (line 210).

```
100 dim u(255)
110 for j = 15*4096 to 65535
120 x = peek(j):u(x) = u(x) + 1
130 next j
200 for j = 0 to 255
210 if u(j) = 0 then print j;
220 next j
```

This could be easily altered for any area of memory, or for any disk file by changing:

```
110 open 8,8,8, "some file"
120 get#8, a$ : sx = st
125 x = asc(a$ + chr$(0)) : u(x) = u(x) + 1
130 if sx = 0 then next j
140 close 8
```

Quick Note: Remember, a COLLECT D0 or OPEN 1,8,15,"V0" never hurts, especially after you see something strange happen. You know what to change for drive 1.

UN–DIMension

As you know, any attempt to DIMension an array that is already in use will result in the REDIM'D ARRAY ERROR. In fact, the only way to DIM an array by the same name twice is to issue a CLR which destroys all your other variables.

In most cases you shouldn't have to define an array more than once. But sometimes a program may lack memory for a particular operation because some array that isn't required is occupying valuable space. The program would be required to determine if array definitions could be erased without losing valuable information. Then, using the following techniques, some or all of the arrays could be eliminated. After performing the sort, etc., the arrays can be re–defined, ready for further use.

In another case, you may have an array that is too small. When your program detects this, invoke UN–DIM and re–DIM the array (by the same variable name) at the new larger size.

This method can not quite be called 'dynamic dimensioning'. First, you must actually eliminate the array before it can be re–defined. Any important data contained in the target array must be re–established after it is re–DIMed. Secondly, you cannot eliminate an array without affecting other arrays defined at a later time. In-otherwords, the last array defined will be the first one erased, and so on. Therefore, it is best to DIM the arrays first that will be considered permanent and DIM the "variable" arrays last.

Function A(Q) (line 100) measures the "distance" in bytes from the Start of Arrays Pointer to the End of Arrays Pointer. When new simple variables are defined, both these pointers change as the arrays get pushed higher in memory. But the size of the arrays hasn't changed. So to erase an array, you simply back up the End of Arrays Pointer by the same distance (line 140). BASIC only looks up to the End Pointer for existing arrays, so if it isn't found DIM is allowed.

The next program is an "untaxed" and less commented version of the program after it.

VIC 20 / Commodore 64 Version (For BASIC 2.0/4.0 subtract 3 from all PEEK/POKE address in the first 5 lines.)

```
100 def fn a(q) = (peek(50)-peek(48))*256
      + peek(49)-peek(47)
110 def fn hi(q) = peek(48) + int(q/256)
120 def fn lo(q) = peek(47) + (q and 255)
130 goto 160
140 poke 50, fn hi(x) : poke 49, fn lo(x) : return
150 rem *** start of program ***
160 dim a(10), c(15), b(15) : a(3) = fna(0)
170 dim j(20), i(20)          : a(5) = fna(0)
180 x = a(3) : gosub 140 : rem clr j( & i(
190 dim j(100),i(100)        : a(5) = fna(0) : rem re-dim
200 dim ad(250)
210 x = a(5) : gosub 140 : rem clr array ad(
220 x = a(3) : gosub 140 : rem clr j( & i( arrays
230 x = 0    : gosub 140 : rem clr all arrays
```

BASIC 2.0/4.0 Version (For VIC/64, add 3 to all PEEK/POKE addresses in first 6 lines.)

```
100 def fn a(q) = (peek(47)-peek(45))*256
      + peek(46)-peek(44)
110 def fn hi(q) = peek(45) + int(q/256)
120 def fn lo(q) = peek(44) + (q and 255)
130 goto 180
140 rem --- clr array subroutine ---
150 poke 47, fn hi(x) : poke 46, fn lo(x)
160 return
170 rem *** start of program ***
180 dim a(10), b(15), c(15) : a(3) = fna(0)
190 rem a(3) = bytes used by first 3 arrays, a(, b( & c(
200 p = 3.14159 : i% = 10 : etc$ = "and so on"
210 rem arrays move up as simple variables are defined
220 rem however, a(3) remains the same
230 dim j(20),i(20)          : a(5) = fna(0)
240 rem new arrays, a(5) = bytes used by all 5
250 r$ = chr$(13) : q$ = chr$(34)
260 rem and perhaps some new variables
270 x = a(3) : gosub 150
280 rem clr arrays j( & i(, leaving a(, b( & c( intact
290 dim j(100),i(100)          : a(5) = fna(0)
300 rem re dim j( & i(
310 dim ad(250)
320 x = a(5) : gosub 150 : rem clr last array
330 x = a(3) : gosub 150 : rem clr j( & i( arrays
340 x = 0    : gosub 150 : rem clr all arrays
```

ERROROUTER **Scott MacLean, Toronto**

Many people have written programs that they do not want to have other people crash out of either by accedent or on purpose. The short program presented here traps all errors and re–runs the program if an error occurs. The program is written in BASIC, with a machine language routine loaded with data statements. It will work on the VIC or 64. Run the program and it will ask; "Install where?". Enter an address of safe RAM in your computer (see below). When you press RETURN it will enter the machine language section and activate it.

Safe places to install

C64	49152 or 828
VIC(5K)	7168 or 828
VIC(+8K)	16354 or 828
VIC(+3K)	7168 or 828

Location 828 is the tape buffer. Use it only if you are not doing any tape operations, otherwise the computer will crash when you get an error. To use this routine in your own programs, enter the data statements and read them into free RAM. Then poke locations 768 and 769 with the LO/HI address of the place you put the program in. It will then be activated.

How it works

Locations 768 and 769 are the locations which tell the computer where to go if it encounters any kind of error. By POKEing these locations with our own numbers, we can tell the computer to execute our own program instead of it's regular error routine. This

program POKEs the numbers representing RUN and a chr$(13) (return) into the keyboard buffer. Then it jumps to the normal error routine. The computer then displays the error and checks the buffer. It sees some characters there and assumes the user typed them, so it displays and executes them, thereby re-RUNning the program in memory.

This program could be used for just about any program you write, it makes it virtually crashproof. I use it on my bulletin board, so if someone manages to crash it, it simply restarts itself, hanging up on the user in the process. I'm sure you'll find many other uses.

```
5 l = peek(768) : h = peek(769)
10 data 169, 82, 141, 119, 2
15 data 169, 85, 141, 120, 2
20 data 169, 78, 141, 121, 2
25 data 169, 13, 141, 122, 2
30 data 169,  4, 133, 198, 76, 256
35 print chr$(147);
40 input "install where";x : y = x
50 read a
55 if a = 256 then 75
60 ck = ck + a
70 poke x,a : x = x + 1 : goto50
75 poke x, l : poke x + 1, h
80 if ck<>2568 then print "data error" : end
90 hi = int(y/256) : lo = y-(hi*256)
100 print "installed at" y
110 poke 768, lo : poke 769, hi : new
```

Line Hider

Line Hider does just that – hide lines of code that you don't want shown without affecting their operation in the program. However, if you use Line Hider to hide a line that is the target of a GOTO or GOSUB, you'll get an UNDEF'D STATEMENT ERROR. Use the next utility for these.

There's just one trick to using it – you must supply the input with the number of the line that comes BEFORE the one you wish to hide. It wouldn't be hard to modify this to hide an entire program!

```
100 rem  save "@0:line hider",8:verify "0:line hider",8
105 rem  * hide a line within your basic program
110 rem  * basic 4.0 : sb = 1025
115 rem  * c64 only  : sb = 2049 (default)
120 rem  * vic only  : sb = 4097 (default)
125 :
63989 sb = 1025 : rem ** set-up for basic 4.0
63990 input "line # of preceding line ";pl
63991 for lp = 1 to (2↑16)-1
63992 num = peek(sb + 2) + peek(sb + 3)*256 : rem * line #
63993 nxt = peek(sb) + peek(sb + 1)*256
63994 if num < pl then sb = nxt : next lp : end: rem * still below
                                                        the line
63995 if num > pl then print "line not found" : end
63996 sh  =  peek(sb) + peek(sb + 1)*256  : rem * position of
                                                        line to hide
63997 nl = peek(sh) : nh = peek(sh + 1) :rem ptrs to next line
63998 poke sb,nl : poke sb + 1,nh :rem bypass the line to hide
63999 poke sh + 2,0 : poke sh + 3,0 :rem and change line #
                                                        to zero
```

Ghost Liner

Ghost Liner does just what Line Hider does, except the line number will be displayed with nothing beyond it. Ghost Liner searches for lines that start with 5 colons. It substitutes the first colon with a zero. When the LIST routine sees this zero, it assumes end of line and goes on to list the next line. RUN is not affected.

```
100 remark * ghost liner – rte
110 remark * cloaks all lines starting
120 remark * with ::::: (5 colons)
130 remark * basic 4.0 : vl = 42 : vh = 43 : sb = 1025
140 remark * c64 & vic : vl = 45 : vh = 46
150 remark * c64 only  : sb = 2049 (default)
160 remark * vic only  : sb = 4097 (default)
170 :
180 : vl = 42:vh = 43:sb = 1025 : rem * basic 4.0 set-up
190 loc  =  peek(vl) + 256*peek(vh)
200 printchr$(147)loc," : maximum "
210 print," : current "
220 if peek(sb)<>58 then 250
230 ct = sb : for lp = 0 to 0 : ct = ct + 1 : lp = (peek(ct) = 58) : next
240 if ct > sb + 4 then poke sb, 0 : sb = sb + 4
250 sb = sb + 1 : print chr$(19)chr$(17)sb : if sb<loc then 220
260 end
```

List Decorator

With all the screen function characters available for changing colour and cursor position, why not make use of them while LISTing as well as when you RUN. List Decorator will take dull, unoticeable remarks and make them bright and easy to spot. The list below shows what value to use for the possibilities. You need not stop at one though – after running it once on itself (see line 160 & 170), LIST the program and insert new @ signs in the same place. Now RUN again. List Decoarator will replace all occurences of "REM @" with RB.

```
rb = 5   for white line (c64 & vic)
rb = 7   for ring the bell
rb = 13 for carriage return
rb = 14 for upper/lower case
rb = 15 to set the top left corner (cbm only)
rb = 17 for cursor down
rb = 18 for reversed program rem lines
rb = 19 for cursor home
rb = 20 for delete char
rb = 21 for delete a line (cbm only)
rb = 25 for scroll down (cbm only)
rb = 28 for red line (c64 & vic)
rb = 29 for cursor right
rb = 30 for green line (c64 & vic)
rb = 31 for blue line (c64 & vic)
```

```
100 rem * list decorator – rte
110 rem * lb = 42 : hb = 43 : sb = 1025 :rem * for basic 4.0
120 rem * lb = 45 : hb = 46      :rem * for c64 & vic
130 rem * sb = 2049              :rem * for c64 (default)
140 rem * sb = 4096              :rem * for vic (default)
150 :
160 rem @ this is how your remark should look when entered
```

```
170 rem @ every occurence is substituted
180 :
63995 lb = 42 : hb = 43 : sb = 1025 : rem basic 4.0 setup
63996 input " replacement byte for @ " ;rb
63997 mx = peek(lb) + peek(hb)*256 : for a = sb to mx
      : b = peek(a) : if b<>143 then285
63998 if peek(a + 1) = 32 and peek(a + 2) = 64 then
      poke(a + 2),rb
63999 next : end
```

Sinhibitors

This next collection of handy POKEs was submitted by Adam Foster of Kingston, Ontario.

Many software companies go through a great deal of trouble to stop program pirates from stealing their software. But no matter how much protection you have on a program, if the pirate really wants to get in, he will.

On the VIC 20 and Commodore 64 there are several easy POKEs to stop the common thief. I stress the word "common" since any experienced pirate will get by these easily.

List Terminator

This feature will prevent others from viewing your program. On both the VIC and the 64 add a line to:

POKE 775, 200

To re-enable LIST, POKE 775 with 167 on the 64 and 199 on the VIC. Unfortunately, it only works if the program has been RUN before they try and LIST it. (see LockDisk later on – M.Ed)

Save Terminator

The 64 version of this stops the saving of your program by disabling the RUN STOP/RESTORE keys. To do this:

POKE 808, 225 : POKE 818, 32

To return to normal POKE both locations to 237. On the VIC, this killer is enabled by:

POKE 802, 0 : POKE 803, 0 : POKE 818, 165

and is disabled with:

POKE 802, 243 : POKE 803, 0 : POKE 818, 133

STOP Key

To disable the STOP key, add:

POKE 808, 225

to your program. POKE 808, 237 turns the STOP key on again. This works on both computers.

Keyboard Killer

POKE 649, 0

turns the keyboard off, and POKE 649, 10 turns it back on for both the VIC and 64.

Etch. . .,., A Sketch.

Not the quickest hi-res graphic aid, but it demonstrates clearly some fundamentals. Like setting up the hi-res screen, testing boundaries and adjusting for max/min, calculating hi-res position to the bit, testing for the fire button, and determining joystick direction. It wouldn't be tough to make this machine language. Written by Dave Gzik, Commodore Canada.

ETCHASKETCH

Here is a neat little program that converts your C64 into an etcha-sketch type tablet. To use this just load the program and run it. You'll need to have a joystick plugged into port 2.

Drawing is accomplished by moving the joystick in the direction you want and this program will draw in eight directions. If you want to lift the drawing pen just hold down the FIRE button and move where you want to go.

This is a very simple BASIC program, there is no cursor to indicate the location of the pen, so you'll be guessing when you lift it off the drawing area.

You can expand on this if you wish but it is rather slow in BASIC. Give it a try it's not that long or tedious.

```
5 rem etchasketch by dave gzik (cbm canada)
10 base = 2*4096 : poke 53272, peek(53272)or8
20 poke 53265, peek(53265) or 32
30 for i = base to base + 7999 : poke i, 0 : next
40 for i = 1024 to 2023 : poke i, 3 : next
50 x = 160 : y = 100 : rem start off point
75 if y<0  then y = 199
76 if y>199 then y = 0
77 if x<0  then x = 319
78 if x>319 then x = 0
80 row = int(y/8) : char = int(x/8) : line = y and 7
90 bit = 7-(xand7): byte = base + row*320 + char*8 + line
95 if fr + jv = 111 then 110
100 poke byte, peek(byte) or 2↑bit
110 jv = 15-(peek(56320) and 15)
111 fr = peek(56320)
120 if jv = 1 then y = y-1 : goto75
140 if jv = 2 then y = y + 1 : goto75
150 if jv = 4 then x = x-1 : goto75
160 if jv = 5 then x = x-1 : y = y-1:goto75
170 if jv = 6 then x = x-1 : y = y + 1:goto75
180 if jv = 8 then x = x + 1 : goto75
190 if jv = 9 then x = x + 1 : y = y-1:goto75
200 if jv = 10 then x = x + 1 : y = y + 1 : goto 75
210 goto 75
```

Editor's Note: Notice how Dave tests the fire button in line 95. This works no matter what direction the joystick is being held. Why? Because the joystick ports are inverted logic. This means when nothing is happening on the joystick (except for the fact that it's plugged in) the joystick register will contain a value of 127 (bits 0-6 on, 7 off which flags port 2). Line 110 un-inverts the value by first looking at only the first 4 bits, and subtracting that from 15 to

get direction values that make a lot more sense. As JV goes up FR goes down so FR+JV remains constant, whether the fire is down or not. But when the fire button IS down, that constant is 111.

C64 Default Screen Colours

This next item comes to us from R.D. Young of James Park, New Brunswick.

If your black and white TV has the blues, or at least if it doesn't like the blue default screen colours that appear on power-up, you can easily POKE in new colours. Then frequently and just as easily, you can watch your new colours disappear with each RUN-STOP/RESTORE key sequence and you must set them all over again. You may even have a favourite colour combination with your colour monitor. . . same problem.

Try the following little program. It loads a machine language program into any desired memory area, changes the "BASIC Warm Start Vector" to point there, and will keep your screen set to your own default colour combination.

The starting location for the machine language program is first selected. My default is decimal 900, the middle of the cassette buffer. Another usually safe place is between 49152 and 53232.

```
10 rem set default colours on run-stop/restore
20 rem by r.d. young
30 input "start location 900[left 5]";ad
40 for i=ad to ad+15 : read x : poke i, x : next
50 hi=int(ad/256) : lo=ad and 255
60 input "screen colour (0-15)  6  [left 3]";c
70 poke ad+1, c
80 input "cursor colour (0-15)  13 [left 4]";c
90 poke ad+9, c
100 poke 770, lo : poke 771, hi
110 sys 65126
500 data 169,   6, 141,  32, 208, 141
510 data  33, 208, 169,  13, 141, 134
520 data   2,  76, 131, 164
```

The defaults in the program are set to blue screen with light green text. Refer to any colour table (pg 159 in 64 User Guide) for colour codes that represent each colour choice. Both the screen and border are set to the same colour (my choice) but a little extra machine language could change all that. Happy RESTOREing!

Tape Saving Notes

Saving to tape from BASIC merely writes to tape everything that lies between the Start and End of BASIC Pointers. Saving to tape from the Machine Language Monitor allows one to save any area of memory because the user supplies the start and end address. The format is:

```
sys 4    ;enter monitor on BASIC 4.0 machines

.s "some name",01,6000,7000
```

. . .which saves all memory from hex 6000 to 7000 on cassette #1 using the name "some name". But the MLM Save always had one drawback. It would not save any memory above hex 7FFF. The problem lies in the tape write routines that Commodore designed years ago with the PET 2001. Commodore assumed back then that tape would never be written with data above 7FFF. So they used the high bit of the high byte of the address to signal end of write. When the current write address matched the end address (ie. End of BASIC Pointer), this bit would be set. The last byte would be output and, in a later part of the tape output routines, this bit would be detected and writing tape would be terminated. However, if the current write address goes above 7FFF, this bit is set naturally, but of course the tape close routine would have no way of differentiating and tape write would terminate.

Without telling anybody, it seems Commodore has lifted that restriction from the tape routines in the VIC 20 and Commodore 64. Although you must install your own MLM program (ie. Supermon, VICMON Cartridge, etc.) the following command will behave perfectly:

```
.s "some name",01,c000,d000
```

. . .will save to tape everything from $C000 to $CFFF. Remember, you must specify the last address desired, plus 1.

RESTORE X

This short machine language loader was submitted by Garry Kiziak of Burlington, Ontario. It allows you to RESTORE the DATA pointer to any DATA line as opposed to the first DATA line. And with just one single SYS. Written for the 64 or VIC 20.

```
10 restr=828:for k=restr to restr+31:read j:poke k,j:next k
20 data 32,253,174,32,158,173,32,247,
        183,32,19,166,176,5,162,17
30 data 76,55,164,165,95,233,1,133,65,
        165,96,233,0,133,66,96
100 for i=1 to 20
110 x=100*(int(rnd(1)*5)+2)
120 sys restr,x
130 read a$ : print a$
140 next
150 end
200 data i'm at line 200
300 data i'm at line 300
400 data i'm at line 400
500 data i'm at line 500
600 data i'm at line 600
```

TransBloopers

Voice For Commodore Computers: Vol. 5, Issue 01, Pg 71, Under sub-heading "Commodore 64 Notes", add line:

```
153 poke ra, peek(ra) and 251 :rem set pa2 low
```

Merging BASIC Programs: Vol. 5, Issue 02, Pg 54, Both programs will work as shown, but the checksum for the 64 version is wrong. Change '51230' in line 140 to 49379. Thanks to Nick Fournier for pointing that out.

And yes, that was Jim. B., '69

Two Reviews: PAL 64 and POWER 64

by Chris Zamara, Technical Editor

PAL 64

PAL 64 is an assembler for the 64 (PAL stands for Personal Assembly Language), written by Brad Templeton and distributed by Pro-Line software. Brad Templeton wrote PAL as a development tool to write POWER (also reviewed in this issue), and POWER and PAL work very well together. Like POWER, PAL was originally written for a PET years ago, so it is thoroughly debugged and tested by now. The 64 version is pretty much the same as the old PET version. On the PAL 64 disk there are some other utilities as well, including Jim Butterfield's SUPERMON 64, RPAL for creating relocatable object code, and a PAL unassembler, which converts object code to PAL compatible source code.

PAL is in some ways a conventional assembler, but very different in others. First of all, it conforms pretty well to the MOS standard in all but a few of the more esoteric conventions. What makes PAL unique, however, is how easy it is to use. Once PAL is installed, an assembler program is entered just as a BASIC program: using line numbers and the built-in editor (using POWER as well, if you have it). To assemble the program, just type RUN. That's it. A SYS command at the beginning of the program directs control to PAL, which does all the rest, interpreting the program as assembler code until it reaches the end or the END pseudo-op (which may be followed by BASIC code).

Most assemblers have their own editor, which produces a source file, which must then be assembled and link-edited to produce an object file. PAL is very easy to use, since the source program is entered using the familiar BASIC editor. Furthermore, you still have BASIC available if you wish to use it. You may even mix BASIC and assembler in the same program and create a hybrid module, using PAL's powerful BAS pseudo-op.

The other good thing about PAL is that it's fast. If the "print" option is turned off (no display of assembly output), even the longest programs assemble in a few seconds, providing there is no disk access. BASIC code may be executed after the PAL source has assembled, so assembling **and** re-running a major program is as simple as typing RUN. PAL's ease of use is its strong point.

PAL has some features found in full-featured assemblers which allow large source files to be created out of a number of smaller modules. The FIL pseudo-op brings the next module into memory for assembling. After the next module is assembled, control is **not** restored to the original source, so the FIL must appear at the end of a module. There are also pseudo-ops which save or load a symbol table to or from disk. This is a good way to communicate between modules.

A unique PAL feature worthy of explanation is the BAS pseudo-op. As mentioned above, use of this command in a source program allows intermixing of machine language and BASIC code (called "hybrid" programs). Assembler labels become available to the BASIC sections of code and can be accessed by a SYS command, for example: SYS "PLOT" could execute a machine language program labelled "PLOT" in the assembler code. When the hybrid program is RUN, the resulting object code will contain a mixture of BASIC and machine language in a single program which may be RUN directly from BASIC. This set-up works especially well when using POWER, since POWER allows you to switch back and forth between the source and object programs (the object program will only display the BASIC sections of code). Without POWER, the object module must be written to disk and loaded in separately to test it, which is a bit awkward. In this case, it's probably easier to just use separate machine language and BASIC programs, at least during development. Using POWER and PAL together (as the PAL manual recommends) makes for a potent programming environment for hybrid systems.

Another PAL advantage is that it is only 4K of code, and is fully relocatable. This means it can be burned into a 4K EPROM and plugged into the expansion port. Of course, living in only 4K also means that PAL does not have all of the features of a big, expensive assembler. No macros, link-editor, callable object modules, or label import/export capability in this package. PAL seems well suited to the 64's character: the Volkswagen rabbit of computers – not quite a BMW, but a good, mid-priced performer.

When using PAL to develop a large program containing smaller modules, the main problem is its lack of a link-editor. A module can't be separately assembled, since PAL must know its start address, which is dependant on all the preceding modules. PAL gets full marks for being easy to use, but using it for a such an application can be a bit clumsy, although there are ways to work around the problem.

Considering the speed (actually, the lack thereof) of the 1541 disk drive, a fully disk-based assembler could be frustratingly slow. Since PAL is not disk-based, by necessity it loses some features of a bigger assembler. To make up for it, PAL seems intended for convenience, and to allow a newcomer to jump right into assembly language without feeling intimidated. In that end it succeeds admirably, while also being thoroughly useable for serious programming. I would even go so far, in PAL's case, to use an over-worked cliche and call it "user friendly".

After reading the manual that came with POWER, I was disappointed with the PAL documentation. It is written by Brad Templeton, who appears to be much better at programming than he is at writing manuals. In all fairness, the manual does explain all of PAL's features and is not difficult to understand, but the organization is less than ideal, and the explanations get a bit muddled at times. Of course, this is in contrast with the POWER manual, which was exceptional. The PAL manual is still better than many.

In conclusion, PAL is an excellent choice for an assembler on the 64. It's very handy when you just want to whip up a little assembler

program and execute it without going through a lot of trouble. PAL can be used for serious system development, but just don't expect macros, a link editor, etc. For the 64, PAL is probably the best assembler available, and its ease of use is a bonus for beginners to assembly language.

POWER 64

Pro-Line's POWER 64 is the 64 version of Brad Templeteon's POWER, a software tool that programmers have been happily using for years now. Developed originally for the PET series of computers, POWER adds features to the editor and BASIC interpreter which increase programming productivity enormously. Unlike some other programmers' aid packages which just tack a batch of extra commands onto the BASIC interpreter, POWER is a well thought out, comprehensive system, which is very easy to use.

Those who have used POWER on PETs already know what it can do, and probably would not attempt to write a program of any magnitude without it. As Jim Butterfield writes, one quickly becomes "addicted" to the use of POWER. POWER 64 is not significantly different from the original POWER available on the PET series, but the POWER 64 disk also contains a useful program called "MOREPOWER", which adds some handy features and disk-accessing commands to the basic POWER package.

Besides adding commands, POWER adds two main features to the BASIC program editor: program scrolling, and "instant action" keys. The scrolling feature allows you to list a BASIC program forward or backward by moving the cursor down or up when it is at the bottom or top of the screen. This eliminates clumsy LIST commands, which invariably reveal a range of program lines which end just before the line you are really interested in. With POWER in place, just cursor down to reveal the next line.

The "instant phrase" feature allows you to assign one or more characters of text to any character on the keyboard, including SHIFTed keys or keys held down in conjunction with the CTRL or Commodore-symbol keys. The text assigned to that key will be printed out at the cursor position as soon as the key is struck. Any number of keys may be defined, and the definition is done by special REM statements within the program currently in memory. This means that different key definitions may exist for each program, and key definitions may be LOADed from tape or disk. Furthermore, many keys are pre-defined with BASIC keywords such as FOR, NEXT, PRINT#, etc. The instant keyword feature can be disabled if desired, as can the user-defined instant phrase keys. A key may also point to a BASIC subroutine which will be executed when the key is pressed. This is a very handy feature, giving the programmer any number of special functions at his fingertips.

There are some command-driven features thrown in to complete the programming environment, including TRACE (an exceptionally good one), a WHY command to point to the source of a program-killing error, and PTR, which restores BASIC pointers destroyed by LOADing a machine language program. There is also the standard fare: AUTO, DEL, DUMP, RENUM, and find/substitute commands. The line renumbering command, RENUM, is particularly useful – it allows you to renumber portions of your program as well as the entire thing. The find command (indicated by a slash) permits "wildcard" matching for characters or groups of characters. There are 15 commands in all, and MOREPOWER adds another 13. To his credit, Brad Templeton seems to have used a degree of restraint when adding commands. The necessary things are there, but one is not overwhelmed by hundreds of commands which would get little use. Instead, POWER makes it easy for the user to add his own commands and create a program like MOREPOWER which can run under the POWER environment. I think this is a good approach, since it makes the utility much more flexible and less cumbersome.

The commands added by MOREPOWER are just as important as the regular POWER commands if you are using a disk drive. With MOREPOWER installed, you may: **LIST** a program on disk without loading it into memory, LOAD and **RUN** a program from disk in one step, **MERGE** a program from disk with the one currently in memory, display a **TEXT** (ASCII sequential) file from disk, display the disk **ERR**or status, send commands to the **DISK** command channel, and change the default **DEVICE** number for LOADing. There are a few other convenient, albeit more mundane commands such as **HEX**, which performs hex/decimal conversions, and **KEY**, which, as previously mentioned, allows you to define an instant action phrase independently of the program in memory.

The manual, which comes with the system disk in the usual PRO-LINE mini-binder, is written by Jim Butterfield. His witty writing style makes the manual a joy to read, and he does an excellent job of explaining POWER and how to use it. It is written from the point of view of an objective user of the package, pointing out all of POWER's strengths and weaknesses (don't worry, the weakness:strength ratio is very small). Besides a casual and informative chat about POWER, Butterfield gives a formal description of all POWER commands in a separate section. Examples, and short learning and practice exercises are also provided. I wish all manuals were written this way.

Looking at the package overall, it seems that POWER gains its usefulness by allowing the programmer to define his own functions, and doesn't box him in with a long list of added commands. The main drawback that I find with the instant-action key system is that you must tack unsightly REM statements to the beginning of a program in order to define the keys (these REMs may, of course, be removed when development has ceased on the program, but we all know that never happens). Also, those definitions only prevail while the current program is in memory. This could be seen as an advantage in that each program can have unique key definitions, but too many times I have LOADed a little utility of some sort and subsequently found myself lost in the wilderness of an unPOWERed keyboard. MOREPOWER allows keys to be defined independently of a BASIC program, but provides no facility for SAVEing the key definitions to disk.

In conclusion, I have the following advice for you: If you do a lot of programming on your 64 in BASIC (or in assembler using PAL), and you wish to increase your productivity, buy POWER 64. Once you get used to it, you will turn out programs much faster, especially large systems involving many subprograms on disk (using MOREPOWER). In addition, using POWER will not take away any fun out of programming, since it adds as little or as much power as you require. If you write programs for profit as well as fun, the list price of approximately $70.00 will be easily repaid in increased productivity. The manual speaks wisely when it says, "Every time you power up [your system], remember to **POWER** up by loading from your POWER 64 disk".

The MANAGER Column

Don Bell
Brantford, Ontario

Creating A New File Or Revising An Old One?

Rather than jumping into a new application in this article, I would like to respond to a problem many users have mentioned in their letters.

When you first design a new application it is often difficult to imagine all the reports or searches you may want. At some time or other you will discover ways of improving on your record entry screen or file design. You may want to adjust layout of prompts and fields on the screen, add new fields, or extend the length of existing fields.

Some minor modifications can be made by revising the old file. More substantial modifications require creating a new file.

IT IS ALWAYS WISE TO ONLY ENTER A FEW RECORDS (say 10) WHEN YOU ARE FIRST DESIGNING YOUR FILE AND ONLY MAKE A SMALL FILE. Then you can experiment with the file design without worrying about blowing away a whole bunch of records that you invested a lot time entering. Also, you will not use up a lot of valuable space on your diskette since you are rewriting an old file instead of writing a new one each time you make a change in the file design.

I will now attempt to explain how you can revise your file design without fear of destroying all those records that you so painstakingly created.

WARNING! Revising your file can be a dangerous business. Before attempting to revise your file ALWAYS make a backup copy of your diskette on a new, unused diskette, using the BACKUP option in the main menu.

Revising An Old File

The word "REVISE" in the CREATE/REVISE option is slightly misleading. Using REVISE in the CREATE/REVISE option, you can ONLY perform minor modifications to your screen format and file design i.e. move the text prompts around or change the field types (numeric or alphanumeric). Any major alterations in your file design will result in your writing a new file and destroying all the records in your old file.

If you wish to revise your file or screen layout, begin by choosing the CREATE/REVISE option from the main menu.

IF YOU DO NOT WANT TO CHANGE THE NUMBER, SIZE OR SEQUENCE OF FIELDS IN YOUR FILE, press 'R' (for revise) and RETURN for the following screen prompt:

CREATE A NEW FILE/REVISE AN OLD ONE? R

Make minor modifications to the screen, making sure all fields longer than 1 character are enclosed in 'up arrows'. Press 'back arrow'.

Answer the next screen prompts as follows, pressing RETURN after each entry.

ARE YOU SURE(Y/N) Y

ANOTHER SCREEN (Y/N)? N

If indeed you have NOT ALTERED the file, the next screen prompt will be:

DO YOU WISH TO ALTER FIELD TYPES?

At this point it is usually a good idea to answer 'Y', then press RETURN. You can now cursor down through fields and check to see if all your field types are correct. Fields requiring only number entries are numeric (e.g. fields for dates or $ amounts). Remember, phone numbers with a space separator are alphanumeric, as a space character is considered alphanumeric. When you are finished checking all the field types press 'back arrow'.

If you HAVE ALTERED the file, the program knows you have altered the file, and confirms this with the following message:

FILE HAS BEEN ALTERED. NEW FILE(Y/N)?

This last prompt is extremely important. In addition to telling you that you have altered the file, the program is asking you if you want to create a new file. If you have only created a few records and don't mind destroying them then answer 'Y'. The program will then rewrite over the old file blanking all the records.

If you don't want to destroy the records in your old file, then answer 'N'. If you answer 'N', then the revise procedure will be aborted here. You now know that you cannot use the

'REVISE' part of the CREATE/REVISE option to make the changes you want to your file. Press 'F2' to abort this operation or return to the beginning of the CREATE/REVISE option.

Creating A New File Using Your Old File And Old Records

If you want to change the number, size or sequence of fields in the file, you will have to create a new file. Don't worry, you don't have to start from scratch. You can use your old screen to create a new screen and transfer records from your old file to the new file.

Enter the CREATE/REVISE option from the main menu. Enter a new filename (i.e. different than your old file name). Answer the next screen prompts as follows, pressing RETURN after each entry.

CREATE USING AN EXISTING SCREEN (Y/N)? Y

ENTER FILENAME? XMASLIST (or whatever name you used for the original file)

You now have the option of changing the border, background and cursor colours.

Your original screen will appear with all the field indicators. Make the necessary changes to the screen and then press 'back arrow'.

Answer the next screen prompts as follows, pressing RETURN after each entry.

ARE YOU SURE (Y/N) Y
ANOTHER SCREEN (Y/N) N

At this point you must decide if there will be enough space on the same diskette for both the old and new files. If either of your files is over 100 records, then it's probably a good idea to put a new formatted diskette in the drive. In any case, it's always a better idea to start a new application on a new diskette.

Answer the next screen prompt as follows, pressing RETURN after the entry.

IS THE FILE DISKETTE IN THE DRIVE? Y

The disk drive will then check to see how much free space is available on the diskette and tell you the maximum number of records you will be able to create in your file. You can then either accept the maximum number or choose to create fewer records. It is best not to choose the maximum number, as you want to save some disk space for your report files. Also, you don't have to make the new file the same size as the old file. The file should at least be large enough to transfer the old records over to the new file.

Copying Records From Your Old File to the New File

Now to copy records from your old file to your new file. First, you will need printouts of the field numbers for both of your files. These will aid in your equating similar fields in your source file and destination file. Begin by entering the MANIPULATE FILES option from the main menu. Then choose the RE–ARRANGE A FILE function in the MANIPULATE FILES menu.

Now there's a small stumbling block at this point in the program which you may or may not have found confusing if your source and destination files are on different diskettes. At the top of the screen you are prompted to ENTER DESTINATION FILE NAME and at the bottom of the screen you are prompted to PLACE THE DESTINATION DISK IN THE DRIVE. Perform the command at the bottom of the screen first i.e. before entering the filename you must first place the destination diskette in the drive. The same applies to the next 2 screen commands – ENTER SOURCE FILE NAME and PLACE THE SOURCE DISK IN THE DRIVE. Place the source disk in the drive and then enter the file name. Refer to page 39 in the manual on Rearranging A File.

You will then equate fields in your new file with fields in your old file. IF YOU DO NOT WISH TO COPY INFORMATION INTO A FIELD, ENTER '0'and press RETURN. (I suggest you correct the documentation on page 39 which wrongly states: "If you do not wish to transfer any data to this field, simply cursor off the line to the next field.)

When you have finished defining which source fields relate to which destination fields, press 'back arrow'.

At this point, arm yourself with the patience of Job. Waiting during the data transfer process may make you feel like an electronic Methuselah. You will have to wait long intervals while the program is either reading the old file, writing the new file, or doing a mysterious garbage collection without telling you. DO NOT TRY TO SHORT CIRCUIT THE DATA TRANSFER PROCESS AS YOU MAY DAMAGE YOUR FILE.

DON'T PHONE – WRITE!

If you have questions regarding this application or you would like to suggest ideas for future columns, please write me a legible, coherent letter, including sample data and screen dumps. I will attempt to answer letters in this column. Write to: Don Bell, BMB Compuscience Canada Ltd., 500 Steeles Ave., Milton, Ontario, Canada, L9T 3P7.

Social Insurance Number Checker

by James Whitewood, Milton, Ontario

In Canada, and several other countries, everyone, everywhere, that is eligible for work, is assigned a Social Insurance Number or SIN. Canadian SINs can be verified with a

simple formula that can be implemented with the Manager '64 Math function.

SIN checking works this way. Using the SIN: 447 188 350

Step 1.	447188350	The last digit (0) is the check digit
Step 2.	4+7+8+3=22	Take the 1st, 3rd, 5th and 7th digits and sum them.
Step 3.	4 1 8 5 x 2 8 3 7 0	Take the 2nd, 4th, 6th and 8th digits and double them
Step 4.	8+3+7+0=18	Sum the digits from Step 3.
Step 5.	22+18 =40	Sum the results of Step 1 and Step 2.
Step 6.	50–40 =10	Subtract the Step 5 result from the next highest multiple of ten. If the difference is 10, the result becomes zero.

If the SIN is valid, the check digit and the number determined at the end of Step 6 will be the same.

Once more using 460 050 461

Step 1.	460050461
Step 2.	4+0+5+4=13
Step 3.	6 0 0 6 x 2 12 0 1 2
Step 4.	1+2+0+1+2=6
Step 5.	6+13 =19
Step 6.	20–19 = 1

Again, the check digit matches and we have a valid SIN.

Using Manager Math we would code:

```
1 TO R90
(N1+1) / 10↑8 TO R91                ;field one is the sin field
WHILE R90 < 10 DO                   ;first we split the
      R91–.5 TO 0 R(R90)            ;sin into
      (R91–R(R90))*10 TO R91        ;individual digits
      R90+1 TO R90
ENDWHILE

R2*2000 + R4*200 + R6*20 + R8*2 TO R92 ;double 2nd, 4th, 6th and 8th, and sum them

11 TO R90
(R92+1) / 10↑4 TO R91

WHILE R90 < 16 DO                   ;split sum into
      R91–.5 TO 0 R(R90)            ;individual digits
      (R91–R(R90)) * 10 TO R91
      R90+1 TO R90
ENDWHILE

R1+R3+R5+R7+R11+R12+R13+R14+R15 TO R93

R3/10 – .5 TO 0 R94                 ;determine the difference
(R94+1) * 10 – R93 TO R95           ;between the sum and
                                    ;the next highest multiple of 10
IF R95=10 THEN 0 TO R95  ENDIF      ;if the difference is ten
                                    ;set difference to zero
IF R95=R9 THEN                      ;if result is equal to check digit
    'VALID SI NUMBER'   TO D1       ;print 'valid' at d1
ELSE                                ;or else
    'INVALID SI NUMBER' TO D1       ;print 'invalid' at d1
ENDIF
```

D1 is a display position that must be set up in advance. It would then be up to the operator to spot the message and change the SIN if necessary.

Hardware Corner

Domenic DeFrancesco
Chris Zamara
Downsview, Ont.

In the last Hardware Corner, we explained what the user port on your PET or 64 is, and gave a brief example of how to use it from BASIC. In this article we will actually connect a simple circuit to the port – a row of 8 LED's (Light Emitting Diodes). Once this circuit is built, you will be able to program to your heart's content, controlling the LEDs in a variety of ways, while learning the fundamentals of digital binary devices.

A Few Notes of Warning

You must remember that assembling these circuits is not like programming. A mistake in a program might mean at worst resetting the computer, but a mistake in assembling hardware could mean an expensive service bill. If you follow the few simple rules outlined below you should have no problem.

1) Always double check your circuit, (especially the power connections to the ICs), before applying the power. I know that sometimes you're so anxious to see if the circuit works that you don't want to bother checking it, but remember a mistake could be fatal to your computer AND pocket book.

2) Never apply more than five volts to the circuit. The ICs in the computer and the ICs that we will be using in our projects can only tolerate a voltage between 0 and 5 volts on their input pins. Any voltage outside this range will permanently damage your electronic components. C64 and VIC 20 users should be aware of the 12 volts AC pins on the User Port. Accidently connecting one of these pins to an IC will surely damage your circuit.

3) Never connect two output pins together. If two output pins are connected together for a long time, excessive currents will flow in the IC and cause damage. Connecting an output to GND or +5 volts will also cause damage.

4) Always turn your computer off when connecting a circuit. If you forget to do this once, and plug a circuit into a live computer, odds are that nothing will happen. However, the risk is there, and if you do it often, one of these times you'll fry something.

If you turn on your computer and it does not power up within the normal amount of time, immediately turn off your computer and check your circuit.

The LED Circuit: Theory Of Operation

Our goal is to connect 8 LEDs to the 8 data lines of the parallel port. The lines from the computer, however, can supply only 1.6 milliamps, which is not enough current to directly drive LEDs. To control the LEDs from the computer, we must use a "buffer" IC between the parallel port lines and the LEDs. The chip we are using, the 74LS240, actually contains 8 inverting buffers, and each buffer's output is capable of supplying up to 24 milliamps. Buffers are available in inverting and noninverting configurations, and an inverting buffer is so called because its output is the inverse of its input. This means that a logic level 0 (zero volts) applied to the input will result in a logic level 1 (+5 volts) on the output, and vice versa. These outputs drive the LEDs, which are connected in series with current limiting resistors. The resistors prevent damage to the LEDs by limiting current to a safe level of about 8 milliamps. The output of each buffer is connected to the cathode (negative side) of each LED. (The cathode is the shorter of the two ends of the LED) Thus, when a high voltage (logic level one) is present at the buffer input, the resultant low voltage on the output of the buffer will turn the LED on. See the schematic diagram in figure 2 to see how the connections are made.

What You Will Need

The only parts you will need for the circuit are listed below. We suggest using a "breadboard" for mounting the components, so that you can easily modify the circuit, or take it apart and re-use the parts for future projects.

Parts Required

Quantity	Description
1	74LS240 Octal inverting buffer
8	Light Emiting Diodes
8	150 ohm resistors, 1/4 watt

It is also recommend that a cable be made up with an edge card connector on one end, (to plug into the computer), and a 24 pin dip connector on the other end, (to plug into the breadboard). You should be able to find the above parts at any electronics supply house.

The User Port to Breadboard cable

For this project and the ones that follow, we will be using the same type of cable to connect the user port and the circuit. This cable will have an edge-card connector on one end, to plug into the user port, and a "24 pin DIP header" which plugs into the breadboard. The cable can be easily unplugged from the board for use in other projects, and can be used to connect virtually any circuit to the user port. To make the cable, you need the following items:

about 1 foot of 24 conductor ribbon cable
24 pin 0.156-inch spacing edge-card connector, with headshell
24 pin DIP header

To make the cable, first separate the individual wires on one end so that there is about 5 cm. of free wire. Strip the ends of the wires, and tin the bare ends with solder. Solder the wires to the edge connector, following the connection diagram in figure 3. It is a good idea to put a headshell over the connector, to maintain the integrity of the solder joints. The 24 pin DIP header goes on the other end of the cable, as in the diagram. Use pressure from a vice to securely clamp the two pieces of the header together, sandwiching the ribbon cable between them.

Beginning Construction

Once you have all the parts you need, plug the cable header connector, the IC, the LEDs, and the resistors into the breadboard as in figure 1. Now connect the wires, following figure 1 and the schematic diagram in figure 2. Photo 1 shows the completed circuit, including the cable. To make the connections, #22 gauge solid wire is recommended (this is the normal telephone-type wire, which is available everywhere in abundant supply). Connecting the circuit for use on the PET/CBM is a bit more complicated, since +5 volts is not available on the user port. You can obtain +5 volts from pin 'B' or pin '2' on the cassette port, using a 0.156 inch, 6 contact connector.

Plugging it in

Once the circuit is built and the cable is wired and plugged in, you are ready to plug in to the user port. With the computer off, plug the large flat connector into the user port, making sure that the correct side is facing up. Turn on the computer. All the LEDs should go on, and the computer should power up normally. If the computer does not power up within the normal amount of time, immediately turn it off and re-check the wiring of the LED circuit.

If everything goes OK so far, you can test out the circuit. First, recall some of the theory about the parallel port from the last article. The parallel port has two memory locations associated with it, the data direction register, and the data register. These locations are as follows:

	PET/CBM	VIC	C64
Data Direction: DDR=	59459	37138	56579
Data Register : DR=	59471	37136	56577

The data direction register controls which of the lines on the parallel port are inputs and which are outputs, and the data register allows you to control the state of the outputs, and read the state of the inputs. That's how it works, in a nutshell - for examples, refer to last article.

With this information under our belt, we should be able to test the circuit: for starters, make all of the LEDs go off. When controlling the LEDs in any way via software, all of the lines on the parallel port must be set as outputs. We do this by setting all bits in the data direction register to ones. In BASIC,

```
POKE DDR, 255
```

. . .will accomplish this, assuming the variable 'DDR' has been set to the appropriate value from the table above.

As soon as the data direction register is set to make all the lines outputs, all of the LEDs should go off. This is because when you turn on the computer, the I/O chip resets with zeroes in all of its registers. This makes all lines low, turning all the LEDs off. If they do in fact turn off, the circuit is probably working properly.

Controlling the LEDs

Once the data direction register is set as above, you can control the LEDs in any of the 256 possible off/on combinations. Set the variable "DR" as indicated in the table, then try this:

```
POKE DR, 1
```

What happens? LED #0 should turn on. Now try:

```
POKE DR, 2
```

. . .and LED #1 should turn on. Now guess what will happen if you POKE DR,3. Enter the POKE and see what happens. If you were right, congratulations. You understand the binary number system. If you expected LED #2 to turn on, here's what is happening. Recall the binary number theory from the last installment of this article: Each bit in the data register controls the corresponding LED on the breadboard, i.e if bit 0 is one, LED 0 is on. Bit 1 controls LED 1 in a like manner, etc. Thus, to turn on LED #2, bit #2 must be set to one. To set any given bit to a one, that bit's value must be added to the number being stored in the data register. The value of each bit can be determined by the following equation:

```
bit value = 2 ↑ (bit number)
```

Using the above equation, we could set bit #2 to a one (turning on LED #2), with:

```
POKE DR, 2↑2
```

If we wanted to turn on LEDs 0, 3, and 4, for example, we could add all the bit values as follows:

```
POKE DR, 2↑0 + 2↑3 + 2↑4
```

Using this simple equation, you can easily control the LEDs from BASIC. We have concocted three short programs (listings 1-3) to create some interesting effects. Try typing the programs in, and look at the code to see how they do what they do. Once you get the idea, try some of your own programs. Becoming comfortable with bit control in this manner is an important first step in understanding the nature of binary devices, and the knowledge will help when controlling any other devices you might want to connect to the parallel port. Meanwhile, you may be able to actually find some practical applications for the LED circuit, for example using the LEDs as status indicators when de-bugging a complicated program.

In the next issue, we'll use the parallel port to read pushbutton switches, and take a look at how to read a keyboard matrix.

Listing 1

A demonstration program for the LED circuit, with it's 'quick and dirty' two line equivalent below. Note the exponential formulas in lines 210-230 and see how they correspond to the patterns on the LEDs.

```
100 rem " Dom's idea and Chris's code **
110 rem " for hardware corner LED cct **
115 :
120 ddr = 56579: rem* data direction reg.
130 dr  = 56577: rem* data register
140 rem* (above values are for c64) *
150 :
160 poke ddr,255:rem set to all outputs
170 :
180 for loop = 0 to 1 step 0
190 :for i = 1 to 4
200 :  for j = 0 to 7
210 :  if i = 1 or i = 3 then p = 2↑j + 2↑(7-j)
220 :  if i = 2 then p = 2↑(7-j)
230 :  if i = 4 then p = 2↑j
240 :  poke dr,p
250 next j,i,loop
```

```
1 poke56579,255:fork = 0to1e30:fori = 1to4
  :forj = 0to7:l = 2↑(7-j):r = 2↑j
2 poke56577,-(i = 2)*l-(i = 4)*r-(i = 1ori = 3)*(l + r):nextj,i,k
```

Listing 2: Knight Rider

The program uses DATA statements to supply a sequence of bit-pattern information. The data could have been calculated by the program, but in this case using DATA statements proved to be a more practical approach.

```
100 rem " Knight Rider "        **
110 rem " for H.C.#2 LED board  **
120 :
130 ddr = 56579: rem* data direction reg.
140 dr  = 56577: rem* data register
150 rem* above values for c64 *
160 :
170 dim d(14)
180 for i = 0 to 14: read d(i): next i
190 :
200 speed = 60: rem* lower value = faster *
210 :
220 for loop = 0 to 1 step 0
230 :for i = 0 to 14
240 :  poke dr,d(i)
250 :  for delay = 1 to speed: next delay
260 next i,loop
270 :
280 :
290 data  1, 3,12,24,48,96,192,128
300 data 192,96,48,24,12, 6,  3,  1
```

```
1 poke56579,255:fori = 0to14:reada:poke56577,a
  :ford = 1to60:nextd,i:restore:goto1
2 data1,3,12,24,48,96,192,128,192,96,48,24,12,6,3,1
```

Listing 3: Roulette Wheel

This program, followed by it's short version, simulate the spinning of a roulette wheel. Pressing SPACE starts the "spin" and eventually one LED will remain lit. The code was written avoiding the use of GOTOs so that it may be placed anywhere within a larger program. The COS function in line 350 is used because it seems to approximate the rate at which a roulette wheel slows down. This is purely empirical construction and does not follow any real physical or mathematical laws for a spinning roulette wheel.

```
100 rem* roullete wheel program for **
110 rem* led circuit            **
120 rem* "(written without using GOTOs)
130 :
140 ddr = 56579: rem* data direction reg.
150 dr  = 56577: rem* data register
160 rem "  (above values are for C64:
170 rem "   see text for PET/VIC values)
180 :
190 poke ddr,255: rem* 255 = all outputs
200 :
210 dim e(7)
220 rem* array e() holds bit values *
230 for i = 0to7:e(i) = 2↑(7-i): next i
240 :
250 print " Spress SPACE bar to 'spin'. "
255 :
260 for k = 0 to 1 step 0: rem* loop *
270 fori = 0to1:geta$:i = -(a$ = " "):next i
280 n = int(rnd(1)*70): print n;
290 :
300 rem* start off at next unlit led *
310 led = led and 7
320 :
325 rem* main 'spin' loop *
330 for i = 0 to 1
340 : poke dr,e(led and 7)
350 : inc = (cos(led/n*π) + 1)/2
360 : led = led + inc
370 : i = -(led> = n or inc<0.04)
380 next i
390 :
400 print " rspin ended "
410 next k
```

```
1 poke56579,255:fori = 0to7:e(i) = 2↑(7-i):next:fork = 0to1e30
2 fori = 0to1:geta$:i = -(a$ = " "):next
  :n = int(rnd(1)*70):printn:l = land7
3 fori = 0to1:poke56577,e(land7):a = (cos(l/n*π) + 1)/2
  :l = l + a:i = -(l>nora<.04):nexti,k
```

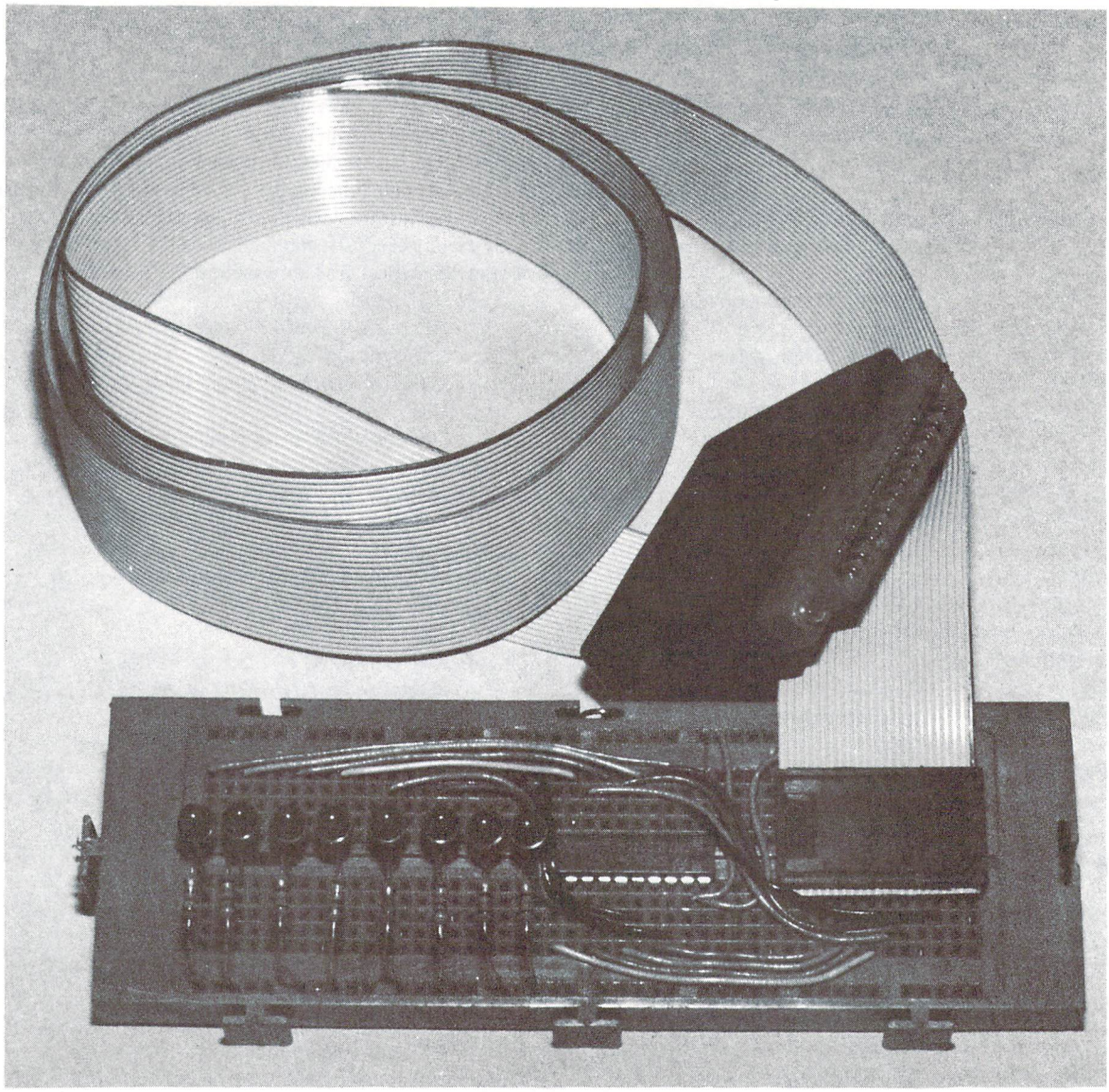


Photo by Ron Ing

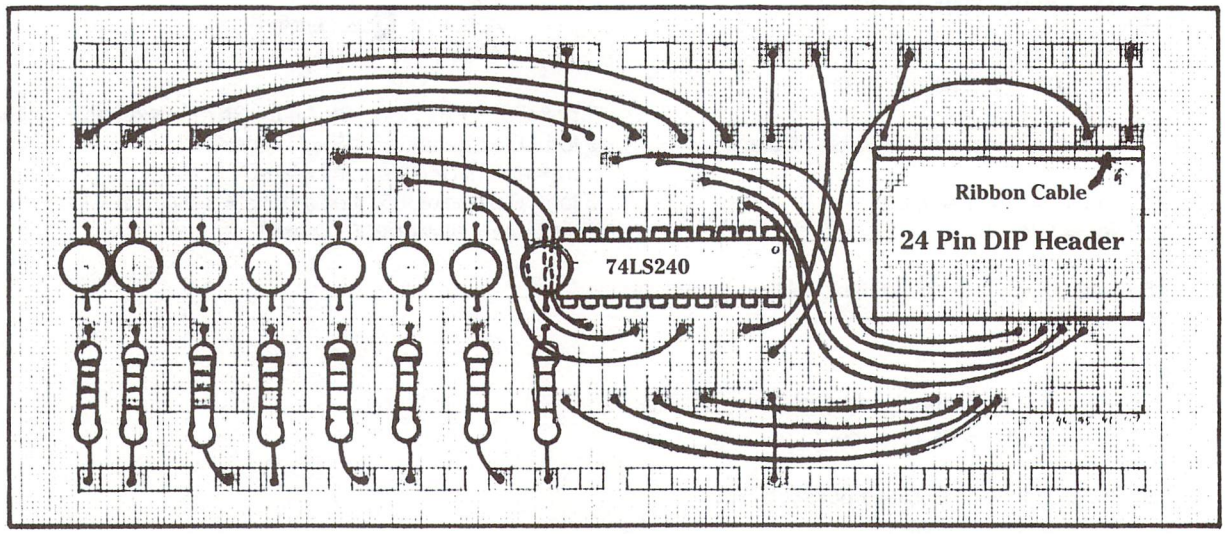


Figure 1

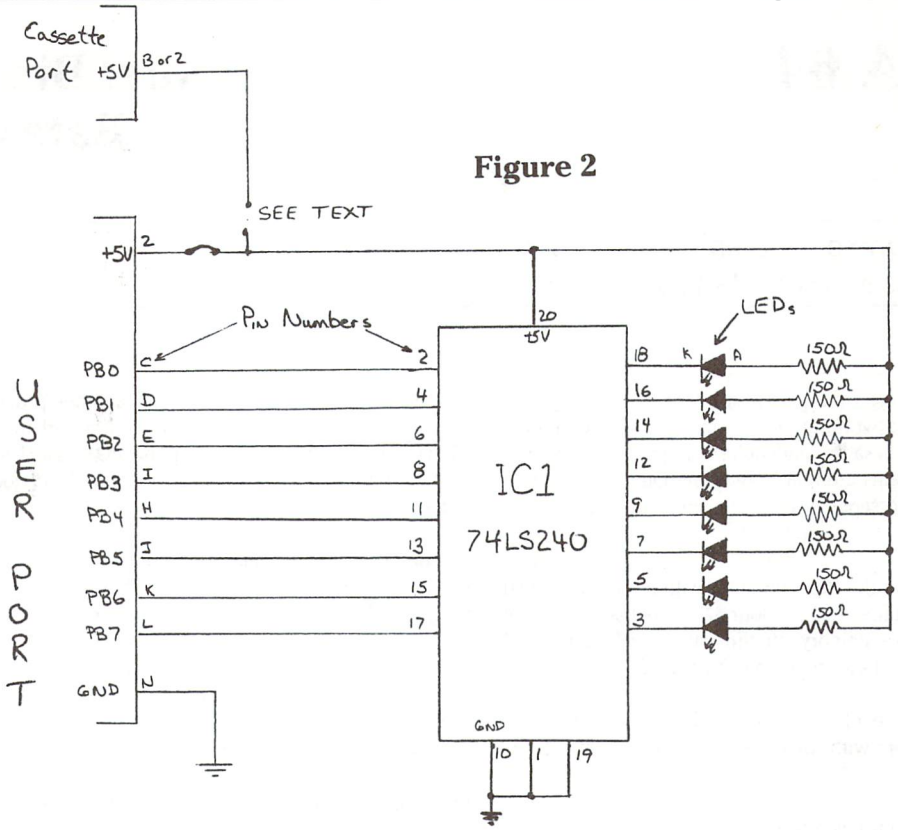


Figure 2

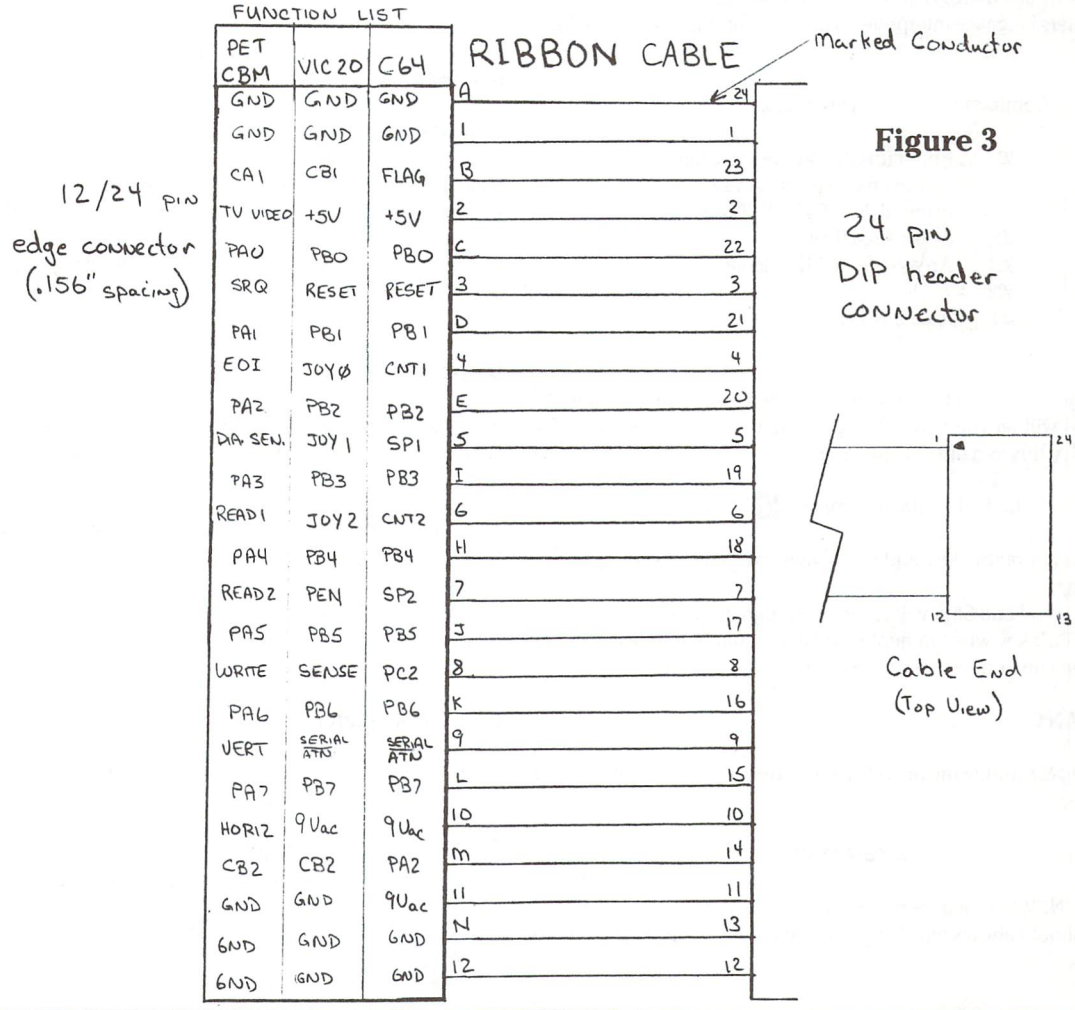


Figure 3

QUADRA 64

Daniel Bingamon
Batavia, OH

Edit 4 Programs On Your Commodore 64 Simultaneously With This Fabulous Memory Partitioning Utility

Memory gets bigger and bigger but people still write small programs. Occasionally you might be working on a program when something comes up. Then you have to save what you're doing and load another program up. If your programs are relatively small, Quadra 64 will let you load both of them into separate areas. It's also handy for testing small subroutines.

This idea was once used on the PET computer with a program called Quadra-Pet which divided memory into four 8K segments. Quadra 64 divides the Commodore 64's memory into three almost 8K segments (3 bytes short of 8K) and one 14K segment (partition no. 3)

Each partition has its own set of variables, 100% localized. No part of either program will interfere with the other, unless you do it deliberately.

This version of this program has a wedge added. Those of you who don't know what a wedge is, it is a process of adding extra commands to the computers language interpreter. The extra commands are provided below:

Command	Description
£	Print current partition number and re-enable pause keys after using RUN/STOP RESTORE
£0	Select Area 0, etc.
£1	Areas 0-2 are 8K in length.
£2	
£3	Area 3 is 14K.

Also:
Pressing the Control and Commodore keys will cause the computer to pause. This will let you halt LISTings, or pause during execution of a program. Try this as a direct command:

```
for j=1 to 10000 : print j;" Q " : next
```

Press Control/Commodore and notice how execution halts even without scrolling.
Pressing Control and Shift will re-enable program from pause.
Pressing STOP key while in quotes will allow computer to leave quote mode when cursor movements are used.

IMPORTANT

After saving & running the program, the correct way to start up Quadra 64 is to type:

SYS 49152

Then type 'NEW' to clear memory. To use each partition for the FIRST time you should always type 'NEW' or use the Initialize program to do that for you.

The best way for a program to call another program in a different partition is to print the wedge command and the needed 'RUN' or 'GOTO' statement on the screen and POKE carriage returns into the keyboard buffer to execute those commands. (a good example is the initialize program.)

Remember to SAVE what you type before running, one mistake can cause a terrible crash and you will have to retype everything. The program will notify you if there is an error in your DATA statements.

Two or more programs can obviously be stored in memory at the same time. It may be possible to have two BASIC programs running at the same time using the keyboard interrupt. One would have to switch program pointers and take snapshots at zero-page and develop a program to link in the background program to run. I once wrote a hi-res graphics program to draw with a joystick while BASIC was running something else using the keyboard interrupt. Foreground and Background ARE possible on the 64, is it worth the attempt?

Special thanks to Jim Butterfield for the memory maps that made this possible.

Editor's Note:

Since Daniel sent us this program, it's become the first thing I load to start work, especially when I'm collecting articles together for the next magazine. I load the drive 0 directory into area 0, the drive 1 directory into area 1 (naturally), and any programs usually go into area 3, the big one - and I still have area 2 for any extra stuff I may need later. It's perfect!

Initialize Program

After running the Quadra 64 loader, you'll need to type NEW in every area, or just run this program – it will do it for you.

```
5 cd$=chr$(17)
10 printchr$(147)cd$cd$cd$"£1"cd$cd$:print"new"
20 printcd$cd$"£2"cd$cd$:print"new"
30 printcd$cd$"£3"cd$cd$:print"new"
35 printcd$cd$"£0"chr$(19)
40 poke198,7:fori=631to637:pokei,13:next
```

Quadra 64 Loader

```
1000 printchr$(147):print"quadra 64 w/wedge"
1010 print:print" commands:"
1020 print:print" £     display partition number."
1030 print" £0-3  select partition number"
1040 print:print" run the initialize program to clean"
1045 print" memory partitions."
1046 print:print"now loading wedge. . .."
1047 rt$=chr$(145):for j=1to20:rt$=rt$+chr$(29):next
1050 fori=49152to49694
```

```
1060 reada:pokei,a:ck=ck+a:printrt$;i
1070 next:print
1080 if ck=58348thenprint" no errors. sys49152 to start. ":end
1090 print" there is a error in your data, do"
1100 print" not attempt to run this before"
1110 print" correcting errors."
1120 poke 49152,96:rem poke in rts to prevent accidental running
1130 stop
1140 data 169, 192, 160, 121, 141,   9,   3, 140,   8
1150 data   3, 169, 193, 160,  82, 120, 141,  21,   3
1160 data 140,  20,   3,  88, 160,   0, 140,   0,   8
1170 data 140,   0,  40, 140,   0,  72, 140,   0, 104
1180 data 169,   0, 133,  53, 169,  28, 133,  54, 169
1190 data   0, 141, 159, 192, 169,  67, 160, 192,  32
1200 data  30, 171, 169,   0, 141,  30, 194,  32, 112
1210 data 193, 234, 234,  96, 147,  81,  85,  65,  68
1220 data  82,  65,  32,  54,  52,  13,  13,  66,  89
1230 data  58,  32,  68,  65,  78,  73,  69,  76,  32
1240 data  66,  73,  78,  71,  65,  77,  79,  78,  13
1250 data  70,  79,  82,  32,  84,  72,  69,  32,  84
1260 data  82,  65,  78,  83,  65,  67,  84,  79,  82
1270 data  13,   0,   0,   0,  32, 115,   0, 201,  92
1280 data 240,  16, 201,  58, 240, 245,  76, 231, 167
1290 data  76, 116, 164,   0,   0,   0,   0,   0,   0
1300 data  72,  32, 229, 193, 104,  32, 115,   0, 201
1310 data   0, 208,  10,  76,  59, 193,   3,   0,   0
1320 data   0,  76,  23, 193, 201,  52, 176, 242,  72
1330 data  41, 207, 141, 159, 192, 104, 201,  48, 240
1340 data  14, 201,  49, 240,  13, 201,  50, 240,  12
1350 data 201,  51, 240,  11, 208, 217,  76, 112, 193
1360 data  76, 151, 193,  76, 177, 193,  76, 203, 193
1370 data   1,   8,   1,   8,   1,   8,   0,  40,   0
1380 data  40,   0,  40,   0,  40,   0,  40,   0,   0
1390 data   1,  40,   1,  40,   1,  40,   0,  72,   0
1400 data  72,   0,  72,   0,  72,   0,  72,   0,   0
1410 data   1,  72,   1,  72,   1,  72,   0, 104,   0
1420 data 104,   0, 104,   0, 104,   0, 104,   0,   0
1430 data   1, 104,   1, 104,   1, 104,   0, 160,   0
1440 data 160,   0, 160,   0, 160,   0, 160,   0,   0
1450 data 201,  75, 208,  29, 169, 167, 141,   9,   3
1460 data 169, 228, 141,   8,   3, 169,   1, 133,  43
1470 data 169,   8, 133,  44, 169,   0, 133,  53, 169
1480 data 160, 133,  54,  76, 135, 192,  76,   8, 175
1490 data 169, 193, 160,  82, 120, 141,  21,   3, 140
1500 data  20,   3,  88, 174, 159, 192, 169,   0,  32
1510 data 205, 189,  76, 135, 192, 165, 197, 201,  63
1520 data 208,   4, 169,   0, 133, 212, 173, 141,   2
1530 data 201,   6, 208,  10,  32, 159, 255, 173, 141
1540 data   2, 201,   5, 208, 246,  76,  49, 234, 162
1550 data   0, 189, 207, 192, 149,  43, 232, 224,  15
1560 data 208, 246, 165,  44, 133, 252, 169,   0, 133
1570 data 251, 168, 145, 251, 173,  30, 194, 201,   0
1580 data 208,   6, 169,   1, 141,  30, 194,  96,  76
1590 data 135, 192, 162,   0, 189, 225, 192, 149,  43
1600 data 232, 224,  15, 208, 246, 165,  44, 133, 252
1610 data 169,   0, 133, 251, 168, 145, 251,  76, 135
1620 data 192, 162,   0, 189, 243, 192, 149,  43, 232
1630 data 224,  15, 208, 246, 165,  44, 133, 252, 169
1640 data   0, 133, 251, 168, 145, 251,  76, 135, 192
1650 data 162,   0, 189,   5, 193, 149,  43, 232, 224
1660 data  15, 208, 246, 165,  44, 133, 252, 169,   0
1670 data 133, 251, 168, 145, 251,  76, 135, 192, 162
1680 data   0, 173, 159, 192, 201,   0, 240,  16, 201
1690 data   1, 240,  20, 201,   2, 240,  24, 208,  30
1700 data 232, 224,  15, 208, 234,  96, 181,  43, 157
1710 data 207, 192,  76, 248, 193, 181,  43, 157, 225
1720 data 192,  76, 248, 193, 181,  43, 157, 243, 192
1730 data  76, 248, 193, 181,  43, 157,   5, 193,  76
1740 data 248, 193,   0
```

```
;* quadra 64, by: daniel bingamon, may 14, 1984
;
;       label definitions
;
*       =   $c000
chrget  =   $73
chrgot  =   $79
ptr     =   $7a
intger  =   $14
temp    =   $fb
eval    =   $ad9e
convrt  =   $b7f7
print   =   $ab1e
dispch  =   $0308
intrpt  =   $0314
basic   =   $002b
memsiz  =   basic+10
tmpadd  =   $14
fst     =   $0800
snd     =   $2800
trd     =   $4800
fth     =   $6800
lmt     =   $a000
;
;       initialization
;
init    lda  #>wedge      ;init wedge
        ldy  #<wedge
        sta  dispch+1
        sty  dispch
        lda  #>intprg     ;enable interupt
        ldy  #<intprg
        sei
        sta  intrpt+1
        sty  intrpt
        cli
        ldy  #00          ;zero first byte
        sty  fst          ;of partitions.
        sty  snd
        sty  trd
        sty  fth
        lda  #00
        sta  memsiz
        lda  #28
        sta  memsiz+1
        lda  #0           ;startup in
        sta  partit       ;partition 0.
        lda  #<msg        ;print title.
        ldy  #>msg
        jsr  print
        lda  #00
        sta  onetim
        jsr  pr1
        nop
        nop
        rts
msg     .byte $93,'quadra 64',$0d
        .byte $0d,'by: daniel bingamon'
        .byte $0d,'for the transactor'
        .byte $0d,$00,$00,$00
wedge   jsr  chrget
        cmp  #'£          ;if pound sign
        beq  parse        ;process wedge.
        cmp  #':
        beq  wedge
exit1a  jmp  $a7e7
exit    jmp  $a474
tapspc  .byte $00,$00,$00,$00,$00,$00
;
;       process wedge command
;
parse   pha
        jsr  savmem       ;store old part. data
        pla
        jsr  chrget
        cmp  #00
        bne  contin
irqint  jmp  intchk
partit  *=*+1
        .byte $00,$00,$00
erroc   jmp  errchk
contin  cmp  #'4
        bcs  irqint
        pha
        and  #$cf
        sta  partit
        pla               ;jump to individual
        cmp  #'0          ;partition routines
        beq  one
        cmp  #'1
        beq  two
        cmp  #'2
        beq  three
        cmp  #'3
        beq  four
        bne  irqint
one     jmp  pr1
two     jmp  pr2
three   jmp  pr3
four    jmp  pr4
;
;       partition select table
;
bnk1    .word $0801,$0801,$0801,$2800,$2800,$2800,$2800,$2800,$0000
bnk2    .word $2801,$2801,$2801,$4800,$4800,$4800,$4800,$4800,$0000
bnk3    .word $4801,$4801,$4801,$6800,$6800,$6800,$6800,$6800,$0000
bnk4    .word $6801,$6801,$6801,$a000,$a000,$a000,$a000,$a000,$0000
errchk  cmp  #'k
        bne  error
        lda  #$a7
        sta  dispch+1
        lda  #$e4
        sta  dispch
        lda  #$01
        sta  basic
        lda  #$08
        sta  basic+1
        lda  #$00
        sta  memsiz
        lda  #$a0
        sta  memsiz+1
        jmp  exit
error   jmp  $af08        ;syntax error
intchk  lda  #>intprg
        ldy  #<intprg
        sei
        sta  intrpt+1
        sty  intrpt
        cli
        ldx  partit       ;print part. no.
        lda  #00
        jsr  $bdcd        ;print integer routine
        jmp  exit
intprg  lda  $c5
        cmp  #63
        bne  skp          ;escape quote mode
        lda  #0           ;when stop key
        sta  $d4          ;depressed.
skp     lda  $028d        ;enable freezing
        cmp  #06          ;with ctrl & comm.
        bne  skp2         ;keys.
cloop   jsr  $ff9f        ;ctrl & shift to
        lda  $028d        ;resume operation.
        cmp  #05
        bne  cloop
skp2    jmp  $ea31
;
pr1     ldx  #00          ;partition 1.
lp1     lda  bnk1,x
        sta  basic,x
        inx
        cpx  #$0f
        bne  lp1
        lda  basic+1
        sta  $fc
        lda  #00
        sta  $fb
        tay
        sta  (temp),y
        lda  onetim
        cmp  #00
        bne  no
        lda  #01
        sta  onetim
        rts
no      jmp  exit
pr2     ldx  #00          ;partition 2.
lp2     lda  bnk2,x
        sta  basic,x
        inx
        cpx  #$0f
        bne  lp2
        lda  basic+1
        sta  $fc
        lda  #00
        sta  $fb
        tay
        sta  (temp),y
        jmp  exit
pr3     ldx  #00          ;partition 3.
lp3     lda  bnk3,x
        sta  basic,x
        inx
        cpx  #$0f
        bne  lp3
        lda  basic+1
        sta  $fc
        lda  #00
        sta  $fb
        tay
        sta  (temp),y
        jmp  exit
pr4     ldx  #00          ;partition 4.
lp4     lda  bnk4,x
        sta  basic,x
        inx
        cpx  #$0f
        bne  lp4
        lda  basic+1
        sta  $fc
        lda  #00
        sta  $fb
        tay
        sta  (temp),y
        jmp  exit
savmemldx  #00            ;save old pointers
savlp2  lda  partit
        cmp  #00
        beq  sm1
        cmp  #01
        beq  sm2
        cmp  #02
        beq  sm3
        bne  sm4
savlp1  inx
        cpx  #$0f
        bne  savlp2
        rts
sm1     lda  basic,x      ;load pointer from
        sta  bnk1,x       ;each partition.
        jmp  savlp1
sm2     lda  basic,x
        sta  bnk2,x
        jmp  savlp1
sm3     lda  basic,x
        sta  bnk3,x
        jmp  savlp1
sm4     lda  basic,x
        sta  bnk4,x
        jmp  savlp1
onetim  .byte $00         ;first time indicator.
        .end
```

Your BASIC Monitor Part 2: The Disassembler

Bob Drake
Brantford, Ont.

A dissassembler is a fairly simple program. It uses mainly brute force to do its job. A location is peeked. The value there determines an entry in a table of values. The table entry is the mnemonic (new-mon-ick) code for the operation. Mnemonics are the abbreviations for machine language operations. They are also called *op codes* or operation codes.

The second part of the chore is to determine how many more bytes are required by the mnemonic. Implied operations such as BRK need no more. Immediate, zero page and relative operations need one more byte. Absolute operations need two more bytes.

The third part of the operation is to write the whole thing down in an acceptable form. Using 6502 standards for assemblers this means including a # sign for immediate mode, $ on hex addresses, brackets and ,X or ,Y on the various indexed modes. Again this is primarily a brute force job.

Before adding the disassembler to your monitor, you had better fix a bug which crept into part 1.

```
7510 print
7520 input " number " ;n$
```

That's it.

The added code for the disassembler is listed here. Line 125 initializes all the data at lines 8000 and on. This is without a doubt the worst part of creating a disassembler or assembler. The data are listed for all the accepted 6502 op codes. I have mixed methods on the data. Those codes with many addressing methods are listed in OP$(1) to OP$(21) and the mnemonics are in one string MN$(1). OP$(22) has all the codes using implied addressing, OP$(23) has all the branches with relative addressing and OP$(24) has the jumps. If an unknown value is located, three question marks are printed. The simplest table would have had 255 entries. Peeking at a memory value would have located the mnemonic. But there is an assembler coming in part 3. And we needed to know the addressing mode. So, the program trades a slight clumsiness for an easier solution.

The only tricky part is the calculations required for the branches. The branch or conditional GOTO's (BCC, BMI, BPL etc.) all use a second byte to create a jump forward or backward. The jump is calculated from the beginning of the NEXT instruction. If the second byte is 128 ($80) or less, the jump is forward that amount. So, just add that value to the location of the next instruction to get the destination. If the second byte is bigger than 128 ($80) then the jump is BACKWARDS by an amount of 255 minus the value. This is accomplished in lines 9580-9590.

A couple of quick notes. Since the number of needed bytes isn't known until the first byte is decoded, the program automatically takes in the maximum of three bytes each time. An address is then constructed as:

```
ADDR$ = " $ " + HI$ + LO$
```

This saves considerable work in printing the disassembled code. As well, the low or second byte is readily available for the instructions requiring it alone.

We'll add the assembler next time!

```
125 gosub 8000
220 r$ = " xmrpslg*cd "
350 on r gosub 0,1000,2000,3000,4000,4140,
    5000,6000,7500,9000
8000 rem data for assembler/disassembler
8010 rem imm/zer/zer-x/zer-y/abs/abs-x/
    [10 spaces]abs-y/ind x/ind y/acc
8020 dim mn$(4),op$(24)
8030 mn$(1) = " adcandaslbitcmpcpxcpydeceorincld
    aldxldylsrorarolrorsbcstastxsty
```

```
8040 op$(1) = "696575**6d7d796171**"
8050 op$(2) = "292535**2d3d392131**"
8060 op$(3) = "**0616**0e1e******0a"
8070 op$(4) = "**242c**********"
8080 op$(5) = "c9c5d5**cdddd9c1d1**"
8090 op$(6) = "e0e4****ec*********"
8100 op$(7) = "c0c4****cc*********"
8110 op$(8) = "**c6d6**cede********"
8120 op$(9) = "494555**4d5d594151**"
8130 op$(10) = "**e6f6**eefe********"
8140 op$(11) = "a9a5b5**adbdb9a1b1**"
8150 op$(12) = "a2a6**b6ae**be******"
8160 op$(13) = "a0a4b4**acbc********"
8170 op$(14) = "**4656**4e5e******4a"
8180 op$(15) = "090515**0d1d190111**"
8190 op$(16) = "**2636**2e3e******2a"
8200 op$(17) = "**6676**6e7e******6a"
8210 op$(18) = "e9e5f5**edfdf9e1f1**"
8220 op$(19) = "**8595**8d9d998191**"
8230 op$(20) = "**86**968e*********"
8240 op$(21) = "**8494**8c*********"
8250 rem implied
8260 mn$(2) = "brkclccldcliclvdexdeyinxinynop
     phaplaphpplprtirtssecsedseitaxtxatay"
8270 mn$(2) = mn$(2) + "tyatsxtxs"
8280 op$(22) = "0018d858b8ca88e8c8ea486808284060
     38f878aa8aa898ba9a"
8290 rem relative
8300 mn$(3) = "bccbcsbeqbnebmibplbvsbvc"
8310 op$(23) = "90b0f0d030107050"
8320 rem jumps
8330 mn$(4) = "jmpjmpjsr"
8340 op$(24) = "4c6c20"
8350 return
9000 rem disassembler
9010 print" r disassemble memory"
9020 print" hold r shift R to pause: r return R to stop"
9021 rem vic*replace line 9020 with lines 9022
9022 print" hold r shift R to pause    r return R to stop"
9030 gosub 4280
9040 if t<=f then t=f
9050 if f<0 or t<0 or f>65535 or t>65535 then 1260
9060 for m=f to t
9070 n=m
9080 gosub 7030:rem convert to hex
9090 for pr=1 to p
9100 print#pr, by$":";
9110 next pr
9120 t$=""
9130 by=peek(m+2):gosub7000:hi$=by$
9140 by=peek(m+1):gosub7000:lo$=by$
9150 addr$=" $"+hi$+lo$
9160 by=peek(m):gosub7000
9170 for i=1 to 24
9180 for j=1 to len(op$(i)) step 2
9190 if by$<>mid$(op$(i),j,2) then 9250
9200 c=i :rem which code
9210 f1=1
9220 po=(j+1)/2:rem position
9230 i=24 :rem exit the for-next nicely
9240 j=len(op$(i))
9250 next j,i
9260 if f1<>0 then 9300
9270 mn$="???"
9280 t$=by$+"      "
9290 goto 9630
9300 rem found value and position of op code
9310 f1=0
9320 mn=c-20:pp=po:if c<22 then mn=1:pp=c
9330 mn$=mid$(mn$(mn),(pp-1)*3+1,3)
9340 if c>21 then 9490
9350 if po=1 then mn$=mn$+" #$"+lo$
9360 if po=2 then mn$=mn$+" $"+lo$
9370 if po=3 then mn$=mn$+" $"+lo$+",x"
9380 if po=4 then mn$=mn$+" $"lo$+",y"
9390 if po=5 then mn$=mn$+addr$
9400 if po=6 then mn$=mn$+addr$+",x"
9410 if po=7 then mn$=mn$+addr$+",y"
9420 if po=8 then mn$=mn$+" ("+addr$+",x)"
9430 if po=9 then mn$=mn$+" ("+addr$+",y)"
9440 if m<>10 then m=m+1-1*(m>4)
9450 t$=by$+" "+lo$+" "+hi$+"  "
9460 if po<5 then t$=by$+" "+lo$+"    "
9470 if po=10 then t$=by$+"      "
9480 goto 9630
9490 if c<>24 then 9550
9500 m=m+2
9510 t$=by$+" "+lo$+" "+hi$+"  "
9520 if po=2then mn$=mn$+" ("+addr$+")"
9530 if po=1 or po=3 then mn$=mn$+addr$
9540 goto 9630
9550 if c=22 then t$=by$+"      ":goto 9630
9560 rem jumps
9570 t$=by$+" "+lo$+"    "
9580 if by>128 then by=by-255
9590 n=m+by+2
9600 gosub 7030
9610 mn$=mn$+" $"+by$
9620 m=m+1
9630 for pr=1 to p
9640 print#pr,t$"   "mn$:rem vic*print#pr,t$
     :print#pr,"    "mn$ :rem 5 spaces
9650 next pr
9660 if peek(653) then 9660:rem look for shift key
9670 get a$: if a$=cr$ then 9690:rem look for return key
9680 next m
9690 return
```

Picprint: A High-Resolution Screen Dump Utility

by Chris Zamara, Technical Editor

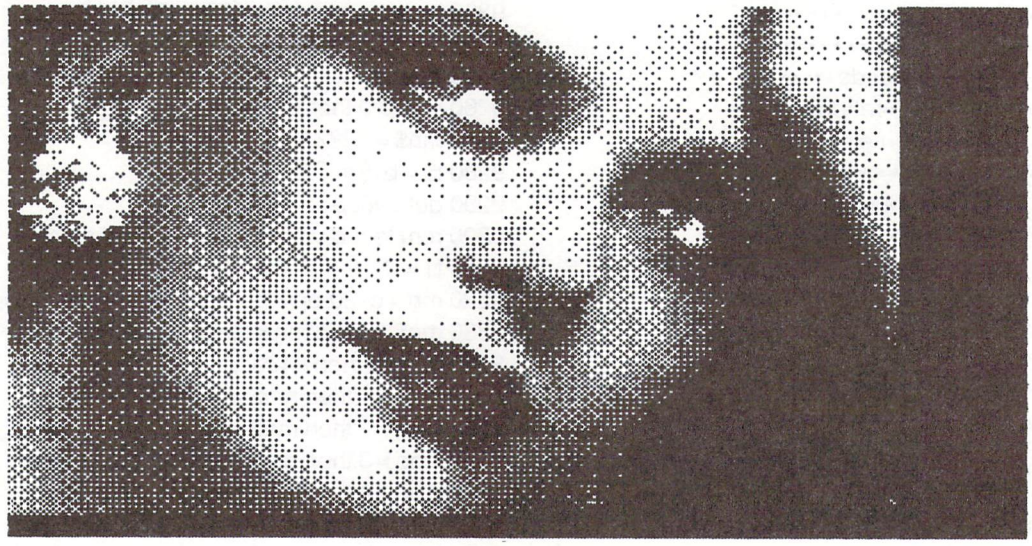


Photo 1

The Commodore 64's high resolution mode allows for some real eye-catching displays. High resolution pictures are available on disk, many which have been digitized from actual photographs. An example is the well-known photograph of "Karen" used to advertise Epson printers. Photo 1 shows Karen produced using Picprint and a Star Gemini-10X printer.

Picprint is an interrupt-driven program which allows viewing of bit map or normal video mode, and dumping the bit map screen in one of four formats: normal, normal reversed, wide, and wide reversed (reverse mode is necessary because some pictures are stored that way). The wide dump prints the picture so that it takes up the width of the page, and horizontally stretches it in the process. The function keys are used to select the above functions as follows:

F1: toggle text/bit map video mode
F3: normal hi-res dump
F5: wide hi-res dump
CTRL: when used in conjunction with F3 or F5, reverses the printed image (switches black to white and vice versa)

The version of Picprint as seen in Listing 2 is written specifically for the Gemini-10X printer, but should work unmodified with many other popular dot-matrix makes which have a high resolution graphics mode. The printer setup codes in lines 320-340 may have to be changed for printers requiring different control characters to enable graphics mode. If dual density mode is not available on your printer, just use the codes for normal mode. This will only affect "wide" picture dumps.

The control sequence necessary to change the linefeed size may also be different on other printers. The program sends this sequence in lines 1370-1420, which may be changed accordingly (change the arguments in the "lda #" instructions.). This listing sends data to the printer via a Cardco interface, which must be set to "graphics mode" so that it does not interpret any of the bit-map data as C64 graphics or control characters. This is done by selecting the secondary address as in line 1280, which may be set to zero (change to ldy #0) if not using a similar interface.

Once Picprint is initialized, the high resolution picture currently residing at location $2000 (hex) may be viewed at any time by pressing F1. Colour memory for the picture is supplied from text video memory, so the screen must be cleared (or filled with any one character) to get a clear view of the hi-res screen. Pressing F1 again returns to normal text mode. This switch may even be made while a picture is being loaded from disk, to see the screen gradually fill up with picture data. At any time, the current picture can be dumped using the function keys as outlined above.

A Couple of Usage Notes:

- To dump a picture residing in an area of memory other than $2000, change the definition in line 450. (The screen viewed using F1 will still be the one at $2000.).

- For some reason, Picprint will not dump to printer if POWER 64 is active in the system (it dumps to screen instead). Before a dump, POWER must be turned OFF, and may be re-activated with SYS 704 later.

Picprint lives at $c000, so you can keep it there safely, and enter SYS 49152 whenever you get the urge. This links Picprint with the system IRQ, and RESTORE will disable it again. For an easy way to enter and initialize Picprint, use the BASIC loader in Listing 1. Now you've got a perfect match: your computer and printer can make beautiful pictures together.

Listing 1: BASIC Loader

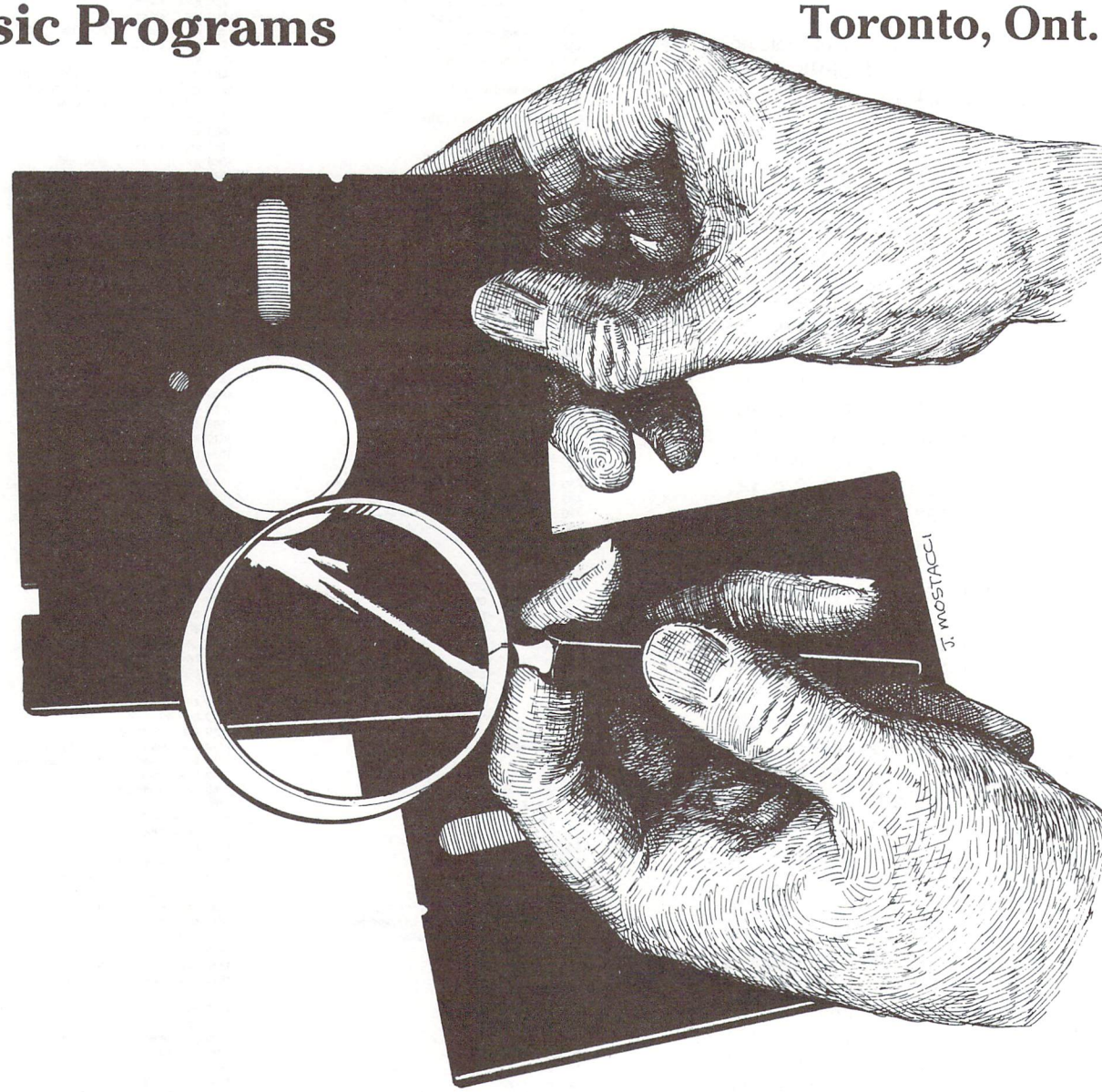
```
1000 rem picprint loader
1010 for j=49152 to 49498 : read x
1020 poke j,x : ch=ch+x : next
1030 if ch<> 41532  then print " data error "  : end
1040 sys 49152 : print  " PICPRINT Enabled "
1050 data  76,  34, 192,   0,   1,   1
1060 data 100, 128,  64,  32,  16,   8
1070 data   4,   2,   1,  27, 121, 192
1080 data   3,  27,  75,  64,   1,   0
1090 data   0,   0,   0,   0,   0,   0
1100 data   0,  24,   0,   0, 120, 169
1110 data  47, 141,  20,   3, 169, 192
1120 data 141,  21,   3,  88,  96, 165
1130 data 197, 201,  64, 208,   8, 169
1140 data   0, 141,   4, 192,  76,  49
1150 data 234, 173,   4, 192, 208,  35
1160 data 169,   1, 141,   4, 192, 165
1170 data 197, 201,   3, 240,  27, 201
1180 data   6, 208,   8, 169,   1, 141
1190 data   5, 192,  76, 123, 192, 201
1200 data   5, 208,   8, 169,   0, 141
1210 data   5, 192,  76, 123, 192,  76
1220 data  49, 234, 173,  17, 208,  73
1230 data  32, 141,  17, 208, 173,  24
1240 data 208,  73,   8, 141,  24, 208
1250 data  76,  49, 234, 169,   0, 141
1260 data   3, 192, 173, 141,   2,  41
1270 data   4, 240,   5, 169, 255, 141
1280 data   3, 192, 173,   6, 192,  32
1290 data 195, 255, 173,   6, 192, 162
1300 data   4, 160,   4,  32, 186, 255
1310 data 169,   0,  32, 189, 255,  32
1320 data 192, 255, 174,   6, 192,  32
1330 data 201, 255, 169,   0, 133, 251
1340 data 169,  32, 133, 252, 169,  27
1350 data  32, 210, 255, 169,  51,  32
1360 data 210, 255, 169,  16,  32, 210
1370 data 255, 169,  25, 141,  31, 192
1380 data 162,   0, 189,  15, 192, 172
1390 data   5, 192, 208,   3, 189,  19
1400 data 192,  32, 210, 255, 232, 224
1410 data   4, 208, 237, 169,  40, 141
1420 data  32, 192, 169,   0, 162,   7
1430 data 157,  23, 192, 202,  16, 250
1440 data 162,   0, 160,   0, 177, 251
1450 data 141,  33, 192, 230, 251, 208
1460 data   2, 230, 252, 173,  33, 192
1470 data  57,   7, 192, 240,   9, 185
1480 data  23, 192,  29,   7, 192, 153
1490 data  23, 192, 200, 192,   8, 208
1500 data 234, 232, 224,   8, 208, 216
1510 data 162,   0, 189,  23, 192,  77
1520 data   3, 192,  32, 210, 255, 172
1530 data   5, 192, 240,   6,  32, 210
1540 data 255,  32, 210, 255, 232, 224
1550 data   8, 208, 231, 206,  32, 192
1560 data 208, 172, 169,  13,  32, 210
1570 data 255, 165, 197, 201,  63, 240
1580 data   5, 206,  31, 192, 208, 130
1590 data 169,  27,  32, 210, 255, 169
1600 data  64,  32, 210, 255, 173,   6
1610 data 192,  32, 195, 255,  32, 204
1620 data 255,  76,  49, 234
```

Listing 2: Source Code

```
  50 sys700
 100 ;    " PICPRINT "
 110 ;hi-res dump utility
 120 ;use function keys f3,f5 and f7
 130 ;f7 -toggle hi-res mode
 140 ;f3 -dump picture
 150 ;f5 -dump wide picture
 160 ;ctrl -reverse dumped picture
 170 ;
 180 ;assembled on pal 64
 190 ;
 200 ;chris zamara - july 10/84
 210 ;
 220 .opt oo
 230 * = $c000
 240 jmp  init          ;bypass variables below
 250 ;
 260 eorflag   .byte 0
 270 keyflag   .byte 0
 280 bigflag   .byte 0
 290 filenum   .byte 100 ;file number used
 300 exp       .byte 128,64,32,16,8,4,2,1
 310 ;
 320 dualres   .byte 27,121,192,3
 330 ;control characters for printer
 340 ;dual density graphics mode
 350 ;
 360 normres   .byte 27,75,64,1
 370 ;normal density graphics mode
 380 ;(values may vary among printers)
 390 ;
 400 bits      * = * + 8
 410 rowcnt    * = * + 1
 420 colcnt    * = * + 1
 430 cbyte     * = * + 1
 440 ;
 450 screen    = $2000   ;hires screen memory
 460 scrnptr   = $fb     ;zero page pointer
 470 keybd     = 197     ;current key pressed
 480 ctrlflg   = 653     ;ctrl/shift flag
 490 normirq   = $ea31   ;normal irq entry
 500 ;
 510 ;kernal routines used:
 520 chkout    = $ffc9
 530 chrout    = $ffd2
 540 close     = $ffc3
 550 clrchn    = $ffcc
 560 open      = $ffc0
 570 setlfs    = $ffba
 575 setnam    = $ffbd
 580 ;
 590 ;
 600 ;
 610 init = *
 620 ;redirect interrupt vector
 630 sei
 640 lda  #<keychk
 650 sta  $0314
 660 lda  #>keychk
 670 sta  $0315
 680 cli
 690 rts
 700 ;
 710 ;
 720 keychk = *
 730 ;get function key presses
 740 lda  keybd        ;look for key pressed
 750 cmp #64           ;64 is no key
 760 bne  keydn
 770 lda  #0
 780 sta  keyflag
 790 jmp  normirq
 800 keydn = *
 810 lda  keyflag
 820 bne  out
 830 lda  #1
 840 sta  keyflag
 850 lda  keybd
 860 cmp #3            ;f7, switch video mode
 870 beq  flip
 880 cmp #6            ;f5, dump wide picture
 890 bne  f3chk
 900 lda  #1
 910 sta  bigflag
 920 jmp  dump
 930 f3chk = *
 940 cmp #5            ;f3, dump normal picture
 950 bne  out
 960 lda  #0
 970 sta  bigflag
 980 jmp  dump
 990 out = *
1000 jmp  normirq
1010 ;
1020 ;
1030 flip = *
1040 ;toggle bit map graphics mode
1050 lda  $d011
1060 eor  #$20
1070 sta  $d011       ;bit map mode
1080 lda  $d018
1090 eor  #$08
1100 sta  $d018       ;character base
1110 jmp  normirq
1120 ;
1130 ;
1140 dump = *
1150 ;dump hi-res screen to printer
1160 lda  #0
1170 sta  eorflag     ;reverses when 255
1180 lda  ctrlflg     ;ctrl/shift flag
1190 and #4           ;check for ctrl key
1200 beq  norevrs
1210 lda  #255        ;reverse if ctrl
1220 sta  eorflag
1230 norevrs = *
1240 lda  filenum
1250 jsr  close       ;close file
1260 lda  filenum
1270 ldx  #4
1280 ldy  #4          ;sa = 4 for cardco i/f
1290 jsr  setlfs      ;open filenum,4,4
1291 lda  #0          ;no filename
1292 jsr  setnam
1300 jsr  open
1310 ldx  filenum
1320 jsr  chkout      ;(like cmd)
1330 lda  #<screen
1340 sta  scrnptr     ;set up screen memory
1350 lda  #>screen    ;pointers
1360 sta  scrnptr + 1
1370 lda  #27         ;printer control-esc
1380 jsr  chrout      ;send to printer
1390 lda  #51         ;set linefeed to
1400 jsr  chrout      ;16/144 inches
1410 lda  #16
1420 jsr  chrout
1430 lda  #25         ;25 rows
1440 sta  rowcnt
1450 row = *
1460 ldx  #0
1470 prtmode = *
1480 lda  dualres,x
1490 ldy  bigflag
1500 bne  big
1510 lda  normres,x
1520 big = *
1530 jsr  chrout
1540 inx
1550 cpx  #4
1560 bne  prtmode
1570 lda  #40         ;40 columns
1580 sta  colcnt
1590 ;
1600 column = *
1610 lda  #0
1620 ldx  #7
1630 bitsclr = *      ;clear out cell
1640 sta  bits,x      ;storage area
1650 dex
1660 bpl  bitsclr
1670 ldx  #0
1680 ;
1690 cellsum = *
1700 ldy  #0
1710 lda  (scrnptr),y
1720 sta  cbyte       ;store byte from cell
1730 inc  scrnptr     ;point to next byte
1740 bne  dobits
1750 inc  scrnptr + 1
1760 ;
1770 dobits = *
1780 ;add all bits in cbyte
1790 lda  cbyte
1800 and exp,y
1810 beq  notset      ;check next bit
1820 lda  bits,y      ;if set, add to
1830 ora  exp,x       ;'bits' array
1840 sta  bits,y
1850 notset = *
1860 iny
1870 cpy  #8          ;8 bits
1880 bne  dobits      ;next byte
1890 ;
1900 inx
1910 cpx  #8
1920 bne  cellsum     ;next cell
1930 ldx  #0
1940 ;
1950 sendbits = *
1960 ;send all 8 bytes in cell
1970 lda  bits,x
1980 eor  eorflag
1990 jsr  chrout      ;print bit image
2000 ldy  bigflag
2010 beq  justone
2020 jsr  chrout      ;two more times for
2030 jsr  chrout      ;big picture
2040 justone = *
2050 inx
2060 cpx  #8
2070 bne  sendbits    ;next byte in cell
2080 ;
2090 dec  colcnt
2100 bne  column      ;next column
2110 ;
2120 lda  #13
2130 jsr  chrout      ;next line
2140 lda  keybd       ;check keyboard
2150 cmp #63          ;for stop key
2160 beq  abort       ;abort if pressed
2170 dec  rowcnt      ;do next row
2180 bne  row
2190 ;
2200 abort = *        ;dump completed
2210 lda  #27         ;initialize printer
2220 jsr  chrout      ;
2230 lda  #64         ;with " ESC @ "
2240 jsr  chrout      ;(optional)
2250 lda  filenum
2260 jsr  close       ;close file
2270 jsr  clrchn      ;clear i/o channels
2280 jmp  normirq
2290 .end
```

Comparing Two Basic Programs

Jim Butterfield
Toronto, Ont.



I often have several versions of the same program. Sometimes, I can't decide which of two versions of a program – say, DUMMY versus DUMMY5 – is the current version. It's an awkward job to list them both and look through the code trying to find the points of difference. Thus, I've often used program BASIC COMPARE which will draw my attention to the parts which don't match.

BASIC COMPARE uses disk: it checks the two programs as they lie on disk. It depends on the fact that the line numbers will still correspond between the two programs. So if you've performed a program renumber between versions, this program won't help.

Style

This program has an odd attribute: it seems not to use numbers. Let me explain a little further. Some of us are used to file formats such as OPEN 1,8,2,. . . or PRINT#1, but in this case the numbers are almost always missing. Instead, there's a variable. In other words, you might see OPEN J,8,J+1. . . If J has a value of 1, the statement becomes the equivalent of OPEN 1,8,2. . ., or if J is two, the statement performs the same as OPEN 2,8,3.

Why all this? Is it just to be obscure? No: there's a lot of "fiddly" work to be performed on the program files, and it's much easier to deal with the **general** file J rather than the

specific files 1 and 2. Any time you write GET#1, you know that sooner or later you'll have to write an equivalent GET#2; it seems a lot of work.

Variables J1 and J2 have a special function: they set up a loop. When we handle files, we might want both of them, or a particular one. For example: when we have found that the two files don't match, and want to go to the next line, we may need to do any of the following:

- If the files had similar line numbers but the lines didn't match, we want to advance both files to the next program line;
- If file 1 had a line number that was smaller than file 2, we want to advance file 1 only to the next program line;
- If file 2 had a line number that was smaller than file 1, we want to advance file 2 only to the next program line.

Now: we set J1 and J2 to the appropriate "start" and "end" file numbers. To do both files, we have J1 = 1 and J2 = 2, and when we execute FOR J = J1 TO J2, we go from file 1 to file 2 inclusive. When we want to use only one file, we set J1 and J2 to the same value; for file one, J1 and J2 would both have a value of 1, so that FOR J = J1 TO J2 will exercise the one file only.

Array L is there to prevent us from reading a file if we don't want to read it. At end-of-file, we set L() to 9 and won't read that file any more. And when we're reading through lines of code, L() gets set to 1 when we see the end of the line.

All this makes the program somewhat hard reading, but try your hand at working it out: you may find it worthwhile.

And if reading complex code isn't your cup of tea, you may still use the program to compare Basic programs.

Running the Program

This program will run on any Commodore machine that is fitted with disk. Basic programs are not fast, but they do tend to be universal.

The program isn't too chatty, and isn't super fast, but it does the job. It will ask PROGRAM NAME? – answer with the name of one of the Basic programs you wish to compare. It will ask the question again: give it the other program name.

Now the program will clunk away, looking at the two files. If it finds a line in one program that doesn't exist in the other, it will give the name of the program containing the line, and the line number. If it finds a line which exists in both programs, but contains different coding, it will print "*DIFFERENCE*" with the line number.

When it's finished, it stops. If you like, you can add error counts, output to printer, or whatever seems like fun.

PROGRAM: BASIC COMPARE

```
100 open 15,8,15
110 z$ = chr$(0)
```

Open the two program files:

```
120 for j = 1 to 2
130  input " program name " ;n$(j)
140  open j,8,j + 1,n$(j) + " ,p,r " :get#j,a$,a$
150  input#15,e,e$,e1,e2:if e then print e$:end
160 next j
170 n$(3) = " * difference * "
180 j1 = 1:j2 = 2
```

Get the next line from each file:

```
200 for j = j1 to j2
210  if l(j)>0 goto 270
220  get#j,a$,b$
230  if a$ + b$ = " " then l(j) = 9:n(j) = 1e9:goto 270
240  get#j,a$,b$
250  n(j) = asc(a$ + z$) + 256*asc(b$ + z$):n(3) = n(j)
260  if st<>0 then l(j) = 9
270 next j
```

If both files are finished, quit:

```
280 if l(1)>1 and l(2)>1 goto 450
```

Compare the two files:

```
290 j0 = 0:j1 = 1:j2 = 2
300 if n(1)<n(2) then j2 = 1
310 if n(2)<n(1) then j1 = 2
315 if j1 = j2 then j0 = j1
```

If line numbers match, compare lines:

```
320 for j = j1 to j2
330  if l(j) = 0 then get#j,a$(j)
340  if a$(j) = " " then l(j) = 1
350 next j
360 if a$(j1)<>a$(j2) then j0 = 3
370 if l(j1)<1 or l(j2)<1 goto 320
```

Advise of any errors:

```
400 if j0>0 then print n$(j0); " line " ;n(j0)
410 for j = j1 to j2
420  l(j) = l(j)-1
430 next j
440 goto 200
```

Close the files and quit:

```
450 for j = 1 to 2
460  close j
470 next j
```

Unveiling The Pirate Part 1: Current Methods

Richard T. Evers, Editor

In this article I will be releasing information that is known by few, of which those in the know hold to be a very deep and dark secret. In my opinion this has been going on for a little too long. Software piracy has run wild for quite a few years now, and it is about time for a little information to leak out about what is happening, how it is happening and how to impede its cancerous spread just a slight bit.

Though I do not profess to know great scads of information about every form of protection and deprotection known to mankind, I have collected enough to put together a fairly thorough presentation. This article is but the first in a trilogy on this subject. The balance can be found in this issue, and are guaranteed to please even the knowledgeable reader. As I have just stated, this trilogy is for the sole purpose of enlightening programmers about the reality of piracy and how to prevent your creations from becoming just another notch on a pirates belt. This is not a diabolical scheme to hatch new flocks of hackers, even if it appears so at times.

Program protection is incorporated by people who want to protect their creations from copyright infringement. Program deprotection is an occupation taken on by those who find that normal programming is often very dull. Piracy adds a bit of spice to an otherwise terrific occupation. This spice of life costs software manufacturers millions of dollars every year in lost revenue, and it usually does not put a cent in the average pirates pocket. I stress the point that this is usually done for the thrill, not monetary gain, because this fact alone makes the prosecution of pirates in court very difficult. The courts are usually good for offenses that can be easily proven, which is rarely the case with software theft. For more information on the legal aspects, flip to part 3.

There are five methods commonly used to protect software from illegal distribution, which are as follows :

1 – Diskette protection
2 – Dongle Protection
3 – ROM Protection
4 – Program In ROM Pack
5 – No Protection

Diskette Protection

This form of protection is one which I favour the least. This technique, though cheap to incorporate and often one that will work, is pretty bad news for the average user. Once a user has purchased the protected diskette, they have one or two copies to work with. As most people can confirm, accidents do happen. If your disk drive packs up, your dog eats your diskette, or even if you happen to mess up the diskette yourself, you are in trouble. Some manufacturers give you two diskettes. Very nice. They also give you a card that entitles you to get another disk, at a nominal charge, if you return your fouled up diskette to them. The trouble with this is the software manufacturer lives in some distant city, leaving you with only one method of transporting the diskette short of an expensive courier - the mail. And in all probability, the manufacturer won't spare the expense on the former. So if the postal service is as careful with your mail as it usually is with mine, then expecting a new diskette back in workable condition is more of a fantasy than anything else. Even if the diskette does not look physically damaged, you will often find it riddled with new and improved read errors. Just terrific. It seems that troubles accumulate faster than you can get rid of them.

Diskette protection has other disadvantages. First, it really makes your disk drive work for its money. The read errors and strange formatting tricks cause your drive to virtually have a stroke every time a protected disk is read in. This protection also takes up to three times longer to load in than normal, being especially noticeable with the 1541 drive. For an encore, disk protection is often written for a specific drive, excluding all others. For those of you with a dual drive setup with your 64, you may often be out of luck trying to LOAD in a protected disk. When faced with more than one drive, or with a RAM/ROM or interface combination that the program can't figure out, the software purposely bombs out. A rotten trick to play on someone who has invested in your program.

Pirates seem to enjoy disk protection though. The challenge alone makes your program an easy candidate for the next "unprotection". Between the bit copiers available on the market, and rewriting the software to stop checking for all the errors on disk, there is a great quantity of bootleg software available that was originaly disk protected. Not only that, but these deprotected wonders are often better

than the originals. They LOAD in faster, and save you disk space by allowing more than one program per diskette. These two factors alone have the average user avoiding software that is protected this way.

Some time ago I was shown a method to break a few of the simple disk protected programs available. The trick to this is to first backup a copy of the protected disk on a 4040 drive, or any dual drive with swing down doors, then start the backup a second time but with a twist. Open and close the drive door about ten times, or until the backup procedure ends. Then remove the diskette from the drive. A read error has been created on the new copy that will closely resemble what is found on the original. If the software manufacturer has relied entirely on this single read error for protection, then the game has been lost within a five minute period. Even if their method of protection was a little more extreme, a good hacker will end up winning. For all of these reasons, I do not recommend disk protection at all.

Dongle Protection

This form of protection is my personal favourite. You have at least a fighting chance against the pirate with this one, with the victor often the manufacturer. In case you are unsure of what dongle protection is, let me explain. A dongle is a rude name for a hardware apparatus that is plugged into your computer. On the PET/CBM series, a dongle can be located on the user port, or on either of the two cassette ports. On the Commodore 64 and Vic 20, you can locate them on the user port, game cartridge port, joystick ports and cassette port. Quite a few options. Now for the explanation of what they do. Inside the dongle can be found anything from one piece of wire to a complete assortment of electronic components. With the proper combination, and the proper location, a program can check to make sure that the dongle is in place. For an added thrill, use the results generated by the dongle in the calculations and operation of the program itself. Anything from timers or pulse multipliers to frequency generators or filters can be included. Therefore, even if the hacker can manage to stop the program from checking for the dongle, the program may never work properly again.

There are ways around dongle protection though. The simplest method is to break into the dongle and find out what's inside. If this can be achieved the hacker has a 50/50 chance of reproducing it.

If the dongle is filled with some form of material to stop breakage, the hacker may assume that the covering has been placed there simply to disguise virtually transparent protection. Not transparent in the sense that there is none, but transparent in that it can be quickly reproduced by those in the know, if so inclined. Some dongles are merely a jumper between two pins. Protection like this lasts about as long as one cup of coffee. A little more thought an a dongle can send a hacker to a caffeine rehabilitation center. It's up to you to decide how clever to make it.

At this point the hacker has a few options. Crack off or dissolve the material encasing the components, or X-Ray the entire key to see what's inside. If the identication hasn't been removed from the components, and the wiring isn't purposely misleading, then the X Ray technique will probably work. It's amazing how a friendship with a dentist can be beneficial to a pirate.

To stop the hacker from gaining any ground by chipping away at the covering material on your dongle, place a few very important thin wires throughout the material itself. Once the chipping begins, these wires will be cut by the illustrious chipper, thus making the dongle useless. If enough wires are used, the key will become useless to the hacker by the time they reach any important components. In a proper casing, a dongle will be destroyed before it reveals itself.

Dissolving the material that covers your components is one method that can prove effective if care is not taken to disguise the operation of the circuitry or the identification of the components. There is one sure method to discourage the hacker from this technique. Use a material that is impervious to most solvents. Most software manufacturers use whatever plastic material they can find, like epoxy resin. There are many commercially available chemicals that can dissolve epoxy in relatively no time at all. And it's a shame to allow a hacker to win so easily.

There is one substance I use that is impervious to solvents, or heat for that matter. It is called methyl methacrylate, or quite simply, denture material. This can be purchased in many forms, with the easiest and least expensive being Tray Material. Tray Material is true denture acrylic, but manufactured for a vastly different purpose. Though the dental profession frowns on sales outside of its little community, try a few of the smaller dental supply companies, or smaller dental manufacturers. These companies will often deviate from normal procedures, with the correct amount of prodding. And your dongle producing department will feel much more confident. Expect to pay about $8.00 per pound up for the material.

The reason why dongle protection is a favourite with me is because it's terrific for the user. The installation of the dongle is often very easy, and there is no limit to the number of copies that can be made of the program disk. There is also that security blanket knowing that the pirate has to actually get into your code and figure it out how to stop the check for the dongle to break your beast. If care was taken in the design of the program, and if thought was given to use the results generated by the dongle in the actual operation of the program, then the hacker may be in for an indefinite amount of work.

For a final analysis on this one I recommend it whole heartedly. The cost is higher for the manufacturer in relation to disk protection, but the end result is better business. A replacement key can be shipped through the mail, without

damage in most cases. There will be no undue wear on the users drive through use of your software, and you can expect less pilferage with a dongle protected program. Pretty good all around, but still not impervious to a determined hacker. However, most, if not all, will throw in the towel after buying their third or fourth package in their attempts to unprotect your program.

ROM Protection

ROM protection is a form of protection that is disliked by many. A technique that applies more to the PET/CBM user than any of the others, this type of setup uses a single ROM placed in either the $9000 or $A000 socket in the computer. The ROM will have anywhere from 2 to 4K of code burnt into it to help stop illegal usage. The installation of this ROM by a user is the pitfall here. Broken pins and improper installation are too often the end result for the inexperienced. It is also a simple challenge for most pirates, and can be financially prohibitive. The initial cost to produce a ROM far outweighs the effectiveness in most cases.

The most common method programmers use to bypass ROM protection is to have a soft ROM built into their computer. A soft ROM is a device made from RAM that can appear to be ROM in the eyes of the software. ROM contents can be saved to disk and then loaded into the soft ROM, thus fooling the program into believing the ROM is actually there. Soft ROMs are available from many sources, and can be installed in little time. Their average cost is about $150.00, and can be used for the $9000 socket, $A000 socket, or both. Your choice. This method of piracy is not one that software manufacturers worry about though. They worry about the hacker that actually rewrites their program to work without the ROM.

This method of deprotection was more common a few years ago than it is now. All the truly fine programs that were ROM protected have already been broken. The method used to break the program is usually to relocate the contents of the ROM somewhere else in RAM, then rewriting the program to access it in the new spot. The user may lose a bit of memory, but the programs will still work.

Another method used was to rewrite the program to not check for the ROM at all. Some programmers, when designing ROM protection, never actually put much thought into how they were going to protect it until it was too late. The routines in ROM were not used for anything, therefore the programs were usually very simple to break. Many other ROM protected programs available have been been well designed in the protection department, with vital code actually placed within, but still to no avail. The pirate knows where the protection is, and can often see the access points with a simple disassembly. The balance of breakage occurs with a little time and effort. But it can usually be accomplished. Pretty rotten, but possible.

The final method that the hacker would take to break a ROM protected program is to physically copy the ROM itself. ROM burners are available from numerous sources, with average cost riding around $100.00. What these burners allow you to do is mass produce most ROMs with EPROMs, as long as you have an original copy of the ROM on disk. For the price of a ROM burner, and the price of the EPROMs, about $15.00 each, the pirates can do whatever they like. Some pirates have been known to photo copy manuals, burn new ROMs and sell the pirated packages for reduced prices, which is a very sleazy way to steal a dollar.

When all has been taken into consideration, ROM protection is not the best way to protect your program. Not only is it a pain for the average user, especially if a few ROM protected packages are used, but it's an easy mark for most pirates. If at all possible, stay away from this one.

Program Located In RON Pack

This is the protection most encountered today in the games that Commodore releases. This technique is one that is pretty good, and very nice for the user. A simple plug in of the ROM pack into the game socket, and your program comes alive. Very nice, but hardly impervious to the inventive hacker.

There is one device designed and used by pirates that is constantly in use destroying ROM Pack protected games. After the computer and cartridge combine to allow the program to begin, a press of two buttons concurrently will bring this poor game to its knees. This high tech black box has effectively stopped the program in its tracks without destroying the colour table, zero page or any of the other equally important areas in RAM. With this step taken, a simple SAVE to disk and a bit of work later will produce another broken game to add to an already overflowing collection of pirated programs. What a rotten trick, to design a electronic pirate. Write a bit of code, toss in a bit of hardware, and presto, instant hacker. The human element has finally been taken out of the piracy game.

Another method used to break ROM pack protection is so simple that most do not think to use it. Reproduce the ROM and make your own pack. The trouble that pirates find with this is the cost of the ROM pack. When mass produced, the cost for raw materials is very low. When bought by the average hacker, the price is very high. And the cost of the ROM pack has to be shelled out each and every time a copy is to be made. Too high for most pirates, which is exactly what the industry hopes for. A hacker is usually too cheap to spend the money on the programs themselves, when they know that most of the their software came for the price of a diskette, if they bought the diskette at all.

Making cartridges can get pretty involved too. Some use several PC boards laminated together with enough interconnections to make X-Rays of the unit so confusing that they're useless. And like a good dongle, any attempt to dismantle the boards usually ends up destroying them.

In my opinion, this form of protection rates just below the dongle. Your program will be quite safe, but a little more costly to produce. You be the judge, for you alone know what market your program is geared towards. If the market is huge, there is a pretty good chance that your profits will still be huge, even with breakage.

No Protection At All

The last form of protection for this article is zero protection in the major sense of the word. I know that most people don't agree too much with that one, but it could be viable. All it would take would be some careful prior planning.

By prior planning I mean that you should write your program with a really fine manual in mind. Games players usually don't need the manual, but business and application software can be worthless without it. Your manual can be printed so it's tough to photo copy, and photocopied books stand up better as evidence in court. Besides, photocopying is too expensive for chintz hackers. The software might be fabulous, but supplying a manual with every copy they wish to hand out will have pirates moving on to less documented packages.

Other protection might be considered like a dealer/user contract, to be signed by the user, and a serial number placed on each diskette, in a spot few would ever think to look. Make it clear with a message in your program that without the contract the user is in violation of the law. Forged contracts are even more frightening to a reputable business than a copied manual. Produce a truly fine overall package and most would rather own one legally than accept a duplicate.

Serial numbers will help too. Write the serial number in the manual in a few locations, so that it will reproduce if photo copied. Remember that most people don't look through the entire manual before photocopying it. Copyright law does apply to manuals, so guard yourself well. The combination of dealer–user agreement and manual will stand up in court if necessary. The serial number written on to the diskette is also a great help. Don't inform anyone that the serial number is on the diskette, just keep a log of it somewhere, like in your records, and on the contract that the user signs. If you ever find that copies of your program are circulating, a quick check onto the disk surface will determine where the diskette originated. With that determined, legal action is your next step.

A word on the placement of the serial number. Write the number in a spot on the surface of the diskette that is sure not to be disturbed by any disk activity. For all of the disk types numerous hiding places do exist, you just have to locate them. For an encore, encrypt the serial number in such a way that if it was found, it could not be easy deciphered. The low/high ASCII value is fine, but the low/high number EOR'd with a number known only to your company, would be just right. Have your program check for the number, just to make sure it is there. If the number isn't, fry the disk itself. Then put the computer into a death loop, just to even the score a little for them trying to steal your creation. When your rights are at stake, protect yourself to the hilt.

Another point to ponder with this technique is to store the serial number on disk along with a constant used for calculations within your program. Numerous programs on the market use their protection as part of the calculation process, for setting up the screen dimensions, some mathematical calculations, and for a variety of other reasons. Why not do the same thing. Instead of frying their diskette, let them continue using the program. If the serial number isn't there, the constant will be absent from their calculations.

Just imagine how much money it would cost a company to use a pirated accounting package protected in this manner. They might not realise that anything was wrong until year end. Then suddenly they would have an entire fleet of auditors ripping through their records to find out how they made/lost all that money during the year. For the few hundred dollars the company saved on the program, they lost it a hundred times over in man hours to correct the mistakes. A pretty good way to get even with a cheap firm. If the firm has the nerve to complain, ask them for the diskettes, manual and user contract from the point of purchase. Try not to react too quickly, and the unsuspecting firm may lead you to the source. Then bring the curtains down on both of them. A lawyer is the next step, and you have a pretty good shot at winning too.

And Finally, The End

Other methods exist to protect and deprotect programs, but these are really just further extensions to those already covered. As mentioned at the beginning, this article is one of three that has been prepared for the occasion. As you have discovered, this one deals with the methods used to copy protect your programs. Part two I enjoyed writing most; Programming Sleight of Hands, an article that will take you through some practical methods to protect your programs, and how these methods are often superceded. The final article is about the legal aspects of piracy, a product of some quite extensive research.

Whatever your choice of reading, I hope that you have enjoyed this issue. Your views on this subject, and on everything that we have printed in this issue, are welcome. We will be able to refine our magazine into the jewel we know her to be capable of becoming, with just a little help from you. Express your views on paper, disk or cassette tape, and send them in to us. By piecing together what our readers like and dislike about our publication, we can produce the highest quality magazine found anywhere.

May you find that all of your bugs have four wheels and an engine in the rear. RTE

Unveiling The Pirate Part 2: Programming Sleight Of Hand

One of the most exciting things a programmer can master is the art of making your computer do weird things. Our Bits and Pieces section is one example of where grown men and women search out and publish new and improved ways to make your computer beg for mercy. The same is true with most programmers who get in the mood for some really hot protection. It is all just an extension of the same idea, to make your computer perform tricks that are not normally part of its act. This article has been written to show you some new and improved ways to add confusion and pain to the lives of pirates everywhere. Though not a glossary on every method that can be achieved, it should suffice to whet your appetite and start you on the trek to find some more. Protection can be addicting.

A Quick Word On Compiling

Though not a subject that excites many in the programming community, I felt it best to discuss compiling for a few minutes, just to let you know that it is still around and is prone to breakage if attacked by the serious pirate.

Compiling a BASIC program is usually a reliable way to discourage most hackers. Once the infamous words of COMPILED BY . . .. are discovered, most hackers turn away. Again, there are exceptions to this rule. Available through the bootleg community right now are a few decompilers for the most widely used compilers. There is one that I am quite familiar with that can decompile a compiled program back into its original state as it was before compiling ever took place. If your only method of hiding protection was compiling, then forget it. This one will bare to the world whatever tricks you were performing. A pretty low life trick, but it is extremely effective.

If the chances are that your program will not be too badly exposed if decompiled, then compile to your hearts content. Just remember that there are a number of hackers out there right now with their decompilers, revving up their disk drives in anticipation. Make sure that your code is a little more tricky in the method used to check for protection. In this way they may have to work a little harder to achieve the same end result, if it can be done.

POKES, Line Tricks And DOS Tricks

As Mary Poppins once stated so elegantly, "these are a few of my favourite things." In this I support her convictions all the way. Give me a memory map and a computer, then stand back. It can be quite a challenge to set my devious little mind to work on new and improved methods to supercede the evil wishes of that master of deceit, the pirate. Aar de aar Billy, have you broken that package yet?

Below will be found all sorts of tricks that have either been well concealed in the past, or never thought of before. Combining several techniques is your best defense. However your protection must share space with your program. Too much protection and your program may run out of memory. It's up to you.

Auto Check For Protection: CHRGET

For anyone wishing to have their program automatically check for protection, CHRGET is often found to be rather handy. Located at $70–$87 with the PET/CBM and $73–$8A on the C64/VIC, a slight change within will allow you to reroute the CHRGET flow to where ever your protection desires. For the PET/CBM, locations $79–$7B can be modified quite effectively to include a JUMP followed by a 16 bit address. The same applies for locations $7C–$7E on the C64 and VIC. If you reproduce everything from these locations down to the end of CHRGET, and master the basic concepts of how CHRGET works, then you are sure to be able to CHRGETize your protection with few snags.

There is one very large disadvantage in using CHRGET for executing vast amounts of code though. BASIC execution will be slowed down relative to the amount of extra code CHRGET is expected to execute in its quest for protection. Keep your protection code compact and your program won't suffer too badly in performance.

Locations To POKE About With

The table on the following page is not a chart of yet undiscovered orifices to prod about in, but a compendium of locations in RAM that can really turn a computer on. Proceed below and open up a vast new world of tricks you can play on your yet unsuspecting computer.

#01 USR Function Jump

As has been explained by so many people in the past, the USR function in BASIC can be used quite effectively to access machine code from BASIC. By altering the jump address to where ever you like, indirect methods of accessing protection or plain and simple code can be found. A SYS to this address will also produce some fine results, if you are so inclined. Whatever your requirements, this vector can be handy at times.

40 Col PET	80 Col CBM	C64	VIC 20	Description
$0000–$0002	$0000–$0002	$0310–$0312	$0000–$0002	(01) USR Function Jump
$0090–$0091	$0090–$0091	$0314–$0315	$0314–$0315	(02) Hardware Interrupt Vector
$0092–$0093	$0092–$0093	$0316–$0317	$0316–$0317	(03) Break Interrupt Vector
$0094–$0095	$0094–$0095	$0318–$0319	$0318–$0318	(04) NMI Interrupt Vector
N/A	N/A	$031A–$031B	$031A–$031B	(05) OPEN Vector
N/A	N/A	$031C–$031D	$031C–$031D	(06) CLOSE Vector
N/A	$00E9–$00EA	$0324–$0325	$0324–$0325	(07) INPUT Vector
N/A	$00EB–$00EC	$0326–$0327	$0326–$0327	(08) OUTPUT Vector
N/A	N/A	$0328–$0329	$0328–$0329	(09) Test STOP Vector
N/A	N/A	$032A–$032B	$032A–$032B	(10) GET Vector
N/A	N/A	$0330–$0331	$0330–$0331	(11) LOAD Link
N/A	N/A	$0332–$0333	$0332–$0333	(12) SAVE Link
$0028–$0029	$0028–$0029	$002B–$002C	$002B–$002C	(13) Pointer : Start Of BASIC
$0034–$0035	$0034–$0035	$0055–$0056	$0055–$0056	(14) Pointer : Top Of Memory
$009E	$009E	$00C6	$00C6	(15) # Chars in Keyboard Buff
N/A	$00E3	$0289	$0289	(16) Max Size of Keyboard Buffer
N/A	N/A	$0300–$0301	$0300–$0301	(17) Error Message Link
$03FA–$03FB	$03FA–$03FB	N/A	N/A	(18) Monitor Extension Vector

#02 Hardware Interrupt Vector

Another name for this vector is the IRQ Vector, one that I am sure you have seen mentioned in our Bits and Pieces section. This vector can be changed to point to whatever code you want executed repeatedly, like a check for protection. One point to remember before performing any major operations with this vector, though. The more pre-interrupt code you have, the slower will be the execution of normal code. An important trade off to consider.

Another point to remember is to save the processor status and all registers that are corrupted on the stack before any operations. Then restore the values before transferring control back to the system.

For 4.0 BASIC people, you have probably noticed that you cannot use the disk drive for very much if the IRQ procedure has been changed. Try the following trick and I am sure you will be pleased. Point the BRK interrupt vector at your code, then point the IRQ vector at $E454. This location is a zero byte, or a BRK instruction. With every IRQ, the machine will break, then shoot over to your code. Clean, simple and a pretty neat trick to know. And all SAVEs and LOADs from disk will go off without a flaw.

One final word on the IRQ. If all else doesn't interest you then point the normal IRQ vector 3 bytes forward to disable the STOP key. Pretty boring, but what the heck, it works.

#03 Break Interrupt Vector

As mentioned above, the BRK vector can help you out with your IRQ driven wonder. But it also can be used for a few other things. Often, when trying to break programs, break points are desired at specific spots to check how certain activities are going. For this, a BRK instruction will be placed in the machine code for the program to BRK into the machine language monitor when encountered, or back into BASIC mode with the C64 and VIC unless a monitor is in operation. At other times, even when the program has been protected to the hilt, it can be made to crash. The first thing that a 4.0 hacker will usually do is break into the machine language monitor to look at the code, or save everything for later viewing. Point the BRK interrupt vector at reset:

$FD16 (L/H = 22/253) for 4.0 BASIC
$FCE2 (L/H = 226/252) for the C64 & VIC

Every time a BRK is encountered, the computer will reset to a cold start. Dirty pool, but why make life for a hacker easy.

#04 NMI Interrupt Vector

Hooked up to my 8032 is a little device called the break box. Jim Butterfield and a few others have been talking of this little wonder for years. With the break box you can stop a program in its path and go to READY mode. Or you have one more option, at least with my particular model. You can break directly into the 4.0 monitor, without disturbing the program in the least. Well, for the first action of breaking into READY mode, there is a cure. Point the NMI vector at reset (see above). When an attempt is made to break to READY mode, you will be met with a cold start of the computer. A pretty simple way to deter a few hackers.

#05 OPEN Vector

This vector is only accessible by those with either the C64 or Vic 20, which is a real shame for all the 4.0 users reading. Commodore really had their heads screwed on straight the day they introduced neat vectors like this. To confuse the heck out of pirates everywhere, change the meaning of OPEN within your program. Point it at anything that you feel like, and watch the confusion grow. Imagine LOADing in a program with a simple:

OPEN 5,8,5," program name "

Flip the LOAD/VERIFY flag and try it. It seems to work. If all else does not appeal to you, point the OPEN vector at reset when not in use. Might never be used, but why not. Some hacker may use a technique involving the OPEN statement, of which a cold start would come as a surprise.

#06 CLOSE Vector

As with the OPEN vector, the CLOSE vector is also limited to those with either the C64 or VIC. The same techniques apply with this vector as was with the OPEN vector, so cloud the issue a little when using this one and point it to everything but what it really is. As I have said before, if all else fails, point at reset while not in use.

#07 INPUT Vector

This is a favourite of mine. Place the following auto run code somewhere in memory, and point the input vector at it once in program mode. As long as you do not use an INPUT statement anywhere in your program, this bit of code will not be executed until you have gone back to READY mode. Once that happens, the code is executed and your program will be re-run all over again. Terrific if someone is trying to crash your program, or you don't want anyone to see your program crash. Keep this code in mind when thinking about re-routing some of the other vectors too. Just imagine the confusion level if every move the pirate makes causes the program to re-run.

Another use for this vector is to point it at reset, as I have belabored with every vector so far. The moment your computer goes back into READY mode, the machine will execute a cold start. It's mean, but it works.

*** Auto Re-Run Code For 4.0 BASIC, C64 and VIC 20 ***

```
lobas  = $28   ;4.0 basic - low  byte start of basic
loval  = $01   ;4.0 basic - low  byte value of start of basic
lobas  = $2b   ;c64 & vic
loval  = $01   ;c64 & vic (default)
hibas  = $29   ;4.0 basic - high byte start of basic
hival  = $04   ;4.0 basic - high byte value of start of basic
hibas  = $2c   ;c64 & vic
hival  = $08   ;c64 (default)
hival  = $10   ;vic (default)
lovar  = $2a   ;4.0 basic - low  byte start of variable
lovar  = $2d   ;c64 & vic
hivar  = $2b   ;4.0 basic - high byte start of variable
hivar  = $2e   ;c64 & vic
loend  = $c9   ;4.0 basic - low  byte end of program
loend  = $ae   ;c64 & vic
hiend  = $ca   ;4.0 basic - high byte end of program
hiend  = $af   ;c64 & vic
clr    = $b5e9 ;4.0 basic - reset basic and perform 'clr'
clr    = $a659 ;c64
clr    = $c659 ;vic
fix    = $b4b6 ;4.0 basic - fix chaining
fix    = $a533 ;c64
fix    = $c533 ;vic
cont   = $b74a ;4.0 basic - perform 'cont'
cont   = $a857 ;c64
cont   = $c857 ;vic

;*** auto re-run with reset of start of basic***

lda #loval  ;reset start of basic to normal
sta lobas
lda #hival
sta hibas
lda loend   ;reset end of program/start of variables
sta lovar
lda hiend
sta hivar
jsr clr     ;reset basic and do 'clr'
jsr fix     ;fix chaining
jmpcont     ;perform 'cont'
```

#08 OUTPUT Vector

The OUTPUT vector is another versatile vector that can be used for all sorts of functions. For every key that is pressed on the keyboard, the OUTPUT vector is used. For many of your disk operations, the OUTPUT vector is also used. Point this vector at the code of your choice, make sure that the code is not too verbose, save and retrieve all corrupted registers on the stack, and an alternative to CHRGET protection has been found. Those of you with PETs are excluded from this one though. It seems Commodore wasn't that bright when the PET evolved, but adapted later for all the others. For all around use, this one is a pleasure to use. Few people know of its existence, so why not use this factor to your advantage.

#09 Test STOP Vector

Again, another point scored for C64 & VIC users. One more vector that the PET/CBM owners drool to have. This vector allows you to disable the STOP key:

POKE 808,239

. . .disable the RUN/STOP-RESTORE combo:

POKE 808,225

. . .or whatever you can find to disable when rummaging about through the code. Again, as I have mentioned earlier, develop some really interesting code for your protection, then point a vector at it. This one is as good as you will probably find, so consider it when looking for a new and improved vector to take advantage of.

#10 GET Vector

Another vector you C64 and VIC owners have over the rest of the world, and a pretty good little beast to have on your side. As you probably know, GET is a BASIC keyword, associated with a command to GET characters from some device or another, usually the keyboard. Point this vector at some ingenious protection code, and watch as the code is executed every time you attempt to GET anything within your program. A sneaky and underhanded way to bring a pirate to his knees, but who really cares. Its your program,

so why not try to keep it that way.

#11 LOAD Link

Add this one to the C64/VIC resume, for the PET/CBM owners have been left in the cold once again. Without flogging a dead horse any more than necessary, change this link address to point elsewhere and you may be able to confuse many who peek into the lower recesses of your code. Use your imagination and you will be surprised as to the number of tricks that can be performed with just a few vectors and a couple of knowledgable protection routines.

#12 SAVE Link

Take a peek up above and read again, this time remembering that we are discussing a different vector. PET/CBM owners have once again been given the dirty end of the stick, with the C64 & VIC people coming out with another winner on their side. Use imagination and plenty of raw, brute spunk, and watch as the hackers fall aside to the power of your code. Science fiction writing has always been a secret ambition of mine.

#13 POINTER: Start Of BASIC

Executing BASIC at a different location other than normal is an interesting way to confuse a moronic pirate. LOAD in your normal program, change the Start of BASIC, then chain in the next section of code. Execution will be immediate, and confusion will suddenly run rampant. In this way, you could have numerous programs in memory at the same time, all stacked above each other with each performing a specific function. If broken, only one section would easily be viewed. A technique that is rather unorthodox, but logical considering the circumstances.

#14 POINTER: Top Of Memory

As most already know, lowering the top of BASIC memory will give you a place to store additional code where it can't be disturbed. Alter these pointers to anywhere you like, and you can mix a good proportion of BASIC and machine code together in harmony.

#15 Number Of Characters In Keyboard Buffer

For some auto boot routines, this one is terrific. A few issues back I wrote an article about using the keyboard buffer and this location for an easy boot technique. The technique is very fast and simple and a pleasure to see in action. Once the program knows what is to be done, print the actions required on the screen in calculated positions. Poke carriage return characters (CHR$(13)) into the keyboard buffer to coincide with the screen contents, then poke the number of characters presently in the keyboard buffer into this location. The next step is to END the program. The keyboard buffer will take over from there to do what ever you have requested. A nice sleight of hand.

#16 Maximum Size Of Keyboard Buffer

This location is one which can come in very handy. Normally set to allow 10 characters into the buffer at a maximum, you can alter this for any value up to 255. Alter this location to any value you care before executing routines that require an input from the keyboard, then down to 0 once the routine has received all that it wants. If the program ever crashes for whatever reason, what good would it do anyone? The keyboard has been effectively turned off, thus stopping the code snoop in his tracks. Sneaky and very reassuring to use.

#17 Error Message Link

A programmer that I know, Brian Munshaw, developed an ingenious method of using this link to his advantage on the Commodore 64. He has designed a fabulous graphics package, but did not want to use CHRGET to check for his special keywords. Using CHRGET wedges quite often slows down BASIC execution a little bit too much when heavy checking is required. Brian designed a little bit of code that is pointed to by the error message link. Normal keywords will not generate an error, but pretty much everything else will. This code checks the error generated to see if it was in the range of one of the new keywords, or just an ordinary error by some spastic programmer. If the error was the new keywords fault, then the code will branch to the routine to handle it. For normal errors it will jump to the normal error processing routine. Pretty terrific, and also a great way to design packages that will confuse many people. Design your own language for your program, and incorporate some protection techniques into the language. The work required to fix your program up would be more than most hackers would want to allocate.

#18 Monitor Extension Vector

This vector is used only on 4.0 BASIC machines by utilities that extend the machine language monitor with extra commands. Change this one to reset, $FD16, and even if you have missed a few of the pirates spies, the moment they try to disassemble your code, or what ever else they have planned with their extended monitor, the computer will jump to a cold start. For two pokes, not bad.

There are many more POKEs that are as yet still in hiding, and will remain so for the balance of this issue. By stretching this article on too long in this area, many other points would have to be skipped. In future issues we will be printing new and improved POKEs to add to this collection.

And now, DOS tricks and other shenanigans to further inflame the boils of irate pirates worldwide.

Playing Tricks With DOS

The disk drive should be classified both as a tool and a toy, at least for me. These versatile little containers of brilliance can liven up even the dullest of days, and make the average program sing with delight. Not only can you tell the drive to make some pretty obtuse maneuvers, but you can also tell the disk that it enjoys what is happening. In this way, you can let fly with a few smoke bombs even the most astute hacker can't see through. This is war, so let them have it with all you got!

Keep in mind that multiple stacking of disk protection tricks can be of great benefit in the final outcome of this war.

Change Block Count Of Files

You can physically change the block count of files on disk, and confuse the heck out of people who copy your files. They will never be completely sure if they got the right file, or the complete file. There will also be the chance that they would never look at a small file when they are trying to locate the main program. To change the block count, look in the directory track of the disk. Held in low/high byte fashion, they are the last two bytes in the files directory entry. Anything from 0 to 65535 is acceptable.

If an easier route is desired to change the block count, take a peek through this issue and you are bound to trip over a program I wrote to change the block count of files, scratch protect files, and back-up protect your diskettes. Another utility, DiskMod, will help you do the same thing, but it's as little less automatic.

Re-Direct Track & Sector Pointers Of Files Back Unto Thine Self

For a bit of fun with your disk, foul up a couple of files in this way. Files are held on disk with the first two bytes of each block as the pointer to the next block of data. By changing this pointer to point back to one of the blocks prior, copying files could be somewhat tedious at best. A COPY command would go forever.

From program mode you could do a few things to compensate. Read a specific section of the file in one byte at a time, then close up the file. Or you can write to the disk to reset the pointers correctly, LOAD in the program, then reset the pointers back in your strange fashion. A hackers nightmare is born.

LOAD And SAVE Programs In A Novel Way

When preparing this article I was indecisive whether to let you know this little gem of protection. The number of people who know of it is a mystery to me, but I do know it to be a carefully guarded among those who hold the secret. The purpose of our magazine is to educate, so lets peek into this one and become educated.

Program files can be saved to disk as SEQ or USR files, with the right technique. These same files can then be LOADed back into memory and re-executed as PRG files.

To SAVE a program file as a sequential file try this :

```
save "0:filename,s",8
```

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

```
load "0:filename,s",8
```

The file will appear on the directory as SEQ, but it will really be PRG. To SAVE and LOAD USR files, substitute a 'u' instead of the 's' in the above examples.

Make Disk Non BACKUPable

As stated earlier, there is a program to be found in this issue that will perform this trick with your diskettes. It will allow you to protect your disk from regular duplication. Not only this, but you cannot SCRATCH files, SAVE to the disk or RENAME any files for all time thereafter. The only trouble is that COPYing is still allowed, but we're working on it.

On the disk surface, there are markers called DOS Version Identifiers. These little beasts tell the drive what version of DOS the diskette was formatted on. By altering these bytes, the drive will be fooled into believing that the disk is wrong for it. For a 1541/2021/4040 diskette, the identifier is a 65 decimal, or $41. It is located on track 18, sector 0 as the third byte from the start. Change this to what ever you like, and your disk will become an alien.

On the 8050/8250 drives, the DOS Version Identifier is located on track 38, with both drives using sectors 0 and 3, and the 8250 also using sectors 6 and 9. The normal byte value is 67 decimal, or $43, and is also the third byte from the start. Change this animal to whatever you like, and watch the hackers sweat just a little more.

Now I am going to pop the bubble that was just created. While pouring over some maps that I have been preparing on the disk drives, I came up with a technique to backup a backup protected disk. All that you do is tell the drive that it really is as strange as the disk says it is. Then you can write to the disk, SCRATCH, SAVE and BACKUP. Performed from within program mode, you could control just what activity is taken with your disks. The code is as follows.

```
10 id = ascii value of new DOS identifier
20 open 15,8,15
30 print#15, "m-w" chr$(159)chr$(16)chr$(1)chr$(id);
40 close 15
```

This code works for 4040, 8050 and 8250 drives. For the others, excuse me but I haven't come up with the code yet. Maybe in a future issue.

Change File Type On Disk

This technique, though not one that should be encouraged by anyone but a person bent on the destruction of all hackers, is pretty inventive to say the least. Confusing the heck out of a hacker is exactly what the following will do. Below I have prepared a chart with disk massaging tricks.

HEX	Description
$00	Unclosed DEL File (*)
$01	Unclosed SEQ File (*)
$02	Unclosed PRG File (*)
$03	Unclosed USR File (*)
$04	Unclosed REL File (*)
$80	Closed DEL File
$81	Closed SEQ File
$82	Closed PRG File
$83	Closed USR File
$84	Closed REL File
$88	Closed DEL File - fouled up file type description - copys as DEL
$89	Closed SEQ File - fouled up file type description - copys as SEQ

$8a Closed PRG File – fouled up file type description – copys as PRG
$8b Closed USR File – fouled up file type description – copys as USR
$8c Closed REL File – fouled up file type description – won't copy
$c0 Scratch Protected & Closed DEL File
$c1 Scratch Protected & Closed SEQ File
$c2 Scratch Protected & Closed PRG File
$c3 Scratch Protected & Closed USR File
$c4 Scratch Protected & Closed REL File

Bit Representations :

bit 0	off = DEL file, on = SEQ file
bits 0 & 1	bit 0 off and bit 1 on = PRG file
bits 0 & 1	bit 0 on and bit 1 on = USR file
bits 0, 1 & 2	bit 0 off, bit 1 off and bit 2 on = REL file
bits 3 – 5	not normally used
bit 6	off = normal file / on = scratch protected file
bit 7	off = open file / on = closed file

As the charts above show, there is quite a bit to be learned about the diskette itself. For years people have been wanting to scratch protect their files, with the best example being teachers in schools with hundreds of students bent on file destruction. Commodore, with their typical streak of brilliance, have never bothered to tell anyone much of anything at all. Well, files can be scratch protected on every version of DOS so far. The trick is to set bit six of the file type indicator on the directory entry of the file. Once set, a '<' will appear to the right of the file type, and the file cannot be scratched. Everything else is still allowed though. Don't you think that the time has come for Commodore to start letting us in on the workings of their drive units? They have never used this feature for anything whatsoever, but they knew it was there all along. Why couldn't they let a few people in on their secret, and maybe benefit a whole lot of people in the process? I leave you to fill in the answer to this one.

The file type values of $89 to $8b are of special interest to me. You could save a PRG file to disk, then change the file type description on disk to anything from $89 to $8b. These files will not LOAD in normally, but they will COPY correctly. From program mode you could COPY the file on disk first to something like a USR file, LOAD the USR file in as PRG, then SCRATCH the recently copied file. It is a lot of work, but it would prove to be good armor against the ever illustrious hacker.

Change Header And Filename To Foul Up Directory

Filenames and headers are often not thought of to foul up the hacker. Most users of software never have to do a directory of the disk anyway, so why not throw a few curve balls in the directory department.

By curve ball I mean a few pseudo control characters. Try the following header manipulation and I am sure you will quickly understand what I am getting at.

Format A Disk, With A Twist

```
10 fu$ = chr$(141) + chr$(19) + chr$(19) + chr$(147)
   + chr$(15) + chr$(143)
20 open 15,8,15, " n0: " + " " + fu$ + " "
30 close 15
```

For those of you with the CBM machines, you will benefit most from this demonstration. Not only will this header, once viewed through a catalog, clear the screen, it will also set two windows at the top left hand corner of the screen. In this way, once a catalog is performed, the keyboard will appear useless. To remedy the situation, touch the HOME key twice in a row. This will clear the windows and bring your computer back to life.

The reason for the fouled up header is the first character in string FU$, CHR$(141). On the keyboard you can reproduce this one as a capital reversed M. It is the illustrious carriage return, and allows you into a whole new world of program foul ups. If you were to look at Jim Butterfields Super Chart, numerous interesting character combinations can be thought up that will allow you to foul thine hacker. Use your own special combinations of these characters, preceded always by CHR$(141) to royally mess up filenames on disk, REM statements in BASIC programs, and variables within BASIC programs. You can also change the colour of program listings with the right combo of characters. For more information on this marvelous technique, read Jeff Goebels column this issue. Jeff discusses methods to further protect your programs with control characters.

Ye Olde Standbye

One final trick to perform with file names is the old stand by. Look below for a directory entry that can be yours with the right combination of key presses.

```
67    ": filename     prg
```

To get this result, you have to SAVE the program in a little bit different manner. Type in:

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

. . .but press the (shift) key with the space bar between the two colons. This shifted space will fool the disk into believing that the filename is finished, and that it should write the balance outside of quotes. This is due to the fact that filenames are stored on disk padded with shifted spaces, chr$(160)'s. In case you do not already know, 67 is an imaginary block count. Your block count will be what ever size your program takes up on disk.

And A Way And A Way And A Fife And Drum

A rather hasty departure from the DOS you might feel, but quite a bit has just been covered for your protection needs. It is now your job to figure out how best to use the information I have supplied to create your own disk protected land mines.

Auto RUN Programs

Most programs that you find that are auto run are just too simple to break. My favourite method to do this is to reset the start of BASIC to the stack before LOADing in the program. For 4.0 BASIC people try poke 41,1. Once you have LOADed most auto runners, the program will crash with SYNTAX ERROR? displayed prominently on the screen. Now poke 41,4 and you can list the program. At this point some of these programs are rather touchy to play with. Type in 63999 then press (return). You have effectively deleted line 63999, which most likely doesn't exist anyway. This will also allow you to do whatever you like with the program from there on in.

This method is just too easy for what otherwise would be a pretty good technique. Auto boot programs should automatically reset the start of BASIC before executing the auto run code. Jim Butterfields 'Lock Disk', Richard Mansfields 'Bootfixer', and a host of others allow you to make your programs auto run, but they do not reset the start of BASIC. Quite a while ago I modified Richard Mansfields Bootfixer to do this, and today I bring these mods to you. You can find this program in Compute! Magazine of October 1982, Issue 29 on pages 170 & 172. The program is good, but not perfect. First, this program works only with 1541, 2031 and 4040 drives. The reason is simple. On line 100 the variable T=18. To convert to 8050/8250 change this variable to T=38. Next, to produce a better auto run code that is more difficult to break, enter the following lines.

```
475 data 169, 1, 133, 40, 169, 4, 133, 41
510 for pb = 105 to 129 : read by : print#15, " b-p:2 " ;pb
```

Once your programs have been modified with the new Bootfixer, they will be impervious to most attacks. But do not be mislead. There are plenty of other techniques that, while they are not as simple, will break auto run code. Reset all your vectors to computer reset on the execution of your program, and lay a few more traps to confuse the hacker, and you may produce your own version of a hacker cracker. Try tucking valuable code below BASIC, use this code often within program mode, and the hacker will be more prone to leave your program auto run. One final point to remember before releasing your bullet proof wonder. Fire up your favourite wordprocessor and load in your program as a text file. Now look closely at the garbage on the screen. Can you gain any knowledge from this display that would help you break your own code? If your answer is no, pat yourself on the back. If not, foul up the program a little more and try again. Who knows, maybe some hacker will respect your efforts so much that your code will be left alone.

Use Some Of The Other Programs In This Issue

Even if 100,000 people read this article, rest assured that this figure represents only a small fraction of the total number of Commodore users world wide. Not only that, but you also now know that it is easier to protect than de-protect a program, given the right attitude. Protection can be fun, and with plenty of imagination and raw courage, you might tame the savage hacker. Take the extra measure of legal contracts, serial numbers and fine quality manuals, and you might win in this technological war. And take one final step, as explained below.

We have published quite a few articles and programs this issue to protect your creations, so read them with a notebook at hand. Jot down everything that interests you, and cross reference your notes to the pages of our magazine. Take some time with the protection of your program and flowchart it out carefully. Remember that hackers often deserve their title. With careful thought and imagination you might be able to outsmart even the craftiest pirate. The hacker will try every trick that can be thought of to destroy your work. Figure out how you would break your own program, then further strengthen your defenses. But try to keep in mind that the extra time is worth the extra revenue. If your program is poor to begin with, all the protection in the world will not make it any better. What ever you decide to do, be imaginative about it and change your methods of attack from update to update. Just have to keep those hackers on their toes.

Many of you will notice that quite a few tricks remain unsaid in this article. The reason for this is simple. Not everybody is as interested as you are in protection, so why fill an entire issue with one specific train of thought. In future issues I intend to cover this subject further, so write to us and give us your thoughts on the matter. We may have gained a few enemies, but hopefully we have also found many new friends. The purpose of this article was not to harvest a new crop of hackers world wide, but to educate and warn programmers about the extent of piracy today, and how to combat it. I hope that you feel this has been accomplished.

A Final Few Swings At The Insidious Pirate

No matter how much the pirates try to justify their actions, one fact remains. They are thieves, and are stealing from the pockets of their fellow programmers. A theft in any other form would constitute a crime, one punishable by law. This method of theft is over the heads of our best legal minds, so how much legal protection can we expect. The best legal protection in the world will not help in the case of a crime that cannot be proven. Our only true hope is to somehow unite the programmers and hackers into one, thus eliminating the destructive element.

It would not be a surprise if one day we find few new programs coming onto the Commodore scene. Why bother spending mega hours of time and energy in the creation of a truly fine piece of work, only to have your just rewards stolen by an over zealous thrill seeker. Hackers, please take my advice. Write a program that you know will sell, protect the heck out of it and market it. If your hacker friends do not get to it first you may be able to walk away with enough profit to buy yourself a much greater thrill. Buy a Porche and drive yourself into a frenzy. Buy an island and become a recluse. Buy a distillery and become permanently intoxicated. Just buy something that will give you a greater rush than what is experienced by staying up all night staring at your computer screen, bashing away at little tiny keys, and making zero profit for yourself and the company that you ripped off. The acceptance of yourself by your other high tech pirates is nice, but imagine how nice it could be skiing in Switzerland, surfing in California, mountain climbing in British Columbia, racing in Monte Carlo or doing whatever else you find to tickle your fancy. Drop the underground software network and stop stealing from your friends.

Unveiling The Pirate Part 3: The Legal Issue

"To quote me the authority of precedents leaves me quite unmoved. All human progress has been made by ignoring precedents. If mankind had continued to be the slave of precedent we should still be living in caves and subsisting on shellfish and berries." – Viscount Philip Snowden

It is always pleasant to begin an article with some brilliant quotation, just to get the mind moving a little quicker. With a slight update to these words of wisdom, it would read as such. Why do we have to rely on laws designed in days gone past before the advent of modern technology will protect us from infringement of our rights as creators of programming marvels. Virtually every law that has ever been updated can be traced back to the comparison of some antiquated precedent that is somehow misconstrued to apply today. In legal cases where the situation is always the same and always has been, why the heck not. But we are living in a new age, with technology advancing at a rate never before anticipated.

Just think back a few years and compare how quickly everything is advancing today. For this reason it would be nice to find out why the legal system is stuck in the middle ages flogging the same precedents over and over again till nobody really knows how they apply. I applaud Steve Wosniak and his group at Apple for their legal attack on the invasion of the clone people. His group took an antiquated legal system and challenged it to become better. Though the existing laws in most countries are still so relaxed that these crimes can take place, the group at Apple made the legal community stand up and take notice.

The CLONE

With Apple, their problem was with corporations copying their computer, software, manuals, cases, literature and whatever else they could get their sleazy hands on. The companies responsible for these miscarriages of justice were located primarily in Taiwan and Hong Kong, with a little bit of activity in Switzerland and the USA. The way that it appears to have happened is that numerous corporations were set up to clone the Apple Computer complete. Those who have ever started a corporation know that shareholders of a corporation are only responsible for as much money as they have invested in the corporation. The clone corporations made very high profits with the sale of the clones, then declared dividends quarterly to drain all of the money out as quickly as possible. If legal trouble starts brewing up over the clones, dissolving the corporation and starting another was usually the answer. Always one step ahead of the law, these clone people got very rich, and Apple lost out in numerous sales.

If the Apple subject interests you as it did me, then try to get hold of two fabulous books that cover this area extensively. They are available from the publisher direct, but may not be from book stores.

> Software Protection and Marketing
> Computer Programs and Data Bases;
> Video Games and Motion Pictures
> Volumes One and Two
> by Morton David Goldberg – Chairman
> Practicing Law Institute
> New York City
> Course Handbook Series #159 and #160 – 1983

Copyrights And How They Apply To You

Protection under the law for infringements of the rights of software manufacturers is a hopelessly messed up series of mistakes, all tied together in the law books of today. Copyright protection of computer software does apply to a limited degree in the United States, but does not apply in the least in Canada. Elsewhere in the world, many countries are on par with Canada. It seems that the United States is the worlds battle grounds for legal mistakes, with the rest of the world following suit well after all the excitement has died down.

In the United States software is and is not protected by Copyright law for the same reasons. While peering through numerous law books, I have discovered great quantities of hypocritical legal turns of events than should have never occured. For example, lets look at the legal issue of copyright law and how it applies to software.

"(C) COPYRIGHT" – Is It Applicable?

One argument against copyright protection of software is that copyright protection is afforded to matters that relate directly with

human beings. Books, magazines, art and music are just a few of the areas covered by copyright law. With this argument it has been stated that the design and flowcharting work that goes into software development is protected under copyright law. So is the actual source listing of the program. But the moment the program is actually entered into the computer and is activated, it no longer falls under the protective blanket of copyright. This view states that the program no longer relates to human beings, only to the computers for the sole purpose of telling the computer what to do, and recording what responses the computer came back with. The fact that the user related directly with the computer has no bearing on this argument. The computer has become the middle man and therefore excludes the software from copyright protection.

The second argument, this time finding that copyright does apply to computer software, was used with this same example. It was stated that computer software is protected by copyright law because it does relate to human beings right across the line. Computers were designed by human beings, and so are computer programs. Binary 0's and 1's mean absolutely nothing to the computer, only to the programmer. The computer relates directly with changes in voltage and current flows throughout it circuitry. It does not care about binary coding in the least. A programmer could, if persuaded to do so, decipher exactly what a computer program does, once it has been entered into the computer. Whatever language the code is written in, it can be painfully figured out by peering through this apparent machine code. Therefore, this argument states that copyright law does apply to computer software, with all prior arguments being invalid. The courts liked both explanations, therefore no real answer has been arrived at.

Many more instances of copyright law and software battles rage throughout the legal books of today. As it stands, governments worldwide have stated explicitly that they would look into the matter and come to some form of conclusion as soon as possible. The USA have changed their copyright laws a few times, but still to little avail. In Canada we have the white paper, another series of bleeps and blunders to further occupy the courts for many years to come. The legislature has promised new rules are coming, but making them so they won't go obsolete with the technology is the hard part.

Whatever the story, expecting the law to do all the work for you is foolish at this point in time. A bit of thought and careful planning will help you produce a product that may provide a good legal defense if so inclined. Let's now advance into this subject a little deeper.

Legal Avenues To Take

As is obvious, written information can be protected under copyright law. A manual for your program fits into this category. The source listing, flowcharting and all else that has been written down also fits neatly into this little cubical of legal mindset, but doesn't really help the matter at all. Rely on the manual.

The sacred statement "(C) Copyright 1984 Company Name", is a mandatory requirement if copyright protection is desired. This entire statement ensures that the copyright protection that you have opted for applies in most of the countries in the world. Whether or not a complete circle is required around the letter I do not know, but I feel that it should not matter. If the courts go to this extreme to prove something ineligible for protection, then there is something wrong with the courts. Make sure your program displays a copyright notice on the screen at least once during the execution, and write this statement into your manual at least once, just to make sure that everyone knows your intentions.

Create your program in such a way that it requires at least some intelligence to operate it. Write a manual that will illuminate the way for all who attempt to use you creation. Design the manual well, making sure to place a specific serial number in a few key places throughout. Design a legally binding contract between your company and the end user, making sure to state the serial number somewhere prominent on the form. Make sure to write this serial number someplace on diskette where little attention will be generated because of it, and you may have started on your way to partial legal protection. If photocopies of your manual start to appear, and the serial numbers have not been removed, then copyright infringement can most likely be proven in court.

If copies of your program start to appear in a broken state, it may be difficult but not impossible to push the matter in court. The serial number combined with the signed legal contract may be a ticket to recompensation. But often this will not be the case, as I will explain below.

A high percentage of programs are broken mainly for the thrill of breaking them. Once broken, they are quickly spread around from friend to friend, often over the telephone lines, to further weaken your chances for compensation for this crime. Many of the offenses take place in private individuals homes, with the end result being given away to others who share the same sentiments. Once these pirated versions have been passed over the telephone lines, little to nothing can be done to stop its spread. And proving the crime in court is next to impossible, for the offense was probably not witnessed by anyone who will admit to it in court. Checkmate, the pirate wins.

As has been witnessed with the prosecution of video tapes pirates, other legal avenues do exist for protection that are just waiting to be tried in the courts. Though I do not profess to know how this would apply, I have been told that fraud can be proven in this matter, with a jail term and fine applied to any found guilty. This sounds pretty good to me, for it could be applied to anyone caught distributing illegal material, even if the person distributing the material is not the pirate, just someone who managed a copy and wanted to give another copy away to a friend. Talk about a quick way to put a curb in the spread of illegal software.

Legal protection is a single avenue of protection that does not appear to work in a vast majority of cases. Software piracy, unless blatantly obvious as the Apple cases were, is difficult to prove at best, and even more difficult to find laws that will stick. If this fraud situation can be tried and proven in court, then we might finally have some ammunition to work with. But until then, the legal system is not a viable method for software protection in the least.

My final recommendation in this rather volatile situation is to protect the living heck out of your program every which way that you can, and follow the courts as closely as possible. With luck and Providence prevailing, the eternal light may shine down upon the courts and appoint a few computer whiz kid judges. With a computer freak holding the gavel we may finally advance into the computer age as we should have all along.

Piracy vs. Protection: Who Loses?

by Chris Zamara, Technical Editor

"I have read warnings on software packages that state damage to your disk drive may occur if you attempt to make a copy of the program".

A lot of talk is going on about the problem of software "piracy", but a new problem in the computer field is emerging as a result: software protection. Software producers, worried about slipping profits due to unauthorized program copying, are creating major problems for the consumer.

Welcome to the age of the ultra-delicate program: change the system configuration slightly, and it blows up. Plug in a different disk drive – even update the ROMs in your current one, and it refuses to boot (you vill use zee drive zat vee specify or you vill suffer!). Got a handy interface card of some type on your system? Forget it! The program may not like it.

And the way in which programs express their dislike for the environment in which they are living can be frightening. Some will run for a period of time, and then crash at an inopportune (due to Murphy, the worst) time. I have read warnings on software packages that state damage to your disk drive may occur if you attempt to make a copy of the program. I would be very reluctant to buy software from any company making such a claim, even if it isn't true. We are now faced with the situation of the program vs. the user: the software is sitting in the computer, eyeing the user's actions suspiciously. If he should make a wrong move – one which the program, in its infinite wisdom, judges to be an encroachment on its legal space – wham! Reset system, wipe out disk, cook drive, whatever.

The sad result of all of these precautions is that the people who are hurt most are not the computer "pirates", but the paying software consumer. The so-called "pirate", usually no more harmful than a computer enthusiast, is not fazed by such skillful protection schemes. Some pirates "break", or de-protect programs for the fun of it – the harder it is to break, the greater the challenge, and the more fun they have. Other illegal users of these broken programs collect huge libraries of such software just for the sake of collecting it. They would never buy any of the stuff even if they couldn't break the protection. The majority of software users are the computer-naive user: the person who wants a certain program – along with the manual, package, and warranty – and is willing to pay for it. This is the kind of person who knows the least about what program will work with which system configuration and why, and the person most likely to get hurt by a picky and suspicious program. He merely wants to pay a fair price for a program that will do the job. Which brings the next topic to mind: a fair price.

Software companies must obviously make profits on the programs that they sell. But spending many man-hours developing elaborate protection schemes is a waste of time, and obviously raises development costs, ergo, selling price. If that time was spent on the program itself, perhaps the resulting product would be good enough to generate tidy profits from the abovementioned average software consumer. People would be more willing to buy programs which weren't protected to death, and which sold for a reasonable price. Sure, there would be unauthorized copies floating around – probably in the hands of people who wouldn't have bought the program anyway. In the music business, records sell millions (for a reasonable price) even though a tape is so easy to make. Software companies and programmers want to charge huge amounts of money for programs which they claim took years to develop. This usually says more about their lack of programming skill and efficiency than it does about the sophistication of the program.

True, some special purpose packages must sell for a relatively high sum because of the limited market appeal they have. And there is nothing wrong with copy protection per se, providing it has no adverse affects on the ruggedness of the program (a tall order). But until the software producers address the needs of the average software consumer, they will face declining sales due to people who refuse to buy fragile, limited, and hazardous products. And refuse they should.

Spiffy Listings!

Jeff Goebel
Georgetown, Ont.

Most of us have written a BASIC program, or at least typed one in from a magazine. When we are finished we have a mess. If you've ever tried to de-bug someone elses BASIC program, you'll know how hard it is to find anything in a program listing.

Other languages allow you to indent and format listings neatly. Commodore does not. However, it doesn't have to be this way; we can CHEAT with our listings. We can emphasize important points and hide secret sections while generally tidying up the way our LIST looks. In this article I discuss ways to make your listings neater, more colourful, and easier to de-bug. I also describe a few ways to make your listing impossible to list or correct. There are versions for all Commodore computers. All this, and you'll never have to enter the machine language monitor.

First let's start with the easy ones. . . wouldn't it be nice to be able to INDENT certain routines. In more structured languages, all nested loops are pushed right by a few spaces so anyone looking at the listings at a glance can quickly see what is going on. It is possible to do this in Commodore BASIC too. You just have to know how.

TYPE: 10 print " hi

Now LIST. What happened? All those leading spaces are ignored. Now try this:

TYPE: 10 X print " hi

When you list it this time, the shifted X will have vanished and the spaces will not. Don't ask me why or how it works; just be thankful that it does. If we place ANY shifted character in our line, it will be ignored, but the trailing spaces will remain, making it possible for us to indent all we want. Another use for the above system;

TYPE: 10 X X

Again, press SHIFT then X. This time, when we list we see nothing but the statement number. This has little use but it looks nice. Besides, this one will confuse the hell out of people who don't know what you've done. They'll be looking all over to see where you've hidden line 10.

The REM statement of Commodore BASIC is ironically one of the most versatile commands of the language. Oh sure, it doesn't actually DO anything, but we can use it to create some pretty bizarre effects. First of all, we'll start with the simple ones. Have you ever tried including SHIFTED letters in a REM statement? It doesn't work quite the way we expect. For some reason, CAPITAL letters are converted to BASIC text commands when we re-list. This has something to do with the way BASIC is tokenized. Anyway, we don't have to know WHY it does this, all we want to know is how we can use it to our advantage. Try this:

10 rem QWERTYUIOPZXCVBNMASDFGHJKL (Vic & 64)
10 rem QWERTYUIOPLKJHGFDSAZXCVBNM + (Pet series)

Don't be startled when you list it. You'll see a screen full of BASIC commands and you'll be greeted by a ?SYNTAX ERROR. If this is the first line of your program, the computer will stop here. It won't even list to the printer. The key is the last character! On the 64 or Vic, it's a capital L and on the Pets it's a graphic symbol (the cross). All of the other letters are un-tokenized when listed but those two characters have no command equivalents so the computer regards them as an error. Naturally, this doesn't stop people from typing LIST 20- but it may confuse a few. If we want a regular REM statement to include upper case letters or graphic symbols, without causing the effect above, all we need is an opening quotation mark. This allows us to have BOLD STANDOUT REMS that can easily be seen while listing. Therefore, REMs like:

10 rem " (c) 1984 Jeff Goebel " are acceptable.

We can also take this unusual feature one step further. With the right format, we can actually make our listings EXECUTE certain functions when listed. Imagine the possibilities opened up when we can write programs that RUN when we LIST them. Of course it's not quite that glamorous. I can't tell you how to get your lists to do trig functions or elaborate mortgage calculations but we can get it to do some pretty nice screen displays. Let's stop promising and start with the examples. The format is exact so I'll describe it as I go.

TYPE: 10 rem " (RETURN)

1) Cursor up to the spot just after the first quote and type RVS ON (CTRL + 9 on Vic & 64).
2) Now type a SHIFTED 'M' - it should appear in reverse field. This is the key character that makes it all possible.
3) Then type RVS OFF, a quote and the delete key. This puts you back into 'quote mode'. We can now follow this M with any sequence of cursor controls or control characters we want, and when the program is listed they will not only function correctly, but they will not be visible.
4) Let's try a CLEAR SCREEN first. Type the CLR key. A reverse capital S should appear. Now hit return and LIST. BINGO! Your screen clears!
5) Now re-type the line but follow the 'CLR' with a few down cursors and a RVS ON. This time the reverse lower case 'r' will

appear. That's ok. We want it to. Now enter a title like; Jeff's Listing! End off with a few more down cursors and hit return. When we list, we will see the phrase come up centre screen, nicely emphasized in reverse, and the rest of the listing (if there was any) will follow a few lines down.

You are probably beginning to think of many other ways this can be used. Remember, ALL the cursor controls AND control characters can be used. I generally use it to force my listings into lower case (follow the 'M' with a small 'n' to flip to lower case and a shifted 'N' to force upper case) and then I 'lock' the mode by following it with a control 'h' so that people can't flip it back to upper case with the Commodore key. Then I centre a title and underline it with a graphic character using just the right combination of cursor movements.

There are some REALLY neat things you can do on the 8032/ Superpet computers. One these systems, there are extra control codes for setting windows, deleting to end of line, scrolling up and down and even ringing the bell. All of these can be incorporated into the listing. If we set up a nice graphic title page and then set a top left window just below it, all of the program will list UNDER the window. You can keep your name on the screen during the entire list and then re-set the window in the last statement. (Editor's Note: I use the first line of my program to store 'DSAVE"@:PROG NAME"' and use Jeff's technique to make it stay on the top line. When I want to make a disk update, I simply hit HOME twice and Return. Sure saves a lot of typing.)

Since these computers do not have an actual control key, you will have to follow a slightly different procedure.

TYPE: 10 rem " and hit return as described above.

Then cursor up and type RVS, then the shifted 'M'. Now type an 'o' to set the top left or a shifted 'O' to set the bottom right window. A 'g' will ding the bell. Two small 's'es will reset the windows. Here is an interesting example:

10 rem "Mready.MOog

It may take you a few try's to figure out the easiest procedure to get this but after the first time, it becomes simple. If you list this program now, you will see only the '10 rem' and nothing else but when you try to move your cursor, it will appear that your computer has crashed. In effect, all that has taken place is that you have set a window 1 character by 1 character around your cursor. To reset to normal, type the HOME key twice. If this is the first line in your program, the rest of the lines will still list, but nobody will be able to see them because they will be listing only on a 1 x 1 screen. They will still list to the printer and if you want to see them, all you have to do is type: LIST 20-. However, there's no reason why you can't put more of these through your program.

One of the most useful ways to make this work is for colour changes. Since the colours are simply control functions on the 64 or VIC (colours are not available on the Pet series), it is a simple matter of inserting the correct keystrokes after the 'M'. If we want to force a blue listing, simply follow the 'M' with a 'control 7'. This way, we can have all of our subroutines list in different colours which make them really easy to spot. Keep in mind that a blue listing on a blue background is invisible and some other colour combinations are virtually impossible to read. (ie: red on blue) Stick to one or two 'safe' colours and keep alternating.

If we are imaginative, the rems can be used to do some even more incredible things. If we plan our listings, we can use rem statements to help misrepresent portions of our program. For example, new any programs in memory and enter line 10 as sys1024. We can now use a rem in line 20 to make it LOOK like sys2024 if we want. Enter this as line 20:

20 rem " (and hit return)

Now cursor up beside the quotes and type RVS on. Now type the following exactly. All characters should appear in reverse print:

MQQ]]]2q

On the Pet series, add an additional ']'. If all went right, your listing should appear normal except that line 10 now reads sys2024. It will still actually BE sys1024 and listing line 10 by itself will prove this, but go beyond 10 and it will LOOK like sys2024.

Using a similar method, we are able to create totally MOCK statements, or hide portions of statements from view. One of the cursor controls not yet touched on is the DELETE key. It too can be incorporated into our statements. With it, we can change lines totally or vanish any trace of them. Try:

10 print "hi" :rem" and hit return.

As usual, cursor up to the last quote and hit the RVS key. Now type 18 small 't's (19 for pets). Now type RVS OFF and add a MOCK statement like:

20 for t=1 to 1000 : new : next

When you list this, you'll see just the mock BUT when you run it, the REAL line will be executed. You may notice a FLASH of the statement being printed and then deleted but this goes un-noticed if the technique is used in the middle of a program listing. Remember also; YOU know it's there and are looking for it. Others will be somewhat more unsuspecting

I'll leave you to think of NEAT ways to combine these tricks to create either nice neat listings that everybody can read very easily, or tricky nasty listings that are virtually impossible to list. I'm sure there are probably as many other methods as I have described here, and maybe this article will prompt some computer WHIZ KIDS to say; "I can do that!" and submit an article with their 'list spiffers'.

A few quick pokes to play with:

PET: POKE 19,32
C64: POKE 22,35

This vanishes all the statement numbers in a listing without effecting the run. It works with printers too. To reset it, deliberately cause some ?SYNTAX ERROR from the keyboard.

Collecting: Another View

When logging on to a BBS recently, I read a message from a depressed 64 owner who had a few bootleg programs but wanted more. He asked; "Why is it nobody will GIVE me programs? They always want to TRADE!" This got me thinking. Why is it nobody wants to GIVE software away? This thought inspired this article. A look into the "SOFTWARE COLLECTOR". This article is a comparison between the average BOOTLEGGER to the average STAMP COLLECTOR.

Stamp collections hold no useful value to the collector. The stamps may have a financial value but basically, there is no USE for them. They are collected simply as "A COLLECTOR'S ITEM". There is a certain amount of prestige in having something others do not.

It may be surprising to realize but most software collectors do not USE the programs they have. They may play a game now or then, or use the word processor, but often the bulk of the collection is for the same purpose. It gives satisfaction to know they have something that others don't, or that others have had to pay for. This is why you'll find some software collections number in the hundreds: they keep EVERYTHING!

When a stamp collector has a "million dollar stamp" and he trades it, he expects a similarly valued stamp in return. The person who trades for that stamp knows he is getting a "million dollar stamp". It will always be a million dollar stamp, and in five years when he trades it, he'll expect to get a million dollars or more for it. Stamps seldom go DOWN in value.

On the other hand, when the bootlegger has a "HOT PROGRAM"; perhaps a program that's not yet on the market, when he trades it, he still retains a copy so the "market value" goes down one notch. He too, expects something of comparable value in return. This is where the problem lies. In the bootleg there is a specific circuit. Simply put, there are people who know people who know people who know the bootleggers. The closer you are to the actual pirates, the newer and more valuable your programs will be. Unfortunately, the people at the end of the chain seldom have anything new that the others don't. For the "bottom rungers" it becomes a catch 22 situation. They are unable to trade for new programs because they don't have anything new to trade with.

In the bootleg world, items hold prestige only so long. The value of a program is determined by the speed of distribution. After a month or two, you'll find EVERYBODY has it; so it's worthless. Seldom worthless enough to throw it out, but it goes on a disk somewhere, or it becomes a "giveaway" to cousins or friends who just bought a 64. (Irrelevant note: American Express has recently released THE PLATINUM CARD because everybody has the gold card now so it's no longer a status symbol.)

This forms another group of collectors. They are the friends of collectors whose collection is typically three months behind everyone elses. This is where our friend who logged on the BBS lies. He goes to school and is trading for other three month old programs. He's aggravated because he knows there are better things available but he can't get them, until three months from now. The guys at the top laugh at him. I don't think any cases of suicide have been reported yet, but it is certainly a frustrating situation to be in.

Editor's Note:

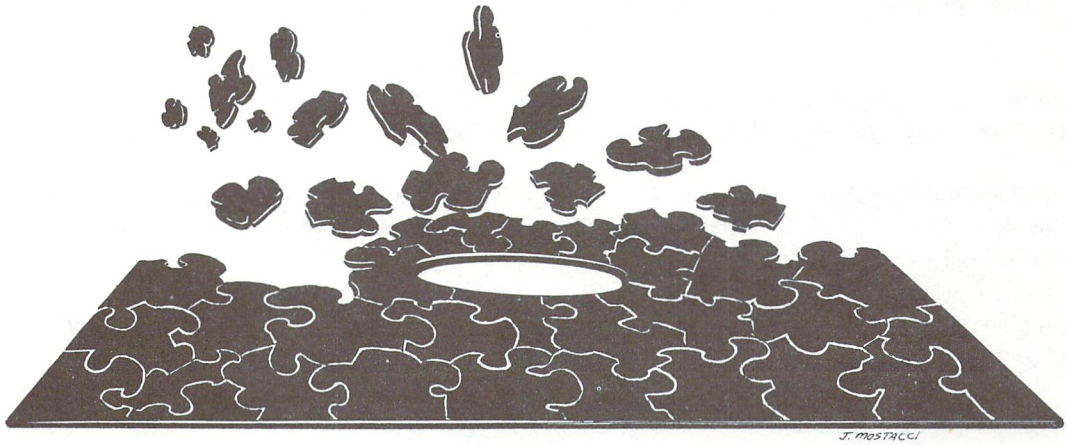
The preceding was submitted anonymously. Although we don't normally publish articles from phantom writers, we felt the information contained would give some insight into the situation we've based this issue on.

Pirates know better than anybody the ramifications of distributing pirated software: fundamentally it boils down to lost profits. The challenge of piracy is not going to "go away". But few pirates see any challenge in distributing to the most number of people. A 'clique' is developing that may even help limit the problem. Perhaps if pirates were to unionize, only a pirate would be able to get a copy of an unprotected program. And since there are only a small number of hackers with enough smarts to crack some of the elaborate protection schemes that some programs host, the overall distribution of broken software would drop considerably.

A fantasy? Probably. But if pirates want to continue unprotecting software, they'll need supply for their twisted habit. If software companies stand to lose their shirts by developing new software, it won't be long before development slows to a standstill. You pirates can help change this bleak outlook for the authors by exercising just a little self control.

Scrambling A BASIC Program

Michael Bertrand
Madison, WI



Michael's is one of a couple of techniques presented in this issue for encrypting a BASIC program file.

These scramble programs codify a given BASIC program into a meaningless sequence of bytes. If the original program is scratched, the codified version is completely secure from unauthorized use. Recovering the original program requires another application of the scramble program, using the same multi-digit seed used to create the scrambled version. Every single byte of the original program is changed in creating the scrambled "program", but in such a way that the procedure can be reversed to recover the original program. The subject programs are read from and written to disk.

There are three different scramble programs:

1) " scramble.bas " — a BASIC version which runs on PET/CBMs and the Commodore 64
2) " scramble.ml " — a machine language version for PET/CBMs
3) " scramble64.ml " — a machine language version for the Commodore 64

The same scramble program must be used in coding and uncoding, since slightly different algorithms are used in the BASIC and machine language versions. The machine language program runs about 12 times faster than BASIC on my CBM/4040 system (about 5 seconds per kilobyte compared to 1 minute per kilobyte). Arithmetic in the machine language versions depends heavily on floating point accumulator ROM routines. Machine language aficionados are invited to disassemble the code, but in what follows I will be discussing the BASIC version " scramble.bas "

Formally speaking, a BASIC program is a finite sequence of bytes, or whole numbers between 0 and 255. In the Commodore DOS, the last 3 bytes are 0, and the first 2 bytes of a program on disk contain the starting load address in low byte– high byte order.

For example, consider the BASIC program:

```
10 print " hello
```

On disk, this program appears as the (decimal) sequence:

1 4 13 4 10 0 153 34 72 69 76 76 79 0 0 0

The first 2 bytes indicate the PET/CBM starting load address of $401 = 1025 decimal. For the Commodore 64, the first 4 would be an 8, indicating a starting load address of $801 = 2049 decimal.

A pseudo–random sequence is a repeatable number sequence whose elements are evenly distributed, as determined by statistical tests, among all possible values. The byte–generating pseudo–random sequence used in lines 300–310 of the " scramble.bas " program is z(0),z(1),z(2),. . ., where:

s(0) = user determined seed value between 0 and 1
s(i+1) = frac(197*s(i)), for i>=0
z(i) = int(256*s(i)), for i>=0

frac(x) is a function found on most programmable calculators, though not in BASIC. The definition is as follows:

frac(x)= the fractional part of x
= x–int(x)

The subscript notation — s(0), z(0), s(1), etc. — is used here for facility of exposition. There are no arrays in the "scramble.bas" program, since these values needn't be saved.

Let's assume, for example, an initial seed value of s(0)=.14159265. Applying the formula yields:

z(0)=int(256*s(0))=int(256*.14159265)=36
s(1)=frac(197*s(0))=frac(197*.14159265)=.8937521
z(1)=int(256*s(1))=int(256*.8937521)=228
s(2)=frac(197*s(1))=frac(197*.8937521)=.069164
z(2)=int(256*s(2))=int(256*.069164)=17
. . .and so on

The point of "scramble.bas" is to scramble the bytes of the original program by offsetting each byte of the program with the corresponding element of the pseudo–random sequence. The scrambled "program" is written to disk and is completely unintelligible. Since the entire scrambling procedure is reversible (using the same seed!), the original program can be recovered whenever necessary.

The following example illustrates the application of "scramble.bas" to the program:

10 print "hello

using the seed s(0)=.14159265.

The first line below is the program, the second line the pseudo–random sequence, and the third line the scrambled "program": that is, the sum of the first 2 lines (modulo 256).

```
1) 1 4   13    4   10    0 153   34   72   69   76   76   79   0 0 0

2)       36 228   17 159 209 244    8   86 134   96 152

3) 1 4   49 232   27 159 106   22   80 155 210 172 231   0 0 0
```

Notice that the first 2 and last 3 bytes are left unchanged. Also, the addition is "modulo 256": that is, if the ordinary sum exceeds 256, then 256 is subtracted to keep the result in the range 0–255.

If the name of the program in line 1 is "hello", then the scrambled program is "hello.s". By applying "scramble.bas" to "hello.s", using the same seed s(0)=.14159265, and choosing the unscramble option, the original program is reproduced under the name "hello.s.s"

```
20 rem * scramble.bas prg — codifies input program
25 rem * p$ on the basis of seed s. the codified pro-
30 rem * gram is written to disk, and has the same
35 rem * name as the original, with ".s" appended.
40 rem * the scrambled program is unintelligible.
45 rem * the 'unscramble' option will re-create the
50 rem * original program if the same seed is used.
55 rem * 'scramble.bas' runs on pet/cbm or c-64.
60 rem *  + + + +  by michael bertrand  + + + +
65 rem *
75 :
100 input"Sseed between 0 and 1";s : print
110 input"program to be scrambled/unscrambled";p$
    : print
120 open 5,8,5,"0:"+p$+",p,r"
    : open 6,8,6,"0:"+p$+".s,p,w"
130 input"scramble or unscramble (s/u)";g$
136 if g$<>"s" and g$<>"u" then print : goto130
140 sg=1 : if g$="u" then sg=-1
154 :
156 rem ** line 160 copies the first 2 bytes, con-
158 rem ** taining the load address, to the new file
160 get#5,z$ : gosub200 : print#6,chr$(z); : get#5,z$
    : gosub200 : print#6,chr$(z);
164 :
166 rem **  the next 2 lines contain the main loop
170 get#5,z$ : gosub200 : if z=0 then 400
180 gosub300 : print#6,chr$(z); : goto170
184 :
196 rem ** subroutine 200 recovers a byte's ascii value
200 if z$="" then z=0 : return
210 z=asc(z$) : return
214 :
296 rem **  subroutine 300 offsets the current byte
298 rem **  and generates the next offset value
300 z=z+256+sgn(sg)*int(256*s)
    : z=z-int(z/256)*256
310 s=197*s : s=s-int(s) : return
314 :
392 rem line 400 is gone to when a 0 byte is encoun-
394 rem tered. three contiguous 0's end the codifying
396 rem process — one or two 0's are offset like
398 rem other bytes, and the main loop is returned to.
400 gosub300 : z1=z
410 get#5,z$ : gosub200 : if z then gosub300 : goto460
420 gosub300 : z2=z
430 get#5,z$ : gosub200 : if z then gosub300 : goto470
440 print#6,chr$(0)chr$(0)chr$(0); : close5 : close6 : end
460 print#6,chr$(z1)chr$(z); : goto170
470 print#6,chr$(z1)chr$(z2)chr$(z); : goto170
```

```
20 rem * scramble.ml program — machine language ver-
25 rem * sion for pet/cbm that runs about 12 times
30 rem * faster than the basic program. the algorithm
35 rem * generating the pseudo-random sequence is
40 rem * similar to the basic version, relying heavily
45 rem * on the floating point accumulators. m/l pro-
50 rem * gram resides at $033c-$03e4 (828-996 dec).
60 rem *    + + + +   by michael bertrand   + + + +
70 :
100 input "{S}input seed between 0 and 1";s : print
110 input "program to be scrambled/unscrambled";p$
    : print
120 open5,8,5,"0:"+p$+",p,r"
    : open6,8,6,"0:"+p$+".s,p,w"
130 input "scramble or unscramble (s/u)";g$
140 if g$<>"s" and g$<>"u" then print : goto130
160 get#5,z$ : gosub280 : print#6,chr$(z);
    : get#5,z$ : gosub280 : print#6,chr$(z);
170 for i=828 to 996 : readx : pokei,x : nexti
    : m=856 : gosub200
180 if g$="u" then poke868,56 : poke869,237
    : rem ** replace adc with sbc in m/l **
190 sys905 : close5 : close6 : end
192 :
194 rem ** subroutine 200-230 puts real number s,
196 rem ** in floating point format, into memory
198 rem ** locations [m,m+1,m+2,m+3,m+4]
200 e=int(log(s)/log(2)) : p(0)=129+e
210 p=(s/2↑e-1)*128 : p(1)=int(p) : r=p-p(1)
220 for i=2 to 4 : p=r*256 : p(i)=int(p) : r=p-p(i) : nexti
230 for i=0 to 4 : pokem+i,p(i) : nexti : return
232 :
280 if z$="" then z=0 : return
290 z=asc(z$) : return
300 data 162,  5, 32,198,255, 32,228,
         255, 72, 32,204,255,104, 96
302 data  72,162,  6, 32,201,255,104,
          32,210,255, 32,204,255, 96
304 data   0,  0,  0,  0,  0,136, 69,
           0,  0,  0,  0,  0, 24,109
306 data  90,  3, 72,169, 88,160,  3,
          32,216,204,169, 93,160,  3
308 data  32, 94,203, 32, 66,205, 32,
           2,206, 32,137,201,162, 88
312 data 160,  3, 32, 10,205,104, 96,
          32, 60,  3,240,  9, 32,100
314 data   3, 32, 74,  3, 76,137,  3,
          32,100,  3,141, 98,  3, 32
316 data  60,  3,208, 27, 32,100,  3,
         173, 99,  3, 32, 60,  3,208
318 data  33,169,  0, 32, 74,  3,169,
           0, 32, 74,  3,169,  0, 32
320 data  74,  3, 96, 72,173, 98,  3,
          32, 74,  3,104, 32,100,  3
322 data  32, 74,  3, 76,137,  3, 72,
         173, 98,  3, 32, 74,  3,173
324 data  99,  3, 32, 74,  3,104, 32,
         100,  3, 32, 74,  3, 76,137,  3
```

```
20 rem * scramble64.ml — machine
25 rem * language version for c-64.
30 rem * identical to code in scram-
35 rem * ble.ml, except for locations
40 rem * of some rom routines. m/l
45 rem * resides at $c33c to $c3e4
50 rem * (49980 to 50148 dec).
55 rem * + +  by michael bertrand  + +
70 :
100 input "{S}input seed between 0 and 1";s : print
110 input "program to be scrambled or unscrambled";p$
    : print
120 open5,8,5,"0:"+p$+",p,r"
    : open6,8,6,"0:"+p$+".s,p,w"
130 input "scramble or unscramble (s/u)";g$
140 if g$<>"s" and g$<>"u" then print : goto130
160 get#5,z$ : gosub280 : print#6,chr$(z);
    : get#5,z$ : gosub280 : print#6,chr$(z);
170 for i=49980 to 50148 : readx : pokei,x : nexti
    : m=50008 : gosub200
180 if g$="u" then poke50020,56 : poke50021,237
    : rem * replace adc with sbc in m/l *
190 sys50057 : close5 : close6 : end
200 e=int(log(s)/log(2)) : p(0)=129+e
210 p=(s/2↑e-1)*128 : p(1)=int(p) : r=p-p(1)
220 for i=2 to 4 : p=r*256 : p(i)=int(p) : r=p-p(i) : nexti
230 for i=0 to 4 : pokem+i,p(i) : nexti : return
280 if z$="" then z=0 : return
290 z=asc(z$) : return
300 data 162,  5, 32,198,255, 32,228,
         255, 72, 32,204,255,104, 96
302 data  72,162,  6, 32,201,255,104,
          32,210,255, 32,204,255, 96
304 data   0,  0,  0,  0,  0,136, 69,
           0,  0,  0,  0,  0, 24,109
306 data  90,195, 72,169, 88,160,195,
          32,162,187,169, 93,160,195
308 data  32, 40,186, 32, 12,188, 32,
         204,188, 32, 83,184,162, 88
312 data 160,195, 32,212,187,104, 96,
          32, 60,195,240,  9, 32,100
314 data 195, 32, 74,195, 76,137,195,
          32,100,195,141, 98,195, 32
315 data  60,195,208, 27, 32,100,195,
         173, 99,195, 32, 60,195,208
316 data  33,169,  0, 32, 74,195,169,
           0, 32, 74,195,169,  0, 32
318 data  74,195, 96, 72,173, 98,195,
          32, 74,195,104, 32,100,195
320 data  32, 74,195, 76,137,195, 72,
         173, 98,195, 32, 74,195,173
322 data  99,195, 32, 74,195,104, 32,
         100,195, 32, 74,195, 76,137,195
```

Two Password Protection Tools

G. Denis
Greenfield Park, Que.

Have you read the papers or been to the movies recently?. Frankly, even if you are living on a deserted island, you have probably heard about the latest avalanche of computer break-ins. You have probably also heard about how easily most of them could have been avoided: by using a password at sign-on time, for example.

Techniques Of Password Protection

This article describes two programs that provide password protection for program and data files.

The first program is a small Basic routine that you insert at the beginning of your own program. Through an option, it will ask for a password the first time it is run. From then on, the only way to list or run the protected program is to enter its password first.

The second program makes a protected copy of any file using a password. The only way to ever use the protected file is to rerun it through the program using the same password.

Using Cryptography

We will be using one of many techniques in Cryptography, the art of concealing the meaning of a message. We will take an existing program or data file, assign it a password (or Key) of up to eighty characters, and run it through an encryption routine that will completely transform it into unrecognizable garble. The result of this procedure is called a cryptogram. It is stored just like the original program or file but the only way to use the cryptogram is to transform it back into its original form. To "decrypt" a file, you must use the original password and run the file through a decryption routine. Only the original password can be used: anything else will garble the program or file beyond recognition.

Back To Basics: The Logical Operators

Before going through the actual programs, let us review some background information. The encryption is done by using the Exclusive Or (XOR) operator. You have probably already used the logical operators (sometimes called Boolean Operators) AND, OR, NOT. The XOR operator is a less commonly used member of the family (in fact, Commodore's Basic does not include a XOR operator although the 6502 machine language does allow for a XOR).

The logical operators perform their corresponding logical operation on each binary digit (bit) of their operands. For example, in C = A AND B, assuming that A, B and C are each made up of 8 bits, bit#0 of C is equal to the result of AND'ing bit#0 of A with bit#0 of B. Bit#1 of C is equal to bit#1 of A AND'ed with bit#1 of B and so on. . . The result of AND'ing two bits is shown in Table 1 as well as the effect of all the other logical operations on all the possible bit combinations.

Table 1

AND	OR	NOT	XOR
1 AND 1 = 1	1 OR 1 = 1	NOT 0 = 1	1 XOR 1 = 0
1 AND 0 = 0	1 OR 0 = 1	NOT 1 = 0	1 XOR 0 = 1
0 AND 1 = 0	0 OR 1 = 1		0 XOR 1 = 1
0 AND 0 = 0	0 OR 0 = 0		0 XOR 0 = 0
Result is 1 if both bits are 1	Result is 1 if either bit is 1	Each bit is complemented	Result is 1 if one or the other but not both

Exclusive OR Properties

An interesting property of the XOR operator is it's so called "reflexivity" property. Yeppp! you heard right, "reflexivity". If we XOR a byte with a fixed value K, we obtain a new value for the resulting byte. And if we take this result and XOR it with the same value K we obtain the original byte value. Example 1 will convince you of this: assume 15 is the value of our byte and 34 is the value K. The result of the XOR operation is 45. If we were to forget that our original value was 15, we can work our way back to it from 45 providing we remember our value K (34).

a) 15 XOR 34 = 45
b) 45 XOR 34 = 15

The reflexivity of the XOR operator is the secret to our encryption/decryption technique: we read in a file byte by byte, XOR each byte with a chosen password (our K value)

and store the new byte in our destination file (cryptogram). The new file is now unrecognizable. Unless we have the key, the file is useless to us. If we have the password, we can restore our file.

This simple technique easily lends itself to further refinements with the use of a sequence of K values that transform successively all the bytes contained in a file.

Great! Now we know how to scramble a file using the XOR operator. Right? Wrong! We don't have an XOR operator in Basic!

Constructing The XOR Operator

If you studied Boolean operators in math, you probably remember that we can build the XOR operator using the AND, OR and NOT operators. Right?. . . Well if you did not, some intuition. . . can give us an answer. Let us look at Table 1. In C = A XOR B, C has the value one if: "A is one and B is zero" OR "A is zero and B is one". More simply:

C = (A AND NOT B) OR (NOT A AND B)

. . .Well, take my word for it. . .

Manipulating BASIC

Listing 1 is the small BASIC routine that is to be inserted at the beginning of any program you want to password-protect. This routine does the actual encryption and decryption of the program following it. Somehow, it has to find out where the protected part of the program is located in memory. In order to understand how it does this, we need to know how BASIC programs are stored in memory.

Programs are stored in the Commodore-64 memory starting at location 2048 ($0800) (see Diagram 1). The first byte is allways a zero byte. Each BASIC line has a two byte long LINK field and a two byte long LINE NO. followed by the actual BASIC code and one NULL byte indicating END of LINE. The LINK field normally contains a two byte address pointing to the next line's LINK field. The end of a BASIC program is indicated by a dummy line with the LINK field containing two NULLS.

Some Useful Addresses

Addresses 43-44 ($2B-$2C) in page zero contain the Start-of-BASIC pointer (a two byte address) to the first line's LINK field. They normally contain the value 2049 ($0801). Addresses 45-46 ($2D-$2E) are BASIC's Start-of-Variables pointer which normally points just beyond the program storage space, where the variables used in the program are stored.

An interesting routine belonging to the BASIC-in-Rom is located at 42291 ($A533) and is called the RE-CHAIN routine. By doing a:

SYS(42291)

. . .you can force all the LINK fields to be recalculated. This routine is called by BASIC whenever you LOAD a program; it ensures that wherever your program is loaded into memory, it's LINK addresses point to the right places.

Now, let's look at how the RUN, LIST, LOAD and SAVE instructions use these different storage spaces and pointers.

LOAD transfers your program starting at the address found in 43-44. When it has finished, it updates locations 45-46. It then executes the RE-CHAIN routine up to the NULL LINK field (end of program).

RUN and LIST use the address stored at 43-44 to begin their operation and will not execute or list beyond the NULL LINK field.

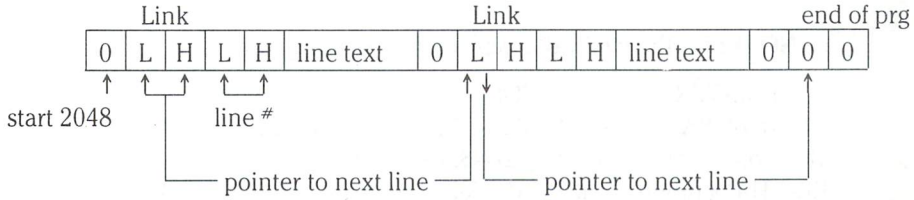
SAVE stores whatever is located between the addresses contained in 43-44 and 45-46. Usually, this would be your program.

As you will soon find out, the program that follows will make clever use of what was just described.

PROGRAM 1: Password Protected Program

Type the routine shown in Listing 1 at the beginning of the program you want to protect. Check for errors and save it.

Diagram 1: BASIC Text Line Structure



Running It

In order to protect or encrypt your program, type "RUN10". There should be some more program lines beyond line 160, as noted in the listing, or there will be nothing to encrypt. You will be asked for a password. Enter any number up to 80 characters. After a few seconds, the message "DONE" should appear. The program is now encrypted and can be saved.

To run a protected program, simply LOAD it and type "RUN". You will be asked for the password under which it was last protected. Typing in a wrong password will corrupt your program and probably the machine. If you type in the right password, the program should soon be executing properly (after it has been decrypted).

When you run the program, you will notice that the password is "invisible" as it is being typed. Line 70 changes the cursor color to blue (the same as the background color) before asking for the password.

You will probably also notice that if you list a program in its encrypted form, only the password routine is visible: the encrypted part will not be listed.

How It Works

Rather than explaining the listing line by line, I will go through it's main functions:

-Determination of the memory location at the end of the password routine (or the beginning of the actual program): Line 160 is crucial in this matter since the two dollar signs (ASCII value 36) at the end of the line will indicate the end of the password routine. Line 20 positions us to the first line's LINK field. Using the Start-of-BASIC pointer, line 30 calculates the address of the next LINK field. We will be jumping from LINK field to LINK field until line 40 detects the two dollar signs at the end of line 160. Variable A now points to the first BASIC line to be encrypted or decrypted.

-Acceptance of the password: Line 70 accepts the password. Line 90 places the password into an integer matrix.

-Determination of the location of the end of the program: Line 100 calculates the position of the last character of the last line using the Start-of-Variables pointer (Remember: BASIC stores its variables just after the end of the program).

-Encryption/decryption routine: Lines 110-120 do the actual cryptographic XOR'ing. As index J is sequentially moving through memory, index I uses each password character value as a new key. Using multiple key values reduces the probability of a "pirate" deciphering the password.

-Hiding of the encrypted program: To make sure the encrypted part of the program does not show when a LIST command is passed, line 130 inserts a dummy NULL LINK value right after the password routine. BASIC is then fooled into thinking the program ends there because of the NULL LINK.

-Re-enabling of the decrypted program: The decryption process resets the NULL LINK value previously set during encryption (last function). Line 140 will replace the dummy NULL LINK by a non zero value. Line 150 calls the RE--CHAIN routine to make sure the LINK field is put back to it's correct value.

PROGRAM 2: File Protection

The program shown in Listing 2 is complete by itself. It will encrypt or decrypt any disk file.

Running It

Type "RUN". You will be asked for the name of the file to be encrypted and the name under which the encrypted version is to be created. To decrypt a file simply pass it through the program a second time using the original password.

How It Works

-Opening of the files: Lines 10 through 70.

-Acceptance of the password: Line 80 accepts the password. Lines 110 to 140 put the password into an integer matrix.

-Encryption/decryption: Lines 150 to 220 input a byte, encrypt/decrypt it and send it out to the new file. Line 210 does the XOR'ing. Similarly to Program 1, index I uses each password character value as a new key.

Conclusion

Although more foolproof encryption methods exist, the one presented here offers the advantage of being effective yet simple enough to be implemented quickly.

NOTE: On the PET/CBM, replace the following addresses:

POINTER	C64	PET
Start-of-BASIC	43-44 ($2B-$2C)	40-41 ($28-$29)
Start-of-Variables	45-46 ($2D-$2E)	42-43 ($2A-$2B)
RE-CHAIN routine	42291 ($A533)	46262 ($B4B6) BASIC 4.0
		50434 ($C442) BASIC 2.0

Listing 1

```
0 goto20 : rem***password decryption
10 s=-1 : rem***password encryption
20 a=peek(44)*256+peek(43)
30 a=peek(a)+peek(a+1)*256
40 ifpeek(a-2)<>36orpeek(a-3)<>36thengoto30 : rem***locate end of password rout.
50 if s=0 and (peek(a)<>0 or peek(a+1)<>0) then goto160
60 if s=-1 and peek(a)=0 and peek(a+1)=0 then print "allready protected" : end
70 input "password: ← ";p$ : print "Z " : if p$="" then 130
75 rem control chars in 70 are blue, then lt. blue
80 dim p(len(p$))
90 for pl=1 to len(p$) : p(pl)=asc(mid$(p$,pl,1)) : next
100 b=peek(46)*256+peek(45)-4
110 for j=a+2 to b : i=i+1 : if i>pl-1 then i=1
120 c=peek(j) : d=p(i) : pokej,(candnotd) or (notcandd) : next : rem*** pokej, 'c xor d'
130 if s=-1 then pokea,0 : pokea+1,0 : print "done" : end : rem***hide program
140 pokea,1 : pokea+1,1 : rem***un-hide program
150 sys(42291) : rem***re-chain basic links
160 clr : rem$$
1000 ********************************************************
1100 print "important!! line 160 must end with two dollar signs
1200 print "your program begins anywhere beyond line 160
1300 ********************************************************
```

Listing 2

```
5 rem***file encryption/decryption***
10 open15,8,15
20 input "source file name,type";f$,t$
30 open 5,8,5,f$+",+t$+",r"
40 gosub1000
50 input "destination file name";f$
60 open 6,8,6,f$+",+t$+",w"
70 gosub1000
80 input "password: ← ";p$:print "Z "
90 print "wait. . ."
95 rem***no password entered forces a zero value key. i.e. no encryption***
100 if p$="" then pl=1 : p(1)=0 : goto150
110 dim p(len(p$))
120 for pl=1 to len(p$)
130 p(pl)=asc(mid$(p$,pl,1))
140 next
145 rem***beginning of copy loop***
150 get#5,c$ : if c$="" then c$=chr$(0)
160 sx=st
170 i=i+1
180 if i>pl-1 then i=1
190 c=asc(c$)
200 d=p(i)
205 rem***c xor d same as (c and not d) or (not c and d)***
210 print#6,chr$((candnotd) or (notcandd));
220 if sx=0 then 150
225 rem***end of copy loop***
230 close5
240 gosub1000
250 close6
260 gosub1000
270 close15
280 end
1000 input#15, e, e$, f$, g$
1010 if e<>0 then printe, e$, f$, g$ : close5 : close6 : close15 : end
1020 return
```

Disk Defender

David Cobb
Windsor, Ont.

This program was designed and written for a C64 with a 1541 disk drive. (see Editor's Note). It allows one to protect and unprotect individual program files. Once a file is protected, only the user who protected the file can have access to the program. The gives the user's disk complete privacy.

The program asks the user to enter a 5 character combination code. It is vital that you remember this code. Since the code is not recorded by the program, if you forget it you will not be able to recover any files protected by that code.

Should you enter an incorrect code you may never see that file again. The program, in such a case, would attempt to reverse the locking procedure using the incorrect code which would compound the encryption. To unlock such a file, it would first have to be unlocked with the incorrect code to create a file that could be unlocked with the original correct code. As you can see, it is necessary to remember the code you make up.

This method may seem harsh on people with bad memories, however it should stop software pirates cold. Without the combination code, protected files are locked up solid. The odds of anyone guessing your code are one in 50 trillion. You have a better chance winning a lottery.

The program protects files by using your combination code. For example, suppose your code is "42345". The last digit 5 is added to the first byte in the first sector of your disk file. Then all the numbers, except the first one (4), rotate one position to the right. The last number 5 is moved to the second position. Now your code is "45234". The cycle then repeats itself, only the last digit 4 is added to the second byte of the sector. The first digit serves as a counter. Every time a cycle is complete the counter is decremented by 1. When it reaches 0 the entire code is reset to its original state (45234). The process continues for the entire sector.

Note: For the first RUN of Disk Defender, use a diskette that contains nothing important. If you make a mistake entering the program you may harm some files.

The user need only remember the original code. The program keeps track of all rotations and alterations on disk. Files are unprotected by the same process, only in reverse. Instead of adding, the last digit is now subtracted from each byte. However, the code digits still rotate right.

Disk Defender will only protect files specified by the user. If Disk Defender is not on the disk containing the protected files, it is a good idea to put it there. Otherwise you will need to LOAD it from another disk and substitute the disk with your protected files before RUNing.

You might even include it as part of another program. Disk Defender will protect any type of file.

Editor's Note: Disk Defender was written for the 64/1541 but with minor modifications will work with any Commodore machine and drive type. The only change required lies at line 10020. Variable T & S represent the first track and sector of the directory. For 4040 drives, leave as is (18 and 0). For 8050 and 8250 change T to equal 39. For hard disk change to T = 0, S = 1.

When entering the program, you'll notice that a lot of the code can be entered by making minor changes to previous lines. Lines 120 to 170 can be entered in no time by merely changing the previous line number and a couple of other characters each line. Notice how lines 200 to 310 are virtually the same as 300 to 410. Same with the first few lines from 10000 on, and 20000 on.

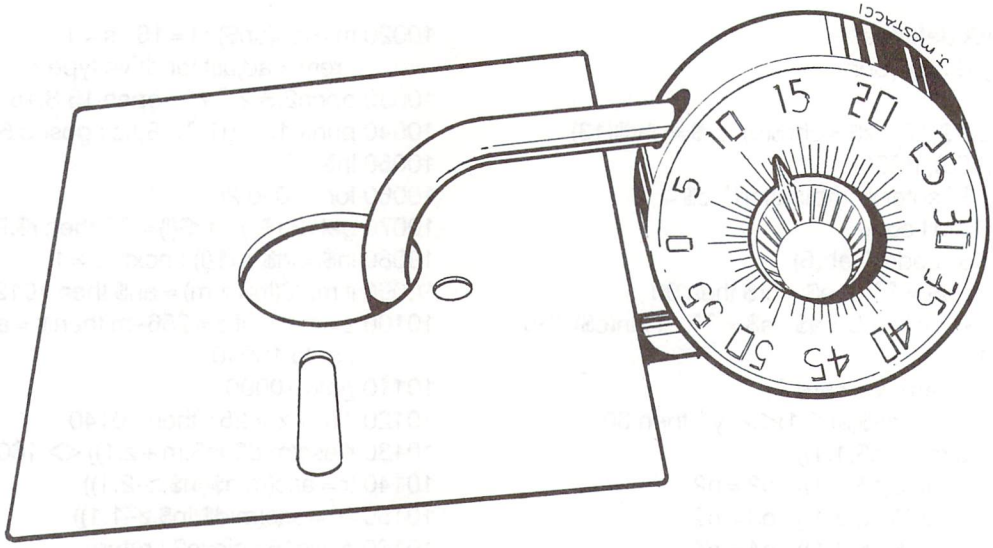
David has chosen to "rotate" the first block of the file only. But since the forward track and sector pointers are also encoded, the disk has no way of knowing where the next sector lies. When programs are involved, 256 bytes can hold a lot of code unless you have 256 bytes of REM statements. But the first 256 bytes of a long SEQ file will be long forgotten when the next block is found. Seasoned pirates, within 5 guesses, could find the next block of a Commodore disk file which would, in this case, lead them to the rest of that file. Perhaps David's program should do all blocks of a file.

Finally, Disk Defender could be used in conjunction with some other protection schemes in this issue to make a program pretty tough to crack.

```
10 rem  disk defender
20 rem  by david cobb
30 rem
40 dw$ = chr$(17) : z$ = chr$(0) :cr$ = chr$(13)
50 dim r$(255), r(256)
60 print dw$ " enter combination " :c$ = " "
70 print  " 5 digit code "
80 print dw$ " code: " tab(6);
90 get a$ : if a$ = " "  or a$ = cr$ then 90
100 print  " * " ;:c$ = c$ + a$ : a$ = " "  : if len(c$)<>5
      then 90
110 print cr$ " are you sure  " ;
111 input an$ : if left$(an$,1)<> " y "  then 60
120 n1 = asc(mid$(c$,1,1))
130 n2 = asc(mid$(c$,2,1)) : o2 = n2
140 n3 = asc(mid$(c$,3,1)) : o3 = n3
150 n4 = asc(mid$(c$,4,1)) : o4 = n4
160 n5 = asc(mid$(c$,5,1)) : o5 = n5
170 print dw$ " lock or unlock file? (l,u) " dw$
180 input an$ : if an$<> " l "  and an$<> " u "  then 170
190 if an$ = " u "  then 320
200 rem *** lock file ***
210 gosub 10000
220 cn = n1
230 gosub 20000
240 for l = 0 to 255
250 r(l) = r(l) + n5
260 cn = cn-1 : if cn = 0 then gosub 40000 : goto 280
270 gosub 40500
280 if r(l)>255 then r(l) = r(l)-256
290 next : gosub 30000
300 print  " file:  " an$ "  is locked. "
310 close2 : close15 : end      : rem ** end of lock **
320 rem *** unlock file ***
330 gosub 10000 : cn = n1 : gosub 20000
340 for l = 0 to 255
350 r(l) = r(l)-n5
360 cn = cn-1 : if cn = 0 then gosub 40000 : goto 380
370 gosub 40500
380 if r(l)<0 then r(l) = r(l) + 256
390 next : gosub 30000
400 print  " file:  " an$ "  is unlocked. "
410 close2 : close15 : end      : rem ** end of unlock **
10000 rem * find track & sector of file *
10010 print  " enter name of file "  : input an$
10020 m = len(an$) : t = 18 : s = 1
        : rem * adjust for drive type *
10030 open2,8,2, " # "  : open 15,8,15
10040 print#15, " u1: " 2;8;t;s : gosub 50000
10050 ln$ = " "
10060 for l = 0 to 254
10070 get#2,r$(l) : if r$(l) = " "  then r$(l) = z$
10080 ln$ = ln$ + r$(l) : next : z = 1
10090 if mid$(ln$,z,m) = an$ then 10120
10100 z = z + 1 : if z = 256-m then s = asc(r$(1))
        : goto 10040
10110 goto 10090
10120 if m + z > 256 then 10140
10130 if asc(mid$(ln$,m + z,1)) <> 160 then 10100
10140 tr = asc(mid$(ln$,z-2,1))
10150 sc = asc(mid$(ln$,z-1,1))
10160 close15 : close2 : return
20000 rem * read track & sector *
20010 open2,8,2, " # "  : open 15,8,15
20020 print#15, " u1: " 2;8;tr;sc : gosub 50000
20030 for l = 0 to 255
20040 get#2,r$(l) : if r$(l) = " "  then r$(l) = z$
20050 r(l) = asc(r$(l))
20060 next
20070 close15 : close2 : return
30000 rem * write track & sector *
30010 open2,8,2, " # "  : open 15,8,15
30020 print#15, " b-p: " 2;0 : for l = 0 to 255
30040 r$(l) = chr$(r(l))
30050 print#2,r$(l); : next
30060 print#15, " u2: " 2;8;tr;sc
30070 gosub 50000 : return
40000 rem * check code *
40010 cn = n1 : n2 = o2 : n3 = o3 : n4 = o4 : n5 = o5
40020 return
40500 rem * rotate user code *
40510 oo = n5 : n5 = n4 : n4 = n3 : n3 = n2 : n2 = oo
40520 return
50000 rem * error check *
50010 input#15, a, a$, q1, q2
50020 if a = 0 then return
50030 print  " error  " ;a
50040 print a$
50050 print  " track  " ;q1;"  sector  " ;q2
50060 close15 : close2 : end
```

LockDisk: Force RUN On LOAD

Program by Jim Butterfield
Presented by Karl J.H. Hildon, Managing Editor



The only way to get programs to benefit from all those POKEs you put inside is to make them execute those POKEs. You can add all the protection in the world but if LOAD and SAVE is still allowed, you've accomplished nothing.

LockDisk is a utility that adds data to the beginning of a program file. This data has been carefully chosen.

The first two bytes of a program file on disk represent the address at which the LOAD routine will place that file in memory. LockDisk changes those bytes to a spot well beneath the Start of BASIC text space. When the new doctored file comes into the machine, several delicate memory locations get clobbered by the added data. After LOAD does its part, the machine is literally taken by surprise. Instead of giving control back to the keyboard, the machine takes an abrupt detour straight to RUN.

Naturally, LockDisk disables the STOP key. Jim does some other really nasty things to the operating system too. But if we reveal too much about LockDisk, its potentcy will be severely diluted.

Two Versions

LockDisk is listed below for both the Commodore 64 and PET/CBM machines. Sorry, no VIC version; too many memory configurations.

Both versions are used the same way. You'll be asked for the name of the program you want locked. LockDisk checks to make sure this file is a normal program by testing the start address. If the low byte is not CHR$(1), LockDisk quits.

Then you supply a new program name as the title of your locked file. The 64 version assumes you have a single drive; the PET/CBM version will let you pick the drive number if you have two. LockDisk takes off from there.

The Finished Product

Once done, try LOADing the new file. If you've written your program with no vulnerable INPUT statements or anything else that might relinquish control, chances are you'll need to power down to get your machine back.

Unlike PET/CBMs, the 64 has a couple features that forces LockDisk to be a little craftier. To make the new locked file do an auto RUN, a non-relocating LOAD must be specified:

```
LOAD "LOCKED PROGRAM",8,1
```

If you don't, the LOAD routine will ignore the new start address, relocate the file away from the hot spots, and the auto RUN is defeated, you say. Not quite, hacker breath! LIST it and see.

OK. Now hit RUN/STOP-RESTORE. Thought you had it licked, eh? Maybe next time.

A Humble Start

LockDisk won't last long against the seasoned pirate. But with enough extras you'll be able to keep him busy for a while. And for the average user, LockDisk will quickly discourage any dubious intentions.

LockDisk isn't above improvement. The deeper you go into lower memory, the more you can add to your protection efforts. Remember, there's more pointers down there than you can shake a stick at. And if you change them, have your program check for those changes. If it sees they've been set back to normal, anything from a warning to intense cruelty is optional. Just be sure about it though. You don't want to go newing disks that belong to honest users with equipment you haven't accounted for.

Above all, don't short change yourself. LockDisk is only one utility you can take advantage of. There are lots more.

LockDisk for the Commodore 64

```
5 print " auto start 64 – jim butterfield "
10 open 15,8,15
20 input " name of program " ;n$
30 open 1,8,3,n$ + " ,p,r " : input#15,e,e$,e1,e2
40 if not e then get#1,a$,b$ : if a$<>chr$(1) then e = 1 : e$ = " oops! "
50 if e then printe$ : close1 : stop
60 input " name of converted program " ;c$
70 open 2,8,4, " 0: " + c$ + " ,p,w " : input#15,e,e$,e1,e2
80 if e then printe$ : close2 : stop
90 data 192,2,0,8,1,0,147,34
100 for j = 0 to 7 : readx : print#2,chr$(x); : nextj
110 for j = 1 to len(c$) : print#2,mid$(c$,j,1); : nextj
120 data 34,44,56,44,49
130 for j = 0 to 4 : readx : print#2,chr$(x); : nextj
140 for j = 0 to 52–len(c$) : print#2,chr$(0); : nextj
150 data 139, 227,  52,   3, 124, 165,  26, 167
160 data 228, 167, 134, 174,   0,   0,   0,   0
170 data  76,  72, 178,   0,  49, 234, 102,   2
180 data  71, 254,  74, 243, 145, 242,  14, 242
190 data  80, 242,  51, 243,  87, 241, 202, 241
200 data 237, 246,  62, 241,  47, 243, 102, 254
210 data 165, 244, 237, 245,  32,  89, 166,  76, 174, 167
220 for j = 0 to 57 : readx : print#2,chr$(x); : nextj
230 for j = 0 to 1221 : print#2,chr$(32); : nextj : print#2,chr$(0);
240 get#1,a$ : sw = st : if len(a$) = 0 then a$ = chr$(0)
250 print#2,a$; : if sw = 0 goto 240
260 close1 : close2 : close15
270 input " want to do more programs " ;a$
280 if a$ = " y " or a$ = " yes " goto 10
290 sys peek(65532) + peek(65533)*256
```

LockDisk for the PET/CBM

```
100 print chr$(147) " run–only    (c) 1981 jim butterfield " : open3,0
110 close1 : close15 : print " basic program to protect? " ; : input#3,n$
120 print : open15,8,15
130 open1,8,3,n$ + " ,p,r "
140 input#15,dz : if dz<>0 goto110
150 get#1,a$,b$ : if a$<>chr$(1) and b$<>chr$(4) goto110
200 close2 : print " name for protected version? " ; : input#3,p$ : print
210 print " write to drive? 0 " chr$(157)chr$(157); : input#3,d$ : d = val(d$) : print
220 if d<0 or d>1 goto210
230 open2,8,4,chr$(d + 48) + " : " + p$ + " ,p,w "
240 input#15,dz : if dz<>0 goto200
300 for j = 0 to 2 : print#2,chr$(j); : nextj
310 for j = 1 to 255 : print#2,chr$(2); : nextj
320 for j = 1 to 3 : print#2,chr$(0); : nextj
330 readj : if j>255 goto400
340 n = n + 1 : print#2,chr$(j); : goto330
400 for j = 515 + n to 1024 : print#2,chr$(peek(j)); : nextj
410 get#1,x$ : sw = st : if x$ = " " then x$ = chr$(0)
420 print#2,x$;
430 if sw = 0 goto 410
440 close2 : close1 : close15
450 print " want to do more? " ; : input#3,z$
460 if asc(z$) = 89 then run
470 syspeek(65532) + peek(65533)*256
700 data 165,144,164,145,16,12,24,105,3,144,1,200,141,130,2,140,131,2
710 data 162,18,189,84,2,157,111,2,202,16,247,154,169,1,72,72,72,72,72
720 data 169,122,160,2,120,133,144,132,145,88,169,5,133,158
730 data 165,40,133,42,165,41,133,43,160,0,162,3,177,42
740 data 230,42,208,2,230,43,201,0,208,242,202,208,241
760 data 108,148,0
770 data 147,82,85,78,13,0,0,0,0,0,0,32,234,255,169,255,133,155,76
780 data 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,108,252,255,256
```

Drive Protect

Richard T. Evers, Editor

YOU JUST BIT INTO YOUR DISK AND INSERTED YOUR TOAST INTO THE DISK DRIVE

Drive Protect has been written for everyone who has ever lost important information on diskettes, due to accidental scratching of files, or wiping out of entire disks. The program below will help stop these little misadventures, and also provide you with a few more avenues of protection as an added benefit.

By using Drive Protect you can perform up to three differant forms of protection that you probably didn't know were possible. The first will allow you to SCRATCH protect any files that you choose, for the specific purpose of thwarting the cloddish people who make lending out disks a horror. With this feature enabled, SCRATCHing will be disabled, but SAVEing @, LOADing, reading from and COPYing are still allowed. Not an earth shattering event in the protection department, but still one that does come in handy when the cards are not in your favour.

The second form of protection is one that will be usefull to a very small minority of readers. This feature will enable you to change the block count of specific files on disk. Though it may appear to have few immediate protection benefits, for the block count will appear correct on the new copy if copyied, it does allow you to purposely confuse the heck out of a lot of would be file snoopers. In the game of protection, confusion can be truly effective.

The third and final form of protection available is one that I feel will benefit most who use it. It rewrites your diskette in such a way that you cannot duplicate it easily. This also means that you can no longer SAVE to the disk, or write to it, along with not being able to BACKUP at all. But you still can COPY from it. This is accomplished by changing the DOS version identifiers on the disk surface. The data pertaining to this procedure can be found in the following chart.

Drive	ID	Index	Location
1541	65	2	Track 18, Sector 0
2031	65	2	Track 18, Sector 0
4040	65	2	Track 18, Sector 0
8050	67	2	Track 38, Sectors 0 & 3
8250	67	2	Track 38, Sectors 0, 3, 6, & 9

As the chart shows, the DOS ID is normally a 65 ascii for the 1541/2031/4040 drives, and 67 ascii for the 8050/8250 machines. A change of this value on the disk surface, on the track & sector(s) specified, with an index of 2, which is the third byte from the start, will produce the desired effect. If these points seem too much for you to remember, don't worry. Drive Protect has been written to do all the work for you. The only action required of you is to key the program in, RUN it, then specify your intent. With a groan of your disk drive, and a flash of its lights you will find your diskette takes on a new shape (essentially, not physically). It is for this reason that I suggest you use a test diskette until you're sure Drive Protect is working properly.

Further variations can be made to this program to allow you to rename your files, change the pointers to the next directory block, and the pointers to the first data block that applies to the file chosen. These little improvements I leave entirely at your discretion. If you create some twists, drop us a letter and let us know how you managed it. Chances are if you found your twist imaginative, some of our other readers would also.

```
100 remark * richard evers - june 1984
110 remark *       drive protect
120 remark * - scratch protect your files.
130 remark * - change the block count of files.
140 remark * - back up protect your diskette.
```

```
145 cs$ = chr$(147) : dc$ = chr$(17) : ry$ = chr$(18) : rn$ = chr$(146)
150 print cs$ " drive protect - transactor magazine "
155 print dc$ " specify drive type " dc$
160 print " ( 1 ) 1541/2031 "
165 print " ( 2 ) 4040 "
170 print " ( 3 ) 8050 "
175 print " ( 4 ) 8250 "
180 print dc$ " your choice : " ;
185 input dt : if dt = 0 or dt>4 then 180
190 if dt = 1 then dt = 18 : dl = 144 : dh = 2 : di = 4 : rv = 1 : pt = 18 : ps = 0 : rem 1541/2031
195 if dt = 2 then dt = 18 : dl = 150 : dh = 67 : di = 4 : rv = 1 : pt = 18 : ps = 0 : rem 4040
200 if dt = 3 then dt = 39 : dl = 96 : dh = 67 : di = 8 : rv = 2 : pt = 38 : ps = 0 : rem 8050
205 if dt = 4 then dt = 39 : dl = 96 : dh = 67 : di = 8 : rv = 4 : pt = 38 : ps = 0 : rem 8250
215 print dc$ " protection : (1) file or (2) disk " ;
220 input fd$ : if fd$ = " 2 " then 475 : rem * disk protect
230 input " drive#, filename : " ;d$,f$
235 if len(f$)>16 or d$<" 0 " or d$>" 1 " then 230
240 input " scratch protect file (y/n) " ;sp$ : sp = sp$ = " y "
245 input " change block count (y/n) " ;cb$ : cb = cb$ = " y "
250 if cb = 0 then 280
255 input " change to what value " ;nb
260 if nb > 65535 then 255
265 ch = int(nb/256) : cl = nb - ch*256
270 :
275 rem *** check if file exists - continue on if so
280 open 15,8,15
285 open 5,8,5," " + d$ + " : " + f$ + " " : get#5,a$ : if st then 460 : rem * not there
295 print#15, " m-r " chr$(dl)chr$(dh); : rem * find sector of file in directory
300 get#15,s$ : sec = asc(s$ + chr$(0))
305 print#15, " m-r " chr$(dl + di)chr$(dh); : rem * find index within directory
310 get#15,i$ : ind = asc(i$ + chr$(0))
315 :
320 open 6,8,6," # "
325 print#15, " u1 " ;6;val(d$);dt;sec; : rem * set up to read the data
330 print#15, " b-p " ;6;0; : rem * position the buffer to the start
335 get#6,pl$ : if pl$ = " " then pl$ = chr$(0) : rem * next dir block low
340 get#6,ph$ : if ph$ = " " then ph$ = chr$(0) : rem * next dir block high
345 if ind = 2 then 365 : rem * it's the first file in
350 for x = 2 to ind-1 : get#6,a$ : ss$ = ss$ + chr$(asc(a$ + chr$(0))) : next x
355 :
360 rem *** time to bring in the actual data about the file chosen
365 get#6,ty$ : if ty$ = " " then ty$ = chr$(0) : rem * file type
370 get#6,tr$ : if tr$ = " " then tr$ = chr$(0) : rem * first data track
375 get#6,se$ : if se$ = " " then se$ = chr$(0) : rem * first data sector
380 for x = 1 to 16 : get#6,a$ : na$ = na$ + chr$(asc(a$ + chr$(0))) : next x
385 for x = 1 to 9 : get#6,a$ : bs$ = bs$ + chr$(asc(a$ + chr$(0))) : next x
390 get#6,bl$ : if bl$ = " " then bl$ = chr$(0) : rem * block count low
395 get#6,bh$ : if bh$ = " " then bh$ = chr$(0) : rem * block count high
405 if sp then ty$ = chr$(asc(ty$)or64) : rem * set bit 6 for scratch protect
410 if cb then bl$ = chr$(cl) : bh$ = chr$(ch) : rem * change block count
415 :
420 rem *** and finally, bring in the balance of the data
425 if ind = 226 then 440 : rem * data already read in
430 for x = ind + 30 to 255 : get#6,a$ : es$ = es$ + chr$(asc(a$ + chr$(0))) : next x
440 open 7,8,7," # "
445 print#15, " b-p " ;7;0; : rem * position the buffer to the start
450 print#7,pl$;ph$;ss$;ty$;tr$;se$;na$;bs$;bl$;bh$;es$;
455 print#15, " u2 " ;7;val(d$);dt;sec; : rem * let it know where to write
460 close 5 : close 6 : close 7 : close 15 : rem * and close it all up
465 print chr$(147) " file protection complete !! " : end
470 :
475 print chr$(147) " disk protect - stops the backup blues "
480 input " drive number " ;d : if d>1 then 480
485 open 5,8,5," # " : open 6,8,6," # " : open 15,8,15
490 for lp = 1 to rv
495 print : print " pass " lp " of " rv " : track " pt " sector " ps
500 print#15, " u1 " ;5;d;pt;ps : print#15, " b-p " ;5,0 : rem * read from ch#5
505 print#15, " b-p " ;6,0 : rem * write to ch#6
510 b$ = " " : for x = 0 to 255
520 : get#5,a$ : a$ = left$(a$ + chr$(0),1) : if x = 2 then a$ = " * "
525 : b$ = b$ + a$ : if x = 128 then c$ = b$ : b$ = " "
530 next x
535 print#6,c$;b$; : print#15, " u2 " ;6;d,pt,ps
540 ps = ps + 3 : next lp
545 print#15, " i " + mid$(str$(d),2) + " "
550 close 5 : close 6 : close 15 : print chr$(147) " disk protection complete !! " : end
```

DiskMod: Examine Diskettes Sector By Sector

Program By Jim Butterfield
Presented By Karl J.H. Hildon

If you've been lucky enough to never have a diskette go bad on you, then perhaps you should stop reading this article. Kinda like hitting your thumb with a hammer – it never happens until you're thinking about it.

The fact is, diskettes do go bad. Not just Read Errors, but chain pointers get mixed up, directories mysteriously drop filenames, and files clobber each other by fighting for the same sector. Why? Oh sure, sometimes it's the programmers' fault. But I've known my drives to gobble a file or two on me before – and at least once with no program present at all.

Once I tried to COPY a file from one drive to the other on my 8050. The next Directory I did showed me a full disk with only the first 8 filenames. There were no Read Errors. In fact I recovered everything. Using DiskMod I simply called up the sectors of the directory track, fixed the pointers, and everything was back to normal. Then I took a backup.

Read Errors are tough to recover from even at the best of times. Sometimes you can try the other drive, or another unit, and find it goes away. But a true Read Error, for instance a faulty disk surface, is virtually impossible to beat. However, if the damage is limited to only a small portion of the disk, DiskMod will let you sneak around and salvage what you can.

Consider a SEQ that contains mailing list data. The file is, say, 100 blocks long. The weather is sunny, you start printing your list, and ZAP! A bolt of lightning comes out of nowhere and cooks the second sector of the file. You get to see what's in the first sector, but it can't get past the second sector to find the third. After you try this 6, 7, maybe 8 times, you start to get a little nervous, right? (don't worry, this hasn't happened to our mailing list, yet)

Well hold it right there. A **little** nervous is all you should get. At best you've only lost 1 block, not all 100! (At worst you've lost all 100 but we're not gonna talk about it) Pull out DiskMod. Your disk error routine should tell you where to start. You know:

```
50000 open 1, 8, 15 : input#1, e, e$, et, es
50010 print e$, "track:"; et ", sector:"; es
```

. . .or with BASIC 4.0 just print DS$. Run DiskMod and give it a drive number. Then use the values for ET and ES as your first response to:

track, sector ?

DiskMod will do it's best to read that sector. If it can't, don't lose hope. There are still 98 blocks to go. The idea here is to just make sure it's actually an error. If DiskMod does read the sector successfully, there may be another reason for your trouble. We'll assume there was an error for now. (We'll get to non–read error problems in a minute).

Read the first sector of the directory.

```
4040: 18, 0
2031: 18, 0
1541: 18, 0
8050: 39, 0
8250: 39, 0
```

Note: DiskMod automatically adjusts for your screen size. Make sure the first lines of both programs are entered as shown, including cursor movements. Since there are two versions of DiskMod following, I'll be using general terms to describe the operation. I'll get more specific later on.

DiskMod will display the contents of the sector. The BAM (Block Availability Map) is always first. DiskMod will report the coordinates of the next sector. All you need do is respond with N for Next and DiskMod will read it and display it. Keep going until you see some filenames you

recognize, especially the one that contains the error. Look carefully at the display. All numbers are shown in hexadecimal for the sector contents. Notice the first two bytes of the sector represent the track and sector of the next block of the directory, in hex of course. The first byte after them is the file type byte. After that is the first track and sector of the file, and beyond that the file name itself. Check the tables on the next page for Directory Formats.

Using the files' track and sector bytes, issue an R for Read and enter them in. Remember you'll need to convert to decimal yourself (ie $13 becomes 19 dec). DiskMod will now show the contents of the first block of our mailing list. Once again, the first two bytes of a sector represent the track/sector coordinates of the next sector (unless the track is zero, that means it's the last sector) But when we try to show the next sector, we're gonna run into that nasty Read Error. Instead, we'll go around it to the sector after that. But how? The second sector contains those coordinates and we can't read them.

You may have noticed by now that sectors are allocated approximately 3 apart each time. This is how DOS uses up blocks – about 3 apart. If the DOS used consecutive sectors the data transfer rate would suffer speed loss. Imagine you're a drive head with a diskette spinning underneath you. You read one block and determine where the next one is. But by the time you do that, the next sector has already gone by and you have to go all the way around once to get there again. By spacing them 3 apart, the DOS has just enough time to read the block and prepare for the next one just as it's coming into view. Clever eh? When the end of a track is reached, DOS goes back to the beginning of the track and starts using the inbetween blocks, again about 3 apart each time. When the whole track is full, DOS starts again with another track that is usually not too far away. Back to our problem.

So the next sector is probably not far away. Usually it's not hard to recognize the data once you find it. Record the coordinates. What we do now is go back to the sector previous (the first sector in this case), and change the forward coordinates to point at the third sector, effectively jumping over the bad block. You may have lost 1 block, but at least now you can get to the remainder of the file. You'll have to rebuild the lost data manually.

Complications come up when a diskette gets somewhat full, especially after a lot of Scratching files or Saves with replace have been done. When a disk gets almost full, the DOS can litterally scatter a file all over the disk as it fills the last remaining free sectors. You probably have diskettes right now like this. Tracing these files once one block goes bad can get irritating at best.

Jim has another program he has yet to release into public domain; Disk Dupe. DiskDupe takes an otherwise ruined diskette and salvages whatever it can onto a fresh disk. Then, with his usual talent for writing programs approaching artificial intelligence, Jim builds a new directory after searching out lost files. The program examines the forward chain pointers of each sector and, based on the length of the chain, decides whether it could be a file worth recovering. An artificial directory entry is made that points to the recovered data allowing you access to it from the rebuilt copy of the disk. Perhaps Jim will let us publish DiskDupe?

Non Read Error Loss

Remember what I said before? Sometimes a disk can get clobbered even though there is no Read Error anywhere on the surface. Who knows why but it happens. The directory is suddenly missing files that you damn well know are there!

Also, information can be written on disk without a directory entry, ie. direct access files. You may want to examine them too, or any other perfectly good sectors for that matter.

What do you do? Right. Pull out DiskMod. But first, make a backup. No sense making mods to a disk when it might lead to more trouble that could have been avoided. With a backup you can start over again if you have to.

A quick check of the directory track will no doubt show you the problem. Chances are one directory sector has been pointed around the block containing the missing entries. In my case, the BAM was pointing deep into the directory track at the last directory block. I traced the chain from the block that DOS normally uses as the first block for the directory. Eventually I reached the last block (track pointer equals zero) which was the only one showing. Everthing seemed OK so I went over to the BAM block and altered its forward pointer to the first block. Back to normal in five minutes.

Easy To Use

DiskMod is as easy to use as the BASIC screen editor. Once a sector is displayed, DiskMod will prompt you for a command. If you want to change the contents of a sector, ignore the command prompt and cursor up/over to the byte you want. Type the new value right over the old, being careful not to change the length of the display line with any accidental Deletes or Inserts. Hit Return and DiskMod will write the new values back to the same block and display the block all over again. You can change all or part of the line, but only one line at a time. Remember, all values are in hex, but if you make a mistake simply change it again.

DiskMod will only display as many bytes as will fit on your screen. The first line of each program does a test for screen size. This is why it MUST be entered exactly as shown. If you have 80 columns, you'll see the whole sector. With less than 80, DiskMod will show only part of the sector. To see the rest enter S for Swap. Naturally, the S command is not recognized with 80 columns.

Caution: Almost all protective mechanisms are bypassed by DiskMod. Make sure you don't try to modify diskettes formatted on an alien drive. That is, don't write to 1541s with a 4040, and vice versa, etc.

Also, if you try to access tracks that don't exist (like track 50 on a 4040) you will hang up.

Two Versions

Two versions of DiskMod are listed below. The first is an all purpose version that is pure BASIC. It will work with just about any Commodore machine and drive type. The second will work only with BASIC 4.0 machines and 8050 or 4040 drives (IEEE 2031 might work but hasn't been tested). This version contains machine language making it considerably faster than the all BASIC DiskMod. However, unlike version 2, version 1 can be modified - version 2 will have to be used as is.

All Purpose Version

First you'll be asked for the drive. If you have a dual IEEE unit, enter 1 or 0. For single drives enter the letter 's'. Don't be alarmed by the disk activity that immediately follows. This is only an initialize command and is perfectly normal.

Next you'll be asked for Track and Sector. Enter these in decimal. For ideas take a look at the charts following the article. To Quit, enter "0, 0".

If the block you read has a valid next sector, you can enter N for Next. DiskMod will take you there. Otherwise, enter R for Read and supply new track and sector coordinates.

Again, to change as block, use the screen like a "sector editor". Remember, one line at a time.

BASIC 4.0 DiskMod

This version has a few more commands than the last one. First, Map will display a graphic Block Allocation Map. The asterisks indicate allocated sectors. Hitting any key returns the block display.

Use and Free will allocate or de-allocate the block you are looking at. Allocate all you like, but remember, when you Free a block you're telling the DOS that it's OK to use next time it goes looking for a place to store something.

Next takes you to the next block in the chain, if there is one. Watch it. You are not completely protected against non-existent tracks or sectors.

Read lets you pick your own Track and Sector. Use Q to Quit.

Entering BASIC 4.0 DiskMod

Unlike the BASIC version, this has a machine language module that lives just above the BASIC portion. Let's start with that.

Type in the program with all those DATA statements. Fix any mistakes, SAVE it, and RUN it. A program file called "MACH PART" will be written to drive 0 that we'll be using later.

NEW and enter the BASIC part. The listing shows lots of spaces mostly for neatness. Omit them. When you're finished, PRINT FRE(0). If the free space is 28417 or less, it's too big. Go back and remove some spaces between commands until FRE(0) is 28417 or greater. If you have a programming utility stuck in high RAM, use the formula below. It will adjust for the lower Top of Memory pointer that will affect FRE(0).

```
print 31740-peek(52)-peek(53)*256 + fre(0)
```

Small enough? Good. SAVE it just to be safe. Now LOAD the machine language PRG file "MACH PART" that was created by the last program. It will drop into memory just above the BASIC part. Now:

```
save"0:DiskMod 4.0",8
```

This will write both the BASIC and Machine language parts to disk as one program. If you want, LIST the program right to the end. If you did anything wrong it won't go unoticed, believe me.

The finished file should take up 16 blocks on your disk. With a fresh machine, LOAD it back and you're ready to go. Don't try to make any changes though. If you do, the machine language will shift in memory and you're in for a crash. Of course if you get Syntax errors or anything, you'll have to make changes. After that, you must repeat the final building procedure before attempting to RUN it.

In Closing

Would someone like to convert the BASIC 4.0 version to the C64? It's not as simple as just changing the machine code. 1541 disks have several internal changes compared to the earlier IEEE drives. For clues, take a look at how Jim handles the difference between drive '0' and drive 's' in the BASIC version. If the machine language portion can be tied in, Commodore 64 users will enjoy the increased speed.

One last time, be careful with DiskMod. It can cause more harm than good if used improperly. With enough preparation and understanding you should have no trouble. After that you'll find DiskMod indispensible, especially when you need it most.

```
100 print " Sq disk viewer/changer    jim butterfield "
110 print " q caution - use care - this program "
120 print " can wreck your disk if used "
130 print " without care & understanding! "
140 for j=20 to 85 : if peek(32768+j)<>32 goto160
150 nextj : stop
160 l1=j : s1=l1/5
170 s2=s1*16-1 : s3=5+s1*3
180 dim a(255)
190 b$=chr$(17) : input " drive# ";d$ : if d$= " s "
    then d$= " 0 " : b$=chr$(3)
200 if d$<>" 0 " and d$<>" 1 " goto190
210 open 15,8,15," i " +d$ : gosub500
220 open2,8,2," #0 " : gosub500
230 print " sqqqqqqqqqqqqqqqqqqqpv track,
    sector  0,0[5left] ";
240 input t, s
250 if t<1 or t>77 then close2 : close15 : end
260 print " working " : print#15," b-r:2," ;d$;t;s : gosub500
270 print#15," b-r:2," ;d$;t;s : gosub500
280 for j=0 to 255
290 print#15," m-r " ;chr$(j);b$
300 get#15,a$ : if a$= " " then a$=chr$(0)
310 a(j)=asc(a$) : nextj
320 p=0
330 print " S track " ;t;" sector " ;s
340 for j=p to p+s2 steps1 : print : print " ] " ;
350 v=j : gosub800 : k$= " " : print " - " ; : for k=0 to s1-1
    : v=a(j+k) : gosub800 : print " " ;
360 next k : print " - " ; : for k=0 to s1-1 : v=a(j+k)
    : if (vand127)<32 then v=32
370 v2=v and 63 : if v2=44 or v2=58 or v2=34 then v=32
380 print chr$(v); : nextk,j
390 print : print " r n R ext / r r R ead " ; : if s1<9 then
    print " / r s R wap " ;
400 print : print " next track,sector: " ;
410 if a(0)=0 then print " none " : goto430
420 print a(0);a(1)
430 print " command > " ;
440 input c$ : z=asc(c$)
450 print " Q " : if s1<9 and z=83 then p=128-p : goto330
460 if a(0)<>0 and z=78 then t=a(0) : s=a(1) : goto250
470 if z=93 goto530
480 goto230
500 rem
510 input#15,e,e$,e1,e2 : if e=0 then return
520 print " r disk error: R " e;e$,e1;e2 : end : return
530 if len(c$)<s3 goto230
540 if mid$(c$,4,1)<>" - " or mid$(c$,s3+1,1)<>" - " goto230
550 c1=2 : gosub700
560 c3=c2-1 : for k=1 to s1 : c1=k*3+3 : gosub700
570 print#15," m-w " ;chr$(c3+k);b$;chr$(1);chr$(c2)
580 next k
590 print#15," u2:2," ;d$;t;s : gosub500
600 goto270
700 c2=0 : for j=0 to 1 : c%=asc(mid$(c$,c1+j)) : if c%<58
    then c%=c%-48
710 if c%>64 then c%=c%-55
720 if c%<0 or c%>15 then stop
730 c2=c2*16+c% : nextj : return
800 v=v/16 : for l=1 to 2
810 v%=v : v=(v-v%)*16 : if v%>9 then v%=v%+7
820 k$=k$+chr$(v1)
830 print chr$(v%+48); : next l : return
```

DiskMod 4.0 Machine Code: Enter, Run, and Save this first.

```
100 rem mach code for diskmod 4.0
110 for j=1 to 243 : read x : ch=ch+x : next
120 if ch<>31709 then print " checksum error " : end
130 restore : open 8,8,8," 1:mach part,p,w "
140 print#8,chr$(0)chr$(17); : rem start addr $1100
150 for j=1 to 16 : print#8,chr$(32); : next
160 for j=1 to 243 : read x : print#8, chr$(x); : next
170 for j=1 to 253 : print#8,chr$(32); : next
180 close 8 : end
190 rem
200 data 162,  18, 134, 191, 160,   0, 132, 190, 162
210 data  15,  32, 201, 255, 162,   2, 189, 253,  17
220 data  32, 210, 255, 202,  16, 247, 165, 190,  32
230 data 210, 255, 169,  17,  32, 210, 255,  32, 204
240 data 255, 162,  15,  32, 198, 255,  32, 228, 255
250 data 160,   0, 145, 190,  32, 204, 255, 230, 190
260 data 208, 208,  96,  72,  74,  74,  74,  74,  32
270 data  84,  17, 104,  41,  15, 201,  10, 144,   2
280 data 105,   6, 105,  48,  76, 210, 255, 162,  18
290 data 134, 191, 202, 134, 193, 162,  16, 134, 136
300 data 165, 190,  32,  73,  17, 169,  60,  32, 210
310 data 255, 169,  32,  32, 210, 255, 169,   0, 133
320 data 192, 160,   0, 177, 190, 170,  41, 127, 201
330 data  32, 144,  10, 144,   8, 201,  44, 240,   4
340 data 201,  58, 208,   2, 169,  32, 145, 192, 138
350 data  32,  73,  17, 169,  32,  32, 210, 255, 230
360 data 190, 230, 192, 166, 192, 228, 194, 208, 212
370 data 169,  62,  32, 210, 255, 160,   0, 132, 192
380 data 177, 192,  32, 210, 255, 200, 196, 194, 208
390 data 246, 169,  13,  32, 210, 255, 198, 136, 208
400 data 165,  96, 162,  20, 165, 208, 201,  18, 144
410 data  12, 202, 201,  25, 144,   7, 202, 202, 201
420 data  31, 144,   1, 202, 134,  72,  70, 139, 102
430 data 138, 102, 137, 169,  46, 144,   4, 198, 136
440 data 169,  42,  32, 210, 255, 169, 157,  32, 210
450 data 255, 169,  17,  32, 210, 255, 198,  72,  16
460 data 225,  96,   0,  82,  45,  77,  39,   0,   0
```

DiskMod 4.0 BASIC Part:

```
100 print " S " :print " disk viewer  (c) jim butterfield "
110 l2=8 : if peek(32848)=4 then l2=16
120 t9=35 : dim s%(t9),e$(20)
130 data 17, 20
140 data 24, 19
150 data 30, 17
160 data 35, 16
170 c1$=chr$(1) : c2$=chr$(16) : c3$=chr$(17)
    : c4$=chr$(33)+c2$
180 for j=0 to 20 : e$(j)= " e " +str$(j) : nextj
190 e$(2)= " block not found "
200 e$(3)= " no synch "
210 e$(4)= " block not present "
220 e$(5)= " checksum error in data "
230 e$(7)= " verify error "
240 e$(8)= " write protect! "
250 e$(9)= " header checksum "
260 e$(10)= " overrun "
270 e$(11)= " id mismatch "
280 e$(14)= " format "
290 e$(16)= " decode err "
```

```
300 t1 = 1
310 read t,s : if s>s9 then s9 = s
320 for j = t1 to t : s%(j) = s : nextj : t1 = t + 1 : if t<t9 goto310
330 data 1, 2, 4, 8, 16, 32, 64, 128
340 for j = 0 to 7 : readp%(j) : nextj
350 t5 = 10
360 input " drive# " ;d : if d<0 or d>1 goto340
370 open 15,8,15 : t$ = chr$(1) : s$ = t$ : id$ = " .. "
    : c$ = chr$(192 + d) : d$ = chr$(3) : gosub1220
380 i = 0 : s$ = c1$ : c$ = chr$(176 + d) : t = 18 : t$ = chr$(t)
    : s = 0 : s$ = chr$(s)
390 gosub1220 : if e<>1 then stop
400 print#15, " m-r " + c4$ : get#15,i1$ : if i$ = " "
    then i$ = chr$(0)
410 print#15, " m-r " + chr$(34) + c2$ : get#15,i2$
    : if i2$ = " " then i2$ = chr$(0)
420 id$ = i1$ + i2$
430 d$ = chr$(4) : t$ = chr$(18) : s$ = chr$(0)
    : c$ = chr$(128 + d)
440 print#15, " m-w " + chr$(41) + c2$ + chr$(4) + id$ + t$ + s$
450 print#15, " m-w " + chr$(19) + c2$ + c1$ + t$
460 gosub1240 : if e<>1 then stop
470 d$ = chr$(3) : print : input " track,sector " ;t,s
480 if t = 0 then stop
490 t$ = chr$(t) : s$ = chr$(s) : c$ = chr$(128 + d) : gosub1220
    : if e<>1 then stop
500 s% = s/8 : s1 = s-s%*8
510 p3 = 4*t : p1 = p3 + s% + 1 : p2 = p%(s1)
520 print#15, " m-r " + chr$(p1) + chr$(18) : get#15,b$
530 b = len(b$) : if b then b = asc(b$)
540 print#15, " m-r " + chr$(p3) + chr$(18) : get#15,b$
550 p4 = len(b$) : if p4 then p4 = asc(b$)
560 print " S " ;
570 l1 = l2
580 print#15, " m-r " + chr$(0) + chr$(17) : get#15,r$
590 t1 = 0 : if r$ = " " goto630
600 t1 = asc(r$)
610 print#15, " m-r " + chr$(1) + chr$(17) : get#15,r$
620 s1 = len(r$) : if s1 then s1 = asc(r$)
630 gosub1120
640 if l2 = 8 then print " swap / " ;
650 if b and p2 then print " use / " ; : goto670
660 print " free / " ;
670 ift1>0thenprint " next / " ;
680 print " map / read / quit / print[17spaces] "
690 input " [3shifted spaces,3left] " ;q$
    : print " sqqqqqqqqqqqqqqqq " : rem home,17down
700 if len(q$)<l2*3 + 4 goto780
710 x$ = mid$(q$,3,1) : if x$<> " < " goto780
720 e = 0 : r = 1 : gosub1330
730 x$ = " m-w " + chr$(v) + chr$(17) + chr$(l2)
    : if e>0 then stop
740 for j = 0 to l2-1 : r = 5 + 3*j : gosub1330
    : x$ = x$ + chr$(v) : nextj
750 print#15,x$
760 c$ = chr$(144 + d) : gosub1240
770 goto490
780 for j = 1 to len(q$) : q = asc(mid$(q$,j))
    : if q = 63 or q = 32 then nextj
790 for j = 1 to 1 : next : if q = 85 goto1030
800 if q = 70 goto1050
810 if q = 78 and t1>0 then t = t1 : s = s1 : goto490
820 if q = 82 goto 470
830 if q = 81 then end
840 if q = 80 goto 1000
850 if q = 77 goto 890
860 if q<>83 goto640
870 s7 = s7 + l2*16 : if s7>255 then s7 = 0
880 print " S " ; : l1 = l2 : gosub1130 : goto640
890 print " S bam map q " : for j = 0 to 20
    : print mid$(str$(j),2) : nextj
900 for j = 1 to 35 : j% = j/10 : j1 = j-j%*10
910 z$ = " " : if j1 = 0 then z$ = chr$(j% + 48)
920 print " s " ;tab(j + 2);z$:printtab(j + 2);
    chr$(j1 + 48); " [1left,1down] " ;
930 for k = 0 to 3 : j1 = j*4 + k
940 print#15, " m-r " + chr$(j1) + chr$(18)
950 get#15,z$ : z = len(z$) : if z then z = asc(z$)
960 poke136 + k,z : nextk : poke208,j : sys4550
970 if peek (136)<>0 then print " ? " ;
980 nextj : getz$
990 getz$ : if z$ = " " goto990
991 goto880
1000 open4,4 : l1 = 16 : cmd4 : gosub1130
1010 print#4 : print#4 : close4
1020 goto640
1030 if b and p2 then b = b-p2 : p4 = p4-1 : goto1070
1040 goto640
1050 if b and p2 goto640
1060 b = b or p2 : p4 = p4 + 1
1070 d$ = chr$(4) : c$ = chr$(144 + d)
1080 print#15, " m-w " + chr$(p1) + chr$(18) +
     chr$(1) + chr$(b)
1090 print#15, " m-w " + chr$(p3) + chr$(18) +
     chr$(1) + chr$(p4)
1100 gosub1240 : if e<>1 then stop
1110 d$ = chr$(3) : goto880
1120 sys4368 : s7 = 0
1130 a$ = " [allocated] " :ifbandp2then a$ = " [free]     "
1140 print " track " ;t; " sector " ;s;a$; " id = " ;id$
1150 poke194,l1 : poke190,s7 : sys4447
1160 if t1 = 0 then print " no next block " : goto1180
1170 print " next sector: track " ;t1; " sector " ;s1
1180 return
1190 for l = 1 to 2 : w% = w : w = (w-w%)*16
     : if w%>9 then w% = w% + 7
1200 printchr$(w% + 48); : nextl
1210 return
1220 print#15, " m-w " + c4$ + chr$(4) + id$ + t$ + s$
1230 print#15, " m-w " + chr$(18) + c2$ + c1$ + t$
1240 n = 0
1250 print#15, " m-w " + d$ + c2$ + c1$ + c$
1260 print#15, " m-r " + d$ + c2$ : get#15,e$
1270 e = len(e$) : if e then e = asc(e$)
1280 if e>127 goto1260
1290 if e<>1 then n = n + 1 : if n<t5 goto1250
1300 if e>20 then e = 20
1310 if e<>1 then print " disk error: " ;e$(e)
1320 return
1330 v = 0 : for k = r to r + 1 : x = asc(mid$(q$,k))
1340 if x<58 then x = x-48
1350 if x>64 then x = x-55
1360 if x<0 or x>15 then e = 1 : goto1380
1370 v = v*16 + x : nextk
1380 return
```

Commodore Disk Specifications

Model	D9090	D9060	8250	8050	4040	2031	1541
Drives per Head	1	1	2	2	2	1	1
Heads per Drive	6	4	2	1	1	1	1
Formatted Storage							
Capacity per Unit	7.47 MB	4.98 MB	2.12 MB	1.05 MB	340 KB	170KB	170KB
Max Sequential Files/Drive	7.41 MB	4.94 MB	1.05 MB	521 KB	168 KB	168 KB	168KB
Max Relative Files/Drive	7.35 MB	4.90 MB	1.04 MB	183 KB	167 KB	167 KB	167KB
Disk System Buffer	4 KB	4 KB	4 KB	4 KB	4 KB	2 KB	2KB
Disk Formats							
Cylinders (Tracks)	153	153	77	77	35	35	35
Sectors per Cylinder	128	192	–	–	–	–	–
Sectors per Track	32	32	23–29	23–29	17–21	17–21	17–21
Bytes per Sector	256	256	256	256	256	256	256
Blocks Free	29162	19442	8266	4104	1328	664	664
Transfer Rates (bytes per second)							
Internal to Unit	5 MB	5 MB	40 KB	40 KB	40 KB	40 KB	–
IEEE–488 Bus	1.2 KB	1.2 KB	1.2 KB	1.2 KB	1.2 KB	1.2 KB	–
Access Times (milli-seconds)							
Track-To-Track	3	3	5	*	30	30	30
Average Track	153	153	125	**	360	360	360
Head Settling Time	15	15	–	–	–	–	–
Average Latency	8.34	8.34	100	100	100	100	100
RPM	3600	3600	300	300	300	300	300
	* Track-To-Track: Micropolis 8050 = 30 ms. Tandon 8050 = 5 ms. ** Average Track: Micropolis 8050 = 750 ms. Tandon 8050 = 125 ms.						
Physical Dimensions							
Height (inches)	5.75	5.75	7.0	7.0	7.0	5.5	3.0
Width (inches)	8.25	8.25	15.0	15.0	15.0	8.0	7.0
Depth (inches)	15.25	15.25	13.75	13.75	13.75	14.25	13.0
Weight (pounds)	21	21	28	28	28	20	10
Electrical							
Power (Watts)	200	200	60	50	50	40	35
Voltage (all models)	110 – 120 VAC. 60 Hz						

Disk Utility-Command Set

Command	Abbreviation	Format
Block-Read	B-R	" B-R: " ch;dr;t;s
Block-Write	B-W	" B-W: " ch;dr;t;s
Block-Execute	B-E	" B-E: " ch;dr;t;s
Buffer-Pointer	B-P	" B-P: " ch;p
Block-Allocate	B-A	" B-A: " dr;t;s
Block-Free	B-F	" B-F: " dr;t;s
Memory-Write	M-W	" M-W " adL/adH/nc/data
Memory-Read	M-R	" M-R " adl/adh
Memory-Execute	M-E	" M-E " adl/adh
User Command	U	" ux:ch;dr;t;s

CH	The channel number in DOS: identical to the Secondary Address in the associated OPEN statement
DR	The Drive number: 0 (or 1 floppy dual drives)
T	The Track number: 1 through 154 (depending on the model#)
S	Sector number : 0 through 112 (depending on the model#)
P	The pointer Position for the buffer pointer
ADL	The Low byte of the Address (use CHR$(ADL))
ADH	The High byte of the Address (use CHR$(ADL))
NC	The Number of Characters: 1 through 34
DATA	The actual data in hexadecimal. This is transmitted by using the CHR$ function, ie. CHR$(17) would send the decimal equivalent of hexadecimal 11
X	The index to the user table
PARMS	The Parameters associated with the U command (optional)

Sector Distribution By Track

	Number of Sectors		
Track Number	4040	2031	1541
1 – 17	21	21	21
18 – 24	19	19	19
25 – 30	18	18	18
31 – 35	17	17	17

Track Number	8050	8250	
1 – 39	29	29	
40 – 53	27	27	
54 – 64	25	25	
65 – 77	23	23	
78 – 116		29	
117 – 130		27	
131 – 141		25	
142 – 154		23	

D9060/D9090 – 153 tracks per recording surface (4 on D9060 and 6 on the D9090) with 32 sectors per track

User Command Jump Table

Standard Syntax	Alternate Syntax	Function
U1	UA	Block-Read replacement
U2	UB	Block-Write replacement
U3	UC	Jump to $1300
U4	UD	Jump to $1303
U5	UE	Jump to $1306
U6	UF	Jump to $1309
U7	UG	Jump to $130C
U8	UH	Jump to $130F
U9	UI	Jump to $10F0 (NMI)
U:	UJ	Power-Up vector (reset)

BAM (Block Allocation Map) Formats

4040, 2031, and 1541 BAM Format – Track 18 Sector 00

Byte#	Description	Data
0–1	Track–Sector of first Directory block	18–00
2	ASCII 'a' Identifies DOS 2.6 format	65
3	Reserved for future DOS use	00
4–143	Bit map of available blocks	tracks 1–35

8050 BAM Format

Byte#	Description	Data	
		BAM 1 Tr38 / Sc00	BAM 2 Tr38 / Sc03
0–1	Track–Sector of next BAM block	38–03	39–01
2	ASCII 'c' Identifies DOS 2.5 format	67	67
3	Reserved for future DOS use	00	00
4	Lowest track # mapped in this BAM block	01	51
5	Highest track # (+1) mapped in this BAM block	51	78
6	Number of unused blocks on track:	1	51
7–10	Bit map of available blocks on track:	1	51
11–255	(BAM 2: 11–140)Bit map of available blocks on tracks:	2–50	52–77

8250 BAM Format

Byte#	Description	Data			
		BAM 1 Tr38 / Sc00	BAM 2 Tr38 / Sc03	BAM 3 Tr38 / Sc06	BAM 4 Tr38 / Sc09
0–1	Track–Sector of next BAM block	38–03	38–06	38–09	38–01 (Dir)
2	ASCII 'c' Identifies DOS 2.7 format	67	67	67	67
3	Reserved for future DOS use	00	00	00	00
4	Lowest track # mapped in first BAM block	01	51	101	151
5	Highest track # (+1) mapped in first BAM block	51	101	151	155
6	number of unused blocks on track:	1	51	101	151
7–10	bit map of available blocks on track:	1	51	101	151
11–255	(BAM 4: 11–25)Bit map of available blocks on tracks:	2–50	52–100	102–150	152–154

D9060 / D9090 BAM Format – Track 1 Sector 0 (normal location)

Byte#	Description	Data
0–1	Track–Sector pointer to next BAM block	$FFFF = last
2–3	Track–Sector pointer to previous BAM block	$FFFF = first
4	Lowest track # mapped in this BAM block	
5	Highest track # (+1) mapped in this BAM block	
6	Number of blocks unused on this track	
7–10	Bit map of available blocks on this track	
11–255	Bit map of the next 49 tracks	

Directory Format

2031, 4040, 1541 Directory Header – Track 18 Sector 00

Byte#	Data	Description
1–143		Reserved for 2031 BAM
144–161		Diskette name, padded with shifted spaces
162–163		Diskette ID number
164	160	Shifted space
165–166	50, 65	ASCII '2a' identifies DOS version and format
167–170	160	Shifted spaces
171–255	00	Not used

8050, 8250 Directory Header – Track 39 Sector 00

Byte#	Data	Description
0–1	38, 00	Track–Sector to first BAM block
2	67	ASCII 'c' identifies DOS 2.5 format
3	00	reserved for future DOS use
4–5		Not used
6–21		Diskette name, padded with shifted spaces
22–23	160	Shifted spaces
24–25		Diskette ID number
26	160	Shifted space
27–28	50, 67	ASCII '2c' identifies DOS version and format
29–32	160	Shifted spaces
33–255	00	Not used

D9060 / D9090 Directory Header – Track 0 Sector 0

Byte#	Data	Description
0–1		Track–Sector pointer to bad track and sector list
2–3	00,255	Identifies DOS 3.0 format
4–5	76, 00	Track–Sector of first directory block
6–7	00, 00	Not used
8–9	01, 00	Track–Sector of first BAM block

2031 Directory Blocks - Track 18 Sector 01 through 18

4040 Directory Blocks - Track 18 Sector 01 through 18

8050 Directory Blocks - Track 39 Sector 01 through 29

8250 Directory Blocks - Track 39 Sector 01 through 29

D9060 / D9090 Directory Blocks - Starting on cylinder 76, uses all Tracks - Sectors 00 through 31, then expands to additional blocks as required, providing 'unlimited' Directory size.

Byte#	Description
0–1	Track–sector pointer to next directory block
2	File type
3–4	Track–sector pointer to first file block
5–20	File name, padded with shifted spaces
21–22	Track–sector of first side sector if RELative file
23	Record length if relative file
24–27	Reserved for future file information
28–29	Track–sector pointer for replacement
30–31	Number of blocks used by the file
32–255	Seven more 32–byte file entries (same as 2–31 above, plus two additional unused bytes)

	Additional Notes
1	32 bytes per file entry, except the first entry is 30 bytes
2	Total of eight (8) file entries per directory block
3	File types are: Scratched Files $00 Sequential Files $01 Program Files $02 User–Defined $03 Relative Record $04
4	File type codes are OR'ed with $80 when file is properly closed
5	Track value of 00 in byte zero indicates the last used block in the directory. Sector value then shows next byte to use

The Transactor
The Tech/News Journal For Commodore Computers

PAYS $40

per page for articles

We're also looking for professionally drawn cartoons!

Send all material to:

The Editor
The Transactor
500 Steeles Avenue
Milton, Ontario
L9T 3P7

Volume 5 Editorial Schedule

Issue#	Theme	Copy Due	Printed	Release Date
1	Graphics and Sound	Feb 1	Mar 19	April 1
2	The Transition to Machine Code	Apr 1	May 21	June 1
3	Software Protection & Piracy	Jun 1	Jul 23	August 1
4	Business and Education	Aug 1	Sep 17	October 1
5	Hardware and Peripherals	Oct 1	Nov 19	December 1
6	Programming Aids & Utilities	Dec 1	Jan 19	February 1/85

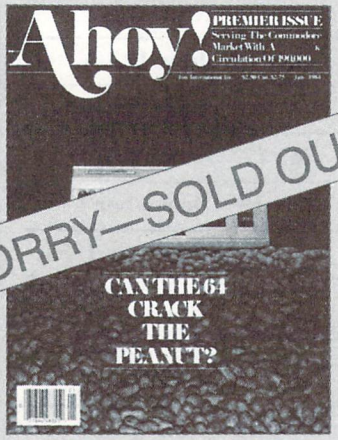
Volume 6 Editorial Schedule

1	Communications & Networking	Feb 1	Mar 21	April 1/85
2	Languages	Apr 1	May 20	June 1
3	Implementing The Sciences	Jun 1	Jul 18	August 1
4	Hardware & Software Interfacing	Aug 1	Sep 21	October 1
5	Real Life Applications	Oct 1	Nov 19	December 1

Advertisers and Authors should have material submitted no later than the 'Copy Due' date to be included with the respective issue.

Ahoy!

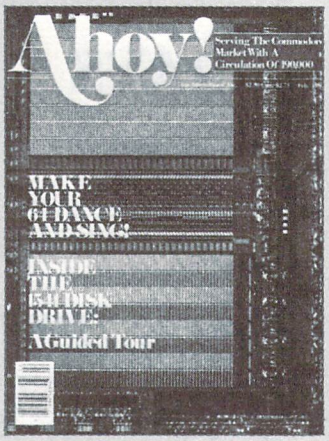
Back Issues



SORRY—SOLD OUT

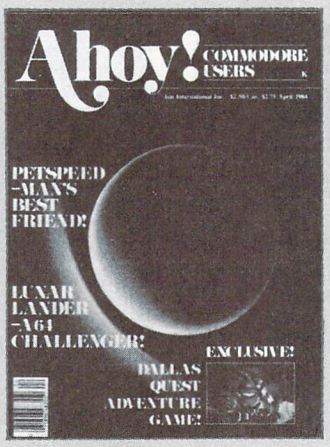
ISSUE #1—JAN. '84 $4.00
The 64 v. the Peanut! The computer as communications device! Protecto's Bill Badger interviewed! And ready to enter: the Multi Draw 64 graphics system! The Interrupt Music Maker/ Editor! A Peek at Memory! Programming Sequential Files!

Don't punch another key without a complete collection of *Ahoy!* and the programming strategies and product analyses each issue provides. Order while supplies last!

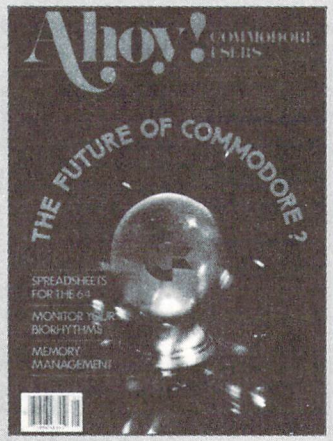


ISSUE #2—FEB. '84 $4.00
Illustrated tour of the 1541 disk drive! Synapse's Ihor Wolosenko interviewed! Users groups! Artificial intelligence! And ready to enter: Music Maker Part II! Night Attack! Programming Relative Files! Screen Manipulation on the Commodore 64!

ISSUE #3—MAR. '84 $4.00
Anatomy of the 64! Printer Interfacing for the 64 and VIC! Educational software: first of a series! Commodares! And ready to enter: Space Lanes! Random Files on the 64! Easy Access Address Book! Dynamic Power for your 64!



ISSUE #4—APR. '84 $4.00
Petspeed and Easy Script tutorials! Printer interfacing and educational software guide continued! Lower case descenders on your 1525! Laserdisc! The Dallas Quest Adventure Game! And ready to enter: Apple Pie! Lunar Lander! Name that Star!



ISSUE #5—MAY '84 $4.00
The Future of Commodore! Inside BASIC program storage! C-64 Spreadsheets! Memory Management on the VIC and 64! Educational Software Guide continues! And ready to enter: Math Master! Air Assault! Biorhythms! VIC 20 Calculator!

Send coupon or facsimile to:
Ahoy! Back Issues, Ion International Inc., 45 West 34th Street—Suite 407, New York, NY 10001

Ahoy!

Please Send Me The Following:

____ Copies of issue number ____
____ Copies of issue number ____
____ Copies of issue number ____

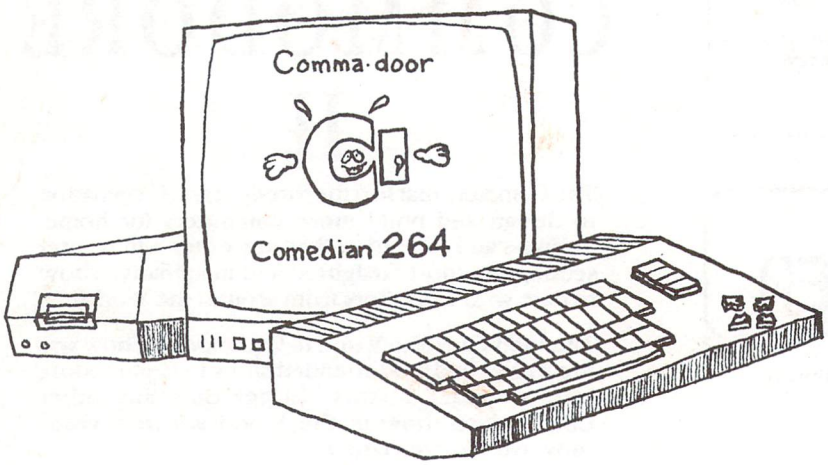
Enclosed Please Find My Check or Money Order for $_____

(Outside the USA please add $1.00 for every copy)

NAME_____
ADDRESS _____
_____
CITY_____
STATE_____
ZIP CODE_____

In 1982 Commodore introduced the breathtaking Commodore 64.
Now, in 1984 Comma-door announces the preposterous Comedian 264.

It bytes. It barks. It even glows in the dark. Does less, costs more, The new Comedian 264.



Features:

20K EFROM (easily forgettable read only memory)
64K SAM (sequential access memory)
7251 Macroprocessor
66 triple stroke keyboard
16 programable function keys
Built in Commodore Basick (V3.1416 + circular commands)
CP/M compatable

Graphics chip

GLIC chip (graphic laser interface chip)
320×4 pixie display
8 user defiable spiteful graphics (24×2 pixies)
2 character sets (Japanese is standard)

Sound chip

SICK chip (sound interface condensed kernal)
8 line octave plus
6 line sestet equals
1 sonnet
3 voices (tenor, bass, soprano)
2 waveforms

Potato chip

HOSTESS chip
Simulates five different flavours:
salt and vinegar
barbeque
plain
rippled
taco

Comedian 264

It's the biggest joke on the market.

R. Brath '84

GIVE YOURSELF A HAND

If you read Transactor, you're not the average Commodore owner. You didn't buy your computer to see what someone else's software could do. You want a discovery tool.

Over the years of teaching computer programing to adults and children, I've been struck by how much more they discover about programming when they have a thing, a device which they're writing for. One that's easy to build, easy to understand, easy to program, and fun to use.

ENTER THE " *HELPING HAND* "

One of the simplest, yet most powerful gizmos you can plug into your computer. It's a drawing device, a controller, a musical instrument, a discovery tool.

It has two essential parts: a board, and a pointer. As you guide the pointer around the board, the computer always knows where you are.

NO SECRETS AND LOTS OF SURPRISES

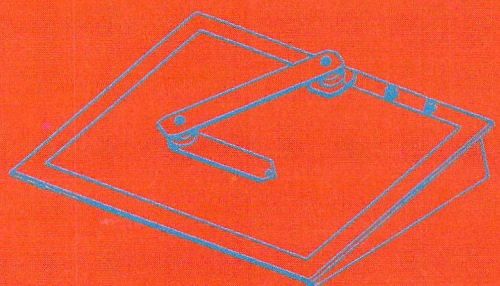
Unlike its high-tech cousins, Koalapad ® and Powerpad ®, the ***HELPING HAND*** is something you can take apart, tinker with, put back together. Kids assemble their own in class. All they need is a screwdriver.

The ***HELPING HAND*** has two potentiometers mounted in its lucite pointer arm and a durable-surface 12" x 16" drawing board you can write on and wipe off. Two pushbuttons conveniently mounted right on the edge of the board act like an extra set of function keys. The tutorial-based manual shows you how to put it together and check it out. And getting you started are some sample programs in Basic with lots of comments.

GIVE YOURSELF ANOTHER HAND--a disk, with many more programs, for registered owners.

TO ORDER

Send a check or money order (U.S. funds) for $50 plus $4 postage and handling (U.S. and Canada). Please specify what computer you have. If you want the disk, add another $15 for the C64 ® or $8 for the VIC ® version.



PERSIMMON PERIPHERALS ROUTE 2, BOX 2306A CLAYTON, GEORGIA 30525

GET THE MOST OUT OF YOUR COMMODORE-64 OR VIC-20 COMPUTER

ULTRABASIC-64... Add 50 commands: graphics, music, TURTLE and game features.Tutoric¹ demo plus. **TAPE $24.95 DISK $27.95**

ASSEMBLER-MONITOR-64 High speed language development. Eleven function editor. Screen editing of source file. **DISK $32.95**

MERCURE-64... Simple, powerful file management with fast design, entry search report capabilities. Tutorial. **DISK $32.95**

SYNTHY-64... Sets the standard for all of the rest. Best 64-synthesizer anywhere. Samples and manual. **CASSETTE $24.95 DISK $27.95.** Also available: 3 great companion music albums; **Classical, Christmas,** and **Ragtime Sing-Along. DISK $12.95 Each.**

GRAPHICS DESIGNER-64... Menu-driven drawings, floor plans and illustrations etc.. Slide program capability. **DISK $32.95**

TAS-64... Full featured technical analysis for stock market evaluations. Manual or entire update capability thru DJNRS.Printer hardcopy. **Disk $84.95**

MACHINE LANGUAGE BOOK Learn all instructions. Access ROM routines, I/O. Listings for Assembler, SIMULATOR, more. **200+PAGE BOOK $14.95**

ANATOMY OF A COMMODORE-64 Complete guide. Full comment ROMS list, detailed internals, descriptions. **300 PAGE BOOK $19.95**

CHARTPAK-64... Professional qualtiy pie, line and bar charts. Menu driven, interactive, hardcopy. **DISK $42.95**

CHARTPLOT-64... Same fine features as CHARTPAK-64 with high quality output to plotters. **DISK $84.95**

ANATOMY OF THE 1541 DISKDRIVE Explains sequential random and program files, DOS, full ROM listing, sample programs. 320 pp. book **$19.95**

ZOOM PASCAL-64... Produces 6502 machine code for speed. Floating point, Integers, strings File handling. **DISK $39.95**

POOL-64/20... Play Fullrack or nine ball using hires graphics. Vic-20 required 8K expander. **TAPE $14.95 DISK $17.95**

MASTER-64... Indexed files; powerful screen managment; excellent printer generator; programmer's aid; BASIC 4.0 commands; machine language monitor. NO RUNTIME ROYALTIES. 150 pp. manual for program developers. **Disk $84.95**

SUPER DISK UTILITY-64... Speed copy 4 ways: Total, Bam, Append or File. Dump or modify sectors. More. **DISK $22.95**

XREF-64... Sorted BASIC cross-reference on screen or printer Fast ML Sort. Add your own tokens. **DISK ONLY $17.95**

FREE CATALOG Ask for a listing of other Abacus Software for Commodore-64 or Vic-20

DISTRIBUTORS

Great Britain:
ADAMSOFT
18 Norwich Ave.
Rochdale, Lancs.
01-788-8963

Belgium:
Inter. Services
AVGuillaume 30
Brussel 1160, Belgium
2-660-1447

France:
Micro Application
147 Avenue Paul-Doumer
Rueill Malmaison, France
1-732-9254

Canada East:
KING MICROWARE LTD.
5950 Cote des Neiges
Montreal, Quebec H3S 1Z6
514/737-9335

West Germany:
DATA BECKER
Merowingerstr 30
4000 Dusseldorf
0211/312085

Sweden:
TIAL TRADING
PO 516
34300 Almhult
476-12304

Australia:
CW ELECTRONICS
416 Logan Road
Brisbane, Queens.
07-397-0808

New Zealand:
VISCOUNT ELECTRONICS
306-308 Church Street
Palmerston North
63-86-696

*** DEALER INQUIRIES INVITED**

AVAILABLE AT COMPUTER STORES, OR WRITE:

Abacus Software

P.O. BOX 7211 GRAND RAPIDS, MICH. 49510

For postage & handling, add $2.50 (U.S. and Canada), add $5.00 for foreign. Make payment in U.S. dollars by check, money order or charge card. (Michigan Residents add 4% sales tax.)

FOR QUICK SERVICE PHONE 616-241-5510