

CDC 00594

The Transactor

■ The Tech/News Journal For Commodore Computers

95% Advertising Free!

Nov. 1986: Volume 7, Issue 03. \$3.50

Programming The Chips

- A Comparison of 6 Disk Backup Utilities
- All About the C128 Video Display Controller
- Keyboard Expander: A Fabulous Keyboard Utility for the 64
- Switcheroo Pivot: Make the 8500 and Z80 in the C128 Work Together
- Special Report On EXPO 86
- Program Trace Monitor: A C64 Machine Language Profiler
- Programming the CIA Time of Day Clocks
- A Peek At Amiga Disk Structure
- Raster IRQ's on the 64



**FREE
T-Shirt
Offer!**
see page 77



Good news!

If you want to get the most out of your Commodore 128 or 64, we have good news for you. The Pocket 128 and 64 Series of Software both offer you serious, professional quality software packages that are easy to use and inexpensive.

How easy?

Pocket 128 or 64 Software is so easy, you're ready to start using it as soon as it's loaded into memory. Even if you've never been in front of a computer before, you'll be up and running in thirty minutes. In fact, you probably won't ever need the reference guide... 'help' is available at the touch of a key. That's how easy.

How serious?

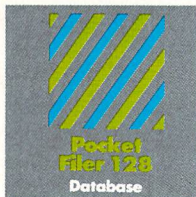
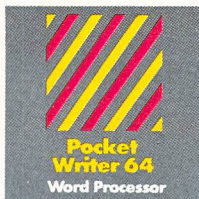
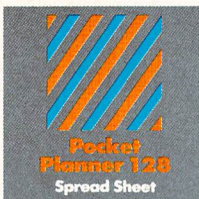
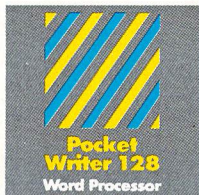
Pocket 128 or 64 packages have all the power you're ever likely to need. They have all of the features you'd expect in top-of-the-line software, and then some. The good news is that Pocket 128 or 64 Software Packages are priced way down there... where you can afford them. Fast, powerful, easy to learn and inexpensive. Say, that is good news!

All for one and one for all

Pocket 128 or 64 Software Packages offer you something else you might not expect... integration. You can combine the output of Pocket Writer, Pocket Filer and Pocket Planner into one piece of work. You can create a finished document with graphs, then send individually addressed copies.

The bottom line is Solutions

The word solutions is our middle name and bottom line. When you purchase Pocket 128 or 64 software, you can count on it to solve your problems.



For information write to:

Digital Solutions
30 Wertheim Court, Unit 2
Richmond Hill, Ontario
Canada L4B 1B9
telephone (416) 731-8775

Pocket Writer 128 or 64 Word Processing

What you see is what you get

With Pocket Writer 128 or 64, there's no more guessing what text will look like when you print it. What you see is what you get... on screen and in print. There are no fancy codes to memorize, no broken words at the end of a line.

Easy to learn and sophisticated. Pocket Writer 128 or 64 offers standard word processing features plus...

- on-screen formatting and wordwrap
- on-screen **boldface**, **underlines** and **italics**
- no complicated format commands to clutter text
- on-screen help at all levels
- spelling-checker lets you add words to your dictionary
- 40 or 80 columns on screen
- files compatible with PaperClip™ or other word processors

Pocket Planner 128 or 64 Computerized Spreadsheet

Make fast work of budgeting and forecasting

Pocket Planner 128 or 64 software lets you make fast work of all your bookkeeping chores. Cheque books, household accounts, business forecasting and bookkeeping are just some of the jobs that Pocket Planner 128 or 64 packages make easier. You can even create four different kinds of graphs.

Accurate, sophisticated and easy to use. Pocket Planner 128 or 64 offers standard spreadsheet features plus...

- accuracy up to 16 digits, about twice as many as most spreadsheets for the Commodore 128 or 64
- sideways printing available on dot matrix printers, for oversized spreadsheets that won't fit on standard paper
- **on-screen help** at all levels
- compatible with VisiCalc™ files
- 80 column on-screen option for the Commodore 64 in addition to the standard 40 columns
- graphics include **bar**, **stacked bar**, **line** and **pie** graphs that can also be used in word processing files
- **smart evaluation** of formulae for accurate complex matrices

Pocket Filer 128 or 64 Database Manager

Database management made easy

With Pocket Filer 128 or 64, you can organize mailing lists, addresses, inventories, telephone numbers, recipes and other information in an easily accessible form. Use it with Pocket Writer 128 or 64 (or other word processors) to construct individually customized form letters.

Pocket Filer 128 or 64 packages are fast, sophisticated and truly easy to use. In addition to standard database features they offer...

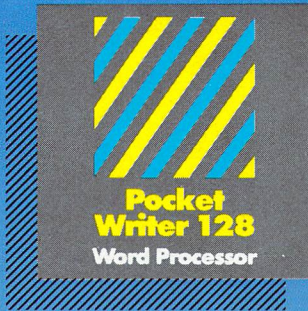
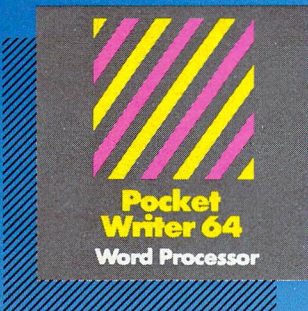
- use up to 255 fields per record (2,000 characters per record)
- sorts by up to 9 criteria, can save 9 different sorts
- print **labels** in multiple columns
- flexible report formatting including **headers** and **footers**
- optional password protection including **limited access viewing** or **updating**
- on-screen help at all levels
- print from any record to any record
- arithmetic and trigonometric functions in **reports** using up to 16 digit accuracy

™PaperClip is a registered trademark of Batteries Included

™Visicalc is a registered trademark of Software Arts

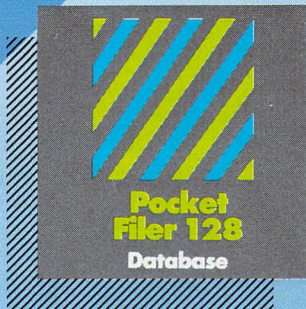
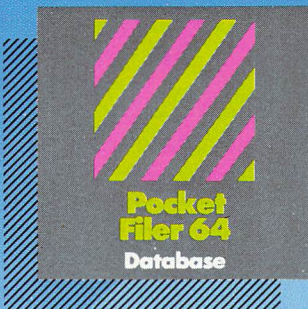
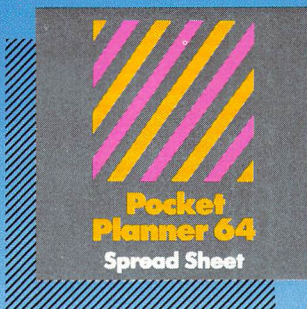


Solutions!



PW 128/64 Dictionary
also available at \$14.95 (U.S.)

MAIL ORDERS:
Transactor Publishing Inc.
500 Steeles Avenue
Milton, Ontario, L9T 3P7
1-416-878-8438
Or use order card at center.

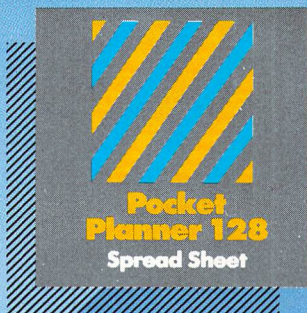


Only The Name Is New

The professional, full-featured software line from Digital Solutions is now called Pocket Software.
Pocket Writer 128/64.
Pocket Filer 128/64.
Pocket Planner 128/64.
The names are new, but this super software is still the same.

From now on, when you hear the word Pocket, it means software that's full-featured, handy and easy to use.

Pocket Software at prices that won't pick your pocket.



Best-selling software for Your Commodore 128 or 64

You want the very best software you can find for your Commodore 128 or 64, right?

You want integrated software — word processing, database and spreadsheet applications — at a sensible price. But, you also want top-of-the-line features. Well, our Pocket 128/64 software goes one better.

With Pocket 128 or 64, you'll find all the features you can imagine ... and then some. And Pocket 128/64 is so easy to use, you won't even need the reference guide. On-screen and in memory instructions will have you up and running in less than 30 minutes, even if you've never used a computer before.

The price? It's as low as you'd expect for a line of software called 'Pocket'. Suggested Retail Price for the 64 software is \$39.95 (U.S.) and \$49.95 (U.S.) for the 128. Any of the 64 products may be upgraded to their 128 version for \$15.00 (U.S.) + \$3.00 shipping and handling. (Available to registered owners from Digital Solutions Inc. only.)

Pocket Writer 128 or 64, Pocket Planner 128 or 64 and Pocket Filer 128 or 64 ... **Solutions** at sensible prices from Digital Solutions Inc.

International & Distributor enquiries to:

Serious software that's simple to use.



30 Wertheim Court, Unit 2
Richmond Hill, Ontario
Canada L4B 1B9
telephone (416) 731-8775

JOIN TPUG

The largest Commodore Users Group

Benefit from:

Access to library of public domain software
for C-64, VIC 20 and PET/CBM

Magazine (10 per year) with advice from

Jim Butterfield
Brad Bjomdahl
Liz Deal

TPUG yearly memberships:

Regular member (attends meetings)	—\$35.00 Cdn.
Student member (full-time, attends meetings)	—\$25.00 Cdn.
Associate (Canada)	—\$25.00 Cdn.
Associate (U.S.A.)	—\$25.00 U.S.
	—\$30.00 Cdn.
Associate (Overseas — sea mail)	—\$35.00 U.S.
Associate (Overseas — air mail)	—\$45.00 U.S.

FOR FURTHER INFORMATION:

Send \$1.00 for an information catalogue
(tell us which machine you use!)

To: TPUG INC.
DEPT. A,
101 DUNCAN MILL RD., SUITE G7
DON MILLS, ONTARIO
CANADA M3B 1Z3

COMAL INFO If you have COMAL— We have INFORMATION.

BOOKS:

- COMAL From A To Z, \$6.95
- COMAL Workbook, \$6.95
- Commodore 64 Graphics With COMAL, \$14.95
- COMAL Handbook, \$18.95
- Beginning COMAL, \$22.95
- Structured Programming With COMAL, \$26.95
- Foundations With COMAL, \$19.95
- Cartridge Graphics and Sound, \$9.95
- Captain COMAL Gets Organized, \$19.95
- Graphics Primer, \$19.95
- COMAL 2.0 Packages, \$19.95
- Library of Functions and Procedures, \$19.95

OTHER:

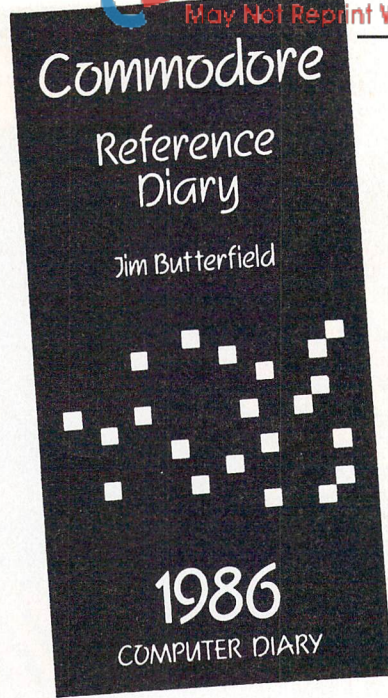
- COMAL TODAY subscription, 6 issues, \$14.95
- COMAL 0.14, Cheatsheet Keyboard Overlay, \$3.95
- COMAL Starter Kit (3 disks, 1 book), \$29.95
- 19 Different COMAL Disks only \$94.05
- Deluxe COMAL Cartridge Package, \$128.95
(includes 2 books, 2 disks, and cartridge)

ORDER NOW:

Call TOLL-FREE: 1-800-356-5324 ext 1307 VISA or MasterCard
ORDERS ONLY. Questions and Information must call our
Info Line: 608-222-4432. All orders prepaid only—no C.O.D.
Add \$2 per book shipping. Send a SASE for FREE Info
Package or send check or money order in US Dollars to:

COMAL USERS GROUP, U.S.A., LIMITED
5501 Groveland Ter., Madison, WI 53716

TRADEMARKS: Commodore 64 of Commodore Electronics Ltd.;
Captain COMAL of COMAL Users Group, U.S.A., Ltd.



From The Guru Himself! The 1986 Commodore Reference Diary

A 65 page reference section that includes:

- All hardware specifications including the C128 and PC10/20
- Useful memory locations
- Useful programs
- SuperCharts
- BASIC and machine language hints
- Hexadecimal conversion
- Sound, video
- and more

The full calendar and date book includes:

- National holidays in ten countries
- Personal notes
- 1987 forward planner
- Name, address, telephone section

Just \$5.95

(plus 50¢ postage and handling)

Order Your Copy Today!

Canada
The Transactor
500 Steeles Avenue
Milton, Ontario
L9T 3P7

USA
The Transactor
277 Linwood Avenue
Buffalo, New York
14209

Dealer Orders:

Canada
Norland Agencies
251 Nipissing Road
Milton, Ontario
L9T 4T7
(416) 876-4774

USA
MicroPace Distributing
1510 North Neil Street
Champaign, Illinois
61820
1 800 362-9653

Volume 7
Issue 03
 Circulation 64,000

The Transactor

Programming The Chips

Start Address Editorial **3**

Bits and Pieces 6

- C-128 Program Entry Tips
- Stop Amiga Disk-Death
- Easy PAL to CBM Source Conversion
- Micro STP
- Border Cross
- Disk Protection
- Some C-128 Tips
- Disabling The C-128 RUN Key
- C-64 Upper/Lowercase Timescroll
- Free C-128 CP/M Manual
- Non-Random Surprises
- Watchacallit?
- Quicker Assembly on the Amiga
- Changing Baud Rates For The C-64
with 1650/6240 Modems
- Borrowing Money?

News BRK 77

- Transactor Writer's Guide Finally Finished
- Subscription Timing
- Free Transactor T's with Mag + Disk Subscription
- Transactor Disk Price Increase
- Refund Policy
- Allow 2 to 6 Weeks
- Transactor Mail Order News
- Transactor Disks, Back Issues, and Microfiche
- Sending Cheques For Transactor Products
- The Transactor Communications Disk
- CompuServe and Quantum Link
- The Commodore Show, September 20 and 21, 1986
- Twin Cities 128: The Commodore 128 Journal
- iNet 2000
- High-Powered FREE Terminal Program for the C64
- Hardware Interconnection Products
- Aegis Ships CAD Program for the Amiga

Letters 10

- Column For Machine Language Sub-Routines?
- New Testament Available On 1541 Disk Format
- Flexible Vector Problem
- G-Link Info
- The SID Chip: Does It Really Exist?
- Transactor Disk Directorytions
- Super Empty SuperPET
- Verifizer for the C128
- Disk Un-Assembler Once Again
- Super Kit Users Notes
- Games From The Outside Out
- Another Vote Against The Atari Article
- Rx For Grammar
- Spel Cheker

TransBloopers ... 14

- Amiga Startup-Sequence
- Eliminating SAVE@
- Improving The SYS Command
- SYMSS 3.1 Fixes
- C128 Memory Maps

TransBASIC Installment #11 15

EXPO 86 a computerist's report from the Vancouver World's Fair 22

The AX2000 a 2 megabyte review 28

A Peek At Amiga File Structure 30

Disk Copier Comparison 6 backup utilities pitted head to head ... 33

Who Do You Trust? a hourly synopsis of a *fun-filled* evening 40

THE CLOCK a dual alarm for the 64 CIA TOD clocks 42

VIC-II Chip Interrupts 46

Switcheroo Pivot Z80 and 8500 CPUs working together in the C128 48

Program Trace Monitor a machine language profiler 52

The 8563 VDC Chip all about the C128 Video Display Controller chip .. 57

Keyboard Expander the last keyboard utility you'll ever need! 64

Compu-toons 81

**Note: Before entering programs,
 see "Verifizer" on page 4**

The Transactor

The Tech/News Journal For Commodore Computers

Editor in Chief
Karl J. H. Hildon

Editor
Richard Evers

Technical Editor
Chris Zamara

Art Director
John Mostacci

Administration & Subscriptions
Anne Richard
Kathryn Holloway

Contributing Writers

Ian Adam	James A. Lisowski
Daniel Bingamon	Jack Lothian
Neil Boyle	Scott Maclean
Anthony Bryant	Don Maple
Tim Buist	David Martin
Jim Butterfield	Steve McCrystal
Betty Clay	Stacy McInnis
Gary Cobb	Jim McLaughlin
Jack Cole	Steve Michel
Jeffery Coons	Chris Miller
Pierre Corriveau	Terry Montgomery
Robert V. Davis	Michael Mossman
Elizabeth Deal	Gerald Neufeld
Frank E. DiGioia	Noel Nyman
Yijun Ding	Dave Pollack
Paul T. Durrant	Richard Perrit
Michael J. Erskine	Terry Pridham
Jack Farrah	Raymond Quirling
R. James de Graff	Gary Royal
Jim Grubbs	John W. Ross
Tom Hall	Louis F. Sander
Bob Hayes	Fred Simon
Andy Hochheimer	P. A. Slaymaker
John Holttum	Edward Smeda
David Hook	Darren J. Spruyt
Tomas Hrbek	Aubrey Stanley
Robert Huehn	David Stidolph
Tom Hughes	Richard Stringer
David Jankowski	Nick Sullivan
Chris Johnson	Karel Vander Lugt
Mark Jordan	Audrys Vilkas
Clifton Karnes	Steven Walley
Jesse Knight	Jack Weaver
James E. LaPorte	Evan Williams
William Levak	Chris Wong

Production
Attic Typesetting Ltd.

Printing
Printed in Canada by
MacLean Hunter Printing

The Transactor is published bi-monthly by Transactor Publishing Inc., 500 Steeles Avenue, Milton, Ontario, L9T 3P7. Canadian Second Class mail registration number 6342. USPS 725-050, Second Class postage paid at Buffalo, NY, for U.S. subscribers. U.S. Postmasters: send address changes to The Transactor, 277 Linwood Avenue, Buffalo, NY, 14209 ISSN# 0827-2530.

The Transactor is in no way connected with Commodore Business Machines Ltd. or Commodore Incorporated. Commodore and Commodore product names (PET, CBM, VIC, 64) are registered trademarks of Commodore Inc.

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, 416 878 8438. Note: Subscriptions are handled at this address ONLY. Subscriptions sent to our Buffalo address (above) will be forwarded to Milton HQ. For best results, use postage paid card at center of magazine.

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 or illustrations will be included with articles depending on quality. Authors submitting diskettes will receive the Transactor Disk for the issue containing their contribution.

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') is a straight line 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 "[10 spaces]flush right "

Cursor Characters For PET / CBM / VIC / 64

Down - [q]	Insert - [I]
Up - [Q]	Delete - [L]
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 - [L]	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 - [F]	F5 - [G]
F2 - [I]	F6 - [K]
F3 - [F]	F7 - [H]
F4 - [J]	F8 - [L]

**Please Note: The Transactor's
phone number is: (416) 878-8438**

Quantity Orders:

U.S.A. Distributor:
Capital Distributing
Charlton Building
Derby, CT
06418
(203) 735 3381
(or your local wholesaler)

Master Media CompuLit
261 Wycroft Road Box 352
Oakville, Ontario
L6J 5B4 V5C 4K6
(416) 842 1555 604 941 7911
(or your local wholesaler)

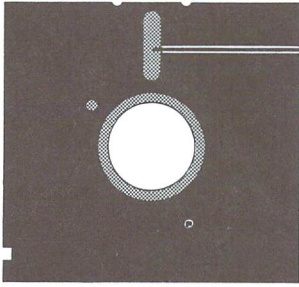
Norland Communications
251 Nipissing Road, Unit 3
Milton, Ontario
L9T 4Z5
416 876 4774

SOLD OUT: The Best of The Transactor Volumes 1 & 2 & 3; Vol 4 Issues 04, 05, 06, and Vol 5 Issues 03, 04 are available on microfiche only

Still Available: Vol. 4: 01, 02, 03. Vol. 5: 01, 02, 04, 05, 06. Vol. 6: 01, 02, 03, 04, 05, 06. Vol. 7: 01, 02, 03

Back Issues: \$4.50 each. Order all back issues from Milton HQ.

All material accepted becomes the property of The Transactor. All material is copyright by Transactor Publications Inc. Reproduction in any form without permission is in violation of applicable laws. Please re-confirm any permissions granted prior to this notice. Solicited material is accepted on an all rights basis only. Write to the Milton address for a writers package. The opinions expressed in contributed articles are not necessarily those of The Transactor. Although accuracy is a major objective, The Transactor cannot assume liability for errors in articles or programs. Programs listed in The Transactor are public domain; free to copy, not to sell.



Starb Address

Copy Copy

My camera-ready art for page three this issue is a mixture of brain overflows, mainly because I couldn't produce just one that would cover the entire page. With so many projects on full burn over the last two months (Writer's Guide, Bits Book, Q-Link/CIS, iNet, T-Shirts, Communications Disk, Micro Sleuth, disk ROMs, typesetter formats, plus others), I must admit I've been a little out of touch with the market outside my immediate neighborhood of magazine dealings. So much so that I even made some phone calls to help refresh my opinionatedness. And today, with all front burners on full for so long, my neurons aren't holding as strong a charge as usual. I suppose every writer has this experience at one time or another - you start to ramble just so you don't have to expand your type size and leading in order to fill a page and appear like a uninspired door-knob that can't... er, see what I mean? Ok Karl, get on with it.

First I'd like to retract a statement from page three last issue. I suggested there probably isn't one computer hobbyist out there without possession of at least one copy of a some piece of proprietary software that they haven't paid for. I suppose I exaggerated a little, but only to make a point: manufacturers need to offer more in package than just a program to accrue sales. By including a "batch of goodies" with the software, the customer who sincerely wants the item will be less willing to settle for the bare-bones, unprotected version. Besides, getting the occasional freebie isn't the worst crime in the world. But large-scale distribution of a pirates de-protecting efforts is, quite simply, detestable. That was my main point, which brings me to my next.

We've been receiving a number of letters regarding local duplication of Transactor Disks. Here's a few clippings:

"... I would like to sell Transactor Disks through our club. . ."
"... we felt that since the programs are considered public domain that the disks could be included in the club library. . ."
"... we would also like a subscription to The Transactor Disk, which we would make available to club members, with your permission."
"allow members who can show us a copy of the magazine to copy the disk. . ."
"... it seems to be a nuisance to mail off for them. . ."
"... the price doesn't seem to be a problem. . ."
"... do not believe our members would be willing to purchase subscriptions at \$45 even though that is a reasonable charge for 6 disks that save typing the programs in. . ."

Get the picture? Since introducing The Transactor Disk, we've received the same request no less than 50 times. Our policy is, "free to copy, not to sell" (see bottom of opposite page). Although the disks are marked "(c) copyright. . .", there is no copy protection and we couldn't even afford any attempts at thwarting copy activity. But consider this: we spend a lot of time on those disks - testing, editing, commenting, sometimes de-bugging - Chris, Richard, and myself spend between one and two days to complete each. Even with the price increase, they still cost less than disks from many of the other magazines. And what little profit we take (important revenue for us) doesn't even put a dint in our total expenses.

I'm not saying you shouldn't copy the programs. In fact, if you would like to put them into your club libraries, please do. Most club software libraries are split into categories. All we ask is that you "de-unify" the disks into their respective categories in the true sense of a "library". Much like you can copy an article from any magazine at your local public library, even at a nickel a page you wouldn't copy the entire mag - it would cost less to go buy one. Same for the disks. If the programs are split between two or three library entries, members can still obtain machine-readable versions of the programs, but the unified collection will cost less. If someone makes the odd dupe, from the pages or sectors, so be it. But in the context of my second paragraph, our product is software with inexpensive (usually required) documentation. Alternately, we offer a magazine with an optional inexpensive service that can easily be justified in dollars per hour. Either way you look at it, wide spread duplication will only help to shorten our life as a supplier. Remember, we wouldn't be the first computer publication to become another page in Chapter 11, although we're probably the first to meet you half way.

Did I hear someone say, "you hypocrites - your SuperKit/1541 will make copies of protected packages, and you're selling 1541 upgrade kits that make only minor changes to the original.?" No? Good. Because once you buy a piece of software, you also buy the rights to personal use as you see fit. Super Kit is for making personal backups - handing them out is against the law, and that's not why we sell it. Secondly, we're not cloning disk drives - we're upgrading the operating system that 1541 owners have every right to use. Case in point: an upgrade ROM is available for the onboard computer of new Corvettes. It adds about 60 h.p. at around \$10 per horse. Apparently the ROM contains only slight mods to the original, but owning the car means you have the right to install those mods. It's just like adding headers - they're the same at both ends, different in the middle. Nobody is breaking the law, but if a second manufacturer makes headers of identical design, room exists for a conflict.

My last spill is open for discussion, and responses are welcome. When the CBM 8096 was released, Visicorp decided they would not produce a version of Visicalc that would access the extra 64K. I don't know who, but someone else did. The 96K version powered up with an incredible 70K of workspace (10K on the 8032 version) and it was even offered to Visicorp free of charge! They still didn't want it, but that did not make it ok to give out. Now that Visicorp has folded, what about Visicalc. Is it public domain? I'm sure the "Y/N" fraction contains many arguments here. Some would say nobody will suffer. . . except their existing competitors of course. What about other suppliers who may be gone, but whos software is not forgotten. And what about the law? Does anyone know? Personally, I could go either way on this one. So readers, any ideas?

There is nothing as constant as change, I remain

Karl J.H. Hildon, Editor In Chief

and I thought I wouldn't need to scrunch this page this time. huhg

Using "VERIFIZER"

The Transactor's Foolproof Program Entry Method

VERIFIZER should be run before typing in any long program from the pages of The Transactor. It will let you check your work line by line as you enter the program, and catch frustrating typing errors. The VERIFIZER concept works by displaying a two-letter code for each program line which you can check against the corresponding code in the program listing.

There are five versions of VERIFIZER here; one for PET/CBMs, VIC or C64, Plus 4, C128, and B128. Enter the applicable program and RUN it. If you get a data or checksum error, re-check the program and keep trying until all goes well. You should SAVE the program, since you'll want to use it every time you enter one of our programs. Once you've RUN the loader, remember to enter NEW to purge BASIC text space. Then turn VERIFIZER on with:

SYS 634 to enable the PET/CBM version (off: SYS 637)
 SYS 828 to enable the C64/VIC version (off: SYS 831)
 SYS 4096 to enable the Plus 4 version (off: SYS 4099)
 SYS 3072,1 to enable the C128 version (off: SYS 3072,0)
 BANK 15: SYS 1024 for B128 (off: BANK 15: SYS 1027)

Once VERIFIZER is on, every time you press RETURN on a program line a two-letter report code will appear on the top left of the screen in reverse field. Note that these letters are in uppercase and will appear as graphics characters unless you are in upper/lowercase mode (press shift/Commodore on C64/VIC).

Note: If a report code is missing (or "--") it means we've edited that line at the last minute which changes the report code. However, this will only happen occasionally and usually only on REM statements.

With VERIFIZER on, just enter the program from the magazine normally, checking each report code after you press RETURN on a line. If the code doesn't match up with the letters printed in the box beside the listing, you can re-check and correct the line, then try again. If you wish, you can LIST a range of lines, then type RETURN over each in succession while checking the report codes as they appear. Once the program has been properly entered, be sure to turn VERIFIZER off with the SYS indicated above before you do anything else.

VERIFIZER will catch transposition errors like POKE 52381,0 instead of POKE 53281,0. However, VERIFIZER uses a "weighted checksum technique" that can be fooled if you try hard enough; transposing two sets of 4 characters will produce the same report code but this should never happen short of deliberately (verifier could have been designed to be more complex, but the report codes would need to be longer, and using it would be more trouble than checking code manually). VERIFIZER ignores spaces, so you may add or omit spaces from the listed program at will (providing you don't split up keywords!). Standard keyword abbreviations (like nE instead of next) will not affect the VERIFIZER report code.

Technical info: VIC/C64 VERIFIZER resides in the cassette buffer, so if you're using a datasette be aware that tape operations can be dangerous to its health. As far as compatibility with other utilities goes, VERIFIZER shouldn't cause any problems since it works through the BASIC warm-start link and jumps to the original destination of the link after it's finished. When disabled, it restores the link to its original contents.

PET/CBM VERIFIZER (BASIC 2.0 or 4.0)

```

CI 10 rem* data loader for 'verifier 4.0' *
CF 15 rem pet version
LI 20 cs=0
HC 30 for i=634 to 754:read a:poke i,a
DH 40 cs=cs+a:next i
GK 50:
OG 60 if cs<>15580 then print '***** data error *****':end
JO 70 rem sys 634
AF 80 end
IN 100:
ON 1000 data 76,138, 2,120,173,163, 2,133,144
IB 1010 data 173,164, 2,133,145,88,96,120,165
CK 1020 data 145,201, 2,240,16,141,164, 2,165
EB 1030 data 144,141,163, 2,169,165,133,144,169
HE 1040 data 2,133,145,88,96,85,228,165,217
OI 1050 data 201, 13,208,62,165,167,208,58,173
JB 1060 data 254, 1,133,251,162, 0,134,253,189
PA 1070 data 0, 2,168,201,3,32,240,15,230,253
HE 1080 data 165,253,41, 3,133,254,32,236, 2
EL 1090 data 198,254,16,249,232,152,208,229,165
LA 1100 data 251,41,15,24,105,193,141, 0,128
KI 1110 data 165,251,74,74,74,74,24,105,193
EB 1120 data 141, 1,128,108,163, 2,152,24,101
DM 1130 data 251,133,251,96
  
```

VIC/C64 VERIFIZER

```

KE 10 rem* data loader for 'verifier' *
JF 15 rem vic/64 version
LI 20 cs=0
BE 30 for i=828 to 958:read a:poke i,a
DH 40 cs=cs+a:next i
GK 50:
FH 60 if cs<>14755 then print '***** data error *****':end
KP 70 rem sys 828
AF 80 end
IN 100:
EC 1000 data 76,74, 3,165,251,141, 2, 3,165
EP 1010 data 252,141, 3, 3,96,173, 3, 3,201
OC 1020 data 3,240,17,133,252,173, 2, 3,133
MN 1030 data 251,169,99,141, 2, 3,169, 3,141
MG 1040 data 3, 3,96,173,254, 1,133,89,162
DM 1050 data 0,160, 0,189, 0, 2,240,22,201
CA 1060 data 32,240,15,133,91,200,152,41, 3
NG 1070 data 133,90,32,183, 3,198,90,16,249
OK 1080 data 232,208,229,56,32,240,255,169,19
AN 1090 data 32,210,255,169,18,32,210,255,165
GH 1100 data 89,41,15,24,105,97,32,210,255
JC 1110 data 165,89,74,74,74,74,24,105,97
EP 1120 data 32,210,255,169,146,32,210,255,24
MH 1130 data 32,240,255,108,251, 0,165,91,24
BH 1140 data 101,89,133,89,96
  
```

VIC/64 Double Verifier Steven Walley, Sunnymead, CA

When using 'VERIFIZER' with some TVs, the upper left corner of the screen is cut off, hiding the verifier-displayed codes. DOUBLE VERIFIZER solves that problem by showing the two-letter verifier code on both the first and second row of the TV screen. Just run the below program once the regular Verifier is activated.


```
KM 100 for ad = 679 to 720:read da:poke ad,da:next ad
BC 110 sys 679: print
DI 120 print " double verifizer activated ":new
GD 130 data 120, 169, 180, 141, 20, 3
IN 140 data 169, 2, 141, 21, 3, 88
EN 150 data 96, 162, 0, 189, 0, 216
KG 160 data 157, 40, 216, 232, 224, 2
KO 170 data 208, 245, 162, 0, 189, 0
FM 180 data 4, 157, 40, 4, 232, 224
LP 190 data 2, 208, 245, 76, 49, 234
```

```
DI 1140 data 20, 133, 208, 162, 0, 160, 0, 189
LK 1150 data 0, 2, 201, 48, 144, 7, 201, 58
GJ 1160 data 176, 3, 232, 208, 242, 189, 0, 2
DN 1170 data 240, 22, 201, 32, 240, 15, 133, 210
GJ 1180 data 200, 152, 41, 3, 133, 209, 32, 113
CB 1190 data 16, 198, 209, 16, 249, 232, 208, 229
CB 1200 data 165, 208, 41, 15, 24, 105, 193, 141
PE 1210 data 0, 12, 165, 208, 74, 74, 74, 74
DO 1220 data 24, 105, 193, 141, 1, 12, 108, 211
BA 1230 data 0, 165, 210, 24, 101, 208, 133, 208
BG 1240 data 96
```

VERIFIZER For Tape Users

Tom Potts, Rowley, MA

The following modifications to the Verifizer loader will allow VIC and 64 owners with Datasets to use the Verifizer directly (without the loader). After running the new loader, you'll have a special copy of the Verifizer program which can be loaded from tape without disrupting the program in memory. Make the following additions and changes to the VIC/64 VERIFIZER loader:

```
NB 30 for i = 850 to 980: read a: poke i,a
AL 60 if cs<>14821 then print "*****data error*****": end
IB 70 rem sys850 on, sys853 off
-- 80 delete line
-- 100 delete line
OC 1000 data 76, 96, 3, 165, 251, 141, 2, 3, 165
MO 1030 data 251, 169, 121, 141, 2, 3, 169, 3, 141
EG 1070 data 133, 90, 32, 205, 3, 198, 90, 16, 247
BD 2000 a$ = "verifizer.sys850[space]"
KH 2010 for i = 850 to 980
GL 2020 a$ = a$ + chr$(peek(i): next
DC 2030 open 1,1,1,a$: close 1
IP 2040 end
```

Now RUN, pressing PLAY and RECORD when prompted to do so (use a rewind tape for easy future access). To use the special Verifizer that has just been created, first load the program you wish to verify or review into your computer from either tape or disk. Next insert the tape created above and be sure that it is rewound. Then enter in direct mode: OPEN1:CLOSE1. Press PLAY when prompted by the computer, and wait while the special Verifizer loads into the tape buffer. Once loaded, the screen will show FOUND VERIFIZER.SYS850. To activate, enter SYS 850 (not the 828 as in the original program). To de-activate, use SYS 853.

If you are going to use tape to SAVE a program, you must de-activate (SYS 853) since VERIFIZER moves some of the internal pointers used during a SAVE operation. Attempting a SAVE without turning off VERIFIZER first will usually result in a crash. If you wish to use VERIFIZER again after using the tape, you'll have to reload it with the OPEN1:CLOSE1 commands.

Plus 4 VERIFIZER

```
NI 1000 rem * data loader for "verifizer +4"
PM 1010 rem * commodore plus/4 version
EE 1020 graphic 1: scnlr: graphic 0: rem make room for code
NH 1030 cs = 0
JI 1040 for j = 4096 to 4216: read x: poke j,x: ch = ch + x: next
AP 1050 if ch<>13146 then print "checksum error": stop
NP 1060 print "sys 4096: rem to enable"
JC 1070 print "sys 4099: rem to disable"
ID 1080 end
PL 1090 data 76, 14, 16, 165, 211, 141, 2, 3
CA 1100 data 165, 212, 141, 3, 3, 96, 173, 3
OD 1110 data 3, 201, 16, 240, 17, 133, 212, 173
LP 1120 data 2, 3, 133, 211, 169, 39, 141, 2
EK 1130 data 3, 169, 16, 141, 3, 3, 96, 165
```

C128 VERIFIZER

```
CF 1000 rem * data loader for verifizer 128
HA 1010 rem * commodore c128 - 40 and 80 column mode
DH 1020 cs = 0
HL 1030 for j = 3072 to 3226: read x: poke j,x: cs = cs + x: next
CB 1040 if cs<>19526 then print "checksum error!": stop
CP 1050 print "sys 3072,1: rem to enable"
CB 1060 print "sys 3072,0: rem to disable"
ME 1070 rem
FG 1080 data 201, 0, 208, 13, 120, 165, 253, 141
FK 1090 data 20, 3, 165, 254, 141, 21, 3, 88
MD 1100 data 96, 120, 173, 21, 3, 201, 12, 240
OJ 1110 data 17, 133, 254, 173, 20, 3, 133, 253
MF 1120 data 169, 44, 141, 20, 3, 169, 12, 141
OM 1130 data 21, 3, 88, 96, 165, 240, 201, 13
EI 1140 data 208, 94, 165, 22, 133, 250, 162, 0
ON 1150 data 160, 0, 189, 0, 2, 201, 48, 144
NH 1160 data 7, 201, 58, 176, 3, 232, 208, 242
IJ 1170 data 189, 0, 2, 240, 22, 201, 32, 240
ML 1180 data 15, 133, 252, 200, 152, 41, 3, 133
DE 1190 data 251, 32, 147, 12, 198, 251, 16, 249
DN 1200 data 232, 208, 229, 56, 32, 240, 255, 169
LM 1210 data 19, 32, 210, 255, 169, 18, 32, 210
LE 1220 data 255, 165, 250, 41, 15, 24, 105, 193
HC 1230 data 32, 210, 255, 165, 250, 74, 74, 74
KE 1240 data 74, 24, 105, 193, 32, 210, 255, 169
OF 1250 data 146, 32, 210, 255, 24, 32, 240, 255
NC 1260 data 108, 253, 0, 165, 252, 24, 101, 250
LF 1270 data 133, 250, 96
```

B128 VERIFIZER

Elizabeth Deal, Malvern, PA

```
1 rem save "@0:verifizerb128",8
10 rem* data loader for "verifizer b128" *
20 cs = 0
30 bank 15:for i = 1024 to 1163:read a:poke i,a
40 cs = cs + a:next i
50 if cs<>16828 then print "** data error **": end
60 rem bank 15: sys 1024
70 end
1000 data 76, 14, 4, 165, 251, 141, 130, 2, 165, 252
1010 data 141, 131, 2, 96, 173, 130, 2, 201, 39, 240
1020 data 17, 133, 251, 173, 131, 2, 133, 252, 169, 39
1030 data 141, 130, 2, 169, 4, 141, 131, 2, 96, 165
1040 data 1, 72, 162, 1, 134, 1, 202, 165, 27, 133
1050 data 233, 32, 118, 4, 234, 177, 136, 240, 22, 201
1060 data 32, 240, 15, 133, 235, 232, 138, 41, 3, 133
1070 data 234, 32, 110, 4, 198, 234, 16, 249, 200, 208
1080 data 230, 165, 233, 41, 15, 24, 105, 193, 141, 0
1090 data 208, 165, 233, 74, 74, 74, 74, 24, 105, 193
1100 data 141, 1, 208, 24, 104, 133, 1, 108, 251, 0
1110 data 165, 235, 24, 101, 233, 133, 233, 96, 165, 136
1120 data 164, 137, 133, 133, 132, 134, 32, 38, 186, 24
1130 data 32, 78, 141, 165, 133, 56, 229, 136, 168, 96
1140 data 170, 170, 170, 170
```

Bits and Pieces

Got an interesting programming tip, short routine, or an unknown bit of Commodore trivia? Send it in – if we use it in the Bits & Pieces column, we'll credit you in the column and send you a free one-year's subscription to The Transactor

C-128 Program Entry Tips

David J. Tajkowski
Baldwinsville, NY

Here are some tips for C-128 users who enter magazine program listings using line verification utilities such as Transactor's VERIFIZER. These tips can help make program entry less painful and time-consuming.

The first is almost too easy – if you are in eighty-column mode simply type the command FAST. This will reduce the noticeable delay from the time you hit enter until the cursor returns or the next line number appears (if you are using automatic line numbering).

The next tips concern the use of the function keys. The placement of the C-128's function keys just above the numeric keypad and the programmability of the keys can greatly ease the entry of programs. For example, when entering BASIC loaders which require many lines of DATA statements, re-define F1 as 'DATA' and F2 as ','; then it will not be necessary for your hand to leave the area of the keypad at all.

It is sometimes necessary to "go back" and re-check the verification codes of parts or all of a program. Changing the definition of F7 to eliminate the CHR\$(13) will make listing ranges of lines easier by eliminating the need to type in the word LIST.

Lastly, you can automate the entire process even more by creating a disk just for program entry. Set up the disk to auto-boot the verification program (VERIFIZER) and add the function key re-definitions to the program itself. Then when the disk is booted you will be set for fast, efficient and accurate entry of programs.

Stop Amiga Disk-Death

One of the most common causes of Amiga disk failure is pressing the eject button while the drive light is on. This can mess up important pointers on the disk, making it totally unreadable. Ejecting a disk at the wrong time is easy to do, especially since versions 1.0 and 1.1 of the operating system often put up disk-change requesters before the drive light has gone out – oblige the prompt speedily and your reward can be a zapped disk. (This has been corrected in Kickstart 1.2.)

The general rule, then, is **never** press the drive eject button when the red light is on. Well, almost never. If the system crashes with

the light on and it looks like it won't be coming back up with a "guru meditation #" or by re-booting on its own, you should eject the disk before you re-boot with Amiga/Amiga/CTRL. A forced re-boot seems to zap the disk on occasion.

If the above advice comes too late and you already have a bad disk in your collection, you may be able to recover it with "DiskEd", a direct-access disk editor that comes with the developer's package for the Amiga. DiskEd requires that you understand the format of Amiga's file structure (Explained in an article next issue), but an automatic disk-fixer called DiskDoctor is on the new version 1.2 software support disk. Just say diskdoctor df0: (or df1:), and the program does its best to automatically fix up a bad disk the best it can. Having a disk restoring program on hand may save you a lot of re-typing time.

Easy PAL to CBM Source Conversion

A CBM assembler format source file is just a sequential text file with no line numbers, while the PAL assembler uses a format identical to a BASIC program. A surprisingly easy way to create a CBM-format file on disk from a PAL file in memory involves just one POKE to list the program without line numbers. On the C-64, the entire conversion process can be accomplished like this:

```
open 1,8,2,"filename,s,w":cmd 1: poke 22,35: list  
(then poke 22,25 to return to normal listing mode)
```

On PET machines, the POKE is 19,32; and 19,22 to get back to normal.

Micro STP

Frank DiGioia, Stone Mountain, Ga

Chris Zamara published a nifty ML program in Transactor Vol. 5, Issue 6 called STP. Later, in volume 6, issue 4, Jack Weaver submitted a 10 line BASIC program to Bits & Pieces which worked similarly to Chris's STP. Well, I couldn't resist adding my two cent's worth. Here is a one-line BASIC program that I wrote which has all the power of Chris's original program:

```
open 2,8,2,"file":poke 781,2: poke 812,73: sys 65478
```

When you use it to execute a sequential file of BASIC direct mode commands, it will execute all the commands in the file and then attempt to execute blank lines until you hit STOP/RESTORE.

When used to merge or tokenize a sequential program listing from disk, it will perform the desired function and then terminate at end of file with a syntax error (caused by the "READY." produced by LIST). Not bad for a BASIC one-liner.

(Not bad indeed! What's more, you can go one step further and execute any series of commands from a sequential file without having to RESTORE. Just end the command file with CLOSE2:POKE812,47:SYS65484, and the sequence will terminate properly, closing the file and returning the system to normal! Other than the fact that it displays the lines being entered, the "nifty" STP program has nothing on this one! -CZ)

Border Cross

Steve Lofton, Centralia, IL

Computing around one day, I was wishing, "It sure would be nice if I could afford a Transactor subscription. But they're BIG league, I said to myself." My hopes of winning a Bits & Pieces scholarship crashed. Then my Commodore 64 whispered to me.

"Human unit, don't feel chr\$(17) and chr\$(31), send in that cute IRQ routine you were working on yesterday."

"Okay", I said, stroking her keys, "I'll give it a try, Connie Commodore."

So here it is. This program will scroll a multicolour bar in the border whenever a key is being held down. This adds new fun when you cursor to and fro.

```
10 rem border cross - steve lofton
20 for i=49152 to 49173: reada: pokei,a: next
30 data 169, 11, 141, 143, 2, 169, 192, 141
40 data 144, 2, 96, 160, 0, 206, 32, 208
50 data 200, 208, 250, 76, 72, 235
```

Disk Protection

Ric O'Neill, Calgary, AB

Hiding on track 18, sector 00 of disks formatted on 1541 drives are many important byte values used by DOS. Manipulating these values can have a dramatic effect on how DOS interprets the data read from the disk.

One particular byte of interest is the third byte on sector 00. This byte indicates to DOS what type of drive was used to format the disk. DOS uses this byte to determine if it can write to the disk. If it finds an illegal value for the DOS type it will not allow the drive to write to the disk. This so-called "soft protect" has been used by some to prevent writing to a disk. This is done by changing the value of byte three to almost any value other than 65. It seems to be an accepted fact that the 1541 disk drive cannot write to a disk formatted with any DOS type other than 65.

This is not correct. Disassembly of the code that checks the DOS type reveals that a DOS type of 00 can be written to also. DOS evidently checks for a DOS type of 00 first and if found continues the write job. Here's the section of code:

```
$D530 LDA $0101 Load DOS type for drive # 0
$D533 BCC $D538 Branch to $D538
$D535 LDA $0102 Load DOS type for drive # 1
$D538 BEQ $D53F If DOS type is 00 branch to write job
$D53A CMP $FED5 If not compare to type 65
$D53D BNE $D572 If neither branch to error routine
$D53F TXA Continue write job
```

Some C-128 Tips Andy Hochheimer, Wallaceburg, Ont.

I found that you can enter the 128's Machine Language Monitor by holding down the RUN/STOP key at power-up.

I've also found that some 80-column display programs don't take advantage of the FAST command. They instruct you to insert the disk, then power up with the 40/80 key down. Instead try this: Depress the 40/80 display key, power up, type FAST, then insert the program disk and enter the command BOOT. The program will boot just as if you had the disk in as they instruct. Some programs really benefit from this.

Disabling The C-128 RUN Key

**George Leotti
Glenolden, PA**

Have you 128'ers ever pressed SHIFT-RUN/STOP by mistake, and lost what was in memory? Here is my solution to this mis-key:

```
poke 4104,0 disables RUN key
poke 4104,9 enables RUN key
```

This works by changing the number of characters in the RUN key's definition to zero, effectively disabling the key.

For more on defining the RUN and HELP keys, see last issue's Bits & Pieces section. -CZ

C-64 Upper/Lowercase Timescroll

**Jimmy Watters,
Sussex, NB**

Here is a neat "time scroll" for the C-64. Type it in and RUN it, making sure to enter the correct number of spaces. The more text on the screen, the better the result.

```
10 print "n";:[16 spaces]print "N";:[16 spaces]goto 10
20 rem each ":" is followed by 16 spaces
```

Try changing the number of spaces to get different effects. Pressing a key changes things too. To really notice what is happening, add the following line:

```
5 for i=1 to 999: print chr$(169);: next
```

Free C-128 CP/M Manual

**Paul Reeves,
Hamilton, Ont.**

I have stumbled across a very helpful piece of information which will help all of you who hate spending 20-30 dollars on computer

manuals. If there was only a way to make one yourself. . . Now there is; to get a "FREE" 40 page CP/M manual for the C-128, put the CP/M System Disk in drive A and type the following:

For a C-128 and a 1541:

```
(A>) pip lst:=help.hlp
```

For a C-128 and a 1571:

```
(A>) pip e:=a:help.*
(A>) help [extract]
(A>) lst:=help.dat
```

With the 1541, the manual will have a few pages of junk at first but after that all will be fine. When using the 1571 version, it will ask you to insert disk E into drive A; when it does, insert a blank disk formatted double-sided. You need to do this because all the information will not fit on one side (the Help.HLP file is 83K).

Non-Random Surprises

**Harold Anderson,
Oakville, Ont.**

Type this into your C64 and run it:

```
10 n=rnd(0)*1024
20 poke 1024+n,24: poke 55296+n,1
30 goto 10
```

You would expect to have the whole screen filled in a random manner with white X's. Surprisingly, what you will find is that 3/4 of the locations are never filled and you end up with a pattern of vertical stripes on the screen. Clearly, the number returned by RND(0) is not very random. If you change RND(0) to RND(1) the problem disappears.

I investigated this at some length and found that the number returned by RND(0) is of the form:

```
RND(0)= A × 1/256 + B × 1/(256 × 256 × 256)
Where A is any integer 0 to 255
and B is any integer 0 to 31
```

This means that there are only 256×64 (16384) different values of RND(0). Also notice that the second term in the above formula is *much* smaller than the first. The result of this fact is that the possible values of RND(0) are grouped in tight bundles around the values indicated by the first term of the formula. This bunching explains why the screen is not filled evenly in the test program.

The values of A and B in the formula are apparently derived from the time, since messing with the length of time between successive calls for RND(0) seems to affect the randomness even further.

The RND function on the C64 should be regarded as follows: A call for RND(0) gives one of 16384 possible starting "seed" values for 16384 possible random series. A call for RND(1) gives a value derived from the last value found by the RND function. While RND(1) returns values which are truly randomly spaced, each successive value is derived from the previous value according to

certain exact rules. Hence if two series start with the same seed, they will be identical through their entire length.

Watchacallit?

J.G. Krol, Anaheim, CA

The symbol * is widely used in computer programming, as to indicate multiplication. The name of * is not "star". It is called *asterisk*.

The symbol / is also widely used, as to indicate division or optional selection. The name of / is not "slash". It has two different names depending on its usage. When / is used mathematically to indicate division, like $4/5 = 0.8$, it is called *solidus*. When / is used to indicate optional selection, like his/her, left/right, or and/or, it is called *virgule*.

A unit of time that is 1/60 (that's a solidus) of a second is widely used within Commodore computers. It is called in Commodore documentation a "jiffy". That is an unnecessary neologism. Ancient Babylonian astronomers divided an hour of time or a degree of angle into 60 equal parts, each called a *minute* (MIN-ute), which name is cognate to minute (my-NOOT) meaning very small. For a smaller unit they divided a minute into 60 equal parts. That took a second division by 60, so they called the unit a *second*. For a yet smaller unit, they divided a second into 60 equal parts. That took a third division by 60, so they called the unit a *third*. This "sexagesimal" system is still widely used 5000 years later for measuring time and angle. It can be extended to "fourth", "fifth", and so on as needed. Thus a so-called "jiffy" is actually a *third*, except that the news hasn't reached Commodore's tech writers yet, not after 5000 years, which shows just how slow the mail is these days.

Mr. Krol is right - I looked it all up in a dictionary. But please, programmer's jargon is confusing enough. You want to start using Ancient Babylonian? -CZ

Quicker Assembly on the Amiga

**Norman MacDonald
Medicine Hat, Alberta**

While using the Amiga Assembler I was disappointed at the amount of time it took to assemble a short program when some of the 'include' files were included in my source file. I have come up with a simple solution to the problem that cut the assembler's workspace and the assembly time in half. Here is what I did: I wished to use some of the Intuition structures for setting up a screen and window, so I made a source file as follows:

```
include 'exec/types.i'
include 'intuition/intuition.i'
```

You may wish to add other include files than the ones I used, but be sure to have 'exec/types.i' as the first include file. When my source file was set up, I assembled the source file above with the following lines from CLI:

```
assem :infile -c w250000 -i :include -e :outfile.equ
```

This assembled an "EQU file" of all symbols that were present in the source file. This "EQU file" may then be used in your source

and will perform exactly as the include files would. You may even wish to examine the file and trim some more fat off it by deleting the definitions that you do not use.

The advantages of doing this is a large decrease in the assembly time and less workspace is required to assemble a program. The main disadvantages to doing this is that you do not have access to the macros defined in the 'include files'. Another disadvantage is that if you include two of these "EQU files" in an assembly and they contain duplicate symbols, you will end up with some errors during assembly. One solution to this problem is to change all the EQUs in the EQU file to SETs; this will allow the symbols to be defined more than once.

This is the fastest method of decreasing assembly time on the Amiga that I have found. I hope it will be of use to anyone doing assembly work on the Amiga.

A further increase in speed can be obtained by making an "include" directory in the RAM-disk ("makedir ram:include"), copying the include (or EQU) files that you need to "ram:include", and using "ram:include" instead of ":include" as the "-" assembly option. If you have expansion RAM beyond the built-in 512K, you can put everything in the RAM disk (source, libraries, include files, object, etc.) and get super-fast assemblies. The same applies for compiling. Even if you don't have extra RAM (a 2 Meg box is nice, but expensive), try squeezing as much as possible into RAM before assembling. You'd be surprised at how much faster things go when the system doesn't spend its time waiting for the read/write head to seek. -CZ

Changing Baud Rates For The C-64 with 1650/6240 Modems

James Shelley
South Bend, In.

After reading Tony Valeri's article in Volume 6 Issue 2 and the tip from Daniel Bingamon on modem speed-up to 450 baud, I thought a different approach might be of interest. Several of my friends and I have been working on a terminal program to use with our BBS. We found it desirable to change the baud rate in our terminals and BBS while on-line. As the 1650/6240 modems can't communicate at 450 baud while set at 300, a way was needed to change the baud rate and not re-run the terminal program or BBS or close and re-open the RS-232 file. We use the four lines of BASIC below. It will change the baud rate on the terminal and the BBS without any ill effects. This method has been tested to 1200 bps with good results. This does not change any parity, word length, etc. When the user logs off the BBS, a GOSUB in the log-off routine passes through these lines with the variable 'baud' set to 300, and the BBS is back to 300, ready for the next call.

```
10 open 5,2,3,chr$(6) + chr$(0)
20 :
30 rem - more pgm. lines -
40 :
50 input " baud rate > " ;baud
60 a = int(1022730/baud/2 - 100) : c = int(1022730/baud)
70 d = int(a/256) : e = a - d * 256 : poke 661,e : poke 662,d
80 d = int(c/256) : e = c - d * 256 : poke 665,e : poke 666,d
```

If the programmer's reference guide is consulted, it is obvious how the memory locations are used. Locations \$0295 - \$0296 are "RS-232 non-standard BPS". Locations \$0299 - \$029A are "RS-232 Baud rate". With about 40-50 terminal programs, along with the programmer's reference guide and The Transactor, we came up with this bit of painless BASIC code. I hope it helps ease the pain of those who have had a chance to try and swallow the Programmer's Reference Guide's advice on Baud change.

Borrowing Money?

After those fun programs, here's something that's actually useful. Given the rate of interest of a loan and number of payments, it prints a table with monthly payments from \$10 to \$1000 per month in one column, and the amount of the loan in another. With this table at your disposal, you can find out how much you want to borrow based on what you are willing to pay each month. Alternatively, you can look along the right-hand column until you find out how much you need to borrow, then look across to the left column to see what it'll cost you for each payment. So, if you're buying a new car, you can scan from Honda up to Porsche until you see what you can afford.

The program formats the numbers so that the columns line up at the decimal point regardless of the values. As listed, it prints table to the screen - change the '3' in line 120 to '4' to get a print-out. The payments on the left side go from \$10 to \$1000 in steps of \$10 - you can change this with the for-next loop in line 200

```
PD 100 rem** loan principal - cz '86 **
AD 110 s$ = "[13 spaces]":l = 10
DK 120 open 1,3: rem 3 = screen, 4 = printer
CJ 130 input " annual interest (%)[4 spcs]12
    [4 crsr lefts]":ai
DD 140 input " number of payments[5 spcs]48
    [4 crsr lefts]":n
CO 150 i = ai/1200 :rem monthly interest
OO 160 print#1,chr$(13): rem blank lines
JF 170 print#1," interest = " ai "% , # payments = " n
OF 180 print#1," payment " , " amount of loan "
CD 190 :
CM 200 for pv = 10 to 1000 step 10
BG 210 rem compute amount of loan
MN 220 p = pv*(1-(1+i)^(-n))/i
CM 230 rem format amount field
KA 240 p$ = right$(s$ + str$(int(p*100 + .5)),l)
NK 250 print#1,pv,left$(p$,len(p$)-2) " . " right$(p$,2)
AF 260 next pv
LL 270 print#1: close1
```

Letters

Column For Machine Language Sub-Routines?: I think that it is fair to say that The Transactor caters somewhat to the machine language programmer. How about a column devoted to machine language subroutines such as division, multiplication, base conversions, memory moves, etc. Published routines could be divided into two categories:

- 1) The shortest – for programmers trying to cram as much into a small place as possible.
- 2) The fastest – measured in machine cycles, for the programmer who needs blinding speed.

Any routine could be updated if it's faster or shorter than its predecessor. The result of all of this would be a fantastic library of subroutines that would be useful to any ML programmer. Who knows, these routines could even be compiled into a book at a future time! There are plenty of clever ML programmers out there sitting on hundreds of useful routines – why re-invent the wheel?

Rick Nash, Millersburg, Ohio

Sounds interesting enough. At the moment we tend to include these in the Bits and Pieces section – perhaps a sub-section of Bits could be used to host your idea. Of course it all depends on what kind of mail we receive, so, starting now, if we get enough fast and short MLs, we'll group 'em like you say. Same deal as Bits – free subscriptions for those published. Thanks for the input!

New Testament Available On 1541 Disk Format: *Once in a while, I receive a telephone call or letter that makes my day. Today was such a day. Randall J. Bernard in Arizona called to thank us for running his letter with regards to the help he required to key in the New Testament (see Volume 7, Issue 01). He said that he now has the entire New Testament on four diskettes and has 60 people working on keying in the Old Testament. All that was still required was final editing of the New Testament and waiting for the keying in of the Old Testament to be completed by all the people who volunteered. He was overwhelmed by the response generated by his single letter. In truth, it also overwhelmed me.*

According to Randall, the final editing stage would be the most time consuming thing to come. One suggestion was to ask the help of a few Bible study groups to check for errors and omissions. If anything, this would ensure an accurate transcription of the Testament. Randall asked if I could mention that anyone who wanted the New Testament on disk could have it for a phone call. Now, I don't know about you but I feel that Randall has already put enough into this venture. Between the time and effort that he has already expended, and the additional time and effort from his volunteers, I feel that it would be unjust to expect Randall to pay for all the expenses. Randall mentioned to me, though, that it would be morally wrong to charge anyone for the Bible. In this respect I completely agree.

A nice way around this problem could be that anyone who wants the New Testament, and later the Old Testament, on diskette,

should supply Randall with an equal number of diskettes (4 for the New and approximately 10 for the Old) plus return postage. For people outside of the US, remember that only US postage is good in the USA. Ask your post office for help. They will gladly oblige. One additional thing to consider: Randall will have to pay for mailers to mail your diskettes. Any small amount that you could send towards the cost of the mailers would surely be appreciated. To contact Randall:

Randall J. Bernard
Box 630
Morenci, Arizona, 85540
(602) 865-3550

One final thought. Many different religious backgrounds make up the world. As such, I feel that Randall's efforts should only be the beginning. It would be beneficial for so many people to also have English translations on disk for the Torah, Talmud, Koran and other religious scriptures. If anyone would like to begin efforts similar to that of Randall, please drop us a letter. If you are fully willing to expend the effort, we will be more than willing to spread the word.
Richard Evers

Flexible Vector Problem: Murphy's law has intervened in my efforts to use your Flexible Vector program (Transactor Volume 6 Issue 02 page 64).

Actually this same law has a corollary originated by Theophilus Cole during the last century and it has become quite well known in some circles as Coles Law which, with the spaces removed represents a statement much more palatable than Murphy's law, even though it has the same meaning.

Although I have copied the code with due vigilance and some attention to detail, I am unable to obtain the results expected. The code assembles well with PAL and responds to SYS by placing the 40-digit message on the top line. The difficulty is with the non-erasability which doesn't exist in my rendition of the program. Scrolling North erases lines, ultimately including the top-line message. Run-Stop/Restore has the same effect.

As an earnest novice at assembly language, I would greatly appreciate your advice in pointing to the error I must have made which prevents the non-erasure option from working.

Bob Tischer, Starkville, MS

Below is a corrected version of the 'Flexible Vector Management' program. The problems you encountered were not your fault. At the time, Chris forgot one important fact of life with the 64; the older versions of the 64 had problems with colour and screen memory. Line #'s 410-420 will solve this problem. Colour and screen memory are both updated during each interrupt to display the message in white at the top of the screen.

```

100 rem save "0:irq message",8
110 rem ** chris zamara - volume 6 issue 02 page 64
120 :
130 sys 700
140 .opt oo
150 *      =   $c000
160 ;
170 irqvec =   $0314
180 ;
190 ;** initialization routine **
200     sei          ;disable interrupts
210     lda #<newirq ;low byte of destination address
220     sta irqvec   ;irq vector low
230     lda #>newirq ;hi byte of destination address
240     sta irqvec + 1 ;irq vector hi
250     cli          ;enable interrupts
260     rts         ;back to the calling prg or basic
270 ;
280 ;** new irq routine **
290 newirq = *
300     jsr displin ;do disp routine
310     jmp $ea31   ;normal irq dest
320 ;
330 ;** irq vector-driven routine **
340 ;displays message on top scrn line
350 displin = *
360     ldx #39      ;message = 40 char (0-39)
370 ;
380 sfil = *
390     lda message,x ;next char from message
400     sta $0400,x   ;store in next scr mem byte
410     lda #1        ;** char colour white **
420     sta $d800,x   ;** store it in colour memory **
430     dex          ;do all 40
440     bpl sfil     ;byte #0 is the last one
450     rts
460 ;
470 message .asc "0123456789012345678901234567890
123456789"
  
```

G-Link Info: I think I want a G-Link Interface, as described in the May, 1986 Transactor, but need more information.

My purpose is to connect a C-64 computer to an MSD SD-2 dual disk drive, using the parallel IEEE port to shorten disk access time. Questions are:

1. Does the G-Link connect to the C-64 expansion for parallel operation?
2. If so, what speed advantage can I expect?
3. What additional cable must I supply?
4. Is necessary software included? (The article stated complete transparency.)
5. Has the G-Link been used with the MSD drive with success? Just what can I expect?

Willard E. Wright, Bellevue, Washington

The G-Link IEEE-488 interface connects to the C-64's cartridge port to provide true, transparent IEEE-488 bus access. The only extra that you will need to make use of our interface is a single Pet to IEEE cable. You can expect a data transfer rate increase, in relation to a normal 1541, in the area of 50 to 650 percent, depending on the drive used. The G-Link has been used with the MSD drives with quite a degree of success.

Recently, I wrote an in-depth article about the G-Link IEEE-488 interface to run in this issue. Unfortunately, it was decided that our reading audience contained too few people who would benefit from this article. I can honestly state that the decision to cut my article really hurt.

To make amends to everyone who would have liked to learn a bit more about the G-Link, we are offering it to you for the price of a letter, phone call or visit to our office (we'll also be including it with future G-Link orders). If you are at all interested in the inner workings of the G-Link, or just would like to learn a bit more about the IEEE-488 bus, then drop us a line. It would be nice to find that my efforts were not wasted.

The SID Chip: Does It Really Exist?

A long and frustrating story made as brief as possible:

10/4/85 - I sent a money order for \$28.00 to Commodore Business Machines, C-2659, 1200 Wilson Dr., West Chester, PA 19380 to cover purchase and shipping of 1 6581 SID chip, pn 906112-01.

Result: 6 weeks later, no chip.

C. 11/15/85 - I send a letter inquiring about the status of the above order.

Result: nothing.

C. 12/15/85 - Another letter amplifying first.

Result: silence.

1/31/86 - I send tracer to Postal Service on money order.

Result: money order was cashed by Commodore Business Machines 11/8/85. Status of chip: nowhere to be found.

I am sending this letter to Commodore and also several magazines that serve Commodore users because I am hoping that if I can't get a response out of CBM, I can at least air my complaint in public.

Calling (215) 436-4200 is a waste of time. The phone is almost always busy and when it rings, no one answers.

This particular part cannot be purchased from any third party that I know of. I can almost live without sound but some software uses the random number generator in voice 3.

Roy M. Randall
 6032 Edsall Road #203
 Alexandria, Virginia 22304

What can I say? Obviously, your order went missing and nobody has been designated to trace it back. I know for a fact that quite a few people at Commodore read The Transactor. Perhaps somebody from CBM will contact Roy and help straighten this out. If all else fails, try dialing (215) 431-9100. This number at 1200 Wilson Drive, West Chester does work and is usually not busy.

Transactor Disk Directorytions: I just received the disk for Volume 5, Issues 1-3. When I put the diskette in my disk drive, I write on my computer:

```
LOAD "*" ,8
and
LOAD "*" ,8,1
```

Then when I write RUN the computer comes back to READY. When I LIST, I get. . .

```
10 REM --- VOLUME 5 ISSUE 1 ---
READY.
```

Please Help! What is the problem? My computer is a Commodore C-64. It is not the first time that I LOAD a diskette. Could it be that the program is not written on the diskette. I am waiting for your answer.

Roger Laplante, Chateauguay, Quebec

There are 118 directory entries on that particular diskette. Of these entries, 115 are usable programs for some type of Commodore computer. Of these programs, many are Bits and Pieces items, others are simply to be Loaded and Run, then the balance are source code for the concepts covered in the related issue. You really need to read each issue before attempting to use the programs in such an elementary manner. The program that you Loaded was the first program on disk. However, it is not really a program but rather a marker that merely acts as a header to the files that come after it. If you were to look at the directory, you would have noticed that the filename listed is the Volume and Issue number covered on that portion of the disk. If you were to look through the directory even further, you would find that similar markers exist for the start of entries for Volume 5 Issue 02 and Issue 03.

There was a second reason for saving this marker as the first entry in the directory: since it serves no other purpose it can be scratched and replaced by some other more useful program that you would rather have as the first program in the directory. Using LOAD "" would then give you access to your "favourite" program without entering the entire filename.*

Super Empty SuperPET: We bought a SuperPET last fall, which helps us in our office. We do not have any programs to go with it, so we have to create every program for use in the office.

Is it possible to get from you or some other people any programs such as: inventory, mailing list and programs for office use?

Bruno Cote, 80 Leroux #104, Granby, Quebec, J2J-1S1

There are hundreds of programs available for the SuperPET. The trick is to find them. In my own personal collection I have many of the word processors, spread sheet programs, data base packages, accounting systems and such that have appeared in past. The trouble is, passing them about gratis is not too kosher. My suggestion is to locate a copy of the 'Commodore Software Encyclopedia' by Sam's Books. It lists and describes many of the packages written for all of the Commodore computers plus provides addresses for the suppliers. A few good company names to remember in your quest is Batteries Included, BMB Compuscience, BPI Micro-Systems, Handic Software, ProLine Software, and Softwerx. These are but a few of the good software manufacturers and suppliers for

the SuperPET, but whether they still stock programs may be another matter.

Verfizer for the C128: I couldn't get the C128 version of verifier going. Is it me or you? I double double checked everything twice but I still could not get it to work. I finally disassembled it and found that it starts with a branch. Is that right?

Mike Iafrate, Parkersburg, WV

Verfizer for the C128 works as listed - to make sure, I keyed it in verbatim from the pages. Remember, you must be in 40 column mode. To activate, "SYS3072,1" will do the trick (actually, anything from 1-255 will work as the parameter after the comma). To de-activate, SYS3072,0 does it. I have noticed one problem, though. SYS3072,0 does not always do it on the first shot. Twice in a row always makes sure that it is turned off.

The branch at the beginning of the code is correct. The C128 allows passing of the accumulator, x and y registers plus status register via the SYS call. The code begins with - BNE INSTALL - In simple terms, if the accumulator is greater than zero then activate the routine else de-activate. A little bit simpler than the 64.

Disk Un-Assembler Once Again: Thanks to Transactor and J. Lothian for the Disk Un-Assembler for the Commodore 64 in The Transactor (Volume 6 Issue 04). Thanks also to John R. Menke for his letter in The Transactor (Volume 6 Issue 06).

The program of Mr. Lothian and letter of Mr. Menke both contain small errors. They both recommend adding line 61:

```
61 MD(36)=0:MD(44)=0
```

This cannot be done as the arrays are first dimensioned in line 120, and values are read into the array in line 130.

I agree with Mr. Menke that BIT must be converted to BYTE with the Commodore assembler. He is incorrect, however, when he says:

"I note that lines 2020-2060 are never accessed in the un-assembler, which is a shame because they would signal that a BIT operation was converted to a .BYTE. Also, lines 310-360 in the program are excess baggage, although it would be nicer if that weren't the case."

The simple answer to the BIT to .BYTE problem is DON'T ADD LINE 61. The best solution is to change line 1430 and line 2050.

```
1430 if n>10 the on(n-10) gosub 1950, 1980, 2010, 2040
```

In line 2050, reduce the (8 spaces) to (1 space) and as Mr. Menke mentioned in line 1480, be sure to include a space between .BYTE and \$ in line 2050 also.

With these modifications, you can un-assemble using EITHER BIT or .BYTE and if (BIT to .BYTE) is selected in lines 330-360, the program will signal that a BIT operation was converted to .BYTE. The MD(36)= 14 and MD(44)= 14 in line 360 accomplishes this in line 1430, which accesses line 2040 to signal the change. Any other .BYTE directives are listed normally.

Patrick P. Malloy, Milwaukee, WI

Super Kit Users Notes: *We have received quite a few letters and phone calls from people who have bought SuperKit/1541 through the magazine. It seems a few people have had problems getting SuperKit to duplicate itself as stated in the manual. Well, according to the people at Prism, SuperKit is quite capable of this task. The trouble may be that you have more than one Commodore 64, one 1541, and one monitor in your system, or that some hardware mod or plug in cartridge is causing problems, or your disk drive is slightly out to lunch. As a final might-be, it could be that the disk you received is indeed bad. Many different factors could be causing you problems. SuperKit, with its vast speed and power seems to be very particular about system configuration. According to Prism, a re-write is under way that will straighten this problem out.*

Now, to cover each problem sequentially. If you have a printer, extra disk drive, modem or virtually any extra peripheral device connected in any way to your computer or drive (even if it is not turned on), this could cause problems. The Epson and Gemini printers are the biggest trouble makers for SuperKit. Disconnect everything extra completely from your system. Absolutely remove the connectors and your problems might vanish. Please remember that extra peripherals also include Fast Load cartridges and such that are always taken for granted and forgotten. SuperKit is fast enough without any additional help.

Hardware mods could include a new set of ROMs for your 64 or 1541 that have made improvements on Commodore's design. There are lots of super ROMs out there that beef up the I/O handling plus provide all sorts of programming goodies, but SuperKit hates them. To get all the neat goodies for you to play with, the ROM must contain substantially altered code from the original. Normal ROM entry points are no longer, some features have been completely removed to make way for the neat stuff and pieces of code that are involved in critically timed routines are no longer so critical. Any number of mods to your ROMs could affect SuperKit, and for that matter any software package, with unpredictable results. If you know that your system and SuperKit do not cohabitate well together, then return the package to us (with sales receipt included). It makes little sense for you to retain a package that cannot live with your system.

The final problem that we know about could be that your drive is not as tight as it was when conceived. The head could slap around a bit too much, the belt could have a bit of slack and any number of other factors could turn your drive into something SuperKit has difficulties dealing with. When trying to duplicate SuperKit using the Super Nibbler on side two of the diskette, set the maximum track number to 40. By telling SuperKit to bit copy as far out as track 40, a slight bit of internal drive slack will not be a problem. If your drive problems are worse than being just a little bit out to lunch, then perhaps a service call is in order.

One other note: after using the Super Nibbler, it's best to send an Initialize command to your drive. This brings the head back from the "far reaches".

Let's hope that these few points make life with SuperKit easier. It is truly a fine product, one that we gladly stand behind in every way. It's just a shame that it cannot live in peace with more than its immediate family. Perhaps the much desired re-write taking place in Texas right now will cure it of its unsociable attitude. We're pretty sure it will. You may also want to look at the two articles in this issue that address other aspects of SuperKit and other copy utilities.

Games From The Outside Out: I look forward every other month to receiving my Transactor. For the most part, it is far above average. Mostly, this letter is not as much of a "complaint" as it is a "reader's point of view".

The cover for September said "The Tech/News Journal For Commodore Computers". It also said "Games from the Inside Out" showing pictures of demons, fighter planes, tanks, etc.

Mentally, I combined the slogan of Transactor, the image of the dark secrets of Zork exposed, or Enchanter, or if not that then a close look at multicolour, fast action of Zaxxon, or a clue as to the fabulous programming techniques of Beach Head, or Seven Cities of Gold. Oh, the thrill, the anticipation. . .

Then, eagerly, I scan from cover to cover: One (1) Game! Animals? I had that 12 years ago (and a better version to boot) and IN BASIC!(!?)

Four valuable pages which could be spent telling me about Commodore were used for a useless ATARI review. Oh, the pain. If I want reviews, which I don't, I'll buy Compute!, which I don't.

Four more pages for Animals, and another wasted on silly cartoons. In those NINE pages you could have told a lot about Commodore.

I am especially interested in how things are done and why they are done in ML. A source of various ROM routines with comments could be a great teaching tool for a Transactor column, as could more information regarding the disk drive.

Please don't misunderstand - Transactor is by far the best, but it is the best only when it is different! Please, no more reviews (especially NON-COMS) and no more cartoons to waste valuable space!

Wayne Gurley, Wills Point, Texas

Another Vote Against The Atari Article: At the risk of being labeled an Amiga jingoist, I am a little distressed as to the inclusion of the "Atari ST Notebook" in the latest issue. If the article was included because the Atari is a 68000 machine, then by that logic, I expect we will start seeing articles on the Macintosh. One reason why I subscribe to your magazine is because of its narrow focus on Commodore products.

My chief complaint is that the pages used for the Atari article could have been used for an article on the Amiga operating system. The Amiga operating system is woefully lacking in documentation.

Despite this mild complaint, I think you are producing an excellent magazine. Keep up the good work.

J. Richard McEachern, Saint Louis, Missouri

Ouch. Being hit twice in a row about the Atari article really hurts! We ran the article because it was exceptionally well written, without any biased Atari hype. It did not contain any program listings, so it didn't require a machine in order to learn something from it.

Although it went against our morals to cross over to the other side for that article, we felt it was worth it. Jack Cole is such a good

author, and a master in R+D, that we had to run the article. At one point, we had been tampering with an idea for a 68000 magazine as an insert within the Transactor every issue. The extra magazine portion was abandoned, but Jack had already been working on the article to help us out. To finish the article Jack fought against a nasty flu bug to make deadline. Jack did pull through, and his Atari Notebook was the result. With such devotion, would it have been morally right for us to reject his efforts, especially after we had initially solicited his aid? We say NO!

It might also be interesting to note that the ST is more like a Commodore than anything Atari has built. Furthermore, the Amiga is more like an Atari than anything Commodore has produced. In fact, the Amiga project was originally commissioned by the pre-Tramiel Atari Corp. Jay Minor, who is mostly responsible for those three workhorse VLSI chips in the Amiga, was also the chief designer of the "Antic" chip in the early Atari machines. And the ST project is staffed mostly by former Commodore employees - it's even possible that they started the ST before becoming ex-Commodore. A twist of events like this leaves one pondering to be sure.

Referring back to Mr. Gurley's letter, it would take months to unravel the secrets of some games without source code from the author(s). Time like that we unfortunately just don't have. Quite simply, the bulk of any issue is a collection of submissions, sprinkled with the odd item that we have time to write ourselves. And lastly, although we can't promise to exclude the cartoons, we can promise that comparisons will take priority over reviews. The odd review may show up when it's important, but we will try to keep them to a minimum.

Rx For Grammar: There ain't no such word as "irregardless".

Dr. James N. Little, Walnut Ridge, Arkansas

Too bad. . . It really was such a nice word to have around.

Spel Cheker: Through the magazine in Vol 7 #1, you misspelled "colonel" as "kernal". It's "churnal", isn't it? You even misspelled it on the kovar.

Jeff Jourard, Santa Monica, California

TransBloopers

Bits & Pieces: Volume 7 Issue 02 page 12

In the Amiga Startup-Sequence on page 12, the first greater-than (>) in the DATE function argument should be a less-than (<).

Elininating SAVE@: Volume 7 Issue 02 page 33

The article did not mention that the complete ROM exists on the Transactor disk for that issue. The file is 16k long and contains a dump of the entire SAVE@-fix ROM for the 1541. It's a "PRG" type file with a start address of \$2000. Remember to strip the first two bytes if your EPROMer doesn't - otherwise the start address will be sent to the EPROM and throw everything off by two bytes!

Improving The SYS Command: Volume 7 Issue 01 page 64

The description of the routine at \$AD9E, described under the heading Routines used in passing parameters, said that the routine evaluates an expression and if the result is a string, puts it at \$0100. In reality, the string doesn't go here at all; the routine returns a pointer to the string descriptor in locations \$64 and \$65 (these point to three bytes containing the string's address low, address high, and length). Another point that should have been mentioned is that you should call \$B6A6 after \$AD9E to clean up the string descriptor stack. After calling \$B6A6, a pointer to the string itself will be in locations \$22 and \$23 (also in .X and .Y), and the accumulator will hold the length of the string. If you call \$B6A3 instead of \$B6A6, it will first check that the expression was of a string type.

SYMASS 3.1 Fixes

The Symass 3.1 Assembler on Transactor Disk #12 has a bug that makes it corrupt the symbol table when it encounters an expression defining a label as a function of another label (for example LENGTH = ENDTAB-TABLE). To fix Symass 3.1 and turn it into 3.13, LOAD in SYMASS 3.1, make the following POKES, and SAVE the updated version as SYMASS 3.13.

POKE 3304,56
 POKE 2965,234
 POKE 5057,51

(The above changes turn SYMASS 3.1 into 3.12, as described in the article "Save SYMASS Symbols" in Volume 7, Issue 02)

POKE 3312,98
 POKE 3316,99
 POKE 3318,98
 POKE 3323,98
 POKE 3328,99
 POKE 3332,99
 POKE 3370,98
 POKE 3389,98
 POKE 3393,98
 POKE 5057,51

C128 Memory Maps: Volume 7 Issue 01 page 29

Mike Mellinger from Connellsville, PA wrote in to draw our attention to a few more errors in these maps. Below is a list of the incorrect lines, with the corrections to the right.

D800-D8E7 should be D800-DBE7
 1178-1197 should be 1178-1179
 1271-1274 4721-4274 should be 1271-1274 4721-4724
 Decimal addresses from 2595 through 2602 should be 2597 through 2604 (they're all off by 2).

TransBASIC Installment #11

Nick Sullivan
Scarborough, Ont.

TransBASIC Notes

TransBASIC has been a regular Transactor feature for almost two years. Those who have been following the series know all about it. Recently, however, we've received letters to the effect of "what is TransBASIC?". Quite simply, TransBASIC is a method of adding new commands to BASIC (see "Part 1:" below). The commands come in 'modules' which may contain one or more commands OR functions. After merging the modules of your choice, the entire lot is assembled and linked into BASIC. The new commands can then be used just like any of the other commands that are already in the BASIC ROM when the C64 is powered up.

The TransBASIC Disk

The TransBASIC Disk contains all of the modules published so far and it comes with its own assembler, SYMASS 3.1. Any combination of modules can be linked into BASIC with only a few simple steps. From start to finish is usually no more than a couple of minutes. . . even less once you get the hang of it. It comes with a handy reference for just \$9.95. See the order card at center page.

Note: A bug has been found in the SYMASS Assembler that shows up when you define a label as a function of another label. So far we haven't noticed it while using the TransBASIC Disk, only with other unrelated source code. Check the Transbloopers column in this issue for the fix.

TransBASIC Parts 1 to 8 Summary:

Part 1: *The concept of TransBASIC – a custom command utility that allows one to choose from a library only those commands that are necessary for a particular task.*

Part 2: *The structure of a TransBASIC module – each TransBASIC module follows a format designed to make them simple to create and "mergeable" with other modules.*

Part 3: *ROM routines used by TransBASIC – many modules make use of ROM routines buried inside the Commodore 64. Part 3 explains how to use these routines when creating new modules.*

Part 4: *Using Numeric Expressions – details on how to make use of the evaluate expression ROM routine.*

Part 5: *Assembler Compatibility – TransBASIC modules are written in PAL Assembler format. Techniques for porting them to another assembler were discussed here.*

Part 6: *The USE Command – The command 'ADD' merges TransBASIC modules into text space. However, as more modules are ADDED, merging gets slow. The USE command was written to speed things up. USE also counts the number of statements and functions USED and updates the totals (source line 95) automatically.*

Part 7 – *Usually TransBASIC modules don't need to worry about interfering with one another. When two or more modules want to alter*

the same system vector, however, a potential crash situation exists. Part 7 deals with avoiding this problem.

Part 8 – *Describes the five modules for Part 8.*

Part 9 – *Describes the six modules for Part 9, and makes first mention of The TransBASIC Disk.*

Part 10 – *Describes the six modules for Part 10, and details some minor bugs in the modules "MC GRAPHICS", "MOVE & FILL", and "PRG MNGMNT".*

TransBASIC Installment #11

In this issue you'll find only one module, but it's a big one. The author is Paul Adams of Revelstoke, British Columbia, who wrote it to fill a personal need. Paul writes:

"I was writing a program to log my daily running (by foot, not by computer) and needed some commands that would allow better graphing capabilities than I could find. Of course, being an Engineer, I like to graph everything. . ."

Well, if you like to graph everything too, this is the module for you. Paul has come up with the interesting concept of combining two hi-res screen areas into one double-height drawing area for his graphs. His plotting commands have the capability of addressing this area with y-coordinates from 0 to 399 (to make things Cartesian, he has put the zero y-coordinate at the bottom rather than the top of the screen).

On this big plane you get the capability of drawing lines and rectangles, either solid or in a dot pattern of your choosing. You can also write text to the screen, including a variety of special characters that come in handy in graph work.

Because the hi-res screens take up 8000 bytes apiece, plus another 1000 bytes of video matrix for each one, you sacrifice a bit of BASIC memory when you use this module. In fact, before you start using the hi-res, you should give the command:

```
poke 56,140: poke 55,0: clr
```

to protect the BANK 2 video matrix at \$8C00 and your BASIC strings from writing all over each other.

A couple more things: if you should commit a syntax or other error while you're in one of the hi-res screens, you will have to type the DEFAULT command blind to get back. Alternatively you could include William Turner's TRAP command from last issue in your dialect, and make the DEFAULT command part of your TRAP routine.

Drawing large rectangles can take quite a while in hi-res, especially when they're completely solid rather than patterned. I found myself

wanting a way to switch between the upper and lower sections of the drawing area so that I could monitor what was going on, so I added a few lines to the plotting routine that will cause the screens to be swapped when you press and release the CTRL key (the swap actually happens on the release). This feature is only operational while a plot is actually in progress. Sometimes I also wanted to see debugging information that I was printing to the text screen during a series of plots, so I also added a trap for the LOGO key. The text screen is made visible whenever this key is held down during a plotting operation; the current hi-res screen is restored when the key is released.

If you look in the code for DEFAULT routine (in the neighbourhood of program line 13080), you'll see a patch I had to put in to cure a very odd hardware-related bug that occurred in the very standard routine that switches video banks from BANK 3 to BANK 0. The net effect of the bug was that the RTS at the end of that routine was frequently executed as a JSR (a \$20 rather than \$60), which in this case was a JSR into the never-never. The patch, which consists of synchronizing the bank-switch to the position of the raster beam, does fix the problem, but I have no notion why and would love to hear from someone who does.

The unpatched code works correctly on a number of other 64s that I have tried. Mine is a standard-issue machine except that it has a Cockroach TurboROM installed for fast disk operations. It seems unlikely that the Cockroach is the cause of the bug, which occurs whether or not the fast DOS routines are enabled, but perhaps it's not impossible. If anyone out there wants to test and report on the following code, particularly if you also have a Cockroach ROM installed, I'd be interested to hear what comes of it:

```

lda $dd02 ;set up CIA #2 DDR
ora #3
sta $dd02
loop lda $dd00 ;switch to bank 3
and #$fc
sta $dd00
lda $dd00 ;switch to bank 0
ora #3
sta $dd00
lda $028d ;exit if SHIFT down
beq loop
rts
    
```

If you don't have the bug, this routine will just make your screen go bananas till you press SHIFT. If you do have the bug, something else will happen.

Back to graphing: the print command GPRINT in the GRAPHCMDS module is set up, as I mentioned above, for the Epson MX-80, and won't work on anything not compatible with that machine. If you want to drive another printer with this module, however, there's no reason why another version of GPRINT couldn't be dropped into the module. Because another printer might need more code than Paul required for his MX-80 (the 6 by 7 print-head matrix of the 1525/MPS801, for example, is a bit tricky to write hi-res dumps for) I have left some unused program lines that you can spill over into if necessary (14614 to 14998).

Just so the news, when it finally breaks, won't come as a life-threatening shock, I would like to officially start the rumour now that next issue's TransBASIC column will be announced as the last of the series. You don't have to believe it if you don't want to, but it just might be true. See you then.

PLOT (Type: Statement Cat #: 175)

Line Range: 12694-12972

Module: GRAPHCMDS

Example: PLOT X,Y

This statement sets the specified pixel in the hi-res drawing area. The pixel is cleared if erase mode has been selected. X and Y can range from 0 to 319 and 0 to 399 respectively.

GCLR (Type: Statement Cat #:176)

Line Range: 12974-13016

Module: GRAPHCMDS

Example: GCLR

The hi-res drawing area is erased.

GCOL (Type: Statement Cat #:177)

Line Range: 13018-13036

Module: GRAPHCMDS

Example: GCOL 16*6 + 14

The hi-res foreground and background colours are set by this command. The foreground colour is set by the upper nybble of the parameter, and the background colour is set by the lower nybble.

UPPER (Type: Statement Cat #: 178)

Line Range: 13038-13094

Module: GRAPHCMDS

Example: UPPER

The upper portion of the hi-res drawing area is displayed (y-coordinates 200 through 399).

LOWER (Type: Statement Cat #: 179)

Line Range: 13042-13094

Module: GRAPHCMDS

Example: LOWER

The upper portion of the hi-res drawing area is displayed (y-coordinates 200 through 399).

DEFAULT (Type: Statement Cat #: 180)

Module: GRAPHCMDS

Example: DEFAULT

The screen is restored to the default text mode, with the ROM's uppercase/lowercase character set, and screen memory at \$0400.

LINE (Type: Statement Cat #: 181)

Line Range: 13114-13548

Module: GRAPHCMDS

Example: LINE X1,Y1,X2,Y2

A line is drawn between the specified coordinates using the pattern set in the last PATTERN command. The default pattern is \$FF (a solid line).

SLINE (Type: Statement Cat #: 182)

Line Range: 13096-13548

Module: GRAPHCMDS

Example: SLINE XL,YL,XH,YH

A line is drawn between the specified coordinates using a single-dotted pattern (1 dot on, 1 dot off). This has the effect of setting the first PATTERN parameter to \$55.

DLINE (Type: Statement Cat #: 183)

Line Range: 13100-13548

Module: GRAPHCMDS

Example: DLINE X1,Y1,X2,Y2

A line is drawn between the specified coordinates using a double-dotted pattern (2 dots on, 2 dots off). This has the effect of setting the first PATTERN parameter to \$33.

NLINE (Type: Statement Cat #: 184)

Line Range: 13104-13548

Module: GRAPHCMDS

Example: NLINE XL,YL,XR,YR

A solid line is drawn between the specified coordinates. This has the effect of setting the first PATTERN parameter to \$FF.

QLINE (Type: Statement Cat #: 185)

Line Range: 13108-13548

Module: GRAPHCMDS

Example: QLINE X1,Y1,X2,Y2

A line is drawn between the specified coordinates using a quadruple-dotted pattern (4 dots on, 4 dots off). This has the effect of setting the first PATTERN parameter to \$0F.

CHAR (Type: Statement Cat #: 186)

Line Range: 13550-13956

Module: GRAPHCMDS

Example: CHAR X,Y,CHR\$(5) + "GUMBOOT"

The string is printed at the specified location in the hi-res drawing area. Most control characters are ignored. However, the cursor control characters have their normal effects, and the following have special meanings, as follows:

- CHR\$(18) (CTRL-9) Reverse on
- CHR\$(146) (CTRL-0) Reverse off
- CHR\$(31) (CTRL-7) Uppercase/lowercase
- CHR\$(30) (CTRL-6) Uppercase/graphics
- CHR\$(144) (CTRL-1) Erase mode on
- CHR\$(5) (CTRL-2) Erase mode off

Note that the last two of these are equivalent to setting the erase mode with the ERASE command in this module.

SCHAR (Type: Statement Cat #: 187)

Line Range: 13958-14094

Module: GRAPHCMDS

Example: SCHAR 141,277,5

A special character (specified as the third parameter) is drawn at the specified coordinates. The special characters are either 3 by 3 or 5 by 5 pixels in size. Those available are:

- #0 : a 5 by 5 diagonal cross (x shape)
- #1 : a 5 by 5 orthogonal cross (+ shape)
- #2 : a 5 by 5 hollow square
- #3 : a 5 by 5 hollow diamond
- #4 : a 3 by 3 diagonal cross
- #5 : a 3 by 3 orthogonal cross
- #6 : a 3 by 3 hollow square
- #7 : a 3 by 3 hollow diamond
- #8 : a 5 by 5 solid square
- #9 : a 5 by 5 solid diamond
- #10: a 3 by 3 solid square
- #11: a 3 by 3 solid diamond

BAR (Type: Statement Cat #: 188)

Line Range: 14122-14220

Module: GRAPHCMDS

Example: BAR X1,Y1,X2,Y2

Plots (or clears, in erase mode) a bar in the area bounded horizontally by X1 and X2, and vertically by Y1 and Y2. The bar is drawn with the pattern last set by the PATTERN command. The default pattern is solid (both parameters \$FF).

SBAR (Type: Statement Cat #: 189)

Line Range: 14096-14220

Module: GRAPHCMDS

Example: SBAR X1,Y1,X2,Y2

A bar is drawn in the specified region using a single-dotted pattern (one dot one, one dot off). This command sets the pattern parameters to \$55 and \$AA.

DBAR (Type: Statement Cat #: 190)

Line Range: 14104-14220

Module: GRAPHCMDS

Example: DBAR X1,Y1,X2,Y2

A bar is drawn in the specified region using a double-dotted pattern (two dots one, two dots off). This command sets the pattern parameters to \$33 and \$99.

NBAR (Type: Statement Cat #: 191)

Line Range: 14112-14220

Module: GRAPHCMDS

Example: NBAR X1,Y1,X2,Y2

A solid bar is drawn in the specified region. This command sets the pattern parameters to \$FF and \$FF.

GSAVE (Type: Statement Cat #: 192)

Line Range: 14222-14384

Module: GRAPHCMDS

Example: GSAVE "BIG-GRAPHS"

The drawing area (or the portion of it specified in the SIZE command) is saved as a sequential file to drive 0, unit number 8. The characters ".g" are appended to the first fourteen characters of the specified filename. The two hi-res screens that make up the drawing area are stored contiguously in the file, which can hence be easily re-examined only with the GLOAD command. The GSAVE routine makes use of logical file #2 and secondary address #2 -- make sure these are available before attempting a save.

GLOAD (Type: Statement Cat #: 193)

Line Range: 14230-14440

Module: GRAPHCMDS

Example: GLOAD "HUGE-GRAPHS"

This restores a saved graph to the drawing area from drive 0, unit number 8. If only a portion of the screen was saved to this file, the GCLR command should be given before reloading. As with GSAVE, logical file #2 and secondary address #2 must not be currently in use when this command is given.

SIZE (Type: Statement Cat #: 194)

Line Range: 15008-15068

Module: GRAPHCMDS

Example: SIZE 240

An upper boundary is set in the hi-res drawing area, for use by the GSAVE and GPRINT commands. The boundary is set as a y-coordinate (up to 399, which is also the default). The specified number is then rounded up to the nearest 4-character (32 scan-line) boundary for actual use. The purpose of this command is to save disk space and printing time when part of the drawing-area is unused.

LMAR (Type: Statement Cat #: 195)

Line Range: 15000-15006

Module: GRAPHCMDS

Example: LMAR N

The printer left margin is set to N columns. The default is 11.

GPRINT (Type: Statement Cat #: 196)

Line Range: 14442-14612
 Module: GRAPHCMDS
 Example: GPRINT
 The current drawing area (to the limit set by the SIZE command) is printed to an Epson MX-80 print. Logical file #3 is used by this command, and must be available when the command is issued.

ERASE (Type: Statement Cat #: 197)

Line Range: 15070-15082
 Module: GRAPHCMDS
 Example: ERASE 0
 If this command is given with a non-zero parameter, erase mode is turned on. This results in all drawing commands clearing the affected pixels instead of setting them. Erase mode is turned off by using a zero parameter, as in the example.

PATTERN (Type: Statement Cat #: 198)

Line Range: 15084-15154
 Module: GRAPHCMDS
 Example: PATTERN N1,N2
 The plot patterns are set for the BAR and LINE commands. LINE uses only the first of the two patterns; BAR uses both. For BAR, the N1 parameter is applied vertically, and the N2 parameter horizontally. The patterns effectively create a mask over the region being drawn to: bits that are set to 1 in the mask will be affected (set or cleared, according to the erase mode), while 0 bits are not affected. The default patterns are \$FF and \$FF, selecting all bits.

Editor's Note

This incredibly huge TransBASIC module is listed here only for reference. Although the Verifier codes are shown, I'll be shocked to find out someone actually entered it by hand. My apologies for the type size - it's the only way we could justify including it.

Program 1: GRAPHCMDS

```

CI 0 rem graphcmds (paul adams)
FH 1:
MP 2 rem 24 statements, 0 functions
HH 3:
MN 4 rem keyword characters: 113
JH 5:
FF 6 rem keywords #175 to #198
LH 7:
BO 8 rem =====
NH 9:
AF 39 setfss = $ffba
KD 40 setnam = $ffbd
IG 41 open = $ffc0
CL 42 chkin = $ffc6
IH 43 close = $ffc3
IB 44 clrchn = $ffc5
DB 45 getin = $ffe4
MD 46 chkout = $ffc9
GF 47 chrout = $ffd2
GK 48:
OK 49 tmp = $57 ;more zp storage
AB 50 tmp2 = $59
KB 51 tmp3 = $5a
OB 52 tmp4 = $5c
NN 400 .asc "ploTgclRgcoLuppeR"
IC 401 .asc "loweRdefaulTline"
JM 402 .asc "slineEdlinEnline"
IJ 403 .asc "qlinEchaRschaR"
OM 404 .asc "baRsbaRdbaRnbaR"
GH 405 .asc "gsavEgloaDsizE"
JI 406 .asc "lmaRgprinTerasE"
OD 407 .asc "patterN"
KC 1400 .word plo-1, gcl-1, gco-1, uppe-1
GM 1401 .word lowe-1, defft-1, plin-1
BN 1402 .word slin-1, dlin-1, nlin-1
AN 1403 .word qlin-1, cha-1, scha-1
FG 1404 .word bba-1, sba-1, dba-1, nba-1
LM 1405 .word gsav-1, gloa-1, siz-1
JG 1406 .word lma-1, gprin-1, eras-1
FN 1408 .word patt-1
OI 2620 usfp ldx #0
AJ 2622 stx $0d
MA 2624 sta $62
BH 2626 sty $63
ON 2628 ldx #$90
HM 2630 sec
NH 2632 jmp $bc49
AM 2634:
FJ 3694 powers.byte 1,2,4,8,16,32,64,128
GO 3696:
GD 12694 plo jsr get2 ;read x,y params
NA 12696 plo1 sec :entry
IE 12698 lda #$bf ;redef y, ty=399-y
NG 12700 sbc y
KE 12702 sta ty
NI 12704 lda #1
KF 12706 sbc y+1
CB 12708 sta ty+1
OK 12710 bcs plo2 ;y>399 - too big
EJ 12712 rts
MI 12714 plo2 ldx #$c0 ;set base to upper
GB 12716 ldy #$c0 ;or lower hi-res
MB 12718 lda y+1 ;base=$a000 (upper)
PE 12720 bne plo3 ;base=$c0c0 (lower)
ID 12722 lda #$c7 ;lower y=0-199
JK 12724 cmp y ;upper y=200-399
EP 12726 bcs plo4
CP 12728 plo3 ldx #0
AH 12730 ldy #$a0
AJ 12732 plo4 stx tmp
LH 12734 sty tmp+1
IF 12736 ldx x ;test x<320
JG 12738 lda x+1
MA 12740 beq plo5 ; yes
IP 12742 cmp #1
LK 12744 bne plo2 ; no
HG 12746 cpx #$40
GM 12748 bcs plo2 ; no
GA 12750 plo5 ror ;divide x by 8
EI 12752 txa
NP 12754 ror ;3 left with hi bit
NK 12756 lsr
PK 12758 lsr
HF 12760 sta col ;column number
NO 12762 txa ;find bit # in col
PH 12764 and #7
NO 12766 eor #7 ;calc 7-(bit #)
OL 12768 tax ;conv to bit mask
NM 12770 lda powers,x ; in bitp
KJ 12772 sta bitp
AG 12774 ldx ty ;find row and line
IB 12776 lda ty+1
OJ 12778 ror
HA 12780 txa ; divide by 8
CK 12782 ror
JM 12784 lsr
LM 12786 lsr
DM 12788 sta row ;row number
    
```

```

JL 12790 lda ;find line number
LJ 12792 and #7
NL 12794 sta rast
BP 12796 lda row ;add row*216 to
AM 12798 ldx #6 ; base
PK 12800 jsr cmltad
AI 12802 ldx #2 ;add row*21(6+2)
KE 12804 jsr mladd ;for total row*320
GB 12806 lda col ;add col*(213)
FF 12808 ldx #3
JL 12810 jsr cmltad
NP 12812 ldy #0 ;.y is index later
KL 12814 sty tmp2
FJ 12816 lda rast
MH 12818 jsr add
ED 12820 sei ;disable irq
DH 12822 lda 1 ;switch out roms
LD 12824 and #$fc
BP 12826 sta 1
PK 12828 lda (tmp),y ;get byte to change
JL 12830 eor erflg ;invert if erasing
GK 12832 ora bitp ;add point
NL 12834 eor erflg ;invert if erasing
EN 12836 sta (tmp),y ;write to screen
OG 12838 lda 1 ;switch in roms
KA 12840 ora #3
BA 12842 sta 1
OC 12844 cli ;enable irq
EK 12846 ldy $dd00 ;get current screen
HK 12848 lda $028d ;test logo key
AI 12850 tax ;(save shift reg)
DM 12852 and #2
NH 12854 beq plo7 ; not pressed
AN 12856 jsr defft ;text screen on
AJ 12858 plo6 lda $028d ;wait for release
HA 12860 tax
NM 12862 and #2
GL 12864 bne plo6 ; not released
HE 12866 tya ;get old screen
HN 12868 and #3
FK 12870 cmp #3 ;test text
BJ 12872 beq plo11 ; yes
DA 12874 eor #1 ;switch screens
LD 12876 bcc plo9 ;branch always
KC 12878 plo7 txa ;test ctrl key
HO 12880 and #4
PE 12882 beq plo11 ; not pressed
MK 12884 plo8 lda $028d ;wait for release
NO 12886 and #4
PI 12888 bne plo8
HN 12890 tya ;switch screens
EG 12892 and #3 ;test which scrn on
EF 12894 plo9 bne plo10 ; not lower
KN 12896 jmp uppe ;turn on upper scrn
JN 12898 plo10 cmp #1
JH 12900 bne plo11 ; not upper
CN 12902 jmp lowe ;turn on lower scrn
EF 12904 plo11 rts
AO 12906:
GC 12908 cmltad pha ;entry to pre-clear
NN 12910 lda #0 ; tmp2
ML 12912 sta tmp2
KP 12914 pha
KO 12916:
FB 12918 mladd asl ;mult by 21(xreg)
OA 12920 rol tmp2 ;high byte tmp2
BA 12922 dex ;low byte .a
PK 12924 bne mladd
EP 12926:
GO 12928 add pha ;save low byte
EA 12930 clc ;add to tmp
ML 12932 adc tmp
KA 12934 sta tmp
GJ 12936 lda tmp2
LJ 12938 adc tmp+1
JO 12940 sta tmp+1
EA 12942 pla ;restore low byte
MH 12944 rts
IA 12946:
IF 12948 get2 jsr $ad8a ;get numeric arg
HC 12950 jsr $aefd ;check comma
AM 12952 jsr comx ;check range
GC 12954 jsr $b7f7 ;conv to integer
EP 12956 sty x ;save 1st word
DI 12958 sta x+1
BF 12960 jsr $ad8a ;get 2nd arg
OM 12962 jsr comy ;check range
AD 12964 jsr $b7f7 ;conv to integer
ON 12966 sty y ;save 2nd word
OI 12968 sta y+1
GJ 12970 rts
CC 12972:
EO 12974 gcl lda #$bf ;clear 32 pages
FF 12976 sta t3 ; of memory at
HJ 12978 lda #$ff ;$a000 and $e000
JA 12980 sta t5 ; (upper and lower
HO 12982 ldx #$20 ; hi-res screens)
HF 12984 ldy #$40 ;incompl. top page
FK 12986 lda #0
OL 12988 gcl1 pha ;entry from gcol
    
```




```

IB 13584 bmi cha4 ;unknown ctrl char
CB 13586 cha3 tya ;get routine addr
PL 13588 asl
BO 13590 tax
HG 13592 lda ccrtns,x
IH 13594 sta t2
FC 13596 lda ccrtns + 1,x
OH 13598 sta t3
PF 13600 jsr cha5
BA 13602 cha4 jmp cha20
AJ 13604 cha5 jmp (t2)
MJ 13606 ;
BH 13608 ctrls .byte $12,$92,$1f,$9e,$90
JD 13610 .byte $05,$11,$1d,$91,$9d
CK 13612 ;
IH 13614 ccrtns .word ccr,ccrx,ccl,ccu,cce
HI 13616 .word ccex,ccdn,cort,ccup,cclt
IK 13618 ;
LC 13620 ccr lda #4 ;reverse on
BB 13622 .byte $2c
KH 13624 ccl lda #8 ;upper/lower case
JD 13626 ora choff
KE 13628 bne ccu1
EL 13630 ;
PH 13632 ccrx lda #$db ;reverse off
NB 13634 .byte $2c
JK 13636 ccu lda #$d7 ;uppercase/graphics
KD 13638 and choff
AO 13640 ccu1 sta choff
GD 13642 rts
CM 13644 ;
BK 13646 cce lda #$ff ;erase on
LC 13648 .byte $2c
LE 13650 ccex lda #0 ;erase off
MC 13652 sta erflg
CE 13654 rts
OM 13656 ;
CE 13658 ccnd lda #$f8 ;cursor down
HD 13660 .byte $2c
AL 13662 ccup lda #8 ;cursor up
LH 13664 jmp yoff ;add offset
IN 13666 ;
GO 13668 ccr lda #8 ;cursor right
BE 13670 .byte $2c
BG 13672 cclt lda #$f8 ;cursor left
EI 13674 jmp xoff ;add offset
CO 13676 ;
GG 13678 cha6 and #$e0 ;retain bits 5-7
GJ 13680 cmp #$60 ;test $60-$7f
KC 13682 bne cha7 ;no
IC 13684 txa
HP 13686 sec
BL 13688 sbc #$20
FE 13690 tax
FF 13692 bne cha9 ;branch always
FO 13694 cha7 cmp #$80 ;subtr $40 for
FP 13696 and #$40 ;either b6 or b7
BF 13698 bcc cha8 ;set
KP 13700 adc #$3f
OH 13702 cha8 sta t2
JA 13704 sec
OD 13706 txa
ML 13708 sbc t2
JF 13710 tax
GB 13712 cha9 lda #0
MB 13714 sta tmp ;clear for rotate
EO 13716 txa ;hi byte offset
KB 13718 ldx #3 ;times 8
GO 13720 cha10 asl
OE 13722 rol tmp ;save carry
DF 13724 dex
KK 13726 bne cha10
QP 13728 sta tmp4
OB 13730 lda choff ;scr code offset
KB 13732 clc
NJ 13734 adc tmp ;add high offset
HK 13736 sta tmp4 + 1 ;hi byte offset
JB 13738 lda #$f9 ;set offset
CO 13740 jsr yoff ;y=y-7
EI 13742 ldy #7 ;char byte counter
FB 13744 cha11 sei ;block interrupts
PC 13746 lda 1 ;i/o ram out
FN 13748 and #$fb
NI 13750 sta 1
FI 13752 lda (tmp4),y
OL 13754 tax ;save char byte
JC 13756 lda 1 ;i/o ram in
BK 13758 ora #4
HJ 13760 sta 1
EK 13762 cli ;enable interrupts
OD 13764 tya ;save byte counter
CE 13766 pha
FL 13768 txa ;save char byte
GE 13770 pha
PO 13772 ldx #7 ;char bit counter
MK 13774 cha12 pla ;rstr char byte
OJ 13776 asl ;bit to carry
HI 13778 pha ;save char byte
CL 13780 bcc cha17 ;skip if bit = 0
  
```

```

ON 13782 lda x + 1 ;test x < 0
NN 13784 bmi cha14 ;yes
OP 13786 cha13 lda #$fe ;subtract $0140
NC 13788 sta tmp ;until x is neg
KD 13790 lda #$c0
OE 13792 jsr xoff2 ;x=x-140
JC 13794 bpl cha13 ;repeat while pos
HC 13796 cha14 lda #1 ;add $140
DE 13798 sta tmp ;until x is pos
HB 13800 lda #$40
GF 13802 jsr xoff2 ;x=x+140
ED 13804 bmi cha14 ;repeat while neg
IP 13806 lda y + 1 ;test y < 0
HP 13808 bmi cha16 ;yes
EC 13810 cha15 lda #$fe ;subtract $0190
HE 13812 sta tmp ;until y is neg
OC 13814 lda #$70
PH 13816 jsr yoff2 ;y=y-190
IF 13818 bpl cha15 ;repeat while neg
HB 13820 cha16 lda #1 ;add $0190
NF 13822 sta tmp ;until y is pos
OD 13824 lda #$90
HI 13826 jsr yoff2 ;y=y+190
OE 13828 bmi cha16 ;repeat while neg
FB 13830 txa ;restore bit ctr
JN 13832 pha ;and save
AP 13834 jsr plo1 ;plot single point
PO 13836 pla ;restore bit ctr
JN 13838 tax
JE 13840 cha17 dex
NG 13842 bmi cha18 ;finished byte
LP 13844 lda #1 ;set offset
ID 13846 jsr xoff ;x=x+1
AJ 13848 jmp cha12 ;get next bit
MA 13850 cha18 pla ;waste char (done)
HN 13852 pla ;byte counter
NO 13854 tay
LN 13856 dey
EE 13858 bmi cha19 ;finished char
DJ 13860 lda #$f9 ;set offset
FF 13862 jsr xoff ;x=x-7
PA 13864 lda #1 ;set offset
DF 13866 jsr yoff ;y=y+1
HN 13868 jmp cha11 ;get next byte
FH 13870 cha19 lda #1 ;set offset
CF 13872 jsr xoff ;x=x+1
IG 13874 cha20 inc $22 ;inc string ptr
FE 13876 bne cha21
PN 13878 inc $23
KA 13880 cha21 dec len ;decrement length
BG 13882 beq cha22
AC 13884 jmp cha1 ;get next char
PM 13886 cha22 rts
GL 13888 ;
II 13890 xoff pha ;push offset
OP 13892 cmp #$80 ;test negative
DH 13894 bcs xof1 ;yes
LE 13896 lda #0 ;high byte 0
FC 13898 .byte $2c
PE 13900 xof1 lda #$ff ;high byte $ff
CN 13902 sta tmp
IN 13904 pla
IM 13906 ;
FM 13908 xoff2 clc ;add offset to x
IA 13910 adc x
GF 13912 sta x
AK 13914 lda tmp
FP 13916 adc x + 1
DE 13918 sta x + 1
ME 13920 rts
IN 13922 ;
MK 13924 yoff pha ;push offset
AC 13926 cmp #$80 ;test negative
GJ 13928 bcs yof1 ;yes
NG 13930 lda #0 ;high byte 0
HE 13932 .byte $2c
DH 13934 yof1 lda #$ff ;high byte $ff
EP 13936 sta tmp
KP 13938 pla
KO 13940 ;
KO 13942 yoff2 clc ;add offset to y
LC 13944 adc y
JH 13946 sta y
CM 13948 lda tmp
IB 13950 adc y + 1
GG 13952 sta y + 1
OG 13954 rts
KP 13956 ;
AC 13958 schar jsr get2 ;get x, y
JB 13960 jsr $aefd ;check comma
JC 13962 lda #$fe ;set offset
BL 13964 jsr xoff ;x=x-2
NC 13966 lda #$fe ;set offset
ML 13968 jsr yoff ;y=y-2
IP 13970 jsr $b79e ;char # to .x
IE 13972 txa
MP 13974 sta tmp3 ;save - mult x 5
EM 13976 asl ;x 2
GM 13978 asl ;x 2
  
```

```

CM 13980 adc tmp3 ;+ 1
KH 13982 sta tmp3 ;save address
GK 13984 ldy #5 ;byte counter
AP 13986 sch1 lda tmp3 ;restore address
MI 13988 tax ;addr to .x
NJ 13990 inc tmp3 ;incr addr
CG 13992 lda scx,x ;get data byte
NP 13994 ldx #5 ;bit counter
LD 13996 sch2 asl ;bit to carry
FJ 13998 pha ;save byte
IH 14000 bcc sch5 ;no plot
OI 14002 lda x + 1 ;test x in bounds
AN 14004 beq sch3 ;yes
GH 14006 lda x
MC 14008 cmp #$40 ;no
MJ 14010 bcs sch5 ;no
MF 14012 sch3 lda y + 1 ;test y in bounds
ON 14014 beq sch4 ;yes
BI 14016 lda y
FE 14018 cmp #$90 ;no
GK 14020 bcs sch5 ;no
OJ 14022 sch4 txa ;save counters
EE 14024 pha
BI 14026 tya
IE 14028 pha
DC 14030 jsr plo1 ;plot x,y
JK 14032 pla ;restore counters
BK 14034 tay
MF 14036 pla
BK 14038 tax
JI 14040 sch5 lda #1
MP 14042 jsr xoff ;x=x+1
PL 14044 pla ;restore byte
AC 14046 dex ;decr bit counter
IF 14048 bne sch2 ;next bit
PM 14050 lda #1
NA 14052 jsr yoff ;y=y+1
DJ 14054 lda #$fb
DB 14056 jsr xoff ;x=x-5
IN 14058 dey ;byte counter
KC 14060 bne sch1 ;next byte
KN 14062 rts
GG 14064 ;
AI 14066 chars - 5 left bits of 5 bytes
NM 14068 scx = *
AG 14070 .byte $88,$50,$20,$50,$88 ;5x5 x
BB 14072 .byte $20,$20,$fc,$20,$20 ;5x5 +
GP 14074 .byte $fc,$88,$88,$88,$fc ;5x5sqr
FB 14076 .byte $20,$50,$88,$50,$20 ;5x5dmd
AB 14078 .byte $00,$50,$20,$50,$00 ;3x3 x
CL 14080 .byte $00,$20,$70,$20,$00 ;3x3 +
EP 14082 .byte $00,$70,$50,$70,$00 ;3x3sqr
JN 14084 .byte $00,$20,$50,$20,$00 ;3x3dmd
LM 14086 .byte $fc,$fc,$fc,$fc,$fc ;5x5sqr
EH 14088 .byte $20,$70,$fc,$70,$20 ;5x5dmd
EA 14090 .byte $00,$70,$70,$70,$00 ;3x3sqr
JO 14092 .byte $00,$20,$70,$20,$00 ;3x3dmd
EI 14094 ;
BG 14096 sba ldx #$55 ;single dot vert
NL 14098 lda #$aa ;single line horiz
LP 14100 bne nb1
MI 14102 ;
DB 14104 dba ldx #$33 ;double dot vert
NH 14106 lda #$99 ;double line horiz
DA 14108 bne nb1
EJ 14110 ;
IF 14112 nba lda #$ff ;no pattern
NO 14114 tax
EF 14116 nb1 stx dot
NN 14118 sta ldot
OJ 14120 ;
PC 14122 bba lda dot ;save y pattern
MH 14124 sta tmp3
JD 14126 lda ldot ;save x pattern
MK 14128 pha
EB 14130 jsr get2 ;get x1,y1
MH 14132 sty y1 ;store parameters
JF 14134 sta y1 + 1
IP 14136 lda x
KJ 14138 sta x1
DO 14140 lda x + 1
AG 14142 sta x1 + 1
BN 14144 jsr $aefd ;check comma
HC 14146 jsr get2 ;get x2,y2
EI 14148 sty y2 ;store y parameter
OD 14150 sta y2 + 1 ;x2 saved in 'x'
LK 14152 ba1 ldy y2 ;init y countdown
BD 14154 lda y2 + 1
LK 14156 sty y ;y
ED 14158 sta y + 1
HO 14160 lda tmp3 ;restore y pattern
GN 14162 sta dot
DH 14164 ba2 sec ;test outer (x)
JD 14166 lda x ;loop complete
KI 14168 sbc x1
BA 14170 lda x + 1
AF 14172 sbc x1 + 1 ;yes
CO 14174 bcc ba5 ;rotate x pattern
LA 14176 ror ldot
  
```




DK	14178	php	; save low bit	LB	14376	bne gd6		MN	14574	lda #Se0	; set to low scrn
GD	14180	rol ldot		OE	14378	gd8	jsr clrchn	NE	14576	sta tmp+1	
ON	14182	plp	; recover low bit	LB	14380	lda #2		ND	14578	ldy #0	
JO	14184	ror ldot	; complete rotation	JI	14382		jmp close	BB	14580	beq gpr3	; branch always
OG	14186	bcc ba4	; no draw if bit=0	GK	14384			JD	14582	gpr10	ldx #3 ; restore printer
MG	14188	ba3	sec	AD	14386	gd9	jsr chkin	GE	14584	gpr11	lda ptrf,x ; cr, line space
CF	14190	lda y	; loop complete	BF	14388		jsr getin	BN	14586	jsr chrout	
DK	14192	sbx y1		GM	14390		sta hite	DL	14588	dex	
KB	14194	lda y+1		OI	14392		tay	EH	14590	bpl gpr11	
JG	14196	sbx y1+1		EO	14394		lda \$90	BH	14592	gpr12	jsr clrchn ; close channel
HP	14198	bcc ba4	; yes	OO	14396		bne gd8	IP	14594	lda #3	; file #
HL	14200	jsr dplot	; draw if patt bit = 1	NG	14398		jsr getin	NH	14596	jmp close	; close file
FD	14202	lda #fff	; countdown y	MN	14400		sta hite+1	MH	14598		
CD	14204	jsr yoff		OL	14402		sta tmp+1	DB	14600		; prt conditioning data strings
JP	14206	bcs ba3	; cc means y = -1	KH	14404	gd10		JK	14602	ptrs	= *
DG	14208	ba4	lda #fff ; countdown x	AA	14406		sta (tmp),y	BH	14604		.byte \$00,\$0d,\$44,\$1b,\$08,\$41,\$1b
HD	14210	jsr xoff		CP	14408		lda \$90	MK	14606	ptrr	= *
FP	14212	bcs ba1	; cc means x = -1	MP	14410		bne gd8	KL	14608		.byte \$01,\$40,\$4b,\$1b,\$09,\$0d
JM	14214	ba5	pla ; restore x pattern	CP	14412		iny	MJ	14610	ptrf	.byte \$32,\$1b,\$0d
PD	14216	sta ldot		CP	14414		bne gd10	KI	14612		
GH	14218	rts		KJ	14416		inc tmp+1	OL	15000	lma	jsr \$b79e ; set left margin
CA	14220			BI	14418		beq gd8	IM	15002		stx ptrs+1 ; for printing
AB	14222	gsav	ldx #0 ; save to disk flag	DH	14420		lda tmp+1	MD	15004	lma1	rts
OB	14224		lda #w ; write	LF	14422		cmp #c0	EB	15006		
JD	14226		bne gd1 ; branch always	MP	14424		bne gd10	HA	15008	siz	jsr \$ad8a ; get graph height
KA	14228			PD	14426		lda #e0	OM	15010		jsr comy ; check range
HD	14230	gloa	ldx #1 ; load frm disk flag	JL	14428		sta tmp+1	AD	15012		jsr \$b7f7 ; conv to integer
PJ	14232		lda #r ; read	PN	14430		bne gd10	EK	15014		sty tmp3 ; save height
AB	14234			GN	14432			EA	15016		sta tmp3+1
DL	14236	gd1	sta gnsuf+5	IJ	14434	gfnam	= *	NA	15018		ldx #a0 ; max height
GE	14238		stx tmp2	ED	14436		.asc "0:0123456789abcd.g,s,w"	EK	15020		stx hite+1
FG	14240		jsr \$ad9e ; eval filename	IO	14438	gnsuf	.asc ".g,s,w"	MM	15022	sz1	sec
PI	14242		jsr \$b6a3 ; make descriptor	ON	14440			NO	15024		lda #f6 ; test height>367
HE	14244		cmp #f0f ; test len > 14	NL	14442	gprin	lda #0 ; no name	EN	15026		sbx tmp3
MD	14246		bcc gd2 ; no	NC	14444		jsr setnam	BK	15028		lda #1
KN	14248		lda #e0e ; max len	EG	14446		lda #3 ; file #	EO	15030		sbx tmp3+1
FH	14250	gd2	pha ; save len	NL	14448		ldx #4 ; printer	NK	15032		bmi lma1 ; finished
GI	14252		tax ; counter	DO	14450		ldy #fff ; no secondary addr	DL	15034		clc ; tmp3 = tmp3 + 32
JP	14254		ldy #0	OD	14452		jsr selfs	JD	15036		lda #20 ; (4 char heights)
AM	14256	gd3	lda (\$22),y ; get char	KO	14454		jsr open	CM	15038		adc tmp3
IG	14258		sta gfnam+2 ; copy to buffer	AF	14456		bcc gpr1 ; no error	AB	15040		sta tmp3
EJ	14260		iny	IA	14458		jmp gpr12	NK	15042		lda #0
NG	14262		dex	FM	14460	gpr1	ldx #3 ; file #	EN	15044		adc tmp3+1
CK	14264		bne gd3	FG	14462		jsr chkout	CC	15046		sta tmp3+1
EE	14266		ldx #0 ; add '.g,s,r/w'	EF	14464		ldx #6 ; count 7 chars	AB	15048		clc ; hite = hite + 5*256
AI	14268	gd4	lda gnsuf,x	LA	14466	gpr2	lda ptrs,x ; get char	DM	15050		lda #5 ; (4 char heights)
CO	14270		sta gfnam+2,y	FL	14468		jsr chrout ; to printer	MB	15052		adc hite+1
AK	14272		iny	ND	14470		dex	GC	15054		cmp #c0 ; test off \$a000 scr
OJ	14274		inx	BN	14472		bpl gpr2	OJ	15056		bcc sz2 ; no
JC	14276		cpw #6 ; set tmp to start	BF	14474		lda #0	AM	15058		cmp #e0 ; test on \$e000 scr
DL	14278		bne gd4	AB	14476		sta tmp	CN	15060		bcs sz2 ; yes
GB	14280		pla ; get length	MO	14478		lda hite+1	FO	15062		ora #20 ; advance pointer
AE	14282		clc	NO	14480		sta tmp+1	JL	15064	sz2	sta hite+1
FO	14284		adc #8 ; adjust	LM	14482		ldy hite ; lb addr counter	LM	15066		bne sz1 ; branch always
PF	14286		ldx #<gfnam	HC	14484	gpr3	ldx #5 ; set prt to nxt row	CF	15068		
BG	14288		ldy #>gfnam	DO	14486	gpr4	lda ptr,x ; send cr, ht, and	KA	15070	eras	jsr \$b79e ; condition
DJ	14290		jsr setnam	AI	14488		jsr chrout ; 320 dots	BA	15072		txa ; erase flag
AD	14292		lda #2 ; lf #	BF	14490		dex	JO	15074		beq era1
OD	14294		tay ; sec addr	NO	14492		bpl gpr4	NP	15076		ldx #fff
DF	14296		ldx #8 ; device #	GM	14494		lda #28 ; column counter	PM	15078	era1	stx erflg
EK	14298		jsr selfs	AP	14496		sta tmp3	EN	15080		rts
AF	14300		jsr open	OH	14498	gpr5	ldx #7 ; get 8 lines data	AG	15082		
FK	14302		bcc gd5	ND	14500		sei	CK	15084	patt	jsr \$b79e ; get line pattern
DI	14304		cmp #4 ; test filenotfound	BH	14502		lda #fc ; switch out roms	OM	15086		stx dot
BM	14306		bne gd8 ; no - real error	JJ	14504		and 1	BI	15088		jsr \$aefd ; check comma
BF	14308	gd5	ldy #0 ; clr save startaddr	BI	14506		sta 1	IN	15090		jsr \$b79e ; get bar pattern
KM	14310		sty tmp	EE	14508	gpr6	lda (tmp),y ; get byte	HA	15092		stx ldot
ID	14312		ldx #2 ; file #	GA	14510		sta buff,x ; to buffer	CO	15094		rts
LG	14314		lda tmp2 ; save/load flag	AJ	14512		iny	OG	15096		
IO	14316		bne gd9	NO	14514		bne gpr7	EP	15098	x1	.word \$0040
FN	14318		jsr chkout ; open output chan	DP	14516		inc tmp+1	IP	15100	x1	.word \$0040
AN	14320		lda hite ; graph height	KG	14518	gpr7	dex	HA	15102	y2	.word \$0080
IE	14322		tay ; lb addr counter	BB	14520		bpl gpr6	LA	15104	y2	.word \$0080
LM	14324		jsr chrout	BI	14522		lda #3 ; switch in roms	OB	15106	dx	==+5
EF	14326		lda hite+1	DK	14524		ora 1	DC	15108	dy	==+5
PM	14328		jsr chrout	FJ	14526		sta 1	AE	15110	sx	==+5
IG	14330	gd6	sta tmp+1 ; hb addr counter	OE	14528		cli	FE	15112	sy	==+5
KG	14332	gd7	sei ; disable irq	HL	14530		tya ; save addr lo	ED	15114	x	.word \$0080
LF	14334		lda 1 ; switch out roms	AE	14532		pha	ID	15116	y	.word \$0080
DC	14336		and #fbc	AN	14534		ldy #7 ; byte counter	FF	15118	tmpf	==+5
JN	14338		sta 1	NB	14536	gpr8	ldx #7 ; bit counter	JD	15120	xo	==+5
EK	14340		lda (tmp),y ; get byte to save	NB	14538	gpr9	rol buff,x ; bit to carry	ND	15122	yo	==+5
CI	14342		pha	KD	14540		rol	HE	15124	ty	==+2
AF	14344		lda 1 ; switch in roms	FI	14542		dex	PD	15126	bitp	==+1
MO	14346		ora #3	FD	14544		bpl gpr9	CP	15128	row	==+1
DO	14348		sta 1	DA	14546		jsr chrout ; to printer	LE	15130	rast	==+1
AB	14350		cli ; enable irq	PI	14548		dey	MK	15132	col	==+1
IJ	14352		pla	HD	14550		bpl gpr8	HG	15134	dot	.byte \$ff
PI	14354		jsr chrout ; output byte	OI	14552		pha ; restore addr lo	CE	15136	ldot	.byte \$ff
EC	14356		lda \$90 ; check status	JK	14554		tay	OP	15138	addr	==+2
JA	14358		bne gd8 ; exit condition	NF	14556		dec tmp3 ; test row done	IF	15140	erflg	.byte \$00
IP	14360		iny	FN	14558		bne gpr5 ; no	GB	15142	tx	==+2
AB	14362		bne gd7	OG	14560		cpy #40 ; test screen done	BJ	15144	ty2	==+2
LF	14364		inc tmp+1	BN	14562		bne gpr3 ; no	KO	15146	hite	.word \$a000
MC	14366		beq gd8	DA	14564		lda tmp+1	FE	15148	buff	==+8
PD	14368		lda tmp+1	CP	14566		cmp #fff ; low scr bottom	AD	15150	choff	.byte \$d0
HC	14370		cmp #c0 ; top of upper block	EK	14568		beq gpr10 ; restore prt	JD	15152	len	.byte \$10
KB	14372		bne gd7	ED	14570		cmp #fb ; upper scr bottom	IK	15154		
KF	14374		lda #e0 ; set lower block	HB	14572		bne gpr3				

EXPO 86

Vancouver, British Columbia

Commodore and IBM head-to-head. . .

by Ian Adam, P.Eng., Vancouver, BC.

Abstract

Any World's Fair promises a unique blend of fun and education, and Expo 86 being held in Vancouver until October of 1986 is no exception. The World's Fair theme of Transportation and Communication offers a number of items of interest to computerists. Computers are used for everything from running the transportation system, to coordinating displays and finding lost children.

Where there are computers there is the Transactor, and your reporter selected two items of particular interest. The World's Fair visitor information system is a useful package of applied IBM technology, each terminal combining a personal computer with touch-screen technology and an interactive videodisc to create a friendly and useful information kiosk. Not to be outdone, Commodore is well represented with the Amiga Studio Theatre, a sophisticated package of arts and computers.

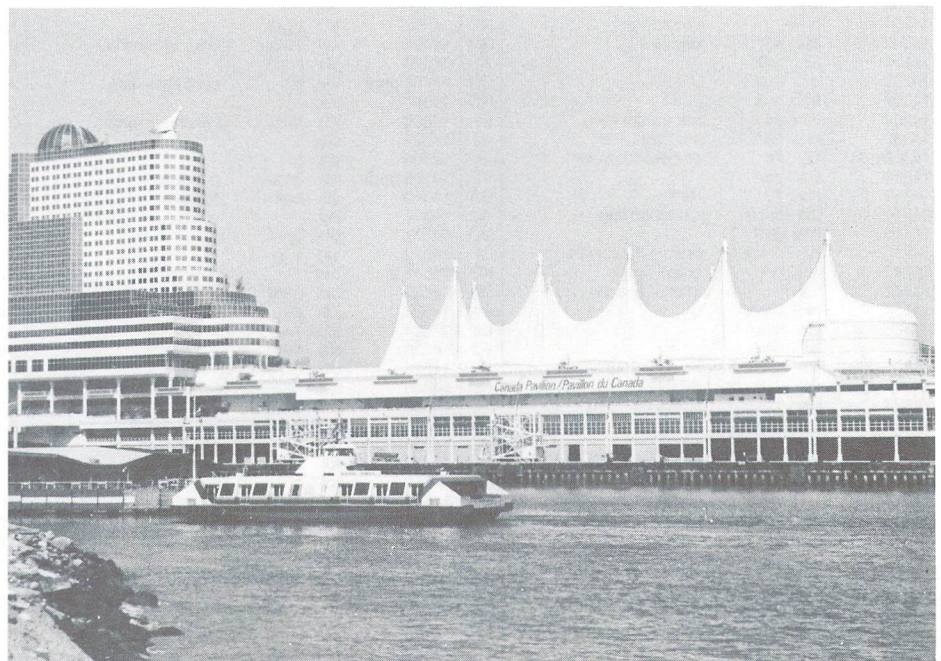
A Trip To EXPO

Now maybe I'm giving away a lot about my age, but I still harbour many fond memories of a trip to Montreal in 1967 to see Man and his World, the famed Expo '67. Simply the mention of a World's Fair still evokes such a wide range of images and expectations for me. On the one hand, like any fair, it suggests an occasion for people to gather for a good time: the proverbial fun and games, but in an unusual, exotic context. At the same time, however, a World's Fair promises so much more. By virtue of its international nature, it represents an opportunity for each country's best ideas and products to be brought to one place, and put on display for all the world to see.

Montreal's fair lived up to all of those expectations, and did so admirably. It combined fun and world-scale information into an overall package that 'clicked'. So it was with some mixed



Expo Centre, one of the theme pavilions at Expo 86. Its displays include the Futures Theatre, in which the audience can vote on issues affecting the future using illuminated buttons in the arms of the seats. Background: the Ontario pavilion includes a miniature replica of Niagara Falls.



The Canada Pavilion, with its five-sailed fabric roof, located on a pier jutting out into Vancouver Harbour. The rounded portion to the right contains the Amiga Studio Theatre. Foreground: the Seabus, a harbour ferry.

feelings that I first heard of the plans for a World's Fair in Vancouver. Yes, the mention of a Fair brought back all those great memories, and what could be more convenient than right here in Vancouver? At the same time, however, Montreal had set such a high standard away back in 1967, that I feared Expo 86 might somehow tarnish the concept by not quite living up to that image. After all, there have been some recent disappointments in the Exposition arena.

However, these fears turned out to be groundless. Expo 86 has been able to stand on its own two feet, and is extending the Canadian reputation established in Montreal for putting on a good show. Since this will probably be the last fair of the century in North America, I thought a bit about the aspects that are contributing to its success.

First, physically, it has a beautiful waterfront site in downtown Vancouver, very much reminiscent of Montreal's island fair. With ocean and harbour alongside, set against a backdrop of city and mountains, the site alone is enough to draw the attention of one's psyche away from the humdrum of day-to-day life, and open up a broader vision of the world. And that, remember, is one of the primary roles of a World's Fair.

Second, the fair has attracted considerable international attention. Some 52 countries are represented at Expo 86, along with a dozen provinces and states, better than 40 corporate participants, and a number of special theme pavilions. This alone is sufficient to ensure a wide variety of exhibits and a good level of interaction between nations.

Third, the World's Fair theme of Transportation and Communications is particularly topical, and offers ample latitude for exploration of high technology.

Vancouver is a city that was founded on transportation, being the western terminus of the Canadian railways, and the second busiest port in North America (eclipsed only by New York). 1986 is the 100th anniversary of the City, as well as of the transcontinental railway. 1986 is also a time when communication is rapidly taking over many of the functions of transportation, promising to provide an alternative to many types of personal travel. Computers will be a big part of the communications revolution, just as they have played a major role in transportation. As a result, the fair's theme offers a great deal of material of interest to all computerists.

A World's Fair should naturally demonstrate the leading edge of technology, and every memorable fair has had its own particular specialization, an area of technology where notable advances were brought forward and identified. At Montreal, much of the exploratory work was devoted to film, and the results of that effort are still with us today: giant screens, split-screen images, and all-enveloping film experiences have become familiar elements of our society, all traced back to the Montreal fair. The 1985 exhibition in Tsukuba, Japan, was devoted specifically to the theme of technology and its impacts. Robots that drew portraits, spun tops, and played sheet music were the most memorable exhibits.

In Vancouver, a major legacy will be in computers, and Expo 86 may well be known as the fair where computers came into their own, both running the fair and relating to the visitors. In the theme area of transportation, the rapid transit line to carry you to the site is fully computerized, requiring no driver. Even the City's traffic signals are coordinated by a new computer system. In the field of communications, there is the Futures Theatre, in which illuminated buttons built into the arms of the chairs allow the audience to vote on issues during

a multi-media presentation. These votes are then tallied by computer, and reported back to the audience.

Contributions to a fair can take many forms, ranging from formal national pavilions to the 'official supplier' of everything from soup to nuts and bolts (the latter having been developed ad infinitum at the Los Angeles Olympics). At Expo 86, two computer companies have made substantial contributions, IBM Canada and Commodore. While both demonstrate new installations of their equipment, the contrast between the two applications is fascinating.

Information System:

IBM's contribution takes the form of applied big-blue technology. Although I didn't go to the fair looking for IBM material to document, I felt the system was so fascinating that you might like a peek at it. The company has developed and is supplying a computerized visitor information system that will assist fair-goers in finding their way about the site. True to the tradition of World's Fairs, this system offers a demonstration of new technology, while at the same time it is a source of help and entertainment for the user.



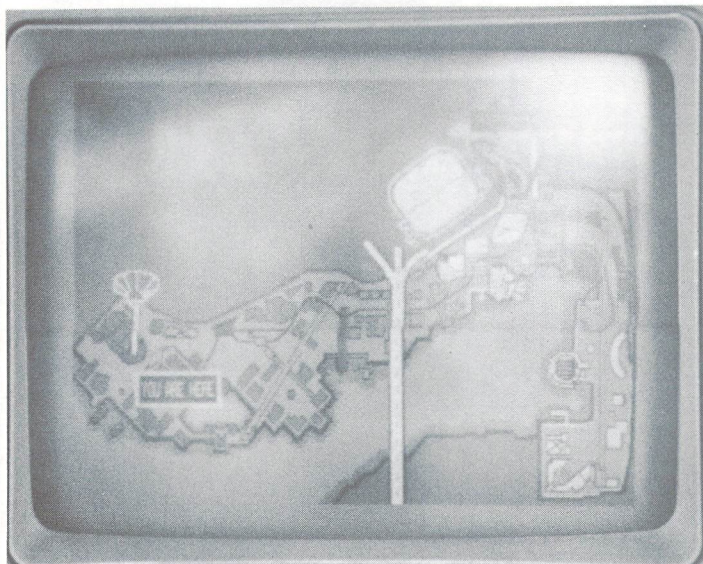
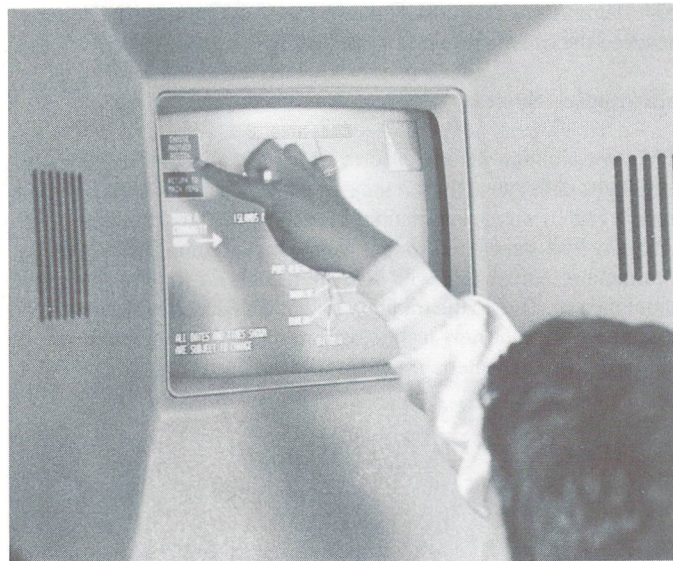
A typical IBM Information System kiosk containing 7 touch-screen terminals all coordinated by a master PC/AT.

At first glance, the system could pass for a video game, with the familiar colour monitor mounted in a cabinet. Faced with a series of menus, the user simply touches the screen in order to select the information desired. After a short pause, videodisc footage of the selected item is shown on the screen. This combination – a sort of 'Dirk the Daring goes to the fair' – is particularly well-suited to the application, and offers trouble-free access to the computer, even for the casual, non-computer-oriented user. Behind this seemingly simple facade, however, lurks an impressive collection of the state of the art in interactive computing technology.

The touch screen is an essential element of the overall concept, providing trouble-free input from the user. A standard keyboard was considered for input, but raises the obvious concerns of intimidating users not familiar with computers, as well as the potential for damage

from spills, chewing gum, and so on. A gummed-up keyboard would mean a non-functional information system.

Similarly, the laser videodisc is critical to the overall plan, providing television-quality moving images, and contributing to the overall system image. The videodisc is a standard consumer item, requiring only that the data be addressable by track. There is no program code on the disc.



Intermingled with the video footage are screens of computer graphics, used primarily for menus for the touch screen and for some elementary animation (for example, to show amusement rides). Even a 20-Meg hard disk has its limitations of speed and capacity (at up to 112K for a digitized photo), so this animation will never be mistaken for videodisc imagery. Nonetheless, it is an additional feature that introduces some variety. The system also has the capability to overlay computer graphics onto the output of the videodisc, for example to superimpose current messages about pavilion conditions or special events.

Quite naturally, IBM has used the PC/AT as the heart of each display. IBM's representatives indicated that the system could as easily run on an XT or PC, but the AT's improved speed and other capabilities provide for a superior user interface. (One can't help but wonder what sort of system could have been created if the heart were Amiga gold instead of blue!) The consoles are arranged in clusters of seven

screens. Each cluster, or kiosk, is linked internally by PCNetwork to its master PC/AT, as well as to the 4381 mainframe on a batch basis.

Program Material

Aside from the technology, putting together the program discs has its own challenges. One problem is just assembling the program material, when it has to come from 50 different countries and many other sources. Many countries are unable or unwilling to release information about their pavilions, or simply will not conform to the required format and schedule.

Another challenge is language. Bilingualism is a fact of life in Canada, with both English and French entrenched as official national languages. The stereo capability of the videodisc matches this perfectly, with one channel used for each official language. However, the system has not taken up the challenge of the 20 or more languages that could be appreciated by World's Fair visitors!

The Potential

The World's Fair information system is strictly a prototype at this point, assembled specifically for this purpose. However, one immediately conjures visions of its potential. With little modification other than a new videodisc, each console could be adapted to such tasks as education, catalog shopping, or flight information at an airport. The handicapped, if they have the mobility to reach the screen, could also benefit. As evidenced by the usual lineups, the IBM information system is a definite hit at Expo 86. This suggests strongly that it has good potential for public acceptance in these other applications.

Commodore's Amiga Theatre

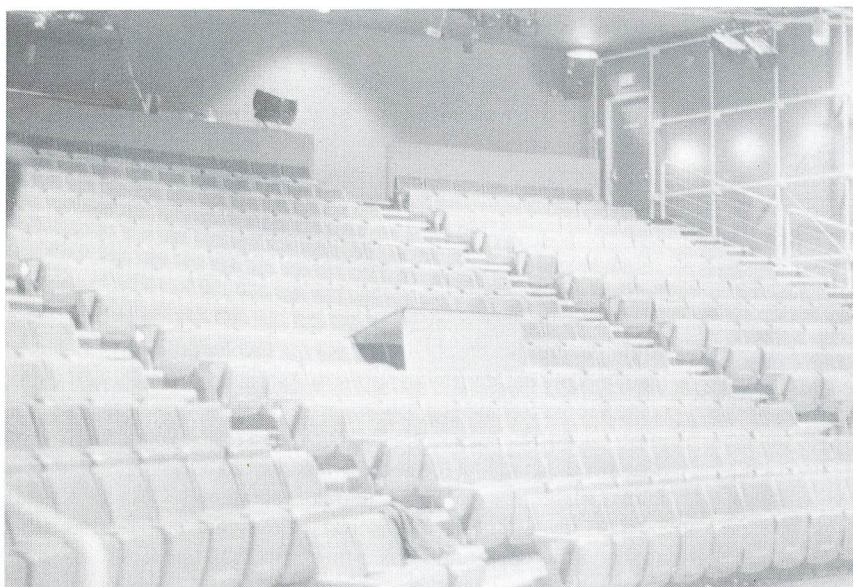
In contrast to IBM's straight display of high-tech, Commodore has chosen an artistic forum to highlight the capabilities of the Amiga computer. Commodore is an official corporate supporter at the Canada Pavilion, and has assumed sponsorship of the Amiga Studio Theatre, a unique facility that complements live stage performances with computerized special effects. This sponsorship consists of two elements: as well as the usual financial support, Commodore has supplied a number of Amiga computers for artistic use and theatre operations.

The Canada Pavilion is a short train ride away from the rest of the site, on a pier jutting out into Burrard Inlet, part of the Port of Vancouver. Its five-sailed fabric roof is one of the trademarks of Expo '86, also covering a cruise ship terminal. (Just so you hoser don't get confused about where you are, the portal to the host pavilion features the world's largest hockey stick and puck!)

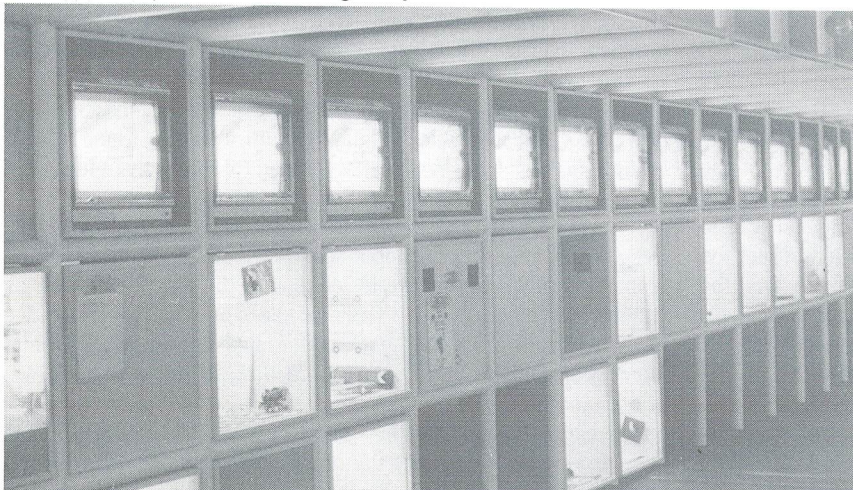
The Amiga Studio Theatre is a 380-seat facility located near the harbour end of the pavilion: in the accompanying photo, it is located in the round structure near the right side of the pier. Doug Welch, the proud Theatre Manager, filled me in on some of the details. Billed as Canada's most technologically sophisticated theatre, this studio has an intimate atmosphere with a distinct techno-mystical edge. While the live performances themselves are clearly artistic in nature, the stage is augmented by 6 projection screens, each 7 1/2 by 10 feet in size (approx 2.2 by 3 metres). The three screens above the stage are fixed in place, and illuminated by ceiling-mounted video projectors; the remaining three are at stage level, movable, and use a self-contained rear-projection system. The six projectors, in turn, are backed up by a technical extravaganza of Amigas, video cameras and recorders, lasers, and other special effects.



The lobby of the Theatre features a colourful neon and sculptured mural display.



Interior of the Amiga Studio Theatre – seating area. The theatre is in the process of being set up for another act.



Some of the 25 monitors used to entertain waiting patrons in the lobby.

During the course of the fair, the theatre has a wide variety of scheduled performances, including dance, small orchestras, and stage plays, up to nine events a day. The majority of these use Amiga computers to generate performance material to complement the live performances. For anyone seriously interested in this machine, or even just in the potential of computers in the arts, the theatre is worth a trip to the fair all on its own.

The number of Amigas and the jobs they look after in the theatre are truly impressive. First of all, Commodore is supplying computers to the performing artists for advance preparation of performance material. At any one time, 18 to 20 machines are out on loan in this manner to groups who are commissioned to produce artistic works. Later, when these groups arrive at the theatre, there are two more Amigas in service in the control room. Graphic material which the performing groups have prepared previously and stored on tape or disc can then be displayed; in addition, the Amigas are applied to routine tasks such as projecting title credits onto the screens. A third Amiga in the control room is reserved for interaction with the audience in a current show, which I'll discuss further in a moment. Finally, yet another Amiga is used to entertain visitors waiting to enter the theatre. This computer prepares graphics displays (some pre-packaged, others developing as they run), and shows them on some 25 monitors arranged in the anteroom.

During a short visit to the control area, I tallied six videodisc players, six tape players, chroma-key and Fairlight equipment, some 27 video monitors, and two windows (real ones) offering the best stage view in the house. Horrors – the four Amigas even had to share shelf space with three Apple II's!

Sample Applications

Your intrepid reporter selected a couple of performances to view the results – all in the line of duty, of course. The technical capabilities of this setup are difficult to describe in words, so perhaps it's better simply to explain how they are applied in these two performances.

The first is a production by The Great Canadian Theatre Company of Ottawa, entitled 'All I Get is Static'. At its core, this is a fairly traditional stage play concerning the lot of Reginald Fessenden, a contemporary of Marconi who was the first to transmit voice by radio, but was forgotten by history. The performance begins with a five-minute graphic-and-sound introduction: the graphics were prepared on the Amiga and other studio equipment, and include a mixture of digitized images, pure graphics, and video wallpaper. This is followed by the main performance, a live half-hour play by three actors. This would be a worthwhile performance in its own right,

but is made far more entertaining through the layering of additional characters, as well as background graphics, from the video system. With this performance, all graphics were prepared in advance, and displayed from videotape.



The control room of the Amiga Studio Theatre: the foreground controls are for switching and special effects, while the Amiga in the background is reserved for the TheatreSports event.

It would have been all too easy to detract from a worthwhile play by superimposing unnecessary techno-pop graphics, merely to demonstrate the computers' capabilities. However, that is certainly not the case with this performance. The graphics are very carefully integrated, woven into the production and throughout the plot, so that they seem to be a natural part of the play. Oh yes, there's one more role for an Amiga - being used as a stage prop.

The second performance makes much more dynamic use of the Amiga. This production, by an improvisational group called Vancouver TheatreSports League, actually uses members of the audience to construct a sort of life-sized video game; through twists of plot too contorted to detail here, the audience eventually has to rescue the theatre from an alien threat. In the course of achieving this, good use is made of a variety of special effects, mostly computerized.

While the basic plot of this story is established in advance, it unfolds differently with each performance. At various points, members of the audience suggest bits of information - names, places, and concepts - that are then worked into the developing plot. For example, someone in the audience is asked to name a famous time or place in history. A

control-room Amiga contains a list of scenes stored on videodisc, and an appropriate background is selected to represent the desired locale. Cast members then improvise a scene in an off-stage studio, which is superimposed on the video footage and projected for the audience. Later in the show, another volunteer is taken from the audience and included in the composite scene. The location, props, and plot direction can all be varied in accord with audience input. While these concepts are not entirely new, they are freshly presented in this production, and made far more realistic as a result of the many videodisc backgrounds available through the Amiga.

Even more innovative use of the Amiga is made when a number of audience members, up to 52 in total, are given touch-sensitive input devices to influence the progress of the game. Each of these devices consists of two electrodes embedded in small Velcro cuffs that are attached to two fingers of the subject's hand. According to the story line, these 'fingerbinders' are used to measure and collect energy to power the theatre: in fact, they are measuring the galvanic resistance of the subjects' skin, an indicator of their mental state. This is where the extra computers in the control room come into the picture - an Apple II collects this information and passes it to the Amiga. The Amiga then constructs a graph showing the 'energy level' using different coloured bars for the 52 subjects; this graph is projected in real time for the audience, and fed back into the story to influence the outcome. A little corny, perhaps, but very effective.

Other productions I wasn't able to see include dancers on-stage accompanied by video images of themselves, Amiga-synthesized voices welcoming visitors to the studio, and many more fabulous shows promised for later in the summer (after press deadline, unfortunately).

Behind the Scenes

Putting on innovative shows of this type involves the participation of many people besides the performers. As well as the many assistants, control room technicians, and so on, innovative work requires someone to make it happen. I already mentioned Doug Welch, the Theatre Manager. In charge of the several theatres and other performances is John Crompton, Director of Cultural Programming for the Canada Pavilion. It was his original vision at the inception of the pavilion that provided the opportunity for a theatre combining artistic performances with high-tech effects. While John is responsible for overall coordination and lining up performers, it is Diana Bockus whose experience and creative forces have brought the Amiga Studio Theatre to life. Over the past eighteen months, she designed the studio facilities, oversaw their installation, and worked with each of the performers in adapting their material. Any expert in this business must be self-taught, since authoritative books and courses are only now becoming available. Originally from Toronto, Diana's background includes five years in development of interactive video; in a young art-form such as this, five years looks a lot like forever!

Despite the imposing title of Video Development Manager, Diana enthusiastically gave me the inside look at the technology, starting with her own office. While a PC/XT sat idle to one side, she proceeded to demonstrate on a fully-equipped Amiga system, with Tecmar memory expansion and 20 Meg hard disk, an extra 3 1/2" drive, printer and mouse, all hooked up to a Sony LDP 1000A videodisc player and extra monitor. A venue like a World's Fair is a mixed blessing - while it provides the opportunity for new techniques to blossom, the tight five-month schedule leaves little room for experimentation. Nonetheless, Diana indicates that there will continue to be

progress throughout the fair. She also sees ample area for continuing growth of the medium, with the logical next steps being into artificial intelligence, and into total automation of the theatre, with manual override as necessary.

As for software, much of the work is done by custom programming, either totally written for the task at hand, or else modified from standard material. Software to retrieve videodisc imagery, for example, is actually a two-part job; one is the supervisory task of controlling the machinery itself, while the other is the interface with the user, who must be able to select from among hundreds of images quickly and efficiently. The software for this purpose was written by Bill Henning, a programmer at nearby Simon Fraser University, and handles the different tasks well. One example of software requiring modification is the use of chromakey to superimpose live actors on the background scenes. The frame-grabber and genlock could be applied to this task, but the present software configuration simply does not respond to the requirements of live performance, primarily in terms of the time needed to access these functions.



Your intrepid reporter takes a self-portrait from the frame-grabber display's monitor



The office of Diana Bockus, Video Development Manager of the Amiga Studio Theatre. This Amiga is fully equipped with 2 megabytes, a 20-meg hard disk, extra 3 1/2" disk, and videodisk player and monitor. A prairie scene from the videodisk player is displayed on the second monitor.

Other software currently being used includes DPaint, DPSlide, and DPVideo for titling and other graphics effects. These also had to have some modifications to improve access time.

Other Applications

Beyond the Studio Theatre, Amigas are also in use in the interactive displays in the main pavilion area. In a viewing area, a visitor stands in front of a video camera, while his or her face is continuously displayed by the frame grabber. At the touch of a button, the image is frozen: the Amiga then produces that image at several different resolution levels, and displays them on a series of video monitors. This seemingly simple process is sufficient to draw a crowd and keep them entertained.

Well, I'm sorry to talk at such length, but the IBM information system and the Amiga Studio Theatre seemed to offer such optimistic views of the future of computers, that I just had to elaborate. However, they only scratch the surface of the computer activities at Expo '86. Computers are used routinely for just about everything, from running the traffic lights and train system, to finding your lost children after the computerized fireworks.

Just one last example - not far from the Amiga Studio Theatre, there are two computerized robot arms that methodically manipulate a piece of paper. This activity quickly draws a curious crowd, anxious to learn what the robots are up to. . . until they finish folding their paper airplane, and launch it into the crowd!

That amusing demonstration perhaps epitomizes the World's Fair - while you're having fun, you hardly notice that you're being educated as well. Judging by the enthusiastic response of the crowds, the many computerized

displays at Expo 86 have quickly earned public acceptance in their applications, thus contributing actively to broader role of a World's Fair and, in the process, to a better understanding of our world.

For further information, contact the author at:

4425 West 12th Ave., Vancouver BC V6R 2R3 604-224-5879
or:
IBM Canada Ltd
PO Box 10132, Vancouver BC V7Y 1G1 604-664-6600
or:
Expo 86 Canada Pavilion, Vancouver BC V6C 3C1 604-666-2000

The Comspec AX2000: 2 Megabyte RAM expansion for the Amiga

Chris Zamara, Technical Editor

Price: \$1276 (\$899 US) (plus appropriate taxes)

Warranty: 1 year parts and labour

Manufacturer: Comspec Communications Inc.
153 Bridgeland Avenue, Unit 5
Toronto, ON. M6A 2Y6
(416) 787-0617

Think about two Megabytes for a moment. That's 2,048 Kilobytes. Putting that much memory on a micro sounds akin to strapping a Jet engine to a Chevette, but believe it or not, it fits the Amiga more like a high-boost turbocharger.

You see, the Amiga, by its very nature, loves using lots of memory. Its internal 512K can quickly fill with screens and windows opened by programs in the system. The multi-tasking nature of Amiga makes it only too convenient to bring in another program on a whim and have it sitting around in a window for whenever you need it. 50K here and 100K there can quickly add up. (If anyone is trying to run an Amiga with the minimum 256K, you know what I mean.) Another RAM-eater is the built-in RAM disk, which can be used to speed up program development, or cut down disk swaps with a single-drive system. All of these factors contribute to making Amiga a machine which greatly benefits from added memory. That's a nice way to call it memory hungry.

Before getting into the review about the Comspec board specifically, I'd like to say a few things about memory on the Amiga in general - one 2 Meg expansion will function much like another.

Amiga holds two flavours of memory: "chip" ram, and "fast" ram. Chip ram is accessible by the video chips, sound chip and DMA channels, and cannot be expanded beyond the internal 512K. This 512K comprises the built-in 256K plus the optional 256K expander which plugs in behind the front panel. All graphics information, such as screens, gadgets, sprites, windows, etc. must be stored in this chip ram, so the number of graphic-intensive programs you can run at once is limited no matter how much expansion RAM you add. The video and sound chips access chip RAM directly, and have priority over the CPU. If you are in hi-res mode (640 X 200 or 640 X 400) and using 16-colours, for example, the CPU will only be able to access chip RAM during the screen's vertical and horizontal retrace - about 24% of the time. This will obviously slow things down if you're trying to execute a program residing in chip RAM while displaying a hi-res screen.

Fast RAM, on the other hand, does not conflict with DMA access from the chips, so a program executing from fast RAM will run at full speed regardless of any graphics or sound activity. A stock Amiga has NO fast RAM, and any RAM expansion beyond the 512K chip RAM is "fast". Thus, Comspec's box reviewed below gives you 2 Meg of FAST RAM.



Since non-graphic information such as programs are best stored in fast RAM, the operating system tries to use fast RAM whenever a program asks for some memory. If a program just asks for any old memory and doesn't specify what kind, the system will allocate fast RAM. That creates problems for some programs, because they were written on non-expanded Amigas and their authors didn't think of allocating chip RAM for graphics information (for example, a gadget or sprite may be coded as part of the program itself, which will reside in fast RAM). To the user, that means that some programs that worked fine on the standard Amiga won't look right when you run them on the expanded machine. That may lead you to believe that the fault is with the added RAM, but the fault is actually with the program itself. Even version 1.0 of Amiga's operating system had a little trouble with RAM expansion. So beware - the software you're using now may not run properly with extra RAM on the system, but the problems will probably be fixed on program updates and all future programs. The changes aren't that hard to make.

So, if added RAM doesn't necessarily let you run more programs, and some of your existing software won't work with it, is it really worth the significant amount of cash that you'll have to put out for it? That all depends on what you're using your Amiga for. If you are developing programs, in C, assembler, or another language, the use of a large RAM disk can speed up compiles greatly. If you use AmigaDOS commands from CLI frequently, you may want to copy all commands to RAM and have them execute instantly. If you only have a single drive, bolting on a super-fast 2 Meg RAM drive can definitely make life easier for you. And even for just running application programs, you can have more data in memory at once

for your project, cutting down disk access and allowing longer documents, bigger spreadsheets, faster database access, etc.

The greatest demand for expansion RAM, though, will probably come from software developers, who will want to keep all of their CLI commands, software tools, editor, compiler, libraries, source files, and include files in RAM. You just can't do that with 512K. In fact, for this kind of programming, even running just an extra 1 Meg would be tight. The added 2 Meg, along with the internal 512K is just about right for a good "turbocharge", and isn't as much of an overkill as it sounds.

To get an idea of what the added RAM could do for compile times, I tried compiling Commodore-Amiga's program "doty.c" using the Lattice C Compiler under three different conditions. The Compile and link times for the three tests are given in minutes and seconds.

In the first test, the source code, compiler, linker, libraries and include files are all on a single disk, and the .q, .o and executable files are being written to the same disk. The second test involved putting just the source and object files in RAM and using everything else from a single disk. In the final time test, everything - Compiler, linker, include files, libraries, and source - was stored in RAM. Using all of the includes from the developer's disk, plus a few favorite DOS commands costs just under a Meg of RAM.

Test 1: Everything on a single disk:

Compile: **02:50**

Link: **02:02**

Test 2: Source and object in RAM:

Compile: **01:27**

Link: **01:44**

Test 3: Everything in RAM:

Compile: **00:44**

Link: **01:12**

As you can see, in this example compile time was over 3.8 times faster when everything was in RAM. For source consisting of many files, the RAM advantage would probably be even greater. To even further reduce your waiting times, there's still room left in RAM for your favorite editor and any programming utilities you like to have kicking around. No wonder software developers like extra RAM so much.

There are disadvantages to using RAM as a disk like this, though. First of all, setting up for your programming session can take quite a while, as copying a disk full of files into RAM is no instant operation. And the worst part is that every time you have to re-boot (can you say, "Guru Meditation Time"?) you will lose *everything* in the RAM drive, meaning re-copying again. So you'll still have to save any work you're doing on disk once in a while. Of course, no amount of RAM expansion is a substitute for a hard drive, but it would be so nice if there was some way for the RAM's contents to survive a re-boot.

Comspec's 2-Meg RAM Box

Comspec Communications is a Toronto-based company known among many Commodore users for their "Microshare multi-user

system" line of products. The Transactor has been convinced enough about Comspec's reputability to offer their RAM board on our mail-order card. The AX2000 goes for \$1276 (Canadian, plus federal and provincial taxes), and the 1-Meg AX1000 is \$1035. (US prices are \$899 and \$729 respectively.) If you're a little wary of a review about a product sold by the magazine reviewing it, keep in mind that we reached our conclusions about this RAM board before deciding to take it on.

The AX2000 is a small steel box which plugs into the expansion slot on the right side of the Amiga. It has an identical slot on its side for further expansion, and you can snap in the plastic slot cover from the Amiga if you don't need to plug anything else in. The box is Amiga-White, and measures 9 1/4" by 4 1/4" by 1 3/4" (not including the protruding edge-card connector). The extra 1 3/4" that it adds to the Amiga's width isn't too demanding of desk space, and the box is the same height as the Amiga, creating a perfect spot to the right of the monitor to place a second drive. Quite a compact package for all that RAM, really.

Besides the unit's small size, there are other factors which give the impression of a well designed piece of hardware. The two-Megabyte board draws only 500 milliamps of current, the same as competing one-Megabyte units. Because of the low power draw and the design of the board, you can plug in two RAM boards, one into the other, and add a whopping four Megabytes. There are no ventilation slots anywhere on the box, since the little heat that is internally generated is dissipated through the steel case. And a big plus circuit-wise is that the board causes no wait states, never making the CPU wait for the RAM to catch up to it.

The RAM will "auto-config" itself, providing that you're using at least release 1.2 of Kickstart and Workbench, which have auto-config implemented. This means that once you plug the box into the Amiga, the RAM will automatically be added to the system when the Amiga is powered up and fed its usual two-disk diet. The auto-config process lets you plug in any configuration of RAM, devices or whatever without worrying about setting DIP switches to avoid memory conflicts. If you are using older operating system versions, you can add the RAM to the system using a program included on a disk which comes with the RAM box. This program will do a RAM test, then add it to the system. You can include a call to this program in your Startup-Sequence to automate the procedure, so running the RAM with older system versions isn't really any more hassle.

The disk that comes with the AX2000 is actually a Workbench 1.1 disk with a special Startup-Sequence, two extra commands in the C directory, and two "Read__Me" files. The Startup-Sequence is cleverly designed to first test the RAM and print installation instructions if the test fails. This way, you can boot the disk when you first get the RAM and learn how to install it, then use the same disk once it's properly plugged in.

I have been using the AX2000 for several months now and have had no problems with it. The difference it makes to programming productivity and the amount of "work" it seems to save is incredible. Now that I've been spoiled, life with the Amiga just wouldn't be the same with a measly 512K. Then again, once an 8-Meg board comes out, I'll probably be saying the same thing about only 2.5 Meg. Now, about installing that Jet engine. . .

A Peek At Amiga Disk Structure

Betty Clay
Arlington, Texas

In order to understand the Amiga file structure, it is necessary to learn some terms that have not been familiar to Commodore users. We are accustomed to a directory track; the Amiga has a hash-chain and a root block. We speak of the BAM; the Amiga has bit-map pages. We are accustomed to a single directory on the center track, and the Amiga can have many directories in various locations, and even of different types. We are accustomed to program, sequential, and relative files, with most sectors looking very much alike. The Amiga has root directory blocks, user-directory blocks, header blocks (also called keys), list blocks, extension blocks, and data blocks, not to mention corrupt blocks and deleted corrupt blocks! Once the new words are learned, however, there are some similarities.

The Amiga directory begins with the root block, which stores the location of other directories on the disk, and of files not assigned to one of the other directories. Just as we are accustomed to the directory track being at the center of the disk, so is the root block on the center track of the Amiga disks.

On the current Amiga drives, a disk is formatted with eighty tracks, each having twenty-two sectors, and each sector having 512 bytes. Each sector is divided into 128 slots containing 32 bits each, and each slot may hold one 32-bit value, two 16-bit values, or four 8-bit values. The Amiga calls the tracks 'cylinders' and the sectors are called 'blocks', which is already a familiar term. There is also a 'surface' number, which is always given as zero by the DiskEd disk editor, but as either zero or one by DiskDoctor!.

The root directory on each Amiga diskette is at block 880 (track 40, sector 0) on the current drives. The block contains 128 slots for information, numbered 0 through 127. The zero slot holds the file type identifier (a '2' for 'short file'). Slots one and two in the root block always contain zeros; slot three currently holds the value 72, but on other drives might hold other values. Seventy-two is the value obtained when you subtract the number of slots that hold identifying information (the first six and the last fifty offsets) from the 128 that are available. If a future DOS is set for blocks longer than 512 bytes, then there will be more than 72 slots left for the hash table, and slot three will hold a different value. Offset four is unused on the root block, and number five holds the checksum on all types of sectors.

Hash Tables

Offsets six through seventy-seven of the directory blocks contain the 'hash-table'. 'Hashing' is a mathematical process through which the name of a file is changed to a numerical code, using a formula that will ensure that every possible name will be encoded to a number within the range allowed for a particular use, in this case a number between six and seventy-seven, inclusive. The formula is built into the DOS and is automatically calculated for us, so we do not need to know exactly what the formula is. To understand the hash table, we need only to understand that every possible file name will hash to a value of six through 77, the same numbers as the slots in the hash table.

If your file name has a hash value of twelve (and if it is the first file in the directory that has that hash number), then the number of the block in which your file begins will be stored in slot twelve of the directory block. Each slot contains the block number of a file whose name hashes to the same value as the slot number. Were this the end of the matter, that would mean that a directory would be limited to seventy-two files and that no two could have the same hash-value. This would be an intolerable limitation, so we must consider the hash-chain also.

The Hash-Chain

Suppose that you have saved a file called "filename" in the root directory. "Filename" has a hash-value of 59. Suppose this file begins with its header block in block 884. Slot 59 of the root directory will now contain the number 884. Suppose that later you wish to save a file in the root directory and call it "File4", which also has a hash value of 59. This file will begin on, say, block 985. When the root directory is examined prior to storing "File4", it will be discovered that slot 59 already contains the number 884. Block 884 will then be read, and offset 124 (the hash-chain slot) of that block will be examined to see whether it contains a block number or a zero. Should it contain a block number, that block would be examined to see if it contains a zero in slot 124, and so on until a block in the chain is found with zero in this offset. When the zero is found in the hash-chain slot, the zero will be replaced with 985, placing "File4"

on the proper hash-chain. Then "File4" will be written to the disk, beginning with its header-block in block 985. Slot 4 of block 985 will hold the number of the next block used for "File4", and slot 124 will contain a zero, showing that this is the last file on the chain at this time. It is this hash-chain that permits the Amiga to have an apparently limitless number of directory entries, yet the drive has a relatively small number of items to read in order to locate a specific file.

When you give a command for "File4" to be read back, a similar process is followed. That is, the name will be converted back to its hash-value, 59. Then offset 59 of the root directory will be examined, and the 884 stored there will cause block 884 to be read. The two filenames will be compared and, since they do not match, slot 124 will be read to find the number of the header block of the next file with a hash-value of 59. In our case, offset 124 will contain the number 985. Block 985 will then be read, and the file name there compared with "File4". These will match, so the file will be loaded.

The Other Fifty Slots

At the end of the root block are fifty more offsets which contain information about the disk. Slot 78 holds a flag to indicate whether the bitmap is valid, and is followed by twenty-five slots reserved for the bitmap pages. The following three slots, numbers 105-108 on current disks, contain the date and time the disk was last altered. The date is held as a single number, and is the number of days between Jan. 1, 1978 and the date of alteration. (Don't forget to include leap years.) The time is stored as the number of minutes since midnight and as the number of seconds since the hour. The next twelve slots are reserved to hold the name of the disk, which is limited to thirty characters. Slots 121 - 123 hold the date and time of the disk's creation, the next three are not used on a root block, and the last one holds a number that indicates the secondary file type (1 for a root block).

User Directory Blocks

A user-directory block is quite similar to the root block. The first six slots are the same except that slot one holds the block's own number. The hash-table is just like that in the root block. Instead of the bitmap, however, there are protection bits and twenty-five slots in which your file comments are stored. Slots 105-107 contain the time and date the disk was formatted, held in the same format as those in the root block, numbers 108-123 are reserved for the name of the directory, and slot 124 is for the hash-chain. Slot 125 of the user-directory holds the header-block number of its parent directory, number 126 is not used, and 127 contains a '2' to indicate the secondary type 'user directory'.

If you indicate a file path when you store your files, the header-block number is stored in the proper offset in the user-directory rather than in the root block. Suppose that you had made a user-directory called "Datafiles", which has a hash

value of 32, and had stored "Filename" in the user-directory. Then the root block will store the block number assigned to the user-directory in offset 32, and offset 59 of the user-directory block will contain block 884. This is why the full directory path must be spelled out when you call for a file. The root block must be read to find the user-directory, the user-directory block read to locate the key (block number) of the file, and then the header-block of the file read to find the blocks in which the data is stored. If you want a printout of the keys to your files, you can get it by typing:

```
LIST TO PRT: directoryname KEYS.
```

File Header Blocks

A file-header block contains 512 bytes, 128 slots, but they are used differently here. Offsets zero and one contain the usual information (file type and number of this block), but on these blocks offset two contains the number of blocks used for the file, three tells how many of the slots (6 - 77) are used to store numbers of data blocks, and slot four holds the block number of the first data block. Slot five holds the checksum, as usual. The next seventy-two offsets hold the numbers of the data blocks used for the file. The block numbers begin at the end of the table, slot 77, and move upward to number six. (In most cases, the files are stored in eleven consecutive sectors, skip twenty-one sectors, and then use eleven consecutive sectors again. The Amiga reads an entire track at one time, making this an efficient method of storage.) Offset number 80 holds protection bits, and number 81 contains the size of the file in bytes. There are twenty-two slots for your file comment, and the rest of the information is the same as in the user-directory except for number 126. It is used only if the data-block table is too large to fit into the seventy-two slots in this block. When a file contains more than seventy-two blocks, another block is designated to hold the data about the overflow. It is called an extension block, and its number is stored in slot 126.

File List Blocks

The extension (file list) block begins with the now-familiar six items of information: the file type, the block's own number, the number of offsets in this sector used for the block list, the number of the first data block, and the checksum. The data-block numbers again begin in slot 77 and are stored upward until they reach slot 6. If necessary, yet another extension block is assigned. The area reserved for comments and the date are unused on these blocks, and there is no hash-chain offset, but the last three slots are used to indicate the number of the parent directory, the number of the next extension block (if any), and the secondary file type.

Data Blocks

On these blocks, only the first six items of file data are used. They hold the type of block, the block's own number, the

sequence number in the file for this block (is it number 5, etc.), the number of bytes of data in this block, the number of the next data block in this file, and the checksum. The remaining bytes are used for the data. Normally, all data blocks will be completely filled except the last, so the number of bytes will be 488. The last block will contain fewer bytes, and slot 4 will contain zero for a forward pointer.

Editing Disks

Of what use is this information? Well, it answers some of the questions that are asked about disk organization, of course. But it is essential that the hash-chain and the hash-table be understood if you are to edit your disks. File recovery is an important part of disk maintenance, and files cannot be recovered if you don't understand the structure.

A disk came to me recently that was almost full, but the root block contained only one key number, and that one was incorrect—a completely corrupted disk. Every time we tried to read a directory, or even insert the disk in the normal way, we had to reset the computer. Yet we recovered that disk!

The disk editor we used was DiskEd, from the WACK disk included in the developers' package. This disk is now available for anyone to purchase. In order to recover the disk, we first had to locate the header blocks of all the files we wanted to resurrect. Most user directories and header blocks are stored in the blocks between 880 and 1100. The blocks with smaller or larger numbers are usually data blocks. By using DiskEd's 'g' command to read the blocks in this range, we noted the header blocks and hash-value of each of the files we needed. With our information complete, block 880 was read, and the number of each header block was restored in the offset that matched the hash value of its file name. AmigaDOS is pretty smart. It refused to accept the wrong block number in a slot! When all of the headers had been restored to their proper positions in the root block, the checksum was corrected and the block written back to the disk. The directory was called, and the entire disk was reclaimed. Only the root block had been damaged, but without it the remainder was useless.

Had we wanted to eliminate a file that was making it impossible to validate a disk, we could have put a zero into the slot that referenced the bad file. This would make the remainder of the disk usable, but that file would be permanently lost once something was written to the disk. On the other hand, it could also have been restored by placing the header-block number back in the root block. Perhaps a reading of the file would make it possible to correct its error and save the file.

The documentation for DiskEd is sketchy, and it has not reached the popularity level it deserves. To use DISKED, place the WACK disk in the drive and, at a CLI prompt, type:

```
cd df1:wackstuff
Disked df1: (or df0:).
```

You will see a notice that DiskEd has been activated, and each line after that will begin with a prompt, "#". Some commands you will need to use are:

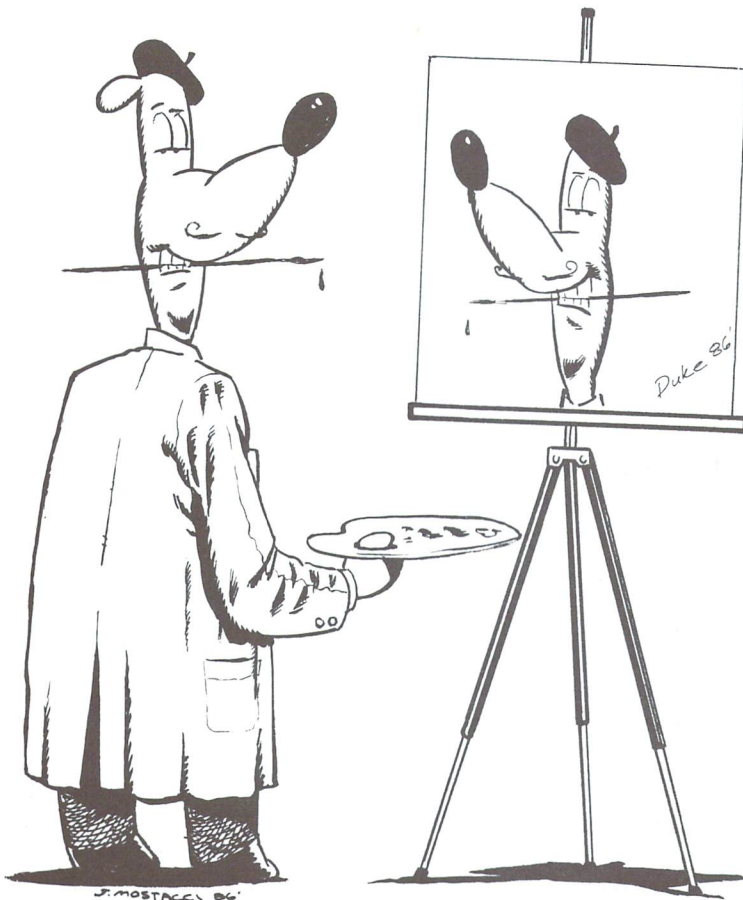
- g** – to read the block you want to change or read (example: g 880)
- i** – to display information about the block just read
- h** – to get the hash-value (and thus the slot number) of the filename. Type h and the file name.
- t** – to display the slot number or range of numbers you want to examine. Put a space between the numbers in the range – not a hyphen, as in 't 6 20'.
- /** – to change a number in the slot. Type the slot number, a slash, and the number you want to place in that slot. Example:36/5 will change the value in slot 36 to 5.
- k** – to correct the checksum after you have made a change
- x** – to set or unset the write-protect mode
- p** – to write the corrected block to the disk
- q** – to get out of the disked program

If you follow this sequence of commands, you should be able to edit your disk. It is not a complete explanation of DiskEd. There are fourteen other commands available, but the nine given here are sufficient. They will permit you to change any byte of information on the disk. I have found DiskEd to be reliable and easy to use.

Other disk editors? I just received a pre-release version of Kickstart and Workbench v1.2, and there is a DiskDoctor in the new "c" directory. For my first effort with it, I duplicated a disk known to contain an error. DiskDoctor read the disk looking for all the world like a formatting operation, except it said "Reading cylinder 1", etc. Then came a series of reports showing the files that were being replaced on the disk. Whenever a serious error was encountered, it was reported on the screen as in "Hard error on cylinder 67." Multiple efforts were made to correct each error. Finally, I was asked whether corrupt files in a named directory should be deleted. The files were not named – just the directory. I answered 'yes', and found that the programs were not only deleted, the blocks in which they had resided were completely reformatted as though they had never been there. On another disk, I answered 'no', and the files were not deleted. I tried it on a disk that had managed to get the directory into an endless loop, and DiskDoctor corrected the error with no effort at all on my behalf. Remember that I was using an alpha version of an early pre-release version. Perhaps when this version is complete, there will be no need for knowledge on our part to repair disks, but for now, it is better to learn the file structure and do it ourselves. With DiskEd, we have complete control over every item written out to the disk. With DiskDoctor, we just sit and watch it do its thing. It is easy as formatting a disk or making a backup. Each of these tools will fill a need for most of us. DiskDoctor is fast and easy, but the control offered by DiskEd will permit us to change any byte on the disk. We'll want to do it both ways.

Disk Copier Comparison

David Martin
Seabrook, Texas



Diskmaker V3.3

DiskMaker was introduced several years ago and has since seen very little improvements in its arsenal. However, earlier this year, Basix released V3.3, only a small improvement over earlier versions, that incorporates modular design and speed increases.

The modular design comes in the form of updates to the disk nibbler called "Masterkey" modules. These updates are available individually or through subscription at a very high price. Recent updates have also included speed increases during disk copying.

I advise Commodore users to avoid this program until it sees vast improvements. The program should see the following improvements to justify its high cost of \$49.95, which even then will cost too much. It needs the addition of programs that will allow a fast copy on two drives as well as a fast file copier. Also, the addition of disk utilities would be wonderful. Basix has advertised a disk utility package called "Toolkit" for over a year that retails at \$39.95 and offers the features that DiskMaker sorely lacks. However, the reader can see in the comparison chart that expenses run high with products from this company, when other vendors offer more features for under \$30.00. Again, until Basix prices change or features are added, this is one of the titles to avoid at all costs. I rank this title a one on the 1-10 scale due to its high cost, no added features and updates that are not vastly improved since version 1.1.

Keymaster

Keymaster is MegaSoft's newest disk copier, which retails at \$29.95 and does incorporate some of today's newest disk copy ideologies (parameters, fast DOS, etc.).

The program supports various disk drive configurations (single or dual) using a 1541 or 1571. Also, the program is fast and will copy disks on a single drive in under three minutes and under one minute on dual drives. Additionally, it supports a fast format that formats disks in about 10 seconds.

Keymaster also offers a method for fast file copying using a rather unique method of transferring files from disk to disk.

Copy protection is becoming more and more complex month after month, often making it extremely difficult for a person to protect his newest software investment by keeping a duplicate copy on hand.

In fact, some of the newest techniques push the 1541 to its limits or beyond. They will often go where no 1541 has gone before, recording data well out past track 35, even out as far as track 40. These schemes are apparently very sensitive to drive alignment problems, which means that if the alignment is off, the program may not Load and Run.

Today users face a better opportunity in creating backups of their valuable software through the use of state-of-the-art disk copy programs which can considerably speed up copy times and provide many other features to enhance disk drive usage.

The scope of this review is to help the reader decide which copier is best in terms of features and cost while leaving the final decision up to him. Therefore, I am basing my recommendations on cost and available features on a comparison basis, since not every program offers the same features at the same prices. So, make yourself comfortable as we begin to explore the wide world of disk copiers.

This is done by fast loading a file into memory and then fast saving it back to disk. A drawback to this method is that, since the program does not use a buffer, which could hold more than one program, the user is forced to swap disks many times (at least 2 or more for each file he wishes to copy). Keep in mind that this method will support one file at a time no matter how large or small the file is.

A further enhancement to Keymaster is the ability to copy disks using parameters for backing up some disk titles easier than before. The program comes packaged with 50 parameters, and more are available on disk at \$10.00 per disk from MegaSoft.

Here again, I advise readers to avoid this program since it is not cost effective due to the lack of features and lack of support from MegaSoft (I'm a registered owner who was not advised of recent updates from them). This program does perform as advertised, under strict hardware configuration conditions, as mentioned in its two page manual. My dual drive fast file copy test failed because of Keymaster's lack of 1571 support, a problem that the other programs did not face. Lack of support and features as well as high cost ranks this program a low three on a scale of 1-10.

Copy II 64/128

Copy II is Central Point Software's entry as a disk copier, produced by a company that is well known for producing disk utility software for other name brand computers. This program does offer high speed copying and a few utilities in its arsenal. It even offers a different method of presenting parameters via an enclosed leaflet that tells how to set the programs parameters to copy certain disks easier and accurately.

Currently, Copy II does support the C64 and C128 as well as the 1541 and 1571 disk drives. It does provide a working copier on these drives in single drive or dual drive set-up. The copier is able to examine the hardware present to determine the single or dual mode of operation, and uses the information to load the appropriate program.

The utilities that Copy II currently supports are a delete file program as well as a fast load for BASIC. It would be nice to see more utilities for this program that could help a user in determining how to set the programs parameters. This currently is impossible, and requires that the user refer to their enclosed parameter sheet. Thus, Copy II can very quickly become outdated.

Again, as a copy program, Copy II does not offer anything that makes it worth a cost of \$39.95 (and half price for updates). Here, the program is similar to Diskmaker since it offers very little for such a high cost.

Although compounded by the lack of support outside the copy process, the program does work well, but would work better with some form of disk scanner with which users could check disks not listed on the parameter sheet for protection to set the program up accordingly. Hopefully, we will see these changes

in the near future from a company that supports better products for other machines. At this time I do not recommend the purchase of this program because it is not cost effective and offers little added features, unlike other programs. This program is awarded a rank of five on the 1-10 scale.

Disector V3.0

Disector is the program from Starpoint Software that started it all a few years back. The program has grown quite a bit from version 1.0 that we all started with at one point or another. Recent improvements to the program in version 3.0 have been a welcome sight to the copier world with improvements that have followed what I hope to see as a continuing trend towards programs that carry more features at a very good price.

Disector is a program that has improved by supporting fast DOS routines for both single and dual drive owners. These programs are rather fast and do function on both 1541's and 1571's as well. I did, however, experience some trouble with the dual disk copier when it got wind of certain protection schemes (the single disk copier didn't experience any problems). I would expect that an update will clear these problems up.

One disappointing feature of Disector is its lack of a fast file copier contrary to the packages advertisement of an "Ultra fast file copier. ...". I found this to be very disappointing after examining the program's other features.

The program also provides a parameter copier which only supports about 40 parameters (this is also disappointing - see the comparison chart). This utility is basically like the others, which require you to do a nibble copy then perform some type of special function using the program.

For the most part, I do think that Disector is going in the right direction and found that its disk drive monitor utility, that is yet to find a rival, well worth the purchase price of \$39.95. The monitor itself will disassemble unimplemented opcode, and I find it to be a very invaluable tool in my programming and tinkering with the C64 and 1541 DOS. It does not currently support printed output, but I have been told that a new version of the program now supports printer output from the monitor. A note here: A majority of the programs in Disector do not support printer output where it is sorely needed. For example, in the Format Editor you cannot print the results of a disk scan. Currently, users are required to hand-write information about the 1 to 40 tracks on the diskette - this can lead to writer's cramp, which is quite painful, believe me! Hopefully, future versions will clear up this very important problem.

Here again, I can see a program that comes from a reputable company and would suggest its purchase for the drive monitor alone. Hopefully, some of its flaws will be repaired and some additions made to its features in a future update. Although this program is awarded a seven on the 1-10 scale, it is still not the most cost effective. However, read on and find the cure.

Fasthack'em V3.0

Fasthack'em, one of the most popular disk copiers, by Mike J. Henry of Basement Boys Software, is the program that started current trends in disk copiers. In a manner of speaking, this is the one the others are trying to beat, and currently, no one has come close (but read on).

The program is quite good and does come with some rather interesting programs supporting Commodore drives as well as the MSD dual drive. At a retail cost of \$29.95, I found it very good value for the following three reasons.

First, the program does support a fast load program that is the only one currently available, to my knowledge, that will work on both the 1541/71 as well as the MSD. The program is intelligent and, when activated, selects the type of drive in use and works accordingly. This is nice because I can now load programs faster on my MSD, which also allows more storage than the 1541.

Second, the program supports a method in which the user may disconnect the computer from the drives (MSD or 2 1541/71's) and still copy disks. Fasthack'em will then take over the drives and allow copying without the computer. Now, you can fool your friends with C64 multi-tasking as you use a word processor at the same time, or something like that (I usually use my modem while all this is going on). However, remember not to let them see the disconnected cables from the drives.

Third, the program not only supports the use of 1541/71's in C64 mode, but will support them in C128 mode as well. I hope to see the C128 support expand more and more in the near future. Currently, the program will copy a double sided 1571 disk in a little over a minute. A little note here about speed; the program will copy most disks in under two minutes, and sometimes under three, depending on the amount of data and/or degree of protection. However, Fasthack'em does have a few drawbacks from an otherwise well thought out program.

First, the program currently does not support any form of disk utility such as disk editors or scanners. These would be very helpful in exploring the disks that Fasthack'em will not copy. At times, when copying disks, a parameter change within the nibbler might help a user create the backup, hence the need for disk scanners and editors. Currently, users are unable to do so and must experiment with the various copy programs on the disk. Secondly, the program parameter copier, of which a majority of programs have duplicated, will often display instructions for copying disks and then exit to BASIC. It would be nice if the parameter copier would execute as a copier without having to load the various other programs in the Fasthack'em arsenal.

The program Fasthack'em comes from a very reliable author who plans on expanding Fasthack'em in the near future. I highly recommend this program for MSD dual drive owners, as it supports that drive quite well. This alone makes it well worth the purchase price for MSD owners, as no one other than

Basement Boys produces such a great MSD utility product. Also, the auto copy feature that does not require the computer is nice for us lazy users, or people who wish to make multiple backups (sigh, if only we could get it to put the disks in!).

Although I greatly enjoyed the usage of Fasthack'em, I must say that its lack of utilities is giving some of its competition an edge, and hopefully we will see it expand and be more valuable in the near future (I am told that Version 4.0 will support some utilities).

For MSD dual drive owners, I do recommend the purchase of this product since it currently is the best and only supporter of the MSD. However, for 1541 users, it does not really support them to the full extent that it should, utility wise (which would extend its ability to function). I am ranking this program a nine on the 1-10 scale due to its lack of available features.

Super Kit/1541

Super Kit/1541, from Prism Software, is an innovative new product that supports a majority of features for 1541/1571 users for a low cost of \$29.95.

Having recently received this new product, I find it to have the flavour and flare that I have been looking for for sometime. At last, someone has released a full featured disk copier and disk explorer for the Commodore 64/128. It uses most of the current methods of disk copying as well as enhancing some of the old stand bys.

SK is currently the fastest around, with Fasthack'em following a close second. I found that version 1.1 was easy to work with although a few problems do arise that become annoying after a while. For instance, a user is unable to return to the main menu from any of the other program modules. In later versions, this feature will be added, if I know my users out there (and I do because - wink! - I am one). For the most part, this is the major complaint with SK.

Another minor problem is the inability to view a directory from the disk copiers (entire disk copiers, NOT the file copier). This is also something that is now being incorporated into a future version, and has been promised to appear in version 1.2 or above this summer. For the time being, keep those disks labeled!

Super Kit (SK) is truly the most cost effective copier to date. It supports both single and dual drives (but not the MSD). The program comes on a double sided disk featuring a total of ten programs. Seven of these appear on side one and are composed of dual and single nibblers as well as normal disk copiers. The rest of the programs deal with file copying and disk editing. Side two, composed of the last three files, includes another nibbler (single drive only) as well as a scanner and disk parameter copier.

The copiers included on the disk on side one are the fastest to date, and involve the usage of recent advances in programming

methods. The programs will operate at the maximum speed possible without major data loss or corruption on Commodore drives. The single drive copiers prove to be very fast and reliable, giving the user a choice of whether or not to verify the copies being made (which can speed up the copy process if turned off). The dual versions offer the same type of options as well as something interesting to do while you wait, which, by the way, doesn't last very long – the dual versions are so fast that it makes it worth going out and buying a second drive, especially if you do a lot of disk copying (i.e. users groups). The thing that I miss here is the automatic option of Fasthack'em that one can become dependent on. However, the program does provide some entertainment to ease the pain of losing the automatic option. During the dual copy process, the program plays music and displays graphics. Although it ain't Mr. Jackson, it does suffice, and seeing those 1541's on the display working as hard as the ones on my desk was truly cute, to say the least.

The SK program is also the first program to provide a dual file copier as well as one for single drives. The copier program operates in two different modes of operation; normal mode and Super DOS mode. The program will copy or scratch files from the disk in the same manner as other programs of this type. It also supports a fast format that makes it easier to prepare disks to receive files. The program will copy SEQ, USR, PRG, and even DEL files from disk to disk. Operation is the fastest to date in either single or dual drive modes. Please Note: According to an inside contact at Prism Software, they are currently revising the file copier to include features like fast verify and fast scratch, among other things. Watch for these in future updates.

The secret of the Fast File copier's speed is in the two modes of operation that it supports: normal and Super DOS. The normal mode uses standard 1541 DOS, except it does speed it up a whole lot. It only uses the standard DOS method of storing information on disk where the files are linked normally. In Super DOS mode, the sector blocks are linked every other fifth block instead of every other tenth. For example, links usually follow like this: 17,0 to 17,10 . . . With Super DOS, they link like this: 17,0 to 17,5 etc.. This can speed up load times considerably as well as speed up copy times. This is because a normal 1541 has a very slow rate of data transfer. When data is retrieved from disk, it normally takes about ten sectors of movement of the diskette in relation to the head at 350 rpm to pass the data out of the bus. With Super DOS, data transfer is so quick that by changing the spacing to every five sectors, a noticeable improvement is provided over its already fast load and copy times.

The File Copier also provides a BAM editor which will allow a user to view and edit the Block Availability Map on the disk. A user enters a plus or minus to allocate or de-allocate disk blocks, then can re-write the edited version back to disk.

The remainder of side one provides the user with several utilities, a sector editor and a GCR editor. These programs are invaluable tools for those who like to tinker with new disk protection schemes or develop their own ideas for such things.

The Sector Editor is basically a standard sector editor with a built in machine language monitor, which does support the printer for printed outputs of displayed data (the editor itself also supports the printer). The monitor is a standard ML monitor that supports assembling/disassembling as well as number conversions. However, it does not support a method of editing memory in the drive or unimplemented opcodes, things that I hope future versions will support. In this module, the BAM editor is also supported (see above).

The best disk utility published to date, that others have presented in hardware or have attempted to do in either software or hardware, has finally surfaced in SK with the debut of the Group Code Recording Editor included in SK, for the basic cost of \$29.95 (with all the above and more!). Basix has advertised such an addition to its DiskMaker product for over a year now at a price of \$39.95 (along with some other utilities), however, such a product has yet to be delivered (I'm still waiting . . . will someone tell Basix to cancel my order?).

The GCR Editor is a wonderfully innovative addition to anyone's software library, be they novice or hacker. This will really allow you to access the DOS in a way you could not do before. It helps users to actually study and devise means or methods of recording data on the 1541 disk drive. As a tool for exploring different disk formats, it has proven to be an invaluable tool and has helped me backup disks that I was not able to do before. Plus, it has brought out the adventurous hacker out in me and taught me a few things I didn't know before about standard DOS formats or the unnatural ones.

Well, enough raving. On to the details. The program provides a means of scanning headers as well as viewing and editing tracks and data blocks. A user can then access a buffer area and edit these sections, viewing and changing the information in the buffers. Later, he may re-save the new information onto disk.

Please Note: I must honestly not recommend this program for the novice user. However, you're only new once, so tinker away at it and experiment on non-important disks. The manual offers some help for novice users by offering a list of books to read (see the end of this article) that will help tutor you in really getting to know the DOS. They are highly recommended. Also, the manual offers a small "book" in the back by Rob Vaughn that is called "The History of Commodore 64 Program Protection", a very nice change of pace, giving the reader a few hints and tips.

The only problems that I encountered with the GCR editor was the lack of printer support for dumping the data buffers containing the GCR information. One reason for this is quite possible: when dealing with the DOS, a programmer often cannot clearly access the printer due to the havoc caused by some of the DOS routines. So, for the time being, this is something that I would like to see added to future versions as well. But, to date this has only slowed my poking around a little bit and I am still enjoying myself.

Side two of the SK disk provides even more innovative features. A Super Nibbler, a Super Disk Surgeon and the Super Scan.

The Super Nibbler is a very fast nibbler copier for single drives only, that does not support any parameter changes except track ranges for copies. It is fully automatic and will detect most copy protection schemes. This program is used to create backups of SK that will only copy to one generation. In other words, the copies will not copy. The manual mentions that backing up your original SK disk is recommended not only to protect the original but also to "customize" the program to best work on your particular drive. This program was tested on what I consider to be the hardest to backup program that I own: Diskmaker V3.3. It would copy the program only if I skipped track 6 which I call a "Wall" track. A "Wall" track is one that has never been formatted, thus no sync. A drive attempting to read such a track will "hang". "Hanging" means that the drive will spin continuously and freeze up on that track.

However, the program created a backup of Diskmaker in under a minute that worked just like the original. Except for the parameter change above, I did no other editing. The program automatically detected the disk protection and acted accordingly, producing a working copy in no time at all. I would try this one first on all your programs before trying the others. It works rather well.

The Sector Surgeon is the doctor to see if your disk backups still will not work properly. Although it isn't the cure for everything, this little program contains 270 parameters for copying a good amount of today's most popular programs. Perhaps I should correct myself and say "large" program. But no matter what the size, the program does do a wonderful job.

The Surgeon even approaches disk parameter copying in a new and innovative method that, by far, surpasses all other programs around. This is accomplished after the user selects the parameter he wishes to use. After selection, the program turns itself into a fast disk copier (single drive only), therefore removing the need to load another program to copy the disk. During the copy process, the parameter copier will use the parameter provided to make a working backup of the program.

The copies made with Sector Surgeon may often run better than the originals, and will often run on drives they did not run on before. This program is designed to remove those nasty head rattling protection schemes that so many of us have learned to hate. Also, once again, the copies are "customized" to run better on your particular drive.

I readily approve of the Sector Surgeon doing all my delicate disk surgery and hope to see a dual drive version soon. This program also comes with the largest amount of parameters to date, and is yet to be beaten in ease of use.

The last program on side two, Super Scan, provides users a method of scanning disks for DOS errors and standard or non-standard densities. This can aid in making backups of most

software using the dual or single drive nibblers. In other words, you can set some program's parameters according to the information you receive from a disk scan.

This is a utility that is most important in exploring disks, and has yet to be bested by another vendor (although Disector does come close). The format of this program's pictorial graph is a form of BAM which shows all the disk tracks and how they look. It helped me find out the exact tracks I needed to copy for duplicating Diskmaker V3.3. The "Wall" track (Track 6) appeared as a pi sign that was not documented in the manual. Perhaps it is something that might possibly change in its appearance. My advice to users is that if you see anything other than the standard 1, 2, 3, 4, then it's most likely a "Wall" track, so skip it when copying or the drive will "hang."

This utility was truly marvelous except for its lack of printed output (Note: Disector would also not print out its reports for errors and density), and the inability to set a range of tracks to test. My advice is this: When scanning a disk, do a density scan first because if you find a "Wall" track on the disk, doing an error scan may cause the drive to "hang" in the error checking mode of operation (density scan is not affected and I have no idea why). The information provided is not too lengthy in detail, but printed outputs for reference would be a nice addition. However, currently the lack of printer output of density and errors is only annoying; future updates will fix this problem. Note: the display that SK uses during a scan is easier to record than Disector's and is also shorter. Also, the addition of selectable scanning ranges has been promised in a future upgrade. I was told that version 1.2 would also correct many of the scanner's problems.

A final part of SK's arsenal is the Autobooter and Super DOS disk fast load modules, which are the fastest around. These utilities are placed on the SK disk to provide users a means of getting their own programs to fast load in various methods.

The Autobooter allows a user to create the appropriate fast loader for his program with a selection of which Super DOS to use. The Super DOS programs offer much support of various modes of operation having to do with selecting whether or not to have the screen on or off as well as working around the computer's interrupts. The Super DOS modules, at their best, are able to load 150 blocks of data in about 10 seconds. One note here is that Super DOS formatted files are still standard DOS format compatible; they are made differently so that Super DOS can load them faster (see above). The manual advises file copying the original files in Super DOS mode to get the best speed out of the fast boot.

Super Kit/1541 is the newest kid on the block of disk copiers, and is to become the standard that the others will seek to follow. On a cost/feature basis, I highly recommend this program because of its very visible and innovative techniques used. At the cost or less of other copiers, you can get a whole lot more in Super Kit/1541. This program is awarded a rank of a perfect ten, because as the advertisement truthfully states Super Kit/1541, "Has it all!"

Conclusions

The programs reviewed here follow two basic characteristics that make them important; cost and the number of features. Out of the group of six, the programs can be divided into two groups; the ones to buy and the ones not to buy.

The first three programs reviewed, Diskmaker, Keymaster and Copy II fall into that last category. These programs do not provide enough to sustain their high costs. An examination of the comparison chart will show you what is meant by this. I truly feel that these products should cost less than they do until more additions are made to their arsenals.

The following programs, Disector, Fasthack'em and Super Kit/1541 should be purchased for a variety of reasons.

Disector's machine language monitor is very well put together and can be an invaluable aid to the programmer. A person in need of such a utility should run out and get Disector if you are desperate for such a monitor. However, pretty much of the rest of Disector proves to be a disappointment (Disector V2.0 was a whole step forward - V3.0 was a half step), especially the file copier which is said to be "Ultra Fast" on the program's packaging. Unfortunately, this is an outright mistake on their part.

The program Fasthack'em is a wonderful piece of software from an author who could, and should have offered more with his package. The program is invaluable to us MSD fans and C128 owners (as it supports the C128 in C128 mode). Fasthack'em is a reliable product that I would advise purchasing if I were an MSD or C128 owner, but as a C64 and 1541 owner, it does not yet support you as fully as it should.

The program Super Kit/1541 is truly a light in the dark, and is currently unrivaled in the cost/feature area and is the one that I recommend investing in as a user's first disk copy utility. Being the first product from a new company, Super Kit/1541 is truly amazing, and I'm sure we will see more from Prism in the months and years to come as they provide more and more C64/128 support. Watch the software waves as this is the one that the others are going to start trying to beat! Currently, at \$29.95, no one surpasses it.

Suggested Readings

I would suggest that readers interested in learning more about the DOS and disk protection methods pick up the following books:

- 1) Inside Commodore DOS by Richard Immers and Gerald Neufeld
- 2) 1541 User's Guide by Gerald Neufeld
- 3) Program Protection Manual I by CSM Software
- 4) Program Protection Manual II by CSM Software

Benchmark Table

Normal Copy - Standard DOS Disk 0 Blocks Free.
Protected Copy - Attempted Backup of DiskMaker V3.3.
File Copy - 4 Standard Files each 150 Blocks long.

Equipment Used

Single Drive - Commodore 64 and 1541
Dual Drive - Commodore 128¹ and two 1571's
MSD Drive - Commodore 64 and two MSD SD-2's²

Footnotes: 1 - In Commodore 64 Mode
2 - FastHack'em Only

FastHack'em MSD SD-2 Benchmarks

Program Name	Normal	Protected
Standard Nibbler	01/04	01/00
Deep Nibbler	01/09	01/06

Note: See Benchmark Table for Disks used for testing.

Time is Minutes/Seconds

Disk Copier Publisher Information

DiskMaker V3.3

Basix
3463 State Street Suite 1541L
Santa Barbara, CA 93105
Orders : (805) 687-1541 ext. 34
Tech : (805) 682-4000 ext. 33

Disector V3.0

Starpoint Software
122 S. Broadway
Yreka, CA 96097
(916) 842-6183

Keymaster

MegaSoft, LTD.
PO. Box 1080
Battleground, WA 98604
(800) 541-1541
(206) 687-5205

Fasthack'em

Basement Boys Software
PO. Box 30901
Portland, OR 97230-0901
(503) 761-1114

Copy II 64/128

Central Point Software
9700 SW Capital Hwy #100
Portland, OR 97219
(503) 244-5782

Super Kit/1541

Prism Software
401 Lake Air Drive Suite D
Waco, TX 76710
Orders : (817) 757-4031
Tech : (817) 751-0200

Disk Copier Comparison Chart

Name	DiskMaker	Keymaster	Copy II 64/128	Disector	Fasthack'em	Super Kit/1541
Version	3.3	1.0	2.6	3.0	3.0A	1.1
Publisher	Basix	MegaSoft	Central Point	Starpoint	Basement Boys	Prism Software
Level (1)	N	N	N	N and E	N	N and E
Docs (2)	1	3	6	8	7	10
Easy to Use?	Yes	Yes	Yes	Yes	Yes	Yes
# of Disk Swaps(3)	3 to 5	3 to 5	3 to 4	3 to 5	3 to 5	3 to 4
Backs up self?	No	No	Yes	Yes	Yes	Yes (C)
2 drive option(s)?	No	Yes	Yes	Yes	Yes	Yes
Works with MSD?	No	No	No	Yes (6)	Yes	No
Price	\$49.95	\$29.95	\$39.95	\$39.95	\$29.95	\$29.95
Backup Cost	\$15.00	No	backupable	backupable	backupable	Backupable
Upgrade Cost	\$24.95	\$10.00	\$20.00	\$5.00	\$12.00	\$10.00
Speed	(1 drive - 1541)(4) V=Verify VF=Verify Off See Benchmark Table for Disks/Files Used for Testing					
Normal Copy	01/45	01/24	02/13	01/31	V02/31 VF01/46	V02/13 VF01/37
Protected Copy	03/40	01/22	01/59	01/41	V02/09 VF01/39	V00/59 VF00/45
File Copy	No	02/30	No	16/00	02/33	Nrm:02/17 SD:01/52
Speed	(2 drive - 2 1571's)(4) See FASTHACK'EM MSD Benchmark Table for MSD Information					
Normal Copy	No	02/03	02/43	00/59	V00/57 VF00/45	V01/14 VF00/46
Protected Copy	No	01/57	03/40	Failed	02/18	00/26
File Copy	No	Failed	No	16/00	No	Nrm:02/14 SD:01/47
Utilities						
Sector Editor?	No	No	No	Yes	No	Yes
GCR Editor?	No	No	No	No	No	Yes
Fast Format?	No	10 Sec	15 Sec	8 Sec	10 Sec	10 Sec
Fast File Copier?	No	Yes(8)	No	No	Yes (9)	Yes (A)
Parameters?	No	Yes 50	Yes 200 (7)	Yes 40	Yes 100	Yes 270
Fast Loader? (5)	No	No	Yes	No	Yes 1541/MSD	Yes (B)

KEY: 1 - Novice/Expert 5 - Fast Boot for BASIC 9 - 100 Blocks/20 Seconds
 2 - Scale 1 to 10 6 - Not All Features A - 150 Blocks/23 Seconds
 3 - Single Drive Only 7 - Not on Disk B - 150 Blocks/10 Seconds
 4 - Minutes/Seconds 8 - One File at a Time C - Copy Protected

NOTE: Programs that failed did not work to specs or would not duplicate the Benchmark Disk.

Super Kit/1541 NRM = Normal Fast DOS SD = Super Dos Mode for File Copiers.

Copyright 1986 By David W. Martin

WHO DO YOU TRUST?

by The Disk Orderly

This is mainly a tale of anguish and woe — describing twelve hours in my life that I'd rather not relive. From the frustrations experienced a few weeks ago, I ought to be sent to the funny farm. The pseudonym is to protect the guilty — I'd rather not be known as the perpetrator/victim of these misadventures.

Oh, about the name. . .

If Jim Butterfield is the guru, and Dick Immers/Gerry Neufeld can lay claim to the (deserved) title of being the Disk Doctor(s), then I suppose my qualifications rank me just above the hospital volunteer "candy-stripers". If I could ever reach the exalted heights of "nurse-hood", I would rejoice and probably buy a real computer. Suffice it to say that the term "veteran" could apply to my experience. I've done lots of programming (including machine language), taught BASIC and written a few articles in the past eight years. None of this experience qualified me for what that fateful day brought! To set the scene:

It was a dark and stormy night. . .Oops, apologies to Snoopy for that lapse.

One day I purchased an Accolade game, since the SUPERKIT/1541 package would enable me to obtain a backup. I once proclaimed that I'd never patronize software publishers who abuse legitimate owners as well as thieves, but that noble principle has been abandoned, albeit reluctantly.

I raced home with the SUPERKIT like a kid with a new toy. I should have known what was in store for me when I was unable to make a backup copy of the second side of the disk. Reset, reload, recopy. Futility, through three full cycles. Now try the SUPERKIT original to make a game backup. No luck here either, as the three hour mark came and went.

Drag out the 1541 alignment utility from Transactor Volume6, Issue2; the light doesn't blink at all while confirming my alignment is dead centre. On to Plan B. . .

Hour Four

So I call up my friendly dealer and ask him if he has a well-aligned drive in the store. A 1571 running in 1541 mode will do the trick. I hustle down there with a few blanks and make copies of both the SUPERKIT and the Accolade game. The copies run perfectly on his equipment. Now, I dash home and

hope that maybe the copies made there will run at my place. My second hope is that those same copies will allow me to make further copies on my own equipment. My third hope is that my dinner will still be edible (some say I haven't missed a meal in my life). Well, it was one. . .two. . .three strikes I'm out. As hour four comes to a close, I'm no further ahead and losing whatever patience I had at the outset.

Hour Five

On doing some deep thinking, I recalled running a speed test from an American copier program. It reported some months ago that I was running 309-310 r.p.m. If alignment wasn't the bugbear, then maybe speed was killing me. After all, I reasoned, the SUPERKIT must be using its own carefully-timed routines to re-create all those fancy sectors from protected disks.

So I located a drive speed test program from RUN magazine (July 1985), typed it in and found my drive to be precisely on-speed. The same issue told me how to disassemble my drive for alignment/speed adjustments, but I couldn't picture myself performing such a drastic action. Yet.

Who do I believe now? The article did hint that if the drive was way off speed, then the program might erroneously report all was well. Do I hang my hat on that?

Hours Six and Seven

These were totally wasted, as I pretended that the elapsed time may have healed all the wounds. I repeatedly fired up the computer, tried to copy both disks again, loaded the copied programs and watched the computer crash. Time after time. With the same agonizing results.

Hours Eight and Nine

It was time to call for help. The chap who had shown me the other drive speed program lives just across town, and a nine p.m. phone call would not threaten our friendship. He and his 64 welcomed me. Now I had three sets of programs (both SUPERKIT and game): originals, copies made on my drive of each (none worked), and copies made on the dealer's equipment. One by one, minute by (passing) minute, they were fed to my friend's equipment. All failed. Even a full attempt to copy

everything on his system was a dismal failure. Then I loaded the original game disk, and it didn't even run properly! So much for the scientific method—if I'd tried that one first it would have saved two hours.

When we located his drive speed program, lo and behold, my friend's drive clocked in at 311 r.p.m. According to that copier program, both our drives were out-of-whack. (His alignment was also fine). Now I was firmly convinced that speed was the problem. After all, I knew of no serious complaints with the SUPERKIT package. Maybe the RUN program WAS wrong in its OK status report.

Hour Ten

So I go home, with disks spilling from every pocket. Sure enough the borrowed speed test program gave me the 310 r.p.m. verdict, as it had reported a long time ago. With that RUN article in hand, I swallowed hard (three times) and started to disassemble my drive. Me, the mechanical klutz, actually taking my drive apart! Friends say, "You did what?", when I relate this part of the saga.

With the drive upside down, and the chassis resting in its cover, I notice that there's strobe markings for 50 Hz and 60 Hz. When the drive is set spinning, the strobe markings are completely stationary. (I woke up my wife after midnight to help me find an old fluorescent desk lamp, so that there'd be no mistake). The markings didn't budge, not a millimeter per minute. Nothing.

OK, so who do I believe now? The RUN speed test and the strobe marks confirm 300 r.p.m. speed. I'm sure Ontario Hydro knows enough to deliver 60Hz electricity! The other drive speed program says 310 r.p.m. And the SUPERKIT won't make copies. What's going on here?

Hour Eleven

Taking a screwdriver in hand, I nudge the adjustment screw a little, until the copier drive speed program reports it's now 300 r.p.m. Meanwhile the strobe marks start spinning around like the label on a record. With the drive upside down, I proceeded to make my backup of the SUPERKIT and then the game.

Crossing all my fingers and toes, I reload the clones and try them out. Miracle of miracles, everything works! But what about all the stuff recorded at the "old" speed? Having a hundred disks that won't work now, just to have a game and a disk utility, is not a fair trade-off.

I decide that an intermediate speed, of say 305 r.p.m., according to the copier speed test, might let me work with both old and new disks. After setting this speed, I again make successful clones of everything. A few other commercial disks seem to load OK, though the testing is superficial.

But what will happen when I flip the drive over, and seal the case?

Hour Twelve

Gulping even more, I dust off the RUN instructions on how to drill a hole in the 1541's case. This makes the drive speed adjustment screw accessible (once you have re-installed about a thousand tiny screws). A 5/8-inch diameter hole in the bottom of the case. Now my friends KNOW that I've totally flipped.

Now if anything goes awry, I can crank the speed back to what it was. I'm in the home stretch, as I put the chassis back into the case, turn it over and try everything over again: speed (both programs), clone the game and the SUPERKIT again, and run a few other disk programs. The RUN program now tells me that my speed is off by 3 ms, whatever that means. At least I have a calibration mark for when this all stops working for me!

You should be able to determine by now that it is past two a.m., with Tuesday morning's alarm looming. After this marathon, I'm too wired to sleep, so I play a few rounds with my game clone.

But what a price!

Conclusion

Sometimes persistence pays off. My scientific training helped me to accomplish my goal. How many of you would have trashed both disks earlier in the adventure? Was it worth the effort? Probably not.

I blame Accolade for one. Their "warranty" is a postcard, which offers a \$3 rebate on another Accolade purchase if you answer their market research questions. By now we've all seen the "non-warranties" on software that don't even promise that the disk will load. Accolade sinks to new depths—they don't even mention that there is anything in the package that you just bought! Maybe that postcard is what I paid \$40 for?

These questions disturb me:

How is it that two separate speed test programs, both reputable, give wildly different readings? Which do I believe?

Why did I have to disregard the accurate strobe markings, and the RUN speed test results? In adjusting my drive speed to the copy program's version of 300 r.p.m., then the SUPERKIT began to work. Why?

If anyone can help to explain what has happened here, I'd love to hear about it. Right now, I find it hard to know who to trust.

THE CLOCK for Commodore 64

Donald P. Maple
Calgary, Alberta

*It is very easy to become oblivious to all worldly things
while deeply involved in some profound programming task. . .*

This handy little program will make sure that you will never again miss anything because you got carried away while working on your computer. It will enable you to display two time zones and set audio and visual alarms for both!

How many times have you missed your favourite TV show, been late for a date . . . etc, all because of your computer? It is very easy to become oblivious to all worldly things while deeply involved in some profound programming task on your 64, such as solving the ultimate question of life, the universe and everything. (The answer to which, of course, is 42!) Well, despair no more because the solution to all that is nigh! The following program will offer not only one, but two clocks, totally independent and both with alarm features. These can be used to keep track of two time zones or one for the time and the other as a stopwatch or anything else you can think of. . . Further more, they will not interfere with anything you happen to be running at the time! All this interest free, and no money down! If you can't wait any longer and/or are not really interested in the technical nitty-gritty of the program itself read on and the next section will reveal all the "HOW TOs"

Shake Before Use

First and foremost you need peace of mind! Having achieved it, type in the BASIC loader using the Verifier. Having completed the typing save the program first, and then run it. The screen shows the range of memory locations into which the machine language program is being loaded. The counter in reverse video will constantly change to reflect the memory locations stuffed as well as to provide a visual clue that the program is running. The BASIC loader contains the checksum to assure that the data loaded is correct. If everything has gone well you can issue the SYS 12*4096 in line 60, otherwise the innocuous "CHECKSUM ERROR" will be displayed to indicate it is time to get agitated all over again, with optional 4-letter words. . .

Assuming that everything has gone well, or you have regained your composure, you will now see two clocks in reverse video on the topmost line of the screen. The left one in white, affectionately known as CLOCK 1, and the right one in black, nicknamed surprisingly enough, CLOCK 2! Note that the clocks are not running. To get them going use the function keys

described below. They will start up only after the time has been set. CLOCK 1 is controlled by holding the CTRL key in addition to the function keys and CLOCK 2 is controlled by holding the Commodore key in addition to the function keys. Or, if you wish, the F-Keys tell the computer what to do, and the CTRL or Commodore keys tell it who to do it to. This way the normal F1-F8 are undisturbed.

Hold CTRL for Clock 1, C= for Clock 2.

F1/2 – will toggle the display on and off. Sometimes the actual display of one or both clocks is not desirable. It may get in the way of whatever is listed on the screen. Pressing the key will disable the display, then pressing the key again will bring it back. The clock itself is unaffected by this and continues ticking whether the display is on or off.

F3/4 – will toggle the alarm on and off. The visual feedback for the alarm is the asterisk following the AM/PM flag. If the asterisk is present the alarm is on, if it's absent the alarm is off. Again if the display of the clock is off the alarm is unaffected and will wake you up as expected.

When the alarm is triggered the display color of the clock rapidly changes and there is also an audible sound. The sound will be heard even if the computer happens to be playing a song! When the display is off, the alarm sound will still be audible but there will be no rapidly changing colors. The reason for not awakening the display at this time is to prevent the destruction of whatever is in the appropriate corner on the screen. After all if the display was turned off it must have been for a reason!

Once activated the alarm will go on for a couple of forever, unless turned off by pressing the same F3/4. At this time the display (if visible) returns to the default color, the buzzing sound stops and the asterisk disappears.

F5/6 – sets the time

F7/8 – sets the alarm

When these keys are pressed a snapshot of the current time or alarm setting is displayed, using the default color, in the following format:

TIME : 021055P
ALARM : 030000P

The cursor is on the first digit and the time/alarm can be set by overtyping. To make the program as short as possible there is no error checking and anything typed in a format other than the default snapshot will set the timers to unusual settings. However, this is easily remedied by resetting.

Having set the time/alarm, type "A" for AM or "P" for PM. Finally before pressing RETURN the color of display can be changed in the same fashion as in BASIC, that is by pressing CTRL or Commodore key and numbers 1-8. The color can actually be changed any time before RETURN is pressed.

When setting the alarm, the alarm flag (asterisk) is turned on automatically. Note also that when the 64 is initially turned on the two clocks are not running. They are started up by setting the time.

THE CLOCK will not only work in direct mode (such as BASIC editor or SUPERMON) but from within most programs as well. The only programs that will disable it are those that change the hardware interrupt vector. Also, while the editor is in the quote mode the setting of both time and alarm, are temporarily disabled.

Unlike the BASIC time functions obtained from the "jiffy-clock" THE CLOCK is unperturbed by I/O activity on the serial bus. The display may freeze temporarily but the clock continues ticking. This is obvious when the display resumes.

During the testing, a couple of idiosyncrasies of the 6526 chip, that contain the clocks, have been observed.

When setting the time/alarm any time beginning with 12 in the hours will cause the chip to flip the AM/PM flag. For example to set 5 minutes past noon (if "120500P" is used), the chip will turn in into 12:05:00 AM! Apparently the clock sets the flag first and then when it sees "12" it advances the flag causing this problem. To circumvent this, in the example above use "000500P" for correct setting. Strangely enough the clock will handle the "12" properly if it arrived at it by itself. For example 11:59:59 AM will turn into 12:00:00 PM and eventually into 01:00:00 PM.

The second "bug" has to do with the alarm. For example if the alarm is set for 03:10:00 PM it will go off as expected when this time is reached. We would now turn the alarm off. However, if the alarm is turned back on it will be triggered again when the time reaches 03:11:00 PM??? The apparent reason for this is in the way the internal clock handles the carry. When the clock rolls 03:10:59 it turns briefly into 03:10:00 before the carry is

applied to minutes, turning 10 into 11. This brief time, however, is enough to trigger the alarm. The same logic applies to hours as well.

If you are unhappy with the audio alarm, the pitch can be changed by poking assorted values in the following locations:

CLOCK 1 - 49785 (\$C279) freq lo
49787 (\$C27B) freq hi
CLOCK 2 - 49786 (\$C27A) freq lo
49788 (\$C27C) freq hi

Having completed the customizing (sound, colours) the whole program can be saved using a monitor:

S "CUSTOM CLOCK",08,C000,C26E

Next time load your own clock using a non-relocating load:

LOAD "CUSTOM CLOCK",8,1

and activate it with:

SYS 12*4096

This command is actually a toggle and it alternates THE CLOCK on and off. When turned off the actual timers keep on ticking preserving the correct time. However, having disabled THE CLOCK program, the alarms will not be checked. Note that before saving your own version, THE CLOCK must be turned off! This is so that when enabled it will set the interrupt correctly.

The Life And Times Of 6526 CIA

CIA, as we all know contrary to unsubstantiated rumors, stands indeed for COMPLEX INTERFACE ADAPTER. Your Commodore 64 has two of these. But what is this adapter and what does it do? For one thing, it lives to communicate and it loves to talk to peripherals in particular. As such it serves as a middle person between the processor and the outside world. Outside world, being a fairly complex machine, could not be handled with only one interface so that's why your 64 has two. One controls the RS-232 (modem for example), certain memory management aspects and the serial bus (to which we connect peripherals such as disk drive, printer, plotter, etc). The other samples the keyboard and everything that hangs off control ports 1 and 2, i.e. joysticks, paddles, light pen, trackballs, Koalapad etc.

In order to chat with all of those, timing is essential to any computer. That's why each CIA has three clocks on board. Two of them are 16 bit timers and the third is a Time Of Day (call me "TOD") clock. While the operating system uses the first two timers extensively it has totally forgotten TOD. And that's where we come in.

Each 6526 CIA has 16 registers used for various purposes. CIA 1 lives at \$DC00 (56320) and CIA 2 at \$DD00 (56576). For this program we are only interested in 6 of those registers.

- \$08 (8) – TOD 10ths of seconds
- \$09 (9) – TOD seconds
- \$0A (10) – TOD minutes
- \$0B (11) – TOD hours
- \$0D (13) – Interrupt control register
- \$0F (15) – Control register B

To obtain the real addresses of the registers simply add the offset shown above to the CIA start address. For example to get the hours register for CIA 1, add \$0B (11) to \$DC00 (56320).

An interesting note here. The designers of the 64 have decided not to decode all of the address lines pointing to these registers. This leads to the occurrence of so called "ghosting". What that means in practical terms is that the 16 registers, in this case actually occupy 256 locations. In other words they are repeated 16 times. So, for example, \$DC00 will also appear at \$DC10 and \$DC20 and \$DC30 . . . etc!

The four TOD registers actually access eight internal registers. In other words the same four addresses point to both time and alarm functions. Which one of those functions is accessed depends on bit seven of Control Register B. If this bit contains a zero, writing to TOD registers will set the time, but if this bit contains a one, writing to TOD registers will set the alarm.

- bit 7 of \$0F (15) = 0 – writing to \$08–\$0B (8–11) sets time
- bit 7 of \$0F (15) = 1 – writing to \$08–\$0B (8–11) sets alarm

Note that the state of bit 7 of Control Register B is of importance only when writing to TOD. When reading from TOD we will always get the time. In other words TOD contains a write only ALARM and read/write TIME registers. Because of this THE CLOCK program saves the alarm information independently.

The order in which the TOD registers are read from or written to also plays an important role. Upon reading hours register at \$0B (11), TOD clock stops. It will resume only after reading register \$08 (8) – 10ths of seconds. This does not apply to minutes and seconds. They can be read without stopping the clock. Actually the clock continues tick-tocking internally but the output registers are latched until the read on 10ths of seconds occurs. The same sequence is necessary when writing to either time or alarm registers. The clock will commence only after a write to 10ths of seconds occurs.

To make the life of programmers easy the designers of this chip have decided to store all of the TOD data in BCD (binary coded decimal) and that is how the data is read or written. On the other hand, to make the life of programmers difficult the same designers have decided to use the hour register at \$0B (11) to store the AM/PM flag! If bit seven of this register is zero it

denotes AM, and if it's one then is PM. To be fair, though, the high nybble of this register is not used for anything else.

Finally there is the Interrupt control register at \$0D (13). It is used to enable and sense the occurrence of alarm that was set as previously described. This register is also connected to two internal registers, a read only interrupt data register and a write only interrupt mask register. When the alarm occurs it will create an interrupt. Since there can be several sources of this interrupt the program monitors bit 2 of this register. If this bit is on then the time in TOD matches the time in the ALARM! And that's all there is to it.

To do all this THE CLOCK modifies the hardware interrupt vector. This interrupt occurs 60 times per second and at that time the processor goes to a predetermined location and executes the "interrupt routine". For 6502/6510 type processors the address of where to go is at \$FFFE–\$FFFF. On Commodore 64 it ends up getting the next address at locations \$0314–\$0315. Without getting into it in more detail THE CLOCK changes this vector to point to THE CLOCK itself. The program, however, politely makes note of where these two locations used to point to and, when done, jumps there to resume regular processing. This means that if some other program has already modified the vector, this utility will maintain it. Normally, however, \$0314–\$0315 will point to \$EA31.

Conclusion

This article and the accompanying program have only scratched the surface of this, indeed, complex interface chip. If you have any questions or suggestions you can contact me either through this magazine or directly at the address below.

Donald P. Maple
P.O.Box 23, Station M
Calgary T2P 2G9
CANADA

Editor's Note:

We asked Don for a copy of the source code for this program but apparently he used (believe it or not) the simple assembler in Supermon to write it, and unfortunately, time did not permit us to disassemble and comment it for you.

THE CLOCK: BASIC Loader

EG	10 f=49152:t=49837	LJ	550 data 174, 115, 194, 93, 109, 194, 157, 109
LN	20 print "Spoking from "f" to "t	MI	560 data 194, 96, 160, 147, 44, 160, 152, 165
KP	30 for j=f to t : read x : print "sqr" j	OI	570 data 212, 208, 246, 169, 193, 72, 152, 72
DK	40 poke j,x : ch = ch + x : next	OI	580 data 8, 72, 72, 72, 165, 206, 174, 135
MA	50 if ch<>84866 then print "checksum error" : end	LE	590 data 2, 160, 0, 132, 207, 32, 19, 234
BM	60 rem sys12*4096	OM	600 data 76, 129, 234, 162, 7, 24, 144, 3
EG	70 rem	KA	610 data 162, 15, 56, 173, 134, 2, 72, 8
JG	100 data 120, 174, 20, 3, 172, 21, 3, 173	GL	620 data 172, 115, 194, 185, 113, 194, 141, 134
GO	110 data 135, 192, 141, 20, 3, 173, 136, 192	KK	630 data 2, 169, 8, 133, 251, 133, 253, 169
DM	120 data 141, 21, 3, 142, 135, 192, 140, 136	PK	640 data 220, 24, 109, 115, 194, 133, 252, 133
IM	130 data 192, 88, 96, 162, 0, 181, 251, 72	JG	650 data 254, 160, 7, 177, 251, 40, 144, 3
PJ	140 data 232, 224, 4, 208, 248, 164, 197, 204	EN	660 data 9, 128, 44, 41, 127, 145, 251, 189
IN	150 data 116, 194, 240, 38, 140, 116, 194, 192	IP	670 data 154, 194, 32, 210, 255, 202, 136, 16
GN	160 data 3, 144, 31, 192, 7, 176, 27, 173	HB	680 data 246, 232, 240, 25, 165, 254, 74, 169
PD	170 data 141, 2, 74, 74, 176, 6, 74, 144	FN	690 data 129, 176, 2, 169, 136, 133, 253, 169
GP	180 data 17, 169, 0, 44, 169, 1, 141, 115	NB	700 data 194, 133, 254, 32, 69, 193, 189, 109
MC	190 data 194, 185, 122, 194, 141, 80, 192, 32	IM	710 data 194, 41, 2, 240, 246, 160, 3, 177
LJ	200 data 69, 193, 169, 8, 133, 251, 169, 220	KC	720 data 253, 16, 5, 162, 80, 41, 127, 44
JN	210 data 133, 252, 169, 0, 133, 253, 169, 212	AA	730 data 162, 65, 44, 177, 253, 72, 74, 74
KP	220 data 133, 254, 162, 12, 172, 111, 194, 173	GB	740 data 74, 74, 9, 48, 32, 210, 255, 104
MC	230 data 109, 194, 32, 137, 192, 230, 252, 169	OA	750 data 41, 15, 9, 48, 32, 210, 255, 136
BE	240 data 7, 133, 253, 162, 39, 172, 112, 194	BJ	760 data 208, 233, 138, 32, 210, 255, 160, 7
LM	250 data 173, 110, 194, 32, 137, 192, 162, 3	PC	770 data 169, 157, 32, 210, 255, 136, 208, 250
GJ	260 data 104, 149, 251, 202, 16, 250, 76, 27	LP	780 data 160, 3, 32, 207, 255, 10, 10, 10
EI	270 data 192, 74, 144, 82, 72, 74, 169, 160	EK	790 data 10, 153, 170, 194, 32, 207, 255, 41
KA	280 data 144, 2, 169, 170, 141, 153, 194, 152	FC	800 data 15, 25, 170, 194, 153, 170, 194, 136
AE	290 data 72, 138, 72, 160, 3, 177, 251, 16	IF	810 data 208, 232, 32, 207, 255, 74, 160, 3
NI	300 data 3, 169, 144, 44, 169, 129, 141, 151	IN	820 data 185, 170, 194, 144, 4, 41, 127, 176
JF	310 data 194, 162, 0, 177, 251, 41, 127, 74	LF	830 data 7, 9, 128, 144, 3, 185, 170, 194
HL	320 data 74, 74, 74, 9, 176, 157, 142, 194	FH	840 data 145, 251, 145, 253, 136, 16, 246, 174
GJ	330 data 177, 251, 41, 15, 9, 176, 232, 157	PG	850 data 115, 194, 173, 134, 2, 157, 111, 194
JI	340 data 142, 194, 232, 232, 136, 208, 228, 177	BI	860 data 157, 113, 194, 104, 141, 134, 2, 169
HI	350 data 251, 160, 12, 104, 170, 185, 141, 194	FC	870 data 13, 32, 210, 255, 96, 1, 1, 1
KP	360 data 157, 0, 4, 104, 72, 157, 0, 216	FN	880 data 0, 1, 0, 0, 64, 0, 0, 0
HJ	370 data 202, 136, 16, 241, 104, 104, 74, 144	DP	890 data 0, 0, 0, 10, 5, 117, 43, 69
BM	380 data 73, 165, 252, 74, 162, 0, 144, 1	NF	900 data 114, 0, 0, 0, 8, 0, 0, 0
EM	390 data 232, 189, 109, 194, 10, 176, 22, 160	GN	910 data 0, 0, 0, 145, 83, 160, 176, 176
DL	400 data 5, 177, 251, 41, 4, 240, 51, 189	PB	920 data 186, 176, 183, 186, 178, 183, 160, 129
HM	410 data 109, 194, 9, 128, 157, 109, 194, 160	NC	930 data 141, 170, 58, 69, 77, 73, 84, 32
KN	420 data 4, 169, 0, 145, 253, 160, 0, 189	JA	940 data 32, 13, 58, 77, 82, 65, 76, 65
DP	430 data 121, 194, 145, 253, 200, 189, 123, 194	MO	950 data 32, 13, 0, 0, 0, 8
AE	440 data 145, 253, 160, 4, 169, 33, 145, 253		
CO	450 data 200, 169, 15, 145, 253, 200, 169, 240		
GP	460 data 145, 253, 169, 15, 141, 24, 212, 254		
HE	470 data 111, 194, 96, 169, 1, 32, 104, 193		
CI	480 data 74, 176, 17, 165, 209, 72, 165, 210		
AB	490 data 72, 162, 0, 32, 255, 233, 104, 133		
FD	500 data 210, 104, 133, 209, 96, 169, 2, 32		
ME	510 data 104, 193, 74, 74, 176, 25, 138, 74		
LI	520 data 160, 4, 144, 2, 160, 11, 153, 0		
CF	530 data 212, 189, 109, 194, 41, 127, 157, 109		
KG	540 data 194, 189, 113, 194, 157, 111, 194, 96		

VIC-II Chip Interrupts

Tomas Hrbek
Rocky Point, New York

This month's issue is about programming the Commodore chips. The most known and programmed chip by the average user is the SID chip, controlling sound and a few other things such as game paddles. Then there are the two CIA chips used for the housekeeping functions of the C-64. They are the timers for data transfers between the computer and different devices. They read joysticks, determine which key on the keyboard was hit and also generate the IRQ and NMI interrupts. There is also the VIC-II chip, which manages the Commodore 64 architecture. It switches between the four 16k banks, displays different kinds of modes, as for example the text, hi-res and multicolor mode, controls sprites and other features. In this article I would like to discuss the lesser known features of the VIC-II chip.

As the title of this article suggests, the VIC-II chip is capable of generating its own interrupts. These interrupts are of four different natures. They can be generated by a lightpen, sprite to sprite or sprite to background collisions, or by a specific raster line being reached. The last interrupt mentioned is the least understood by the average programmer, and is the one that I will discuss further.

You might be wondering what this raster line interrupt might be and if you have ever seen it. You probably have, even though you might not know about it. If you own the popular word processor SpeedScript, the raster line interrupt generates the top message line. Other programs that also have this feature are PaperClip, KMMM Pascal, Cad-3D and others. Here, the raster line capabilities are used to highlight a single line, but you will see that raster line interrupts are much more powerful than that.

To gain an understanding of this interrupt, we have to begin by examining how the computer displays information on the screen. Attached to the back of your monitor's picture tube is an electron gun that is used to generate an intense and accurate flow of electrons towards the phosphor coated surface of the inside of the picture tube. Monochrome monitors use one type of phosphor that allows graduations of intensity between white and black (on and off), to provide an illusion of color and shading. Colour monitors use three types of phosphors throughout to produce three colours; Red, Green and Blue (RGB). These phosphors are grouped accurately and very closely together to produce all possible colour combinations depending on the combination of RGB chosen for display. Phosphor, for those who are not sure just yet, is a substance that will produce light when struck by an electron particle.

In order for the picture on the screen to be of any quality, the electron gun controlled by the computer has to generate over 250 visible lines and refresh them 20 times per second. Each line is also divided into about 180 visible points. Because of the speed with which each point has to be refreshed, the two VIC-II registers that tell us the position of the electron gun, contain only the number of the raster line, not each individual point.

As mentioned in the previous paragraph, there are about 250 raster lines displayed on the screen. However, the topmost visible line is raster line number 30, which is in the border area of the screen. The topmost line of the text area is raster line 40. The same is true for the bottom. The text area ends at raster line 240 (200 screen lines, just like in high resolution mode), and the border ends at about 280, although the electron beam continues on a little past that. This is why sprites can disappear behind the screen edges.

The two registers that contain the value of the raster line are located at \$D012 and \$D011. Bit 7 of \$D011 contains the overflow from \$D012. That is because a memory location can only hold a value from 0 to 255. If a value is written to these registers, we instruct the VIC-II chip to generate an interrupt at that location. Before the interrupt can occur, however, we have to do some preparatory steps.

To start, we have to tell the VIC-II that we are enabling one or more of its interrupts. This is done by setting some bits in the Interrupt Mask Register (IMR), location \$D01A. Bit one tells the VIC-II that we want to enable the raster line interrupt; bit two, sprite to sprite collision interrupt; bit three, sprite to background collision interrupt; and bit four, lightpen interrupt. The next three bits are meaningless. Bit seven is used to indicate that any one or more of the previous bits has been set.

Now that we know how to enable the raster line interrupt, we have to adjust the IRQ routine so that we will be able to take advantage of the interrupt generated by the VIC-II. Whenever any chip on the C-64 generates an interrupt, it jumps through a vector at location \$314-\$315, which points to the address \$EA31. However, this vector can be easily changed to point to our own custom routine.

In our IRQ routine, we have to check if this interrupt has been generated by the raster line interrupt. This is done by reading the Interrupt Raster Register (IRR) located at \$D019. This register has the same configuration as the IMR register. It, however, indicates by setting the appropriate bits which device generated the interrupt. Since we only enabled the raster line interrupt, it is sufficient to check bit seven, which indicates that the VIC-II generated this interrupt. Thus, if this bit is not set, we simply proceed with the usual interrupt routine. However, before we do that we have to clear the CIA1 IRR by reading register \$DC0D.

Now that we know that the routine was generated by the VIC-II, we can proceed with our IRQ routine. Before doing that, we have to clear the VIC-II IRR register, or it will generate another interrupt as soon as we exit the IRQ handling routine. This is done by writing the value we read from the VIC-II IRR back to the IRR register. Having done all of this, we can finally write our program.

The Program

My example is a very simple one. It highlights a line on top of the text screen just like in SpeedScript. As I mentioned earlier, the raster line is much more powerful. We can display more than eight sprites on the screen at the same time, different kinds of modes, more character sets, etc. We can even divide the screen into more than one part (split screen). Of course, we can divide it only along the horizontal lines.

After doing all of the preliminary steps, we decided that this particular interrupt was generated by the VIC-II chip. We compare the value of LINE2 to that of the RASLIN register, location \$D012. If the value of LINE2 is greater than that of the RASLIN register, we store the value of LINE2 in the RASLIN register, change the screen color and exit. However, if the LINE2's value is equal or smaller, we load the accumulator with the lower value of the highlighted line (LINE1), store it in the RASLIN register, change the screen color and exit.

Here are some hints on how to utilize the raster line interrupt in other ways. If you do not want text to scroll into the highlighted line on top of the test screen, clear bit three of \$D011. This will tell the computer to display 24 instead of 25 lines per screen. Then if you want to display something in the highlighted line, just poke the message there. Poke the first 40 bytes of screen memory usually starting at 1024 or \$400. If you want to display more than eight sprites each time you switch between the two parts of the screen, just redirect the sprite pointers to the appropriate part of the screen. The same thing applies to the hi-res or multicolor screen. Just repeatedly switch in and out of these modes.

To wrap this session up, I must say that raster interrupts are very powerful feature of the Commodore 64 that is unfortunately almost unknown. It is certainly due to the fact that this kind of feature can be only explored and utilized if one is a machine language programmer. I hope that this article made it clearer how to use the VIC-II chip in other ways than the usual ones. I also hope that it will help you in further exploring the C-64 and in writing better programs.

Basic Loader

```

OE 100 rem save "0:raster irq.ldr",8
KF 110 rem ** by tomas hrbek rocky point, new york
MM 120 rem ** raster demo for the c64
OO 130 for j=828 to 903: read x: poke j,x: ch = ch + x: next
NA 140 if ch<>8431 then print "checksum error!": stop
KH 150 sys828
CN 160 data 120,169, 91,141, 20, 3,169, 3
HL 170 data 141, 21, 3,169, 50,141, 18,208
LE 180 data 173, 17,208, 41,127,141, 17,208
EI 190 data 169,129,141, 26,208, 88, 96,173
CP 200 data 25,208,141, 25,208, 48, 7,173
EP 210 data 13,220, 88, 76, 49,234,173, 18
EB 220 data 208,201, 58,176, 10,169, 1,141
FD 230 data 33,208,169, 58, 76,130, 3,169
GM 240 data 0,141, 33,208,169, 50,141, 18
IH 250 data 208, 76,188,254
    
```

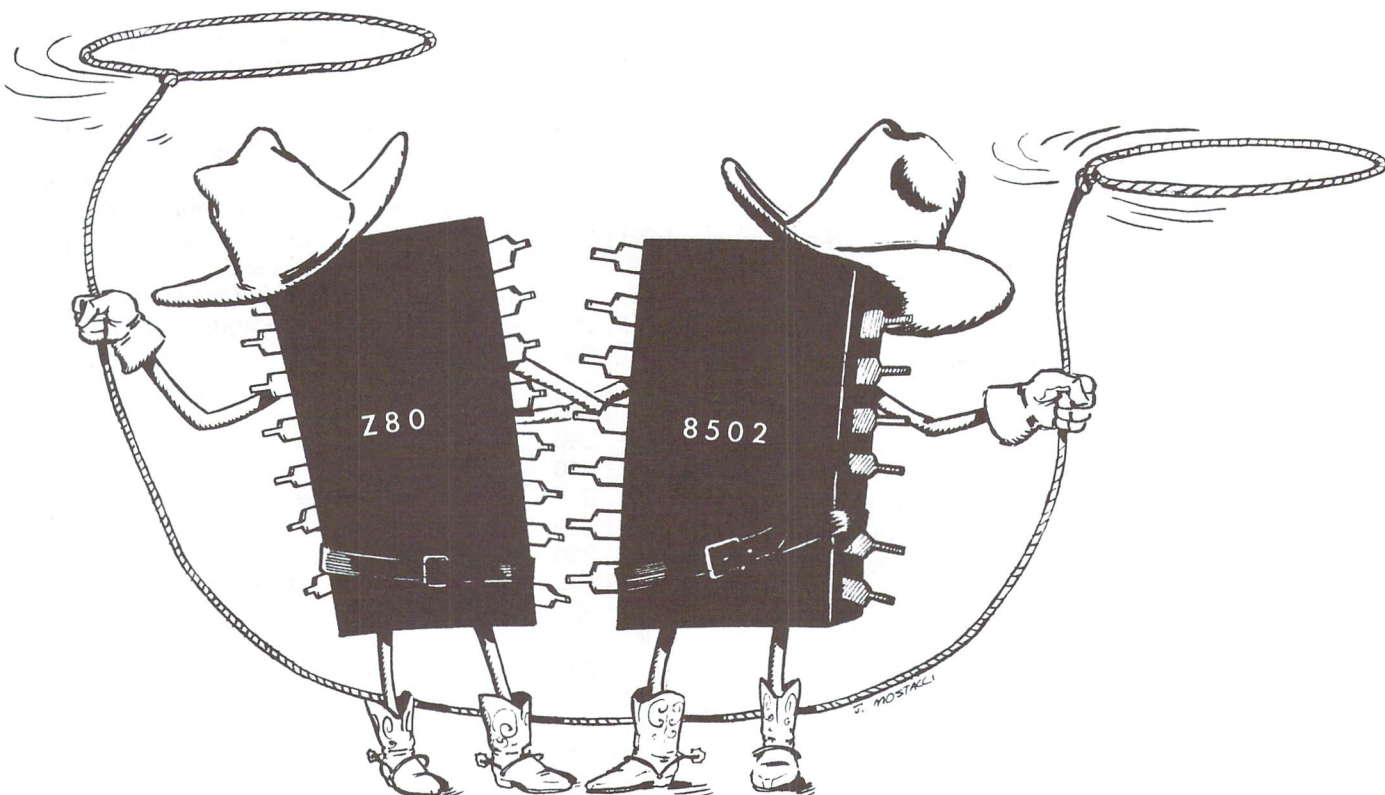
PAL Source Listing

```

PE 100 rem save "0:raster irq.pal",8
PD 110 sys 700
AO 120 .opt oo
PA 130 * = 828 ;cassette buffer
CA 140 ;
GL 150 key = $ea31 ;old irq handling routine
DH 160 cinv = $0314 ;irq vector
NH 170 raslin = $d012 ;raster line
BP 180 irr = $d019 ;flag for video interrupt
EJ 190 xmr = $d01a ;video controller interrupt flag
OK 200 colreg = $d021 ;background color register
PF 210 icr = $dc0d ;flag for timer interrupt
DH 220 line1 = 50 ;start of highlighted line
AF 230 line2 = 58 ;its end
IC 240 color1 = 1 ;raster line color
AO 250 color2 = 0 ;background color
KH 260 ;
OI 270 setup = *
BL 280 sei
LM 290 lda #<irqnew
JO 300 sta cinv
LN 310 lda #>irqnew
JA 320 sta cinv + 1
HC 330 lda #line1
PP 340 sta raslin ;for interrupt
NG 350 lda raslin-1
ID 360 and #$7f ;clear hi bit
PL 370 sta raslin-1
LB 380 lda #%10000001 ;permit irq by raster
FA 390 sta imr
OB 400 cli
GI 410 rts
KB 420 ;
EI 430 irqnew = *
DA 440 lda irr
BL 450 sta irr ;clear interrupt flag
DF 460 bmi vicrastr ;interrupt by raster line
ME 470 ;
DH 480 lda icr ;clear interrupt flag
AM 490 cli ;allow raster interrupt
PI 500 jmp key
EH 510 ;
CH 520 vicrastr = *
PA 530 lda rasin ;read raster line
AE 540 cmp #line2
LE 550 bcs greater ;greater than or equal second value
GK 560 ;
KH 570 lda #color1
KM 580 sta colreg
GO 590 lda #line2 ;next interrupt at 2nd line
DD 600 jmp exit
IN 610 ;
OA 620 greater = *
IL 630 lda #color2
GA 640 sta colreg
OI 650 lda #line1 ;next interrupt
KA 660 ;
AB 670 exit = *
AD 680 sta raslin
CL 690 jmp $fbc ;to wrap things up and exit
CD 700 ;
CK 710 .end
    
```

Switcheroo Pivot

Chris Miller
Kitchener, Ontario



An 8500 & Z80 Working Together In The C128

The C-128 is a two processor system. Inside are an 8500 and a Z80. The 8500 is really just a 2mz. 6510. The Z80 is one of the most advanced 8 bit processors around. The 8500 is a memory based microprocessor. Even just to subtract the contents of one register from another would require some free RAM. Indirect addressing modes and a sparsity of internal registers make the 8500 very reliant on zero page pointers. The Z80 is a register based microprocessor. It has two sets of general purpose registers. Each of these sets contains an accumulator, a status register and six, 8 bit, general purpose registers. The second set can be used for the interrupt flip-flop (IFF) or by the exchange (EXX) command to remember and restore register contents. Data registers can also be paired for 16 bit addressing and arithmetic. In addition to these there are four other 16 bit registers: the PC (program counter), the SP (stack pointer) and the (IX) and (IY) (index) registers.

8-Bit Internal Registers

A	A'	accumulator
B	B'	general purpose
C	C'	
D	D'	
E	E'	
H	H'	
L	L'	
F	F'	flag (status)

16-Bit Register Pairs

BC	B=hi byte C=low byte
DE	D=hi byte E=low byte
HL	H=hi byte L=low byte

True 16-Bit Registers

IX	index
IY	index
SP	stack pointer
PC	program counter

The Z80 has several times as many commands as the 8500; some therefore require more than one byte of opcode. These commands can be functionally divided into 13 groups, each of which is more extensive than its 8500 counterpart where counterparts even exist.

1. The Eight Bit Load Group

The Z80 assembler load instruction, LD, might more aptly be named MOVE. There is no store instruction. Every LD will be followed by two operands delimited by commas. The first operand represents the destination and the second the source, so that the instruction LD(\$C000),A means store the contents of A at \$C000 whereas LDA,(\$C000) would mean load A from \$C000. In Z80 mnemonics, parenthesis define a memory location; otherwise an immediate value is assumed.

2. The Sixteen Bit Load Group

This includes all the commands which move two byte values either between registers or between registers and addresses. Included here are the PUSH and POP instructions which is handy since addresses are what stacks are mainly for.

3. The Exchange Group

Register contents can be swapped with the secondary set or within the primary set. There's nothing like this on the 8500 although we often wish there was.

4. The Block Transfer Group

Set a few register pairs and use one of these to move or fill memory a byte at a time or in a Z80 controlled loop. The short Z80 routine which we will later call from Basic to copy its ROM into 8500 visible RAM uses an LDIR loop.

5. The Block Search Group

As above, the Z80 can automatically control looping by counting down the value contained in the BC pair and incrementing the address pointed to by DE. Ranges of memory are compared with the A register until a match is found or the BC pair decrements to zero.

6. The 8-Bit Arithmetic And Logical Group

These allow for manipulation of one byte values in pretty much the same way 6510 programmers are used to. Addition and subtraction are possible with or without carry.

7. The 16-Bit Arithmetic And Logical Group

Same as above but with two byte values being manipulated. The logical AND, OR and XOR are not found in this group.

8. The CPU Control Group

Processor and interrupt modes and status flags are handled.

9. The Rotate And Shift Group

Many different types of shifts accessing both one and two byte values via a variety of addressing modes are available.

10. The Bit Set Reset And Test Group

These commands provide for complete bit addressing. Each takes two parameters. The first will specify which bit (0-7) is to be set, reset, or tested; the second will designate the register or memory location to be manipulated. For example SET3,(IX+0) would set bit 3 in the address pointed to by the IX register; ie OR it with the number 8.

11. The Jump Group

Conditional and unconditional jumps (direct) and branches (relative) are supported. Anyone who has ever had to fake a conditional jump in 6510 via BNE*+5:JMPFAR or an unconditional branch via SEC:BCS NEAR will appreciate the versatility of this Z80 group.

12. The Call And Return Group

Subroutines may also be called and returned from conditionally or unconditionally.

13. Input Output Group

These are specialized load and store instructions. In the C-128, when accessing I/O memory (D000-DFFF), IN and OUT commands should be used instead of LD.

Programming The Z80 In 128 Mode

```
LD A,$B1
LD ($D505),A
```

The Z80 brings a convenience and conciseness to ML programming that is sure to please and impress 6510 assembly language programmers. I hope the above has whetted your appetite for doing a little exploring. It will inspire you to know that this microprocessor can be used in conjunction with (not at the same time as) the 8500 in the C-128, even from Basic; switching between them is not much more difficult than switching between memory banks once you know how.

Working Together

In order to figure out how the Z80 and 8500 worked together, it was necessary to disassemble some Z80 ROM; in order to access this ROM it was necessary to enter, and get back alive, from Z80 mode—sort of a catch 22 situation. One hundred cups of coffee and about a thousand crashes eventually prevailed. Disassembling the boot sector on the CP/M disk got the ball rolling.

Bit 0 at \$D505 (54533) controls the microprocessor mode. If it is turned on then the 8500 becomes active; if it is off then the Z80 takes over.

You can't just poke it off (believe me). A little housekeeping is first in order:

Disable 8500 interrupts via SEI because you are going to switch to a memory configuration in which Kernal ROM is not visible.

Store a \$3E (62) at \$FF00 (the configuration register). This leaves I/O RAM intact but switches everything else to RAM 0.

You're still not quite ready. The Z80 PC register holds \$FFED after 128 initialization. There is a NOP (\$00) there. The first actual Z80 command goes at \$FFEE. If you look through the monitor you will see a \$CF there. This is an RST8 opcode byte which will cause the Z80 to jump (ReStArt) to its own ROM routine at 0008. You do not want this. After moving some 8500 code into place at \$3000, the Z80 would return control to the 8500. The 8500 wakes up exactly where it left off after you switched to the Z80. If you followed this switch with a NOP (lets not wake it up too fast) and then a JMP \$3000 (like the operating system does) you would go into the 128's boot CP/M routine. This is pretty useless from a programming standpoint, so don't bother. Instead, put your own Z80 code at \$FFEE.

Before you do any Z80 subroutine calls, you should set its stack pointer register (SP) to point to some area that will not interfere with your code or Basic. Switcheroo points it to \$BFFF before calling a user routine. The return address goes into memory at \$BFED in this case.

The last thing the Z80 will have to do is to turn the 8500 back on. There are two ways to do this:

This is inferior. There is a bleed through condition in the Z80 mode using this type of store. A \$B1 will also be written to underlying RAM. (This is where my Z80 cross-assembler sits, making this feature especially bothersome.)

Here is the proper way:

```
LD BC,$D505
LD A,$B1
OUT (C),A
```

Not only does bleed through not occur using OUT storage but I/O memory between \$D000 and \$DFFF can be written to. In our Basic coding sample the background (\$D021) and border (\$D020) are poked via a Z80 OUT. This would not work using LD(\$D020),0 which would put the value in RAM0.

Ordinarily you would have to bear in mind that the Z80 might not necessarily take off at \$FFEE the next time you activated it. It, like the 8500, wakes up where it went to sleep. The best procedure for switching back and forth is to try to always put the microprocessors to sleep in the same spots. These switches could be followed with jump commands. Before invoking them you could set the jump address for the other microprocessor to anywhere you like. Z80 ROM puts a RET (\$C9) command after the 8500 switch allowing the Z80 to CALL the 8500 from anywhere and return when the 8500 switches back. Switcheroo puts an RTS (\$60) after the Z80 switch so that the 8500 can JSR the Z80. This also makes the Switcheroo routine completely relocatable.

Now it just so happens that there are two routines high in RAM 0 through which the two microprocessors can invoke each other. The 8500 invokes the Z80 at \$FFD0. When the Z80 returns control, the 8500 picks up at \$FFDB. Leave the NOP (\$EA). You can take over at \$FFDC (65500).

The Z80 invokes the 8500 at \$FFE1. When the 8500 returns control, the Z80 picks up again at \$FFEE—and so on and so on.

The Switcheroo Pivot

Switcheroo handles the Z80 stack, the user call, and controls the "sleepy time" program counters for the two microprocessors while making use of the RAM routines at \$FFE1 and \$FFD0. The Switcheroo pivot thus allows you to easily execute hybrid programs and, as our example shows, even call the Z80 from Basic.

The Switcheroo code sits at 3000, high in the 128's tape buffer. The address of the Z80 code to be executed should be in the 8500's X (= low byte) and A (= high byte) registers. These can

be passed directly from ML or even Basic via the 128's new improved SYS command, which is exactly what our little Basic example does. The program pokes some Z80 code in at \$6000, calls Switcheroo to execute it and then continues in Basic. The Z80 code copies its ROM into RAM at \$8000. Notice how easy it is to code this move (4 instructions, 11 bytes). The Z80 then pokes the screen colours just to show off.

The Switcheroo pivot isn't long at all, and should pave the way for some serious exploring of the Z80 language and environment in the 128 by 8500 buffs.

Switcheroo Pivot Basic Loader

```

KM 100 rem save "0:switcheroo.bas",8
OI 110 rem ** this program will poke some z80
CC 120 rem ** code into memory at $6000 and then
FD 130 rem ** execute it via the switching
NF 140 rem ** routine at 3000.
KA 150 :
OK 160 rem ** now put z80 code into memory
BK 170 for x=24576 to 24598: read b: poke x,b: next
EA 180 sys 3000,96,0: rem call z80 at $6000
JH 190 print " back alive and well!!!! ": end
MD 200 :
BB 210 rem ** the z80 code to copy 4k rom to $8000
    in ram 0 **
HI 220 data 33, 0, 0 :rem ld hl,0
LN 230 data 17, 0, 128 :rem ld de,$8000
FD 240 data 1, 0, 16 :rem ld bc,$1000
DL 250 data 237, 176 :rem ldir
JP 260 rem ** z80 code to poke background and
    border colour **
LG 270 data 62, 0 :rem ld a,0
GA 280 data 1, 32, 208 :rem ld bc,$d020
HM 290 data 237, 121 :rem out (c),a
EM 300 data 60 :rem inc a
LM 310 data 12 :rem inc c
FO 320 data 237, 121 :rem out (c),a
NC 330 data 201 :rem ret
    
```

Switcheroo Pivot Source Listing

```

DM 100 rem save "0:switcheroo.pal",8
LN 110 open 8,8,1,"0:switcheroo"
JE 120 sys700
FK 130 .opt o8
GO 140 *           = 3000
MA 150 ;
HN 160 z80pc      = $fee      ;z80 wakes up here
FA 170 invokez80 = $ffd0     ;by 8500
AL 180 c128pc     = $fdc      ;wakes up here
BI 190 invoke128 = $fe1      ;by z80
EH 200 z80stk     = $cfff     ;safe place for stack
IE 210 ;
AL 220 :*** 'switcheroo' for hybrid processing ***
GJ 230 :*** by chris miller, may 20, 1986
GG 240 ;
ND 250           sty $ff01     ;set ram 0 config
AB 260           ldy #$31      ;ld sp,($bfff)
BP 270           sty z80pc
FF 280           ldy #<z80stk
MO 290           sty z80pc+1
FG 300           ldy #>z80stk
DA 310           sty z80pc+2
GL 320 ;
CG 330           ldy #$cd      ;call (jsr) opcode
EC 340           sty z80pc+3
NC 350           stx z80pc+4   ;subroutine address
                                (sys parameters)
ON 360           sta z80pc+5
IO 370 ;
NH 380           lda #$c3      ;jp (jmp) opcode
NN 390           sta z80pc+6   ;invoke c128 when
                                can't continue
KL 400           lda #<invoke128
GB 410           sta z80pc+7
KM 420           lda #>invoke128
NC 430           sta z80pc+8
OC 440 ;
AO 450           lda #$60      ;rts opcode
DG 460           sta c128pc
BM 470           jsr invokez80
GF 480 ;
KO 490           lda #0        ;back to basic
MA 500           sta $ff00
MI 510           cli
EP 520           rts
    
```

Program Trace Monitor

Richard Stringer
Dallas, Texas

- A profiler for Commodore 64 machine language programs

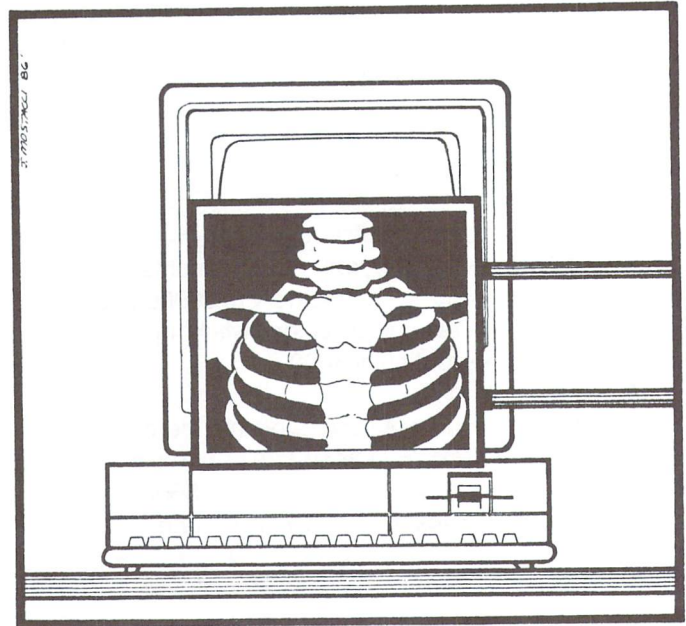
This program will trace the execution of any program and show how often each section of the program is executed. It is primarily designed to be used with machine language programs, but can be used for programs in any language. The output, in the form of a chart, can be used either by another program or the programmer himself to provide information on the execution speed, memory use, location of called subroutines, and general efficiency of the code.

The output of the program is in the form of a chart, which is sent to the printer due to the large amount of data generated. No printer codes are used, so any printer should work. Along the left side of the chart is the high byte of addresses in the traced area, and along the top are the low address bytes, in increments of 16. At the intersection of a high/low address is found a hexadecimal value which indicates how many times code has been executed within that 16-byte area during the trace period. With this layout, you can quickly look up areas of interest in your program to see if a section is being executed, and if so, how frequently.

This program works in an atypical manner for a trace program on 65xx series computers. Since a great deal of programs use or alter the system IRQ interrupt, I found that I could not insert the trace vector into this interrupt and still get acceptable performance. Another more serious difficulty was the frequency of the IRQ interrupt. On the Commodore 64, the IRQ frequency is set at 60 Hertz (every 1/60 seconds). This is far too slow to trace an assembly language program. Also, many programs disable the system interrupt with a SEI instruction, which would shut off the trace. To avoid these problems, the program uses the NMI (Non-Maskable Interrupt) as its interrupt source.

The NMIs can occur at a user-selected rate, allowing you to determine how many times per second the program makes a count. You can select frequencies from about 9800 per second down to less than 50. The frequency is varied by placing a value in location 828 decimal. (A value of zero will default to one.) This value is used as the upper 8 bits of a timer which generates an NMI whenever it counts down to zero. The lower 8 bits is supplied by the variable CCYCLE in the source listing, which is normally set to 100. The 16-bit timer value thus formed indicates the number of cycles between interrupts - on the C64, a cycle is 1/977778 seconds.

The trace is performed by recording the data from the program counter of the interrupted program. This is left on the stack as part of the interrupt sequence, and can be accessed by using the stack pointer as an index into the stack and using indexed addressing.



The actual value of the program counter is not stored. The data buffer is arranged as a two-dimensional array and the data is used as an index into this array. On program initialization, this array is set to zero. The correct array element is simply incremented each time an address is read from the stack. The counts are stored as 16-bit values, allowing counts up to \$FFFF, after which an overflow counter is incremented. Also, a frequency counter is maintained to show how many samples were taken. This is needed to interpret the individual counts. The frequency counter is stored as 24 bits, and the trace shuts off automatically when this counter overflows.

The Trace cannot be active during Input/Output operations due to timing conflicts. For this reason I have implemented two jump vectors to stop and start the trace without resetting any program variables or the accumulated counts. A call to KILL will stop the trace and turn NMI interrupts off. RESET will resume execution and preserve any data in the buffer. A call to CHART at any time will print the chart showing the current counts. Output can be stopped by pressing the SHIFT and COMMODORE keys together. It is important to call KILL before calling CHART to reset the interrupt vectors.

Using The Program

Assuming the program is assembled at \$8000, the entry points look like this:

```
trace = 32768  
kill   = trace + 3  
chart  = trace + 6  
reset  = trace + 9
```


To start the program once it's in memory, do a SYS TRACE. This will zero all counts in the buffer and begin the trace. When you wish to see the chart, do a SYS KILL: SYS CHART. KILL and RESET are used as explained above.

The start and end addresses for which the trace takes place are defined in the source code by the variables CBEGIN and CFINISH. As listed, these are set to \$A0 and \$FF respectively, to trace all code from \$A000 to \$FFFF: the ROMs and the often-used \$C000-CFFF block of RAM. Change these and re-assemble the source to trace other areas. When assembling, note that this source was assembled using the Commodore assembler. The mathematical expression used in the subroutine ZBUFF may cause some assemblers trouble - modify it or compute the values by hand if necessary for your assembler.

All output is in the form of hexadecimal numbers. Leading zeros are suppressed. Zero values are indicated by blank spaces on the chart. Hexadecimal representation was chosen for the counts to fit more information on an eighty column printout.

Interpretation of the chart can be done in several different ways. If the objective of the trace is to improve program performance, the amount of time spent in each section of code can be analyzed to determine what portion of the code is using the most processor time; the most-used routines can then be optimized. If certain branches are never being taken, they can be removed. Branches can be arranged so that the one taken most often is checked for first. There are a great many such small changes that can result in a remarkable increase in execution speed.

If your program is not working at all, a trace chart can show what sections of code are being executed. This is usually all you need to know in order to get it running. Once the problem has been located, the fix is usually quite simple. If the program is going into an endless loop, changing the trace program to perform a JSR CHART before it ends due to frequency counter overflow will let you see where the loop is taking place. If you are jumping to a location out of the program area for some reason, you may be able to see where the jump is going. This is sometimes enough information to determine how the jump took place.

Running a trace on a BASIC program will show you a great deal about how the interpreter works. It will show the location of ROM routines. It will also show which routines are using the most time and resources. IRQ routines, both user and KERNEL, are also traced. All of this information can be useful to a programmer.

If anyone would like to correspond with me regarding this program or any topic related to assembly language programming, feel free to contact me at this address:

Richard Stringer
 7805 Villa Cliff #111
 Dallas, Texas 75228

Program Trace Monitor: Generates PRG object file on disk

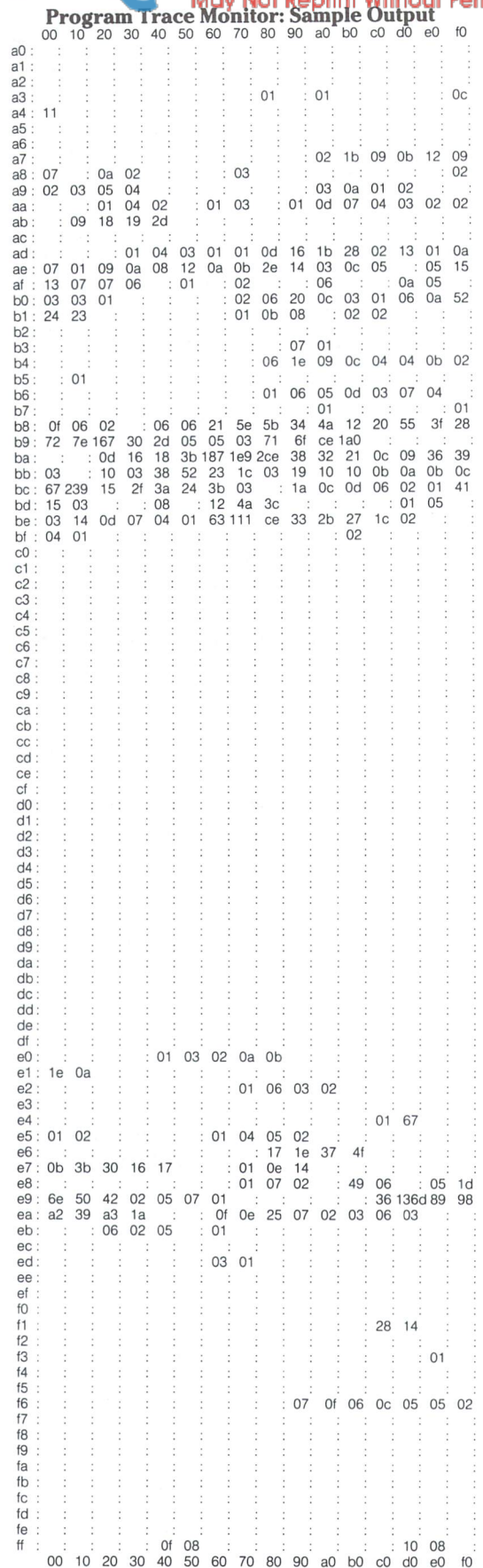
```
PE 100 rem* data loader for "trace" *
OP 110 for i= 1 to 883:read a:cs = cs + a:next
GK 120 if cs<>95794 then print " !data error! " :stop
JD 130 rem data ok, write to file
IA 140 open 1,8,1, "0:trace.com"
OK 150 restore
GO 160 for i= 1 to 883: read a: print#1,chr$(a); next
FO 170 close 1: end
```

IC	1000 data	0, 128, 76, 12, 128, 76, 97, 128
IF	1010 data	76, 39, 129, 76, 29, 128, 32, 38
AM	1020 data	130, 169, 0, 141, 122, 131, 141, 123
NA	1030 data	131, 141, 124, 131, 141, 125, 131, 173
IF	1040 data	24, 3, 141, 114, 131, 169, 123, 141
HA	1050 data	24, 3, 173, 25, 3, 141, 115, 131
LF	1060 data	169, 128, 141, 25, 3, 173, 60, 3
IK	1070 data	208, 5, 169, 1, 141, 60, 3, 141
LE	1080 data	6, 221, 169, 0, 141, 7, 221, 169
HD	1090 data	0, 141, 5, 221, 169, 100, 141, 4
KH	1100 data	221, 120, 169, 65, 141, 15, 221, 169
IB	1110 data	1, 141, 14, 221, 169, 130, 141, 13
KP	1120 data	221, 88, 96, 169, 127, 141, 13, 221
EE	1130 data	169, 0, 141, 15, 221, 141, 14, 221
LO	1140 data	173, 114, 131, 141, 24, 3, 173, 115
OO	1150 data	131, 141, 25, 3, 96, 72, 138, 72
FK	1160 data	152, 72, 169, 2, 44, 13, 221, 240
FN	1170 data	3, 76, 149, 128, 32, 97, 128, 169
JC	1180 data	127, 141, 13, 221, 76, 76, 254, 169
FM	1190 data	127, 141, 13, 221, 169, 0, 141, 14
AO	1200 data	221, 141, 15, 221, 186, 189, 5, 1
BN	1210 data	141, 121, 131, 189, 6, 1, 201, 160
FH	1220 data	240, 2, 144, 16, 201, 255, 240, 2
OL	1230 data	176, 10, 56, 233, 160, 141, 119, 131
NN	1240 data	88, 32, 200, 128, 32, 51, 128, 76
JP	1250 data	129, 234, 32, 29, 130, 169, 0, 141
FA	1260 data	120, 131, 173, 121, 131, 41, 240, 74
FO	1270 data	74, 74, 141, 121, 131, 173, 119, 131
AN	1280 data	162, 4, 10, 46, 120, 131, 202, 16
LC	1290 data	249, 101, 251, 133, 251, 165, 252, 109
OC	1300 data	120, 131, 133, 252, 172, 121, 131, 177
PI	1310 data	251, 24, 105, 1, 145, 251, 200, 177
IH	1320 data	251, 105, 0, 145, 251, 144, 15, 169
EA	1330 data	0, 109, 125, 131, 141, 125, 131, 169
LG	1340 data	255, 145, 251, 136, 145, 251, 238, 122
JP	1350 data	131, 208, 13, 238, 123, 131, 208, 8
HG	1360 data	238, 124, 131, 208, 3, 32, 97, 128
BM	1370 data	96, 169, 160, 141, 117, 131, 32, 250
KD	1380 data	129, 162, 4, 32, 201, 255, 32, 71
BH	1390 data	130, 32, 29, 130, 169, 13, 32, 210
DJ	1400 data	255, 173, 117, 131, 240, 77, 201, 255
GL	1410 data	240, 2, 176, 71, 32, 197, 129, 169
LJ	1420 data	58, 32, 210, 255, 169, 32, 32, 210
BN	1430 data	255, 160, 255, 200, 192, 32, 240, 32
FL	1440 data	173, 141, 2, 201, 3, 208, 7, 169
MP	1450 data	13, 32, 210, 255, 144, 37, 177, 251
LK	1460 data	141, 116, 131, 208, 45, 200, 177, 251
KO	1470 data	208, 43, 32, 57, 130, 76, 89, 129
FA	1480 data	238, 117, 131, 24, 165, 251, 105, 32
DN	1490 data	133, 251, 165, 252, 105, 0, 133, 252
FE	1500 data	76, 58, 129, 32, 71, 130, 32, 96
MO	1510 data	130, 169, 4, 32, 195, 255, 32, 204
EB	1520 data	255, 96, 200, 177, 251, 140, 113, 131
JG	1530 data	172, 116, 131, 32, 182, 129, 169, 32
AC	1540 data	32, 210, 255, 172, 113, 131, 208, 163
DC	1550 data	140, 61, 3, 32, 197, 129, 173, 61
HO	1560 data	3, 32, 220, 129, 76, 212, 129, 32
CG	1570 data	220, 129, 201, 48, 208, 8, 192, 48
JB	1580 data	208, 7, 160, 32, 208, 3, 32, 210
AN	1590 data	255, 152, 32, 210, 255, 96, 170, 41
PG	1600 data	240, 74, 74, 74, 74, 32, 240, 129
NI	1610 data	72, 138, 41, 15, 32, 240, 129, 168
GD	1620 data	104, 96, 201, 10, 144, 3, 24, 105
NH	1630 data	7, 105, 48, 96, 169, 255, 141, 16
DK	1640 data	130, 173, 24, 208, 41, 2, 240, 5
JF	1650 data	169, 7, 141, 16, 130, 169, 4, 162

ID	1660 data	4, 160, 255, 32, 186, 255, 169, 0
LE	1670 data	32, 189, 255, 32, 192, 255, 96, 169
BK	1680 data	126, 133, 251, 169, 131, 133, 252, 96
CF	1690 data	32, 29, 130, 162, 12, 160, 0, 152
KK	1700 data	145, 251, 200, 208, 251, 230, 252, 202
HM	1710 data	208, 246, 96, 140, 113, 131, 160, 209
JL	1720 data	169, 130, 32, 79, 130, 172, 113, 131
AA	1730 data	96, 160, 24, 169, 131, 32, 79, 130
BN	1740 data	96, 132, 53, 133, 54, 160, 0, 177
KO	1750 data	53, 240, 6, 32, 210, 255, 200, 208
CM	1760 data	246, 96, 160, 234, 169, 130, 32, 79
HA	1770 data	130, 172, 60, 3, 169, 0, 32, 182
OI	1780 data	129, 160, 21, 169, 131, 32, 79, 130
KB	1790 data	160, 214, 169, 130, 32, 79, 130, 160
HO	1800 data	100, 169, 0, 32, 182, 129, 160, 21
CD	1810 data	169, 131, 32, 79, 130, 160, 0, 169
BH	1820 data	131, 32, 79, 130, 173, 124, 131, 32
MC	1830 data	220, 129, 32, 212, 129, 173, 123, 131
KD	1840 data	32, 220, 129, 32, 212, 129, 173, 122
GE	1850 data	131, 32, 220, 129, 32, 212, 129, 160
HH	1860 data	21, 169, 131, 32, 79, 130, 160, 92
LF	1870 data	169, 131, 32, 79, 130, 173, 125, 131
AM	1880 data	32, 220, 129, 32, 212, 129, 160, 21
KI	1890 data	169, 131, 32, 79, 130, 169, 13, 32
CC	1900 data	210, 255, 96, 32, 32, 32, 58, 0
NO	1910 data	13, 67, 76, 79, 67, 75, 32, 67
JO	1920 data	89, 67, 76, 69, 32, 40, 65, 41
HG	1930 data	32, 32, 91, 0, 13, 13, 13, 67
IA	1940 data	89, 67, 76, 69, 83, 32, 32, 32
KL	1950 data	32, 32, 32, 40, 66, 41, 32, 32
FO	1960 data	91, 0, 13, 83, 65, 77, 80, 76
LD	1970 data	69, 32, 67, 79, 85, 78, 84, 32
MJ	1980 data	32, 32, 32, 32, 91, 32, 0, 32
PM	1990 data	93, 0, 32, 32, 32, 32, 32, 48
JB	2000 data	48, 32, 32, 49, 48, 32, 32, 50
BC	2010 data	48, 32, 32, 51, 48, 32, 32, 52
PC	2020 data	48, 32, 32, 53, 48, 32, 32, 54
ND	2030 data	48, 32, 32, 55, 48, 32, 32, 56
ME	2040 data	48, 32, 32, 57, 48, 32, 32, 65
LF	2050 data	48, 32, 32, 66, 48, 32, 32, 67
JG	2060 data	48, 32, 32, 68, 48, 32, 32, 69
DF	2070 data	48, 32, 32, 70, 48, 0, 13, 79
MM	2080 data	86, 69, 82, 70, 76, 79, 87, 83
MC	2090 data	32, 32, 32, 32, 32, 32, 32, 32
BL	2100 data	91, 32, 0

Program Trace Monitor: Demo Program (note line 110)

NN	100 if peek(32768) + peek(32769) = 88 then goto 160
DN	110 load "trace.com", 8, 1
NE	120 this program will demonstrate the
NK	130 use of the trace program from basic
HE	140 the same technique is used to trace
DO	150 an assembly language program
CO	160 poke 56, 128: rem protect program from basic
EG	170 trace = 32768: kill = trace + 3: chart = kill + 3
LH	180 poke 828, 10: rem set frequency
FM	190 sys trace: rem start trace
DJ	200 for t = 1 to 110: d = sin(t): f\$ = str\$(int(d) + 1)
GF	210 poke 53281, peek(53281) and 15
IJ	220 print peek(53281): print d, f\$
NM	230 if t = 100 then gosub 250
CK	240 next: end
PB	250 sys kill
KA	260 sys chart
KC	270 return



cycles (b) [0a]
 clock cycle (a) [64]
 sample count [0038cb]
 overflows [00]

Program Trace Monitor: PAL Source Code

EG	1000	open 1,8,1,"0:trace.com"	MN	1860	;milli.on timer b underflow a	OA	2740	beq	issame
DM	1010	sys700	GB	1870	;nmi is generated.vector points	ME	2750	bcs	bugout
KA	1020	.opt o1	EI	1880	;to tstprg	AG	2760	issame	sec
MH	1030	;	IN	1890	;	AJ	2770	sbc	#cbegin
BG	1040	;program name=trace/nmi	JP	1900	setirq jsr zbuff	EG	2780	sta	hibyte
AJ	1050	;	BG	1910	lda #0	MF	2790	;	
AH	1060	;purpose -to trace and record	AI	1920	sta freq	KO	2800	;we need to allow system irq	
KM	1070	;the data from the <pc> in order	ME	1930	sta freq+1	ON	2810	;requests here because we are	
KJ	1080	;to analyze the performance of	KF	1940	sta freq+2	HG	2820	;stealing so much cpu time	
DK	1090	;a program	FF	1950	sta vflag	EI	2830	;	
CM	1100	;	JO	1960	settwo lda vector	GK	2840	cli	
FN	1110	;last update -12/26/85	OC	1970	sta oldvec	JM	2850	jsr mainpg ;index array	
GN	1120	;	LP	1980	lda #<tstprg	NP	2860	bugout jsr settmr	
KA	1130	;author - r.w.stringer	HP	1990	sta vector	DH	2870	jmp \$ea81	
IM	1140	; 7805 villa cliff #111	KK	2000	lda vector+1	GL	2880	;	
LG	1150	; dallas texas 75228	IB	2010	sta oldvec+1	CO	2890	;calculate array index and	
GP	1160	; (214)-327-3039	PB	2020	lda #>tstprg	OF	2900	;increment the proper array var	
IA	1170	;	GA	2030	sta vector+1	CM	2910	;array is- array[0..xx,0..15]	
LG	1180	;special note-i/o cannot operate	BG	2040	settmr lda milli	AN	2920	;of double integer [0..5535]	
HC	1190	;with nmi interrupts functioning	AH	2050	bne sttmb	HG	2930	;array index is calculated as	
GH	1200	;the unset routine (kill jmp	JP	2060	lda #1	NH	2940	; (hibyte-#cbegin)*32 +	
IM	1210	;vector) should be called before	IP	2070	sta milli	LJ	2950	; (lobyte/16)*2 +	
AH	1220	;any input/output	LM	2080	sttmb sta tblo	CO	2960	;buffer start address	
CD	1230	;-buffer size in the	FB	2090	lda #0	GG	2970	;overflow is checked for and	
NG	1240	;present configuration is limited	OK	2100	sta tbhi	GC	2980	;sample count incremented.sample	
IC	1250	;to 3 kbytes.this is room for a	HN	2110	lda #>ccycle	KC	2990	;count is maintained as a 24 bit	
EI	1260	;trace of 24kbytes of address	AM	2120	sta tahi	NB	3000	;integer (2t24)-1.trace is	
KL	1270	;space.to trace more space the	PO	2130	lda #>ccycle	NA	3010	;terminated on sample overflow	
GE	1280	;program will need to move the	IP	2140	sta talo	NH	3020	;a flag is set on var overflow	
BK	1290	;buffer or remove basic from the	PP	2150	sei	KF	3030	;var's are kept in lo/high format	
EC	1300	;current address space.no basic	LA	2160	lda #%01000001	GF	3040	;	
MA	1310	;routines are used	LP	2170	sta tbctrl	JE	3050	mainpg jsr setbuf	
OJ	1320	;	LB	2180	lda #%00000001	PN	3060	lda #0	
IK	1330	;	NA	2190	sta tactrl	IE	3070	sta hibyte+1	
IA	1340	; system equates	BD	2200	lda #%10000010	CG	3080	lda lobyte	
KN	1350	zpg = \$fb	OP	2210	sta irc	PA	3090	and #\$f0	
KB	1360	temp = 53	KD	2220	cli	GD	3100	lsr a	
GE	1370	skey = 653 ;shft/ctrl/cmdre flag	CK	2230	rts	AE	3110	lsr a	
GE	1380	cbegin = \$a0 ;trace start address	GD	2240	;	KE	3120	lsr a	
BC	1390	cfinsh = \$ff ;trace end address	OC	2250	;reset vectors to normal and	CN	3130	sta lobyte	
BH	1400	;above set range for trace	ML	2260	;turn interrupts off.needed for	OI	3140	lda hibyte	
FJ	1410	ccycle = 100 ;cycle count timer a	PP	2270	;i/o operations	NJ	3150	ldx #4	
BK	1420	milli = 828 ;timer b count	OF	2280	;	FJ	3160	mul32 asl a	
DF	1430	vector = 792 ;nmi vector	GD	2290	unset lda #irqoff	HM	3170	rol hibyte+1	
OP	1440	irqoff = %01111111;interrupt mask	IF	2300	sta irc	DC	3180	dex	
MA	1450	bmask = %00000010;timer b mask	BP	2310	lda #0	DP	3190	bpl mul32	
GC	1460	talo = \$dd04 ;timer a latch low	BJ	2320	sta tbctrl	JK	3200	adc zpg	
LH	1470	tahi = \$dd05 ; hi	JJ	2330	sta tactrl	PP	3210	sta zpg	
DN	1480	tactrl = \$dd0e ;timer a control reg	CG	2340	lda oldvec	EK	3220	lda zpg+1	
PE	1490	tblo = \$dd06 ;timer b latch low	PF	2350	sta vector	MJ	3230	adc hibyte+1	
CK	1500	tbhi = \$dd07 ; hi	ID	2360	lda oldvec+1	GP	3240	sta zpg+1	
JP	1510	tbctrl = \$dd0f ;timer b control reg	KF	2370	sta vector+1	MG	3250	ldy lobyte	
AB	1520	irc = \$dd0d ;interrupt control reg	ID	2380	rts	IJ	3260	lda (zpg),y	
HA	1530	; kernal equates	MM	2390	;	MD	3270	clc	
NC	1540	setfls = \$ffa	FA	2400	;stop interrupts.pull address	PK	3280	adc #1	
AC	1550	setnam = \$ffb	OE	2410	;of next program step from stack	DF	3290	sta (zpg),y ;indexed	
HF	1560	open = \$ffc0	DN	2420	;check if in range.subtract offset	CJ	3300	iny ;element	
JE	1570	chrout = \$ffd2	GI	2430	;from hibyte and store.store	KM	3310	lda (zpg),y	
JH	1580	close = \$ffc3	EI	2440	;lobyte and jsr to array routine	FN	3320	adc #0	
CC	1590	clrchn = \$fcc	KG	2450	;on return turn interrupts on	MB	3330	sta (zpg),y	
OE	1600	chkout = \$ffc9	KJ	2460	;reset timers and exit.	NL	3340	bcc test1	
AM	1610	;	MB	2470	;	BA	3350	lda #0	
JF	1620	;the difference between cbegin	EH	2480	tstprg pha	LI	3360	adc vflag	
IH	1630	;and cfinsh can be no larger than	OG	2490	txa	BO	3370	sta vflag	
KF	1640	;\$f.the buffer at the present	AE	2500	pha	BP	3380	lda #\$ff	
DF	1650	;location is only 3k long	FI	2510	tya	IF	3390	sta (zpg),y	
OJ	1660	;in order to trace the complete	EF	2520	pha	DA	3400	dey	
GO	1670	;64k address space you will need	FM	2530	lda #bmask	MG	3410	sta (zpg),y	
MA	1680	;to allocate an 8k buffer.the	BF	2540	bit irc	OI	3420	test1 inc freq	
LH	1690	;chart will be 256 lines long.the	HJ	2550	beq outjmp	BE	3430	bne test2	
GF	1700	;buffer space formula is	LO	2560	jmp swich	EB	3440	inc freq+1	
IP	1710	;buffer=(#kbytes to trace)/8	KP	2570	outjmp jsr unset	FF	3450	bne test2	
OC	1720	;	IB	2580	lda #irqoff	MC	3460	inc freq+2	
IO	1730	* = \$8000 ;program orgin	KH	2590	sta irc	JG	3470	bne test2	
CE	1740	;	FH	2600	jmp \$fe4c	PL	3480	jsr unset	
OE	1750	; jump vectors	LE	2610	swich lda #irqoff	GL	3490	test2 rts	
MJ	1760	trace jmp setirq ;start	IJ	2620	sta irc	CC	3500	;	
KA	1770	kill jmp unset ;stop	BD	2630	lda #0	BI	3510	;print results in chart form	
NM	1780	chart jmp pntcht ;printout	PM	2640	sta tactrl	EI	3520	;with proper formatting.leading	
CL	1790	reset jmp settwo ;restart	LN	2650	sta tbctrl	EM	3530	;zeros are suppressed on two	
OH	1800	;	FG	2660	tsx	BO	3540	;byte hexadecimal numbers	
KK	1810	;set up vectors zero variables	CE	2670	lda \$0105,x	EF	3550	;	
MD	1820	;set up and start timers	AB	2680	sta lobyte	OF	3560	;	
BP	1830	;timer b counts underflows	HF	2690	lda \$0106,x	JB	3570	pntcht lda #cbegin	
GP	1840	;from timer a and counts down	PH	2700	cmp #cbegin	OL	3580	sta fre2	
LB	1850	;the number of underflows in	JJ	2710	beq tryhi	GO	3590	jsr p2scr	
			OO	2720	bcc bugout	PF	3600	ldx #4	
			IP	2730	tryhi cmp #cfinsh	MN	3610	jsr chkout	

AP	3620	jsr	phead	FO	4500	jsr	ho2	OI	5380	lda	#>messg5
FM	3630	jsr	setbuf	NG	4510	tay		JP	5390	jsr	pstrng
CF	3640	pnch1	lda	AD	4520	pla		OP	5400	ldy	#<messg3
JB	3650	jsr	chrout	OJ	4530	rts		EK	5410	lda	#>messg3
AN	3660	lda	fre2	LO	4540	ho2	cmp	HB	5420	jsr	pstrng
FC	3670	beq	pfini	HN	4550	bcc	nixadd	DD	5430	ldy	#<ccycle
IH	3680	cmp	#cfinsh	GE	4560	clc		JN	5440	lda	#>ccycle
BA	3690	beq	chks	FM	4570	adc	#7	DP	5450	jsr	hex16
FE	3700	bcs	pfini	DI	4580	nixadd	adc	CE	5460	ldy	#<messg5
NC	3710	chks	jsr	KN	4590	rts		IO	5470	lda	#>messg5
NI	3720	lda	#"	OG	4600			DF	5480	jsr	pstrng
JG	3730	jsr	chrout	LK	4610			MF	5490	ldy	#<messg4
DC	3740	lda	#32	CI	4620			CA	5500	lda	#>messg4
NH	3750	jsr	chrout	JJ	4630	p2scr	lda	BH	5510	jsr	pstrng
CH	3760	ldy	#255	JF	4640	sta	pindex+1	IB	5520	lda	freq+2
FH	3770	pnloop	iny	BN	4650	lda	53272	CI	5530	jsr	hexout
NL	3780	cpy	#32	DM	4660	and	#2	LF	5540	jsr	nozera
MG	3790	beq	pnch2	EA	4670	beq	ucase	CD	5550	lda	freq+1
MF	3800	lda	skey	BE	4680	lda	#7	AK	5560	jsr	hexout
IB	3810	cmp	#3	LI	4690	sta	pindex+1	JH	5570	jsr	nozera
DE	3820	bne	pnch	HM	4700	ucase	lda	OI	5580	lda	freq
MH	3830	lda	#13	FL	4710	ldx	#4	OL	5590	jsr	hexout
HN	3840	jsr	chrout	PO	4720	pindex	ldy	HJ	5600	jsr	nozera
LJ	3850	bcc	pfini	EE	4730	jsr	setlfs	IN	5610	ldy	#<messg5
OM	3860	pnch	lda	PG	4740	lda	#0	OH	5620	lda	#>messg5
AN	3870	sta	lcnt	PE	4750	jsr	setnam	JO	5630	jsr	pstrng
GB	3880	bne	bumpy	MA	4760	jsr	open	OP	5640	ldy	#<messg7
CB	3890	iny		OI	4770	rts		EK	5650	lda	#>messg7
IB	3900	lda	(zpg),y	CC	4780			HA	5660	jsr	pstrng
IC	3910	bne	pnth	CK	4790			PJ	5670	lda	vflag
MM	3920	jsr	pnth3sp	GD	4800			IB	5680	jsr	hexout
AB	3930	jmp	pnloop	AN	4810	setbuf	lda	BP	5690	jsr	nozera
CA	3940	pnch2	inc	JE	4820	sta	zpg	CD	5700	ldy	#<messg5
EO	3950	clc		NM	4830	lda	#>trbuf	IN	5710	lda	#>messg5
PK	3960	lda	zpg	GD	4840	sta	zpg+1	DE	5720	jsr	pstrng
LP	3970	adc	#32	ON	4850	rts		IO	5730	lda	#13
BA	3980	sta	zpg	CH	4860			DE	5740	jsr	chrout
GK	3990	lda	zpg+1	JE	4870			CG	5750	rts	
NH	4000	adc	#0	GI	4880			GP	5760		
IP	4010	sta	zpg+1	JL	4890	zbuff	jsr	LM	5770		
FH	4020	jmp	pnch1	HJ	4900	ldx	#(cfinsh+1-cbegin)>3	KA	5780		
FA	4030	pfini	jsr	JH	4910	ldy	#0	DK	5790	space	.asc " ;"
HK	4040	jsr	pnthsta	PO	4920	tya		MJ	5800	.byte	0
FM	4050	lda	#4	MP	4930	zloop1	sta	IC	5810		
BF	4060	jsr	close	MC	4940	iny		OD	5820	messg3	.byte 13
NO	4070	jsr	clrchn	FK	4950	bne	zloop1	PE	5830	.asc "clock cycle (a) ["	
MN	4080	rts		AJ	4960	inc	zpg+1	EM	5840	.byte	0
PB	4090	bumpy	iny	BC	4970	dex		AF	5850		
AO	4100	lda	(zpg),y	DM	4980	bne	zloop1	FJ	5860	messg2	.byte 13,13,13
JO	4110	pnth	sty	KG	4990	rts		CM	5870	.asc "cycles (b) ["	
MO	4120	ldy	lcnt	OP	5000			MO	5880	.byte	0
LM	4130	jsr	hex16	GG	5010			IH	5890		
DL	4140	lda	#32	CB	5020			BJ	5900	messg4	.byte 13
NA	4150	jsr	chrout	GP	5030	pnth3sp	sty	OJ	5910	.asc "sample count ["	
JI	4160	ldy	ysave	AN	5040	ldy	#<space	EB	5920	.byte	0
HM	4170	bne	pnloop	GH	5050	lda	#>space	AK	5930		
KM	4180			PK	5060	jsr	pstrng	KM	5940	messg5	.asc "]"
PG	4190			HB	5070	ldy	ysave	CD	5950	.byte	0
ON	4200			EM	5080	rts		OL	5960		
KF	4210	hex16	sty	IF	5090			OK	5970	messg	= *
OE	4220	jsr	pnthex	JJ	5100			NG	5980	.asc " 00 10 20"	
FD	4230	lda	829	MG	5110			EK	5990	.asc " 30 40 50 60"	
IH	4240	jsr	hexout	FD	5120	phead	ldy	DO	6000	.asc " 70 80 90 a0"	
HD	4250	jmp	nozera	AM	5130	lda	#>messg	KB	6010	.asc " b0 c0 d0 e0 f0"	
KB	4260			PP	5140	jsr	pstrng	IH	6020	.byte	0
HL	4270			KA	5150	rts		MB	6030	messg7	.byte 13
OC	4280			OJ	5160			GH	6040	.asc "overflows ["	
GM	4290	pnthex	jsr	GB	5170			GJ	6050	.byte	0
IP	4300	cmp	#"0"	CL	5180			LK	6060		
KD	4310	bne	nozera	BE	5190	pstrng	sty	AB	6070	ysave	.byte 0
JD	4320	cpy	#"0"	HA	5200	sta	temp+1	PM	6080	oldvec	.byte 0,0
OH	4330	bne	nozery	FK	5210	ldy	#0	LK	6090	lcnt	.byte 0
LN	4340	ldy	#32	PM	5220	pstrg1	lda	OK	6100	fre2	.byte 0,0
CJ	4350	bne	nozery	KO	5230	beq	pstrg2	FP	6110	hibyte	.byte 0,0
MO	4360	nozera	jsr	PE	5240	jsr	chrout	BD	6120	lobyte	.byte 0
CC	4370	nozery	tya	CG	5250	iny		JA	6130	freq	.byte 0,0,0
DP	4380	jsr	chrout	BP	5260	bne	pstrg1	LJ	6140	vflag	.byte 0
CB	4390	rts		HD	5270	pstrg2	rts	DF	6150	trbuf	= *
FC	4400	hexout	tax	GB	5280			MO	6160	.end	
HD	4410	and	#\$f0	HB	5290						
OF	4420	lsr	a	KC	5300						
IG	4430	lsr	a	IH	5310	pnthsta	ldy				
CH	4440	lsr	a	GE	5320	lda	#>messg2				
MH	4450	lsr	a	NL	5330	jsr	pstrng				
NL	4460	jsr	ho2	AO	5340	ldy	milli				
CP	4470	pha		BN	5350	lda	#0				
ED	4480	txa		JJ	5360	jsr	hex16				
NJ	4490	and	#\$0f	IO	5370	ldy	#<messg5				

The 8563 Video Display Controller

David Stidolph
Madison, Wisconsin

The 80 column screen of the C128 is driven by a chip called the 8563 Video Display Controller, or VDC for short. The VDC is extremely flexible, and almost every aspect of the display (characters per line, lines per screen, height and width of characters, etc.) can be changed to suit your preferences. Before describing how to use these features I will explain how to access the VDC.

Accessing The VDC

The VDC contains 37 registers (numbered 0 to 36) which control how it displays information contained in its own bank of 16K RAM (this memory is not accessible directly by the CPU – only the VDC can get at it). Access to these registers is restricted to two memory locations – \$d600 and \$d601. The register number you wish to read or write is placed in \$d600 and \$d601 becomes “connected” with the register pointed to by \$d600. This complicated addressing scheme is made a little more difficult by the possibility that the VDC is not ready to connect the register you want immediately following a write to \$d600. This is handled by making \$d600 dual purpose. Writing to \$d600 specifies the register number. Reading \$d600 returns status information. Below is a description of the bits in \$d600:

Location \$d600 (VDC access registers)

	d7	d6	d5	d4	d3	d2	d1	d0
Write	--	--	r5	r4	r3	r2	r1	r0
Read	status	lp	vblank	--	--	ver0	ver1	ver2

When you read \$d600, the highest bit is set if the information at \$d601 is valid (for a read or write). Bit 6 is cleared when the light pen signal has been sensed. Bit 5 is set during vertical retrace. The lowest 3 bits return the version number of the VDC. An easy test to determine a C128, in 64 mode, from a normal C64 is to PEEK this location. A C64 will ALWAYS return a zero.

Programming the VDC seems impossible from BASIC (it is too slow), so all examples will be shown in 8502 assembly code. Furthermore, the 80 column chip is just as accessible from 64 mode, so all examples will work unchanged in either mode.

The following are two routines for reading and writing the VDC registers. When called, register 'X' should have the VDC register number, and the accumulator is used to pass the information. Register 'Y' is not affected.

```

READ   STX $D600      ;WRITE REG# TO VDC
WAIT01 BIT $D600     ;WAIT FOR BIT 7 OF
          BPL WAIT01  ; THE VDC TO BE SET
          LDA $D601   ;LOAD REGISTER DATA
          RTS         ;AND RETURN
;
WRITE  STX $D600      ;WRITE REG# TO VDC
WAIT02 BIT $D600     ;WAIT FOR BIT 7 OF
          BPL WAIT02  ; THE VDC TO BE SET
          STA $D601   ;PUT IN REGISTER
          RTS         ;AND RETURN
    
```

The following is a list of the VDC registers and their functions. Many of the registers provide multiple functions, so each function is given its own line. The default (upon power up) is given for each setting. An 'x' means that the bit position it is in is not used for that function. For example, register 3 has two functions. One uses the upper four bits to control the vertical sync width, and the other function uses the lower four bits to control the horizontal sync width.

VDC Registers

Reg	Normal	Description
0	%01111110	Horizontal Total
1	%01010000	Horizontal Displayed
2	%01100111	Horizontal Sync Position
3	%0100xxxx %xxxx1001	Vertical Sync Width Horizontal Sync Width
4	%00100110	Vertical Total
5	%xxxx0000	Vertical Total Adjust
6	%01010000	Vertical Displayed
7	%00100000	Vertical Sync Position
8	%xxxxxx0x %xxxxxxx0	Video Mode Interlace-Sync Mode
9	%xxx00111	Total Rasters/Character
10	%x01xxxxx %xxx00000	Cursor Mode Cursor Start: Raster Line
11	%xxx00111	Cursor End: Raster Line
12	%00000000	Display Start (HI byte)
13	%00000000	Display Start (LO byte)
14	%00000000	Cursor Address (HI)
15	%00000000	Cursor Address (LO)
16	%00000000	Light Pen Vertical
17	%00000000	Light Pen Horizontal
18	%00000000	Address Pointer (HI)
19	%00000000	Address Pointer (LO)
20	%00000100	Attribute Start (HI)
21	%00000000	Attribute Start (LO)
22	%0111xxxx %xxxx1000	Character Total Width Character Displayed Width
23	%xxxx0111	Char Displayed, Vertical
24	%0xxxxxxx %x0xxxxxx %xx1xxxxx %xxx00000	Block Copy Mode Reverse Entire Screen Character Blink Rate Vertical Smooth Scroll
25	%0xxxxxxx %x1xxxxxx %xx0xxxxx %xxx0xxxx %xxxx0000	Bit-Map Graphics Mode Enable Attributes Semi-Graphic Mode Double Pixel Width Horizontal Smooth Scroll
26	%1111xxxx %xxxx0000	Foreground Color (RGBI) Background Color (RGBI)
27	%00000000	Address Increment Per Row
28	%001xxxxx %xxx0xxxx	Character Set Start 8563 DRAM Type
29	%xxx00111	Underline Raster Line
30	%00000000	Word Count
31	%00000000	Data Byte
32	%00000000	Block Copy Source (HI)
33	%00000000	Block Copy Source (LO)
34	%01111101	Display Enable Begin
35	%01100110	Display Enable End
36	%xxxx0101	DRAM Refresh Rate

Before we look at individual registers and functions, we must look at how the VDC uses its 16K memory bank, and how we can access it.

VDC Memory

The VDC has two 16K by 4-bit dynamic rams attached to it. These rams can only be addressed by the VDC, so any access to them MUST be through it. You first place the address you want to read or write into the address pointers (registers 18 and 19). Then you read or write the data byte register (#31). For example, to load VDC memory location \$10dc with the value \$20, the following code could be used:

```
SET20  LDX #18
        LDA #$10      ;HI BYTE
        JSR WRITE
        INX           ;NOW POINT TO 19
        LDA #$DC      ;LO BYTE
        JSR WRITE
        LDX #31       ;DATA REGISTER
        LDA #$20      ;BYTE TO WRITE
        JSR WRITE
        RTS
```

Due to an undocumented "bug" in the VDC, you should always load the address pointer high byte first. If you load the high byte last, you may get the wrong address.

One nice feature of the address pointers is that after each byte is read or written through register 31, the address pointer is incremented to point to the next memory position. The following code could be used to print my name to the VDC memory (we will use the same location as the starting point):

```
PRTNAM  LDX #18
        LDA #$10      ;HI BYTE
        JSR WRITE
        INX           ;NOW POINT TO 19
        LDA #$DC      ;LO BYTE
        JSR WRITE
        LDX #31       ;DATA REGISTER
        LDY #00       ;BEGINNING OF NAME
PRTCHR  LDA NAME,Y    ;GET CHARACTER
        BEQ DONE      ;END CHARACTER?
        JSR WRITE     ;PRINT CHARACTER
        INY           ;POINT TO NEXT
        JMP PRTCHAR
DONE    RTS
NAME    .BYT 'DAVID STIDOLPH',0
```

Of course the above example ignores the fact that the normal definition of characters does not follow their ASCII equivalents (unless you to set them up that way). The code also does not update the attribute memory.

Fill and Copy Mode

If you want to clear the display by filling it with spaces (or any other data), there is a special register to write the same byte to successive VDC memory locations. This register (30) should be written AFTER you have defined the starting address in the address pointer registers and written the data to the data byte register. You must remember, however, that as soon as you write a byte to the data byte register, the memory location is set to that value and the

address pointers are incremented. This means that you should use one less than the number of characters you want to fill. The following example clears the entire VDC memory.

```
CLEAR  LDX #18          ;SET ADDRESS
        LDA #00         ; POINTER TO START
        JSR WRITE       ; OF MEMORY
        INX
        JSR WRITE
        LDX #24         ;CLEAR COPY BIT
        JSR READ
        AND #$7F
        JSR WRITE
        LDX #31         ;LOAD DATA BYTE
        JSR WRITE       ; WITH 0 (CLEAR)
        LDX #30         ; AND CLEAR REST
        LDA #255        ; OF FIRST PAGE
        JSR WRITE
        LDY #47         ;47 OTHER 256 BYTE
        LDA #00         ; PAGES TO CLEAR
CLRPAg  JSR WRITE       ;CLEAR 256 BYTES
        DEY             ;COUNTER ZERO?
        BNE CLRPAg     ;NO, CLEAR ANOTHER
        RTS             ;ROUTINE DONE!
```

Another feature of the VDC is the ability to COPY one area of VDC memory to another area (up to 256 bytes at a time). This comes in handy for scrolling the screen.

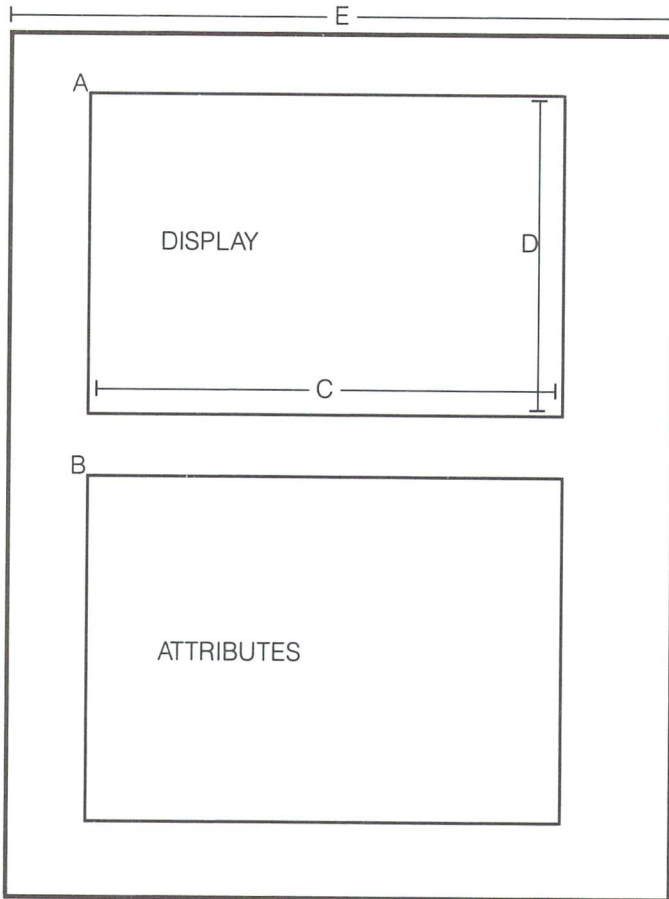
To copy memory the BLOCK COPY bit (register 24, bit 7) must be set. You first set the address pointers (18 and 19) to the destination address and the block copy source registers (32 and 33) to the source of the memory to copy from. Then you write the number of bytes to copy in the word count register (30). As with fill, a zero value indicates 256 bytes are to be copied, however you do not put a byte in the data register (31), so you do not need to write one less than the number you want to word count register (30). The copy process goes from low to hi, so be careful not to overlap your copying.

Setting Up The Display

Most video chips are set to show a fixed number of characters per line and lines per screen. The VDC, however, is more flexible than most display controllers. The number of screen lines is controlled by the vertical displayed register (6). As you increase or decrease the number of lines, you will note that the top line stays fixed at the same position. As you increment the vertical sync position register (7), the screen will "move up", and as you decrease it the screen will "move down". A good formula for setting the register is to add, or subtract, one number for every other line added, or removed, from the normal 25. A more mathematical way to represent this would be:

$$R7 = 32 + (R6 - 25) / 2$$

The VDC allows you to design your display to be a "window" that moves over a larger screen. Figure 1 shows the relationship of the display to the theoretical screen. You could, for instance, have an 80 column display with 25 lines that moves about a 200 column screen with 40 lines. The program examples here will assume only the standard 80 columns.



- A = Start of Display (R12, 13)
- B = Start of Attributes (R20, 13)
- C = Horizontal Displayed (R1)
- D = Vertical Displayed (R6)
- E = Total Number of Columns

R27 (Address Increment per Row) should be set to the Total Number of Columns (E) minus the Number of Displayed Columns.

You can "scroll" your entire display up or down by changing the display start registers (12 and 13). Add 80 to them and the display would start one line down, so the screen would appear to "scroll" down. The following code shows how to move your display one "line" down (this presumes that the new line below the display already has data you want shown).

```

MOVEUP LDX #13      ;GET LO-BYTE
        JSR READ    ;OF DISPLAY
        CLC
        ADC #80     ;ADD 80 AND
        PHP        ;SAVE CARRY
        JSR WRITE   ;STORE NEW VALUE
        DEX        ;NOW THE HIGH BYTE
        JSR READ
        PLP        ;RECALL CARRY
        ADC #00     ;ADD IN CARRY
        JSR WRITE   ;STORE NEW HIGH BYTE
        RTS        ;AND RETURN
    
```

Vertical Smooth Scrolling

The type of scrolling shown above works, but is rather jerky. The VDC, however, has the ability to smoothly scroll, pixel by pixel. Since the display can be flexible in the number of lines that can be displayed, we don't have to reduce the display in order to smooth scroll. In order to smoothly scroll the display "down" a line, just follow these steps:

1. Put the text that will be scrolled onto the screen when it moves down a line into VDC memory. If the text is already there, skip this step.
2. Increment the vertical smooth scroll register (24, bits 4-0) until it is equal to the number of raster lines per character.
3. Add the number of characters per line (register 22) to the display start registers (12 and 13) and set the vertical smooth scroll register back to zero.

Here is an example routine to scroll the screen down one line smoothly. It is assumed that the text on the new line is already there.

```

SMOOTH LDX #24      ;GET VERT SMOOTH
        JSR READ    ;SCROLL REGISTER
        AND #%11100000 ;AND SET TO NO
        JSR WRITE   ;SCROLL
SMTH1  JSR READ    ;READ VALUE
        CLC
        ADC #01     ;AND ADD ONE
        JSR WRITE
SMTH2  LDY #100     ;DELAY VALUE
        DEY
        BNE SMTH2
        AND #%00011111 ;CHECK SCROLL AGAINST
        CMP #7      ;BOTTOM RASTER AND
        BNE SMTH1  ;CONTINUE UNTIL DONE
        JSR READ    ;GET REGISTER AGAIN
        AND #111000000 ;AND CLEAR SCROLL
        JSR WRITE
        JSR MOVEUP  ;NOW MOVE UP ONE LINE
        RTS
    
```

Horizontal Smooth Scrolling

In a similar manner, the screen can be scrolled smoothly left and right. This is a little more complicated because you have to set up the VDC to believe that there are more columns per line than it will display. This is done through the address increment per row register (27). The value of this register is added to the end of each line to get to the beginning of the next line. For example, if you wanted a "virtual" display line of 100 characters, with an actual display line of 80, you would have to put the value 20 into register 27. Simple movement to the left or right one column is then a matter of incrementing or decrementing the display start registers (12 and 13).

Smooth scrolling the screen horizontally is similar in concept to vertical smooth scrolling, except that Commodore has come out with different versions of the VDC which require different initialization, and use, of the smooth scroll register. When you read \$d600 for the status bit, you also get other information which includes the version number of the VDC. If the lower three bits of the VDC are all zero then the smooth scroll off setting is zero. If any of the three bits are on, then the smooth scroll register should be set to seven to be off. The difference between these chips is that the first type is designed to scroll the text to the right, and the second is designed for scrolling to the left. Either way it is a shame Commodore had to cause these incompatibility problems.

Attributes

Like the VIC-II chip, two bytes of memory are used for each character on the display. One byte to determine what character from the character set is displayed, and the other for the color it will be shown in. Unlike the VIC-II chip, every bit in the attribute byte is needed because each character has extra attributes that can be attached to it. In simple terms, the lower four bits determine the color, and the upper four bits determine its attributes. Attribute memory is set up just like the display memory, and has a one-for-one correspondence with it.

Attribute bytes are defined in the following table:

Bit	Attribute
7	Alternate character set
6	Show character in reverse mode
5	Underline the character
4	Blink the character
3	Red
2	Green
1	Blue
0	Intensity

Bit 7 determines which character set is used for displaying the character. If the bit is set the second (or upper) character definition is used. This allows each character to select from either set, allowing 512 possible characters.

Bit 6 is used to display the character in reverse mode. If a pixel in the character definition is on, then it will be displayed of. If it is defined off, then it will be displayed on.

Bit 5 allows you to underline any character (or combination) on the screen. The raster line used for underlining is set by register 29. It could be set to strike through (overline) by changing its value.

Bit 4 will make the character blink. The rate character blinks is set by register 24, bit number 5. If this control bit is cleared, then any character displayed with the blink mode set will be blink fast.

The lower nibble (bits 0-3) selects the color. The following tables shows the result of "mixing" the colors:

Intensity Off

Red	Green	Blue	Color
0	0	0	Black
0	0	1	Blue
0	1	0	Green
0	1	1	Cyan
1	0	0	Red
1	0	1	Purple
1	1	0	Brown
1	1	1	Light Grey

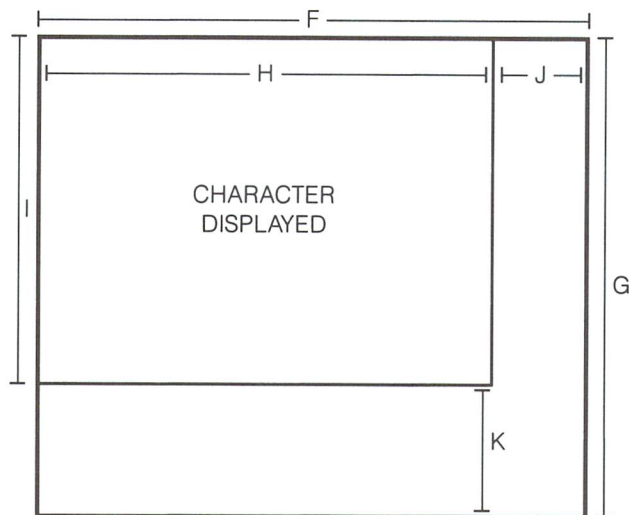
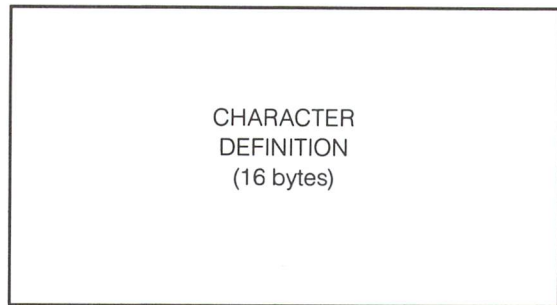
Intensity On

Red	Green	Blue	Color
0	0	0	Dark Grey
0	0	1	Light Blue
0	1	0	Light Green
0	1	1	Light Cyan
1	0	0	Light Red
1	0	1	Light Purple
1	1	0	Yellow
1	1	1	White

One thing to keep in mind, is that the normal C128 default settings for the display has the attribute memory start right after the display memory. If you increase the display size, or scroll down by changing the display pointers, you must move the attribute pointer to make room. I suggest you place attributes at \$1000 to leave plenty of room (4096 bytes for display and attributes each). If you need more memory than that, you can turn off the attributes (the VDC will no longer look for them) by clearing the attribute enable control bit (register 25, bit 6). This would give you 8K for display area, but limit you to monochrome and using the first character set (as well as no underlining, blinking or reverse field).

Character Definitions

The VDC requires 16 bytes of definition for each character (verses 8 for the VIC-II) so that it can show as many, or as few, raster lines per character that you want. By default (on C128 power up), the VDC is set to display characters 8 pixels wide, and 8 pixels high in. This can be changed to make characters 5 pixels wide and 10 lines tall. The only limits are that characters cannot be taller than 32 lines (the VDC will switch to 32 byte character definitions if you define your characters to more than 16 lines) or wider than 8 pixels. You can also define a normal "space" between lines or characters by use of "interspace". The following figure shows the effect of interspace on display characters.



- F = Character Total Horizontal (R22 bits 7-4)
- G = Character Total Vertical (R9 bits 4-0)
- H = Character Displayed Horizontal (R22 bits 3-0)
- I = Character Displayed Vertical (R23 bits 4-0)
- J = Horizontal Interspace
- K = Vertical Interspace

The default settings of the VDC set no interspace between characters. By designing higher density characters, interspace might help make the display more readable. Interspace could also be used to provide a blank area under the character for the underline.

-The Cursor

The VDC provides a hardware cursor. This means that you provide the cursor position (by memory address), turn the cursor on, and if it is within the display area, the cursor will be displayed. The cursor has 4 modes of display:

Register 10 - Cursor Mode

Value	Meaning
0	Solid Cursor (no blinking)
1	No cursor
2	Blinking Cursor (fast)
3	Blinking Cursor (normal)

The cursor can also be set to any height from a block to an underline. You set the starting raster line in register 10 (bits 4-0) and the ending raster line, plus one, in register 11 (bits 4-0). This could be used to show different types of typing modes, like an

insert mode in a word processor. The cursor color is set by the color of the character it is over.

Interlace and Video Mode

The VDC provides three methods of displaying the video information to the RGB monitor. The normal method (by default) is called non-interlaced. This means that the screen is sent 60 times a second and provides a normal display. Another method (used by normal TV's) is to send the even numbered raster scans first, followed by the odd numbered raster, which effectively doubles the resolution of the picture, but cuts the screen "speed" in half. By turning the interlace mode on, the VDC displays the same number of raster lines and duplicates them for the even and odd screens, but sends the second screen one half scan line down, so the lines are better "filled in."

The last display mode is called the interlaced sync and video mode. This causes the VDC to display characters in up to double the resolution, if your monitor can handle it. Try turning on this mode and setting the number of raster lines per character (register 9) to a value greater than 8.

Graphics

The VDC offers a graphics mode that uses the VDC memory as a bit map. Instead of taking the display area and using it as pointers to the character set, the display area is used to display straight binary data. It takes as many bytes to display a raster line as you have columns in the display (each byte equals 8 pixels). In normal display mode (25 lines of 80 columns) this would result in a 640 by 200 display screen. This screen requires 16,000 bytes, so no memory is left for the attributes. In fact, several graphics programs have already been written that allow you to draw on this graphics screen in mono-color mode.

The bit map mode, however, follows the same rules as the text display, so you can shrink the display size (freeing memory) and make use of attributes or smooth scrolling. In order to shrink the display, lower the value of the vertical displayed register (6), or the horizontal displayed register (1). By reducing the number of columns to 72, you could have a 576 by 200 graphic screen, and still have enough room for the attributes. The attributes still work only on a standard character size (8 by 8), so you would only have the equivalent foreground/background selection of the hi-res VIC-II chip, but with 80% more display area!

For those of you wanting to write printer "dump" programs of the bit-map screen, remember that the address pointers (registers 18 and 19) are always incremented after a read of the data register (31), so if you want straight raster data, you only have to specify the start of the bitmap which is stored in display start pointers (registers 12 and 13).

There are other features of the VDC, but more time will be needed to uncover the proper settings of the other registers to use them properly. For now, all I can suggest is to experiment with different registers. The following program was written for the COMAL 2.0 cartridge that allows you to interactively change the different register settings and see the results. For those of you who get The Transactor Disks, there is a package that adds commands to COMAL 2.0 for output on the 80 column screen.

```

0010 // delete "vdc'editor"
0020 // save "vdc'editor"
0030 // by David Stidolph
0040 //
0050 USE system
0060 textcolors(6,6,1)
0070 DIM value(0:47)
0080 DIM name$(47) OF 20, reg(47)
0090 DIM start(47), fin(47)
0100 IF NOT superchip THEN
0110 IF NOT c128'in'memory THEN
0120 link'c128
0130 ENDIF
0140 ENDIF
0150 USE c128
0160 popover
0170 PAGE
0180 PRINT " Initializing VDC display. . ."
0190 init80
0200 read'registers
0210 read'values
0220 display'screen
0230 PAGE
0240 PRINT " Sample text has been written to the 80 "
0250 PRINT " column screen. This program requires "
0260 PRINT " that you have two monitors or can swap "
0270 PRINT " between the 40 and 80 column displays. "
0280 PRINT
0290 helpscreen
0300 //
0310 PROC helpscreen
0320 PRINT " These keys allow you to edit the VDC "
0330 PRINT
0340 PRINT " CRSR-up Move pointer up one register "
0350 PRINT " CRSR-down Move pointer down one reg. "
0360 PRINT " CRSR-left Move pointer to left column "
0370 PRINT " CRSR-right Move pointer to right column "
0380 PRINT " STOP Make POPOVER selection "
0390 PRINT " + Add one to register value "
0400 PRINT " - Subtract one from register "
0410 PRINT " H List of commands "
0420 PRINT " I INPUT number for VDC register "
0430 PRINT " S Show all VDC registers "
0440 PRINT " W Write PROCedure to disk that "
0450 PRINT " will initialize the VDC to "
0460 PRINT " the current settings "
0470 PRINT " ← Restart VDC initialization "
0480 PRINT " Q Quit program "
0490 PRINT
0500 INPUT " Press RETURN to continue: "; dummy$
0510 ENDPROC helpscreen
0520 done:=FALSE
0530 pos:=1
0540 PAGE
0550 display'registers
0560 REPEAT
0570 change'registers
0580 UNTIL done
0590 quit
0600 //
0610 PROC quit
0620 show'registers
0630 textcolors(-1,-1,13)
0640 PRINT " please send these settings to: "
0650 PRINT " COMAL Users Group, USA Limited "
0660 PRINT " 6041 Monona Drive "
0670 PRINT " Madison, WI 53716 "
0680 textcolors(-1,-1,1)
0690 END " " 145 " " 145 " "
0700 ENDPROC quit
0710 //
0720 PROC display'screen
0730 background80(0,0,0,0) // back=black
0740 color80(1,1,1,1) // char=white
0750 display80(1,42)
0760 page80
0770 FOR x:=1 TO 80 DO
0780 IF x<10 THEN
0790 print80(1,x," Line 1 - "(x))
0800 ELSE
0810 print80(1,x,CHR$(x DIV 10)+48)
0820 ENDIF
0830 print80(2,x,CHR$(x MOD 10)+48)
0840 ENDFOR x
0850 FOR x:=3 TO 40 DO
0860 print80(x,1," Line " +STR$(x))
0870 print80(x,71," Right side ")
0880 ENDFOR x
0890 //
0900 FOR i:=0 TO 1 DO
0910 color80(0,1,0,1) // lt. green
0920 print80(4,i*30+10," Color Table - Intensity " +state$(i))
0930 print80(5,i*30+10," ===== ")
0940 print80(6,i*30+10," Red Green Blue COLOR ")
0950 row:=7
0960 FOR r:=0 TO 1 DO
0970 FOR g:=0 TO 1 DO
0980 FOR b:=0 TO 1 DO
0990 color80(0,1,1,1) // lt cyan
1000 print80(row,i*30+10,state$(r))
1010 print80(row,i*30+16,state$(g))
1020 print80(row,i*30+23,state$(b))
1030 color80(r,g,b,i)
1040 print80(row,i*30+29," color ")
1050 row:=+1
1060 ENDFOR b
1070 ENDFOR g
1080 ENDFOR r
1090 ENDFOR i
1100 color80(1,1,0,1) // yellow
1110 attributes(1,0,0,1)
1120 print80(17,27," This text is blinking ")
1130 attributes(1,1,0,0)
1140 print80(19,27," This text is reversed ")
1150 attributes(1,0,1,0)
1160 print80(21,26," This text is underlined ")
1170 attributes(1,1,0,1)
1180 print80(23,26," Reversed, Blinking text ")
1190 attributes(1,0,1,1)
1200 print80(25,25," Blinking, Underlined text ")
1210 attributes(1,0,0,0)
1220 display80(1,25)
1230 ENDPROC display'screen
1240 //
1250 FUNC state$(t)
1260 IF t THEN
1270 RETURN " ON "
1280 ELSE
1290 RETURN " OFF "
1300 ENDIF
1310 ENDFUNC state$
1320 //
1330 PROC read'registers CLOSED
1340 IMPORT name$(0,reg(0))
1350 IMPORT start(0),fin(0)
1360 count:=0
1370 WHILE NOT EOD DO
1380 count:=+1
1390 READ name$(count)
1400 READ reg(count)
1410 READ start(count)
1420 READ fin(count)
1430 ENDWHILE
1440 //
1450 DATA " Horizontal Total " ,0,0,7
1460 DATA " Horiz Displayed " ,1,0,7
1470 DATA " Horiz Sync Pos " ,2,0,7
1480 DATA " Vert Sync Width " ,3,0,3
1490 DATA " Horiz Sync Width " ,3,4,7
1500 DATA " Vertical Total " ,4,0,7
1510 DATA " Vert Fine Adjust " ,5,0,4
1520 DATA " Vert Displayed " ,6,0,7
1530 DATA " Vert Sync Pos " ,7,0,7
1540 DATA " Interlace Mode " ,8,0,0
1550 DATA " Video Mode " ,8,1,1
1560 DATA " Lines Per Char " ,9,0,4
1570 DATA " Cursor Begin " ,10,0,4
1580 DATA " Cursor End " ,11,0,4
1590 DATA " Cursor Mode " ,10,5,6
1600 DATA " Text Start HI " ,12,0,7
1610 DATA " Text Start LO " ,13,0,7
1620 DATA " Cursor Addr HI " ,14,0,7
1630 DATA " Cursor Addr LO " ,15,0,7
1640 DATA " Memory Point HI " ,18,0,7
1650 DATA " Memory Point LO " ,19,0,7
1660 DATA " Attribute HI " ,20,0,7
1670 DATA " Attribute LO " ,21,0,7
1680 DATA " Char Total Width " ,22,4,7
1690 DATA " Char Disp Width " ,22,0,3
1700 DATA " Char Disp Vert " ,23,0,4
1710 DATA " Copy(1)/Fill(0) " ,24,7,7
1720 DATA " Reverse Screen " ,24,6,6
1730 DATA " Char Blink Rate " ,24,5,5
1740 DATA " V-Smooth Scroll " ,24,0,4
1750 DATA " H-Smooth Scroll " ,25,0,3
1760 DATA " Graphics/Text " ,25,7,7
1770 DATA " Mono/ATR - Color " ,25,6,6
1780 DATA " Semigraphic Mode " ,25,5,5
1790 DATA " DBL-Pixel Width " ,25,4,4
1800 DATA " Foreground Color " ,26,4,7
1810 DATA " Background Color " ,26,0,3
1820 DATA " Offset Per Row " ,27,0,7
1830 DATA " Character Base " ,28,5,7
1840 DATA " Underline Raster " ,29,0,4
1850 DATA " Word Count " ,30,0,7
1860 DATA " Data Register " ,31,0,7
1870 DATA " Copy Block HI " ,32,0,7
1880 DATA " Copy Block LO " ,33,0,7
1890 DATA " Disp Enable ON " ,30,0,7
1900 DATA " Disp Enable OFF " ,35,0,7
1910 DATA " DRAM Refresh " ,36,0,3
1920 ENDPROC read'registers
1930 //
1940 PROC display'registers
1950 PAGE
1960 FOR x:=1 TO 47 DO
1970 row:=x; col:=1
1980 IF x>24 THEN row:=-24; col:=21
1990 IF x=pos THEN
2000 PRINT AT row,col: " " 18 " ";name$(x); " " 146 " "
2010 ELSE
2020 PRINT AT row,col: " " +name$(x);
2030 ENDIF
2040 ENDFOR x
2050 textcolors(-1,-1,10)
2060 PRINT AT 24,22: " + ADD - SUBTRACT ",
2070 textcolors(-1,-1,1)
2080 PRINT AT 25,1: " " 18 " " +name$(pos)+ " ";value(pos);
2090 PRINT " " ,TAB(31)," MAX: ";
2100 PRINT USING " ### " 146 " ": max(pos),
2110 ENDPROC display'registers
2120 //
2130 PROC change'registers
2140 REPEAT
2150 new'pos:=pos
2160 valid:=TRUE
2170 CASE KEYS OF
2180 WHEN " q ", " Q "
2190 quit
2200 WHEN " "
2210 init80
2220 read'values
2230 display'screen
2240 WHEN " h ", " H "
2250 PAGE
2260 helpscreen
2270 display'registers
2280 WHEN " w ", " W "
2290 write'registers
2300 display'registers
2310 WHEN " i ", " I "
2320 REPEAT
2330 INPUT AT 25,24,3: " ": dat'num$,
2340 TRAP
2350 ok:=TRUE
2360 new'num:=VAL(dat'num$)
2370 HANDLER
2380 ok:=FALSE
2390 ENDTRAP
2400 UNTIL ok AND new'num>-1 AND new'num<=max(pos)
2410 value(pos):=new'num; make(pos)
2420 WHEN " s ", " S "
2430 show'registers
2440 PRINT
2450 INPUT " Press RETURN to continue: "; dummy$
2460 display'registers
2470 WHEN " " 17 " " // cursor down
2480 IF pos<47 THEN new'pos:=+1
2490 WHEN " " 145 " " // cursor up
2500 IF pos>1 THEN new'pos:=-1
2510 WHEN " " 29 " " // cursor right
2520 IF pos<24 THEN new'pos:=-24
2530 WHEN " " 157 " " // cursor left
2540 IF pos>24 THEN new'pos:=-24
2550 WHEN " + "
2560 IF value(pos)<max(pos) THEN value(pos):+1; make(pos)
2570 WHEN " - "
2580 IF value(pos)>0 THEN value(pos):-1; make(pos)
2590 WHEN CHR$(19) // home
2600 new'pos:=1
2610 OTHERWISE

```

```

2620 valid = FALSE
2630 ENDCASE
2640 IF new_pos<>pos THEN
2650 CURSOR (pos MOD 25) + (pos DIV 25),(pos DIV 25)*20 + 1
2660 PRINT " " + name$(pos);
2670 pos = new_pos
2680 CURSOR (pos MOD 25) + (pos DIV 25),(pos DIV 25)*20 + 1
2690 PRINT " " * 18 * " " + name$(pos); " " * 146 * " "
2700 valid = FALSE
2710 PRINT AT 25,1: " " * 18 * " " + name$(pos) + " " ;value(pos);
2720 PRINT " " ,TAB(31), "MAX:";
2730 PRINT USING "###" * 146 * " " : max(pos),
2740 ENDIF
2750 UNTIL valid
2760 PRINT AT 25,1: " " * 18 * " " + name$(pos) + " " ;value(pos);
2770 PRINT " " ,TAB(31), "MAX:";
2780 PRINT USING "###" * 146 * " " : max(pos),
2790 ENDPROC change_registers
2800 //
2810 FUNC max(r)
2820 RETURN 2*((fin(r)-start(r)) + 1)-1
2830 ENDFUNC max
2840 //
2850 PROC read_values
2860 FOR x = 1 TO 47 DO
2870 byte = read80(reg(x)); mask = 0
2880 FOR y = start(x) TO fin(x) DO
2890 mask = + 2ty
2900 ENDFOR y
2910 byte = byte BITAND mask
2920 value(x) = byte/(2*start(x))
2930 ENDFOR x
2940 ENDPROC read_values
2950 //
2960 PROC make(r)
2970 byte = read80(reg(r))
2980 mask = 0
2990 FOR y = start(r) TO fin(r) DO mask = + 2ty
3000 mask = 255 BITOR mask
3010 byte = byte BITAND mask
3020 byte = byte BITOR (value(r)*(2*start(r)))
3030 set80(reg(r),byte)
3040 ENDPROC make
3050 //
3060 PROC show_registers
3070 PAGE
3080 PRINT " " * 18 * "Current VDC values / CTRL-P for Hardcopy" * 146 * " "
3090 FOR x = 0 TO 36 DO
3100 CURSOR (x MOD 19) + 2,(x DIV 19)*20 + 1
3110 PRINT USING "Register ##: ###" : x,read80(x)
3120 ENDFOR x
3130 PRINT
3140 ENDPROC show_registers
3150 //
3160 PROC write_registers
3170 PAGE
3180 PRINT "Do you wish to write a PROCedure to"
3190 PRINT "disk that would set up the VDC registers"
3200 PRINT "to produce the present display?"
3210 PRINT
3220 REPEAT
3230 INPUT AT 0,0,1: "(Yes/No): y" * 157 * " : answer$,
3240 UNTIL answer$ IN "yYnN"
3250 PRINT
3260 PRINT
3270 IF answer$ IN "yY" THEN
3280 TRAP
3290 PRINT "writing the procedure: SET'VDC to"
3300 PRINT "the file: " * proc.set'vdc * " "
3310 PRINT
3320 OPEN FILE 5, "proc.set'vdc", WRITE
3330 PRINT FILE 5: "0100 //"
3340 PRINT FILE 5: "0110 proc set'vdc"
3350 PRINT FILE 5: "0120 use c128"
3360 FOR v = 0 TO 36 DO
3370 PRINT FILE 5: "0" + STR$(v*10 + 130);
3380 PRINT FILE 5: "set80(",v," ,",
3390 PRINT FILE 5: read80(v),")"
3400 ENDFOR v
3410 PRINT FILE 5: "0500 endproc set'vdc"
3420 CLOSE
3430 HANDLER
3440 CLOSE
3450 PRINT "Disk error: ";ERRTEXT$
3460 ENDTAP
3470 PRINT
3480 PRINT "Press RETURN to continue:";
3490 INPUT AT 0,0,0: " " : answer$
3500 ENDFIF
3510 ENDPROC write_registers
3520 //
3530 FUNC superchip CLOSED
3540 USE system
3550 setpage($84); chip = TRUE
3560 RESTORE chipdata
3570 FOR x = $8014 TO $8018 DO
3580 READ num
3590 IF PEEK(x)<>num THEN chip = FALSE
3600 ENDFOR x
3610 RETURN chip
3620 //
3630 chipdata:
3640 DATA 4,67,49,50,56
3650 ENDFUNC superchip
3660 //
3670 PROC link'c128
3680 PAGE
3690 DIM a$ OF 100
3700 a$ = "LINK " * pkg.c128 * " " * 13 * " "
3710 a$ = "RUN" * 13 * " "
3720 FOR x# = 1 TO LEN(a$) DO
3730 POKE 49151 + x#,ORD(a$(x#))
3740 ENDFOR x#
3750 POKE $c866,0
3760 POKE $c867,192
3770 POKE $c865,LEN(a$)
3780 STOP
3790 ENDPROC link'c128
3800 //
3810 FUNC c128'in'memory CLOSED
3820 USE system
3830 setpage($00); is'c128 = TRUE
3840 RESTORE c128data
3850 FOR x = $800e TO $8012 DO
3860 READ num
3870 IF PEEK(x)<>num THEN is'c128 = FALSE
3880 ENDFOR x
3890 RETURN is'c128
3900 //
3910 c128data:
3920 DATA 4,67,49,50,56
3930 ENDFUNC c128'in'memory
3940 //
3950 PROC popover CLOSED
3960 //copyright 1986 len lindsay
3970 //original by len lindsay
3980 INTERRUPT
3990 IMPORT quit,show_registers,read80
4000 IMPORT write_registers,helpscreen
4010 USE graphics
4020 TRAP ESC-
4030 setup
4040 USE system
4050 DIM start'screen$ OF 1505
4060 getscreen(start'screen$)
4070 clear'keys
4080 popup
4090 setscreen(start'screen$)
4100 clear'keys // optional line
4110 INTERRUPT popover
4120 //
4130 PROC popover
4140 col = RND(3,15)
4150 current'row = RND(2,9)
4160 PRINT AT row,col: " " * 18 * "-----"
4170 PRINT AT row,col: " " * 18 * " VDC Editor Menu "
4180 PRINT AT row,col: " " * 18 * "-----"
4190 PRINT AT row,col: " " * 18 * " C : Set Colors "
4200 PRINT AT row,col: " " * 18 * " ← : Restore VDC and "
4210 PRINT AT row,col: " " * 18 * " : 80 column screen "
4220 PRINT AT row,col: " " * 18 * " H : Help Screen "
4230 PRINT AT row,col: " " * 18 * " S : Show VDC values "
4240 PRINT AT row,col: " " * 18 * " W : Write MERGEable "
4250 PRINT AT row,col: " " * 18 * " : PROCedure to "
4260 PRINT AT row,col: " " * 18 * " : initialize VDC "
4270 PRINT AT row,col: " " * 18 * " Q : Quit Program "
4280 PRINT AT row,col: " " * 18 * "-----"
4290 PRINT AT row,col: " " * 18 * " RETURN to continue "
4300 PRINT AT row,col: " " * 18 * "-----"
4310 REPEAT
4320 done'popping = TRUE
4330 CASE KEY$ OF
4340 WHEN "c", "C"
4350 set'colors
4360 start'screen$(1) = CHR$(inq(1))
4370 start'screen$(2) = CHR$(inq(2))
4380 start'screen$(3) = CHR$(inq(3))
4390 done'popping = TRUE
4400 WHEN "q", "Q"
4410 quit
4420 WHEN "h", "H"
4430 PAGE
4440 helpscreen
4450 WHEN "s", "S"
4460 show_registers
4470 PRINT
4480 INPUT AT 0,0,0: " Press RETURN to
continue: " : dummy$,
4490 WHEN "w", "W"
4500 write_registers
4510 WHEN " " * 13 * " //carriage return
4520 RETURN
4530 OTHERWISE
4540 done'popping = FALSE
4550 ENDCASE
4560 UNTIL done'popping
4570 ENDPROC popover
4580 //
4600 FUNC row
4610 current'row = + 1
4620 RETURN current'row
4630 ENDFUNC row
4640 //
4650 PROC clear_keys
4660 WHILE KEY$> " " DO NULL
4670 dummyesc = ESC //clear stop key
4680 ENDPROC clear_keys
4690 //
4700 PROC setup CLOSED
4710 // setup by jesse knight
4720 TRAP ESC-
4730 FOR x# = 0 TO 12 DO
4740 READ byte#
4750 POKE $c86a + x#,byte#
4760 ENDFOR x#
4770 POKE $c7e2,$6a
4780 POKE $c7e3,$c8
4790 POKE $4d,PEEK($4d) BITOR $20
4800 DATA $a5,$4d,$29,$08,$f0,$06,$a9
4810 DATA $04,$05,$4d,$85,$4d,$60
4820 ENDPROC setup
4830 //
4840 PROC set_colors CLOSED
4850 USE system
4860 USE graphics
4870 PAGE
4880 LOOP
4890 PRINT AT 3,1: " " * 18 * "set colors now"
4900 PRINT AT 6,1: "press " * 18 * " f1 " * 146 * "
border color"
4910 PRINT AT 8,1: "press " * 18 * " f3 " * 146 * "
background color"
4920 PRINT AT 10,1: "press " * 18 * " f5 " * 146 * "
text color"
4930 PRINT AT 13,1: "press " * 18 * " f7 " * 146 * "
or " * 18 * " q " * 146 * " quit colors"
4940 CASE KEY$ OF
4950 WHEN " " * 133 * "
4960 textcolors((inq(1) + 1) MOD 16,-1,-1)
4970 WHEN " " * 134 * "
4980 textcolors(-1,(inq(2) + 1) MOD 16,-1)
4990 WHEN " " * 135 * "
5000 textcolors(-1,-1,(inq(3) + 1) MOD 16)
5010 WHEN " " * 136 * " , "q", "Q"
5020 EXIT
5030 OTHERWISE
5040 ENDCASE
5050 ENDLTOP
5060 ENDPROC set_colors
5070 ENDPROC popover

```

Keyboard Expander

Aubrey Stanley
Mississauga, Ontario

A Total Keyboard Utility For The Commodore 64

In our everyday life we use words to convey our thoughts. A thought, perfect and spontaneous, must nevertheless be articulated by a far from adequate speech process. Unless we use telepathy! In a manner of speaking, the keyboard replaces our vocal chords when we are dealing with the computer. We type in character after character to express a BASIC statement which had formed instantly in our mind to begin with.

We use the Screen Editor in Direct Mode to create program source in memory. And by executing BASIC statements directly, we can control our development environment. Keywords are entered over and over, and certain commands are used frequently. A Key-To-String capability would be of extreme value. We could pre-define keywords, op-codes, pokes, commands, even complete BASIC programs; for instant recall at the touch of a key. And with an interactive feature which paused while we typed in variable data, this function could be even more useful.

The Screen Editor would benefit from a few enhancements to take the toil out of editing programs. We could have an alternative Insert/Delete mode where we worked from the current cursor position instead opening up space in the text or moving the cursor past the final character to be deleted. A Tab key to move automatically back and forth through BASIC and DATA statements, and a Clear from the current cursor position, are also worthwhile candidates.

If we were given the choice, most of us would re-arrange at least some of the keys. Those shifted cursor functions might get relocated to other, unshifted positions. And DVORAK fans would likely re-arrange their entire keyboard.

We only have to browse through the back issues for a wealth of machine language routines designed to improve our development environment. Many of us could put our talents to good use in this area. Trouble is that these individually good programs refuse to co-exist under the same roof. They vie for system vectors, chase zero-page, run in the same RAM, etc. With a little adaptation, and a traffic cop to direct their execution, any one of them could be mobilized for instant action.

Welcome Keyboard Expander. Perhaps not a total utility, but it could well become a cornerstone in your development environment.

The Program

Keyboard Expander can make each key work for you in normal, shift, commodore and control mode. Reposition any key, recall a sequence of keystrokes, or run a routine of your choice. You may create your own configurations or use it in its pristine form for the many Screen Editor enhancements that it has to offer.

The program comes in the form of a loader which is run just like a BASIC program. It will relocate itself up against the top of **available** BASIC memory and then run transparently in the background while you go about your normal work. At this stage all the enhanced editing commands are available, mainly on the function keys.

If you wish to add more features, these are simply defined in a Profiling program as array assignments. Run this BASIC program and your keyboard will take on the profile you've given it. You also have the option to generate another version of Keyboard Expander with the new profile. This version can then be installed in future, instead of the original.

Repositioned Keys

This feature allows you to re-define as many keys as you wish to. The normal, shift, commodore, and control modes are treated quite independently, so you can move individual functions of keys around. Repositioned keys are always active, even in Run mode, allowing you to use them in your programs.

Strings

This feature allows you to assign an unlimited sequence of keystrokes to any key function. When you press the key, the string will instantly be printed to the screen and actioned, as if you had typed it in yourself. A special code can be encoded in the string to pause output while you type in some variable data such as a filename. Then press and release **[RUN/STOP]** to continue. Pressing **[RETURN]** instead, will terminate string output, actioning whatever is displayed on the line. Or you can press **[SHIFT] [RETURN]** to abort the process. The enhanced editing features are temporarily disabled while entering variable data.

The string feature is temporarily disabled in Run mode so as not to interfere with your programs, and also when the editor is in Quote mode so that graphics can be printed normally.

Routines

This feature allows you to tie a ML (machine language) interrupt routine to a key function. The interrupt routine will run in the keyboard (timer) interrupt. It may perform a specific task of its own or be used to initiate a background routine or a parallel program.

A Background Routine, when initiated, will run in every subsequent keyboard interrupt until it disables itself. It can also be started from within a running ML program in the BASIC environment.

A Parallel Program normally multitasks with BASIC by running after alternate interrupts. But it may elect to run exclusively for any period of time. It does not run interrupt code and therefore can do anything a

normal program can do. Like a background routine, it can also be started from a running ML program.

You can use the routine feature to execute your own commands. The command line is converted to ASCII and copied to your program space whenever **CTRL RETURN** is pressed.

Routines tied to keys will not be activated in Quote or Run mode. However, as mentioned, you may activate background or parallel programs in Run mode.

Compatibility Issues

Keyboard Expander will co-exist with other, (properly installed), programs. The word, *Install*, in the context of this article, refers to programs that allocate room for themselves at the top of available BASIC memory by modifying the top of memory pointer downward. Both POWER and PAL (Proline S/W Ltd) run without trouble when installed with Keyboard Expander, and probably SYMASS will too.

Keyboard Expander will not co-exist with programs that modify the keyboard and IRQ vectors (locations 655 and 788). You may well find other keyboard utilities practically redundant as Keyboard Expander allows you to incorporate new features quite easily. Except for these vectors, Keyboard Expander leaves the BASIC environment intact. And it does not use any zero page or free RAM. As far as possible, try to preserve this environment when creating your own configurations.

The memory it takes away from BASIC is 1K of tables and approximately 1.5K of code.

GETTING STARTED

First type in and save the generator program, "KE.GEN". Then run it to create "KE" on disk. Now load and run KE just like a normal BASIC program.

You'll see the start address and a copyright message. The start address is the one you want to SYS to in order to restart after a RunStop/Restore.

Four keys on the keyboard are intrinsic to the program. These are the **SHIFT**, **C=**, **CTRL** and **RUN/STOP** keys. Built-in commands have been mainly assigned to the function keys.

Click Mode: **CTRL C=**

In this mode each key will click as you type (if your volume is turned up). This command will toggle the mode on and off.

Repeats Mode: **CTRL SHIFT**

In this mode all keys will repeat. Again, the command toggles.

Disable/Enable Program: **CTRL RUN/STOP**

KE can be temporarily disabled with this command. The keyboard will then behave as it normally does until you press these keys again. Use this command when you need to print an original key, one that you've otherwise reassigned. However, if you are in quote mode, any string or program assignments you may have made are automatically disabled, so you don't need to disable the program in this case.

Terminate Program: **CTRL C= SHIFT**

KE can be entirely eliminated by pressing these three keys together. The keyboard will revert to its usual state. The memory used by the program will be released to BASIC memory space only if no other programs have been installed after KE. Otherwise the top of memory pointer is left untouched.

Clear Quote Mode: **CTRL HOME**

This command does exactly what it says. It becomes especially useful in KE, because when you're in quote mode all keys print to the screen, even keys that initiate the built-in commands or those you may have assigned to your own routines. This command also clears any inserts left over after you have used **INST** to open up space in the text.

Set Auto Insert/Delete Mode: **C= DEL**

In this mode the **INST** key is no longer active. Instead everything past the cursor on the BASIC line, (logical line of 80 characters), moves right as you type, and the new character is placed at the cursor position. When the BASIC line is full, characters will start to get overwritten.

The cursor and home keys behave as they normally do, so you can move around the text at will.

If repeat mode is on, then all keys will be repeated if held down. Strings assigned to keys, however, will not be repeated.

The **DEL** key behaves differently in this mode. Everything past the cursor, up to the end of the BASIC line, is moved left one position and overwrites the character at the cursor. In effect the key acts as a black hole through which you may drop unwanted characters. It will repeat if held down.

Clear Auto Insert/Delete Mode: **CTRL DEL**

Returns the editor to the normal insert/delete mode of operation.

Set Quote Mode: **C= HOME**

This could be useful when you are editing text between quotes as you then don't have to enter the quote character.

One Character Quote: **F3**

This only affects the next character typed. It is like being in quote mode for one character only.

Forward Tab: **F1**

This command is useful for tabbing through a BASIC or DATA line. Try it to see how it works.

Back Tab: **SHIFT F1**

This command works similarly to forward tab except that the movement is reversed.

Tab To End: **C= F1**

The cursor moves forward to the position past the last character on the BASIC line.

Tab To Start: **CTRL F1**

The cursor moves to the beginning of the BASIC line.

Clear To End Of Line: **SHIFT F3**

All characters on the BASIC line from the cursor onward are cleared.

Clear To End Of Screen: **C= F3**

All characters from the cursor to the end of the screen are cleared.

Screen Colour: **SHIFT F7**

Each time you enter this command the background colour of the screen will change to the next higher colour, i.e., black to white, etc. By using this command in conjunction with the following two for border and text, you should be able to set up an ideal colour combination.

Border Colour: **C= F7**

Increments the border colour.

Text Colour: **CTRL F7**

Increments the text colour. The entire text displayed on the screen changes instantly.

List Freezer: **CTRL F3**

Use this command to freeze the listing of a BASIC program as it flies past on the screen. After typing this command, if you hold the **CTRL** key down, the listing will continue to scroll at a slow rate. Releasing the key freezes the listing again. The **RUN/STOP** key also behaves similarly in this respect, except that the scroll will be faster.

It is really only useful for freezing a List, but will also freeze everything if you use it otherwise.

Get out of a freeze by pressing the **C=** key on its own.

Phosphor Saver: **CTRL F5**

So named because it blanks out the screen completely, as if it were powered off. It also does a complete freeze of anything that is running. Use the **C=** key to get out of this one too.

CONFIGURATIONS

To unlock the door to Keyboard Expander and add your own keyboard profile, you need to run a Profile program. But first a word about the possible memory configurations. There are several, depending on whether you are defining ML routines, whether these will reside in free RAM or are to be installed, and whether you are going to generate an absolute version to disk.

Configuration 1

The original program, KE, you have already seen. It is totally relocatable and generally useful to have around without the overhead of strings and routines.

Configuration 2

KE is again totally relocatable, but you are adding a profile to it, with your ML routines (if any) residing in free RAM. After installing KE, run your profile program. Do not select the option to generate an absolute version. This configuration is useful for testing out a new configuration.

Configuration 3

This is essentially the same as 2, except you want to install your ML routines. As the Profile program defines the absolute start address of each routine, the only way this is possible is for you to have assembled your routines into an absolute memory block at the *real* end of BASIC memory, which is normally \$9FFF. Install your program (or programs) before installing any other programs.

Configuration 4

This is the absolute version of 2. KE must be the first program installed so that it sits at the real top of BASIC memory. Run your profile program and this time choose to generate the absolute version to disk. In future you can install your own version, instead of KE. Be sure it is always the first program installed. Of course you still have to load any ML programs into free RAM.

Configuration 5

This is the absolute version of 3, and one that will find the most use. It is identical to 4 if there are no ML programs. ML programs must first be installed, then KE, thus concatenating the programs at the real end of BASIC memory. Now run the profile program and save the result to disk.

KEYPOWER FROM KEYBOARD EXPANDER

The configuration process will now be explained in terms of Keypower, a program we will generate along the lines of configuration 5.

Type in "KE.PF", the Profile Program, and save it under this name. This program is used to create profile programs by adding definitions in the line range, 100 to 799. It should never be modified. With KE.PF still in memory, type in "KPOWER.DEFS" and save it under the name, "KPOWER.PF".

We are going to include two useful ML routines. One is a BASIC Line Ruler for highlighting program lines and the other is a List Scrolling routine that I have adapted from an article in the TRANSACTOR, Vol 5, Issue 6, by Darren Spruyt. Type in and save the generator program, "SCROLL.GEN", then run it to generate "SCROLL.OBJ" on disk.

Type in and save "INSTALL" which is a general purpose program to install ML programs in memory.

To start with, no program should be installed, not even KE. If KE is the only program installed at this stage, you can use the "Terminate" command, described before, to eliminate it. Otherwise power up or otherwise reset the computer. Load and run INSTALL. Enter the filename, SCROLL.OBJ, when asked, and press return. Now load and run KE. Finally load and run KPOWER.DEFS. It will take about 10 seconds, then ask if you wish to create an absolute version. Follow the steps to do this, saving the file under the name, "KPOWER". You can use Keypower now, as it is in memory.

You must use INSTALL to install Keypower in future. Just loading it as an object program wont do the job. You can make INSTALL specific by replacing line 22 with:- F\$=KPOWER. Also remember that Keypower must always be the first program installed so that it loads at the top of real BASIC memory.

In Keypower, all the KE commands are still there, but now you have some additional ones.

Repositioned Keys

The Up and Left Arrow keys perform the Cursor Up and Cursor Left functions respectively. However, the shifted cursor keys retain their normal usage.

Read Error Channel: **CTRL E**

Prints and executes a one line (line 8000) BASIC program to read the disk error channel. The line is automatically deleted afterwards.

List Directory: **CTRL D**

Prints and executes a BASIC program (lines 8000-8008) which lists the disk directory directly to the screen. Press any key to pause, or Q to quit. The program lines are not deleted, so RUN 8000 will execute the program again. To delete the program lines press **CTRL Z**. If these lines clash with your BASIC program in memory, you can select the "TEST" BASIC Partition (described below).

This program was copied from an example in the POWER Manual.

Test Partition: **CTRL +**

This command creates a temporary BASIC partition immediately following any BASIC program currently in memory. You may then use the Test partition for whatever purpose. To release the TEST partition and return to your original BASIC program environment, press **CTRL** .

Load BASIC Program: **CTRL L**

This function recalls an interactive command string. You enter a variable filename to a (LOAD". . . .",8) command. As for all interactive strings, press and release **RUN/STOP** to continue after entering your data from the keyboard. Pressing **SHIFT RETURN** aborts the whole procedure. If you complete the command yourself (e.g., you may want a ML load instead - . . .",8,1), then the **RETURN** key on its own will both action what is printed on the line as well as terminate the remainder of the string output.

Run BASIC Program: **CTRL R**

Like the previous command, but also runs the program it loads.

Save BASIC Program: **CTRL S**

Interactive command to save a BASIC program.

Load From Directory: **CTRL G**

This function loads the file displayed in a normal directory listing when the cursor is positioned at the beginning of the displayed line. It is a good example of how a tedious manual process can be replaced by a string.

Initialize Drive: **CTRL I**

Initializes the disk drive.

Validate Disk: **CTRL V**

Validates the disk.

Rename File: **CTRL N**

This interactive command lets you enter the new filename and then the old filename. Press and release **RUN/STOP** after each filename is entered.

List Scroll Down: **F5**

This key, when held down, will scroll a BASIC listing on the screen (assuming the BASIC program is in memory). The listing will scroll downward and display BASIC lines in order of decreasing line numbers. If no BASIC lines are displayed on the screen, the listing will start with the highest numbered BASIC line.

WARNING: If a List is currently in progress, press **RUN/STOP** to stop the List before you use this function. This also applies for scrolling up.

List Scroll Up: **F7**

Like list scroll down, only the scroll is upward and lines are displayed in order of increasing line numbers.

The Scroll routines have been implemented as a Parallel program.

BASIC Line Ruler: **C= F5**

This command will put a border (much like a transparent ruler) around the BASIC line at the current cursor row. It is useful for scrutinizing the BASIC line underneath it. The ruler remains on the screen and will move in conjunction with the cursor.

This function can be used simultaneously with all the other functions described so far.

The routine has been implemented as a Background routine. While running, it uses the Tape Buffer (location 828) to generate the sprite data.

Kill Line Ruler: **CTRL F5**

This command terminates the Line Ruler function.

BASIC Keywords

Press the **C=** key in conjunction with the alphabet keys to print the following keywords:

ASC(CHR\$(DATA FOR GOTO INPUT LIST
MID\$(NEXT OPEN PRINT RIGHT\$STR\$(

Profile Update

With an absolute version like Keypower in memory, and no other program installed, it is possible to run a profile program as many time as you wish in order to add new definitions. You can even create disk

versions, but if strings have been replaced by new ones, there still will remain the memory overhead for the old ones. Then it's best, at some later stage, to create a fresh version.

BUILDING YOUR OWN PROFILE

As stated before, you must add your profile to KE.PF to generate a Profile program. Use line numbers in the range 100 to 799 and don't define any variables. Whatever you need to use has been declared in KE.PF. Refer to KPOWER.DEFS for examples.

Each key has its own variable to define it as shown in Table 1. Qualifying variables for the shift factors are shown in Table 3. Which group to use will depend on the sense in which a key is defined, the source key in an array or the target key for reposition.

Each key function has an entry in a two dimension array. The first dimension specifies the shift factor and the second the key. Each feature - reposition, string, routine - has its own array. There are also two variables for defining the start address of a "jump vector table" and a "command line" for ML programs. See Table 2.

When defining strings you will need to use the string variables shown in Table 4.

Repositioned Keys

These get defined in array **ZP()**. For example, ZP(SD,2)=PO, moves the QUOTE function to the unshifted POUND key. The QUOTE function can still be used on its original key as long as you don't make it the target of some other function. ZP(ND,UA)=LA+SF gives you the normal UP ARROW display on the shifted LEFT ARROW key. In the example below, the PLUS and MINUS keys have been completely interchanged.

```
ZP(ND,PL) = MI:ZP(SD,PL) = MI + SF:ZP(CD,PL) = MI +
CF:ZP(TD,PL) = MI + TF
ZP(ND,MI) = PL:ZP(SD,MI) = PL + SF:ZP(CD,MI) = PL +
CF:ZP(TD,MI) = PL + TD
```

Strings

These get defined in the array **ZS\$()**. There are some elementary rules to be followed.

Always start off a new string definition with **S\$**. For example,

```
ZS$(CD,L) = S$ + " LIST " + R$
```

will action the List command when you press **[C=]**.

If the complete string does not fit on the line, add in the remainder on following lines. For example

```
ZS$(TD,B) = ZS$(TD,B) + " REMAINDER "
```

will continue the string from the previous line.

If you define more than 255 characters in a string, you'll get a BASIC error. To get round this, you need to link the string to some other key. Suppose that the string on the original key, C = - D is near or at 255 characters. You may link it to the key, Y, as follows:

```
ZS$(CD,D) = ZS$(CD,D) + LN$ + CHR$(Y)
ZS$(ND,Y) = L$ + " LINKED STRING "
```

Note the linking value, **LN\$**, specifies a normal (unshifted) factor, and **CHR\$(Y)** has its usual sense, as in BASIC. On the second line, **L\$** is used as the first character in the linked string, instead of **S\$** which is used to define a new string. This is always so for a linked key.

Any number of keys may be linked in this way. A key you link to does not lose its original function, but you must **not** use it in another string or routine definition. For this reason, it is advisable to use the normal typing keys and save the shifted functions for strings or routines.

When defining an interactive string, you need to insert the variable, **I\$**, at the positions you wish to pause at. For example,

```
ZS$(SD,K) = S$ + " INSERT DISKETTE " + I$ + N$
ZS$(SD,K) = ZS$(SD,K) + " LOAD " + Q$ + " SYMASS " +
Q$ + ",8" + R$ + " RUN " + R$
```

Routines

These get defined in array **ZR()**. For example, ZR(TD,G)=49195, will run the interrupt routine located at 49195 when you press **[CTRL][G]**. If you are using background or parallel programs, then the variable, **ZV**, must be assigned the address of a 36 byte area for the "jump table". If you are implementing your own commands, the variable, **ZC**, must be assigned the address of a 40 byte area to hold the command line.

MACHINE LANGUAGE INTERFACE

With machine language you can realize the full potential of Keyboard Expander. The simplest type of ML program is the **Interrupt Routine** which you assign to a key. It will be entered as a subroutine within the Keyboard (timer) interrupt every time the key is pressed. The key matrix code (0 to 63), in case you need it, is passed in the .Y register and the shift state in the .X register. These values are also stored by the Kernel in locations 203 and 653.

For Background and Parallel programs, you will need to interface into Keyboard Expander through a "jump vector table". This table takes 36 bytes and contains 12 jump instructions. Only the first six are of use to you. Use the remaining 18 bytes for uninitialized data if you wish.

The jump table cannot be hard-coded because Keyboard Expander can move about in memory. You have to assign the start address of the table to the variable, **ZV**, in your profile program. When this program is run, Keyboard Expander will copy the table into your program space.

You will also need this table if you are initiating background routines or parallel programs from within a running ML program. Here you make use of the USER JUMP address at \$311/\$312. Store the address for your table in low/high byte format. Then change location \$310 from its normal \$4c ("jmp" instruction), to any other value. Keyboard Expander will detect the change and copy the table into your program space. You can be certain the table has been copied when \$310 contains the jump instruction again (within one timer interrupt period).

In PAL source code, your table should look something like this:


```

KEVECTOR = *      ;START ADDRESS
BRSTART  *= *+3   ;START BACKGROUND
BREND    *= *+3   ;END BACKGROUND
PPSTART  *= *+3   ;START PARALLEL
PPEXCL   *= *+3   ;RUN EXCLUSIVE
PPSHARE  *= *+3   ;RUN SHARED
PPEND    *= *+3   ;END PARALLEL
NOTUSED  *= *+18  ;NOT RELEVANT
    
```

Background Routines

JSR BRSTART will initiate a background routine from your interrupt routine or ML program, provided one is not already in existence. Pass the start address in the .X/.Y registers in low/high byte format. A zero status return means your routine has started.

The routine is coded as a subroutine to run within the keyboard interrupt. It will be called on every clock tick until it disables itself with a **JMP BREND**.

Parallel Programs

A Parallel program has the same start/end interface as the background routine, only use **PPSTART/PPEND** instead. When first started, it will multitask with BASIC by getting scheduled after every alternate timer interrupt. This is all right for a memory bound program, but if you're doing I/O to the disk, printer, etc, BASIC will be almost unusable. In this case and also if you need to lock out the keyboard, you must run exclusively by doing a JSR PPEXCL. When you want to multitask again do a JSR PPSHARE.

Command Line

The interrupt routine assigned to the **CTRL RETURN** function has the privilege of receiving the command line if you assign the start address of a 40 byte data area to the variable, ZC, in the profile program. On this keypress, the line will be **converted to ASCII** and copied into the assigned area before your routine is entered. This will allow you to implement commands like renumber, delete, merge, etc, by starting a parallel program if necessary.

Installing ML Programs

Procedures to do this have been discussed in the section on Keypower. Normally, as for SCROLL.OBJ, you shouldn't need to run any initialization code. But if you have to, the INSTALL program is smart enough to enter your initialization routine provided you start your program as follows:

```

JMP INIT      ;to init routine
.ASC ' 'INIT' ;special string
INIT (code)   ;init routine
    
```

CONCLUSION

Keyboard Expander works intimately with the Kernel keyboard interrupt to provide the interface through which you can manipulate any key to your advantage. It has the potential and flexibility to become whatever you want it to be, even a total keyboard utility. So if you do develop any useful routines for it, don't be shy. Send them to the TRANSACTOR so that we all can enjoy them. I'll really look forward to that.

TABLE 1 - KEY VARIABLES

KEYS	VARIABLES
[A] to [Z]	A to Z
[0] to [9]	N0 to N9
[F1] [F3] [F5] [F7]	F1,F3,F5,F7
[DEL]	DE
[RETURN]	RE
[CURSOR RIGHT]	RI
[CURSOR DOWN]	DO
[UP ARROW]	UA
[LEFT ARROW]	LA
[POUND]	PO
[HOME]	HO
[SPACE]	SP
[@] [;]	AT SE
[I] [L]	CM SL
[+] [-] [=]	PL MI EQ
[*] [.] [:]	AS PE CO

TABLE 2 - ARRAY/STORAGE

STORAGE	DESCRIPTION
ZP(3,62)	REPOSITIONED KEYS
ZR(3,62)	ROUTINES
ZS\$(3,62)	STRINGS
ZV	JUMP VECTOR TABLE
ZC	COMMAND LINE

TABLE 3 - SHIFT FACTOR

VAR	USE	VAR	USE	MODE
ND	ARRAY	--	----	NORMAL
SD	ARRAY	SF	TARGET	SHIFT
CD	ARRAY	CF	TARGET	COMMODORE
TD	ARRAY	TF	TARGET	CONTROL

TABLE 4 - VARIABLES USED IN DEFINING STRINGS

VAR	DESCRIPTION
S\$	Always the first in a new string
L\$	Always the first in a linked string
R\$	To encode a RETURN function
N\$	To encode a SHIFT RETURN function
Q\$	To encode a QUOTE character
LN\$	Normal factor when linking to another key
LS\$	Shift factor when linking to another key
LC\$	Commodore factor when linking to another key
LT\$	Control factor when linking to another key

Keyboard Expander: Generates PRG file "KE" on disk.

GN	1000 rem program to create file "ke" on disk	LJ	1620 data 97, 0, 0, 1, 135, 4, 97, 0
OB	1010 rem first loop ensures checksum is correct	LD	1630 data 0, 1, 188, 4, 97, 0, 0, 1
GP	1020 rem second loop writes file	PI	1640 data 248, 4, 97, 0, 0, 1, 29, 5
AM	1030 for j= 1 to 2745 : read x : ch = ch + x : next	CH	1650 data 97, 0, 0, 1, 42, 4, 97, 0
NO	1040 if ch<>276146 then print "checksum error" : end	KD	1660 data 0, 1, 115, 4, 97, 0, 0, 1
CD	1050 restore	FI	1670 data 82, 4, 97, 0, 0, 1, 72, 5
GN	1060 open 8,8,8,"0:ke.p,w"	GL	1680 data 4, 124, 0, 97, 0, 0, 1, 42
BI	1070 for j= 1 to 2745 : read x	MM	1690 data 5, 4, 72, 3, 97, 0, 0, 1
IL	1080 print#8,chr\$(x); : next	OP	1700 data 37, 4, 4, 21, 4, 97, 169, 191
CJ	1090 close#8 : end	MC	1710 data 129, 76, 167, 5, 97, 169, 64, 129
IL	1100 data 1, 8, 13, 8, 10, 0, 158, 50	JH	1720 data 32, 157, 5, 97, 169, 0, 97, 133
GI	1110 data 48, 54, 51, 58, 162, 0, 0, 0	GJ	1730 data 212, 97, 240, 7, 97, 169, 1, 97
LK	1120 data 32, 62, 9, 32, 75, 9, 32, 75	FD	1740 data 133, 212, 96, 96, 97, 169, 1, 97
PH	1130 data 9, 32, 75, 9, 201, 129, 240, 6	BC	1750 data 133, 216, 96, 96, 98, 238, 32, 208
AI	1140 data 169, 70, 32, 210, 255, 96, 165, 25	FH	1760 data 96, 96, 98, 238, 33, 208, 96, 96
JD	1150 data 240, 39, 32, 75, 9, 133, 53, 32	NG	1770 data 98, 206, 134, 2, 98, 173, 134, 2
DP	1160 data 75, 9, 133, 54, 165, 55, 133, 33	FF	1780 data 97, 162, 4, 97, 160, 216, 97, 132
FM	1170 data 56, 229, 53, 133, 55, 133, 27, 133	FF	1790 data 246, 97, 160, 0, 97, 132, 245, 97
IL	1180 data 51, 133, 53, 165, 56, 133, 34, 229	HL	1800 data 145, 245, 96, 200, 97, 208, 251, 97
NM	1190 data 54, 133, 56, 133, 28, 133, 52, 133	PM	1810 data 230, 246, 96, 202, 97, 208, 246, 96
IO	1200 data 54, 165, 27, 133, 29, 165, 28, 133	JH	1820 data 96, 97, 166, 214, 97, 134, 245, 96
KB	1210 data 30, 169, 0, 168, 162, 5, 145, 29	PL	1830 data 232, 97, 181, 217, 97, 48, 3, 96
AL	1220 data 200, 208, 251, 230, 30, 202, 208, 246	JM	1840 data 232, 97, 208, 249, 97, 224, 25, 97
AO	1230 data 169, 3, 24, 101, 27, 133, 29, 169	LG	1850 data 176, 12, 97, 181, 217, 97, 9, 128
IM	1240 data 0, 101, 28, 133, 30, 160, 0, 169	JN	1860 data 97, 149, 217, 98, 32, 255, 233, 96
HC	1250 data 75, 145, 29, 200, 169, 69, 145, 29	HC	1870 data 232, 97, 208, 240, 97, 166, 245, 98
PJ	1260 data 200, 169, 88, 145, 29, 200, 169, 80	EI	1880 data 32, 240, 233, 97, 164, 213, 98, 32
KM	1270 data 145, 29, 200, 200, 200, 200, 165	KM	1890 data 36, 234, 97, 169, 32, 97, 145, 209
MC	1280 data 27, 145, 29, 200, 165, 28, 145, 29	OK	1900 data 98, 32, 218, 228, 96, 136, 97, 48
NI	1290 data 200, 165, 33, 145, 29, 200, 165, 34	CK	1910 data 4, 97, 196, 211, 97, 176, 242, 96
NF	1300 data 145, 29, 165, 27, 133, 29, 165, 28	CN	1920 data 96, 97, 164, 211, 97, 196, 213, 97
FL	1310 data 133, 30, 160, 0, 32, 75, 9, 133	JB	1930 data 240, 81, 97, 132, 245, 97, 166, 214
CD	1320 data 26, 201, 5, 240, 33, 201, 128, 144	MO	1940 data 129, 32, 234, 4, 97, 208, 9, 129
KE	1330 data 3, 32, 42, 9, 165, 26, 41, 127	EB	1950 data 32, 174, 4, 97, 176, 15, 97, 240
MF	1340 data 201, 4, 208, 21, 32, 75, 9, 24	PN	1960 data 249, 97, 208, 59, 129, 32, 174, 4
LH	1350 data 101, 27, 133, 29, 32, 75, 9, 101	MC	1970 data 97, 176, 6, 97, 208, 249, 97, 132
PI	1360 data 28, 133, 30, 76, 171, 8, 76, 103	FJ	1980 data 245, 97, 240, 236, 97, 164, 245, 97
GD	1370 data 9, 201, 1, 208, 20, 32, 75, 9	AA	1990 data 16, 44, 96, 200, 97, 196, 213, 97
KJ	1380 data 24, 101, 27, 32, 49, 9, 32, 75	BB	2000 data 240, 55, 97, 176, 66, 97, 192, 40
LM	1390 data 9, 101, 28, 32, 49, 9, 76, 171	PC	2010 data 97, 208, 49, 96, 232, 97, 208, 46
DG	1400 data 8, 201, 2, 208, 15, 32, 75, 9	AF	2020 data 97, 164, 211, 97, 240, 30, 97, 166
IL	1410 data 24, 101, 27, 32, 49, 9, 32, 75	IB	2030 data 214, 129, 32, 223, 4, 97, 176, 19
GL	1420 data 9, 76, 171, 8, 201, 3, 208, 17	BM	2040 data 97, 208, 7, 129, 32, 223, 4, 97
AN	1430 data 32, 75, 9, 24, 101, 27, 32, 75	NI	2050 data 176, 12, 97, 240, 249, 129, 32, 223
NP	1440 data 9, 101, 28, 32, 49, 9, 76, 171	KF	2060 data 4, 97, 176, 5, 97, 208, 249, 129
BM	1450 data 8, 201, 96, 48, 13, 41, 159, 170	CF	2070 data 32, 174, 4, 97, 134, 214, 97, 132
HK	1460 data 232, 32, 42, 9, 202, 208, 250, 76	OH	2080 data 211, 96, 96, 96, 136, 97, 208, 3
HB	1470 data 171, 8, 0, 32, 75, 9, 32, 49	KM	2090 data 96, 56, 97, 176, 18, 97, 192, 39
DA	1480 data 9, 96, 8, 160, 0, 145, 29, 230	PL	2100 data 97, 208, 1, 96, 202, 97, 177, 209
NC	1490 data 29, 208, 2, 230, 30, 40, 96, 169	IJ	2110 data 97, 201, 58, 97, 240, 6, 97, 201
LB	1500 data 106, 133, 31, 169, 9, 133, 32, 169	GP	2120 data 32, 97, 240, 2, 97, 201, 44, 96
JG	1510 data 1, 133, 25, 96, 8, 165, 31, 197	FL	2130 data 24, 96, 96, 97, 164, 213, 97, 166
MC	1520 data 45, 165, 32, 229, 46, 176, 13, 160	CN	2140 data 214, 97, 192, 40, 97, 144, 7, 97
EO	1530 data 0, 177, 31, 230, 31, 208, 7, 230	AG	2150 data 165, 211, 97, 201, 40, 97, 176, 1
JD	1540 data 32, 76, 101, 9, 169, 5, 40, 96	FK	2160 data 96, 232, 129, 32, 234, 4, 97, 201
JB	1550 data 108, 27, 0, 0, 16, 129, 42, 10	LF	2170 data 32, 97, 208, 204, 129, 32, 223, 4
IE	1560 data 129, 76, 188, 9, 99, 75, 69, 88	EA	2180 data 97, 176, 4, 97, 201, 32, 97, 240
BF	1570 data 80, 1, 16, 0, 4, 24, 0, 97	FN	2190 data 247, 97, 201, 32, 97, 240, 191, 97
KD	1580 data 0, 0, 1, 26, 4, 97, 0, 0	KA	2200 data 208, 186, 97, 166, 214, 97, 164, 211
CB	1590 data 1, 21, 4, 4, 68, 0, 97, 0	PO	2210 data 97, 192, 40, 97, 144, 1, 96, 202
IF	1600 data 0, 1, 51, 4, 97, 0, 0, 1	NB	2220 data 97, 160, 0, 97, 240, 176, 98, 172
BF	1610 data 47, 4, 97, 0, 0, 1, 55, 4	KO	2230 data 32, 208, 97, 169, 16, 98, 141, 32
		EA	2240 data 208, 98, 77, 17, 208, 98, 141, 17
		JP	2250 data 208, 129, 32, 104, 5, 97, 224, 223

CG	2260 data	97, 208, 249, 98, 140, 32, 208, 97	KK	2900 data	9, 4, 96, 104, 129, 141, 10, 4
EM	2270 data	9, 16, 98, 141, 17, 208, 96, 96	JJ	2910 data	96, 104, 129, 141, 11, 4, 96, 104
AC	2280 data	129, 32, 165, 5, 98, 238, 32, 208	OP	2920 data	129, 141, 12, 4, 129, 173, 8, 4
KB	2290 data	129, 32, 135, 5, 129, 32, 104, 5	FM	2930 data	96, 72, 129, 173, 7, 4, 96, 72
AH	2300 data	96, 138, 97, 41, 164, 97, 201, 164	BN	2940 data	129, 173, 6, 4, 96, 72, 129, 173
LL	2310 data	97, 240, 246, 97, 201, 132, 97, 208	KM	2950 data	5, 4, 96, 72, 129, 173, 4, 4
JA	2320 data	6, 98, 206, 32, 208, 129, 32, 174	PN	2960 data	96, 72, 129, 173, 3, 4, 96, 72
MF	2330 data	5, 98, 76, 126, 234, 97, 162, 127	HD	2970 data	129, 173, 12, 4, 129, 141, 8, 4
NG	2340 data	98, 142, 0, 220, 98, 174, 1, 220	JD	2980 data	129, 173, 11, 4, 129, 141, 7, 4
JO	2350 data	96, 96, 96, 72, 97, 169, 32, 129	LD	2990 data	129, 173, 10, 4, 129, 141, 6, 4
DO	2360 data	45, 15, 4, 97, 240, 12, 97, 160	GO	3000 data	129, 173, 9, 4, 129, 141, 5, 4
NG	2370 data	15, 98, 140, 24, 212, 96, 168, 96	NM	3010 data	129, 142, 4, 4, 129, 140, 3, 4
LA	2380 data	136, 97, 208, 253, 98, 140, 24, 212	PC	3020 data	129, 108, 0, 4, 129, 108, 13, 4
GO	2390 data	96, 104, 96, 96, 96, 104, 96, 24	OC	3030 data	129, 173, 14, 4, 97, 208, 8, 129
NK	2400 data	97, 105, 1, 129, 141, 0, 4, 96	FF	3040 data	142, 13, 4, 129, 140, 14, 4, 97
KP	2410 data	104, 97, 105, 0, 129, 141, 1, 4	KN	3050 data	169, 0, 96, 96, 97, 169, 0, 129
IB	2420 data	97, 169, 8, 129, 45, 15, 4, 97	BI	3060 data	141, 14, 4, 96, 96, 97, 169, 129
ED	2430 data	208, 7, 97, 169, 8, 129, 13, 15	NG	3070 data	97, 208, 2, 97, 169, 1, 129, 141
HD	2440 data	4, 97, 208, 8, 98, 76, 126, 234	HG	3080 data	2, 4, 96, 96, 97, 169, 65, 129
NB	2450 data	97, 169, 247, 129, 45, 15, 4, 129	GI	3090 data	141, 2, 4, 129, 76, 249, 6, 97
NA	2460 data	141, 15, 4, 96, 96, 130, 162, 80	CF	3100 data	169, 1, 129, 44, 2, 4, 97, 208
AB	2470 data	6, 131, 160, 80, 6, 98, 142, 20	JA	3110 data	15, 129, 142, 7, 4, 129, 140, 8
IL	2480 data	3, 98, 140, 21, 3, 97, 162, 49	FD	3120 data	4, 97, 162, 0, 129, 142, 6, 4
CB	2490 data	97, 160, 234, 129, 142, 0, 4, 129	JA	3130 data	129, 141, 2, 4, 96, 138, 96, 96
KF	2500 data	140, 1, 4, 96, 96, 130, 162, 79	HC	3140 data	98, 174, 141, 2, 97, 224, 3, 97
FG	2510 data	9, 131, 160, 79, 9, 97, 208, 10	FI	3150 data	240, 113, 97, 224, 7, 97, 208, 72
EA	2520 data	130, 162, 19, 7, 131, 160, 19, 7	KL	3160 data	129, 173, 15, 0, 97, 240, 12, 97
MH	2530 data	97, 208, 4, 97, 162, 72, 97, 160	GI	3170 data	169, 0, 129, 141, 13, 0, 97, 169
OF	2540 data	235, 98, 142, 143, 2, 98, 140, 144	MG	3180 data	160, 129, 141, 14, 0, 97, 208, 14
BI	2550 data	2, 96, 96, 97, 165, 207, 97, 240	HN	3190 data	97, 165, 55, 129, 205, 11, 0, 97
FO	2560 data	12, 97, 165, 206, 98, 174, 135, 2	NJ	3200 data	208, 33, 97, 165, 56, 129, 205, 12
LI	2570 data	97, 160, 0, 97, 132, 207, 98, 32	PA	3210 data	0, 97, 208, 26, 129, 174, 13, 0
GH	2580 data	19, 234, 96, 96, 97, 165, 203, 97	FP	3220 data	97, 134, 55, 97, 134, 51, 129, 172
AH	2590 data	201, 63, 97, 176, 75, 97, 160, 0	DF	3230 data	14, 0, 97, 132, 56, 97, 132, 52
AN	2600 data	96, 10, 96, 10, 96, 10, 97, 144	NA	3240 data	97, 166, 45, 97, 164, 46, 97, 134
LM	2610 data	2, 96, 200, 96, 200, 96, 10, 97	JE	3250 data	47, 97, 132, 48, 97, 134, 49, 97
NC	2620 data	144, 2, 96, 200, 96, 24, 130, 105	NN	3260 data	132, 50, 129, 32, 207, 5, 97, 162
KH	2630 data	16, 0, 97, 133, 245, 96, 152, 131	GP	3270 data	49, 97, 160, 234, 98, 142, 20, 3
NJ	2640 data	105, 16, 0, 97, 133, 246, 97, 169	FL	3280 data	98, 140, 21, 3, 97, 208, 34, 97
OL	2650 data	0, 98, 174, 141, 2, 97, 240, 13	AB	3290 data	165, 203, 97, 224, 4, 97, 208, 34
FB	2660 data	96, 24, 97, 105, 4, 96, 202, 97	JH	3300 data	97, 197, 197, 97, 240, 30, 97, 201
DA	2670 data	240, 7, 97, 105, 4, 96, 202, 97	LK	3310 data	63, 97, 240, 9, 97, 201, 51, 97
LB	2680 data	240, 2, 97, 105, 4, 96, 168, 97	BC	3320 data	208, 22, 97, 133, 197, 129, 76, 31
EP	2690 data	177, 245, 97, 240, 27, 96, 170, 97	LK	3330 data	4, 97, 169, 128, 129, 77, 15, 4
JK	2700 data	41, 63, 97, 133, 203, 96, 138, 97	NE	3340 data	129, 141, 15, 4, 98, 142, 142, 2
NO	2710 data	162, 0, 97, 41, 192, 97, 240, 12	DN	3350 data	129, 32, 113, 5, 129, 76, 73, 8
KA	2720 data	96, 232, 97, 201, 64, 97, 240, 7	FP	3360 data	129, 44, 15, 4, 97, 48, 248, 98
MK	2730 data	96, 232, 97, 201, 128, 97, 240, 2	OC	3370 data	236, 142, 2, 97, 240, 23, 97, 224
PK	2740 data	96, 232, 96, 232, 98, 142, 141, 2	CN	3380 data	5, 97, 208, 11, 97, 169, 128, 98
ED	2750 data	96, 96, 96, 200, 97, 177, 245, 129	FL	3390 data	77, 138, 2, 98, 141, 138, 2, 129
DC	2760 data	141, 16, 4, 96, 200, 97, 177, 245	OB	3400 data	76, 133, 7, 97, 224, 6, 97, 208
NB	2770 data	129, 141, 17, 4, 96, 200, 97, 177	EK	3410 data	4, 97, 169, 32, 97, 208, 208, 129
AH	2780 data	245, 129, 141, 18, 4, 96, 96, 97	JL	3420 data	32, 235, 5, 129, 32, 61, 6, 97
GE	2790 data	169, 76, 98, 205, 16, 3, 97, 240	PK	3430 data	164, 203, 97, 169, 8, 129, 45, 15
EJ	2800 data	23, 98, 141, 16, 3, 98, 173, 17	FB	3440 data	4, 97, 240, 28, 97, 196, 197, 97
CL	2810 data	3, 97, 133, 245, 98, 173, 18, 3	LI	3450 data	208, 16, 97, 192, 64, 97, 240, 12
LM	2820 data	97, 133, 246, 97, 160, 35, 129, 185	GD	3460 data	129, 173, 20, 4, 97, 240, 6, 96
BE	2830 data	152, 9, 97, 145, 245, 96, 136, 97	CI	3470 data	152, 129, 206, 20, 4, 97, 240, 22
FH	2840 data	16, 248, 129, 173, 14, 4, 97, 240	DB	3480 data	96, 96, 129, 32, 165, 5, 97, 169
PE	2850 data	3, 129, 32, 211, 6, 97, 169, 1	CB	3490 data	0, 129, 141, 20, 4, 96, 152, 97
IH	2860 data	129, 44, 2, 4, 97, 240, 83, 97	LE	3500 data	201, 63, 97, 176, 106, 97, 197, 197
EF	2870 data	48, 81, 97, 80, 5, 97, 169, 0	EP	3510 data	97, 240, 102, 97, 36, 157, 97, 16
GJ	2880 data	129, 141, 2, 4, 96, 104, 96, 168	EB	3520 data	98, 129, 32, 113, 5, 97, 166, 212
PN	2890 data	96, 104, 96, 170, 96, 104, 129, 141	IF	3530 data	97, 208, 86, 129, 174, 18, 4, 97

OB	3540 data 240, 81, 129, 174, 16, 4, 97, 240
KI	3550 data 12, 129, 32, 102, 9, 97, 160, 0
HH	3560 data 97, 177, 245, 97, 240, 67, 129, 76
PC	3570 data 184, 8, 97, 133, 197, 97, 201, 1
CH	3580 data 97, 208, 41, 97, 169, 4, 98, 205
GO	3590 data 141, 2, 97, 208, 34, 129, 173, 10
AE	3600 data 0, 97, 240, 29, 97, 133, 246, 129
CL	3610 data 173, 9, 0, 97, 133, 245, 97, 169
CK	3620 data 0, 129, 141, 19, 4, 97, 160, 39
CK	3630 data 97, 177, 209, 129, 32, 113, 9, 97
OL	3640 data 145, 245, 96, 136, 97, 16, 246, 97
GI	3650 data 162, 141, 98, 32, 38, 235, 129, 32
NI	3660 data 155, 5, 129, 32, 218, 5, 97, 164
DM	3670 data 203, 98, 174, 141, 2, 98, 142, 142
LP	3680 data 2, 129, 108, 17, 4, 129, 44, 15
KK	3690 data 4, 97, 112, 3, 98, 76, 72, 235
LL	3700 data 129, 32, 165, 5, 97, 165, 212, 97
GE	3710 data 133, 245, 97, 165, 203, 97, 208, 10
HG	3720 data 98, 174, 141, 2, 97, 240, 1, 96
HB	3730 data 96, 97, 162, 29, 97, 208, 23, 97
PL	3740 data 166, 212, 97, 208, 12, 97, 201, 7
JO	3750 data 97, 240, 224, 97, 201, 3, 97, 144
PO	3760 data 220, 97, 201, 51, 97, 240, 216, 97
DE	3770 data 162, 148, 98, 44, 138, 2, 97, 16
BH	3780 data 11, 97, 160, 4, 97, 197, 197, 97
EJ	3790 data 240, 2, 97, 160, 32, 129, 140, 20
CN	3800 data 4, 129, 32, 31, 4, 98, 32, 38
PD	3810 data 235, 129, 32, 135, 5, 97, 165, 245
FE	3820 data 97, 133, 212, 97, 230, 197, 98, 32
HC	3830 data 72, 235, 129, 32, 174, 5, 129, 76
MC	3840 data 148, 5, 97, 201, 3, 97, 208, 2
CP	3850 data 97, 169, 0, 98, 141, 141, 2, 96
MH	3860 data 200, 97, 177, 245, 97, 133, 203, 129
IM	3870 data 32, 235, 5, 129, 32, 61, 6, 129
AK	3880 data 206, 16, 4, 97, 240, 87, 97, 162
ME	3890 data 1, 97, 208, 2, 97, 162, 0, 129
KJ	3900 data 142, 19, 4, 97, 162, 255, 98, 142
DG	3910 data 3, 220, 129, 32, 102, 9, 97, 208
NI	3920 data 3, 129, 32, 135, 5, 97, 165, 198
IO	3930 data 97, 201, 10, 97, 176, 247, 129, 172
HA	3940 data 19, 4, 97, 177, 245, 97, 240, 65
EL	3950 data 97, 201, 5, 97, 144, 192, 129, 44
KD	3960 data 15, 4, 97, 80, 26, 97, 165, 198
EN	3970 data 97, 240, 6, 129, 32, 135, 5, 129
FO	3980 data 76, 224, 8, 97, 133, 212, 97, 162
IN	3990 data 148, 98, 32, 38, 235, 129, 32, 135
AN	4000 data 5, 97, 165, 198, 97, 208, 249, 97
PA	4010 data 133, 216, 129, 172, 19, 4, 97, 177
DB	4020 data 245, 96, 170, 98, 32, 38, 235, 129
OF	4030 data 238, 19, 4, 129, 206, 16, 4, 97
JC	4040 data 208, 191, 129, 32, 135, 5, 97, 165
BG	4050 data 198, 97, 208, 249, 98, 141, 3, 220
PC	4060 data 129, 76, 146, 8, 129, 206, 16, 4
CI	4070 data 97, 240, 238, 129, 238, 19, 4, 129
HL	4080 data 32, 195, 5, 98, 140, 140, 2, 98
NC	4090 data 238, 32, 208, 129, 32, 135, 5, 97
FK	4100 data 165, 198, 97, 208, 249, 98, 141, 3
FC	4110 data 220, 129, 32, 135, 5, 129, 32, 104
JF	4120 data 5, 97, 48, 17, 129, 32, 104, 5
DF	4130 data 97, 16, 251, 98, 206, 32, 208, 129
AL	4140 data 32, 201, 5, 129, 32, 31, 4, 129
EO	4150 data 76, 189, 8, 98, 76, 52, 234, 129
FH	4160 data 32, 235, 5, 97, 165, 203, 97, 201
FI	4170 data 1, 97, 208, 11, 96, 104, 96, 104

MF	4180 data 98, 206, 32, 208, 129, 32, 201, 5
CF	4190 data 129, 76, 146, 8, 129, 76, 73, 8
PL	4200 data 129, 173, 17, 4, 97, 133, 245, 129
BB	4210 data 173, 18, 4, 97, 133, 246, 96, 96
BN	4220 data 129, 141, 16, 4, 97, 41, 63, 129
GK	4230 data 14, 16, 4, 129, 44, 16, 4, 97
DL	4240 data 16, 2, 97, 9, 128, 97, 144, 5
CG	4250 data 129, 174, 19, 4, 97, 208, 4, 97
NM	4260 data 112, 2, 97, 9, 64, 97, 201, 34
JK	4270 data 97, 208, 8, 129, 77, 19, 4, 129
GK	4280 data 141, 19, 4, 97, 169, 34, 96, 96
GC	4290 data 129, 76, 214, 6, 129, 76, 228, 6
ED	4300 data 129, 76, 252, 6, 129, 76, 234, 6
EF	4310 data 129, 76, 238, 6, 129, 76, 244, 6
NE	4320 data 129, 76, 218, 5, 129, 76, 135, 5
HG	4330 data 129, 76, 165, 5, 129, 76, 174, 5
FE	4340 data 129, 76, 201, 5, 129, 76, 235, 5
NH	4350 data 96, 120, 97, 169, 0, 97, 162, 20
HL	4360 data 129, 157, 0, 4, 96, 202, 97, 16
ND	4370 data 250, 129, 32, 174, 5, 129, 32, 201
CH	4380 data 5, 96, 88, 129, 173, 11, 0, 97
CI	4390 data 133, 99, 129, 173, 12, 0, 97, 133
KL	4400 data 98, 97, 162, 144, 96, 56, 98, 32
IL	4410 data 73, 188, 98, 32, 221, 189, 97, 160
JH	4420 data 1, 98, 185, 0, 1, 97, 240, 6
MH	4430 data 98, 32, 210, 255, 96, 200, 97, 208
HM	4440 data 245, 97, 169, 13, 98, 32, 210, 255
AB	4450 data 97, 162, 0, 129, 189, 1, 10, 98
JJ	4460 data 32, 210, 255, 96, 232, 97, 224, 41
CO	4470 data 97, 144, 245, 96, 96, 112, 75, 69
KB	4480 data 89, 66, 79, 65, 82, 68, 32, 69
DD	4490 data 88, 80, 65, 78, 68, 69, 82, 96
CE	4500 data 13, 118, 40, 67, 41, 32, 49, 57
EB	4510 data 56, 54, 32, 65, 85, 66, 82, 69
AE	4520 data 89, 32, 83, 84, 65, 78, 76, 69
EE	4530 data 89

Keyboard Expander: Generates PRG file "SCROLL.OBJ"

EK	1000 rem program to create file "scroll.obj" on disk
OB	1010 rem first loop ensures checksum is correct
GP	1020 rem second loop writes file
FA	1030 for j=1 to 741 : read x : ch=ch+x : next
PO	1040 if ch<>"86049" then print "checksum error" : end
CD	1050 restore
MJ	1060 open 8,8,8,"0:scroll.obj,p,w"
BI	1070 for j=1 to 2745 : read x
IL	1080 print#8,chr\$(x) : next
CJ	1090 close8 : end
AK	1100 data 0, 157, 76, 48, 157, 76, 64, 157
EP	1110 data 76, 43, 159, 76, 145, 159, 0, 0
OC	1120 data 0, 0, 0, 0, 0, 0, 0, 0
ID	1130 data 0, 0, 0, 0, 0, 0, 0, 0
CE	1140 data 0, 0, 0, 0, 0, 0, 0, 0
ME	1150 data 0, 0, 0, 0, 0, 0, 0, 0
KA	1160 data 0, 0, 32, 88, 157, 32, 176, 158
GL	1170 data 32, 114, 157, 32, 225, 158, 240, 245
DM	1180 data 208, 21, 32, 88, 157, 32, 176, 158
ND	1190 data 169, 24, 133, 214, 32, 108, 229, 32
GD	1200 data 243, 157, 32, 225, 158, 240, 238, 76
FF	1210 data 27, 157, 140, 33, 157, 142, 34, 157
KB	1220 data 169, 255, 141, 41, 157, 141, 42, 157
OH	1230 data 104, 24, 105, 1, 170, 104, 105, 0

KA 1240 data 168, 76, 18, 157, 162, 255, 232, 224
 JO 1250 data 25, 240, 11, 181, 217, 16, 247, 32
 BD 1260 data 124, 158, 144, 242, 176, 6, 169, 255
 KA 1270 data 133, 20, 133, 21, 32, 19, 166, 164
 PE 1280 data 95, 228, 44, 208, 5, 196, 43, 208
 BD 1290 data 1, 96, 202, 228, 44, 176, 5, 166
 LD 1300 data 44, 164, 43, 136, 134, 64, 132, 63
 LP 1310 data 160, 0, 177, 63, 240, 4, 200, 208
 PP 1320 data 249, 96, 200, 177, 63, 197, 95, 208
 EK 1330 data 245, 200, 177, 63, 197, 96, 208, 238
 EL 1340 data 136, 152, 24, 101, 63, 133, 95, 165
 IG 1350 data 64, 105, 0, 133, 96, 162, 24, 181
 DG 1360 data 217, 48, 5, 162, 23, 32, 255, 233
 MC 1370 data 162, 0, 32, 104, 233, 165, 217, 9
 LH 1380 data 128, 133, 217, 169, 39, 133, 213, 162
 NB 1390 data 1, 142, 146, 2, 202, 134, 214, 32
 IB 1400 data 240, 233, 76, 68, 158, 162, 25, 202
 KJ 1410 data 48, 11, 181, 217, 16, 249, 32, 124
 FN 1420 data 158, 144, 244, 176, 9, 169, 255, 133
 EG 1430 data 20, 133, 21, 72, 208, 21, 224, 24
 CK 1440 data 208, 4, 169, 13, 208, 245, 224, 23
 DA 1450 data 208, 5, 232, 181, 217, 16, 243, 169
 CD 1460 data 255, 208, 232, 230, 20, 208, 2, 230
 HO 1470 data 21, 32, 19, 166, 165, 218, 48, 9
 CL 1480 data 9, 128, 133, 218, 162, 1, 32, 255
 NO 1490 data 233, 162, 24, 134, 214, 32, 240, 233
 IA 1500 data 104, 48, 3, 32, 210, 255, 160, 1
 AJ 1510 data 177, 95, 240, 37, 141, 35, 157, 169
 PN 1520 data 0, 145, 95, 132, 15, 165, 95, 133
 MJ 1530 data 63, 165, 96, 133, 64, 169, 255, 133
 JP 1540 data 20, 133, 21, 76, 215, 166, 224, 11
 PC 1550 data 240, 13, 173, 35, 157, 160, 1, 145
 DF 1560 data 63, 169, 13, 32, 210, 255, 96, 104
 GD 1570 data 104, 104, 104, 76, 111, 158, 32, 240
 KE 1580 data 233, 160, 255, 200, 177, 209, 192, 39
 LJ 1590 data 240, 38, 201, 32, 240, 245, 201, 48
 AF 1600 data 144, 30, 201, 58, 176, 26, 152, 24
 MC 1610 data 101, 209, 133, 122, 165, 210, 105, 0
 LF 1620 data 133, 123, 32, 121, 0, 142, 36, 157
 BN 1630 data 32, 107, 169, 174, 36, 157, 56, 36
 DN 1640 data 24, 96, 120, 173, 0, 3, 141, 39
 LJ 1650 data 157, 173, 1, 3, 141, 40, 157, 169
 GI 1660 data 100, 141, 0, 3, 169, 158, 141, 1
 BK 1670 data 3, 32, 30, 157, 169, 255, 133, 204
 HL 1680 data 165, 214, 141, 37, 157, 165, 211, 141
 HN 1690 data 38, 157, 169, 0, 133, 211, 88, 32
 HN 1700 data 21, 157, 96, 120, 169, 0, 141, 146
 LA 1710 data 2, 173, 39, 157, 141, 0, 3, 173
 AH 1720 data 40, 157, 141, 1, 3, 169, 0, 133
 GE 1730 data 204, 174, 37, 157, 134, 214, 173, 38
 MO 1740 data 157, 133, 211, 56, 233, 40, 144, 2
 EI 1750 data 133, 211, 32, 108, 229, 88, 172, 33
 BA 1760 data 157, 174, 34, 157, 196, 197, 208, 3
 EO 1770 data 236, 142, 2, 96, 165, 207, 240, 12
 JP 1780 data 165, 206, 174, 135, 2, 160, 0, 132
 PF 1790 data 207, 32, 19, 234, 96, 162, 153, 160
 GO 1800 data 159, 32, 12, 157, 208, 92, 169, 0
 HM 1810 data 141, 46, 157, 141, 47, 157, 162, 63
 PE 1820 data 157, 64, 3, 202, 16, 250, 162, 2
 JG 1830 data 169, 255, 157, 64, 3, 202, 16, 250
 AE 1840 data 162, 2, 157, 91, 3, 202, 16, 250
 JD 1850 data 162, 6, 169, 13, 157, 248, 7, 202
 LI 1860 data 16, 250, 173, 134, 2, 32, 215, 159
 FA 1870 data 169, 96, 141, 16, 208, 56, 169, 216

KF 1880 data 162, 8, 157, 0, 208, 233, 48, 202
 HK 1890 data 202, 16, 247, 169, 8, 141, 10, 208
 PJ 1900 data 169, 56, 141, 12, 208, 169, 127, 141
 AA 1910 data 29, 208, 141, 21, 208, 162, 25, 142
 JL 1920 data 46, 157, 96, 169, 0, 141, 21, 208
 NI 1930 data 76, 15, 157, 36, 157, 16, 244, 32
 PK 1940 data 207, 159, 166, 214, 236, 46, 157, 240
 BB 1950 data 233, 142, 46, 157, 181, 217, 48, 1
 HG 1960 data 202, 138, 10, 10, 10, 24, 105, 49
 JM 1970 data 162, 12, 157, 1, 208, 202, 202, 16
 IC 1980 data 249, 169, 0, 162, 40, 228, 213, 176
 JN 1990 data 2, 169, 127, 141, 23, 208, 76, 144
 HE 2000 data 159, 173, 134, 2, 205, 47, 157, 240
 CO 2010 data 11, 141, 47, 157, 162, 6, 157, 39
 KE 2020 data 208, 202, 16, 250, 96

Keyboard Expander: Profile Program

ON 10 rem -----
 HC 12 rem keyboard expander profile prog
 BM 14 rem ke.pf
 EO 16 rem -----
 IP 18 rem (program variables)
 CH 20 zx=0:zy=0:zd=0:zi=0:zj=0:zb=0:zp=0
 ED 22 rem
 EA 24 rem (factor for repositioned keys)
 FI 26 sf=64:cf=128:tf=192:rem shft,c=,ctrl
 KD 28 rem
 FA 30 rem (array dimensions source keys)
 GH 32 nd=0:sd=1:cd=2:td=3:rem *norm,shft,c=,ctrl
 AE 34 rem
 DL 36 rem (used in strings for keys)
 EE 38 rem
 EN 40 s\$=" ":i\$=chr\$(0):i\$=chr\$(0):rem new,link and i'active
 HG 42 q\$=chr\$(34):r\$=chr\$(13):n\$=chr\$(141):rem quote,
 ret,shft+ret
 LH 44 ln\$=chr\$(3):ls\$=chr\$(1):lc\$=chr\$(2):lt\$=chr\$(4)
 :rem norm,shft,c=,ctrl
 ME 46 rem
 BH 48 rem (variables for keys)
 ON 50 a=10:b=28:c=20:d=18:e=14:f=21:g=26:h=29
 :i=33:j=34:k=37:l=42:m=36
 PM 52 n=39:o=38:p=41:q=62:r=17:s=13:t=22:u=30
 :v=31:w=9:x=23:y=25:z=12
 GJ 54 n0=35:n1=56:n2=59:n3=8:n4=11:n5=16:n6=19
 :n7=24:n8=27:n9=32
 EG 56 f1=4:f3=5:f5=6:f7=3:de=0:re=1:ho=51:ri=2
 :do=7:ua=54:la=57
 MH 58 pl=40:mi=43:eq=53:po=48:at=46:as=49:sl=55
 JA 60 pe=44:co=45:se=50:cm=47:sp=60
 MF 62 rem
 LN 64 rem (jmp vector and command line)
 IG 66 zv=0:zc=0:rem addresses, 0=none
 CG 68 rem
 MM 70 rem (key definition array storage)
 EN 72 dim zs\$(3,62),zp(3,62),zr(3,62)
 IG 74 rem
 DP 76 rem -----
 ID 78 rem - define your keyboard profile -
 KH 80 rem - between lines 100 and 799 -
 ED 82 rem ** dont def any new variables **
 LP 84 rem -----

EH	86 rem	DP	130 zs\$(td,r) = s\$ + "load" + q\$ + "0:" + i\$ + q\$ + ",8" + r\$ + "run" + r\$:rem load-run
OD	800 rem	PL	132 zs\$(td,i) = s\$ + "open15,8,15," + q\$ + "i0" + q\$ + ":close15" + r\$:rem init disk
AE	802 rem (relocate tables)	MN	134 zs\$(td,v) = s\$ + "open15,8,15," + q\$ + "v0" + q\$ + ":close15" + r\$:rem validate
CE	804 rem	HJ	136 zs\$(td,g) = s\$ + "load" [3 spcs],8" + r\$:rem load from dir
LI	806 zi = fre(0):zp = peek(55) + 256*peek(56):rem prog start	MO	138 zs\$(td,pl) = s\$ + "clr:poke43,peek(45):poke44,peek(46) :new" + r\$:rem test part
GB	808 zb = peek(zp + 7) + 256*peek(zp + 8):rem dscr base	CK	140 zs\$(td,mi) = s\$ + "poke45,peek(43):poke46,peek(44):" ON
AP	810 ifpeek(zp + 3) = 75andpeek(zp + 4) = 69and peek(zp + 5) = 88andpeek(zp + 6) = 80then814	ON	142 zs\$(td,mi) = zs\$(td,mi) + "poke43,1:poke44,8:clr" + r\$:rem back to norm
OM	812 print "error: cannot locate keyboard expander" :end	OK	144 rem
MJ	814 zx = (peek(47) + 256*peek(48)) + 9:rem array defs	FJ	146 rem (basic progs on strings)
MM	816 forzi = 0to62:forzj = 0to3	BL	147 rem
LL	818 zd = zb + (zi*16) + (zj*4)	HF	148 rem read error channel
DK	820 ifzp(zj,zi)thenpokezd,zp(zj,zi)	DO	150 zs\$(td,e) = s\$ + "8000open15,8,15:input#15,a,b\$,c,d" :printa,b\$,c,d"
HB	822 zy = zr(zj,zi):ifzythenpokezd + 2,zy-int(zy/256)*256 :pokezd + 3,int(zy/256)	AK	152 zs\$(td,e) = zs\$(td,e) + ":close15" + r\$ + "run8000" + r\$ + "8000" + r\$
NB	824 iflen(zs\$(zj,zi)) = 0then830	GP	154 rem list dir to screen
GC	826 zy = zx + (zi*12) + (zj*3)	EG	156 zs\$(td,d) = s\$ + "8000n\$ = chr\$(0):open15,8,0," + q\$ + "\$0" + q\$ + ":get#15,a\$,b\$" + r\$
KA	828 pokezd + 1,peek(zy):pokezd + 2,peek(zy + 1) :pokezd + 3,peek(zy + 2)	PL	158 zs\$(td,d) = zs\$(td,d) + "8001get#15,a\$,b\$:ifst<>0 then8005" + r\$
IO	830 nextzj:nextzi	FP	160 zs\$(td,d) = zs\$(td,d) + "8002get#15,a\$,b\$:printasc(a\$ + n\$) + asc(b\$ + n\$)*256;" + r\$
OF	832 rem	BH	162 zs\$(td,d) = zs\$(td,d) + "8003get#15,a\$:ifa\$ = " + q\$ + q\$ + "thenprint:goto8006" + r\$
HC	834 ifzvthenpoke785,zv-int(zv/256)*256 :poke786,int(zv/256):poke784,0	JE	164 zs\$(td,d) = zs\$(td,d) + ln\$ + chr\$(a)
CD	836 ifzcthenpokezb-7,zc-int(zc/256)*256 :pokezb-6,int(zc/256)	DH	166 zs\$(nd,a) = l\$ + "8004printa\$::goto8003" + r\$
JE	838 zy = (peek(51) + 256*peek(52))-9:rem new start	NA	168 zs\$(nd,a) = zs\$(nd,a) + "8005close15:poke198,0 :end" + r\$
IE	840 forzi = 0to8:pokezy + zi,peek(zp + zi):next:pokezb-1,255	EK	170 zs\$(nd,a) = zs\$(nd,a) + "8006k = peek(203) :ifk = 64then8001" + r\$
JM	842 pokezb-5,zy-int(zy/256)*256:pokezb-4,int(zy/256)	BI	172 zs\$(nd,a) = zs\$(nd,a) + "8007ifk = 62then8005" + r\$
PF	844 poke55,peek(zb-5):poke56,peek(zb-4):clr :rem new top mem	NN	174 zs\$(nd,a) = zs\$(nd,a) + "8008goto8006" + r\$ + "run8000" + r\$
MG	846 rem	OE	176 rem delete basic lines
FM	848 rem (store absolute version)	MJ	178 zs\$(td,z) = s\$ + "8000" + r\$ + "Q8001" + r\$ + "Q8002" + r\$ + "Q8003" + r\$
AH	850 rem	GF	180 zs\$(td,z) = zs\$(td,z) + "Q8004" + r\$ + "Q8005" + r\$ + "Q8006" + r\$ + "Q8007" + r\$
EC	852 input "create an absolute version - y/n" :f\$	FF	182 zs\$(td,z) = zs\$(td,z) + "Q8008" + r\$ + "Q8009" + r\$ + "Q8010" + r\$ + "Q8011" + r\$
AG	854 iff\$<> "y" andf\$<> "n" then852	GN	184 rem
BM	856 iff\$ = "n" then864	KJ	186 rem (some basic keywords)
DB	858 input "filename for profile" :f\$	JN	187 rem
PH	860 sys57812f\$,8	KI	188 zs\$(cd,a) = s\$ + "asc(" :zs\$(cd,c) = s\$ + "chr\$(" " :zs\$(cd,d) = s\$ + "data"
OB	862 poke174,0:poke175,160:poke193,peek(55) :poke194,peek(56):sys62954	GD	190 zs\$(cd,f) = s\$ + "for" :zs\$(cd,g) = s\$ + "goto" :zs\$(cd,i) = s\$ + "input"
GE	864 clr:end	EP	192 zs\$(cd,l) = s\$ + "list" :zs\$(cd,m) = s\$ + "mid\$(" " :zs\$(cd,n) = s\$ + "next"

Keyboard Expander: Profile Definitions

ID	100 rem -----
DK	102 rem keypower profile definitions
LK	104 rem kpower.defs
OD	106 rem -----
KI	108 rem
PO	110 rem (reposition shift cursor keys)
OI	112 rem
IH	114 zp(nd,ua) = do + sf:zp(nd,la) = ri + sf
CJ	116 rem
OF	118 rem (commands on strings)
GJ	120 rem
IB	122 zs\$(td,s) = s\$ + "save" + q\$ + "0:" + i\$ + q\$ + ",8" + r\$:rem save
FF	124 zs\$(td,p) = s\$ + "open15,8,15," + q\$ + "s0:" + i\$ + q\$ + ":close15" + r\$:rem purge
PJ	126 zs\$(td,n) = s\$ + "open15,8,15," + q\$ + "r0:" + i\$ + " " + i\$ + q\$ + ":close15" + r\$:rem rename
OK	128 zs\$(td,l) = s\$ + "load" + q\$ + "0:" + i\$ + q\$ + ",8" + r\$:rem load

GD	190 zs\$(cd,f) = s\$ + "for" :zs\$(cd,g) = s\$ + "goto" :zs\$(cd,i) = s\$ + "input"
EP	192 zs\$(cd,l) = s\$ + "list" :zs\$(cd,m) = s\$ + "mid\$(" " :zs\$(cd,n) = s\$ + "next"
JF	194 zs\$(cd,o) = s\$ + "open" :zs\$(cd,p) = s\$ + "print" :zs\$(cd,r) = s\$ + "right\$(" "
CI	196 zs\$(cd,s) = s\$ + "str\$(" "
EO	198 rem
DG	200 rem (jmp table and command line)
IO	202 rem
LD	204 zv = 40204:rem zc = 0 no command line*
MO	206 rem
LD	208 rem (interrupt routines)

```

AP 210 rem
EM 212 zr(nd,f5) = 40192:zr(nd,f7) = 40195
    :rem list scroll down/up
IB 214 zr(sd,f5) = 40198:zr(cd,f5) = 40201
    :rem line rule enable/disable
GP 216 rem
NI 218 rem end of defs
  
```

Keyboard Expander: "INSTALL" Program

```

ON 10 rem -----
LJ 12 rem  general install program
JH 14 rem          install
EO 16 rem -----
DK 18 ifa = 1 then 36
GL 20 poke55,0:poke56,32:clr
LO 22 input "filename" ;f$
PE 24 open15,8,15:open1,8,2,f$ + ",p,r"
MP 26 input#15,a,b$,c,d
MO 28 close1:close15:ifa = 0 then 34
PD 30 printa,b$,c,d:end
EO 34 gosub44:a = 1:loadf$,8,1
JJ 36 if a$<> "kexp" anda$<> "init"
    then40
FD 38 sysad
AJ 40 poke55,lb:poke56,hb:new
IE 42 rem
BF 44 open1,8,2,f$ + ",p,r"
GP 46 get#1,x$:ifx$ = "" thenx$ = chr$(0)
NC 48 lb = asc(x$)
KP 50 get#1,x$:ifx$ = "" thenx$ = chr$(0)
CO 52 hb = asc(x$):ad = lb + 256*hb
FF 54 fori = 1 to 3:get#1,x$:next
JA 56 a$ = " ":fori = 1 to 4:get#1,x$
    :a$ = a$ + x$
LL 58 next:close1:return
  
```

Keyboard Expander: List Scroll Source

```

FN 1 rem -----
NL 2 rem  list scroll and line rule
OP 3 rem          scroll.s
IN 4 rem -----
MA 10 open 2,8,1, "0:scroll.obj"
IL 12 rem open 4,4
PN 14 sys 700
BC 16 .opt o2
AO 18 ;---
AN 20 msgflg = 157 ;run mode
LK 22 lstx = 197 ;last keypress
NN 24 blnsw = 204 ;blink sw
ON 26 gdbln = 206
OD 28 blnon = 207
FM 30 tblx = 214 ;line#
GJ 32 ldtb1 = 217 ;screen line links
AJ 34 txcadr = 646 ;col for text
BA 36 gdccl = 647
HJ 38 lstshf = 654 ;last shift
GP 40 ;---
HO 42 * = $9d00 ;install address
GD 44 jmp listup ;scroll up
MM 46 jmp listdn ;scroll down
LL 48 jmp lrule ;line ruler
IJ 50 jmp lkill ;kill line ruler
KK 52 ;
ND 54 ;vector tab for background rtns
EG 56 ;set up by profile program
DG 58 brstart * = * + 3 ;start int rtn
DF 60 brend * = * + 3 ;end int rtn
PM 62 ppstart * = * + 3 ;start parallel rtn
CJ 64 ppexcl * = * + 3 ;run exclusive
JF 66 ppsch * = * + 3 ;run shared
DM 68 ppend * = * + 3 ;end parallel rtn
IH 70 rescu * = * + 3 ;reset cursor
  
```

```

ML 72 bgwork * = * + 15 ;remaining jmps
AM 74 ;
GN 76 ;-----
DK 78 ;the list scroll routines
GM 80 ;
JH 82 ;run time variables
GC 84 skey = bgwork
NJ 86 sshf = skey + 1
BC 88 savel = sshf + 1
AE 90 savex = savel + 1
LB 92 savec = savex + 1
AD 94 savev = savec + 2
LG 96 first1 = savev + 2
MP 98 first2 = first1 + 1
KN 100 ;
LL 102 ;---entry from interrupt
OM 104 listup = * ;scroll up
MP 106 jsr liston
KH 108 lup jsr listin
FN 110 jsr lstup
NM 112 jsr listout
PO 114 beq lup
ME 116 bne lexit
MO 118 ;
NM 120 ;---entry from interrupt
KC 122 listdn = * ;scroll down
OA 124 jsr liston
BF 126 ldn jsr listin
LA 128 lda #24
GH 130 sta $d6
BM 132 jsr $e56c
HK 134 jsr listdn
FO 136 jsr listout
PN 138 beq ldn
GD 140 lexit jmp ppend
EA 142 ;
IL 144 ;---initiate parallel program
CI 146 liston = * ;common point for up/dn
GE 148 sty skey ;save entry key
EK 150 stx sshf ;. . . and shift value
FF 152 lda #$ff
AB 154 sta first1
EB 156 sta first2
AN 158 pla ;ret addr = start of parallel prg
GB 160 clc
BI 162 adc #1
PI 164 tax ;low byte
OC 166 pla
FI 168 adc #0
HK 170 tay ;high byte
DE 172 jmp ppstart ;start parallel program
EC 174 ;
DP 176 ;---list up
OE 178 listup = *
NM 180 ldx #$ff
NB 182 lu04 inx ;all lines
NI 184 cpx #25
CK 186 beq lu12
DI 188 lda $d9,x
PD 190 bpl lu04 ;linked
DL 192 jsr gnum ;get num
PA 194 bcc lu04 ;none
OL 196 bcs lu16
MD 198 ;
LN 200 lu12 lda #$ff ;high line num
CJ 202 sta $14
HJ 204 sta $15
FJ 206 lu16 jsr $a613 ;find line
ME 208 ldy $f5 ;lo byt
ND 210 cpx $2c ;hi byt
CK 212 bne lu20
JE 214 cpy $2b ;with start
GK 216 bne lu20
IN 218 rts ;no prev lines
CF 220 ;
JC 222 lu20 dex ;lower
AK 224 cpx $2c ;. . . than start
HN 226 bcs lu24
NH 228 ldx $2c ;correct
MP 230 ldy $2b
DK 232 dey
LJ 234 lu24 stx $40
OE 236 sty $3f
JE 238 ldy #0 ;search for
MJ 240 lu28 lda ($3f),y
MA 242 beq lu36 ;end of line
JG 244 lu32 iny
EO 246 bne lu28
  
```

```

GB 248 rts ;no more
AH 250 ;
KD 252 lu36 iny ;chk
NN 254 lda ($3f),y ;link lo
NA 256 cmp $5f
FK 258 bne lu32 ;no match
EO 260 iny
LL 262 lda ($3f),y ;chk
PI 264 cmp $60 ;link hi
DO 266 bne lu32
HM 268 dey
OE 270 tya ;make match
GI 272 clc
IM 274 adc $3f
KB 276 sta $5f
KJ 278 lda $40
FP 280 adc #0
AO 282 sta $60
KJ 284 ldx #24 ;chk last line
FO 286 lda $d9,x
HF 288 bmi lu40 ;not linked
PJ 290 ldx #23 ;2nd last
DP 292 jsr $e9ff ;erase
AC 294 lu40 ldx #0 ;to scrfl dwn
GG 296 jsr $e968
JO 298 lda $d9
FG 300 ora #$80 ;fix up link
LC 302 sta $d9
KF 304 lda #39 ;and length
DC 306 sta $d5
NG 308 ldx #1 ;set insrt
DD 310 stx $292
PO 312 dex
HF 314 stx $d6 ;home
AB 316 jsr $e9f0 ;position
EJ 318 jmp list
GL 320 ;
BE 322 ;---list down
LM 324 listdn = *
AD 326 ldx #25
HF 328 ld04 dex
NB 330 bmi ld08 ;all lines
DB 332 lda $d9,x
NK 334 bpl ld04 ;linked
DC 336 jsr gnum ;get line num
NH 338 bcc ld04 ;none
MC 340 bcs ld16
MM 342 ;
BH 344 ld08 lda #$ff ;set
LB 346 sta $14 ;. . . for
AM 348 sta $15 ;. . . 1st line
CD 350 ld12 pha ;save a byte
PA 352 bne ld30
ML 354 ld16 cpx #24 ;last
AB 356 bne ld20
FB 358 ld18 lda #$0d ;ret char
JB 360 bne ld12
FF 362 ld20 cpx #23 ;2nd last
IC 364 bne ld24
KE 366 inx
HD 368 jsr $d9,x
NF 370 bpl ld18
BJ 372 ld24 lda #$ff
HC 374 bne ld12
JI 376 ld30 inc $14 ;lnum
BD 378 bne ld32
JC 380 inc $15
GB 382 ld32 jsr $a613 ;find line
BM 384 lda $da ;2nd line
MK 386 bmi ld36 ;not linked
LE 388 ora #$80 ;unlink it
LJ 390 sta $da
BN 392 ldx #1
ND 394 jsr $e9ff ;erase line
IL 396 ld36 ldx #24 ;last line
OO 398 stx $d6 ;curs row
NP 400 jsr $e9f0 ;set curs
KB 402 pla
OL 404 bmi list
FK 406 jsr $ffd2 ;print ret
OA 408 ;
IG 410 ;---
IJ 412 list = *
LO 414 ldy #1
LL 416 lda ($5f),y
BJ 418 beq ls24
GH 420 sta savel
BJ 422 lda #0
  
```

IM	424	sta	(\$5f),y	;to fool list	OM	600 ;			PL	776	dex
PC	426	sty	\$f	;list quote flg	OL	602 rerv	= *	;rst err vect	EA	778	bpl lr24
EH	428	lda	\$5f		EA	604	lda savev		HA	780	lda #8
AL	430	sta	\$3f		ON	606	sta \$300		IL	782	sta spr + 10 ;s5 x
ID	432	lda	\$60		KM	608	lda savev + 1		HK	784	lda #56
EH	434	sta	\$40		GO	610	sta \$301		BM	786	sta spr + 12 ;s6 x
HD	436	lda	#\$ff	;to max	KN	612 ;			EK	788	lda #\$7f
OH	438	sta	\$14		PH	614 rcur	= *	;rst cursor	OO	790	sta spr + 29 ;exp x
DI	440	sta	\$15		ND	616	lda #0	;set blink on	DL	792	sta spr + 21 ;enable
AH	442	jmp	\$a6d7	;into list	HH	618	sta blnsw		EA	794	lda #25
CD	444 ;				KE	620	ldx savec		ID	796	stx lrspyl
Mf	446 ;---				OL	622	stx \$d6		EJ	798 lr30	rts
JD	448 lserv	= *		;err vec entry	EL	624	lda savec + 1		GJ	800 ;	
JK	450	cpx	#\$0b	;err from	NF	626	sta \$d3		FO	802 ;---	kill line rule
EE	452	beq	ls32	;eval fxd pt	FP	628	sec		HB	804 lrkill	= *
LD	454	lda	savel	;restore	JA	630	sbc #40		BB	806	lda #0
FB	456	ldy	#1		IO	632	bcc rcu10	;not on 2nd	PH	808	sta spr + 21 ;dsabl
NB	458	sta	(\$3f),y		FG	634	sta \$d3		AA	810	jmp brend ;finish
KM	460 ls24	lda	#\$0d	;ret	IM	636 rcu10	jsr \$e56c	;fix curs	CK	812 ;	
HG	462	jsr	\$ffd2		EP	638 ;			MP	814 ;---	
HP	464	rts		;end	IH	640 getkey	= *		CL	816 ruler	= *
IE	466 ;				AB	642	cli		EH	818	bit msgflg ;run
NK	468 ls32	pla:pla			IG	644	ldy skey		BL	820	bpl lrkill
CN	470	pla:pla		;clr rtss	DD	646	ldx sshf		MK	822 ;	
LO	472	jmp	ls24		AL	648	cpy lstx		GB	824 lr40	= * ;position
AF	474 ;				FE	650	bne gk40		EP	826	jsr lrcolor
KK	476 ;---				AH	652	cpx lstshf		MB	828	ldx tblx
LK	478 gnum	= *		;get line num	JO	654 gk40	rts		OC	830	cpx lrspyl
IH	480	jsr	\$e9f0	;set line start addr	GA	656 ;			AC	832	beq lr30
PP	482	ldy	#\$ff		FG	658 ;---	not used: same as rescur		OF	834	stx lrspyl
AD	484 gn10	iny			PB	660 rcur	= *	;reset cursor	FB	836	lda ldtb1,x
KN	486	lda	(\$d1),y		IE	662	lda blnon	;check blink	JC	838	bmi lr42
GP	488	cpy	#39		KF	664	beq rc10		PP	840	dex
CD	490	beq	gn20 + 1	;end of line	KL	666	lda gdbln	;orig char	OI	842 lr42	txa
KF	492	cmp	#\$20		GG	668	ldx gdcol	;orig colr	ID	844	asl a
LL	494	beq	gn10		JO	670	ldy #0		KD	846	asl a
BG	496	cmp	#\$30		IA	672	sty blnon	;clear blink	MD	848	asl a
AJ	498	bcc	gn20 + 1		ED	674	jsr \$ea13	;reset blink	IM	850	cli
JK	500	cmp	#\$3a		BP	676 rc10	rts		EO	852	adc #49
EN	502	bcs	gn20 + 1		MB	678 ;			FD	854	ldx #12
PK	504	tya			CH	680 ;*****			JG	856 lr44	sta spr + 1,x ;y pos
AH	506	clc			AC	682 ;			BB	858	dex
FJ	508	adc	\$d1		NL	684 ;basic line ruler routine			DB	860	dex
JP	510	sta	\$7a		EC	686 ;			OF	862	bpl lr44
KK	512	lda	\$d2		PN	688 sprt	= 832	;sprite data area	LE	864	lda #0
PN	514	adc	#0		CP	690 spr	= 53248	;sprite register	BE	866	ldx #40
CA	516	sta	\$7b		GL	692 scr	= 2040	;sprite variables	IE	868	cpx \$d5 ;max
NE	518	jsr	\$79	;chrgot	BN	694 lrspyl	= bgwork + 13		DG	870	bcs lr46
OE	520	stx	savex		ML	696 lrspyc	= bgwork + 14		IP	872	lda #\$7f
IN	522	jsr	\$a96b	;eval num to binary	AD	698 ;			PJ	874 lr46	sta spr + 23 ;y exp
EB	524	ldx	savex		MP	700 lrule	= *	;entry from interrupt	AH	876	jmp lr30
PI	526	sec			OK	702	ldx #<ruler	;start addr of bg rtn	EO	878 ;	
NF	528 gn20	bit	\$18		DC	704	ldy #>ruler		OD	880 ;---	
OP	530	rts			AE	706	jsr brstart	;start background rtn	NF	882 lrcolor	= *
KI	532 ;				FN	708	bne lr30	;resource is used!	FJ	884	lda txcadr
EO	534 ;---				MD	710 ;			LC	886	cmp lrspyc
LG	536 listin	= *			OO	712	lda #0	;make sprite	OH	888	beq lr58
DL	538	sei			KI	714	sta lrspyl		KL	890 lr50	sta lrspyc
CJ	540 ;				KH	716	sta lrspyc		PM	892	ldx #6
JG	542 serv	= *		;set err vect	OK	718	ldx #3f		AG	894 lr52	sta spr + 39,x;color
CG	544	lda	\$300		NA	720 lr10	sta sprt,x		HD	896	dex
IA	546	sta	savev		JI	722	dex		NH	898	bpl lr52
KG	548	lda	\$301		LL	724	bpl lr10		KA	900 lr58	rts
OM	550	sta	savev + 1		BC	726	ldx #2		MP	902 ;	
ED	552	lda	#<lserv		FJ	728	lda #\$ff		EG	904 .end	
KK	554	sta	\$300		JB	730 lr12	sta sprt,x				
ED	556	lda	#>lserv		DJ	732	dex				
CL	558	sta	\$301		NM	734	bpl lr12				
GK	560 ;				LC	736	ldx #2				
PK	562 scur	= *		;save cursor	CA	738 lr14	sta sprt + 27,x				
FM	564	jsr	rescur	;reset cursor	LJ	740	dex				
IF	566	lda	#255	;set blink off	NN	742	bpl lr14				
FE	568	sta	blnsw		OF	744 ;					
AP	570	lda	\$d6		ND	746	ldx #6				
MP	572	sta	savec		CH	748	lda #13				
LO	574	lda	\$d3		PJ	750 lr20	sta scr,x	;ptrs			
CM	576	sta	savec + 1		HK	752	dex				
NC	578	lda	#0		MN	754	bpl lr20				
NB	580	sta	\$d3	;column	FB	756	lda txcadr				
EN	582	cli			IK	758	jsr lr50	;color			
EP	584	jsr	ppexcl	;run exclusively	NC	760	lda #\$60				
GD	586	rts			CJ	762	sta spr + 16	;msb x			
CM	588 ;				NH	764	sec				
MB	590 ;---				IF	766	lda #216				
DI	592 listout	= *			HF	768	ldx #8				
LO	594	sei			FI	770 lr24	sta spr,x	;x pos			
PD	596	lda	#0		PK	772	sbc #48				
DH	598	sta	\$292	;clr scroll	NL	774	dex				

News BRK

Submitting NEWS BRK Press Releases

If you have a press release which you would like to submit for the NEWS BRK column, make sure that the computer or device for which the product is intended is prominently noted. We receive hundreds of press releases for each issue, and ones whose intended readership is not clear must unfortunately go straight to the trash bin. It should also be mentioned here that we only print product releases which are in some way applicable to Commodore equipment. News of events such as computer shows should be received at least 6 months in advance.

Transactor News

Transactor Writer's Guide Finally Finished

That's right! After 3 years of collecting, compiling, re-arranging, and generally ensuring completeness, The T. Writer's Guide is done. We kept all those requests in a file and have sent out about 200 so far. If you would like one, they're free for the asking. Call or write the office in Milton, Ontario.

Subscription Timing

Remember, if we receive your subscription order within two weeks of the release of an issue, your subscription won't start until the next issue. The mail list is sorted and printed about two weeks before each release date. All the mail is processed, sometimes the "morning of", the day we print up the list. If your subscription order comes in the day after, it won't appear on the "big list", which means it won't be sorted into our mailing at second class postage rates. We can't send out mags at first class postage rates, so if you sent your sub 2-3 weeks before a release date, and your Transactor doesn't show within 2-3 weeks after the release date, you should probably buy one more off the news stand until your subscription gets "engaged".

Free Transactor T's with Mag+Disk Subscription

For a limited time only, subscribe or renew to a combination magazine and disk subscription, and we'll send you a free Transactor T-Shirt! You save 29% off the magazines, 16% off the disks, and get a Transactor T worth \$13.95 (\$17.95 if you order the jumbo size!) The T-Shirts come in 5 sizes (red only), with a 3-color screen featuring Duke, our mascot, dressed in a snappy white tux, standing behind the Transactor logo done in yellow with black "3-D" borders. The screen was done using a special "super-opaquing" process that cost us quite a bit more than those decals that crack and fade. Mine has been through the wash three times and so far shows no signs of "machine punishment" like my other T's have.

Transactor Disk Price Increase

A subscription to 6 Transactor Disks remains at \$45.00. However, the price of single order Transactor Disks has been increased from \$7.95 to \$8.95 each - another good reason to take advantage of the above offer!

Refund Policy

Should any product you order be defective on receipt, return it and we'll send you another for no additional charge. Recently we've had a few items returned because "it's not quite what I wanted". We will credit your account (less shipping and handling) for purchases of other Transactor products, but we ask that you please be sure you need things like G-Links or RAM boards since we can't refund your money. While we're on the subject, although we've never had a subscriber ask for one, there are no refunds on subscriptions.

Allow 2 to 6 Weeks

When a mail order arrives, it's processed and prepared within 5 business days. But we find it takes about 1-2 weeks for the post office to deliver it. Coupled with the fact that it takes 1-2 weeks for the order to get from you to us means it will probably take somewhere between 2 and 6 weeks for an item to arrive from the day you mail the order. Please bear with us.

Transactor Mail Order News

Our mail-order department is expanding nicely, but our mail-order card isn't. Seems we just can't find any more room to put more text without making it so small that you can't read it. So, if you're using the card to order, we suggest you pull it out and cross-reference with the list below for more details.

■ Transactor T-Shirts, \$13.95 and \$17.95

As mentioned earlier, they come in Small, Medium, Large, Extra Large, and Jumbo. They're 13.95 each, \$17.95 for the Jumbo. The Jumbo makes a good night-shirt/beach-top - it's BIG. I'm 6 foot tall, and weigh in at a slim 150 pounds - the Small fits me tight, but that's how I like them. If you don't, we suggest you order them 1 size over what you usually buy. The design is screened using a "super-opaquing" process so they should wear much longer than your ordinary screens and iron-ons.

■ The Transactor Book of Bits and Pieces #1, \$14.95

Not counting the Table of Contents, the Index, and title pages, it's 246 pages of Bits and Pieces from issues of The Transactor, Volumes 4 through 6. Even if you have all those issues, it makes a handy reference - no more flipping through magazines for that one bit that you just know is somewhere. . . Also, each item is forward/reverse referenced. Occasionally the items in the Bits column appeared as updates to previous bits. Bits that were similar in nature are also cross-referenced. And the index makes it even easier to find those quick facts that eliminate a lot of wheel re-inventing.

■ The Tr@ns@ctor 1541 ROM Upgrades, \$49.95

You can burn your own using the ROM dump file on Transactor Disk #13, or you can get a set from us. This is an introductory offer to see how much response we get - we only have 50 sets available. But if they become popular, we'll certainly get more in, and the price may go up. There are 2 ROMs per set, and they fix not only the SAVE@ bug, but a number of other bugs too (as described in P.A. Slaymaker's article, Vol 7, Issue 02). Remember, if SAVE@ is about to fail on you, then Scratch and Save may just clobber you too. This hasn't been proven 100%, but these ROMs will eliminate any possibilities short of deliberately causing them (ie. allocating or opening direct access buffers before the Save).

■ The Micro Sleuth: C64/1541 Test Cartridge, \$79.95 US., \$99.95 Cdn.

This cartridge, designed by Brian Steele (a service technician for several schools in southern Ontario), will test the RAM of a C64 even if the machine is too sick to run a program! The cartridge takes complete control of the machine. It tests all RAM in one mode, all ROM in another mode, and puts up a menu with the following choices:

- 1) Check drive speed
- 2) Check drive alignment
- 3) 1541 Serial test
- 4) C64 serial test
- 5) Joystick port 1 test
- 6) Joystick port 2 test
- 7) Cassette port test
- 8) User port test

A second board, that plugs onto the User Port, contains 8 LEDs that lets you zero in on the faulty chip. Complete with manual. **Note:** This is an introductory offer – prices may go up by next issue.

■ Inner Space Anthology \$14.95

This is our ever popular Complete Commodore Inner Space Anthology. Even after a year and a half, we still get inquiries about its contents. Briefly, The Anthology is a reference book – it has no “reading” material (ie. “paragraphs”). In 122 compact pages, there are memory maps for 5 CBM computers, 3 Disk Drives, and maps of COMAL; summaries of BASIC commands, Assembler and MLM commands, and Wordprocessor and Spreadsheet commands. Machine Language codes and modes are summarized, as well as entry points to ROM routines. There are sections on Music, Graphics, Network and BBS phone numbers, Computer Clubs, Hardware, unit-to-unit conversions, plus much more. . . about 2.5 million characters total!

■ The Toolbox (PAL and POWER) \$79.95

PAL and POWER from Pro-Line are two of the most popular programs for the Commodore 64. PAL is an easy-to-use assembler (most assembler listings in The Transactor are in PAL format), and POWER is a programmer’s aid package that adds editing features and useful commands to the programming environment. They come with two nice manuals, and our price is \$50 less than suggested retail!

■ AX1000 Amiga 1 MEG RAM Box \$729.00 (+ \$100 S&H) U.S.,
\$1035.00 (+ \$25 S&H) Cdn

■ AX2000 Amiga 2 MEG RAM Box \$899.00 (+ \$100 S&H) U.S.,
\$1276.00 (+ \$25 S&H) Cdn

The AX2000 adds 2 Megabytes of “fast” RAM to the Amiga, allowing more tasks to run in the system at once, or for use as a fast RAM-drive. The unit plugs into the expansion connector on the side of the Amiga and duplicates the connector for other devices to plug into. Up to two RAM boards may be plugged in together (limited by the Amiga’s power supply), adding 4 Megabytes. The box has “auto-config”, so with Kickstart 1.2 the RAM will automatically be added to the system when it is booted. If you are using Kickstart 1.0 or 1.1 (no auto-config), you can use the program included with the AX2000 to add the memory to the system, and change your startup-sequence to automatically add the memory on power-up. Standard expansion bus architecture was used in the design of the AX2000, ensuring compatability with all peripherals and operating system releases. The unobtrusive steel box is the same height and colour as the Amiga, and snugs up to the side without taking up much extra space. The unit is built tough and comes with a 1 year manufacturer warranty.

This seems to be the most highly-recommended Amiga RAM board, and the first one to actually be available, so we’re selling it here at The Transactor. You can order the AX2000 or the 1-Meg AX1000 from the subscription form in this issue. Shipping and Handling to the U.S.A. is via courier and includes all customs clearance, or you can opt to clear shipments yourself and have it shipped “collect”.

- Pocket Writer C64 \$39.95 US, \$49.95 Cdn
- Pocket Planner C64 \$39.95 US, \$49.95 Cdn
- Pocket Filer C64 \$39.95 US, \$49.95 Cdn
- Pocket Writer C128 \$49.95 US, \$69.95 Cdn
- Pocket Planner C128 \$49.95 US, \$69.95 Cdn
- Pocket Filer C128 \$49.95 US, \$69.95 Cdn
- Pocket Dictionary \$14.95 US, \$19.95 Cdn

In our opinion, the Pocket packages from Digital Solutions are the best you can get on their own – the fact that they work with each other makes them even better. Planner and Filer data can be loaded into the Writer, Writer text can be sent to the Filer, and etcetera. The Dictionary spell checker works with both versions of the Writer.

■ The GLINK C64 to IEEE Interface \$49.95

The GLINK plugs into the cartridge port, but doesn’t extend the port for more cartridges (for that you’ll need a “motherboard” of some kind). The other side of the GLINK is an IEEE card-edge suitable for a PET-IEEE cable. From there, any IEEE device can be accessed including disk drives, modems, printers, etc. The

GLINK is “transparent” – that means it won’t interfere with programs, except those that rely on the serial routines which it replaces (ie. programs with built-in “fastloaders” for the 1541 won’t like the presence of the GLINK). It has no manual (aside from one page of installation instructions) because it alters nothing and leaves everything unchanged! An on-board switch allows you to select Serial or IEEE. GLINK works with both the C64 and the C128 in 64 mode, but not on the VIC 20.

■ The TransBASIC Disk \$9.95

This is the complete collection of every TransBASIC module ever published up to Volume 7, Issue 01. There are over 120 commands at your disposal. You pick the ones you want to use, and in any combination! It’s so simple that a summary of instructions fits right on the disk label. The manual describes each of the commands, plus how to write your own commands.

■ Super Kit 1541 \$29.95 US, \$39.95 Cdn

Super Kit is, quite simply, the best disk file utility there is. No more losing those valuable copy-protected originals (like what’s happened to me twice too many times). So far we’ve shipped over 300 Super Kits and orders continue to pour in.

■ Gnome Speed Compiler \$59.95 US, \$69.95 Cdn

This compiler is for BASIC 7.0 on the Commodore 128.

■ Gnome Kit Utility \$39.95 US, \$49.95 Cdn

Gnome Kit is a Commodore 128 utility with enhancements for the BASIC editor (like Trace, Find, Rename, Delete, Auto, etc.) as well as enhanced monitor commands, and floppy disk monitor functions.

Transactor Disks, Transactor Back Issues, and Microfiche

All issues of The Transactor from Volume 4 Issue 01 forward are now available on microfiche. According to Computrex, our fiche manufacturer, the strips are the “popular 98 page size”, so they should be compatible with every fiche reader. Some issue are ONLY available on microfiche – these are marked “MF only”. The other issues are available in both paper and fiche. Don’t check both boxes for these unless you want both the paper version AND the microfiche slice for the same issue.

To keep things simple, the price of Transactor Microfiche is the same as magazines, with one exception. A single back issue will be \$4.50 and subscriptions are \$15.00. The exception? A complete set of 18 (Volumes 4, 5, and 6) will cost just \$39.95!

This list also shows the “themes” of each issue. “Theme issues” didn’t start until Volume 5, Issue 01.

- Vol. 4, Issue 01 (■ Disk 1)
- Vol. 4, Issue 02 (■ Disk 1)
- Vol. 4, Issue 03 (■ Disk 1)
- Vol. 5, Issue 01 – Sound and Graphics (■ Disk 2)
- Vol. 5, Issue 02 – Transition to Machine Language (■ Disk 2)
- Vol. 5, Issue 03 – Piracy and Protection – MF only (■ Disk 2)
- Vol. 5, Issue 04 – Business & Education – MF only (■ Disk 3)
- Vol. 5, Issue 05 – Hardware & Peripherals (■ Disk 4)
- Vol. 5, Issue 06 – Aids & Utilities (■ Disk 5)
- Vol. 6, Issue 01 – More Aids & Utilities (■ Disk 6)
- Vol. 6, Issue 02 – Networking & Communications (■ Disk 7)
- Vol. 6, Issue 03 – The Languages (■ Disk 8)
- Vol. 6, Issue 04 – Implementing The Sciences (■ Disk 9)
- Vol. 6, Issue 05 – Hardware & Software Interfacing (■ Disk 10)
- Vol. 6, Issue 06 – Real Life Applications (■ Disk 11)
- Vol. 7, Issue 01 – ROM / Kernel Routines (■ Disk 12)
- Vol. 7, Issue 02 – Games From The Inside Out (■ Disk 13)
- Vol. 7, Issue 03 – Programming The Chips (■ Disk 14)
- Vol. 4, Issue 04 – MF only (■ Disk 1)
- Vol. 4, Issue 05 – MF only (■ Disk 1)
- Vol. 4, Issue 06 – MF only (■ Disk 1)

Notes: The Transactor Disk #1 contains all program from Volume 4, and Disk #2 contains all programs from Volume 5, Issues 1-3. Afterwards there is a separate disk for each issue. Disk 8 from The Languages Issue contains COMAL

0.14, a soft-loaded, slightly scaled down version of the COMAL 2.0 cartridge. And Volume 6, Issue 05 published the directories for Transactor Disks 1 to 9.

Sending Cheques For Transactor Products

If you wish to send a cheque with your subscription/order form, or you wish to conceal your credit card number, you can use an envelope and tape it to the back of the subscription card. The post office has threatened to charge us extra for sloppy business reply mail so please try to use an envelope that is smaller than the card. Can't find one? Just trim the end off the envelope and tape along that edge when fixing it to the card.

The Transactor Communications Disk

The "Transactor Communications Disk" is proving to be a bigger project than we expected. Collating a suitable manual is turning into a task almost as large as putting together a Transactor magazine. But it's not ready yet so don't send any orders. More next issue.

CompuServe and Quantum Link

By the time you read this, The Transactor should have sections up and running on both CompuServe AND Quantum Link.

On CompuServe use "GO CNV" for Commodore News and Views. There you'll find a menu of popular Commodore related magazines. You'll be able to read articles that have appeared in previous issues, as well as download programs we've published (software for downloading from CIS is available free - see the item later in News BRK). Questions concerning magazine related topics will also be answered on a regular basis.

On Quantum Link you'll find us in the "Meet the Press" section of the Commodore Information Service. You'll need the special software from Quantum Link before you can access the articles and software.

The Transactor will be featuring a regular section on our online activities for both services. Next issue we'll be publishing a "Getting Started" summary with complete step-by-step instructions for joining us online.

Industry News

The Commodore Show, September 20 and 21, 1986

The West Coast Commodore Association is having another show at the Los Angeles Airport Hilton, September 20th and 21st. Judging from their February show in San Francisco, this one should be just as big a success. See the ad in this issue.

Twin Cities 128: The Commodore 128 Journal

Last issue we published the address of The Twin Cities 128 Club. We copied the address correctly from an incorrect copy. Here's the correct address:

Twin Cities 128
1607 Hewitt Avenue, Suite 4
PO Box 4625
Saint Paul, MN 55104

iNet 2000

This is the service that the rest of the country has been waiting for! Unlike metropolitan areas, calling your favourite data base from a remote area means calling long distance to the closest public dial port of a national networking service. And that can cost more than the service itself! iNet 2000 is a service operated by Telecom Canada, and (I'm told) is also working on setting up services in the U.S. There is a \$50 subscriber fee, and hourly charges. Then, from a menu, you can connect to over 1000 other online services, (including CompuServe) through iNet. The best part is... for just \$3.00 per month you can call iNet using a 1-800 number! No more long distance charges!

Once connected to CIS, your iNet clock stops ticking. When you exit from CIS, iNet starts the clock again until you connect to another service. The iNet package comes with a lovely manual with reference cards. I haven't tried mine out yet, but next issue I have a complete run-down of my results. If you can't wait until then, call:

Sylvia Abretti
Telecom Canada
410 Laurier Avenue W. Room 240
Ottawa, Ontario, K1P 6H5
1-800 267 7400

High-Powered FREE Terminal Program for the C64

CBterm/C64 is an all-ML terminal program for the C64. It is set far above all the rest of the terminal programs because of a number of special features. First and foremost, the price is right; CBterm is FREE.

As for the rest of the features, CBterm supports the following:

- 40 column display and a split-screen 80 column display. (No extra hardware needed).
- XMODEM and Punter file transfer protocols available. The XMODEM protocol will even correctly download CompuServe .IMG files.
- 22.5K RAM buffer with read/write and save/load to disk. Display RAM to screen or transmit RAM to modem.
- ASCII/PETSCII file conversion built in.
- Works with any and all modems. 1600, 1650, 1660, 1670, HES, Mighty-Mo, Hayes, RS-232 adapters.
- 300 or 1200 baud.
- Auto dial and redial. (Even on a 1660 that does not have carrier detect.)
- VIDTEX Cursor control and positioning mode for use with CompuServe. Includes Low Res Graphics.
- Full printer support. Send RAM data or control codes to printer. Even alter the printer's secondary address while still on line.
- Full DISK control - send commands or list the directory.
- User-programmable FUNCTION KEYS to help automate your log-ins.
- Terminal parameter controls to provide compatibility with all BBS/computer systems.
- Built-in HELP screen.
- Super extra: Direct display of the COMPUSERVE RLE encoded High Resolution Pictures. This includes the NWS Radar Weather Maps, CB Users Pictures, FBI 10 Most Wanted List pictures, etc.
- The high resolution screen (Text or pictures) may be dumped to any STAR or EPSON dot matrix printer. Option overlay programs provide printing of these pictures in a low resolution mode to ANY STANDARD printer. And they can be saved to disk in KOALA picture format and can then be altered with KOALA software.

CBterm/C64 is available for downloading from Data Library 2 of the CB Interest Group SIG on the CompuServe network. This Data Library contains the CBterm/C64 program along with all supporting documentation and overlay programs. They may be downloaded with any XMODEM or CompuServe executive program (Vidtext).

For those people who cannot download, or who are not on the CompuServe network, CBterm/C64 is made available free by sending a blank disk, along with a stamped self-addressed disk mailer to:

Chris Dunn
5848 N. Whipple St.
Chicago, IL 60659

You must include postage and the return mailer.

Any questions can be directed to the SYSOP of the CB Interest Group SIG (CBIG) on CompuServe (GO CBIG) or via EZplex to [76703,717]. CBterm, originally written for use on CompuServe and its CB simulator, has grown into the powerful program it is today and incorporates requests by users as to what they

would like to see in a terminal program. New features are being added and are released in DL2 of the CBIG SIG.

CBterm is distributed free, for non-commercial or private use so that all C64 users can enjoy the world of telecommunicating. CBterm/C64 and all related files are (C) 1985 by Chrisdos. They may NOT be used for any commercial purpose. All rights reserved. VIDTEX, EXplex and CB simulator are TM, Compuserve Inc. Chris Dunn (Chrisdos) is an information provider/SYSOP and is not an employee of Compuserve Inc.

The mail-in disk copy offer may be terminated at any time.

Hardware Interconnection Products

Master Software has announced four new interconnection products for Commodore computers: "Modem Master", "Modem Master Plus", "Y-NOT?", and the "80 Mono Cable".

"Modem Master" is a four-foot extender for the user port (modem port) of the VIC 20, Commodore 64, SX-64, Plus/4, and Commodore 128, allowing devices to be placed in a location more convenient than behind the computer. The extender uses tangle-proof ribbon cable and connectors which are keyed to prevent incorrect installation. List price of "Modem Master" is \$29.95.

"Modem Master Plus" is a "Modem Master" with a reset switch. The reset switch is buffered to prevent electrical damage to the computer, and programs to recover a BASIC program that was in memory at the time of reset are included. Price is \$34.95.

The question of whether two printers can be used simultaneously with Commodore computers is answered: Y-NOT? "Y-NOT" is a six-foot long "Y" cable for the six-pin serial port of all Commodore computers. "Y-NOT" contains one male six-pin plug and two female six-pin jacks, and can be used to operate two printers or to separate the disk drive and printer to opposite sides of the computer set-up for added system flexibility. "Y-NOT?" is priced at \$15.00.

The "80 Mono Cable" will produce an 80 column monochrome display from the Commodore 128 in 80 column mode on any composite colour monitor or monochrome monitor. It is six feet long and plugs into the RGBI port of the computer and into the video input jack of the monitor. The "80 Mono Cable", priced at \$9.00, is ideal for word processing and spreadsheet applications, or any time a colour display is not needed.

All prices include shipping to USA and Canada destinations. Discounts are available for dealers, distributors, and computer clubs. Add 3.00 for C.O.D. (Maryland residents must include 5% sales tax.)

Master Software
6 Hillery Court
Randallstown, MD 21133
(301) 922-2962

Aegis Ships CAD Program for the Amiga

Santa Monica, CA -- Aegis Development, Inc. has begun shipments of Aegis Draw, 1 2-dimensional CAD program for the Commodore-Amiga computer. This is the program that many owners and potential owners of Amigas have been waiting for, and Aegis Draw is being touted as the program the machine was created for.

"Based on our initial orders and end user requests, there are thousands of people out there who have been waiting for Aegis Draw before buying their Amiga. Now that the product is shipping, you can expect to see Amiga sales increase significantly", Company president David Barrett stated. He went on to say that, "This program was created specifically for the Amiga, and is not simply a port from another computer. Therefore, we take advantage of many of the Amiga's characteristics which make this program so powerful, such as

multitasking and multiwindowing. Aegis Draw is only the beginning. We are planning 'expanded CAD' products in the future.


Aegis Draw requires a basic Amiga computer, 512K or RAM, and one disk drive (although two drives and/or a hard drive is highly recommended, as well as additional memory). Allowing use of the multitasking capabilities of the Amiga, Draw will give the user two windows to work in -- either two separate drawings, or two windows of the same drawing (more windows are allowed when more memory is available). Some of Draw's significant features include:

- Infinite levels of Zoom
- 16 customizable colours
- Lines, arcs, circles, rectangles, and freehand drawing tools
- Independent window display features
- Numeric display
- Background plotting/printing
- Undo feature allows last operation to be undone
- Full rotation of any object/part
- Interchangeable parts between drawings
- Automatic dimensioning of any part or piece of drawing
- Customizable scale and grid size per window
- Numerical input for precise positioning of objects/drawings
- Horizontal or vertical printout
- Works with digitizing tablets, plotters, or printers supported by the Amiga.
- Cloning of any part of a drawing
- Full editing controls
- Grid and Data snap (on/off toggle)
- Save as IFF or ASCII files
- Customizable plotter driver
- Parts library for each drawing
- Resizing of any object/part

Aegis Draw also takes advantage of keyboard input with corresponding keystrokes for the most popular tasks. This program can be used by novices as well as experienced CAD engineers for simple to complex design. Aegis Draw is available at authorized Amiga dealers and software stores. The suggested retail price is \$199.95 (U.S.). For more information, contact:

AEGIS DEVELOPMENT, INC.
2210 Wilshire Blvd. #277
Santa Monica, CA 90403 (213) 306-0735

WCCA PRESENTS



SEPTEMBER 20 & 21 1986
SHOW TIMES 10AM - 6PM P.S.T.
LOS ANGELES AIRPORT HILTON
CALL 213-410-4000 for hotel reservations

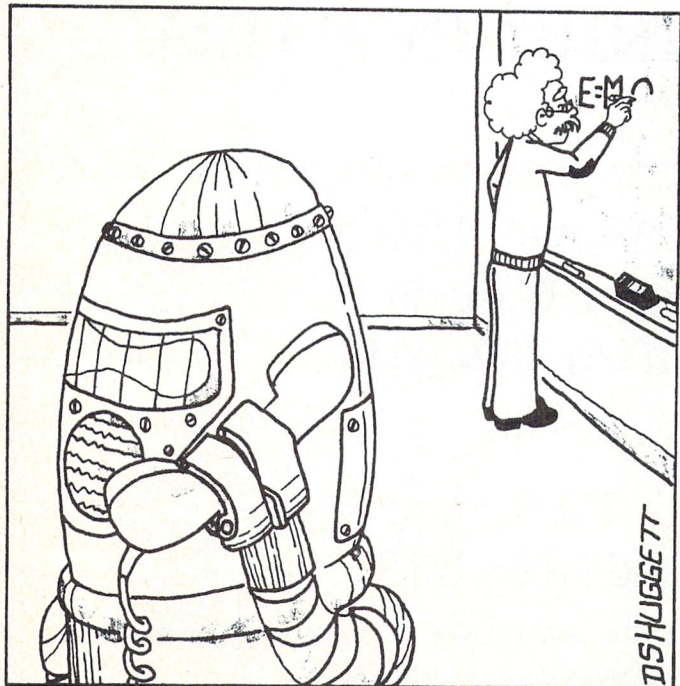
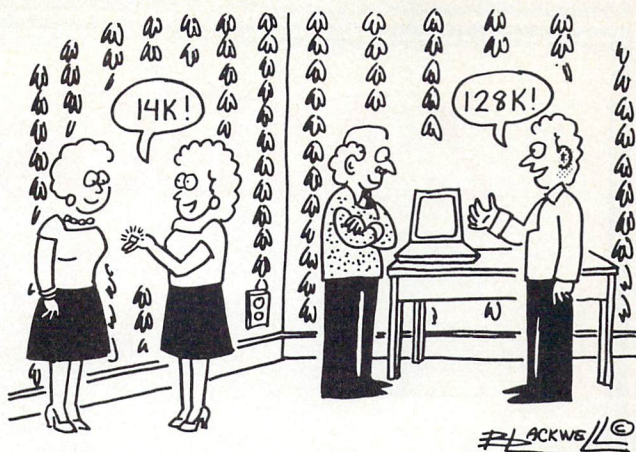
- EXHIBITS, EVENTS, AND DOOR PRIZES
- NATIONAL COMMODORE SPEAKERS
- SHOW SPECIALS & DISCOUNTS
- SEE THE LATEST INNOVATIONS IN HARDWARE/SOFTWARE TECHNOLOGY FOR THE COMMODORE MARKET

The only West Coast exhibition and conference focusing exclusively on the AMIGA, Commodore 128, and C64 marketplace

REGISTRATION FEES: ONE DAY \$10.00 TWO DAYS \$15.00

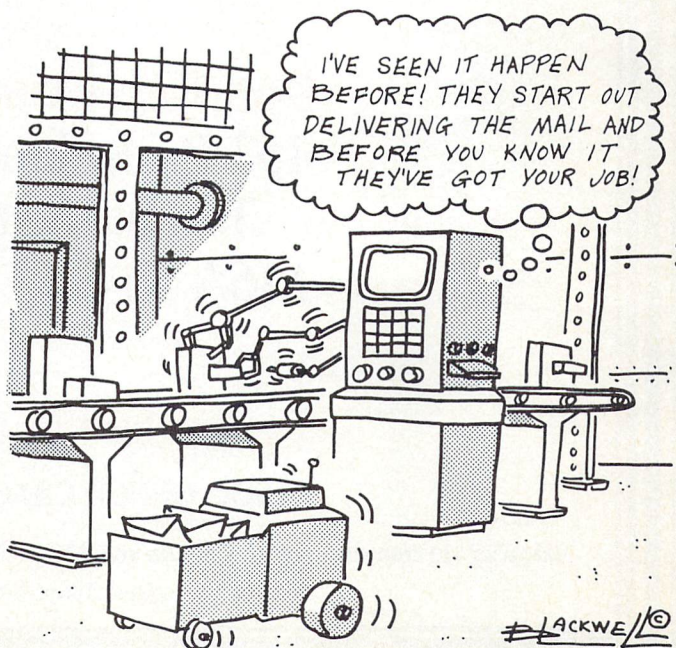
FOR MORE INFORMATION AND DETAILS CONTACT:
WEST COAST COMMODORE ASSOCIATION, INC.
P.O. BOX 210638
SAN FRANCISCO, CALIFORNIA 94121
(415)982-1040 BETWEEN 8AM-5PM PST

Computoons



"I have an older model, but it's very intelligent. . ."

IT WON'T WORK HELEN...
 YOU'RE APPLE AND I'M COMMODORE



2 MEGs For Your AMIGA

A must for software developers

Allows more programs to run simultaneously and faster

Can be used to increase system RAM and/or as a FAST RAM DRIVE

Uses standard memory bus architecture to allow for future compatibility

Allows full use of memory expansion port for additional peripherals

AX2000 2 MEG RAM Board \$899.00 U.S. (\$1276.00 CDN)

AX1000 1 MEG RAM Board \$729.00 U.S. (\$1035.00 CDN)

Complete in case, nothing else to buy!

1 year manufacturer warranty!

DEALER INQUIRIES INVITED

Comspec Communications Inc.

153 Bridgeland Avenue, Unit 5

Toronto, Ontario, Canada

M6A 2Y6 (416) 787-0617

Mail order through The Transactor

(see order card and News BRK)

Shipping via courier: within Canada add \$25.00. To U.S.A. add \$100.00 U.S. - includes customs clearance

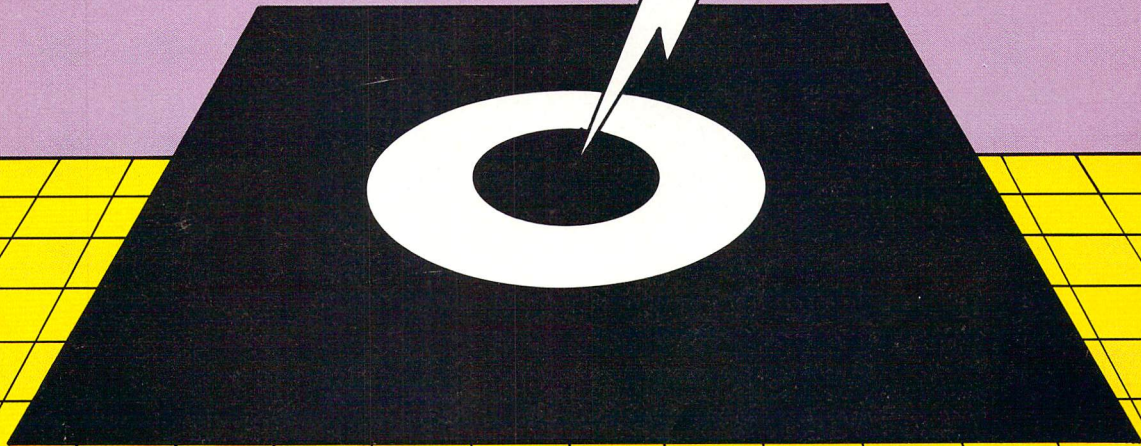
AMIGA is a registered trademark of Commodore Business Machine.

I N T R O D U C I N G

www.Commodore.ca
May Not Reprint Without Permission

SUPER KIT 1541

Has it all!



BY MARTY FRANZ & JOE PETER

SINGLE/DUAL NORMAL COPIER

Copies a disk with no errors in 32.68 seconds. dual version has graphics & music.

SINGLE/DUAL NIBBLE COPIER

Nibble Copies a disk in 34.92 seconds. Dual version has graphics & music.

SINGLE/DUAL FILE COPIER

7 times normal DOS speed. Includes multi-copy, multi-scratch, view/edit BAM, & NEW SUPER DOS MODE. In Super DOS Mode, it transfers 7-15 times normal speed, copies 150 blocks in 23 seconds.

TRACK & SECTOR EDITOR

Full editing of t&s in hex, dec, ascii, bin. Includes monitor/disassembler with printout commands.

GCR EDITOR

Yes disk fans, a full blown sector by sector or track by track GCR Editor. Includes TRUE Bit Density/Track Scan.

3 SUPER DOS FAST LOADERS

Over 15 times normal DOS speed. Super DOS Files are still Commodore DOS compatible. Imagine loading 150 blocks in 10 seconds.

SUPER NIBBLER/ SUPER DISK SURGEON

Quite frankly, these will provide you the user with the backup you need! Even copies itself.

\$29.95 u.s.

PLUS \$3.00 SHIPPING/HANDLING CHARGE — \$5.00 C.O.D. CHARGE

PRISM
SOFTWARE

401 LAKE AIR DR., SUITE D • WACO, TEXAS 76710
ORDERS (817) 757-4031 • TECH (817) 751-0200
MASTERCARD & VISA ACCEPTED

See center page for
mail order card.

SUPER KIT/1541 is for archival
use only! We do not condone
nor encourage piracy of any kind.

THE TIME SAVER



J. MOSTACCI

Type in a lot of Transactor programs?
Does the above time and appearance of the sky look familiar?
With The Transactor Disk, any program is just a LOAD away!

Only \$8.95 Per Issue
6 Disk Subscription (one year)
Just \$45.00
(see order form at center fold)

Also check out the TransBASIC Disk
Complete with 24 page manual, just \$9.95!
See The TransBASIC Column in this issue.