www.Commodore.ca
May Not Reprint Without Permission

The Transactor

The Tech/News Journal For Commodore Computers

95% Advertising Free! Jan. 1987: Volume 7, Issue 04. $3.50

Gizmos and Gadgets

FREE T–Shirt Offer!
see page 77

Plus: CompuServe, Transactor's New Online Headquarters!

And: Stamping out 1541 REL File Bugs!

CDC 00594

www.Commodore.ca
May Not Reprint Without Permission

Good news!

If you want to get the most out of your Commodore 128 or 64, we have goods 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?

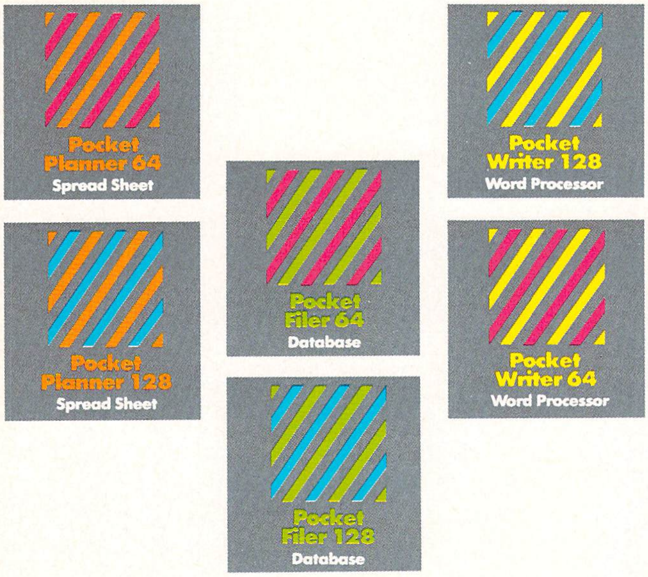
Pocket 128 or 64 packages have all the power you're ever likely to need. They have all of the features you'd expect in top-of-the-line software, and then some. The good news is that Pocket 128 or 64 Software Packages are priced way down there . . . where you can afford them.
Fast, powerful, easy to learn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

www.Commodore.ca
May Not Reprint Without Permission

Solutions!

Pocket Writer 64
Word Processor

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

Pocket Writer 128
Word Processor

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

Pocket Planner 64
Spread Sheet

Pocket Filer 64
Database

Pocket Filer 128
Database

Pocket Planner 128
Spread Sheet

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.

Digital Solutions Inc.

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

Pocket Writer 128 and 64 are now available in French.

www.Commodore.ca
May Not Reprint Without Permission

THE TIME SAVER

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.

www.Commodore.ca
May Not Reprint Without Permission

Volume 7
Issue 04
Circulation at Large
72,000

The Transactor

Gizmos and Gadgets

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

www.Commodore.ca
May Not Reprint Without Permission

The Transactor
The Tech/News Journal For Commodore Computers

Editor in Chief
Karl J. H. Hildon

Editor
Richard Evers

Technical Editor
Chris Zamara

D'Artagnan Editor
Nick Sullivan

Art Director
John Mostacci

Administration & Subscriptions
Anne Richard
Kathryn Holloway

Contributing Writers

Ian Adam, Jim Barbarello, Tim Bolbach, Anthony Bryant, Tim Buist, John Bush, Jim Butterfield, Betty Clay, Gary Cobb, Jack Cole, Tom K. Collopy, Robert V. Davis, Elizabeth Deal, Rolf A. Deininger, Frank E. DiGioia, Paul T. Durrant, Michael J. Erskine, Jack Farrah, William Fossett, Jim Frost, Miklos Garmaszeghy, Martin Goebel, R. James de Graff, Tim Grantham, Bob Hayes, John Holttum, David Hook, Tomas Hrbek, Robert Huehn, Tom Hughes, David Jankowski, Bob Jonkman, Mark Jordan, Clifton Karnes, Lorne Klassen, Jesse Knight, James E. LaPorte, William Levak, James A. Lisowski, Scott Maclean, Don Maple, David Martin, Steve McCrystal, Stacy McInnis, Jim McLaughlin, Steve Michel, Chris Miller, Terry Montgomery, Michael Mossman, Gerald Neufeld, Noel Nyman, Kevin O'Connor, Dave Pollack, Richard Perrit, Terry Pridham, Raymond Quirling, Gary Royal, John W. Ross, David Shiloh, Fred Simon, P. A. Slaymaker, Edward Smeda, Darren J. Spruyt, Aubrey Stanley, David Stidolph, Richard Stringer, Karel Vander Lugt, Audrys Vilkas, Steven Walley, Jack Weaver, Evan Williams, Chris Wong

Production
Attic Typesetting Ltd.

Printing
Printed in Canada by
MacLean Hunter Printing

Program Listings In The Transactor

All programs listed in The Transactor will appear as they would on your screen in Upper/Lower case mode. To clarify two potential character mix–ups, zeroes will appear as '0' and the letter "o" will of course be in lower case. Secondly, the lower case L ('l') 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**	T
Up	Q	**Delete**	t
Right	]	**Clear Scrn**	S
Left	[Lft]	**Home**	s
RVS	r	**STOP**	c
RVS Off	R		

Colour Characters For VIC / 64

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

Function Keys For VIC / 64

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

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

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.

Quantity Orders:

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

Master Media
261 Wyecroft Road
Oakville, Ontario
L6J 5B4
(416) 842 1555
(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, 04

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

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.

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.

www.Commodore.ca
May Not Reprint Without Permission

Start Address

Not Enough Minutes in an Hour!

About 5,356,800 seconds ago I was ending my last editorial. And believe me, every one of those seconds were squeezed for every fraction! If anyone's interested, I always write page 3 last. Once the other 79 pages are complete, it allows me to concentrate on this task alone, as opposed to dealing with 79 others simultaneously. In about 11 hours from right now, I'll be boarding a flight to L.A. for the WCCA show that starts Saturday – and I have yet to pack! This kind of time accounting has been daily routine since the last Start Address, and squeezing a summary into one page is going to be a challenge. Here goes.

After catching up on some much needed sleep (re: V7, I03, pg3) it was back to work on the Bits Book. The typesetting equipment centers around a 12 year old, 10 meg hard drive – the kind with the removable platter. It's a multi-user system with 4 work stations. The odd read/write error meant re-booting the system from scratch (3 min.). Occasionally a "refresh" would be necessary to unscramble files containing hours of work (20 min.).

Meanwhile, Chris was working frantically on the 1541 upgrade ROMs, Richard was drowning in articles for this issue, and no sooner was the Bits Book done, when CompuServe calls requesting we meet to discuss the operation of their Commodore section. August 3 I was on a plane to Columbus (no long weekend for me). Airport to CompuServe HQ (25 min.). At 3:00 AM we weren't half way through the list of details. 8:30 Monday morning we were back at it, and didn't stop 'till after midnight.

Previous to this I had already planned to visit Capitol Distributing in Derby Connecticut. Between there and Columbus lies West Chester. Tuesday at 7:15, take off for Philadelphia. I dropped in on Paul Higginbottom, Dave Berezowski, Liz Deal, Bob Albright, and a number of others. Three 17-hour days later, I'm back at Philadelphia being told my luggage isn't going to make it to my plane bound for New Haven, and of course, the gate is the furthest one down the corridor. Philly to New Haven (1 hour), to Capitol (25 min.), and through a list of magazine distribution concerns in one afternoon. Back to New Haven, land in LeGuardia, off to Toronto, arrive Friday the 8th, 9 PM.

Ah, this weekend I'm going to relax, or so I thought. Waiting for me at home was my CompuServe manuals, and a package of hardware from Intelligent I/O. Then I get a call from The Toronto PET Users Group. "How much would you charge us to supply Transactors to TPUG members with a bound-in TPUG insert?". Coming up with a price was the simplest part. The details involved would prove to be enormous. Foremost was the extent of subscribers that subscribe to both TPUG and The T. Fortunately our mailing lists are both maintained using the IBM Manager. A quick analysis (10 hours, thanks to Rich and Chris) would show an overlap of just 350 dual subscribers. A meeting or two later, it was set – the next Transactor would be supplied to almost 9,000 more people than before.

IRQ: For this issue only, there will be about 350 subscribers receiving two copies of The T. One will contain a TPUG insert, one won't. We have a plan to eliminate this duplication, but there just wasn't time to implement it for this issue. A refund would be impractical as some U.S. subscribers would end up paying $7.00 U.S. to cash a cheque for an average of $7.50 Canadian. We've tossed around several ideas including free books, disks, etc., gift subscriptions, and extending subscriptions. One way or another, if you're part of the subscriber intersection set, you'll receive the full dollar value of material you paid for, if not more. The next issue will have all the details.

RTI: September 1, Nick Sullivan, Editor of TPUG Magazine joins The Transactor. Producing the insert would require time that none of us had, not to mention the additional task of managing the CompuServe activity. The issue you're holding was already underway, and the typesetting equipment was feeling more ill than ever. System crashes were more frequent, approaching logarithmic – where "number of terminals in use" was the exponent. Needless to say, this was making it difficult to get any work done, and the trip to L.A. was coming up fast. If the T. wasn't finished, I would have to cancel. Donna and Richard are expecting a new addition to their family, and it was looking like Nick was about to take his first business trip.

Then the ultimate disaster. Tuesday September 2 it was raining most of the day, and well into the night. I left the typeshop at about 4:00 AM., only to arrive home and find two feet of water at the bottom of my stairway. It took about 3 seconds to sink in, that if there's two feet of water at the bottom of the stairs, there's also two feet of across the entire basement! This, of course, includes my computer room where I do nearly all of my end of the production. The power bar to my equipment was completely submerged. Also, a VCR, an oscilloscope, guitars, amplifiers, our TV, hundreds of books and magazines (most collectors items), the Anthology original film, two drawers full of diskettes, our furniture and carpeting, washer/dryer, furnace, floor freezer, and dozens of other items were damaged or completely ruined. About $12,000 in losses total. Wednesday we gutted the entire basement. Our driveway and backyard had so much strewn about articles, it looked like a garage sale convention.

Surprisingly, not one piece of computer equipment was affected. In fact, my SuperPET was still running my terminal program to the modem and the 64 was still flashing its cursor! So much for the theory of unfriendly relations between water and electricity. A few days later my transformer to the 64 packed it in, but I think it was approaching fubar anyway. And I must admit, our TV converter box was burnt to a crisp! The whole ordeal sliced about 4 days out of my forecast.

Well, our basement is almost dry, CompuServe is buzzing and we're all getting up off the steep part of the learning curve, the magazine is done, and I'm going home to pack – I've got 9 hours. So, correction, it was 5,308,200 seconds ago I was typing. . .

There is nothing as constant as change, I remain

Karl J.H. Hildon, Editor In Chief

and I just remembered, the cover still isn't finished - Arghg!

www.Commodore.ca
May Not Reprint Without Permission

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 828  to enable the C64/VIC version     (off: SYS 831)
SYS 3072,1 to enable the C128 version      (off: SYS 3072,0)
```

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 (verifizer 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.

VIC/C64 VERIFIZER

```
KE  10 rem* data loader for ' 'verifizer' ' *
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
```

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
```

www.Commodore.ca
May Not Reprint Without Permission

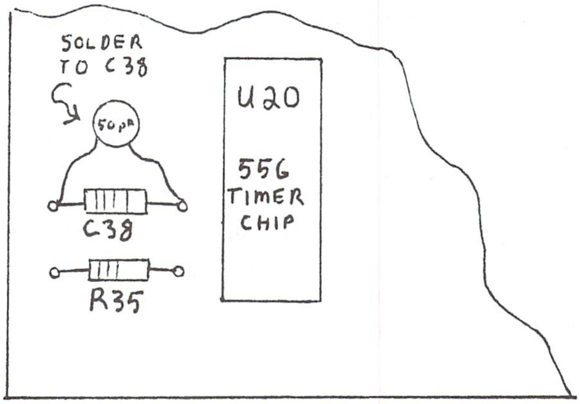
b i t s

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

C-64 RESTORE Key Sensitizer

Paul Bahlawan
Mississauga, Ont.

On some 64s the RESTORE key has to be tapped several times before the computer will respond. With reference to the C-64 schematic you can see the RESTORE key is coupled to the 556 timer chip with a capacitor. This capacitor will only allow high frequency pulses to be passed, therefore it is necessary to tap the key quickly. By soldering a 50 pF capacitor (marked "500") in parallel with C38 you allow lower frequency pulses to pass. (Any low value capacitor should work, but 50 pF seems fine.) Now the RESTORE key will respond to normal keystrokes, which is much nicer than a lot of tapping.



A Quirk In Calculated Array Subscripts

Arne Storjohann
Scotland, Ont.

Type in the following bit of code and run it:

```
10 a$(0) = "cell 0"
20 a$(1) = "cell 1"
30 x = 2.1 - 1.1
40 print "a$(";x;")=";a$(x)
```

Since the variable X equals one, the string "cell 1" should be printed in line forty. Right? Wrong! Because of the fact that all decimal numbers can't be converted exactly (only a close approximation can be achieved) to floating point numbers and vice-versa, the value of the variable X given as 'X=(2.1 - 1.1)' will be stored differently than if it were given as 'X=1'. Since array subscripts can only be integer values any decimal portion of a calculated array subscript is simply chopped off. This leads to the quirk in line forty. Change the 'a$(x)' to 'a$(int(x+.05))'. This will take care of the problem. Any time you have to calculate an array subscript using non-integer values it's a good idea to use the INTeger function in this way.

Unassembler Files to SYMASS 3.13

Lorne Chartier
Calgary, Alta.

Volume 7, Issue 01 introduced an exceptional, PAL compatible assembler entitled SYMASS 3.13. The assembler featured a wide variety of functions that were extremely useful for assembling quality machine code. However, without a compatible disassembler, you cannot edit or examine ML programs that lack a source file. Fortunately, with a little ingenuity and the help of a couple of previous Transactor programs, you can easily remedy this problem. Type in the unassembler from Volume 6, Issue 04. Following are the changes to the program to make it SYMASS compatible.

```
 172 input"starting line number";ln
 174 input"increment";li
1185 p$="[SPACE]sys 700":gosub 2150
2150 p$=str$(ln)+"[SPACE]"+p$+xx$: ln=ln+li
2155 print#6,p$;: gosub 2220: lc=lc+1: return
```

Now save the program. When run, it will ask for a starting line number, and a line increment. This is the feature that makes it compatible with SYMASS -- any disassembling will create a sequential source file to disk that will include sys 700 as the first line, and line numbers before each line. The final step is to turn

www.Commodore.ca
May Not Reprint Without Permission

this sequential file into a BASIC–format (SYMASS compatible) program using Chris Zamara's STP program from Volume 5, Issue 06; or the C-64 BASIC STP found in the bits and pieces column in the same issue as the unassembler. Use STP to convert the file to BASIC, then save the resulting source. This file is entirely compatible with SYMASS 3.13, and can be assembled immediately after loading. Once you have changed the unassembler to its new format, the conversions take no time at all.

Using the DOS Wedge With Two Drives

Joel Pickett, Levelland, Texas

I use the DOS support program that comes with the 1541 disk drive. I have two drives, but the DOS program only works on one. I modified the DOS loader so it will run on the drive it is loaded from. To do this, line 5 (below) was added -- it peeks location 186, which holds the number of the last device used. Also, the 'dv' in line 10 replaces the '8'.

```
5 dv=peek(186): rem  location 186 is current device #
10 if a=0 then a=1: load"dos 5.1",dv,1
20 if a=1 then sys 12*4096+12*256
30 new
```

The DOS support program (at $CC00) gets the current device number from location 186 and stores it internally at $CC77 (52343). Whenever you want to use a DOS command on another drive, simply POKE 52343,(device number).

Should you disable the DOS with a warm start (sys 64738), you can often run it again this way:

```
poke 186,8: sys 52224: return
```

Fast File

Rick Nash, Millersburg, Ohio

Here is a short utility that can speed up programs that read from disk files. It works with any kind of file, but it especially handy for direct access (reading a given sector), since the INPUT command is not always reliable under these circumstances. The INPUT command stops reading data whenever it sees a delimeter character (carriage return, colon or comma), so to read unpredictable data the GET command must be used to read the bytes one at a time. This is far too slow for most applications. The program below, Fast File, will read a given number of bytes from a disk file into a string variable, and only stop reading when the given number of characters have been read, or end of file occurs. It reads the data as fast as the disk drive can supply it, since the program is in machine language.

The syntax for using Fast File is:

```
sys 49152,#f,n,v$
```

where 'f' is the file number (the # must be present), 'n' is the number of characters to read, and 'v$' is the name of a string variable that will receive the data.

For example, to read a sequential file:

```
1000 open 1,8,2,"file"
1010 sys 49152,#1,255,a$
1020 print a$;
1030 if st=0 then 1010
1040 close 1
```

To read 128 bytes of track 18, sector 0 (you can't read all 256 bytes of a sector, since a string can only hold 255 bytes):

```
1000 open 15,8,15
1010 open 2,8,2,"#"
1020 print#15,"u1:";2;0;18;0
1030 sys 49152, #2, 128, a$
1040 print a$
1050 close 15
```

The program is fully relocatable; just change the assignment in line 30 of the BASIC loader below. Using Fast File instead of GETs will give you typical speed increases of nine to eleven times!

```
NK  10 rem** fast file **
NE  20 rem read from a file into a variable
PG  30 a=49152: rem  program is relocatable
AA  40 print"usage: sys";a;",#<file#>,<# bytes>,
       <string var$>"
BK  50 for i=a to a+85: read d: c=c+d: poke i,d: next i
HC  60 if c<>11661 then print"!data error!": stop
KL  70 :
HC  100 data  32, 253, 174, 169,  35,  32, 255, 174
GM  110 data  32, 158, 183, 134, 251,  32, 253, 174
EN  120 data  32, 158, 183, 134, 252,  32, 253, 174
HN  130 data  32, 139, 176, 133,  73, 132,  74,  36
IN  140 data  13,  48,   3,  76, 153, 173, 165, 252
EP  150 data  32, 125, 180, 166, 251,  32, 198, 255
AO  160 data 176,  15, 165, 252, 240,  26, 160,   0
OM  170 data 165, 144, 208,   8,  32,  19, 238, 144
MA  180 data   8,  76, 249, 224, 132,  97,  76,  80
DP  190 data 192, 145,  53, 200, 196, 252, 144, 232
OA  200 data  32, 204, 255,  76, 100, 170
```

Modifying The Epyx Fast Load Cartridge

James Craig, Waco, TX

When using the Epyx Fast-load cartridge with the C-128, you have to shut off the machine and install the cartridge in order to switch from C-128 to 64 mode. Besides being a nuisance, this can quickly wear out the cartridge port.

I decided something had to be done. I took the Fast Load cartridge apart and found that my troubles were little ones. I installed a switch in the "EXROM" line to take the ground off the circuit when using C-128 mode. By throwing the switch to connect the ground and hitting the reset button, I was immediately in C-64 mode with the Fast Load cartridge enabled! To go back to C-128, just throw the switch to disconnect the ground, then hit reset again.

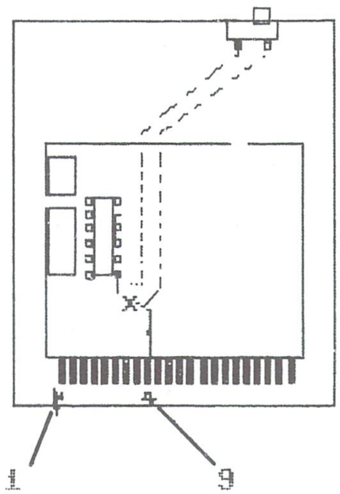
To open the cartridge, feel around the top surface for the indentation of the screw that holds the unit together. Just cut away enough to remove the screw. Cut around the box at the seam. then using a

www.Commodore.ca
May Not Reprint Without Permission

knife blade, pry up all around the box and lift straight up to avoid damaging the interlocking catches.

Install a SPST slide or toggle switch at any convenient location. This could even be outside the case someplace. Cut the printed circuit lead from the #9 male prong about where it makes a bend going to the EXROM connector. Solder a wire on each side and run to each terminal of the switch -- it doesn't make any difference which wire goes where on the switch. Reassemble the case and you're in business. Enjoy your C-64 again!

FAST LOAD



1541 Disk Swap Checker

John Chong, Syracuse, NY

The following program waits until the current disk in the drive is removed, and another disk (or the same one) re-inserted. It does this by checking the write-protect status of the drive to see if a disk is there or not. It only works if the disks being inserted are NOT write-protected, and even then it can be fooled if you partially remove and then re-insert the disk. Although not bullet-proof, the program shows the technique of checking the write-protect status, and the subroutine at 3000 that actually does the checking may come in handy in one of your programs.

```
2000 print" please change disks. "
2010 open 15,8,15
2020 gosub 3000: if a<>0  then 2020
        :rem  wait for disk to be removed
2030 gosub 3000: if a<>16 then 2030
        :rem  wait for no disk in drive
2040 gosub 3000: if a<>0  then 2040
        :rem  wait for disk to be inserted
2050 for d = 1 to 1500: next: close 15
2060 print" ok, thanks! "
2070 end
2080 :
3000 print#15, " m-r " ;chr$(0)chr$(28)chr$(1)
        :get#15,a$:a=asc(a$)and16:return
```

Easy Retrieval of Last Filename Used

Dave Newberry, Duluth, Minnesota

In the Bits & Pieces section of Volume 6, Issue 06, Jeffrey Coons wrote in with a one-liner that allowed you to find the name of the last file used (Finding the missing file page 5). Though the line works well, there is an easier way to achieve the same result. A single SYS call is all it takes to get the name of the last file accessed. The magic number is 62913. A **SYS 62913** will print the filename on the screen for all to see.

Chromatic Scale Register Values

Arne Storjohann, Scotland, Ont.

The following routine generates the SID chip register values which correspond to eight octaves of chromatic scale. The values are separated into high and low byte format and stuffed into two ninety-six element integer arrays to allow for maximum speed of use later in your BASIC program. Due to the ninth place constant D, the values generated are exceedingly precise, limited in resolution only by the 1 through 65535 range imposed by the SID chip. The usual approach is to use data statements and read the 192 values into an array, but with a running time of less than three seconds, this routine is much more compact, efficient, and above all, a more elegant solution.

Anyone who has ever tried to program music on the 64 will appreciate this algorithm!

```
LI  110 rem** routine to generate chromatic
MP  120 rem** scale register values (hi/lo)
AO  130 rem** by arne storjohann - 86,05,04
AA  140 :
EH  150 dim lo%(95),hi%(95): g=2↑(1/12)
DK  160 f=3520*g*g: d=0.06095948: b=256
FF  170 for i = 95 to 0 step -1: n=f/d: hi%(i)=n/b
DP  180 lo%(i)=n-hi%(i)*b: f=f/g: next
CD  190 :
LH  200 rem ** demo **
GE  210 :
MM  220 s=54272: for i = s to s+15: poke i,0: next
BB  230 poke s+5,96: poke s+6,251: poke s+4,33
OA  240 poke s+24,15: for i=-72 to 72
FJ  250 x=71-abs(i)+16: poke s,lo%(x)
EF  260 poke s+1,hi%(x):for j=1to200: next
EF  270 next: poke s+4,32: end
```

C-64 Underlined Characters

D. Munro, Port Elizabeth, South Africa

This program is based on the C-64 italics program in Bits & Pieces, Volume 7 Issue 01. Instead of giving italics in place of reverse characters however, it gives underlined characters. Both of the 64's built-in character sets are altered, so that underlined letters are available from either upper/lowercase or graphics modes. The new character set is located from 8192 (hex $2000) to 12287 ($2FFF). Consequently, the start of BASIC is moved to $3001.

www.Commodore.ca
May Not Reprint Without Permission

After running the program, the normal characters are unchanged but all reversed characters are replaced by underlined characters. Due to the fact that reversed characters no longer exist, the cursor is now denoted by a flashing underscore instead of a reverse space. When the cursor is moved over a character, it just flashes an underscore beneath the character instead of flipping it to and from reverse field. To return to the normal character set, hit RUN/STOP–RESTORE or POKE 53272,21.

After running "underline", all BASIC programs may be loaded and saved normally, as the operating system takes care of relocating to the new start of BASIC. Just be sure to LOAD with ',8' instead of ',8,1'.

Here's the program. *Make sure you SAVE it before running!*

```
DP  10 rem* data loader for " underline " *
LI  20 cs=0
KG  30 for i=49152 to 49257:read a:poke i,a
DH  40 cs=cs+a:next i
GK  50 :
AB  60 if cs<>14259 then print" !data error! " : end
AD  70 sys 49204
EM  80 print chr$(147); " poke 44,48: poke 12288,0: new "
IE  90 print chr$(18); " reverse characters are underlined! "
MC  100 poke 631,19: poke 632,13: poke 198,2: end
CO  110 :
BP  1000 data 162,  16, 160,   0, 185,   0, 208, 153
BN  1010 data   0,  32, 200, 208, 247, 238,   6, 192
BH  1020 data 238,   9, 192, 202, 208, 238,  96, 162
MB  1030 data   8, 160,   0, 177, 251, 202, 208,   4
IE  1040 data 162,   8, 169,   0,  73, 255, 145, 251
KE  1050 data 200, 208, 240, 230, 252, 165, 252, 197
PM  1060 data 253, 208, 232,  96, 173,  24, 208,  41
IF  1070 data 241,   9,   8, 141,  24, 208, 120, 169
NE  1080 data  51, 133,   1,  32,   0, 192, 169,   0
KM  1090 data 133, 251, 169,  36, 133, 252, 169,  38
GG  1100 data 133, 253,  32,  23, 192, 169,   0, 133
BN  1110 data 251, 169,  44, 133, 252, 169,  48, 133
OD  1120 data 253,  32,  23, 192, 169,  55, 133,   1
ID  1130 data  88,  96
```

Machine Language Debugging Tip — **John Augustine, Reading, PA**

It is hard to avoid mistakes. In fact, I am reminded of Murphy's Law more than ever when composing machine language source code. To help me track down what sections of code are executing and what sections are not, I use an area of memory that I initialize with zeroes using an ML monitor or other means. Then, at strategic points in my code, I add a simple 'INC ADDRESS' (the start of the area initially filled with zeroes). At other points, I 'INC ADDRESS+1', then ADDRESS+2, etc., making notes of the program locations for reference. After you set up all of your test points, assemble your source and test-run the resulting object code. After your program has run, or you've exited with a RESTORE or reset, use an ML monitor or PEEKs from BASIC to examine the contents of your test area of memory. The numbers you see will show you if the parts of your program with the INC instructions executed, and how many times they were executed (up to 255).

One word of caution when using this technique: be careful that you do not put the INC instructions at points in your program where the state of the processor status flags are vital. For example, DO NOT insert the INC instruction between a compare and branch instruction, as the INC will alter the flags and cause an incorrect branch. If you must put the INC in such a location, or you're not sure if you need the status flags, just put a PHP instruction before, and a PLP instruction after the INC to save and restore the processor status register.

Twisted Sister Goes Digital — **Kevin Smith, Edmonton, AB**

Yes, now you too can convert your $1,000 computer system into a $10 cassette player! First enter this short machine language routine into your Commodore 64. Now the hard part: try to remember where you left your ancient datassette.

Next, pop in your favourite cassette tape and listen to your computer choke on "Twisted Sister".

```
100 for i = 49152 to 49180: read a: poke i,a: next
110 print " press play on tape " : wait 1,16,16: sys 49152
120 data 169,  11, 141,  17, 208, 169,   7, 133
130 data   1, 173,  13, 220,  41,  16, 240, 249
140 data 169,  15, 141,  24, 212, 169,   0, 141
150 data  24, 212,  76,   9, 192
```

Touch Typer's Trick — **James Yost, Boston, MA**

For touch typists who would like to find home position by touch after hitting RETURN: place a small drop of epoxy in the centre of the index finger home keys. That raised dot saves plenty of looking back at the keyboard. Never leave home without it!

Commodore 128 Bits

Program Stashing — **Charles Van Lingen, Mossley, Ont.**

When I purchased a 1750 RAM expansion unit for my C–128 I was eager to use it with my BBS software to switch between BASIC programs. One would tend to think that you could store and retrieve a program from a RAM bank with the following statements:

```
stash 45000, 7168, 7168, [bank #]
fetch 45000, 7168, 7168, [bank #]
```

This *does* work if you only wish to run the program in the other bank and not edit it, but the top of text pointer must be set to allow editing. I came up with this formula which I define as function keys in my programs:

```
key 4, " b=[SPACE]:slow:bank0:stash 2,4624,4624,b
  : stash 45000,7168,7168,b:bank15:fast " + chr$(27) +
  " j " + chr$(29) + chr$(29)

key 6, " b=[SPACE]:slow:bank0:fetch 2,4624,4624,b
  : fetch 45000,7168,7168,b:bank15:fast " + chr$(27) +
  " j " + chr$(29) + chr$(29)
```

(Note: leave out the FAST command in 40 column mode)

www.Commodore.ca
May Not Reprint Without Permission

When you use these keys, enter a bank number from 0–7 (or 0–1 if you have a 1700) for your program to be stored to or retrieved from, then press RETURN. In this way, you can work on up to 8 programs simultaneously, quickly switching from one to another as the need arises. This isn't a particularly efficient way to use the extra memory but it is quick and painless and provides a sort of crude (but FAST) ramdisk. According to the manual, the fetch and stash commands work at one megabyte per second, but I haven't bothered to check it out. Anyway, I highly recommend the expansion unit if you are into programming and I hope these keys help.

C–128 Additional BASIC

Ian Adam
Vancouver, BC

So you think the Commodore 128 is a fantastic improvement over the 64, because of all those extra features -- in fact, *everything you could possibly want* is right there in that computer! Wrong, byte breath! There's something they didn't tell you about.

Yes it's true: BASIC 7.0 contains an additional command that isn't documented in either the 128 System Guide or the Programmer's Reference Guide. The extra command is RREG, and it returns the values contained in the CPU's registers after the last SYS command to whatever variables you specify.

The main application of this is following a machine–code routine. SYS has been expanded to allow passing variables to the routine, and RREG provides the reverse function, getting values back. The syntax is also the same:

```
sys 4864,1,5,5,0: rem jump to code and place values shown
                      in the a, x and y registers

rreg a,x,y,s: rem put register values in variables shown
```

Accessing the 80–Column Chip

Ian Adam

David Stidolph's article in Volume 7 Issue 03 showed how to work the registers in the 8563 video controller. This allows the programmer access to a wide variety of fascinating capabilities.

Because BASIC was seen as being too slow, David provided short machine language routines for reading and writing to the registers. There is a way to get at the chip from BASIC, however. Assuming you're still in BANK 15, there are ROM routines to take care of the details.

The routine at 52684 will write the value in the accumulator to the video chip register specified in X, while that at 52698 will read a register. For example, this program will list the current value of all registers:

```
for i=0 to 36: sys 52698,0,i: rreg a: print i,a: next i
```

This program will allow you to tinker with the registers at will. Of course, you will need David's table showing the description of each register.

```
10 do:
20  input "register #";x
30  sys 52698,0,x: rreg a
40  print "current value";a
50  input "new value";a
60  sys 52684,a,x
70  for i=1 to 8: print "0123456789";: next
80 loop
```

C–128 HELP and RUN/STOP definition

Tim Thompson
Gadsden, AL

The Commodore 128 actually has ten programmable function keys. Eight of them are the normal ones located above the numeric keypad. The ninth is the SHIFted RUN/STOP key, and the tenth is the HELP key. While the first eight have a built–in BASIC command to reprogram them, the other two do not. There is a Kernel routine, however, which will reprogram any of the ten. The following program will re–define the SHIFTed RUN/STOP key to simply RUN the program (instead of LOAD and RUN).

```
10 z9$="run"+chr$(13)
20 z8 = 9: rem 9 = shift-run/stop, 10 = help
30 for jj=1 to len(z9$)
40 poke 3071+jj, asc(mid$(z9$,jj,1))
50 next: poke 250,0: poke 251,12
60 sys 65381, 250, z8, len(z9$)
```

To re–define any programmable key, simply set Z9$, in line 10, to what you want the key defined as (including a carriage return if needed). Set Z8 equal to the number of the key to redefine. Function keys F1 through F8 are key numbers 1 through 8. The SHIFTed RUN/STOP key is key 9, and the HELP key is 10. You can use this as a subroutine in any BASIC program.

C–128 80–column CHAR bug

Richard D. Young
Greenwood, N.S.

I would describe this as a minor bug: it is potentially disastrous but is easy to avoid. The problem occurs only in 80–column (RGB) mode, and when the CHAR instruction is executed. It affects two memory locations in RAM 0, specifically $D600 and $D601 (54784 and 54785). These two memory locations are clobbered, leaving $D600 with $0F (15) and $D601 with some number that varies with the cursor location set by CHAR. Avoiding the problem is as easy as avoiding use of these two memory locations (few BASIC programs are that long), restoring proper values after execution of CHAR, or avoiding CHAR.

It appears that an image of the 80–column video controller (VDC) registers at $D600 and $D601 are left in RAM 0 when CHAR is executed in 80–column mode. The value $0F refers to the VDC register that controls cursor position, low byte, and the value in $D601 is the value of the cursor position.

To confirm that the problem exists (it may not in all machines), store some number other than 15 in location $D600 (BANK 0), execute a CHAR instruction to print something on the screen, then check $D600 (BANK 0) for the value 15.

www.Commodore.ca
May Not Reprint Without Permission

RAM 0 is, of course, the area for BASIC programs. One way of avoiding disaster with the occasional very long BASIC program is by doing:

```
bank 0: a=peek(54784): b=peek(54785): bank 15
```

. . .before the CHAR command, then:

```
bank 0: poke 547854,a: poke 54785,b: bank 15
```

. . .after. If this area of memory must be used normally (the DOS SHELL utility for example), CHAR should be avoided in very long BASIC programs in 80–column mode.

Protect Those Vectors! — **Philip C. Herold, Seattle, WA**

We all know what pressing RUN/STOP–RESTORE on the 64 does to our IRQ–driven wonders: it resets the IRQ vector and disables them. That doesn't have to be the case on the 128. The BASIC warm–start entry is vectored through $0A00. So after a RESTORE resets the Kernel and interrupt vectors, we can intercept the warm–start routine at its BASIC entry point and put our vectors back. Here's one way to accomplish it:

```
entry   lda  #<setback  ;change the basic entry vector
        sta  $0a00
        lda  #>setback
        sta  $0a01
        jsr  setirq
        rts

setback jsr  setirq
        jmp  $4003

setirq  sei
        lda  #<irqrtn
        sta  $0314
        lda  #>irqrtn
        sta  $0315
        cli
        rts

irqrtn  . . .           ;irq–driven routine starts here
        . . .
        . . .
        jmp  $fa65      ;exit through end of irq routine
```

Keep the code in bank 15, below $4000, to avoid problems. This technique can be applied to any vectors that a warm–start resets, not just the IRQ vector.

Amiga Bits

Printing Greeting Cards with Deluxe Paint — **Lindsey Fong, Sacramento, CA**

Can you believe us greeting card makers have no program yet?! While waiting for the "PRINT SHOP" or "DELUXE PRINT" to be released, I have figured out a way to print greeting cards with "DELUXE PAINT" and my Okimate 20 printer. Here's how it works. When you load DELUXE PAINT and get the CLI prompt, type "preferences". Set the page length to 32, right margin to 5 and left margin to 50. Select the "graphic select" icon and set ASPECT to "vertical" and "SHADE" to "grey scale" or "black and white". Now, close preferences, and enter "dpaint" to run the program.

Now you can "paint" the front of the card using the full screen for your canvas. Don't forget to paint under the control panel by hitting F10 so your picture will be centred on the paper.

Fortunately, DELUXE PAINT has text capability, so you can type messages with your picture. I would suggest that you set you pallette to shades of grey to get a better idea of how your card will look when it prints.

Lining up your paper for printing will depend on the type of printer you have, but I line up the left edge at the "10/9.5" marker box on the printer. The top edge should line up with the top of the printhead. Lining up the paper is not so critical if you use a white background and don't paint near the edges of the screen.

To print, select "print" from the menu. In a few seconds, the front of your card will print. The picture will print sideways on the bottom left quadrant of the paper –– that's what you want. Now clear the screen and work on the inside of your card. To print the inside of your card, remove your previously printed paper, turn it around and insert the opposite side, lining up the paper as before.

Now you can print the inside of your card. If you have an Amiga with 512K, you can use the spare screen option (hitting 'j') and work on completing both pictures first before you print your card.

After you're finished printing, you should have the front of your card on the bottom left quadrant of the paper and the inside of the card on the top right quadrant of the paper, upside–down. Now french fold the paper and PRESTO! You have a greeting card.

This method may take a bit longer than making a card on the PRINT SHOP, but unlike the versions of the PRINT SHOP now available for other computers, you have TOTAL control of how you want your card to look. You are not limited in graphics or lettering placement.

Happy card making!!

The Autographed Amiga — **Joe Foos, Santa Barbara, CA**

The Amiga people have done something very interesting, even though they were not the first: If anyone has opened their Amiga yet, they have probably already noticed that molded into the inside of the top cover are the signatures of all the people involved in designing the Amiga. In case you ever wanted your Amiga to be autographed by one of your Amiga heroes (R.J. Mical, Dale Luck, Robert Pariseau or any others), then your wish has come true. Perhaps this only goes to show how proud the Amiga designers are that they are involved in personal computer history.

www.Commodore.ca
May Not Reprint Without Permission

Letters

No Fun In GAMES: You would do well to read up on truth in advertising. The cover of Sept. '86 "The Transactor" shows an Amiga and its amazing graphics. Then you add GAMES to it. I was thoroughly sucked in. I'm green in the personal computing field and considering buying an Amiga. Buy the mag, get home, open it and what do I get? Data files, tricks on programming, number crunching, etc., etc., ad nauseam. Rest assured it won't happen again.

P.S. Commodore makes a good product. Too bad "The Transactor" smudges its reputation. D. Fraser, Lethbridge, Alberta

It's pretty clear our GAMES issue wasn't quite what you were expecting. Still, I'd ask you not to throw away that issue. When you have a bit more programming under your belt you'll probably find it a lot more useful than you do now. You might even find it entertaining.

When The Transactor covers a particular application field of programming - such as games - we don't tend to provide complete and ready-made example programs for our readers to type in. Instead, we try to explore what makes those programs tick, to provide tools and methods that readers can make use of in their own programs. At the same time, remember that programming is programming, whether the end product is to be a game, a spreadsheet or an operating system, so you'll notice certain common themes - like data files, programming tricks and number crunching - showing up again and again, each time from a somewhat different perspective.

The magazine you acquired is not the magazine you wanted. But we have good evidence for believing that many readers do want a magazine that gets heavily into the technical side of programming and, as I said above, we hope that at some point you'll be one of them.

C128 Memory Questions Plus More, As Addressed To Jim Butterfield: For several years now I have enjoyed reading your articles about the Commodore 64. Perhaps you can answer several questions that I have regarding the Commodore 128. When the computer is first turned on, typing the following:

PRINT FRE(0) returns the free bytes for Basic storage (58,109).
PRINT FRE(1) returns the free bytes for variable storage (64,256).

1. Can you think of the logical reason why Commodore assigned more free bytes to variable storage rather than to storage for the Basic program?
2. What are examples of variables that are stored in variable storage?
3. Is there a way to increase the number of free bytes for Basic storage at the expense of the free bytes for Basic variable storage?

I have taken the liberty of enclosing two short programs that I have written. The first program involves address modification. how can I change line #20 without typing GOSUB 220 and eliminate the SYNTAX ERROR?

The second program INPUTs numbers from the keyboard and sorts them before determining the highest number. How can I change the program to have the computer enter the RANDOM numbers into the SORT routine thus eliminating the need to enter the numbers from the keyboard?

Since I have spent considerable time trying to solve these problems, I would appreciate it if you could be of some assistance.

H.S. Rosenblatt, Las Vegas, Nevada

Address Modification

```
10 gosub 210
20 x=220: gosub x
50 end
210 print "a = 210": return
220 print "b = 220": return
.. Results: A=210, Syntax Error In 20
```

```
10 rem ** sort routine (4 numbers) **
20 rem this routine determines the highest of four
30 rem random numbers. the numbers are 2-14 and any
40 rem number less than 10 that is typed in must be
50 rem preceded by a zero.
80 for y=1 to 4
90 x=rnd(-ti)
100 n=int(rnd(1)*7)+int(rnd(1)*7)+2
110 print n
120 next
130 dim w$(4)
140 for x=1 to 4
150 input "n";w$(x)
160 if w$(x)=" " then x=4
170 next
180 s=0
190 for x=1 to 3
200 if w$(x)<=w$(x+1) then 230
210 a$=w$(x): w$(x)=w$(x+1): w$(x+1)=a$
220 s=1
230 next
240 if s=1 then 180
250 if w$(x)<>" " then print "the highest number is ";w$(4)
260 end
```

Reply From: Jim Butterfield, Toronto, Ontario

Dear Mr. Rosenblatt

Good questions. . .

1. I don't know Commodore's exact reasoning. But when faced with two banks of 64256 bytes each, and the need to set aside

www.Commodore.ca
May Not Reprint Without Permission

buffers and work areas, I'd agree with their choice of removing it from Bank 0 (the program area). Few people will need to write programs exceeding 50K in size; and even if they do, they can usually work around memory limitations by using chaining or overlay techniques. Thus, trimming the program area will seldom be limiting. On the other hand, many programs make use of huge tables of data: arrays of numbers or of strings (say, names and addresses). Many serious users use as much memory space as they can get, and would feel limited if "variable space" were curtailed.

2. *Variable storage contains: three types of variables (floating point, integer, and string), each of which takes up 7 bytes; three types of arrays (floating point arrays take up 5 bytes per item; integers, 2 bytes per item; string descriptors, 3 bytes per item; plus a little overhead to set up each array). Strings are stored in two parts: a "descriptor" which identifies the string, and the string itself, also in bank 1. The details of how each item is stored is a little complex and would take up too much space here; but you're free to PEEK in bank 1 (start at address 1024) to see what kind of things your program has created in memory.*

3. *If a program is too big to fit in bank zero, it's usually better to use chaining (DLOAD), overlay (BLOAD) or new-program (RUN) techniques to expand it rather than trying to take space from bank 1, which would be tricky.*

Microsoft Basic does not allow computed GOTO or GOSUB; a line number is not intended to be contained in a variable. The idea is for the program to be a "rigid skeleton" with no surprise switches in the execution sequence. You might be able to "gimmick" this effect with clever use of the TRAP/RESUME commands, but I recommend against it. Best to use programming constructs such as:

```
ON X GOTO 200,210,230 . . . or,
ON Y GOSUB 250,280,370
```

. . .either of which will allow you to go to a variable place without any program "surprises".

Place your X=RND(-TI) near the start of the program to be executed one time only (X=RND(0) is equally acceptable) . . . line 50, outside the loop, is preferable to the location you show. To generate strings containing random number values, delete lines 140 to 170 and insert:

```
115 W$(Y)=STR$(N)
```

Trust this will help "unblock" some of your problems.

Jim Butterfield

To the readers (and editors) of the Transactor Magazine:
In the case of the People vs. The Transactor Magazine, I have voluntarily placed upon myself the post of Defendant for the actions of the magazine and the people behind it. I must insist that The Transactor is not guilty to the charges of treason, unpatriotism, and criminal negligence in presenting the article entitled "Atari ST Notebook." I think that The Transactor had the right and showed good judgement in including an article on the Atari ST in the September issue of The Transactor.

In the first place, I am sure no one will refute the fact that The Transactor is one of the forerunners when it comes to presenting its readers with new products and developments. Were they not the first ones to publish a fix for the 1541 save with replace bug? Did they not introduce us to the Super Kit/1541 software (which I have bought and enjoyed) in the pages of this very magazine months before any other of the "leading" Commodore magazines like Compute!'s Gazette and RUN even had ads for it? I would think that the ST is such a new and impressive machine that even a strictly Commodore magazine shouldn't totally overlook it.

Secondly, there has been trouble with strictly Commodore magazines in that they tend to give the readers too narrow a viewpoint on the computer industry. The Transactor is better than most in this respect, so I think the ST article was right on target with the direction the magazine has chosen to take. Thirdly, both the Amiga and the ST represent great technological advances. I, for one, am mainly in the computer hobby because I am enchanted by technology and I suspect that it is at least a motivating factor for a lot of you. I would like to encourage The Transactor to present reviews of other new computers which come out in the future provided that 1) they represent new and exciting advances in technology (the advances being in graphics, speed, memory, power, or price to produce) and that 2) they not be some boring IBM clone. Both the Amiga and ST fulfill both of these qualifications admirably.

Lest anyone get me wrong, I am not an Atari fan. I would not buy an Atari XL system for half of the price of my Commodore 64 system (although maybe for a quarter or fifth). If I had the money for an Atari ST system, I would wait just a little longer till I have the few hundred dollars more I need to get my Amiga system. But the ST needs to be taken on its own, forgetting the company behind it and its past blunder computers.

Given the above evidence, I contend that The Transactor magazine must be held NOT GUILTY.

By the way, in response to a letter by Roy M. Randall which appeared in the November issue of The Transactor, Commodore isn't the only place to get custom chips (actually, I didn't even know you could order them directly from Commodore, but then again, Roy has apparently found out the hard way that you can't). Jameco Electronics has for a while been selling Commodore VIC 20 and Commodore 64 (and now C-128) chips. Prices are about $20.00 for the VIC II and SID chips, about $15.00 for the CIA, etc. If you want specs, you can get them for an extra $1.50. And I know that Jameco has them, because I had to order a CIA chip from them already. Jameco regularly has ads in BYTE magazine, or you can ask for a catalog at:

Jameco Electronics
1355 Shoreway Road
Belmont, CA 94002

David Godshall, Goshen, IN

I haven't seen this much heat over an issue since Bill 94 was driven through Ontario parliament recently. The ST, as David has pointed out, is a machine worthy of notice. It may not be everyone's cup of bits, but it is much more powerful and full featured than anything

www.Commodore.ca
May Not Reprint Without Permission

Commodore has ever built, aside from the Amiga. Although it is always nice to stay within a familiar shell, breaking out once in a while does provide a new perspective. Looking at, playing with, and understanding the ST can give you a totally new outlook on the computer age. This outlook may prompt you to abhor the Atari, for all of its faults, adore the Atari, for all of its good points, or remain neutral. Experience, no matter how distasteful, is always invaluable. Here's a kicker. The Atari ST 520/1040 uses the same disk format that an IBM Convertible does. The Atari can read from and write to an IBM PC 3.5" diskette, but the PC cannot read from or write to the Atari diskette. The Atari has a better controller in their drives. Don't you love trivia?

Pete Baczor To The Rescue: The following comes from Pete Baczor, Manager, Customer Support, Commodore West Chester, in response to a letter published concerning the lost order of 1 SID chip from Mr. Roy Randall (also mentioned above).

Dear Mr. Randall:

I read of your plight pertaining to ordering a 6581 SID chip in the most recent issue of The Transactor. Fortunately, yours is not the norm when ordering parts from our company.

Hopefully, by this time you have received the chip you ordered, but just in case you have not, I have enclosed a 6581 for you.

I apologize for any inconvenience that this has caused you. Thank you very much for your continued support of Commodore products.

Sincerely, Pete Baczor

Thank you Mr. Baczor, for attending to what could certainly turn into a distressing situation for even the most seasoned Commodore aficionado.

Sky Travel Lost and Found: Since reading your interesting review on Sky Travel, I have tried to obtain a copy of this program and would appreciate it if you could forward me the name of a supplier who I could contact to obtain a copy.

R.H. Yeates
43 Railway Street
Bluff Point, 6530
West Australia

No problem. From the advice of Commodore Canada comes a sure bet supplier:

Canadian Software Source
5318 Yonge Street
North York, Ontario
M6N 5P9 (416) 229-4513
Contact: James Milne

The package currently retails for $29.95 in Canadian funds.

Looking Back At The 1541 Head Cleaner: As the author, and frequent user of "The Improved 1541 Head-Cleaning Program", I was quite surprised to see in Volume 7, Issue 01 of The Transactor, the letter from Mr. Kerrigan who felt that the program had thrown his drive out of alignment. Although I had not experienced any problems with the program, I reviewed the code I had written in light of this information. I feel certain that if used under ordinary conditions, the program will not harm the drive. This lead me to investigate extraordinary conditions that might account for the reported misalignment. The conditions tested are listed below:

1. After removing the program disk, but before running the program, an attempt was made to load the directory or another program, resulting in a disk error.

2. After loading the program, the drive was turned off, then on again before the program was run.

Condition one above seemed to cause the head to bump once against the stop beyond track 1, but the drive worked fine when the program had ended. Condition two was another matter. Upon power up reset, the drive sets location $24 to 0. When run, the head cleaning program, believing the head is already out at its furthermost step, begins the task of moving it to track 35. Subsequent loads yielded only a flashing red light; however, each time simply sending an "Initialize" command to the drive freed the head, and the drive once again worked flawlessly. Those who occasionally reset their drives between operations may wish to add this line to the program:

```
165 if x=0 then end
```

As pointed out in the Editor's reply to Mr. Kerrigan's letter, never assume your drive is out of alignment until you are sure that the drive head is moving properly.

David Peterson Irvine, CA

Omni Reader Update: Quite a few readers have been kind enough to send us information about one company in California that is selling the Omni Reader. Apparently Byte magazine has been running their ads for some time, but we have been too blind to see the ads. Thanks to all of you, we now can find an Omni Reader. The address of the supplier is listed below:

California Digital
17700 Figueroa Street
Carson, CA 90248
Order: 800-421-5041
Tech/CA 213-217-0500

The advertised price is $179.00 in US funds.

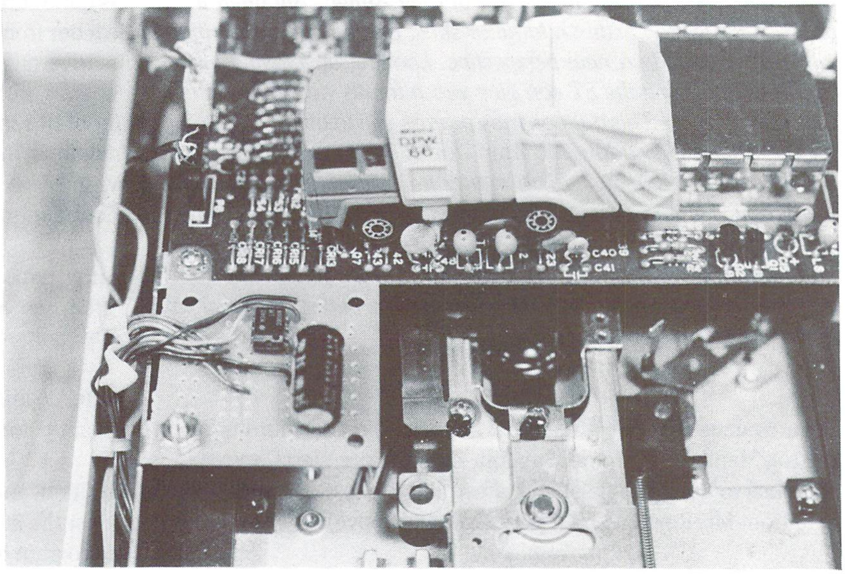
Moving With Caution: I've just finished reading the September 1986 Transactor. Congratulations on another excellent magazine.

I have some comments on two of the articles. "MOVE: A General Purpose Propagating Move Routine" by R.J. DeGraff outlines a very handy memory copying utility with the added benefit of a "fill" command using the "propagating" feature. Readers should be cautioned, however, about using this routine to copy portions of memory that overlap. If 200 bytes are copied to a location starting

www.Commodore.ca
May Not Reprint Without Permission

50 bytes higher in memory, for example, the utility will corrupt the data since it will copy over the original. For any overlapping memory copies, the MOVMEM routine described in the July 85 Transactor should be used. It avoids the problem by starting with the highest byte and working backwards.

"Commodore 128 High-Res Graphics" by Paul T. Durrant is a well written piece of code that does the job elegantly. Paul probably has an early C128 with a revision #7 8563 video chip. VDC register #25, which controls hires and text modes, also holds other information. Specifically, the first three bits hold horizontal smooth scroll data. Unfortunately, the newer revision #8 VDCs use different data in this register, and Paul's code as written will show a nasty sparkling line on the screen, spoiling the hires display on newer C128s. "Superbase 128" fell afoul of this trick, too.



"1541 with on-board garbage collection"

The solution is to add a few bytes of code to change only the text/hires bits and leave the others intact, regardless of what they contain. Attached is the necessary code. Note that it skips seven bytes at the end of Paul's code which he (and I) used for temporary storage.

Noel Nyman, Seattle, Washington

Two Changes In Original Code As Shown

```
00bc7 a2 19      ldx  #$19
00bc9 a9 80      lda  #$80
00bcb 20 ed 0b   jsr  $0bed  ;change in jsr address
00bce 60         rts

00bcf a2 19      ldx  #$19
00bd1 a9 40      lda  #$40
00bd3 20 ed 0b   jsr  $0bed  ;change in jsr address
00bd6 a5 d7      lda  $d7
00bd8 30 03      bmi  $0bdd
00bda 20 2c cd   jsr  $cd2c
00bdd 20 42 c1   jsr  $c142
00be0 20 2c cd   jsr  $cd2c
00be3 4c 0c ce   jmp  $ce0c
```

Addresses $0BE6–$0BEC are used for temporary variables. New code starts at $0BED.

```
00bed 8d ec 0b   sta  $0bec  ;store data temporarily
00bf0 20 da cd   jsr  $cdda  ;get current value in vdc reg 25
00bf3 29 3f      and  #$3f   ;strip off top two bits
00bf5 0d ec 0b   ora  $0bec  ;set top bits based on hires or
00bf8 20 cc cd   jsr  $cdcc  ;text, and store in register 25
00bfb 60         rts
```

P.S. Many months ago a friend and I were using up a roll of film after taking some pics of a 1541 add-on board. In the process, I came up with the enclosed.

Between your articles, letters, AND photos, you are helping make The T. a top-notch journal. Thanks for everything. We appreciate it.

North American Commodore For Use In Europe: I would highly appreciate an authoritative answer to my problems. I am considering the purchase of a Commodore 128 computer with the following peripherals: 1902 Monitor, 1571 Disk Drive, Datassette, Dot Matrix Printer, Joysticks, etc.

This set will be used in Europe with a power supply of 220 VAC/50 Hz. A suitable transformer will step-down the voltage but the frequency will remain unchanged. My question is: will this set work properly at 50 Hz?

I have visited numerous dealers in the New York area and the number of answers "yes", "no" and "I don't know" is roughly equal. A letter mailed a month ago to the manufacturer remained unanswered. If your answer will be "No", then please give me information of a dealer who would be able to handle the problem of delivering specified items either here or to my permanent address in Poland. Obviously, all these items may be easily purchased in Western Europe but with the current exchange rate for US dollars, prices there are double that of the US.

M.H. Trenkner, M.D., Visiting Research Professor
Chairman School, Gdansk, Poland

The system described will work just fine at 50 Hz. Once you have stepped the voltage level up properly, you can expect only a few problems. One problem will be that the occasional North American software package could rely on the IRQ taking place at 60 Hz. instead of 50 Hz. You will never have problems with software that you or your friends/business associates write, but a couple of the commercially available packages in North America could give you problems. One example of a headache in Europe is Prism Software's SuperKit/1541, which is so heavily dependent on a 60 Hz. IRQ that it becomes indignant when faced with anything else. Word is that they're working on a 50 Hz. version. Perhaps shopping for all of your software in Europe is the answer.

www.Commodore.ca
May Not Reprint Without Permission

TransBASIC Installment #12

Nick Sullivan
Scarborough, Ont.

TransBASIC Notes

TransBASIC has been a regular Transactor feature for 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.

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".*

Part 11 – *Describes one huge module called "GRAPHCMDS". It's used for plotting graph data, and printing it effectively. Also mentions that the next TransBASIC Column will be the last in the "series".*

TransBASIC Installment #12

In agreement with the rumour mentioned last issue, this TransBASIC column is the last of the series. This is not to say that The Transactor will not be publishing more modules in the future (in fact, I'd still like to have one or two appear in every issue), but it does mean that TransBASIC will get a lot less space (and require a lot less preparation time) than it has in the past. If you're new to TransBASIC, and want to know what modules have appeared in previous issues, think about ordering the TransBASIC disk (see News BRK or mail order card at center). There you'll find all the modules we've published to date, along with the TransBASIC kernel, the SYMASS assembler, and a number of support utilities that will get you going with the TransBASIC system in no time.

Besides the six modules that appear below, we still have several others on hand that will be published over the course of the next few months, and new submissions are still welcomed. If the backlog gets too big, we always have the option of putting the raw modules (unedited and unintegrated) onto a supplementary disk for people to use as they see fit.

Meanwhile, I'd like again to thank all those authors who have contributed to TransBASIC over the past two years for their time and effort. Programming by committee has a deservedly bad reputation, but in this case it seems to have worked out well.

Owing to a breakdown in the massive TransBASIC bureaucracy, the line assignments for the keywords and routine addresses in Paul Adams' GRAPHCMDS module, published last issue, were incorrect. The official line range for the keywords is 155 through 162; for the routine addresses it is 1155 through 1162.

This time around we have a collection of small modules that you can add to a TransBASIC dialect at very little cost in memory –– or keyboard fatigue. The authors are: Stewart Watton of Windsor, Ontario (STRING$, Program 1); Wayne Happ of North Babylon, New York (UNEW, Program 2; FREE, Program 3; and FACT,

www.Commodore.ca
May Not Reprint Without Permission

Program 4); Andrew Walduck of Barrie, Ontario (SPEEDUPS, Program 5); and Steve Hammer of Muscatine, Iowa (DATAFY, Program 6).

And in closing,

SYS 49155 :REM DISABLE TRANSBASIC

New Commands

STRING$((Type: Function Cat #: 199)
Line Range: 15156–15196
Module: STRING
Example: PRINT " ";STRING$(38,"*")
This function returns the first character of the string argument (the second argument) repeated the number of times specified in the numeric argument.

UNEW (Type: Statement Cat #: 200)
Line Range: 15198–15216
Module: UNEW
Example: UNEW
This statement restores the BASIC program that was in memory prior to an accidental NEW or software reset.

FREE (Type: Function Cat #: 201)
Line Range: 15218–15234
Module: FREE
Example: IF FREE < 256 THEN PRINT "NOT ENOUGH MEMORY"
This pseudo-variable does what the FRE(0) function should always have done, returning the number of bytes remaining in BASIC workspace as an unsigned quantity.

FACT((Type: Function Cat #: 202)
Line Range: 15236–15272
Module: FACT
Example: PRINT FACT(7)
This function returns the factorial of its argument. Arguments in the range 0 through 33 are accepted; smaller arguments generate an ILLEGAL QUANTITY error; larger arguments exceed the 64's floating point capacity and so generate an OVERFLOW error.

FAST (Type: Statement Cat #: 203)
Line Range: 15274–15288
Module: SPEEDUPS
Example: FAST
This statement speeds the CPU operation of a Commodore 64 by blanking the video screen, providing an advantage in processing speed of a bit more than 6 per cent. On a Commodore 128 in C–64 mode it also switches the CPU to 2MHz operation.

SLOW (Type: Statement Cat #: 204)
Line Range: 15290–15304
Module: SPEEDUPS
Example: SLOW
This statement restores the normal operating speed of a Commodore 64 (or Commodore 128 in C–64 mode) after it has been accelerated by the FAST command in this module.

DATAFY (Type: Statement Cat #: 205)
Line Range: 15306–15522
Module: DATAFY
Example: DATAFY 8,5000,10,8,"SPRITE.DAT"
This statement converts a disk file to DATA statements that are appended to the program currently in memory. If there is a load address in the file, that is converted too (and should generally be removed -- just take out the first two DATA items by hand). The parameters are, in order: the disk device number (8 to 11), the starting line number for the DATA (should be higher than the highest line number currently in the program), the line number increment (1 to 255), the number of DATA items per line (1 to 62), and the name of the file containing the bytes to be made into DATA statements.

Program 1: STRING

```
KI  0 rem string (stewart watton, jan/86) :
FH  1 :
EC  2 rem 0 statements, 1 function
HH  3 :
PH  4 rem keyword chars: 8
JH  5 :
NJ  6 rem keyword    routine   line    ser #
EL  7 rem f/string$(  string    15156  199
MH  8 :
OO  9 rem =================================
OH  10 :
BN  624 .asc "string$" : .byte $a8
OD  1624 .word string-1
DN  15156 string jsr  $b79e     ;get # of reps
KO  15158        txa
EL  15160        pha
LM  15162        jsr  $aefd     ;check comma
FH  15164        jsr  $ad9e     ;eval string expr
LC  15166        jsr  $b6a3     ;make descriptor
CO  15168        ldy  #0        ;get first char
EA  15170        lda  ($22),y
CK  15172        sta  t2
LF  15174        jsr  $aef7     ;check right paren
OG  15176        pla            ;# of reps
KN  15178        jsr  $b47d     ;reserve space
FK  15180        tay            ;make index
HD  15182        beq  str2      ;exit if no reps
GG  15184        lda  2         ;copy to str space
IC  15186 str1   dey
AG  15188        sta  ($62),y
NF  15190        cpy  #0        ;test finished
DF  15192        bne  str1      ; no
HD  15194 str2   jmp  $b4ca     ;return the string
CN  15196 ;
```

Program 2: UNEW

```
EK  0 rem unew (wayne happ)          :
FH  1 :
AI  2 rem 1 statement, 0 functions
HH  3 :
PG  4 rem keyword chars: 4
JH  5 :
NJ  6 rem keyword    routine   line    ser #
OL  7 rem s/unew     une       15198  200
MH  8 :
NL  9 rem ================================
OH  10 :
HB  163 .asc "uneW"
FK  1163 .word une-1
```

www.Commodore.ca
May Not Reprint Without Permission

```
KP  15198 une   lda  #1         ;write non-zero to
PC  15200       tay
EE  15202       sta  ($2b),y    ; first link-hi
LE  15204       jsr  $a533      ;re-chain program
KP  15206       lda  $22        ;set start-of-vars
BF  15208       ldy  $23
EG  15210       sta  $2d
JM  15212       sty  $2e
OJ  15214       jmp  $a660      ;perform clr
GO  15216 ;
```

Program 3: FREE

```
DG  0 rem free (wayne happ)              :
FH  1 :
EC  2 rem 0 statements, 1 function
HH  3 :
PG  4 rem keyword chars: 4
JH  5 :
NJ  6 rem keyword    routine    line    ser #
OC  7 rem f/free     fre        15218  201
MH  8 :
OO  9 rem ================================
OH  10 :
GJ  625 .asc "freE"
DP  1625 .word fre-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 ;
II  15218 fre   jsr  $b526      ;garbage collection
CL  15220       sec             ;subtract top of
OM  15222       lda  $33        ; arrays from
GG  15224       sbc  $31        ; bottom of strings
JE  15226       tay
KA  15228       lda  $34
GB  15230       sbc  $32
NI  15232       jmp  usfp       ;conv to float pt
IP  15234 ;
```

Program 4: FACT

```
LD  0 rem fact (wayne happ)              :
FH  1 :
EC  2 rem 0 statements, 1 function
HH  3 :
DH  4 rem keyword chars: 5
JH  5 :
NJ  6 rem keyword    routine    line    ser #
EG  7 rem f/fact     fact       15236  202
MH  8 :
NL  9 rem ==============================
OH  10 :
CM  626 .asc "fact" : .byte $a8
HF  1626 .word fact-1
BE  15236 fact  jsr  $aef4      ;eval argument
DP  15238       jsr  $b7a1      ;conv to int in .x
ED  15240       txa             ;test arg = 0
FC  15242       bne  fac1       ; no
GI  15244       lda  #1         ;0! = 1!
IH  15246 fac1  sta  t2
JG  15248       jsr  $bc3c      ;conv to float pt
DL  15250 fac2  jsr  $bbca      ;copy to $0057
KF  15252       dec  t2         ;decr index
DF  15254       lda  t2         ;index to .a
FB  15256       cmp  #2         ;test if done
IF  15258       bcc  fac3       ; yes
FH  15260       jsr  $bc3c      ;conv to float pt
CO  15262       lda  #$57       ;times accumulated
PE  15264       ldy  #$00       ; value at $0057
CO  15266       jsr  $ba28
EA  15268       jmp  fac2       ;loop
AC  15270 fac3  rts
OB  15272 ;
```

Program 5: SPEEDUPS

```
BC  0 rem speedups (a. walduck, june/86) :
FH  1 :
DH  2 rem 2 statements, 0 functions
HH  3 :
PH  4 rem keyword chars: 8
JH  5 :
NJ  6 rem keyword    routine    line    ser #
JK  7 rem s/fast     fas        15274  203
NN  8 rem s/slow     slo        15290  204
NH  9 :
ID  10 rem ================================
PH  11 :
BE  164 .asc "fasTsloW"
IB  1164 .word fas-1,slo-1
KN  15274 fas   lda  $d011      ;blank screen
EN  15276       and  #$ef
FI  15278       sta  $d011
BG  15280       lda  $d030      ;enable 2mhz mode
CJ  15282       ora  #1
CJ  15284       sta  $d030
CK  15286       rts
OC  15288 ;
OF  15290 slo   lda  $d011      ;show screen
BI  15292       ora  #$10
FJ  15294       sta  $d011
HH  15296       lda  $d030      ;disable 2mhz mode
KO  15298       and  #$ef
CK  15300       sta  $d030
CL  15302       rts
OD  15304 ;
```

Program 6: DATAFY

```
AE  0 rem datafy (steve hammer 3/86)     :
FH  1 :
AI  2 rem 1 statement, 0 functions
HH  3 :
GO  4 rem keyword characters: 6
JH  5 :
NJ  6 rem keyword    routine    line    ser #
PO  7 rem datafy     dafy       15306  205
MH  8 :
OO  9 rem ================================
```

www.Commodore.ca
May Not Reprint Without Permission

```
OH  10 :
AF  39 setlfs   =   $ffba
KD  40 setnam   =   $ffbd
IG  41 open     =   $ffc0
CL  42 chkin    =   $ffc6
IH  43 close    =   $ffc3
IB  44 clrchn   =   $ffcc
DB  45 getin    =   $ffe4
PP  165 .asc "datafY"
HJ  1165 .word dafy-1
IO  9150 errpgm  ldx  $3a
MJ  9152         inx
CM  9154         bne  epg1
AL  9156         rts
IL  9158 epg1    jmp  $af08
OD  9160 ;
LJ  15306 dafy     jsr   errpgm    ;check direct mode
EE  15308          jsr   gn1       ;get device number
AP  15310          sty   dvice
KK  15312          cpy   #8        ;test device >= 8
LE  15314          bcc   daf1      ; no
FL  15316          cpy   #$0c      ;test device < 12
HJ  15318          bcc   daf2      ; yes
OJ  15320 daf1     ldx   #9        ;'illegal dev #'
KN  15322          jmp   $a437
NM  15324 daf2     jsr   getnum    ;get start line #
MM  15326          sty   ln
AD  15328          sta   ln+1
GI  15330          jsr   getnum    ;get incr value
PN  15332          tya             ;test > 0
FK  15334          beq   daf3      ; no
EO  15336          sty   incr
PD  15338          jsr   getnum    ;get items/line
HO  15340          tya             ;test > 0
NK  15342          beq   daf3      ; no
MA  15344          cpy   #$3f      ;test < 63
LL  15346          bcc   daf4      ; yes
BA  15348 daf3     jmp   $b248     ;'illegal quantity'
OO  15350 daf4     sty   numit     ;save items/line
EE  15352          sty   itcnt
DG  15354          lda   #$79      ;open 121,dv,0
MN  15356          ldx   dvice
JE  15358          ldy   #0
KM  15360          jsr   setlfs
DJ  15362          jsr   $aefd     ;check comma
JM  15364          jsr   $ad9e     ;eval filename
NL  15366          jsr   $b6a3     ;set up for setnam
JM  15368          jsr   setnam
OH  15370          jsr   open
NE  15372          ldx   #$79      ;open channel
JA  15374          jsr   chkin
IB  15376          lda   $2d       ;back up start-of-
PP  15378          bne   daf5      ; variables pointer
OM  15380          dec   $2e
FG  15382 daf5     dec   $2d
ID  15384 daf6     lda   #1        ;set fwd-link hi
EO  15386          jsr   incsov
JB  15388          jsr   wrtlin    ;add line #, 'data'
HF  15390 daf7     jsr   maknum    ;add data item
KM  15392          lda   $90       ;test status
AD  15394          bne   daf9      ; end of file
BG  15396          dec   itcnt     ;count down items
GD  15398          beq   daf8      ; line complete
DH  15400          lda   #$2c      ;add comma
```

```
EP  15402          jsr   incsov
CC  15404          bne   daf7
JA  15406 daf8     lda   numit     ;reset counter
MB  15408          sta   itcnt
CB  15410          jsr   incs2     ;add two zeros
DA  15412          bne   daf6      ;branch always
CL  15414 daf9     jsr   incs2     ;add two zeros
GH  15416          jsr   incsov    ; and one more
LL  15418          jsr   clrchn    ;shut down disk
IJ  15420          lda   #$79
DL  15422          jsr   close
BO  15424          jsr   $a533     ;rechain
LK  15426          jmp   $a660     ;basic clr
KL  15428 ;
BG  15430 incs2    lda   #0        ;add two zeros
EA  15432          jsr   incsov    ;call then fall
AM  15434 ;
CN  15436 incsov   ldy   #0        ;index
PF  15438          sta   ($2d),y   ;add to program end
OC  15440          inc   $2d       ;bump sov pointer
AG  15442          bne   ics1
DD  15444          inc   $2e
KB  15446 ics1     rts
OM  15448 ;
AM  15450 wrtlin   lda   ln        ;add line #
GC  15452          jsr   incsov
AH  15454          lda   ln+1
KC  15456          jsr   incsov
MI  15458          lda   #$83      ;'data' token
OC  15460          jsr   incsov
BD  15462          lda   #$20      ;space
CD  15464          jsr   incsov
AO  15466          clc
LN  15468          lda   incr      ;add line inrement
AL  15470          adc   ln
OP  15472          sta   ln
OG  15474          bcc   wrl1
GK  15476          inc   ln+1
HG  15478 wrl1     rts
OO  15480 ;
HG  15482 maknum   jsr   getin     ;get disk byte
CB  15484          tay             ;conv to --
AP  15486          jsr   $b3a2     ; floating point
DE  15488          jsr   $bddd     ; asc str at $0100
HC  15490          ldx   #1        ;skip leading space
HD  15492 mkn1     lda   $0100,x
DJ  15494          beq   wrl1      ;end at first null
NK  15496          jsr   incsov    ;add char to prg
GG  15498          inx
PJ  15500          bne   mkn1
EA  15502 ;
OO  15504 getnum   jsr   $aefd     ;check comma
DH  15506 gn1      jsr   $ad8a     ;eval num expr
GA  15508          jmp   $b7f7     ;conv to integer
MA  15510 ;
KH  15512 numit    .byte 0
ME  15514 itcnt    .byte 0
JD  15516 dvice    .byte 0
MG  15518 incr     .byte 0
NB  15520 ln       .word 0
IB  15522 ;
```

www.Commodore.ca
May Not Reprint Without Permission

TeleColumn

First Transactor Online Conference!
Saturday, November 1, 1986
see below

Welcome to the newest regular feature of The Transactor! TeleColumn is where you'll find out about all our latest activity in the exploding world of online communications.

Those of you who are regulars on the CompuServe Information Network already know that The Transactor has been coordinating the Commodore Programming and Commodore Communications Forums on that service since September 1.

Although most of the activity we're directly involved in is on the CompuServe Information Network, we'll be including any pertinent news regarding the online industry. Multi-user systems is our main interest, but BBS systems and BBS networks are invited to participate by sending us material that would interest Transactor readers. Packet switching networks (ie. Tymnet, Telenet, and DataPac) are also an integral part of the online phenomena, and anyone with tips on using these services are encouraged to share them in TeleColumn.

Equipment capability is the single most important ingredient for effective tele-computing. TeleColumn will be the place to obtain the latest on great new communications hardware and software, and the not so great.

The CompuServe Information Network

Sept. 1, 1986: Transactor Online Finds New Home!

The following is a letter we received from CompuServe welcoming us to our new online headquarters:

> *Dear Mr. Hildon:*
>
> *On behalf of the subscribers, sysops and staff, welcome to the CompuServe Information Service!*
>
> *As I indicated to you in our earlier conversations, we're really pleased to have you and The Transactor aboard as administrators of The Commodore Programming Forum (CMBPRG) and The Commodore Communications Forum (CMBCOM), and we look forward to a long, harmonious and productive working relationship. We're sure that the combination of our service with your acknowledged expertise with the Commodore line of computers will make these forums a hot item with users everywhere.*
>
> *Once again, any time you need help with anything relating to your online activities, please feel free to call, or drop me a line on the system.*
>
> *Sincerely,*
>
> *Jim Rulfs*
> *Manager, Online Computing Services*
> *CompuServe Incorporated*

Thank you Mr. Rulfs. I hope that with a little patience, practice, and perseverance we'll be able to make our online efforts as productive as our offline routine, and one day, maybe vice-versa!

And We're Off!. . . er, On!

The word "information" hardly describes the seemingly endless activities that you can access on the CompuServe Network. As mentioned, Transactor Publishing Inc. will be managing the activities of two sections of The Commodore Network on CompuServe. CBMNET is only one service CompuServe. There are Networks for Atari, Apple, IBM, and everything else from Golf to Rock Music.

The two sections we'll be managing are the Commodore Programming Forum, and the Commodore Communications Forum. Both forums are functionally the same, but are different in content. Each forum has literally hundreds of programs available for downloading at no extra charge other than your connect time charges. CBMPRG has programs aimed at those writing software such as assembler subroutines, programming utilities, and machine language monitors. CBMCOM has programs aimed at the intermediate level programmer, and also contains several terminal programs for just about any modem available.

Both forums have their own Message Boards too. Much like the Data Libraries, the messages contain information that relates to the content of the forum. They're also full of questions and answers for everything from the most common of problems to the obscure.

There are three other forums you should also know about: CBMART is the Commodore Arts/Games forum. This is where you'll find just about any public domain game, along with

www.Commodore.ca
May Not Reprint Without Permission

Doodle and Flexidraw files, CAD programs, music software, plus anything dealing with graphic design.

The Amiga Forum is, naturally, for those of you with Commodore's latest equipment line. And The Commodore Service Forum is run by the Telecommunications Department of Commodore HQ in West Chester. We'll have more details about these forums in future issues. Stay tuned!

Coming up in the very near future (before next issue, barring catastrophes) will be the Transactor Display Area, where you'll be able to get in touch with us directly on magazine-related matters. This area will have lots of uses, including some we haven't yet thought of no doubt, but the following will give you some idea of what to expect:

1. Reading Articles: You'll get an opportunity to catch up on past issues you may have missed by reading articles on-line. Of course, we're hoping too that the availability of Transactor articles in this area will help bring new readers to the printed edition, just as we're hoping that many of you reading this will take the time to look us up on CompuServe.
2. Magazine Mail: Want to write a letter to the editor but you've never got around to putting it on paper? Got a complaint? A comment? A compliment? A subscription or delivery problem? Now you'll be able to get in touch with the Transactor staff more easily than ever before, and get answers faster too.
3. Subscriptions and Mail Order: Do you just hate filling out those little cards in the centre of the magazine? We'll have online ordering, which a lot of people find more convenient than mail order, and we'll be able to keep you up-to-date on new products, prices, and so on.

By the way, Transactor programs will be available in the CBMPRG forum (free, except for connect time charges), and not in the Display Area. As for articles that contain lots of embedded code. . . we'll judge each case on its merits.

The SYSOPS (SYStem OPeratorS)

Keeping our forums running smoothly takes a lot of hard work, and a lot of learning for us. Luckily we have the aid of several very able assistant SYSOPS; in these early days, we depend on them especially heavily for their expert guidance and unfailing energy. Here is a complete list of the sysops on CBMPRG and CBMCOM, along with our User IDs so that you can find us easily on the system. Don't hesitate to get in touch with us if you have any technical questions, or if you have problems using the service.

Karl Hildon	76703,4242
Richard Evers	76703,4243
Chris Zamara	76703,4245
Nick Sullivan	76703,4353
Brian Niessen	76703,4034
Gary Farmaner	76703,3050
Jim Oldfield	76703,4033

You'd also like to meet our neighbours on the CBMART forum. Their names and IDs are:

Betty Knight	76703,4037
Wayne Schmidt	76703,4032
Jake Lund	76703,3051
Steve Sileo	76703,4244

The Amiga Forum also has its own set of SYSOPS:

Steve Ahlstrom	76703,2006
Jim Nangano	76703,4254
Don Curtis	76703,4321

November 1: The First Transactor Online Conference

On Saturday, November 1 at 10:00 PM., Karl Hildon, Richard Evers, Chris Zamara, and Nick Sullivan will all be participating in the first official Transactor online conference! That's right! All four of us will be will be on stage for any inquiry you care to throw at us. Just sign on and GO CBMCOM or CBMPRG, and enter "CO" at the main function prompt. It's possible we may be using facilities other than the regular conferencing area, but these details will be displayed when you arrive. See you there!

Getting Started

If you're a Transactor Subscriber, you may have noticed the CompuServe Intro-Pak bound into this issue. It contains a CompuServe User ID, a Password, plus $15.00 of connect time. It also contains complete instructions for signing on. If you don't have a modem, please don't throw it away - instead, you could give it to someone who does enjoy telecomputing, but we really suggest that you buy yourself a modem and join in! The telecommunications industry is literally exploding. CompuServe has over 250,000 subscribers, with the ratio of those using Commodore equipment at over 1 in 3!

For those of you who are just getting started on CompuServe, here are a few tips to make things a little smoother at the outset:

1. When you sign on, the system normally asks you first for your User ID (formerly known by the now obsolete term PPN, or Programmer Project Number), then for your password. To save time, you can enter both of these on the same line by putting a backslash ("\") after your User ID, then continue straight on with your password. Nothing you type after the backslash will appear on your screen. By the way, on the Commodore 64, the equivalent of a backslash is the British pound sign, just to the right of the minus key (for those using CompuServe's own VIDTEXtm terminal software, use Control £).
2. To get to the CBMNET area, type GO CBMNET at the main system prompt ('G' works just as well as 'GO'). The next thing you should see is a menu that will give you access to the five CBMNET forums: AMIGAFORUM, CBMART, CBM2000, CBMPRG and CBMCOM. You can get to any of these directly (without using GO CBMNET) by typing GO plus the name of the forum you want to visit (e.g. GO CBMPRG). By the way, CBMART is the Commodore Art and Graphics Forum, managed by our good friend Betty Knight of Bellevue, Washington, and CBM2000 is the Commodore Service Forum, which

www.Commodore.ca
May Not Reprint Without Permission

is managed by Commodore itself. Take the time to visit them, too, while you're online. And, of course, those of you with Amigas won't want to miss the AMIGAFORUM, which is currently one of the most active on the system.

3. Once you're in the forum of your choice, the first thing you'll probably want to do is check out the messages. This is probably the easiest way to get a feeling for how and why people use CompuServe. There are usually more than 500 messages on a board at any given moment, so you may want to experiment with message reading, rather than try to read all the messages on your first visit. Try RF (Read Forward), RR (Read Reverse) and the others (Entering 'IN' at the Function prompt will give you complete instructions for using all the forum features), however, the most popular method for reading is RTN which stands for "READ THREAD NEW". A thread is a sequence of related messages, so this command lets you read all the messages relating to a particular subject as though they were numbered sequentially in the message base, which they almost certainly won't be. When you've exhausted one thread, the system will take you back to the point you started from, and pick up the next thread thereafter.

One caution -- if you quit reading in the middle of a thread, your "current message pointer", which is saved for you when you leave a forum, will be pointing to the current message number, not to the start of your thread, and there could well be unread messages (from other threads) intervening. This means that if you go back to the forum later on and type RTN again, you'll miss those unread messages. Should you wish to stop reading the messages deep within some interminable thread, you can issue a 'T' at the prompt between messages which will take you back to the main menu. Jot down the message number that you were "reading replies to", and at the main function prompt, type 'HI' followed by this number. This sets the "HIghest message read" so the next message you read will be this message number +1. Now you can start another RTN.

Next Issue

TeleColumn will be a regular feature from now on, and we hope it will be useful as a kind of liaison between the hardcopy and the electronic activities of the magazine. In TeleColumn #2 we bring you up to date on our first two months online, and we might also tell you about something called Color Mail -- an animated greeting card service run by Hallmark Cards.

We'll also tell you more about iNet, the Intelligent Network. This is a service of Telecomm Canada that's also available in the U.S. It has several features of its own, but the most valuable is the 1-800 numbers available for users in remote areas. Access via these lines costs no more than your regular monthly fee of $3.00 per month plus iNet online charges (which halt once you go through their "gateway" to another service, like CompuServe)

Signing on through DataPac may create problems for those downloading programs. Next issue we'll have more details about DataPac commands necessary for avoiding difficulties.

Until next issue, the next article details the aspects of downloading from CompuServe using Xmodem and 'B' protocols. Downloading with Xmodem protocol is a Catch-22 situation if you don't have terminal software that supports Xmodem protocol. The short BASIC program is a "get-by". It will allow you to download a somewhat superior program using the Xmodem protocol. Once you have the better terminal program, you won't need the program listed next, but you will need it to get by the Catch 22.

See you all next issue, and hopefully on CompuServe before then! Once signed on, type GO CBMPRG or CBMCOM and 'L'eave us a message!

Downloading From CompuServe

by Christopher Dunn, Chicago, Illinois

How to get something for (almost) Nothing.

So, you just logged on to Compuserve, and spent an hour or two looking around at all the goodies. There is the CB Simulator, the games, the financial reports, the user forums, and all the rest, but did you discover all the available free software you can download and run on your C64 or 128? It's ALMOST free, you still pay for your connect time while downloading, but there are hundreds of well written and useful programs available, from pictures and games to full blown BBS systems. This article will help you get started downloading from Compuserve.

I am going to assume that you have familiarized yourself a little with the way Compuserve works, and that you can find your way to the Commodore Fourms. The Fourms (sometimes called a 'SIG' for Special Interest Groups) are akin to local BBS systems you might have in your area. You can leave messages, read bulletins, and up and download files. Each Forum has a group of DATA LIBRARIES (known as a DL) that contains the files. There are sometimes up to 10 DLs with the files they contain in groups. One DL might be games, another might be music programs, etc.

Compuserve supports 4 protocols for transfering files. They are DC2/DC4, "A", "B", and XMODEM. A protocol is simply a standard that both ends of a line agree on and the format in which the data is sent and checked. Of the 4 protocols, DC2/DC4 is only useful for text files, and is basically a RAM buffer capture. "A" protocol is used on some older non-Commodore computers. "B" Protocol is used in Compuserve's Vidtex terminal program and provides for just about automatic transfer of files. XMODEM is also used in most popular public domain terminal program for Commodore equipment. You may notice that Punter protocol is not supported, simply put Punter is a Commodore only protocol, and Compuserve must cater to a wide market of all computer types. XMODEM is much easier to implement, is supported by a wide variety of computers, and is just as fast in transfering a file, if not faster when written in machine language. As a matter of fact, I have included a small

www.Commodore.ca
May Not Reprint Without Permission

XMODEM Bootstrap Downloader terminal program that you can use to download a fullblown XMODEM terminal program from Compuserve.

I will cover the steps required to download with XMODEM protocol from Compuserve here. If you already have a copy of Compuserve's Vidtex, then you are using "B" Protocol, and just about everything is automatic and explained in your vidtex manual.

Once you are in a forum, you can access the Data Libraries by entering: DLn Where n is the number of the Data Library you want to see the files of. This places you into that Data Library and you can now start looking through the files. The display shows the name of the file as it is called on Compuserve, and a description. If you were BROwsing through the DL, you will be prompted to either Read, DOWnload, or continue browsing through the files. At the prompt after each file you can enter: DOW /proto:xmodem DOW for download, and / proto:xmodem tells Compuserve to use XMODEM protocol right off, otherwise you would have been prompted for 1 of the 4 protocols to use. If you know the name of the file you want to download, you can also say so directly from the main data library prompt, simply by:

```
DOW <filename> /proto:xmodem
```

When you request a download in XMODEM, Compuserve will respond:

```
Starting XMODEM Transfer

Please initiate XMODEM transfer
and press <CR> when the transfer
is complete.
```

At this point you do what is required to place your terminal into receiving mode. The file should then start downloading to your disk. When you get an indication that the transfer is finished, you return to terminal mode and hit your <RETURN> key to indicate to Compuserve that the download is ended. You should now have a runable copy of the program on disk. You can download something else, or log off and run your new program.

There are many places to find programs and text files for your computer on line, of course there are the Commodore Forums, but other places as well contain items of interest. All files fall into 2 catagories, TEXT and PROGRAMS. Text are just that, files that contain written information, possibly the documentation for a program, or maybe a cooking recipe. Programs are runable code, such as Basic or Machine Language routines. To help tell Text and Program files apart, a standard was formed in the naming of the files. On Compuserve file names can be 6 characters long, then a period, then 3 more characters. These last 3 characters are called the file name extension. A typical file name might be: CBTERM.TXT The extension indicates this is a TEXT file. 2 special extensions are set aside for programs, and these are BIN and IMG. BIN stands for Binary, and is what is used when you work with XMODEM. IMG stands for Image, and is produced with "B" protocol in Vidtex. Any other extensions are generally text files. TXT, DOC, and MEM could indicate text, documemtation, or memo files. Some files may not even have extensions, but the file description should make clear what the file is. ARC is an extension that means archive, and requires a special program to unpack the file once it is downloaded. ARC is a way of compressing a group of files together into one to save on uploading and downloading time.

As I stated, file names ending is BIN or IMG are programs, you can directly download any BIN file with a XMODEM terminal program, and it should produce a runable program on your disk. IMG files on the other hand were created with "B" protocol, and the file contains a few extra bytes before the start of the program itself. If you download an IMG file with a generic XMODEM terminal program, the downloaded file will not run untill the extra bytes are stripped from the front of the file. There are utilities available for doing this, but by far the easiest thing to do is use a XMODEM terminal program that has the IMG byte stripper built in. The popular terminal program CBterm/C64 is one of these, and directly downloads both BIN and IMG files.

Now to the problem some of you might have, and that is how can you download anything if you don't have a terminal program that supports B or XMODEM protocol. Well you will find a possible solution in the program below. It is a tiny terminal/XMODEM downloading program that I call the Bootstrap XMODEM Downloader (BXD for short). It provides the barest of terminal functions and XMODEM error checking, but will download. You should only really use it to download a full terminal program like CBterm/C64.

BXD should work on both the 64 and 128:

```
CI  5 open5,2,0,chr$(6):dim i%(132)
FH  10 printchr$(14)" Sqqq  Bootstrap XMODEM
       Downloader  Ver 1.0"
LI  20 print"  Written by Christopher Dunn
EN  30 print" qqq Use the <F1> key to start the
       Download"
AH  100 print" [Terminal Mode]
NM  110 get#5,a$:if st=8 goto170
KE  120 a=asc(a$+chr$(0))and127
JK  130 if a=8 then a=157:goto160
JC  140 if a>=65 and a<=90 then a=a+32:goto160
LP  150 if a>=97 and a<=122 then a=a-32
LA  160 print chr$(a);
FI  170 get a$:if a$=" " goto110
JG  180 a=asc(a$+chr$(0)):if a=20 then a=8
        :goto220
CM  190 if a=133 goto 1000:rem do xmodem
GF  200 if a>=193 and a<=218 then a=a-128
        :goto220
II  210 if a>=65 and a<=90 then a=a+32
EB  220 print#5,chr$(a);
PH  230 goto 110
IP  1000 rem xmodem download
```

www.Commodore.ca
May Not Reprint Without Permission

```
CH   1010 ack$=chr$(6):nak$=chr$(21)
          :eot$=chr$(4):b=1
HN   1020 print" S Xmodem Downloader.
MB   1030 print" Enter file name for your disk: ";
          :f$=" ":input f$:if f$=" " goto 100
NG   1040 print" Working! Please standby"
KH   1050 open8,8,8,f$+",p,w"
KH   1060 forx=1to25:get#5,n$:next:q=0:print#5,nak$
AO   1070 get#5,c$:if st=8 goto 1170
MF   1080 q=q+1:i%(q)=asc(c$+chr$(0)):print".";
CC   1090 if q=1 and c$=eot$ then close8
          :print" DONE!":print#5,ack$:goto100
MC   1100 z=0:if q<132 goto1070
EE   1110 print:print" Checking Block"
EB   1120 ck%=0:forx=1to131:ck%=(ck%+i%(x))
          and255:next
NN   1130 if ck%<>i%(132) then print" Bad
          Checksum!":goto1060
HC   1140 forx=4to131:print#8,chr$(i%(x));:next
          :get#5,n$
FD   1150 print" Block "b" OK.":b=b+1
          :print#5,ack$:q=0
EI   1160 goto1070
MN   1170 rem check for time out
PP   1180 z=z+1:if z<500 goto 1070
HH   1190 print" Block time out! Retrying...."
          :z=0:goto1060
```

BXD has 2 main areas, lines 5 – 230 are the terminal routines, and most of that is to convert the Commodore's PETSCII character set to standard ASCII and back again. Lines 1000 – 1190 are the XMODEM download routines. The only shortfall to BXD comes when it has to deal with dialing your modem. There are so many different kinds that there is no simple way to write a dialing routine for all of them. If you have a manual connect (1600) or a HAYES compatible (1670, etc.) just log on using your manual mode or ATDT commands as normal. If you have other types, see if you can dial in on your phone and trick the modem into going on line. Lines 40–90 were left blank so you could write dialing routines for your modem into BXD if required. On the other hand, if you have a BASIC terminal program for your modem already, you could add lines 1000 – 1190 to it so you could call the XMODEM routine.

You should use BXD first off to download a fast, full featured terminal program. I recommend CBterm/C64. CBterm supports XMODEM, 40 or 80 column display screen, dialing routines for just about all modems, full disk and printer support, 22.5K RAM buffer, screen clock, direct display of high resolution RLE graphics and weather maps, and alot more. With optional overlays CBterm will also do New Punter protocol or emulate a Vidtex terminal. CBterm can be found in Data Library 2 (DL2) of the CB Interest Group Forum. You get to CBIG by entering: GO CBIG. Then enter the library with the command: DL2. The filename on Compuserve is CBT45.BIN, so you would type:

```
DOW CBT45.BIN /proto:xmodem
```

and Compuserve would respond with the "Starting XMODEM Transfer. . . . ." message. At this point you would press the <F1> key to put BXD into download mode, and would be prompted for a disk file name. Enter:

CBTERM

BXD will now download the program. As BXD progresses, you will see periods print across the screen, each one is a received character. Xmodem downloads in blocks of 128, so after each 128 characters you will see BXD print it is "Checking Block". If the checksum matches, BXD will print "Block # OK" and write the data to disk. If there was line noise or the data was bad, BXD will print "Bad Checksum!" and have Compuserve resend the data. If a character was lost in transmission, you will see the message "Block time out. . ." and the block will be resent.

If you continue to receive error messages after 4 or 5 attemps by BXD to get a block, then hang up, validate your disk to close the open file, and try from the beginning.

Unless you have a very noisiy telephone line, BXD should work well. CBterm Version 4.5 is 49 DISK BLOCKS long, which will be about 100 XMODEM blocks. At 300 baud it should take about 15 or 20 minits to download. Two other important files for CBterm are CBTP1.DOC and CBTP2.DOC, these are the instructions for using CBterm's many features. You can read these files online or capture them with CBterm's RAM buffer or another terminal program. All CBterm functions are activated by holding the Commodore key and a letter or digit. Once you have a copy of CBterm, you just:

```
load"cbterm",8
```

. . .and RUN. You are prompted for the baud rate, enter 3 for 300 or 12 for 1200. You will then see the opening screen and you can press C= and H for the HELP screen. It will display most of the features and what keys to press.

That is XMODEM in a nutshell. Once you have a copy of CBterm/C64 you can download just about any file on Compuserve, and this includes the IMG files. If you inspect the Data Libraries of CBIG you will find many programs and files for the C64. While not strictly a Commodore forum, CBIG has many Commodore followers. In its DL3 you can find many High Resolution RLE (Run Length Encoded) pictures that CBterm will directly display to screen and printer. These images range from the abstract to the standard computer room nudes. You can also find programs to convert your images to RLE format so you can upload your artwork. Other CBIG DLs contain programs and data like the CB Personal Ads or indexs of files for other computers. Give CBIG a look around while you are there.

If you have any questions or comments about XMODEM, BXD, CBterm/C64 or anything else I might be able to help with, leave a message in CBIG to SYSOP. I will be glad to help. Enjoy Downloading!

www.Commodore.ca
May Not Reprint Without Permission

Build a Modem Emulator

Bob Jonkman
Hamilton, Ontario

. . .The idea was to place two C–64s side by side, with one running a BBS program and the other running a terminal program. . .

Last year at the World of Commodore II show I came across a booth selling connectors for the C–64 user port. These things are as scarce as hen's teeth, and I figured I would buy two, even though I had no immediate application for them. It was a good thing I did, because I haven't found any other source for them, and they came in handy for a BBS demonstration.

The idea was to place two C–64s side by side, with one running a BBS program and the other running a terminal program, without using a modem or phone lines. This way everyone could see how a BBS is run as someone was actually using it.

The most important piece of hardware required is a cable to connect the two RS–232 lines (Transmit to Receive, and vice versa) in the user ports. This allows the two computers to communicate. Two other items are necessary: Something to alert the C–64 running the BBS that the other C–64 was present (the Ring Detect); and something to simulate the carrier signal normally provided by the modem. Without the simulated carrier the BBS would assume that the terminal program had broken the connection, so it would "hang up the phone" and log off. The Ring Detect is faked with two momentary switches connecting the RI lines of the RS–232 ports (one on each machine) to ground; similarly the Carrier Detect is faked by connecting the DCD and CTS lines to ground.

The connections in the user port we are concerned with are:

Pin #	RS–232	Description
A	GND	Protective Ground
B	S_{IN}	Received Data
C	S_{IN}	Received Data
F	RI	Ring Indicator
H	DCD	Received Line Signal (Carrier)
K	CTS	Clear To Send (Carrier)
M	S_{OUT}	Transmitted Data
N	GND	Signal Ground

The complete table can be found on page 143 of the User's Guide (with 6526 ID abbreviations), or page 355 of the Programmer's Reference Guide.

Equipment and Supplies:

5 conductor cable (approx. 2 metres)
2 normally open single pole momentary switches
1 single pole single throw toggle switch
1 medium sized hobbyists box
2 female edge connectors (2 x 12 pin, 5/32" spacing)

Some skill in soldering would be helpful, although this is an excellent project to learn on. You'll also need to drill holes in the hobby box for the switches.

Hook–up

The first thing to do is to put some holes in the hobby box. Drill a small hole in each of the ends of the box (the smallest sides). This will be where the cable goes through. While you're at it, you can also drill the holes for the switches. For a neat looking layout, divide the top of the box into thirds both horizontally and vertically using a pencil (that should look like a tic–tac–toe grid). Drill the holes for the Ring Detect switches at the intersections along the upper line, and drill the hole for the Carrier Detect switch in the centre of the lower line. You might as well mount the switches in the box now. That will make it easier to solder the cable to them later.

Thread the cable through the two holes on the side of the box. It is a good idea to tie two knots in the section of the cable inside the box so that it cannot be pulled out accidentally. Make sure you leave enough slack inside the box so that when you cut the wires they will be able to reach the contacts of the switches.

Remove about 2 inches of the sheath on the ends of the cable, and carefully strip away the sheath between the two knots. At this point I usually assign an order to the wires in the cable according to the resistor codes:

1	Black	6	Green
2	Brown	7	Blue
3	Red	8	Purple
4	Orange	9	Grey
5	Yellow	10	White

This will be the order in which I connect the wires (if all the colours are not in your cable, just use the ones that are in this order).

First, the ground wire. Although two different grounds are indicated in the chart above, for our purposes they are identical and we can connect them together. Connect the first wire to pins A and N of both connectors. You may have to use an extra piece of wire as a jumper to connect A to N on the connectors. Inside the box, connect this wire to one side of all three switches. Again, a bit of extra wire is useful here. Make sure that the wire is still connected all the way through, that is, it should come in one side of the box, connect to each switch, and continue out the box to the other connector.

Second, connect the Ring Indicator. Connect the second wire to pin F on both connectors. Inside the box, cut this wire in two. Connect one end to the remaining terminal of the closest momentary switch, and connect the other end to the other momentary switch. Now, when a switch is pressed it sends a "Ring Detect" signal to one of the computers.

Next, the Carrier Detect. Connect the third wire to pins H and K at

www.Commodore.ca
May Not Reprint Without Permission

each connector. Use some extra wire as a jumper to make it easier. Inside the box, strip some insulation from the middle of the wire, and connect it to the remaining terminal of the toggle switch. When this switch is turned on, it will send a "Carrier Detect" signal to both computers at the same time (with modems, if one detects a carrier it immediately sends a carrier of its own, so that both modems detect carriers).

Now we connect the Transmit line of one connector to the Receive lines of the other. Connect the fourth wire to pin M on one connector. On the other connector fasten this wire to pins B and C. There are no connections inside the box. Connect the fifth wire to pins B and C of the first connector, and to pin M of the second. This sounds awfully confusing, so check Fig. 1 to make sure you've got it right.

There! Everything should now be hooked up and ready for its first trial run. Go over every connection you've made to make sure the wires are connected to the right terminals, and make sure there are no solder bridges (great blobs of solder that connect two or more terminals that shouldn't be connected. Solder bridges are never made by technicians –– they generate spontaneously when everyone has their backs turned. . .). Even if you only have one computer you can still test it out. You'll need a terminal program like TERM24K that has a Ring Indicator in the status line. Plug one of the edge connectors into the user port of our C–64, and then switch it on. Load your terminal program, and watch the status line. Press the Ring Detect switch for that connector. On TERM24K you will see an R appear in the status line. Flip the Carrier Detect switch. You will see a C in the status line. Turn off the power to your computer before you check out the other connector. Of course, if you see smoke coming out of your computer, throw up your hands in despair, wildly run around in circles, and take your computer to Dr. Eric to find out what got fried. If you've followed these instructions, you shouldn't have any problems.

. . .And I Did It My–y–y–y Way. . .

Far be it for me to follow my own instructions. When I bought the hobby box and the switches, I was mostly concerned with appearances. Since my box was black, I bought matching black momentary switches because they looked so much better than red momentary switches. It wasn't until I got home that I found out that black switches are normally closed, and red switches are normally open.

Being too cheap to buy new switches, I found another solution. As long as the RI line is held at about 5 volts, it is off. When it is held at ground potential (0 volts), it is active (sends a Ring Detect signal). What I did was to connect a sixth wire to pin 2 on each connector (a source of 5 volts), and attached that to one terminal on the momentary switch. I connected the other side of the switch to the RI line (the second wire). In addition, I also connected a 1000 Ohm resistor to this wire, and connected the other end of the resistor to ground (See Figure 2). Now, as long as the switch was closed, the 5 volts would go straight to RI line, keeping it off. It would also go through the resistor to ground (without the resistor there would have been a short circuit). When the switch was open (pressed), the RI line would be connected through the resistor to ground (0 volts), making it active. This was just what I was after!

If anyone builds a modem emulator, I'd be interested in hearing from you. You can contact me through the T36 bulletin board in Toronto (416 385–8772, user 29).

MODEM Emulator

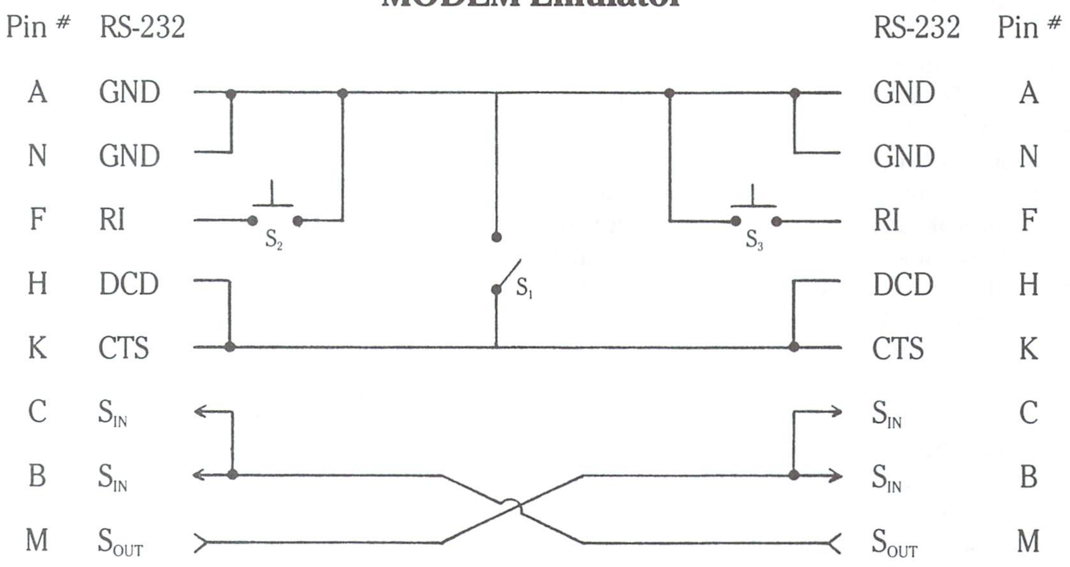


Figure 1

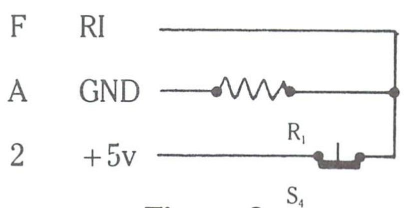


Figure 2

Table 1

S_1	SPST Switch
S_2, S_3	N.O. SP Switch
S_4	N.C. SP Switch
R_1	1 K ohm Resistor

www.Commodore.ca
May Not Reprint Without Permission

Universal RS–232 Cable

Martin Goebel
St. John's, Nfld.

A Simple Do–It–Yourself Project

There is more to connecting a pair of devices on a RS–232 (serial) port than simply plugging them in. This universal cable which is also known as a breakout box, can be used to overcome many problems which are due to different pin designations.

Background Information

The RS–232 standard defines the electrical characteristics for an interface for connecting a piece of data terminal equipment (DTE) and a piece of data communications equipment (DCE) such as a modem. This standard is not as far reaching as might be inferred by the common sales pitch, "Includes a Standard RS–232 Port". In fact, many pieces of equipment with a RS–232 port use the "standard" in different ways. Thus two pieces of equipment, even if they can be plugged together, will not necessarily work as intended.

Consider for instance two microcomputers interfaced with RS–232. Which one is the DTE and which one is the DCE? Another example is the interfacing of certain devices such as printers and plotters. Generally these devices only receive data, but on occasion they also send information back, error messages being an example. To make matters even more complicated, communications between microcomputers is always handled using software. The design of such programs may require that certain electrical connections be present but there is no set standard practice for how the RS–232 is to be used.

Fortunately, the RS–232 standard has sufficient common ground that it is possible to interface most equipment. The trick is to modify the interfacing cable so that the transfer of data occurs on the correct lines as required by the equipment or software. This simple project aids this task by allowing lines to be exchanged using jumper cables. Furthermore, by making this universal RS–232 cable you need never buy another cable no matter what equipment is to be interfaced and it may be cheaper than buying a ready made cable.

The "Standard" RS–232 Interface

The RS–232 uses a conventional 25 pin connector called a DB–25. There are 13 pins in the top row and 12 pins in the bottom row. The male and female connectors are mirror images of each other, thus pin 1 in the male connector can only meet socket 1 in the female connector. The 25 pins are generally assigned to signals according to Table 1. Note that signals on pins 2, 4, 14, 19, 20 and 24 originate with the DTE and that signals on pins 3, 5, 6, 8, 12, 13, 15, 16, 17, 21 and 22 are from the DCE. Pins 1 and 7 are shared and pin 23 is indeterminate. The reserved and unassigned pins may be used for anything.

Table 1: Common RS–232 Pin Designations

Pin	RS–232 Signals	Initials
1	Protective Ground	
2	Transmitted Data	(TXD)
3	Received Data	(RXD)
4	Request to Send	(RTS)
5	Clear to Send	(CTS)
6	Data Set Ready	(DSR)
7	Logic Ground	
8	Carrier Detect	(DCD)
9	. . .reserved. . .	
10	. . .reserved. . .	
11	. . .unassigned. . .	
12	Sec. Carrier Detect	
13	Sec. Clear to Send	
14	Sec. Transmitted Data	
15	Transmit Clock	
16	Sec. Received Data	
17	Receiver Clock	
18	. . .unassigned. . .	
19	Sec. Request to Send	
20	Data Terminal Ready	(DTR)
21	Signal Quality Detect	
22	Ring Detect	
23	Data Rate Select	
24	Transmit Clock	
25	. . .unassigned. . .	

www.Commodore.ca
May Not Reprint Without Permission

RS-232 Usage With Commodore

The usage of the electrical connections varies somewhat and is different among the various Commodore computers. My SuperPET uses only pins 1 to 8 and pin 20. These pins are assigned the functions as in the above table. In addition pin 13 is connected to a +5 VDC power supply. On the B Series, +5 VDC can be found on pin 11 and -12 VDC on pin 18, and it seems pin 24 is implemented. Adapters for use with the VIC-20 and C-64 can result in other minor variances. Obviously you will have to refer to the manual for your particular piece of equipment to be certain about how your RS-232 is implemented.

Because this universal cable allows access to each line, one can easily connect a voltmeter to any pin and one can therefore find out what is going on both from a hardware as well as a software point of view by observation and by trial and error.

Building the Universal Cable

This project is extremely simple to build. It would definitely belong in a beginners category. You will need one DB-25 connector to plug into your computer (check if male or female) and then two more connectors, one male and one female. Then you will need either a 5 foot length of 25 conductor ribbon cable or a few different coloured spools of single conductor wire.

If you get the flat ribbon cable, (Radio Shack #278-772), make sure you buy the solderless DB-25 connectors (Radio Shack #276-1559 and #276-1565). This is actually the easiest way to go as it will save you lots of soldering. The single conductor route is cheaper but soldering the wires into the DB-25 connectors (Radio Shack #276-1547 and #276-1548) is tricky.

Also you will need 50 – 1 1/4 inch finishing nails and a piece of scrap 1/2 inch plywood or particle board measuring about 5 by 8 inches. Later you may also need a package of 8 jumper wires with alligator clips attached. All of this should cost less than a ready made cable.

The actual assembly of the parts is as follows:

1. Make two photocopies of the DB-25 connector and cut them out from the paper. Glue them to the board as shown in the diagram. These will serve as templates for putting in the nails and will provide a means of labelling the pins.

2. Drive the nails into the board in accordance with the template.

3. Attach the connector that will go to your computer to a 2 foot length of cable and at the other end of the cable carefully separate the individual strands of wire for about 4 inches. Strip a 1/2 inch of insulation from each wire.

4. Carefully locate pin #1 on the connector. You may need a magnifying glass but it should be written on the plastic near the pin or socket. Now locate the corresponding wire (you may wish to check using an ohmmeter or a battery and light bulb.

5. Neatly wrap the bare end of this wire around nail #1 and fasten with a dab of solder. (Don't worry, the paper will not burn up!)

6. Connect the remaining wires to the corresponding nails in a similar manner. You need only connect those wires you will actually use on your computer but I recommend connecting all 25 since this device may later be used with some other machine.

7. Attach both a male and a female DB-25 to one end of the remaining 3 feet of ribbon cable, making sure that pin #1 and socket #1 are connected to the same wire. If using the solder-type connectors, you will have to prepare 2 separate cables.

8. Connect the cable(s) to the other bank of nails as in steps 4 and 5.

You are now ready to plug one end of your universal cable into your computer and the other end into the device. Having both genders of plug on the device side allows you to connect regardless of which type of connector the device may have. Connect the jumper cables with the alligator clips to the nails to make the desired connections between the various pins.

The advanced electronics hobbyist may mount this device in a suitable box, install crossover switches to the more common connections and add LED's to indicate signals on the various lines. This device can also function as a null modem by jumping the outgoing lines back to the incoming lines.

Common RS-232 Usage

Some knowledge about the conventional methods of interfacing RS-232 devices is a helpful starting point for using the universal cable in a new application.

A minimal hookup can be accomplished with as few as 3 lines connected. An RS-232 link could be as follows:

```
      2 -------------- 2
DTE   3 -------------- 3  DCE
      7 -------------- 7
```

Such a hookup would give no hardware handshaking capabilities. If 2 DTE's are to be connected, the transmitted data (TXD, pin 2) must be sent to the received data (RXD, pin 3) on the other terminal. Therefore the hookup is as follows:

www.Commodore.ca
May Not Reprint Without Permission

```
      2 ------------- 3
DTE   3 ------------- 2  DTE
      7 ------------- 7
```

Suppose a printer is connected to a terminal. A signal from the printer that its buffer is full may be needed. The printer may not be equipped to send any code back to the terminal. The data set ready line, (DSR, pin 6) may be used:

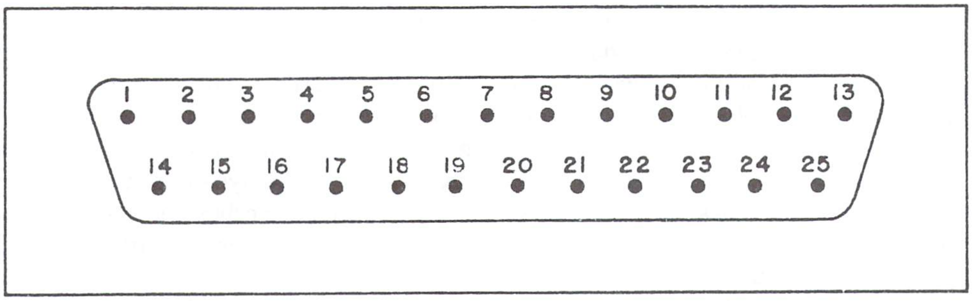
```
      2 ------------- 2
DTE   6 ------------- 6  Printer
      7 ------------- 7
```

DSR has other purposes. It is used with modems to indicate that power is on, for instance. Things get more complicated from here on. RTS and CTS, pins 4 and 5, are a pair of handshaking lines used with half-duplex modems. Carrier detect (DCD, pin 8) is used to indicate the presence of an active device or it may be used to signal a computer that someone is trying to make contact. Data terminal ready (DTR), pin 20) is complementary to DSR, that is the terminal will indicate that it is ready to receive data.

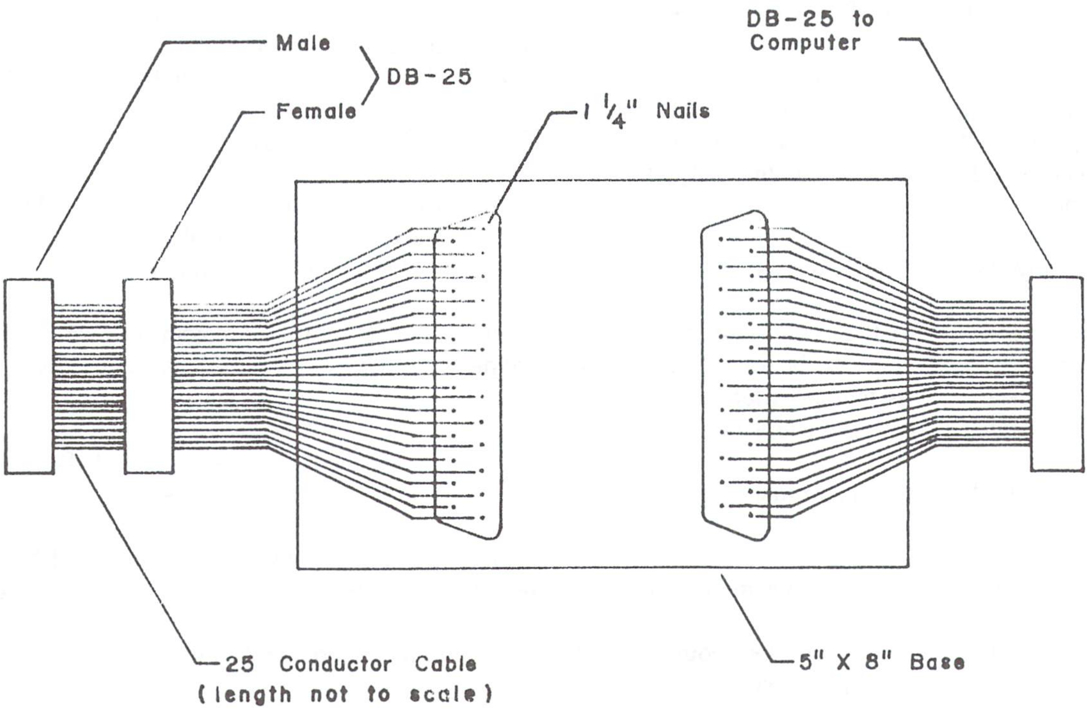
Jumping one line to another is a means of fooling the host computer into thinking that all necessary lines are active. For example, to connect a SuperPET to another computer, say a Radio Shack Model 100, the SuperPET side has pins 4 and 5 jumpered as well as pins 6, 8 and 20. This arrangement is as follows:

```
             1 ------------- 1
             2 ------------- 3
SuperPET     3 ------------- 2  Model 100
             7 ------------- 7
         4 × 5
    6 × 8 × 20
```

One other important line is the protective ground (pin 1). It is used to connect the chassis of the two devices so they have a common ground potential. Sometimes the logic ground is actually the same as the protective ground. The other pins are rarely used or supported. While there may still be voltage differences, communications protocol incompatibilities or software problems which will interfere with proper interconnection of two RS-232 devices, chances are that if pins 1 to 8 and pin 20 are correctly connected, the interface will work.



DB-25 Connector



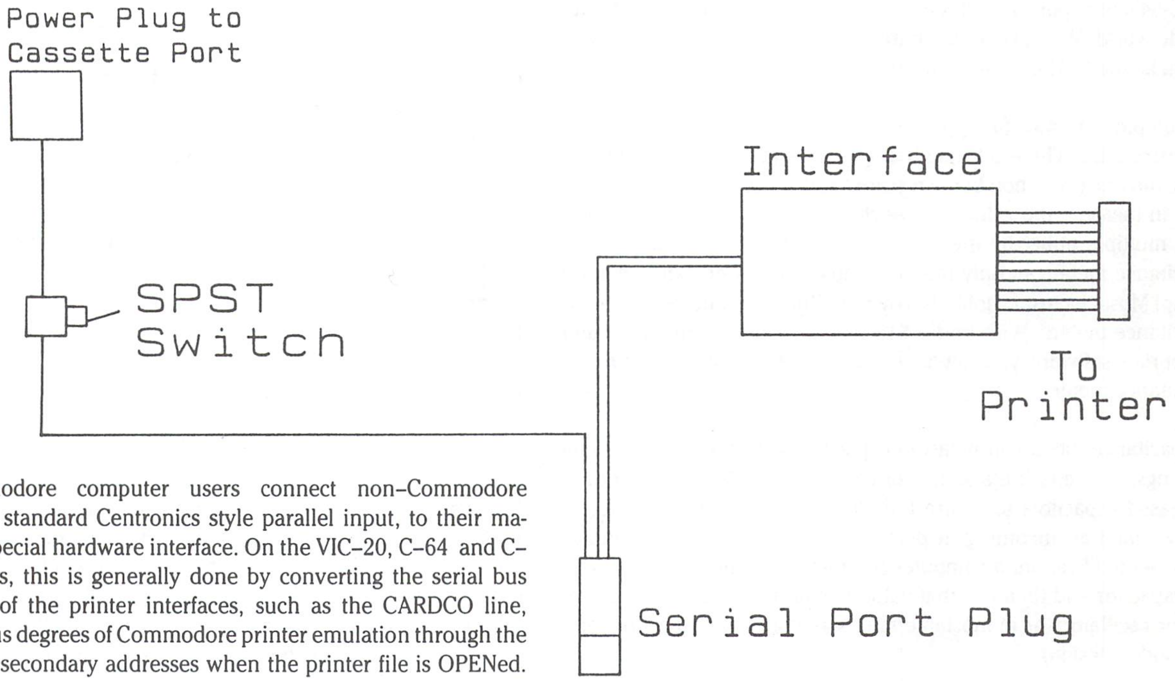
Universal RS-232 Cable Layout

www.Commodore.ca
May Not Reprint Without Permission

A $2.00 Printer Interface Reset Switch

Miklos Garamszeghy
Toronto, Ontario

. . .to exit from a locked-in interface mode, you must normally turn off the power to the computer. . .



Many Commodore computer users connect non-Commodore printers, with standard Centronics style parallel input, to their machines via a special hardware interface. On the VIC-20, C-64 and C-128 computers, this is generally done by converting the serial bus signal. Some of the printer interfaces, such as the CARDCO line, provide various degrees of Commodore printer emulation through the use of special secondary addresses when the printer file is OPENed. The CARDCO interfaces also allow you to "lock in" a particular operating mode, which can only be re-set by turning off the computer. These locked modes disable or enable certain software selectible interface features (such as PETSCII to ASCII conversion) and are generally used when you want to prevent such a selection from occurring accidentally (such as for bit image graphics work, where all sorts of strange character data may be sent to the printer). Unfortunately, to exit from a locked-in interface mode, you must normally turn off the power to the computer. This is not always desirable, especially when you are in the middle of a long program. My solution to this problem is to install a reset switch on the power line to the interface.

Most printer interfaces draw their power from the cassette port. By installing a switch in this power line, the power to the interface can be shut off, thus resetting it without crashing the printer or computer. The switch can be any type of SPST toggle switch, or a normally closed (NC) momentary contact SPST pushbutton can be used instead. A suitable switch can be purchased in a vast variety of styles, with either screw or solder type connections, at an Electronics supply store such as Radio Shack for a few dollars or less. Since the voltage and current handled by the switch is minimal, the electrical rating of the switch is not very important.

Connect the switch as shown in figure 1. Make sure that all connections are neat and tight, with no loose strands of wire hanging off. The switch can be mounted on a small piece of scrap perf board (or similar stiff plastic) or in a small case. (I use an old 35mm film can.) The perf board can be permanently attached to the back or top of the computer with a dab of 5 minute epoxy or similar type of high strength glue. It is also possible to mount a small switch inside the case of some of the larger printer interfaces, such as the CARDCO +G. In this case, make sure that you can locate the correct wire for for the power inside the interface (it should be marked on the circuit board, but use a voltmeter if you are not sure), and that the switch connections will not short out against something inside the interface. The switch can also be permanently installed by making a small hole in the back of the case of the computer.

In addition to acting as a reset switch, a printer interface power switch can also provide other benefits. The most obvious one is that it allows you to cut off the power to the interface when it is not being used. Commodore computer power supplies tend to be stretched to their operating limits -- cutting out unnecessary power drains, however small, may be beneficial to the life of your power supply. The second benefit deals with recognition of the printer when it is turned on. Some combinations of printers and interfaces will not work (i.e. device not present error) unless you turn on the printer before you turn on the computer. (My Roland printer with a G-WIZ interface won't work unless it is turned on first, but my old daisywheel doesn't care when it is turned on.) This would normally present a bit of a complication if, for example, you decided to print out a document with your favorite word processing program, but didn't turn on the printer before you started. In these cases, all that is required is that you turn on the printer before turning on the PRINTER INTERFACE power. With the reset switch installed, this is a simple task!

www.Commodore.ca
May Not Reprint Without Permission

The Commodore 64 Capacitance Meter

Jim Barbarello
Englishtown, NJ

. . .a capacitance meter can only measure capacitance, and can cost $100 and up!

The C-64's user port provides a convenient and easy interface to the outside world. With just a little hardware and the right software, you can make the C-64 do some amazing things.

One simple but powerful application is making the C-64 double as a test instrument. The electronic hobbyist uses many types of components, the most common being resistors and capacitors. A multimeter that can measure the value of a resistor may cost as little as $15 and serve multiple utility by measuring voltage and current also. But a capacitance meter can only measure capacitance, and can cost $100 and up! Most electronic hobbyists own multimeters, but very few own capacitance meters. With under $15 worth of parts, a little time and appropriate software, you can have your C-64 double as a very precise capacitance meter.

A capacitance meter can measure capacitors with cryptic or missing markings, test capacitor stability, or even measure large quantities of purchased capacitors to insure they are within specifications (commonly called an incoming inspection "go-no go" test). With minor software modification, a computerized meter can measure the value of a capacitor and then use that value to compute the other parameters for oscillators or monostable multivibrators (one aspect of computer aided design).

Aside from producing a low cost and useful product, this project will provide you with an insight into how you can experiment with the user port.

MEASUREMENT CAPACITANCE:

If a capacitor is provided with a fixed voltage, it will charge to a specific voltage level within a time that can be determined mathematically. The circuit of Figure 1 is a 555 Timer Integrated Circuit (IC) connected in the monostable (one shot) mode. When a low voltage is provided to pin 2, the voltage at pin 3 immediately rises to the supply voltage (V+) and the unknown capacitor (C) begins charging. After a time equal to $1.09866 \times R \times C$, the capacitor has been charged to two thirds of V+ and the voltage at pin 3 returns to ground. If the same capacitor and resistor are used, this time will not change.

With the value of R and the charging time known, the above formula can be used to calculate the value of C. This very simple circuit forms the basis of an accurate capacitance meter. In practice, the C-64 sends out a very short negative pulse to pin 2 of a 555 IC, starting the timing cycle. It then counts until the voltage level at pin 3 of the IC changes from V+ to ground. The count is used in a formula to calculate the value of the unknown capacitor.

THE HARDWARE:

The schematic diagram of Figure 2 shows the capacitance meter. It differs from Figure 1 in that the 555 Timer IC (U1) is now connected to

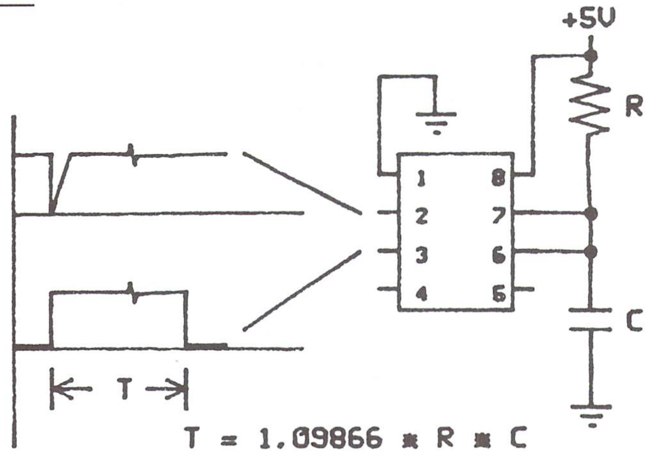


Figure 1: 555 Timer Specs

the C-64 user port, the unknown capacitor has been replaced by two binding posts, and an additional timing resistor and integrated circuit switch (U2) have been added. The user port will provide the trigger and sense U1's status. The binding posts will be used to attach an unknown capacitor to the circuit. The additional IC and resistor will provide the capability to measure a broad range of capacitance values. With R1 only, the meter can measure capacitors with values between 20 picofarads (pf) and about 0.2 microfarads (uf). Placing R2 in parallel with R1 decreases the effective resistance between pins 7 and 8 of U1 from 10 megohms to 10 kilohms. This allows the meter to measure capacitance between 0.1 uf and 150 uf. U2 is an electronic switch. When the input voltage to the control pin (13) is at ground, the switch is open and the resistance between pins 7 and 8 of U1 is 10 megohms. When the voltage at pin 13 of U2 is raised to 5 volts the switch closes, placing R1 and R2 in parallel and decreasing the effective resistance to 10 kilohms. Switch U2 allows the meter to switch ranges under computer control. Power is provided from pins 1 (ground) and 2 (+5 volts) of the user port.

THE SOFTWARE:

While most of the software is written in Basic, the portion that triggers the hardware and counts until done is machine language. This is necessary since, with a capacitor value of 20 pf, the time to be measured by our meter would be 1.09866×20 E-12 $\times 10$ E+6, or approximately 22 microseconds. Basic is just too slow for this task. The machine language utility is imbedded in the Basic program and called via the SYS command.

The software must also set up the user port with line PB0 as an input, and lines PB1 and PB2 as outputs. Pages 360 and 361 of the Commodore Programmer's Reference Guide identify the data direction register at memory location 56579. Poking this location with the number 254 (1111110 binary) causes lines PB7 through PB1 to be set as outputs and line PB0 to be set as an input.

www.Commodore.ca
May Not Reprint Without Permission

Figure 2: Schematic Diagram

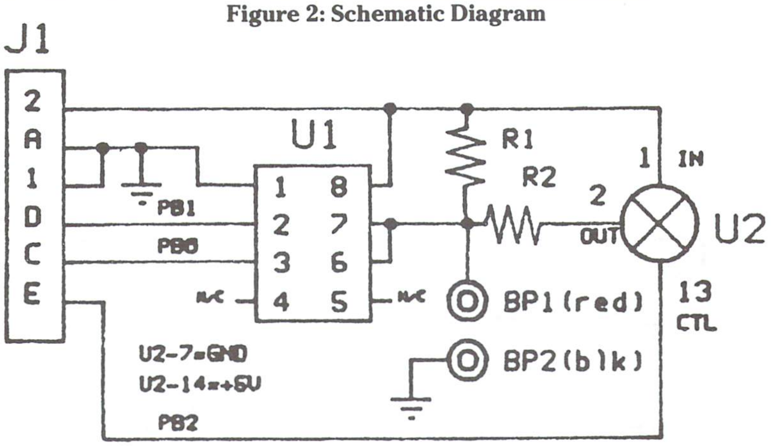


Figure 3: PC Board, Component Side

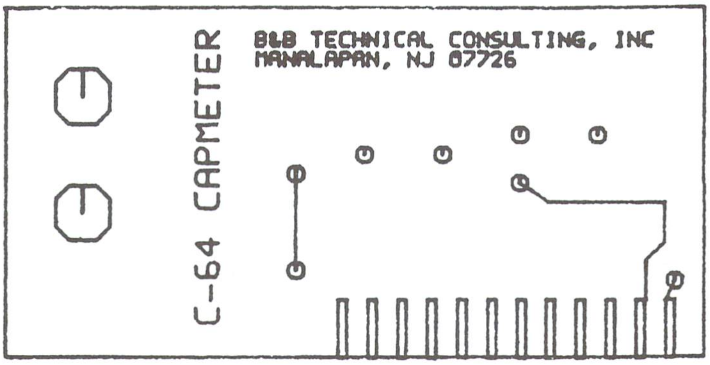
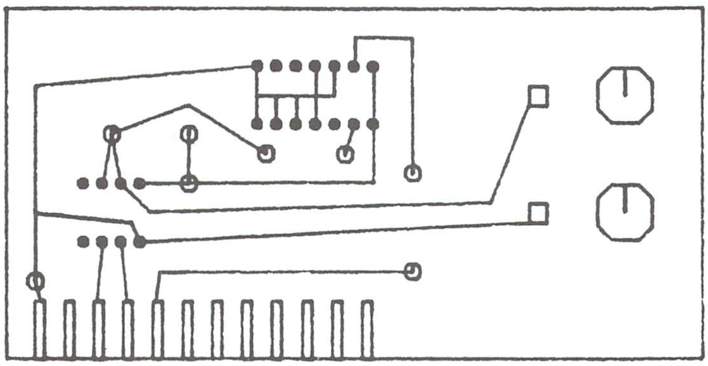


Figure 4: PC Board, Wiring Side



Poking memory location 56577 (CIA chip #1, Port B) changes the voltage level on the lines that have been set as outputs. For instance, poking 56577 with a 2 (00000010 binary) will cause line BP1 to go high, PB2 through PB7 to go low, and PB0 to remain unchanged (since it was set as an input line). Alternatively, peeking 56577, and performing a logical AND on the results (PEEK(56577) AND 1) will indicate PB0's logic state. A zero result means PB0 is low and a one result means PB7 is high. The software first addresses the data direction register at 56579 to define which lines are inputs and outputs. It then momentarily changes the status of line PB1 from high to low to high again, beginning the timing cycle for U1. Then it continually senses the status of line PB0 until it senses a ground voltage condition, counting the number of times it has checked PB0. Finally, the software uses a mathematical relation to convert that count into a capacitance value. If the user selects the low range, the software pokes 56577 with a 2 (00000010 binary), making line PB2 low and opening the U2 switch. If the high range is selected, address 56577 is poked with a 6 (00000110) to keep PB1 high but close the U2 switch. Line PB1 (trigger input) must remain high at all times except when the hardware is to be triggered.

CONSTRUCTION:

While the circuit could be constructed with any standard method (including point-to-point wiring), best results are obtained with a printed circuit board (PCB). Fabricate a printed circuit board using the patterns shown in Figures 3 and 4. When completed, mount the components on the PCB as shown in Figure 5 (clip off the excess resistor leads after soldering and save for jumpering as described below). Note that IC sockets are soldered to the PCB and the ICs inserted in the sockets in the orientation shown. U2 is a CMOS (Complimentary Metal Oxide Semiconductor) device and, as such, is sensitive to static field damage. Handle this IC as little as possible, preferably by the ends. Before handling, touch a ground point (such as the screw holding an electrical outlet cover) to drain any excess charge present on your body. Solder the eleven leads on the 22 pin connector to the component side of the PCB. Turn the PCB over and bend the remaining eleven pins down to touch the eleven PC leads below them and solder to the PCB.

Note the three holes marked "J" in Figure 5. For each hole, place an excess resistor leads in the hole. Solder the lead to the pad on each side of the PCB. Clip off the excess lead.

Mount the two binding posts on the PCB as shown in Figure 5. For each hole, place an excess resistor leads in the hole. Solder the lead to the pad on each side of the PCB. Clip off the excess lead.

Mount the two binding posts on the PCB as shown in Figure 5. Melt a small amount of solder onto each of the two rectangular pads on the PCB. Place the end of a short length of wire onto one of the pads and reheat the solder, connecting the wire to the pad. Attach the other end of the wire to the binding post. Repeat this procedure with another short length of wire, connecting the remaining binding post to the other rectangular pad.

USE:

Type in and save program listing 1 using the name "CAP". Slide the meter connector (J1) onto the user port PC edgeboard (left rear of the computer) so the ICs are on the top surface of the board and the binding posts are on the left. Power up the computer, then load and run the "CAP" program.

A representation of a meter will appear on the screen with a display area (the blue rectangle) near the meter top. Below the display area are four "buttons" labelled F1 (low range), F3 (high range), F5 (clear display) and F7 (off). Pressing any of the corresponding function keys will cause the label to reverse color while the associated function is being performed. The low range is used to measure capacitors between 20 pf and 0.2 uf. The high range measures capacitors between 0.1 uf and 150 uf. For unmarked capacitors use either range. If the capacitor being measured is not within the range selected, the indication "OUT OF RANGE" will appear in the display area of the current reading or message. Pressing F7 ends the program and displays the message "METER OFF – PROGRAM ENDED".

www.Commodore.ca
May Not Reprint Without Permission

OPTIMIZING PERFORMANCE:

Two factors affect the final accuracy of the meter; values of resistors R1 and R2, and the stray capacitance of the hardware. These factors will vary with the specific resistors and fabrication method you use. Note the variables R(0), F(0), R(1) and F(1) in line 10. These are the values of the resistance and stray capacitance for the low (0) and high (1) ranges. To optimize your meter, you'll need a digital multimeter capable of measuring resistance up to 11 megohms (an analog multimeter has an accuracy of about 3 percent and, therefore, is not accurate enough for this task).

With the meter disconnected from the computer, remove both U1 and U2. Measure and note the value of R1 and R2 in megohms (EX: 10.01 for R1 and .00979 for R2). Change the value of R(0) in line 10 to the value you measured for R1. Similarly, change the value of R(1) to the value you measured for R2. Save the modified program. (NOTE: If a digital multimeter is not available, use the nominal values of 10 and .01 for R(0) and R(1).).

Replace U1 and U2, being sure to observe the orientation shown in Figure 5. Reinstall the meter, power up the computer and load the cap program. Edit line 110 to add the statements :PRINT X:STOP at the end of the line. With no capacitor connected, select the low range. A number will appear along with the message "BREAK IN 110". Note this number as F(0). Repeat this procedure, this time selecting the high range and noting the resulting number as F(1). Change the values of F(0) and F(1) to the values you just noted. Delete the :PRINT:STOP statements you added to line 110 and resave the program.

Once this procedure to optimize the program to your specific hardware has been performed, it need never be repeated. The meter will retain its accuracy without any further calibration.

SUMMING IT UP:

This low cost, simple project provides a useful test tool for the electronic hobbyist and shows how the C-64 user port can be used for low cost, effective interface to the outside world. I'd like to hear your thoughts on this type of simple hardware project, and if you'd like to see others in the future. Please address any correspondence to me at RD#1, Box 241 H, Tennent Road, Manalapan, NJ. I'll answer any questions that are accompanied by a self addressed stamped envelope.

List Of Materials

BP1	Red 5-way Binding Post
BP2	Black 5-way Binding Post
J1	12/24 Contact PC Card Edge Connector (.156 spacing, solder eyelet terminals)
R1	10 megohm, 1/4 watt, 5% fixed resistor
R1	10 kilohm, 1/4 watt, 5% fixed resistor
SO1	8 Pin IC Socket (for U1)
SO2	14 Pin IC Socket (for U2)
U1	555 Timer IC
U2	4016 CMOS Quad Bilateral Switch IC

Miscellaneous: Double sided PC board (see text)
two short lengths (1.25"each) of #22 solid wire
solder, etc.

NOTE: A kit containing all parts, the CAP program, a 555 timer design program using direct input from the meter (both on disk) and an instruction manual, is available for $15.00 (plus $2.00 U.S. shipping) from B & B Technical Consulting, Inc., RD#1, Box 241H, Tennent Road, Manalapan, NJ 07726. Specify Kit C64CAP. NJ residents include $0.90 additional sales tax.

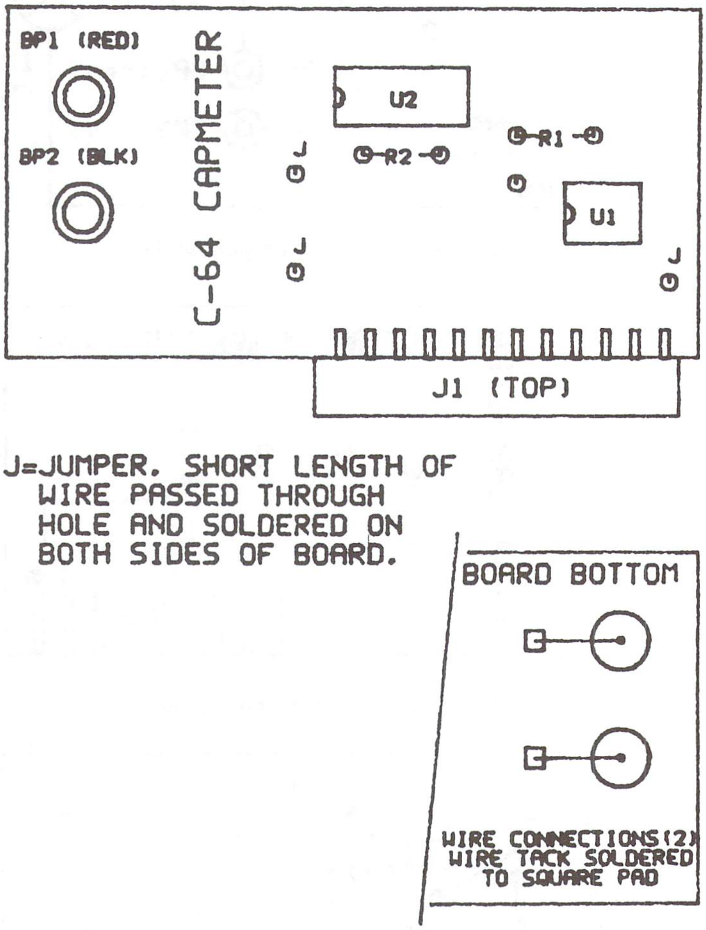


Figure 5: Component Placement

Listing 1: The CAP Program

```
   1 rem **********************************
   2 rem ** capacitance meter software   **
   3 rem ** name: cap                    **
   4 rem ** (c) 1985, j.j. barbarello    **
   5 rem ** manalapan, nj 07726          **
   6 rem ** v 1.1, 11 nov 85             **
   7 rem **********************************
JO 10 gosub 440: print: r(0)=9.75: f(0)=19: r(1)=.00979
    : f(1)=2
EN 20 data 120, 169,   0, 141,   1, 221, 169,   2, 141,   1
KL 30 data 221, 162,   2, 160,   0, 169,   1,  45,   1, 221
BF 40 data 240,  15, 232, 234, 234, 234, 234, 234
OP 50 data 224,   0, 208, 239, 200, 192,   0, 208, 234
NI 60 data 142,   0, 193, 140,   1, 193,  88,  96, 999
DE 70 a=49152: c=a: for i=1 to 16: sp$=sp$+
    "[1 spc]": next
AE 80 b1$=chr$(176)+"CC"+chr$(174)
CF 90 b2$=chr$(173)+"CC"+chr$(189)
IH 100 read b: if b<>999 then poke a,b: a=a+1: goto 100
```

www.Commodore.ca
May Not Reprint Without Permission

```
EG  110 gosub 620: poke 56579,254: poke 56577,6
JJ  120 col=10: ro=9: gosub 670: print b1$: ro=10
        : gosub 670: print "Bf1B";
HG  130 print b$; "←low range": ro=11: gosub 670
        : print b2$
LO  140 ro=12: gosub 670: print b1$: ro=13: gosub 670
        : print "Bf3B";b$; "←high range"
FD  150 ro=14: gosub 670: print b2$: ro=15: gosub 670
        : print b1$
CK  160 ro=16: gosub 670: print "Bf5B";b$;
        "←clear display": ro=17: gosub 670: print b2$
KC  170 ro=18: gosub 670: print b1$: ro=19: gosub 670
        : print "Bf7B";b$; "←off"
OM  180 ro=20: gosub 670: print b2$: goto 270
FO  190 co=12: ro=5: gosub 670: print sp$
NH  200 sys c: x=peek(49409)*256+peek(49408)
NC  210 if ri=0 and x<f(0)+5 then x=0: goto 240
GA  220 if x>1000 then 240
LJ  230 av=0: for i=1 to 10: sys c: x=peek(49409)*256
        +peek(49408): av=av+x: next: x=av/10
FP  240 printchr$(159): if x<=f(ri) then x$=
        "[3 crsr lefts]out of range": x=0
FE  250 co=18: row=5: gosub 670: x=(x-f(ri))/(43300*r(ri))
        : gosub 510: print x$
NA  260 ro=rr: co=11: gosub 670: print fu$
OG  270 get a$: if a$="" then 270
NJ  280 g=asc(a$): if g<133 or g>136 then 270
KM  290 on g-132 goto 300,320,340,360
PG  300 ro=10: co=11: gosub 670: printchr$(18); "f1"
        : rr=10: fu$="f1"
GI  310 poke 49159,2: poke 56577,2: ri=0: fi=0: goto 190
AK  320 ro=13: co=11: gosub 670: printchr$(18); "f3"
        : rr=13: fu$="f3"
OF  330 poke 49159,6: poke 56577,6: ri=1: goto 190
IG  340 gosub 400: goto 270
HN  350 poke 49408,0: poke 49409,0: goto 190
JF  360 rem** end
FG  370 printchr$(147): ro=12: co=10: gosub 670
FH  380 printchr$(18); " meter off ";chr$(146);
        " - program ended."
EO  390 print: print: print: end
MH  400 rem** clear display (f5 function)
OM  410 ro=16: co=11: gosub 670: printchr$(18); "f5"
ND  420 co=12: ro=5 : gosub 670: print sp$
        : for i=1 to 200: next i
NL  430 ro=16: co=11: gosub 670: print "f5": return
IB  440 rem** format screen=
AD  450 poke 53280,6: poke 53281,6: printchr$(147)
AN  460 b$=chr$(30)+chr$(18): bl$="[8 spcs]"
        +b$+"[24 spcs]": printbl$
GD  470 printtab(8);chr$(30)chr$(18)chr$(142);" c-64
        capacitance meter "
LB  480 printbl$: bb$="[8 spcs]"+b$+"[2 spcs]"
        +chr$(146)+"[20 spcs]"+b$+"[2 spcs]"
DC  490 printbb$: printbb$: printbb$
NL  500 for i=1 to 14: printbl$: next i: print bl$: return
NL  510 rem ** format output
BD  520 if x<=0 then return
LC  530 p$=right$(str$(x),4): if asc(p$)<>69 then 580
PO  540 p=val(right$(p$,2)): po=p+2
KH  550 x$=str$(int(x*10↑po+.5))
OA  560 x$=right$(x$,len(x$)-1): if p=5 then
        x$=left$(x$,2)
FO  570 x$=x$+"[1 spc]"+chr$(18)+"pf": return
MN  580 p=1: if x<1 then p=1000: goto 610
EE  590 if x<10 then p=100: goto 610
GA  600 if x<100 then p=10
PI  610 x=int(x*p+.5): x=x/p: x$=str$(x)
        : x$=right$(x$,len(x$)-1)+" uf": return
CO  620 rem* cursor control using plot        kernel ($fff0)
BN  630 data 162, 0, 160, 0, 24, 32, 240, 255, 96, 999
JI  640 a=49300: sc=a
NK  650 read b: if b<>999 then poke a,b: a=a+1: goto 650
AL  660 return
BJ  670 poke sc+3,col: poke sc+1,row: sys sc
EM  680 return
```

Listing 2: Capmeter measuring utility source code

```
     *= $c000   ;execution start at 49152
     sei        ;disable interrupt requests
     lda  #0    ;set register mask for all 0's. Basic program
                 has previously set the data direction register
                 and set PB1 (trigger) high.
     sta  $dd01 ;bring PB1 low to trigger.
     lda  #2    ;set register mask for PB1 high.
     sta  $dd01 ;bring PB1 back high.
     ldx  #1    ;x will be the least significant bit (LSB) of the
                 count.
     ldy  #0    ;y will be the most significant bit (MSB) of the
                 count.
cont lda  #1    ;A to be ANDed with $DD01 contents.
     and  $dd01 ;if timing cycle done, PB0 will be low and
                 ANDing results in zero.
     beq  done  ;if zero result, counting done.
     inx        ;otherwise, increment count by one.
     nop        ;add 10 machine cycles to slow
     nop        ; down the count. This produces
     nop        ; a count consistent with values
     nop        ; of resistance in the hardware
     nop        ; and desired measurement ranges.
     cpx  #0    ;has x reached 256 (overflow to 0)?
     bne  cont  ;no. go back for next count.
     iny        ;yes. increment MSB.
     cpy  #0    ;has count reached 65536?
     bne  cont  ;no. go back for next count.
done stx  $c100 ;store LSB count at $C100 and MSB
     sty  $c101 ; at $C101 for retrieval by Basic prg.
     cli        ;re-enable interrupt requests.
     rts        ;return to Basic program.
.end
```

Listing 3: Utility for use of "PLOT" Kernel for screen cursor placement (source code)

```
     *= $c094   ;execution starts at 49300
     ldx  #0    ;row number will be poked into location now
                 storing #0 when utility is called.
     ldy  #0    ;col number will be poked into location now
                 storing #0 when utility is called.
     clc        ;clear carry flag tells Kernel you want to move
                 the cursor, not read its current location.
     jsr  $fff0 ;call "Plot" Kernel to move cursor.
     rts        ;return to Basic program.
.end
```

www.Commodore.ca
May Not Reprint Without Permission

Commodore 64 Frequency Counter

Lorne Klassen
East Kelowna, BC

Put some of the 64's idle hardware to work!

I have always been interested in practical applications for personal computers. There are many more things that can be done with one besides playing the latest game. This article describes one such application. Many of the features of the chip set in the 64 are either unused or underused by the operating system. The 6526 CIA chips can be used for many other functions besides timing and I/O. The timers in the 6526 can be used to count external signals which are applied to the CNT pin. This pin is available on the user port. By using this feature, one can count external signals and then process that count. There are several applications for this, but one of the most interesting is to use this for measuring the frequency of an applied signal.

How The Program Works

To measure the frequency of a signal, one must count the number of pulses for a certain length of time and then convert that count to the frequency. If the time length used is one second, then the count will be the frequency in cycles per second and no other conversion is necessary. The biggest restriction here is that one is limited to the maximum count that the registers can hold. This can be overcome by either shortening the time length, dividing down the signal before it is applied, or using another register. With this program one can select either one second or one-tenth second gate time. I have used the CIA #2 chip for this program as its timers are not used by the operating system. Only timer A is used , but one can adjust the program to use both timers if a larger count is desired.

By setting bit 5 of the control register for timer A, it will count external signals. The assembly listing is fairly self-explanatory, but a few items should be noted. The IRQ vector is changed to point to our routine. This allows one to update the count more accurately than a BASIC-only program would allow. A start address of $C000 is used but one can re-assemble to a different location if desired.

Since the IRQ happens 60 times a second and we only want to get the count every 0.1 second or 1 second, a flag register is used. This register is first loaded with a value equal to the desired number of IRQ's per count update, then decremented each IRQ. When the flag register has been decremented to zero, the count is updated. The gate value is stored at 822. It contains the value to be loaded into the flag register. If changed while the program is running, it will change the gate time. 822 is set to 60 at start-up.

One problem with the CIA timers is that they are down-counters and what we want is up-counters. By initially setting the counter to $FF and then Exclusive-ORing the final count with $FF results in the counters effectively being up-counters. This is done in the machine code so that it does not have to be done in BASIC. To get the count, one must stop the counter, read out the count, reset the counter and then restart it. After the count is stored, the routine jumps to the regular IRQ routine. Be aware that there could be a slight error here if a very short gate time is used. There is a slight delay between the time the counter is stopped and the time it is restarted. Even when using a 0.1 second gate time this error is not significant. If you use an extremely short gate time, the count should be adjusted to correct this. The count is stored at locations 680 and 681 in standard low byte, high byte format. If the count exceeds $FFFF, the counter will not give a true reading. If this happens either bit 0 or bit 1 of the interrupt control register will be set, depending on which timer is used. To indicate this, the ICR is ANDed with %00000011 to mask off the undesired bits, then stored at location 823. Anything other than a zero here indicates an overflow condition.

A short BASIC program is included more as a demonstration than anything else, although for most low-frequency applications it will suffice. The BASIC program allows the selection of either 0.1 or 1 second gate time and displays the frequency on the screen. If an overflow condition occurs, the word 'overflow' will appear under the count value. This indicates that the count is not correct and the 0.1 second gate should be selected. If you are already using that, then you must either use a pre-scaler to divide down the input signal or modify the program to utilize a shorter gate time. The shortest gate time possible is 1/60 second. This would give a maximum count of nearly 4 Mhz. However, this is too high for the 6526 to count accurately, so a pre-scaler should be used above 1 Mhz to avoid errors.

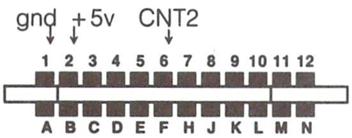
There are many modifications possible, such as storing the frequency at set time intervals or sending the display to a printer. Also, one can use both timer A and timer B. The machine code would have to be changed to include reading timer B. Also the Control Register for B would have to be set to count underflow from timer A. I leave these modifications up to the user.

www.Commodore.ca
May Not Reprint Without Permission

Hardware Notes

Since we are using the CIA #2, we must use the CNT2 connection on the user port as the input for the unknown frequency. This is pin #6. Refer to the diagram shown for more information. Any signal applied to this pin MUST be TTL (+5 volt maximum) compatible. If you are sure your signal is that, then you can apply it directly to this pin. If not, a level shifting circuit must be used. The signal applied should also have a fast rise time to ensure it will be counted. The use of a Schmidt trigger here will eliminate that problem. If your signal has an amplitude of less than about 3 volts then some sort of amplifier must also be used.

Commodore 64 User Port



(Edge-on view)

By using a one second gate value, the maximum frequency is 65535 Hz. With a 0.1 second gate, the maximum frequency is 655350 Hz. If you want to count higher frequencies than this, then you must either shorten the gate time or use a pre-scaler to divide down the input. Shortening the gate time will increase the maximum frequency, but it is best to avoid going any higher than about 1 Mhz. or the chip itself may not count accurately. You also must make sure that any circuitry that the signal is routed through has the necessary bandwidth for your application. Any circuitry used should be mounted as close as possible to the user port. Try to keep all wires as short as possible, to avoid problems.

Listing 1. BASIC portion of the frequency counter program. Run the loader in listing 2 or assemble the machine language portion to disk before running this.

```
KN  10 rem -------   frequency counter   ------
LJ  20 rem ---------   lorne klassen   --------
KO  30 rem -------   east kelowna,b.c.   ------
MJ  40 :
OI  50 rem  uses cnt2 (pin #6) on the user port to read
        in the frequency.
NH  60 rem  any signal applied to this pin must be at
        ttl level.
EC  70 rem  count is stored at 680 and 681.
KP  80 rem  gate time is stored at 822, overflow at 823
OM  90 :
GF  100 c=c+1: if c=1 then load "freq.cntr.
         @c000",8,1
BN  110 cx=-1
FA  120 print "Sq" tab(12) "frequency counter"
JC  130 print tab(12) "-----------------"
GK  140 print: print tab(6) "  press '+' for 1 sec. gate"
CL  150 print: print tab(6) "  press '-' for 0.1 sec. gate"
NJ  160 print: print tab(10) "any other key to quit"
HN  170 sys 49152 :rem  start address
GD  180 c=peek(680)+256*peek(681): if c=cx
        then 230 :rem count has not changed
HG  190 print "sqqqqqqqqqqqq[7 spcs, 7 crsr lefts]"
        ;c;d$, "cycles per second"
BG  200 cx=c
HI  210 if peek(823) then print "overflow"
EA  220 if peek(823)=0 then print "        " :rem 8
        spaces
HE  230 geta$: if a$ = "" then 180
IO  240 if a$ = "+" then poke822,60: d$ = ""
        : goto180
JI  250 if a$ = "-" then poke822,6: d$ = "[1 crsr
        left]0": goto180
NL  260 sys 49155 :rem  disconnect address
OA  270 end
```

Listing 2. BASIC program to create machine-language file "freq.cntr.@c000" on disk.

```
DD  10 rem* data loader for "freq cntr" *
LI  20 cs=0
FI  30 for i=1 to 133: read a: cs=cs+a: next
GK  50 :
BP  60 if cs<>13602 then print "!data error!": end
IO  70 rem create object file on disk
DM  80 open 1,8,1, "0:freq.cntr.@c000"
BC  90 print#1,chr$(0);chr$(192);
PB  100 restore: for i=1 to 133: read a
IL  110 print#1,chr$(a);: next i
DL  120 close 1: end
GP  130 :
BM  1000 data  76,  23, 192, 120, 173,  52,   3, 141
BL  1010 data  20,   3, 173,  53,   3, 141,  21,   3
OO  1020 data 169,   0, 141,  14, 221,  88,  96, 120
DH  1030 data 173,  20,   3, 141,  52,   3, 173,  21
HJ  1040 data   3, 141,  53,   3, 169,  77, 141,  20
BC  1050 data   3, 169, 192, 141,  21,   3, 169, 255
JO  1060 data 141,   4, 221, 141,   5, 221, 169,  60
BF  1070 data 141,  54,   3, 173,  54,   3, 141, 167
MI  1080 data   2, 169,   0, 141,  13, 221, 169,  49
KD  1090 data 141,  14, 221,  88,  96, 206, 167,   2
MJ  1100 data 208,  48, 173,  54,   3, 141, 167,   2
NF  1110 data 169,  32, 141,  14, 221, 173,   4, 221
PI  1120 data  73, 255, 141, 168,   2, 173,   5, 221
PC  1130 data  73, 255, 141, 169,   2, 169, 255, 141
JF  1140 data   4, 221, 141,   5, 221, 169,  49, 141
EF  1150 data  14, 221, 173,  13, 221,  41,   3, 141
GP  1160 data  55,   3, 108,  52,   3
```

www.Commodore.ca
May Not Reprint Without Permission

Listing 3.

Assembler source code for the frequency counter program.

```
BG 100 rem open 1,8,1," @0:freq.cntr.@c000" :rem file
       for object code
OO 110 sys 700 ;pal 64 assembler
AO 120 .opt oo
LP 130 ; save" @0:freq cntr.pal ",8
PB 140 ;---------------------------------
CJ 150 ;-           frequency counter        -
BM 160 ;-             source  code           -
ND 170 ;---------------------------------
PM 180 ; uses cia #2, timer a
GH 190 ; count is stored at 680, 681
GG 200 ; gate value is stored at 822
PB 210 ; overflow sets 823
NL 220 ;.opt o1 ;sends object code to disk
MF 230 ;
FO 240 *        =     $c000       ;start address
CD 250 ;sys 49152-to start counting
PL 260 ;sys 49155-to stop counting and disable interrupt
                 wedge
KN 270 ;system equates
DM 280 cia2     =     $dd00
PC 290 talo     =     cia2+$04    ;timer a count registers
OB 300 tahi     =     cia2+$05
JI 310 icr      =     cia2+$0d    ;cia interrupt control
                                  register
EA 320 cra      =     cia2+$0e    ;cia control register
AF 330 oldirq   =     $0334       ;storage for old irq
GK 340 irqvec   =     $0314
ED 350 flag     =     679
LK 360 count    =     680
EJ 370 gate     =     $0336       ;storage for count down
                                  value
MI 380 overflow =     823
MP 390 ;
JJ 400          jmp   connect
EM 410 ;disconnect routine
ND 420          sei
FC 430          lda   oldirq      ;put old irq vector back
EI 440          sta   irqvec      ; in
HL 450          lda   oldirq+1
HD 460          sta   irqvec+1
JP 470          lda   #$00
ME 480          sta   cra         ;stop timer
IH 490          cli
AO 500          rts
EH 510 ;
EL 520 connect  =     *
MA 530          sei               ;disable interrupts
HI 540          lda   irqvec
OP 550          sta   oldirq      ;store old irq vector
NF 560          lda   irqvec+1
NG 570          sta   oldirq+1
AB 580          lda   #<start     ;point to our routine
HP 590          sta   irqvec
DA 600          lda   #>start     ;same with high byte
NM 610          sta   irqvec+1
JC 620          lda   #$ff
CB 630          sta   talo
FG 640          sta   tahi        ;load timer latch with
                                  maximum count
BB 650          lda   #60
NN 660          sta   gate        ;use a default value of 60
HF 670          lda   gate        ;get count-down value
NL 680          sta   flag        ;put it in the flag register
FN 690          lda   #$00
IC 700          sta   icr         ;disable cia interupts
AG 710          lda   #%00110001
PG 720          sta   cra         ;force load and start
                                  counting
IG 730          cli
NI 740          rts               ;all done so return
EG 750 ;
HF 760 ;counter routine starts here
IG 770 start    =     *
CI 780 ;
AG 790          dec   flag        ;check countdown flag
BB 800          bne   done        ;not timed out so exit
MN 810 getcnt   =     *           ;routine to read count
FH 820          lda   gate
FM 830          sta   flag        ;reset flag for next time
AH 840          lda   #%00100000 ;set bit 5
PD 850          sta   cra         ;to stop timer
KL 860          lda   talo
OA 870          eor   #$ff
PH 880          sta   count       ;convert to up-counter
                                  and store result
EL 890          lda   tahi
MC 900          eor   #$ff
LB 910          sta   count+1     ;same with high byte
FF 920          lda   #$ff
OD 930          sta   talo
KP 940          sta   tahi        ;reset timer latch
NC 950          lda   #%00110001 ;force load + start
                                  timer
AB 960          sta   cra
HP 970          lda   icr
LG 980          and   #%00000011 ;mask off upper 6 bits
                                  of status register
KI 990          sta   overflow    ;and save it
OF 1000 ;
IG 1010 ;
FF 1020 done    =     *
KE 1030         jmp   (oldirq)    ;go to normal irq
                                  routine
GI 1040 ;
GP 1050 .end
```

www.Commodore.ca
May Not Reprint Without Permission

An Inexpensive Teaching Robot For An Inexpensive Microcomputer

Rolf A. Deininger, Kevin O'Connor, and Tom K. Collopy
University of Michigan
Ann Arbor, Michigan

Figure 1. Armatron Robot Arms. The left model on top of the disk drive is unmodified and shows the two joysticks for control. At right the modified robot arm sits on top of the power supply and interface box.

INTRODUCTION

Robotics is a fascinating topic and of great interest to everyone from kindergarten to graduate school. Not a single day passes without articles in newspapers about robots and their replacing humans in the work force. There is a lot of mystique about robots, yet they can be very simply explained and demonstrated. The presently existing robots like the HERO (1) or the RHINO (2) are in the thousands of dollar range and too expensive for the average computer hobbyist and teacher. We were interested in a robot which would cost well below $100 and be controllable by an inexpensive microcomputer also less than $100. We chose the ARMATRON (3) toy robot for under $50 and a VIC-20 computer. More recently, Radio Shack has also been selling this robot for around $30.

THE ROBOT

The ARMATRON toy robot is a marvelous small robot arm powered by one single motor. It has all the functions of an industrial robot—a hand which opens and closes, a wrist, a shoulder, an elbow and a base. It is normally controlled by two joysticks at the base. These joysticks engage and disengage a variety of cams and gears to operate the functions of the robot. These mechanical linkages—a beauty in design—were removed and replaced by six individual motors to be controlled by the computer. Figure 1 shows two of the Armatron robot arms. The robot at left, purchased from Radio Shack, is the unmodified arm which is being controlled by the two joysticks in front. The robot arm at right is the one which was modified for connection to the computer. The box below this arm houses a 6 volt power supply and the circuit board.

www.Commodore.ca
May Not Reprint Without Permission

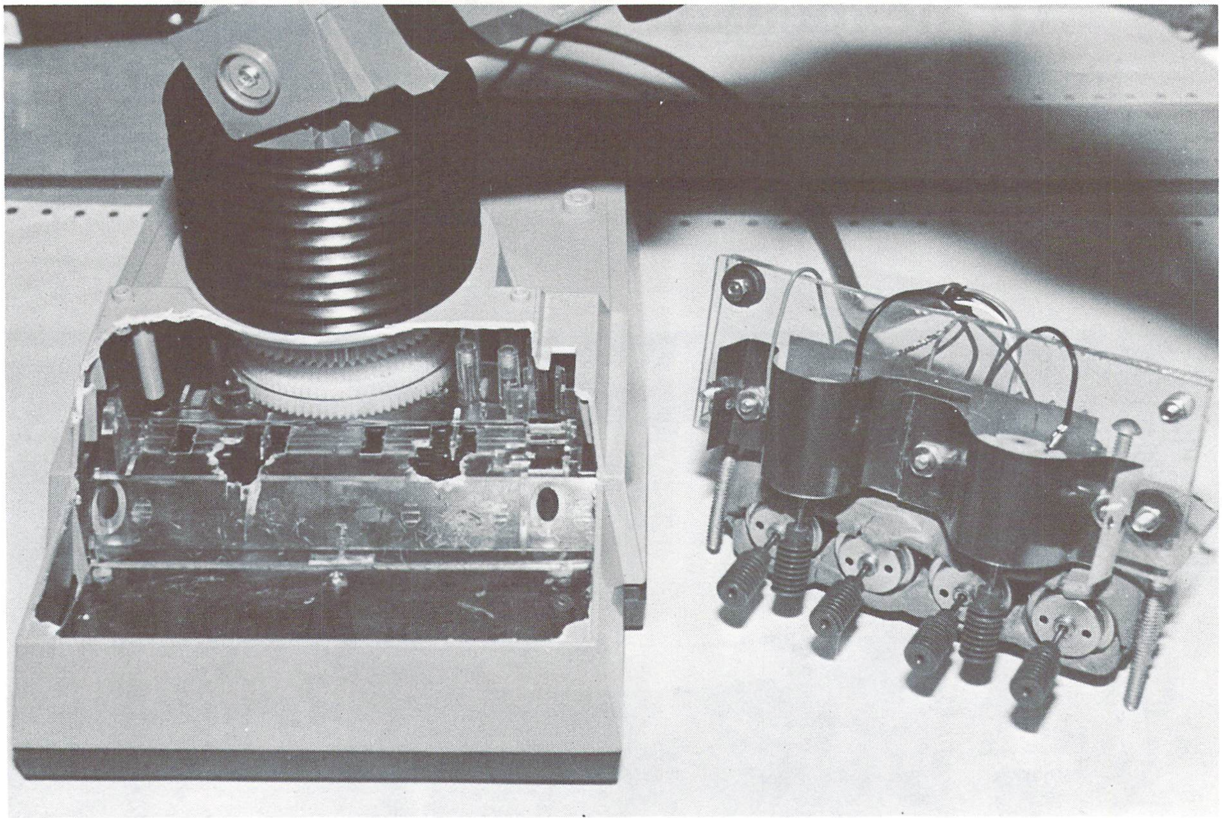


Figure 2. Modification of the Armatron robot arm required the removal of the joysticks. The assembly at right shows the six individual motors with worm gears which drive the robot arm.

It is somewhat difficult to describe the process of removal of the arms, but the entire joystick assembly was removed and replaced by a set of 6 individual motors. Figure 2 shows the open Armatron with the assembly of the six motors sitting to right. Four of the motors were mounted horizontally, and two vertically to connect via the worm gears to the gears of the Armatron which control the six major functions.

THE COMPUTER

The computer chosen was a VIC-20 (4), which is one of the most versatile and inexpensive microcomputers on the market today. The user port of the VIC is ideal for interfacing it to the outside world, and simple POKE statements allow the control of external devices. The game port of the VIC-20, usually used for the paddles and joysticks, is ideally suited for feedback of an analog signal.

THE COMPUTER TO ROBOT INTERFACE

The computer to robot interface was housed together with a power supply in a small box (see Figure 2). Figure 3 shows the general layout of the system and Figure 4 documents the circuit in general form.

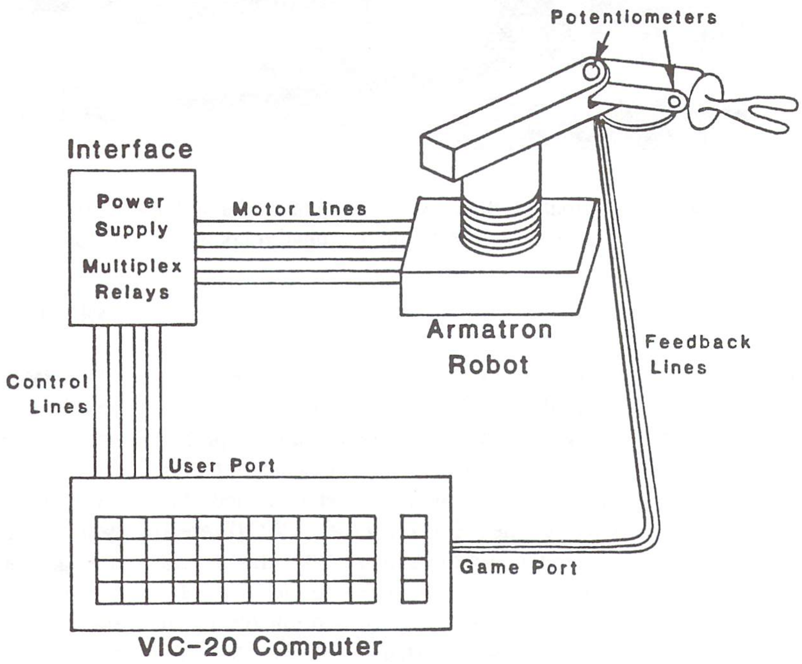


Figure 3. Schematic layout of microcomputer, interface and robot.

www.Commodore.ca
May Not Reprint Without Permission

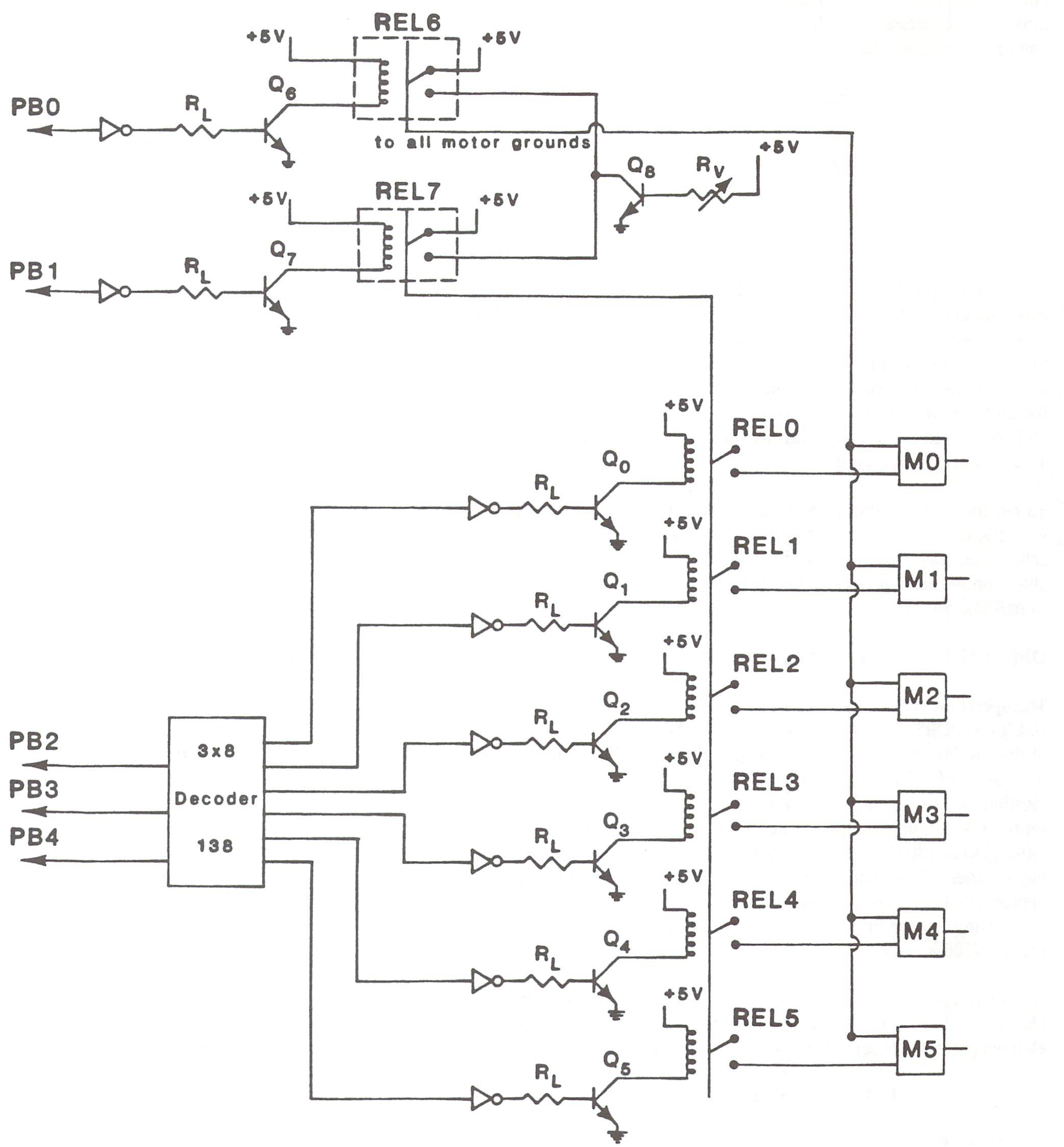


Figure 4. VIC-20 to robot arm interface and controller.

The interface circuit is fairly straightforward and repetitious. It can perform essentially three major functions: (1) manual control of the motor speed, (2) selection of on/off and forward/reverse for all motors, and (3) individual motor selection.

The first function, motor speed control, is regulated by the variable resistor, Rv. The resistor controls the current injected into the base of the transistor (Q_8), which in turn regulates the amount of current passing from the collector to the emitter and through the motor.

The second function, motor direction and switch, is controlled by the two lines PB0 and PB1 on the user port of the VIC-20. When both relays are off (00), or both are on (11), the relays switch between +5 volts and ground, respectively. Thus, when a '00' or '11' is sent to these lines, the motors are tied to the same potential and no current flows; the motors are OFF. If a '10' or a '01' is sent, one relay is tied to +5 volts and the other to ground, thus current may flow to a motor. Going from a '01' to '10' reverses the direction of the motor. The inverters on the input lines are used as line drivers to protect the VIC-20.

www.Commodore.ca
May Not Reprint Without Permission

Finally, the third, and most important function is the motor selection. The motors are addressed with lines PB2, PB3 and PB4 where the following bit patterns represent a distinct motor:

PB4	PB3	PB2	Motor No.
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5

The decoder pulls the selected line low and sets all other lines high; thus inverters are used for each line to reverse this bit pattern (NAND gates were used because of availability of chip). When a line goes high, current flows into the base of the transistor causing it to go into saturation and allowing current to flow from the collection to the emitter. This current closes the relay contact and the motor is switched ON. The transistors used in this function act as switches and are needed to drive the relay. Rv is used as a current limiter protecting the TTL circuitry of the inverters.

To cut the cost of batteries and allow us operation of the motors at various voltages, we used a regulated 5 volt power supply which we mounted in a steel cabinet together with the relay board. The total cost of the power supply, relays, chips connectors and cables was in the order of $50. The 5 volt DC motors were from Radio Shack.

OPERATION OF THE INTERFACE

The operation of this controller is accomplished by POKEing bit patterns to the USER I/O PORT. Memory location 37138 is the Data Direction Register (DDR) of the VIC-20 and controls input/output of Port B. First, one must make the lines used, PB0-PB4, output lines. This is accomplished by writing to memory location 37138 a "bit" pattern where a 1 in the respected line position represents an output line. To make PB0-PB4 outputs, we must send a XX11111 (binary) to 37138 (X = don't care) thus a POKE 37138,31 makes PB0-PB4 all output lines. These lines can now be set high (1) or low (0) by writing the appropriate bit patterns to memory location 373136, which is the actual port B itself. The required bit pattern is shown in Table 1. The BASIC command is POKE 37136,X.

A small program which tests each of the motors in both directions is shown in Listing 1. The motors are controlled by typing the first letter of the robot arm element (i.e. B for base) and the direction (i.e. R for CCW, L for CW).

Table 1: Required Bit Patterns to Operate Motors

PB4	PB3	PB2	PB1	PB0	X	Motor No.	Motor Action
0	0	0	0	1	1	0	Base - rotate CCW*
0	0	0	1	0	2	0	Base - rotate CW**
0	0	1	0	1	5	1	Elbow - rotate CCW
0	0	1	1	0	6	1	Elbow - rotate CW
0	1	0	0	1	9	2	Shoulder - up
0	1	0	1	0	10	2	Shoulder - down
0	1	1	0	1	13	3	Wrist - CW
0	1	1	1	0	14	3	Wrist - CCW
1	0	0	0	1	17	4	Hand - close
1	0	0	1	0	18	4	Hand - open
1	0	1	0	1	21	5	Wrist - up
1	0	1	1	0	22	5	Wrist - down

*CCW - counterclockwise **CW - clockwise

In any robot operation, feedback on the position of the robot arm is essential. These are only two convenient places where a simple potentiometer can determine the position of an element of the robot, namely at the wrist and at the elbow. Therefore, only the movements of the wrist and the elbow are fed back into the VIC-20 (actually, the VIC has only two analog inputs). Two 200 K potentiometers were attached to the wrist and elbow with the wiper arm locked to the elbow and shoulder, respectively. The elbow potentiometer was connected to pin 9 (POT X) and the wrist potentiometer was connected to pin 5 (POT Y) of the Game I/O port. Potentiometer ground was carried to pin 8. A simple PEEK in BASIC will then tell the approximate position of the wrist or elbow.

		PEEK Values and Position
Elbow:	PEEK (36872)	5 far left
		38 centre
		62 far right
Wrist:	PEEK (36873)	120 down
		72 centre
		13 up

A SIMPLE PROGRAM

To demonstrate a simple movement of the robot, the example program in Listing 2 will cause the robot arm to grab an object, lift it over a barrier, rotate it for theatrical effects, and place it down on the other side of the barrier and release it. After a 15 second rest, the robot will pick up the object again and return it to its previous position.

SOME LIMITATIONS

The attachment of the motors to the gears is not as precise as we wished to be. Some motor-gear slippage takes place. Occasionally a motor will jam or will not be pressing hard enough against the gears to drive them. Therefore some adjustments will be needed from time to time. It is also desirable to run the motors at low speed to make them and the gears last a long time.

Since we have feedback on only two movements—the elbow and the wrist—the robot arm must always be put into a known initial position. The robot will return to approximately the same position—not exactly, since there is some play in the plastic gears and linkages.

CONCLUSIONS

The ARMATRON toy robot together with a VIC-20 computer allows a demonstration of robotics at a very

www.Commodore.ca
May Not Reprint Without Permission

low cost. The movements of the robot are not precise enough for a real world application, but are good enough for demonstration and teaching purposes. The mystique of programming and control of a robot is thus simply shown and appreciated by students of all ages.

REFERENCES

1. HERO-1. Manufactured by Heath Company, Benton Harbor, MI 49022

2. RHINO. Manufactured by Rhino Robots, Inc. 2505 S. Neil St., Champaign, IL 61820.

3. ARMATRON. Imported by Tomy Corp. 901 E. 233rd Street, P.O. Box 6252, Carson, California 90749.

4. VIC-20. Manufactured by Commodore Business Machines, Inc., Wayne, PA 19087

Listing 1: Simple test program for robot arm motors

```
NE  1 rem manual control of robot motors
IC  2 rem rolf a deininger july 1983
AE  10 poke 37138,15 :rem all lines output
AB  20 poke 37136,0   :rem turn all motors off
LF  30 dim cs$(13),cn(13)
PB  40 for i=1 to 13:read cn(i):next i
LE  50 data 2,1,6,5,9,10,13,14,17,18,21,22,0
OO  60 cs$="brblerelsusdwrwlhchowuwdst"
MM  70 print"robot motor control":print
AP  80 print "command ";
DP  90 input cm$:cm$=left$(cm$,2)
CO  100 if cm$="en" then poke 37136,0:end
LO  110 for i=1 to 13
AH  120 ifcm$<>mid$(cs$,i*2-1,2) then 130
DJ  125 poke 37136,cn(i):print "   ":go to 80
BG  130 next i
EO  140 print"unknown command"
IC  150 goto 80
```

Listing 2: Example program to lift an object, move it, and return it to approximately the same place.

```
AH  5  rem demonstration program for robot arm
JP  10 rem kevin o'connor april 1983
CE  15 p=37136                :rem port address
OM  20 poke 37138,255         :rem make all lines output
PA  30 poke p,9               :rem shoulder up
JM  40 for i=1 to 15000: next i
OI  50 poke p,21              :rem wrist up
BD  60 x=peek(36873)          :rem feedback for wrist
PP  70 if x<>23 then goto 60
LB  80 poke p,13              :rem spin wrist
HO  90 for i=1 to 10000: next i
PH  100 poke p,2              :rem rotate base cw
PA  110 for i=1 to 15000: next i
LL  120 poke p,6              :rem elbow cw
FD  130 x=peek(36872)         :rem feedback for elbow
HB  140 if x<>48 then goto 130
JH  150 poke p,22             :rem wrist down
GJ  160 x=peek(36873)
BE  170 if x<>77 then goto 160
LC  180 poke p,10             :rem shoulder down
LF  190 for i=1 to 14000: next i
LN  200 poke p,18             :rem open hand
ON  210 for i=1 to 5000: next i
JA  220 poke p,0              :rem off
OL  230 ti$ = "000000"        : rem 15 second wait
BB  240 if ti$<>"000015" then goto 240
JH  250 poke p,17             :rem close hand
AB  260 for i=1 to 5000: next i
PP  270 poke p,9              :rem shoulder up
EL  275 for i=1 to 15000: next i
EH  280 poke p,21             :rem wrist up
IB  290 x=peek(36873)
DL  300 if x<>12 then goto 290
IK  310 poke p,14             :rem wrist ccw
NM  320 for i=1 to 10000: next i
FC  330 poke p,1              :rem base ccw
FP  340 for i=1 to 15000: next i
NH  350 poke p,5              :rem elbow ccw
MF  360 x=peek(36872)
JA  370 if x<>29 then goto 360
PF  380 poke p,22             :rem wrist down
MH  390 x=peek(36873)
ED  400 if x<>77 then goto 390
BB  410 poke p,10             :rem shoulder down
BE  420 for i=1 to 14000: next i
MK  430 poke p,0
IL  440 end
```

www.Commodore.ca
May Not Reprint Without Permission

Low Cost Universal EPROM Programmer

Tim Bolbach, P.Eng.
Toledo, Ohio

Overview

It seems that too often when a computer is used as the control device in an interface project it involves expensive, rare, or large numbers of integrated circuits. Then this is usually supported by a minimum amount of software. The design detailed in the next few pages represents what I feel is a good marriage of hardware and software. The idea for this peripheral came from my need of an inexpensive EPROM programmer to assist in the building of small microprocessor control boards and firmware add-ons for the C64. The system had to be reliable and easy to use. The software had to be capable of copying an EPROM, as well as programming from a manually entered program file. The programmer must also program many different types of EPROM chips. This design is the result of many hours of experimentation.

The programming of an EPROM requires that the system provide a stable address, stable data input, a programming voltage (12.5v – 25v dependent on the EPROM used), and a programming pulse of 50 ms duration. Various other control signals are required by different EPROMs, such as chip enable, output enable, program enable and combinations of the above. Therefore, to make this device universal it had to generate all of the different control signals.

Note: Extreme care must be taken when building any device that connects directly to the expansion port. A small wiring error can cause extensive damage to the computer. It is suggested that a careful check with an ohmmeter be completed before plugging in the programmer.

Hardware

To generate the different signals the circuit uses two Intel 8255 programmable parallel interface chips. These were chosen over 6522's or 6526's mainly from a cost standpoint. From my local supplier (JDR Microdevices) the 8522's represent a 2.5 to 1 savings over the 6522's and a 18 to 1 savings over the 6526's. Not to mention, the 8522's are readily available from many different suppliers and suit the application well. The only other integrated circuit required is a 7400 to select the PIO's.

The universal part of the design comes in with the use of a 24 pin socket and header as a 'personality' module. This allows customizing the pinout of the programming socket for many different types of EPROMs. If the programmer is to be used for only one type of EPROM or family of EPROMs, then the 'personality' socket can be eliminated. Some header pinouts for popular EPROMs are given in this article but are not the only arrangements that can be used.

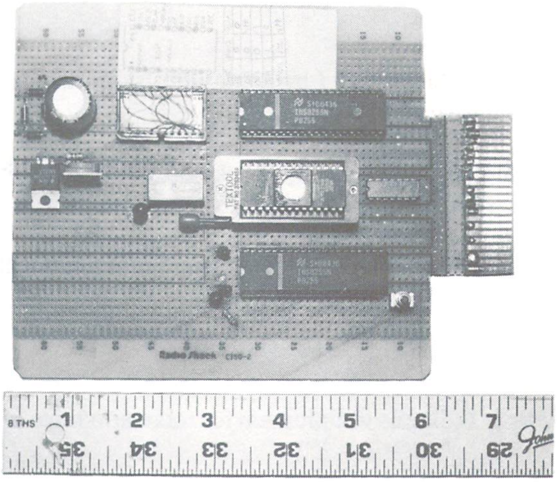


Figure 1

The programmer requires the proper voltage to program the chips. Most popular EPROMs use 25 volts but some like the 2732A use 21 volts. This voltage can be supplied by several batteries with a zener regulator or an AC powered transformer rectifier regulator circuit (see figure 2). The cost of the programmer is affected by the method chosen. I have even used 5 volt to 25 volt converter boards for the supply. This is the easiest method but can be expensive. I used a relay to turn the programming supply on and off. With a little careful circuit design it could be eliminated and a MOSFET switching circuit used. The relay was used for simplicity in the prototype.

Point to point wiring was used on the prototype. Sockets were used to protect the chips. This does increase the cost, but the added protection well outweighs the cost. Wire wrapping is another possible method as the layout is not critical. Care must be taken to keep address leads and data leads as short as possible to prevent radiating RFI. A 28 pin zero insertion force (ZIF) socket is used for holding the EPROM while programming. For 24 pin devices the EPROM is inserted in the rear of the socket. This type of socket prevents damage caused by inserting and removing the EPROM. The transistors shown in the schematic are general purpose NPN switching transistors. They must be rated for collector currents of 150 mA or more. A complete parts list is part of the schematic drawing.

An attempt was made to use as much of the decoded signals that the 64 supplies to keep hardware costs down. Commodore was thoughtful in their planning to leave two I/O pages decoded and ready for interfacing. The programmer uses both the decoded addresses of \$DE00 and \$DF00 for selecting the PIO's. These addresses were reserved for future I/O expansions and help eliminate extra decoding hardware. One problem that this creates is that some firmware cartridges (such as FASTLOAD and SIMON'S

www.Commodore.ca
May Not Reprint Without Permission

BASIC) use these addresses to turn themselves on and off. Since the programmer need be the only device plugged into the expansion port, this should cause no problem.

The 8255's are like the 6522's in that they are programmable, but this is where the similarity ends. The 8255 requires that a control word be written to the control register to configure the entire three ports at one time. Ports A, B and C on the PIO #1 are configured as output ports at all times. These are the low and high address bits and control buss signals to the EPROM. Port B of PIO #2 is the data buss port. During reading of the EPROM it is configured as an input port, but, during programming it must supply a stable data buss input signal to the EPROM and is configured as an output port. The versatile control register allows us to accomplish this with no problem. Refer to the manufacturer's spec sheets on the 8255 for more details on configurations. The chart below gives the addresses for the different control and data ports of the 8255's for the programmer.

PIO #1

$DE00	56832	PORT A DATA	EPROM LOW ADDRESS BYTE
$DE01	56833	PORT B DATA	EPROM HI ADDRESS BYTE
$DE02	56834	PORT C DATA	EPROM CONTROL SIGNALS
$DE03	56835	8255 CONTROL	

PIO #2

$DF00	57088	PORT A DATA	(NOT USED)
$DF01	57089	PORT B DATA	EPROM DATA
$DF02	57090	PORT C DATA	(NOT USED)
$DF03	57091	8255 CONTROL	

Port C of PIO #1 needs some explanation. Bit PC0 is used to turn on the programming supply during programming. Bit PC1 is a '1' during standby but a '0' during reading or programming pulses. Bit PC2 is a '1' during standby, a '0' during reading pulses and '1' during programming. Bit PC3 is a '1' during standby, a '1' during reading and is a '0' during programming pulses. These signals comprise all of the combinations of signals required by most EPROMs for reading or programming. A special signal which uses PC1 and the relay supplies a '1' during standby, a '0' during read pulses and connects the programming voltage (usually 25 volts) to this EPROM pin. This is referred to as OE/VPP on the spec sheets for the 2732 EPROMs. A chart of states for the different control signals is shown in figure 3. These signals plus the 'personality' socket feature allows configuring the programming socket for many applications. I even use the programmer to read masked roms to verify that they are functional.

Personality Socket Terminal	Signal	Read	Standby	Program
24	CE	0	1	0
23	OE1	0	1	VPP
22	OE3	0	1	1
21	PGM	1	1	0
20	VPP/VCC	VCC	VCC	VPP

Figure 3

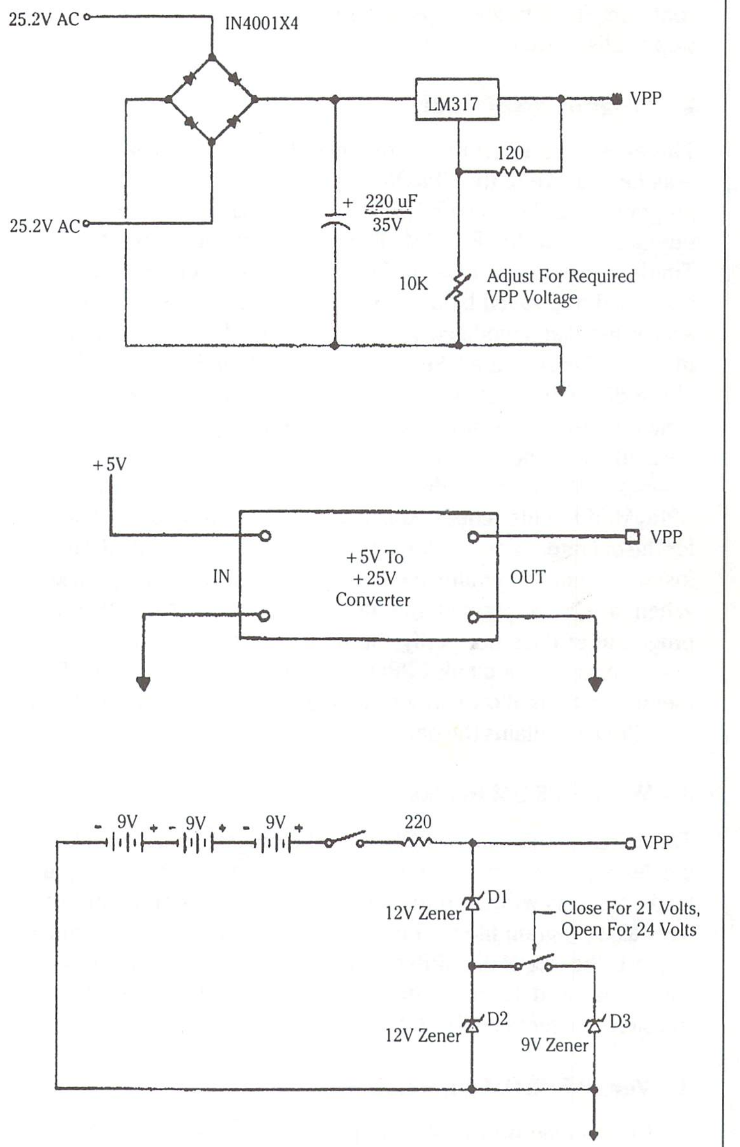


Figure 2

Software

The program supplied was written and intended to be user friendly. The use of menu screens and prompts makes the program straight forward and easy to use. Basic makes the program easily understood so that modifications and customizing is possible. Because it is in the nature of BASIC to be slow, programming an EPROM can take up to 3 minutes per 1024 bytes (1k). I use a compiled version of the program to speed things up. But speed of programming an EPROM should not be a factor unless you are mass programming.

The menu screen provides seven options which are discussed below.

1 – List an EPROM

This function lists the data stored in the EPROM. It is a good check to see if the EPROM was programmed. The address starting at $0000 is displayed along with the data data in HEX format. To

www.Commodore.ca
May Not Reprint Without Permission

pause the listing press and hold the SHIFT key. The listing continues indefinitely so pressing the Commodore logo key will stop the listing and return to menu.

2 – Program an EPROM

This allows the actual programming of the EPROM. The program asks for the size of the EPROM to adjust the loop parameters for programming. The next request is for the name of the file to be programmed on the EPROM. The file is stored as a program file. This is done for a few reasons. First, since it is a program file, it can be loaded and saved by a machine code monitor as such. It can also be listed and modified using the memory dump features of the monitor. The software takes care of eliminating the first two bytes of the disk file which are used as a pointer to the memory load location for file. This also allows you to create your own program file manually to be programmed on an EPROM. PAL can be used to assemble PAL source code as a program file to be written on an EPROM. If the file requested is found, the the programmer waits for the operator to press a key. At this time a blank EPROM can be inserted if one is not already there into the programming socket. When a key is pressed the rest is automatic. Note that the programmer does not verify the data during programming, nor does it check for a blank EPROM before trying to program. Two menu selections allow verifying that an EPROM is erased and that the EPROM contains the data from a particular file.

3 – Write EPROM to disk

This function reads an EPROM from the programming socket and creates a program file on the disk. A dummy two-byte program load pointer is written to the file first to allow this program to be used as a program file by a machine code monitor. The program requests the size of the EPROM and the name of the new file. This feature is used to copy an EPROM or rom to a disk file to transferred later to an EPROM.

4 – Verify EPROM with disk

As it was stated previously, the programmer does not verify the data on the EPROM at the time of programming. This part of the program reads the disk file and verifies it with data on the EPROM. The size of the file is requested first then the name of the file to verify against the EPROM. If an error is found, the option terminates and indicates at what memory location the error was found.

5 – Check for erasure

This option does exactly as it says. A blank EPROM is inserted into the programming socket, then the size of the EPROM is requested and the option begins. Each memory location of the EPROM is checked for a blank word (255 or $FF). Eproms contain all '1's' in the blank state. If a location is found not erased the option terminates and returns to the menu.

6 – Directory

This option lists the directory of the disk on the screen.

7 – List disk file to screen

Option 7 reads a disk file and displays the file in HEX on the screen. It was included to verify that the file created by writing an EPROM to the disk was indeed written.

Figure 4 shows a simple EPROM eraser. This device uses an ultra violet light bulb used in electric dryers years ago. The bulb is still available at appliance part supply houses. Erasing time is approximately 20 minutes. Over erasing can sometimes cause damage to an EPROM so use a timer or clock to time the exposure.

Caution: Do not look at the ultra violet light when it is on. Ultra violet light can cause damage to the eyes. Turn the light on only after making sure that no ultra violet light will escape.

EPROM Programmer Software

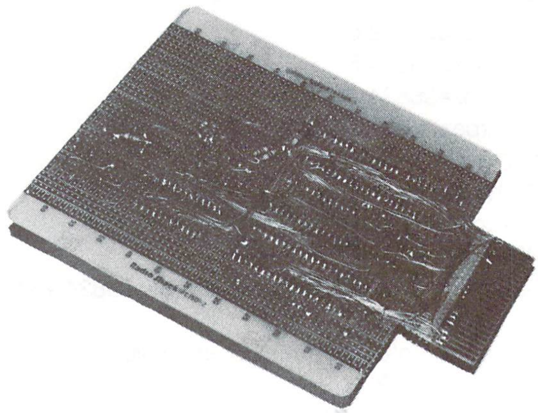
```
AF  1000 rem** program by tim bolbach / graphics
         by rich bozman **
AF  1010 poke56835,128:poke56834,254:poke57091,255
         :poke53281,11:poke53280,0
OG  1020 rv$=chr$(18): sp$=rv$+ "[28 spcs]"
FP  1030 co=36:li=24:gosub3100:print chr$(147)
         chr$(144)
JK  1040 fort=1 to 21
KB  1050 next t
FC  1060 poke2020,195
BD  1070 poke2021,195
ND  1080 poke2022,195
FE  1090 poke2023,253
ON  1100 print"Sqqqq"spc(9)"eprom handler menuq"
NG  1110 print"      1 list eprom on screen"
DM  1120 print"      2 program eprom"
JD  1130 print"      3 write eprom to disk"
BE  1140 print"      4 verify eprom with disk"
MO  1150 print"      5 check for erasure"
FJ  1160 print"      6 directory"
HC  1170 print"      7 list disk file to screenq"
HL  1180 print"[6 spcs]r logo R = commodore key"
LB  1190 poke198,0:wait198,1:geta$
CI  1200 a=val(a$):ifa<1ora>7then1190
MO  1210 onagoto1220,1520,1820,2100,2450,2680,2920
IO  1220 rem *** read eprom ***
PP  1230 c$="0123456789abcdef"
PL  1240 print"Sq"spc(8)"rlist eprom on screen"
HO  1250 print"qshift=rpauseR  shift lock=rholdR
           logo=rmenuqqq"
OC  1260 a=0:poke56835,128:poke57091,255
         :rem**** set ports for read ******
NL  1270 poke56834,254
HF  1280 ad=a
LB  1290 gosub 1440
MB  1300 print rv$;"$";a$;":R ";
BA  1310 for t=0 to 7
NH  1320 ah=int((ad+t)/256):al=(ad+t)-(ah*256)
BC  1330 poke56832,al :poke56833,ah
DA  1340 poke 56834,8
IA  1350 d=peek(57089)
HB  1360 poke56834,254
AH  1370 gosub 1490
DM  1380 print d$;"  ";
OG  1390 next t
```

www.Commodore.ca
May Not Reprint Without Permission

```
KK 1400 print
OB 1410 on peek(653) goto 1410, 1010
CE 1420 a=a+8
LJ 1430 goto 1280
HK 1440 d1=int(a/4096):x=a-(d1*4096)
CN 1450 d2=int(x/256):x=x-(d2*256)
HG 1460 d3=int(x/16):d4=x-(d3*16)
KA 1470 a$=mid$(c$,d1+1,1)+mid$(c$,d2+1,1)
        +mid$(c$,d3+1,1)+mid$(c$,d4+1,1)
EO 1480 return
HO 1490 d1=int(d/16):d2=d-(d1*16)
JA 1500 d$=mid$(c$,d1+1,1)+mid$(c$,d2+1,1)
CA 1510 return
CH 1520 rem *** burn eprom ***
FN 1530 print" S ";:poke56835,128:poke56834,254
        :poke57091,128
HF 1540 print spc(13)" qr burn eprom"
HB 1550 gosub 3120
ON 1560 x=((2↑a)*1024)-1
DI 1570 input" q file name";n$
AI 1580 open8,8,8,n$+",p,r"
EP 1590 open15,8,15
LK 1600 get#15,a$,b$
ID 1610 er=val(a$+b$)
BK 1620 print" q ";a$+b$;
HO 1630 get#15,a$:printa$;:ifa$=chr$(13)then1650
IH 1640 goto1630
NJ 1650 ifer>0thenclose8:close15: fort=1 to 1000:next t
        :goto1010
ID 1660 printspc(7)" qqr press key when ready"
        :poke198,0:wait198,1
JF 1670 get#8,a$:get#8,a$:rem ** get rid of file
        address ***
FB 1680 poke56834,255:fort=1 to 1000:nextt
CK 1690 printspc(12)" q location: ":printspc(13)
        " qq logo  =  r abort"
EI 1700 forc=0 to x:li=19:co=21:gosub3100:
CC 1710 printleft$(sp$,7-len(str$(c)))c
MP 1720 get#8,d$:ifd$=""thend$=chr$(0)
ID 1730 d=asc(d$)
BG 1740 ah=int(c/256):al=c-(ah*256)
FM 1750 poke56832,al:poke56833,ah
IM 1760 poke57089,d
OK 1770 poke56834,5
OL 1780 poke56834,255
AF 1790 ifpeek(653)=2thenc=x
FN 1800 next c
FF 1810 poke56834,254:poke57091,255:close8
        :close15:goto1010
JC 1820 rem *** write eprom to disk ***
PP 1830 print" S ";:poke56835,128:poke56834,254
        :poke57091,255
LI 1840 print spc(9)" qr write eprom to disk"
DE 1850 gosub 3120
KA 1860 x=((2↑a)*1024)-1
PK 1870 input" q file name";n$
AM 1880 open8,8,8,n$+",p,w"
AC 1890 open15,8,15
HN 1900 get#15,a$,b$
KD 1910 er=val(a$+b$):printa$;b$;
MA 1920 get#15,a$:printa$;:ifa$=chr$(13)then1940
FK 1930 goto1920
PL 1940 ifer>0thenclose8:close15: fort=1 to 1000:next t
        :goto1010
KF 1950 printspc(7)" qqr press key when ready"
        :poke198,0:wait198,1
AP 1960 poke56834,254:fort=1 to 400:nextt
KL 1970 printspc(12)" q location: ":printspc(13)
        " qq logo  =  r abort"
CH 1980 print#8,chr$(0);:print#8,chr$(0); :rem ** put in
        fake file address **
FK 1990 forc=0 to x:li=18:co=21:gosub3100:
EE 2000 printleft$(sp$,7-len(str$(c)))c
PG 2010 ah=int(c/256):al=c-(ah*256)
DN 2020 poke56832,al:poke56833,ah
FL 2030 poke56834,8
KL 2040 d=peek(57089)
JM 2050 poke56834,254
OE 2060 d$=chr$(d):print#8,d$;
IG 2070 ifpeek(653)=2thenc=x
NO 2080 nextc
NG 2090 poke56834,254:poke57091,255:close8:close15
        :goto1010
KG 2100 rem *** verify eprom with disk ***
HB 2110 print" S ";:poke56835,128:poke56834,254
        :poke57091,255
GH 2120 print" Sq "spc(7)" r verify eprom with disk"
LF 2130 gosub 3120
CC 2140 x=((2↑a)*1024)-1
HM 2150 input" q file name";n$
EM 2160 open8,8,8,n$+",p,r"
ID 2170 open15,8,15
PO 2180 get#15,a$,b$
CF 2190 er=val(a$+b$):printa$;b$;
BB 2200 get#15,a$:printa$;:ifa$=chr$(13)then2220
CK 2210 goto2200
HN 2220 ifer>0thenclose8:close15: fort=1 to 1000:next t
        :goto1010
CH 2230 printspc(7)" qqr press key when ready"
        :poke198,0:wait198,1
IA 2240 poke56834,254:fort=1 to 400:nextt
CN 2250 printspc(12)" q location: ":printspc(13)
        " qq logo  =  r abort"
AI 2260 get#8,a$:get#8,a$:rem ** get fake file address
        out of the way*
BP 2270 forc=0 to x
KH 2280 li=18:co=21:gosub3100:
GG 2290 printleft$(sp$,7-len(str$(c)))c
BJ 2300 ah=int(c/256):al=c-(ah*256)
FP 2310 poke56832,al:poke56833,ah
HN 2320 poke56834,8
GE 2330 d=peek(57089):print"[2 spcs]"
        left$(sp$,5-len(str$(d)))
LO 2340 poke56834,254
KF 2350 get#8,a$:ifa$=""thena$=chr$(0)
BK 2360 a=asc(a$):ifa<>d then 2430
EJ 2370 ifpeek(653)=2thenc=x
JB 2380 next c
```

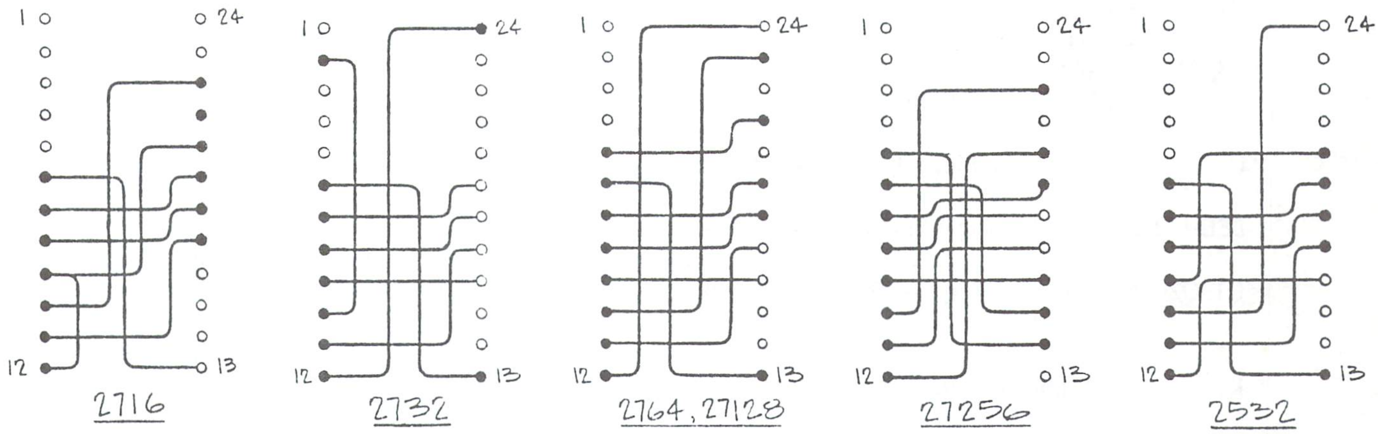
www.Commodore.ca
May Not Reprint Without Permission

```
JJ  2390 poke56834,254:poke57091,255:close8:close15
         :goto1010
BJ  2400 print" Sqqqr eprom program verified R "
DG  2410 printspc(4)" qr press any key to continue"
         :poke198,0:wait198,1
MD  2420 close8:close15:goto1010
FD  2430 print" Sqqqr [6 spcs]!!! error
         found !!![11 spcs] ";
NC  2440 printspc(4)" qr the error is at location : R "
         c:goto2410
CD  2450 rem *** check for erasure ***
EG  2460 print" Sq "spc(13)" r check erasure"
FN  2470 a=0:poke56835,128:poke57091,255:rem* set
         ports for read *
HH  2480 poke56834,254
DM  2490 gosub 3120
LI  2500 poke56834,254
EJ  2510 x=((2↑a)*1024)-1
IJ  2520 printspc(8)" qqr press key when ready"
         :poke198,0:wait198,1
KO  2530 printspc(12)" q location: ":printspc(13)
         " qq logo = r abort"
PP  2540 forc=0 to x
EB  2550 if peek(653)=2 thenc=x:goto2640
MI  2560 li=15:co=21:gosub3100:
OH  2570 printleft$(sp$,7-len(str$(c)))c
JK  2580 ah=int(c/256):al=c-(ah*256)
NA  2590 poke56832,al:poke56833,ah
PO  2600 poke56834,8
EP  2610 d=peek(57089)
DA  2620 poke56834,254
BD  2630 if d<>255 then 2660
MM  2640 next c:ifd<>255then2660
JA  2650 printspc(8)" qq reprom erased":goto2670
KH  2660 printspc(7)" qqr eprom is not erased !!"
JK  2670 printspc(9)" qr press key for menu"
         :poke198,0:wait198,1:goto1010
IO  2680 rem *** directory ***
LD  2690 print" Sq "spc(12)" r disk directory"
HE  2700 print" q [6 spcs]shift = r pause R "
         spc(9)" logo = r menu qR "
JG  2710 open1,8,0,"$0"
DG  2720 get#1,a$,b$
NG  2730 get#1,a$,b$
HH  2740 get#1,a$,b$
IJ  2750 c=0:u$=""
BM  2760 if a$<>'' then c=asc(a$+chr$(0))
OJ  2770 if b$<>'' then c=c+asc(b$+chr$(0))*256
FH  2780 printright$(sp$,9-len(str$(c)))c"  ";
KF  2790 get#1,b$:if st<>0 then 2890
DC  2800 if b$<>chr$(34) then 2790
KO  2810 get#1,b$:if b$<>chr$(34)thenu$=u$+b$
         :b$="":goto2810
BL  2820 get#1,b$:if b$=chr$(32) then2820
DM  2830 printchr$(34)u$right$(sp$,16-len(u$))chr$(34)
         "  ";:c$=""
FK  2840 c$=c$+b$:get#1,b$:if b$<>" " then2840
AF  2850 printleft$(c$,3)
KA  2860 ifpeek(653)=1then2860
AM  2870 ifpeek(653)=2thenclose1:goto1010
LJ  2880 if st=0 then 2730
LD  2890 print" blocks free"
LI  2900 printspc(11)" qqr press key for menu"
         :poke198,0:wait198,1
II  2910 close1:goto 1010
KJ  2920 rem **** display disk file ******
BE  2930 print" Sq "spc(9)" r display disk file"
FD  2940 print" q [4 spcs]shift = r pause R "
         spc(9)" logo = r menu qq "
BM  2950 h$="0123456789abcdef"
GB  2960 input" qr file name R ";n$
OO  2970 open8,8,8,n$+",p,r"
JP  2980 get#8,a$:get#8,a$
DP  2990 for t= 0 to 8191
FJ  3000 for r= 0 to 7
OO  3010 get#8,a$:ifa$=""thena$=chr$(0)
MD  3020 d=asc(a$)
HI  3030 q=int(d/16):w=d-(q*16)
GI  3040 d$=mid$(h$,q+1,1)+mid$(h$,w+1,1)
JE  3050 printd$;"    ";
OO  3060 next r
PI  3070 if peek (653)=2 then close8:goto1010
GO  3080 if peek (653)=1 then 3080
OE  3090 print:next t
GC  3100 poke211,co:poke214,li:sys58732:return
KJ  3110 :
PA  3120 print" qr select eprom size q "
EM  3130 print"    1   2k"
EN  3140 print"    2   4k"
GO  3150 print"    3   8k"
GM  3160 print"    4  16k"
OM  3170 print"    5  32k"
GC  3180 poke198,0:wait198,1
KB  3190 geta$:a=val(a$):ifa<1ora>5then3180
MJ  3200 return
```



www.Commodore.ca
May Not Reprint Without Permission

Personality Socket Wiring



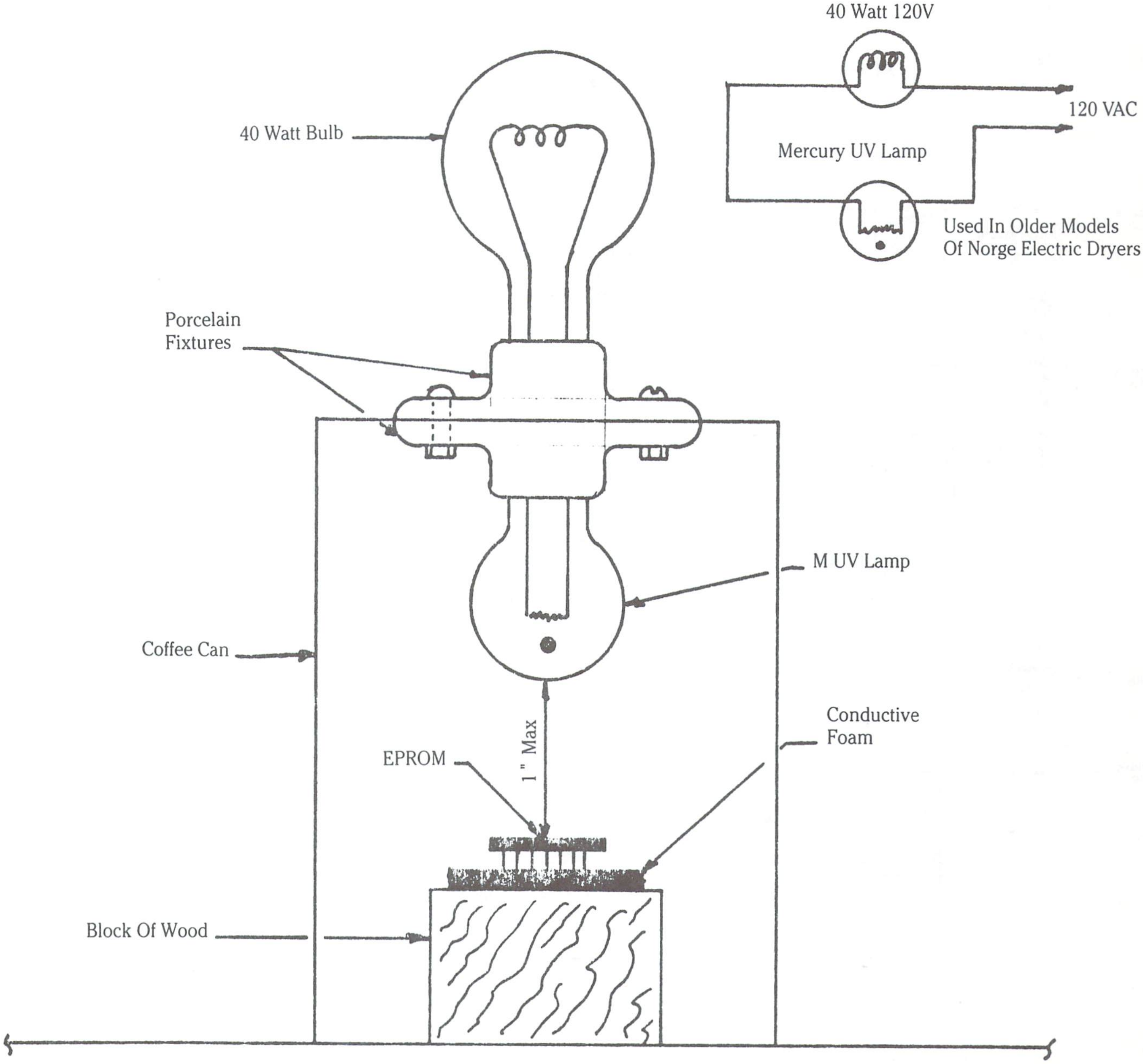


Figure 4: EPROM Eraser

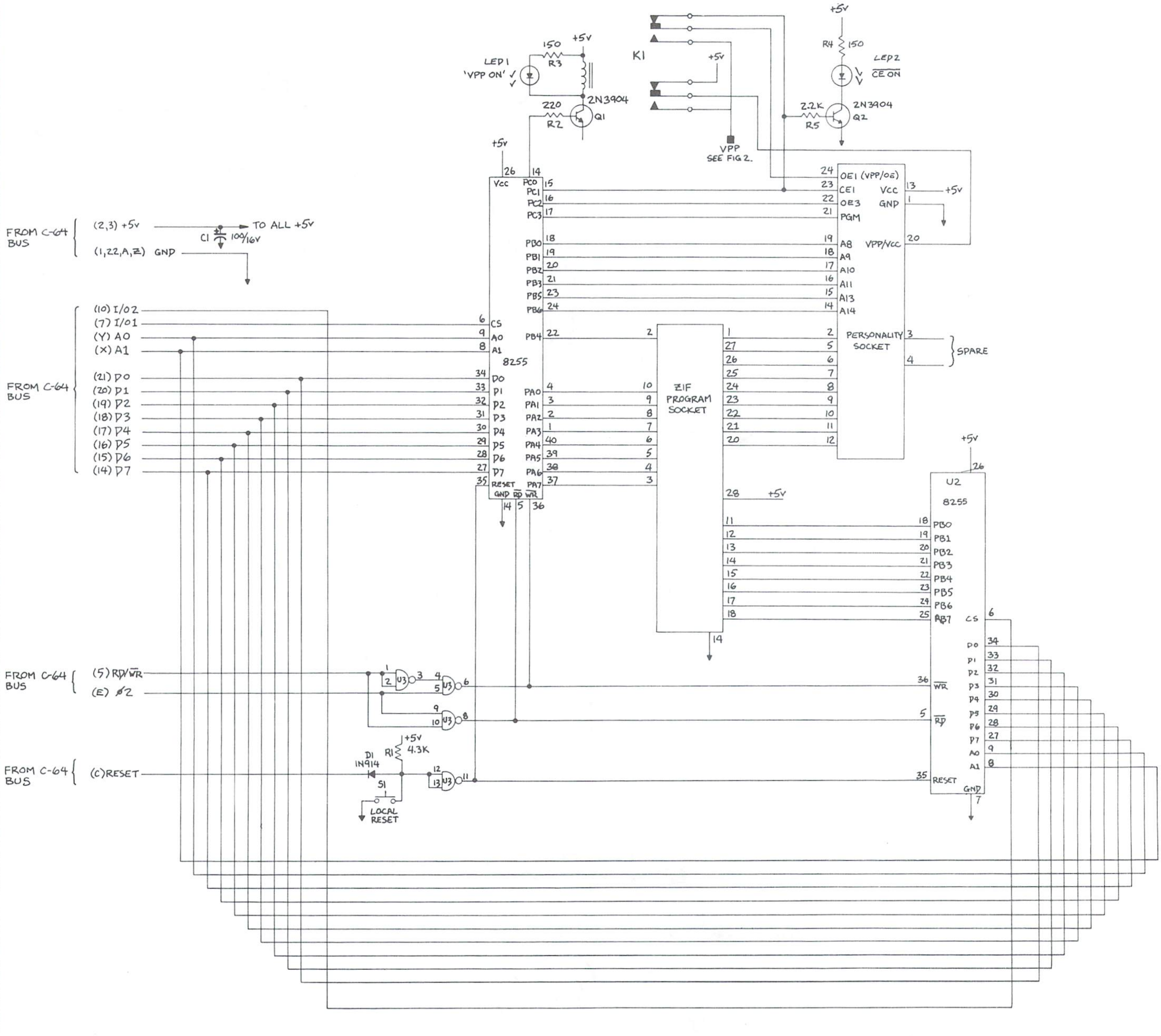
www.Commodore.ca
May Not Reprint Without Permission

Parts List

R1	4.3K 1/4 Watt Carbon Resistor
R2	220 ohm 1/4 Watt Carbon Resitor
R3, R4	150 ohm 1/4 Watt Carbon Resitor
R5	2.2K 1/4 Watt Carbon Resistor
C1	100 uF 16 Volt Electrolytic Capacitor
LED 1, 2	Standard Red Light Emitting Diode
Q1, Q2	NPN General Purpose Transistor 2N3905, or equivalent
K1	DPDT Miniature Relay, 5 Volt Coil
U1, U2	INTEL 8255 PIO
U3	74LS00 Quad NAND Gate
S1	SPST Momentary Pushbutton

Miscellaneous Items

1 – 28 PIN Zero Insertion Force Socket for EPROM
1 – 24 PIN DIP Socket for Personality Socket
2 – 40 PIN DIP Sockets
As Req'd – 24 PIN DIP Header for Personality Plugs



EPROM Programmer Schematic

www.Commodore.ca
May Not Reprint Without Permission

A C64 Cartridge Without EPROMs

John Bush and Noel Nyman
Seattle, Washington

you won't need any expensive programming devices to make your own cartridges for a C64 or C128 with this special technique

Cartridges are convenient and easy to use. Programs on cartridge Load instantly. You can make a cartridge using EPROMs (Erasable Programmable Read–Only Memories) for about $25, if you shop carefully.

But, the EPROMs must be programmed or "burned" using an EPROM burner, which costs about $125. If you make any mistakes, or want to change the programs, you'll need an EPROM eraser, another $40.

The inexpensive EPROM cartridge requires close to $200 in start–up costs.

An alternative is to use RAM (Random Access Memory) in place of EPROMs. RAM can be programmed by the computer itself, and the information can be changed at any time. No additional special equipment is required.

The problem with RAM is that it loses everything in memory when the power is turned off, not exactly what we have in mind for a cartridge. But, by using special CMOS (Complementary Metal Oxide Semiconductor) RAMs that have low stand–by current requirements, we can use a small battery to hold the information in the RAM. The memory is retained even with the computer turned off or when the cartridge is removed. The 4464–15s, made by NEC Corp, used in this project have a typical stand–by current drain of 0.1 micro–amperes. A battery the size of a quarter can power them for several years.

Building The RAM Cartridge

We used a Vector 3795–1 "perf" board for our prototype. It has 44 circuit traces (22 on each side) at the proper spacing to line up with the C64 expansion socket. If you have the equipment to etch your own circuit boards, that may be a less expensive alternative. You may be able to adapt an old cartridge board, or purchase one intended for use in a C64. Be sure that address lines A13 through A15 (pins F, H, and J) are available on the board you use. They aren't needed by EPROM cartridges and may not appear on circuit boards designed for that purpose.

Although we used wire–wrap to build the circuit, any wiring method will work. Sockets are recommended for the integrated circuits, but are not mandatory. Be sure to observe proper precautions when working with the CMOS RAM's. They can be permanently damaged by improper handling.

Figure #1 shows the schematic for an 8K RAM cartridge. Figure #2 has the additional circuitry required to add another 8K. Switch S1 controls the power to the CMOS RAMs. With the switch closed, power comes from the C64. With either S1 open or the computer turned off, the battery takes over and retains the data in memory. S2 controls the READ/WRITE signals to the RAMs. With this switch closed, the computer can change the data. Opening S2 makes the RAMs look like ROM to the C64.

S3 and S4 allow the RAM cartridge to emulate the three types of cartridge used with the C64, which we'll look at shortly. S5 is used only with the 16K version. It allows us to "move" the upper 8K of RAM to an area where it can be programmed. The diodes electrically remove the battery from the circuit when the computer is supplying power and prevents the battery from trying to run the entire C64. The various resistors establish default values for the signal lines and switch the RAMs to their low current stand–by state when S1 is opened.

The 74LS42 is a decoder that monitors the three highest address lines (A13 – A15), and produces a discrete output for each combination of these addresses. There are eight outputs, so we can select eight 8K banks of memory with this chip. Capacitors C1 and C2 are used to remove any noise from the power line. C1 should be placed close to the edge of the board that plugs into the computer. C2 should be mounted as close as possible to the 74LS42.

You may find other 8 x 8K RAMs with similar stand–by current characteristics. If they have 150ns (nano–second) access time or less, they should work for this application. Be sure to get data sheets for them. The pin–outs may be different from those shown on these schematics. See the end of this article for a source for the NEC 4464–15s we used, or check your yellow pages under "Electronic Equipment" for a local NEC distributor.

Parts List

B1	– 3 Volt Circuit Battery (see text)
C1, C2	– 0.05 mfd 12VDC Ceramic Disk Capacitor
D1–D4	– 1N4148 or Similar Small Signal Diode
R1,R3,R4,R5,R7	– 2K 1/4 Watt Resistor
R2,R6	– 22K 1/4 Watt Resistor
S1–S4	– SPST Switches, DIP Arrays Work Well
S5	– SPDT Miniature Switch
74LS42	– 1 of 10 BCD Decoder
4464	– Low Stand–By Current CMOS Static RAM (see text)

www.Commodore.ca
May Not Reprint Without Permission

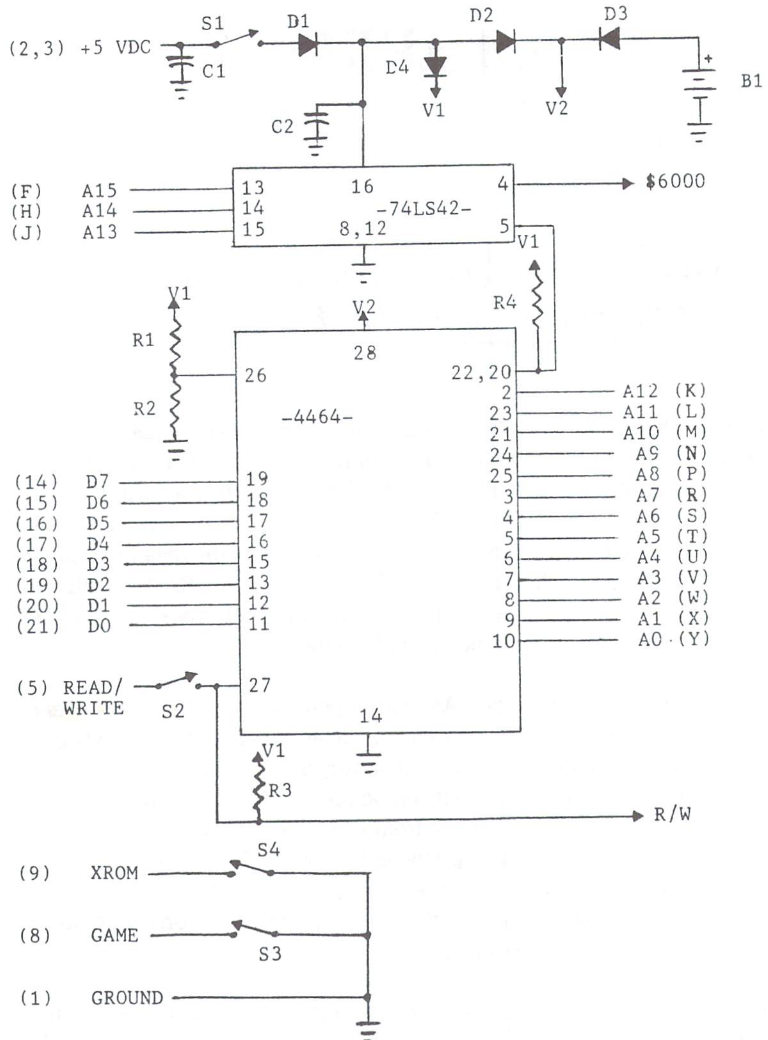


Figure 1: All references in parentheses are pin numbers for the C64 expansion port, see pg.396 of the C64 Programmers Reference Guide.

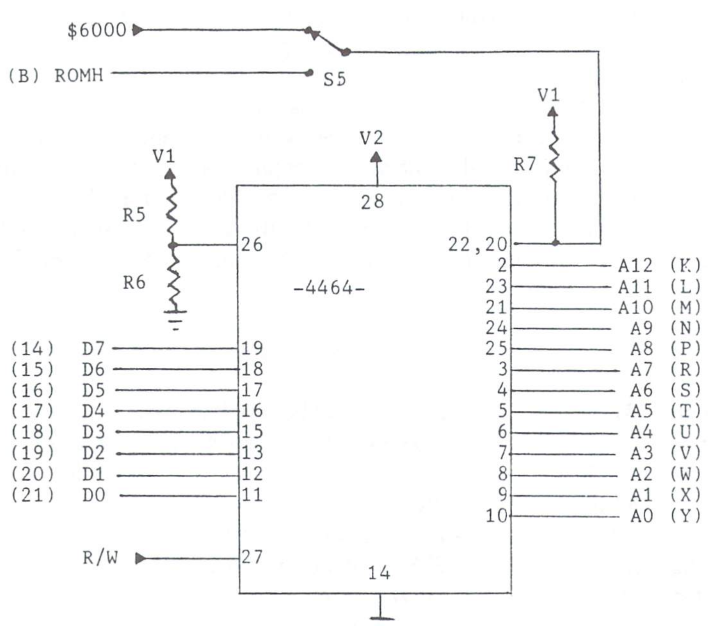


Figure 2: Additional parts required for a 16K cartridge.

How Cartridges Work

The C64 uses a PLA (Programmed Logic Array) to control the access of RAM, ROMs, and cartridges to the address and data buses. For an excellent discussion of how the PLA works, see "Commodore 64 Memory Configurations" by William Levak (Transactor 6–05). Cartridges can have three configurations. The PLA identifies the cartridge by two control lines. These are called "GAME" (pin 8) and "XROM" (pin 9). The RAM cartridge uses switches S3 and S4 to activate the control lines.

An 8K cartridge always appears at address range $8000 – $9FFF. It has an internal jumper that pulls the XROM line low. Closing S4 simulates that configuration. A 16K cartridge also has 8K at $8000 – $9FFF. The upper 8K can reside in one of two other areas. If only the GAME line is low (S3 closed, S4 open), the upper 8K appears at $E000 – $FFFF. If both GAME and XROM are low (S3 and S4 closed), all 16K is contiguous from $8000 – $BFFF.

An 8K cartridge normally contains either a self contained program, or one that uses the BASIC and Kernal ROM routines built into the C64. A 16K cartridge in the $8000 – $BFFF range replaces the BASIC ROM. The upper 8K may contain a modified BASIC, and the lower 8K may have BASIC extensions. The third configuration was intended for games only. Levak's article shows that in this mode, the VIC chip will look for the character set at the upper portion of the $E000 – $FFFF memory. This makes for easier low resolution graphics for games, but is unsuitable as a Kernal replacement. The programs in these cartridges must stand entirely on their own.

All memory chips, RAM or ROM, are switched onto the address and data buses with "chip select" lines. In the C64, the PLA controls these lines, and so decides whether RAM, or one of the system ROMs, or the cartridge is selected. If the PLA senses that a cartridge is in place (through the GAME and XROM lines), and a "READ" command is issued by the microprocessor, the cartridge memory will be selected. The PLA controls this selection through the "ROML" (pin 11) and "ROMH" (pin B) lines. If a "WRITE" command is issued, the PLA switches off the cartridge memory and selects RAM at those addresses instead.

Commodore never intended that cartridges would contain RAM. So the PLA will not write data into our RAM cartridge. To accomplish that, we by-pass the PLA and do our own decoding. Some is done automatically by the 74LS42 chip, and some we control manually with switch S5.

Programming The RAM Cartridge

When the C64 is turned on, reset with an external reset switch, or the "RESTORE" key is pressed, routines in the Kernal ROM look for a cartridge. All cartridges will have 8K starting at location $8000. The Kernal looks for the code "CBM80" starting at address $8004. The high bit of

www.Commodore.ca
May Not Reprint Without Permission

each letter must be set. If the code is there, the normal initialization routines are bypassed, and control is passed to the program in the cartridge. On power-up or hardware reset, the address stored in low-high order at $8000/$8001 is used for an indirect jump. If "RESTORE" has been pressed, the address stored at $8002/$8003 is used instead.

To create an auto-starting program in cartridge, you'll need to install the code phrase and the proper addresses. You may also need to call some of the bypassed initializing routines. You can store machine code in the RAM cartridge without the auto-start phrase and SYS to the code from BASIC or direct mode instead of auto-starting.

If you want to use the RAM cartridge to store a favourite BASIC program, use the program in Listing #1. RUNning the program creates a file called "RAMCART" on disk device #8. You can change those defaults in line 100. The source code of the file is shown in PAL format in Listing #2.

To use the program, install the RAM cartridge, and close S1 and S2. Be sure S3 and S4 are both open. Then turn on the computer. The cartridge RAM is now "in parallel" with system RAM. The two are examined together by the C64, and the same data is stored in each at the corresponding addresses. This step is important. If the two RAMs contained different data, they would fight each other on the data bus.

LOAD the "RAMCART" program with ",8,1". This places the code at the start of RAM cartridge memory. Now LOAD the BASIC program you want to store. Do not RUN it. Type

```
SYS 32882
```

The machine code stored by "RAMCART" will copy the BASIC program into the cartridge RAM. If the program is too big, over 31 disk blocks, you'll get an error message instead. When the "READY" prompt appears, open S2. This disconnects the cartridge from the READ/WRITE line, and the data cannot be changed by the computer.

Turn off the C64. The battery will retain the program in the cartridge RAM. Close S4 to tell the PLA that this is an 8K cartridge, and turn the computer back on. The auto-start code in the RAM cartridge will cause the system to initialize BASIC normally. Then it copies your program back to the BASIC memory area. The "RUN" command is placed in the keyboard buffer and the computer executes it, starting your program.

The RUN-STOP/RESTORE combination will bring you out of your BASIC program and display the "READY" message. To re-RUN the program in the cartridge, use a hardware reset switch or type

```
SYS 64738
```

A different technique is required to program the upper 8K of RAM in a 16K cartridge. We need to use the ROMH line from the PLA to select the cartridge memory, since the PLA will switch system ROM in otherwise. But the PLA will not let us write data to the memory selected by ROMH. S5 switches the upper 8K RAM select line between the ROMH output from the PLA and the $6000 - $7FFF output from the 74LS42. With S5 in the $6000 position, you can change the upper 8K of data by writing to the RAM at this lower location. Moving S5 back to the ROMH side causes the PLA to switch in the RAM at either $A000 or $E000, depending on the settings of S3 and S4.

For example, to change BASIC, place a 16K ram cartridge in the computer. Close S1 and S2, open S3 and S4, and move S5 to the $6000 position. Turn on the computer. LOAD a machine language monitor that resides below $6000 or above $C000, and use it to copy the BASIC ROM to the RAM at $6000. Use the memory examine mode to look at the nine bytes starting at $6378. This is the text "READY." followed by a "RETURN" ($0D), a line feed ($0A), and a terminating zero byte ($00). Use the monitor to change the text.

Now open S2 to lock the changes in RAM, and turn off the computer. Move S5 to the ROMH position. Close S3 and S4. This tells the PLA to place the 8K of RAM with the modified BASIC in the address area normally used by the BASIC ROM. Turn on the computer and you'll see your modified "READY" prompt. You'll also see only 30,719 BASIC bytes free, because the lower 8K of ram cartridge is also switched in by the PLA. You can use the lower 8K to hold BASIC programs, or extensions in addition to any modifications you make to the BASIC operating system.

The switch settings for programming and using the cartridge are summarized in Figure 3.

Figure 3

	S1	S2	S3	S4	S5
Reading From Cartridge:					
8K Cartridge	ON	OFF	OFF	ON	X
16K Cart., Upper 8K At $A000	ON	OFF	ON	ON	ROMH
16K Cart., Upper 8K At $E000	ON	OFF	ON	OFF	ROMH
Writing To Cartridge:					
8K Cartridge	ON	ON	OFF	OFF	X
16K Cartridge	ON	ON	OFF	OFF	$6000

The ram cartridge is fully compatible with expansion cards which allow several cartridges to be plugged in at the same time. Be sure to turn S1 off when you select a different cartridge so the RAM at $8000 will be removed from the buses. You can use the ram cartridge on a C128 also. The GAME and XROM lines aren't used in C128 mode. The MMU (Memory Management Unit) looks for a different code instead. You'll have to write a C128 auto-boot routine, but use the procedure above from C64 mode to install it.

We think you'll find the ram cartridge an inexpensive alternative to purchasing an EPROM burner and eraser to make your own cartridges. Even if you already have EPROM programming equipment, the ease and speed of making changes to your cartridge software may be an asset.

Although Geoduck Developmental is not in the retail component sales business, we will make 4464-15 RAMs and battery/socket kits available at cost for Transactor readers. Please send $15 (Canadian) for each RAM and $5 for each battery and socket. For orders outside Canada or the USA, add $5 for postage. Send orders or any questions or comments on the ram cartridge to:

www.Commodore.ca
May Not Reprint Without Permission

Geoduck Developmental Services
PO Box 58587
Seattle WA 98188
USA

Listing 1: Basic Loader To Create RAMCART Module On Disk

```
FO 1000 rem save "0:ramcart.ldr",8
AH 1010 rem ** by: john bush and noel nyman - seattle, wa
IK 1020 rem ** auto-start support prg
KF 1030 rem ** for c64 ram cartridge
EI 1040 :
CI 1050 rem ** this program will create
JB 1060 rem ** a load ",8,1" module on
HO 1070 rem ** disk called 'ramcart'
MK 1080 :
NC 1090 open 15,8,15: open 8,8,1,"0:ramcart"
BN 1100 input#15,e,e$,b,c: if e then close 15: print e;e$;b;c:
         stop
FH 1110 for j=32768 to 32999: read x: print#8,chr$(x);:
         ch=ch+x: next: close8
ED 1120 if ch<>28345 then print "checksum error!": stop
LC 1130 print "** module created **": end
IO 1140 :
NL 1150 data   0, 128,   9, 128,  94, 254, 195, 194
PI 1160 data 205,  56,  48, 162,   5, 142,  22, 208
LH 1170 data  32, 163, 253,  32,  80, 253,  32,  21
AM 1180 data 253,  32,  91, 255,  88,  32,  83, 228
FO 1190 data  32, 191, 227, 162, 251, 154, 172, 224
KO 1200 data 128, 174, 225, 128, 132,  43, 134,  44
PM 1210 data 172, 228, 128, 174, 229, 128, 132,  95
OD 1220 data 134,  96, 172, 226, 128, 174, 227, 128
KC 1230 data 132,  88, 134,  89, 136, 192, 255, 208
AN 1240 data   1, 202, 132,  45, 134,  46, 169, 160
AB 1250 data 133,  91, 169,   0, 133,  90,  32, 191
AG 1260 data 163, 169,  82, 141, 119,   2, 169,  85
GL 1270 data 141, 120,   2, 169,  78, 141, 121,   2
CA 1280 data 169,  13, 141, 122,   2, 169,   4, 133
NG 1290 data 198, 108,   2,   3,  56, 165,  46, 229
PL 1300 data  44, 170, 165,  45, 229,  43, 168, 224
NE 1310 data  31, 176,  67, 140, 228, 128, 142, 229
GL 1320 data 128,  56, 169, 159, 237, 229, 128, 141
DG 1330 data 229, 128, 169, 255, 237, 228, 128, 141
GF 1340 data 228, 128, 165,  43, 141, 224, 128, 133
CO 1350 data  95, 165,  44, 141, 225, 128, 133,  96
EI 1360 data 164,  45, 166,  46, 200, 208,   1, 232
OG 1370 data 140, 226, 128, 132,  90, 142, 227, 128
KN 1380 data 134,  91, 169, 160, 133,  89, 169,   0
DA 1390 data 133,  88,  32, 191, 163,  96, 169, 204
CH 1400 data 160, 128,  32,  30, 171,  96,  80,  82
FA 1410 data  79,  71,  82,  65,  77,  32,  84,  79
MO 1420 data  79,  32,  76,  65,  82,  71,  69,  10
HP 1430 data  13,   0,   0,   0,   0,   0,   0,   0
```

Listing 2: PAL Source for support program

```
MM 1000 rem save "0:ramcart.pal",8
AH 1010 rem ** by: john bush and noel nyman - seattle, wa
IL 1020 rem ** auto-start support prg for c64 ram cartridge
KH 1030 :
JP 1040 open 8,8,1,"0:ramcart"
LO 1050 sys 700
HE 1060 .opt o8
EB 1070 *          =    $8000
OK 1080 ;
FP 1090 ;*** equates ***
CM 1100 ;
KM 1110 txttab     =    $2b    ;start of basic text
HL 1120 vartab     =    $2d    ;end of basic text
BL 1130 source     =    $5f    ;start of source to copy
KI 1140 end        =    $5a    ;end+1 of source to copy
MC 1150 dest       =    $58    ;end+1 of destination
NC 1160 ndx        =    $c6    ;no of characters in keyboard
                                buffer
BC 1170 keyd       =    $0277  ;start of keyboard buffer
IK 1180 warm       =    $0302  ;basic warm start vector
HA 1190 copy       =    $a3bf  ;copy memory
LK 1200 strout     =    $ab1e  ;print string
LG 1210 vicctrl    =    $d016  ;vic control register
DN 1220 vectors    =    $e453  ;copy basic vectors to ram
DF 1230 init       =    $e3bf  ; initialize basic interpreter
LL 1240 ioinit     =    $fda3  ;initialize i/o
HA 1250 ramtas     =    $fd50  ;initialize memory pointers
HM 1260 restor     =    $fd15  ;restore i/o vectors
EA 1270 cint       =    $ff5b  ;init screen and keyboard
NP 1280 nmicont    =    $fe5e  ;continue with nmi routine
AI 1290 ;
GE 1300 ;*** auto-start basic program ***
EJ 1310 ;
BG 1320 ;place start of code in cartridge vectors
PM 1330 .byte   <start,>start
AE 1340 .byte   <nmicont,>nmicont
KI 1350 ; 'cbm' with bit 7 set
FH 1360 .byte   $c3,$c2,$cd
OI 1370 .asc    "80"
KN 1380 ;
LF 1390 ;'start' calls most of the routines
GK 1400 ;which would be executed if a cartridge
ID 1410 ;had not been detected. system vectors
AD 1420 ;and basic are initialized.
MA 1430 ;
BH 1440 start  ldx   #5
FE 1450        stx   vicctrl
EH 1460        jsr   ioinit
FI 1470        jsr   ramtas
EF 1480        jsr   restor
EF 1490        jsr   cint
KG 1500        cli
MO 1510        jsr   vectors
FN 1520        jsr   init
DA 1530        ldx   #$fb
KA 1540        txs              ;initialize stack pointer
EI 1550 ;
PL 1560 ;copy the basic program from
JH 1570 ;the area under $a000 to the start-of-basic
IP 1580 ;and set up the basic text and variables
DM 1590 ;vectors. place 'run' in the keyboard buffer and
OP 1600 ;enter basic through the warm start vector.
AM 1610 ;
PI 1620        ldy   txtt       ;store start of basic
IJ 1630        ldx   txtt+1     ;saved with program
```

www.Commodore.ca
May Not Reprint Without Permission

```
OK 1640       sty  txttab     ;at op system vector
LE 1650       stx  txttab+1
PM 1660       ldy  stsour     ;store start of source
LJ 1670       ldx  stsour+1   ;at vector for copy routine
LJ 1680       sty  source
DG 1690       stx  source+1
GA 1700       ldy  vart       ;store end of destination
                              (+1)
FA 1710       ldx  vart+1     ;at copy routine vector
FN 1720       sty  dest
HO 1730       stx  dest+1
PJ 1740       dey             ;subtract one from low byte
FA 1750       cpy  #$ff
MP 1760       bne  cont
NH 1770       dex             ;subtract borrow
ND 1780 cont  sty  vartab     ;store op system vector
FK 1790       stx  vartab+1
NN 1800       lda  #$a0       ;end of source (+1) =
                              $a000
PB 1810       sta  end+1
HA 1820       lda  #0
ME 1830       sta  end
KD 1840       jsr  copy
HI 1850       lda  #"r"
KN 1860       sta  keyd
EK 1870       lda  #"u"
KP 1880       sta  keyd+1
DK 1890       lda  #"n"
AB 1900       sta  keyd+2
IP 1910       lda  #$0d       ;<return>
GC 1920       sta  keyd+3
GB 1930       lda  #4         ;number of characters
IB 1940       sta  ndx
JN 1950       jmp  (warm)
OB 1960 ;
DO 1970 ;*** store basic program to cartridge ***
JC 1980 ;calculate the size of the basic text, and
NF 1990 ;print an error message if too large to fit
OB 2000 ;in the cartridge. if okay, subtract the size
MM 2010 ;from $9fff to get the location of the start
DA 2020 ;of the copy to be saved to cartridge. save
PA 2030 ;that vector, and the start and end of basic
ND 2040 ;text for future use. set-up vectors for
JE 2050 ;copy routine and copy program to cartridge.
CI 2060 ;
GL 2070 store sec
NC 2080       lda  vartab+1
DM 2090       sbc  txttab+1   ;find size of basic program
PP 2100       tax
JI 2110       lda  vartab
FN 2120       sbc  txttab
BC 2130       tay
NO 2140       cpx  #$1f       ;max size allowed
CI 2150       bcs  error      ;print error message and
                              quit
DP 2160       sty  stsour     ;store size temporarily
HG 2170       stx  stsour+1
FA 2180       sec
CP 2190       lda  #$9f       ;subtract size from $9fff to
                              find
NP 2200       sbc  stsour+1   ;start of program in car-
                              tridge memory
DD 2210       sta  stsour+1
JG 2220       lda  #$ff
HF 2230       sbc  stsour
PI 2240       sta  stsour
GD 2250       lda  txttab     ;store start of basic for
                              cartridge
EJ 2260       sta  txtt       ;use and in vector for copy
                              routine
JI 2270       sta  source
HC 2280       lda  txttab+1
EP 2290       sta  txtt+1
JG 2300       sta  source+1
PF 2310       ldy  vartab     ;store end of basic (+1) for
                              cartridge
LK 2320       ldx  vartab+1   ;use and vector for copy
                              routine
KP 2330       iny
CK 2340       bne  cont1
KA 2350       inx
BF 2360 cont1 sty  vart
IM 2370       sty  end
IH 2380       stx  vart+1
PL 2390       stx  end+1
OE 2400       lda  #$a0       ;store $a000 (end of car-
                              tridge memory +1)
HO 2410       sta  dest+1     ;in vector for read routine
PF 2420       lda  #0
LD 2430       sta  dest
CJ 2440       jsr  copy
OH 2450       rts
CB 2460 ;
LJ 2470 ;*** print error message ***
GC 2480 ;
LD 2490 error lda  #<message
JC 2500       ldy  #>message
NM 2510       jsr  strout
EM 2520       rts
IF 2530 ;
JG 2540 message    *
AE 2550 .asc  "program too large"
NE 2560 .byte $0a,$0d,$00
AI 2570 ;
AK 2580 ;*** system vector storage ***
EJ 2590 ;
AD 2600 txtt   .word 0        ;start of program in ram
JI 2610 vart   .word 0        ;end of program in ram
IC 2620 stsour .word 0        ;start of source in cartridge
ML 2630 ;
MC 2640 .end
```

www.Commodore.ca
May Not Reprint Without Permission

Upgrade Your C128 With A 48K RAM Disk

Noel Nyman
Seattle, WA

If you tried the C128 RAM Disk programs in Transactor 7-01, you may have been frustrated by the limited memory available for storage and the loss of your eighty column screen. With access to good soldering equipment, a C128 out of it's warranty period, and two new integrated circuit chips, you can easily upgrade your C128's eighty column screen to 64K of RAM (Random Access Memory).

This will give you normal eighty column screen capability plus 48K of RAM to use as file storage, additional text screens, or both.

To make the change, you'll have to unsolder the two RAM chips used by the VDC (Video Display Controller, the 8563 chip). This is NOT a task to be taken on lightly. The C128 uses a double sided board, and the chips sit in tight quarters inside a metal shield. If you don't have both experience with such de-soldering and the proper tools, have the job done by a qualified technician. Anyone who repairs micro-computers should be able to install sockets in place of the RAM chips for a small fee.

The C128 uses two 18-pin 16K DRAM (Dynamic RAM) chips for VDC memory. Each chip stores four bits or one nibble of data. There are only eight address lines (see figure 1). The 8563 sends each address in two parts. The low portion of the address is placed on the bus first, and the RAS (Row Address Strobe) line is brought low. The RAM chips "latch" the low part in internal registers. Then the 8563 places the high portion on the address bus and brings CAS (Column Address Strobe) low.

The RAM chips use the row and column information to select an address from a 64x256 array (16K). They place the corresponding data on the data bus, or store data from the bus depending on the state of the Write line.

Commodore's schematic identifies the chips as 4416's. I'm told there is a pin-for-pin compatible chip numbered 4464. The devices with that number I found turned out to be 24 pin 8x8K CMOS static RAM's, which won't do the job here. If you locate 4464's, be certain that they are 18 pin DRAM's before buying them.

My C128 contains MB81416's made by Fujitsu. Their MB81464 is pin compatible and available for about $8.50 (US). The 41464 from NEC is compatible except for the address lines, and sells for $6.00 (US). It also worked in my computer. The chips in my C128 are 120 nano-second types, a '-12' follows the chip number. The 150 nano-second chips, which are cheaper and more common, also worked in this application.

Another brief warning. There are at least two versions of the 8563 chip (the chip in my machine says "REV 8"). The 64K conversion seems to work with both. But Commodore is under no obligation to support 64K mode in future revisions. If you have a later (or earlier) chip than those we've tested, it may not work in 64K mode. If you're careful about unsoldering the RAM chips, you can replace them in the sockets you install and return your machine to its original form.

After making the chip changes, turn on the C128 in eighty column mode. You should see the normal start-up screen. Connect a forty column monitor or TV set also, so you can enter commands to control the eighty column screen. From forty column mode, enter:

```
POKE 54784,25: POKE 54785,128
```

On Jim Butterfield's 8563 diagram on page 33 of Transactor 7-01, you'll see that bit 7 of register 25 controls bit map or hires mode. If you have a sparkling line on the far right side, you have a newer version of the 8563. Change the '128' in the POKE to '135' to set the Horizontal Scroll bits.

Now we'll look at the next 16K of RAM. On the forty column screen enter:

```
POKE 54784,12: POKE 54785,64
```

Register 12 holds the high byte of the start-of-display address. If you think the screen looks unchanged, you're right. Before we explain, try one more command:

```
POKE 54784,12: POKE 54785,128
```

This time you should see some changes. The VDC, when working in 16K mode, does not support the second highest address bit. So, when you tried to look at the second 16K block, you saw the "mirror image" of the first 16K. For some reason, the highest address bit is supported, and a new 16K block and mirror image appear when you address the upper 32K of memory.

To switch the 8563 to 64K mode, we have to set bit 4 of register 28, labeled "RAM" in Butterfield's diagram. This register also tells the chip where to find the character set data, so we have to leave that information in place. Enter:

```
POKE 54784,28: POKE 54785,48
```

The screen will change dramatically. Patterns of lines (the default values in the RAM chips when they power-up) have infiltrated portions of the text, attribute, and character set areas. The 8563 expects some different RAM chips in this mode (4164's) so it looks at the addresses differently. Now try:

```
POKE 54784,12: POKE 54784,X
```

Where X=0, 64, 128, and 192. You should see four different displays, one for each 16K block. To return to text mode, use the command above to POKE a zero in register 12, then enter:

```
POKE 54784,25: POKE 54785,64
```
(use 71 if you used 135 earlier)

Remember that switching RAM modes scrambled the memory. To return things to normal, you'll have to re-copy the character sets to RAM and cleanup things generally.

Listing #1 creates the ML code to do that. It will also allow you to access all of the added RAM as text screens. After running the program, type:

www.Commodore.ca
May Not Reprint Without Permission

BSAVE "SETUP/SWAP", B0, P3584 TO P3713

This SAVEs the ML to disk. Type "SYS 3584" and after a few moments your startup screen will re-appear. Now type:

SYS 3672, 8, 10, 20: PRINT "SCREEN #8"

You'll be switched to screen #8 (starting at $4000 in the new RAM). To switch back, type:

SYS 3672, 0, 0, 0

The first number following the SYS is the destination screen. Screen #0 is the default screen starting at $0000. The program will not allow you access to screen #1 (the default attribute map) or screens #4 through #7 (the character sets) since the "READY" prompt and anything you type would garble the data.

The second and third numbers are the row and column for the cursor on the new screen. If you don't specify row and column, you will get erratic results.

The "SETUP/SWAP" program is located at $0E00. This overwrites the sprite data area, but makes this program compatible with the RAM Disk programs in Transactor 7-01. Listing #2 shows the modifications to change to a 48K BASIC RAM Disk. No modifications are necessary for the "Memory DRAM" program. Just use a starting lb=0, and hb=64 to begin saving to the RAM at $4000.

You can use additional text screens and RAM Disk at the same time, so long as you don't switch to a text screen area holding a SAVEd file. Text screens #2 and #3 aren't used by the BASIC RAM Disk and are always safe.

Since all text screens share the same attribute RAM (unless you change the vector at registers 20 and 21), any change in character set, color, etc., will change the same screen locations on ALL text screens. This can be a feature or a bug, depending on your application. You can avoid unexpected changes by disabling the attribute map. To do that, clear bit #6 in register 25:

POKE 54784,25: POKE 54785, PEEK(54785) AND 191

Only the upper case/graphics character set will be available. You can select character color for the entire screen by changing the high four bits in register 26. The lower four bits select background color in all modes.

The only problem with your 64K RAM is that a RUN-STOP/RESTORE or system RESET disables it. If you initialize the "BASIC DRAM" program, you won't have the RESTORE problem. "BASIC DRAM" jumps around the RESTORE routines.

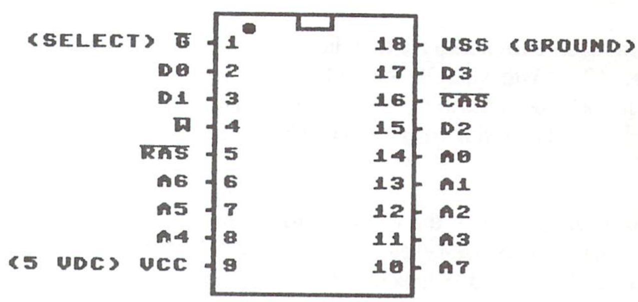
The other method is to change the kernal operating system ROM so the eighty column chip is always initialized in 64K mode. I should have a new version of the kernal available by the time you read this. It will support the 64K chips, have the RAM Disk routines in ROM, and fix the CAPS-LOCK 'Q' bug as well. If you'd like a copy of the code in order to make your own replacement ROM, send $2 (either US or Canadian) and a disk to:

Noel Nyman
Geoduck Developmental Systems
PO Box 58587
Seattle WA 98188

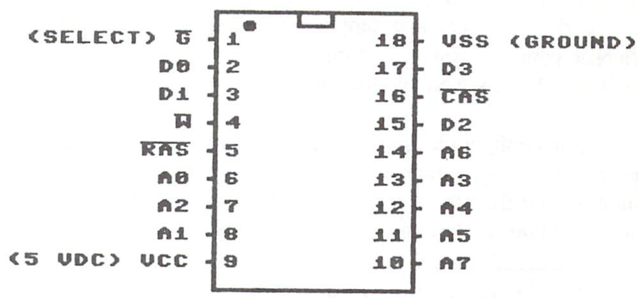
If you can't find the 64K dynamic RAM chips locally, you can contact the following sources. Both have a $25 (US) minimum order restriction.

For 81464s (Fujitsu) contact:	For 41464s (NEC) contact:
Integrated Electronics Corp.	Marshall Industries
1750 124th NE	14102 NE 21st
Bellevue WA 98005	Bellevue WA 98007
206 455-2727	206 747-9100

4416 - 81416 - 81464



41464



Listing #1

```
BM  100 rem save "0:setup/swap.ldr",8
OO  110 for j= 3584 to 3712: read x: poke j,x: ch=ch+x: next
IE  120 if ch<>15875 then print "checksum error!": stop
GP  130 :
LD  140 data 169,  48, 162,  28,  32, 204, 205, 169
ME  150 data   0, 162,  18,  32, 204, 205, 232,  32
HP  160 data 204, 205, 160,   0, 169, 255, 162,  30
OD  170 data  32, 204, 205, 169,  32,  32, 202, 205
EL  180 data 136, 208, 241,  32,  12, 206, 165, 215
NP  190 data  48,  10, 169,  27,  32, 210, 255, 169
HN  200 data  88,  32, 210, 255,  32, 155,  65,  96
HM  210 data   0,   0,  16,  24,   0,   0,   0,   0
OP  220 data  64,  72,  80,  88,  96, 104, 112, 120
GD  230 data 128, 136, 144, 152, 160, 168, 176, 184
FA  240 data 192, 200, 208, 216, 224, 232, 240, 248
PE  250 data 134, 235, 132, 236, 168, 185,  56,  14
HP  260 data 141,  46,  10, 162,  12,  32, 204, 205
FC  270 data 162,  14,  32, 204, 205, 166, 235, 189
GJ  280 data  51, 192,  10, 133, 224, 189,  76, 192
HJ  290 data  41,   3,  42,  13,  46,  10, 133, 225
FL  300 data  96
```

Listing #2: To change the "BASIC DRAM" program from Transactor 7-01 to work with 64K RAM, enter the two replacement lines below in the BASIC loader.

```
BK  2360 data 170, 169, 254, 229, 252,  32, 187,  12
DI  2790 data  76,  51, 255,   0,  64,   0,   0,   0
```

www.Commodore.ca
May Not Reprint Without Permission

The Commodore 128 – Banking On The Turns

Jim Butterfield
Toronto, Ontario

A previous Transactor article talked about the Commodore 128 "memory banks". (See "The C128 – You can Bank On It", The Transactor, July 1986). In case you missed that one, I'll give you a quick summary.

Commodore BASIC seems to indicate that there are 16 banks (numbered 0 to 15) that may be selected by using the BANK command. The same scheme is used in the machine language monitor – an address will be prefixed with a digit from 0 to F – the same bank values of 0 to 15.

But it turns out that the average programmer – with no cartridge, internal ROM, or RAM expansion – can only make use of four of these banks: 0, 1, 14 and 15 (hex 0, 1, E and F).

Going a little deeper into the matter, we find that these 16 "banks" – more accurately, configurations – are really just a sampling of what can be done. A machine language programmer can create 256 different configurations by storing a selected value into address $FF00, the MMU's "configuration register".

Not all 256 configurations are useful. There are sixteen architectures that the ML programmer can use. Only four of them have BANK numbers, but the others can be reached by storing the appropriate value at $FF00. Table 1 shows these combinations.

Table 1. The sixteen 'useful' architectures.

FF00 Poke Value	(Addresses whose first hex digits are:) 0123	4567	89AB	CEF	D	Bank Number	Store to
00	RAM0	ROM	ROM	ROM	I/O	"BANK 15"	
01	RAM0	ROM	ROM	ROM	CGEN	"BANK 14"	FF03
02	RAM0	RAM0	ROM	ROM	I/O		
03	RAM0	RAM0	ROM	ROM	CGEN		
0E	RAM0	RAM0	RAM0	ROM	I/O		
0F	RAM0	RAM0	RAM0	ROM	CGEN		
3E	RAM0	RAM0	RAM0	RAM0	I/O		
3F	RAM0	RAM0	RAM0	RAM0	RAM0	"BANK 0"	FF01
40	RAM1	ROM	ROM	ROM	I/O		
41	RAM1	ROM	ROM	ROM	CGEN		FF04
42	RAM1	RAM1	ROM	ROM	I/O		
43	RAM1	RAM1	ROM	ROM	CGEN		
4E	RAM1	RAM1	RAM1	ROM	I/O		
4F	RAM1	RAM1	RAM1	ROM	CGEN		
7E	RAM1	RAM1	RAM1	RAM1	I/O		
7F	RAM1	RAM1	RAM1	RAM1	RAM1	"BANK 1"	FF02

Note that in all configurations, the first 1K of memory (addresses 0002 to 03FF) is always RAM0. Addresses 0 and 1 are internal to the processor chip.

More Detail

The previous article discussed the configurations, including those created by using values 0E and 4E. Storing $0E into FF00 creates the RAM 0 for addresses up to BFFF; storing $4E creates RAM 1 for this area. The Kernal and I/O take up their normal positions. These two were described as "ideal" configurations for serious machine language stuff: 0E for a program in RAM 0, and 4E for a program in RAM 1. Basic is removed, and you have lots of memory to play with.

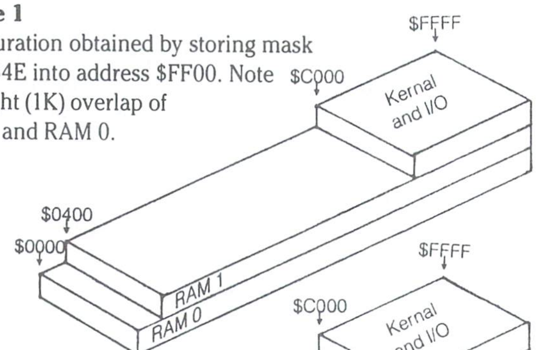
That's correct as far as it goes. But the RAM 1 configuration, created with mask value $4E, has a problem. If the machine language program calls a Kernal routine, the Kernal will want to use locations within RAM 0 memory. Some of these locations are available and ready: as Figure 1 shows, all addresses below 1024 decimal (hex 0400) use RAM 0. For all practical purposes, RAM 1 doesn't start until address 1024.

But other locations in RAM 0 that the Kernal uses are above 0400 . . . and if your program in RAM 1 calls a Kernal subroutine, there's a good chance that the Kernal coding will cheerfully assume that it's viewing RAM 0 and will unknowingly go into RAM 1 for important values. And if it does that, it will probably goof up.

The most important area above $0400 used by the Kernal is in page 0A. Addresses 0A00 to 0AC4 in RAM 0 are used for numerous system things, and the Kernal will foul up if it tries to get (or store) values in RAM 1 by mistake.

The address you're likely to meet first is when you're sending to the screen using the Kernal routine at $FFD2. Location $0A21 (bank zero, of course) is the "screen freeze flag" – it's an interrupt-set image of the "no scroll" key which is located at the top of the keyboard. When this location contains a zero, printing to the screen will take place normally. When it contains any other value, the computer will wait until it's zero. Under normal circumstances, releasing the no-scroll key will put a zero into address $0A21 (bank zero, of course), and the computer will proceed with printing to the screen. But if the computer is watching the wrong memory bank, it will NEVER do the job because it will never see a zero at $0A21.

Figure 1
Configuration obtained by storing mask value $4E into address $FF00. Note the slight (1K) overlap of RAM 1 and RAM 0.



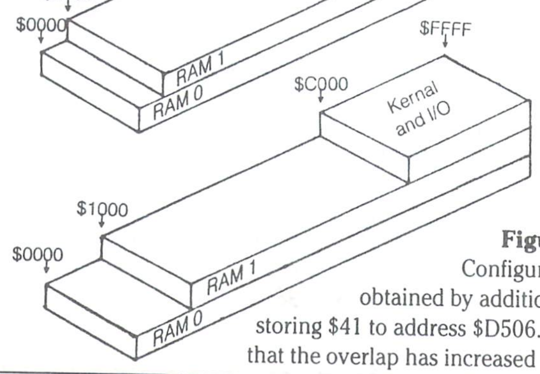


Figure 2
Configuration obtained by additionally storing $41 to address $D506. Note that the overlap has increased to 4K

www.Commodore.ca
May Not Reprint Without Permission

Fixing It

Okay, so if we want to program in RAM 1, we must find some way to "expose" more addresses in RAM 0 for the use of the Kernal. The solution is quick and simple.

Here's the story: we know that the first 1K of memory is always RAM 0, no matter what configuration has been chosen. That size – 1K – is user adjustable. You can adjust it without problems by storing a new value at $D506. That's the register in the MMU that sets "common RAM", which is the proper name for this piece of "bank-shared" memory.

The normal value stored in location $D506 is 4 . . . that creates a shared ("common") RAM for all addresses below 1024 decimal (hex 0400). If we change it to 5, the shared memory area zooms up to 4K: in other words, all addresses below 4096 (hex 1000) will be taken from RAM 0; RAM 1 will never be referenced in this memory area. If you're interested, value 6 would give 8K common RAM and value 7, 16K. But we don't need to go that far.

Compare Figures 1 and 2. Both show the computer in the configuration created by storing a value of $4E into address $FF00. Figure 1 is "normal" common RAM . . . Figure 2 shows "extended" common RAM, created by putting a value of 5 into the register at $D506.

Once we've extended the common RAM, as shown in Figure 2, the Kernal will give us no trouble . . . it has easy access to the memory it needs in bank zero, page 0A.

No Problems

You should understand that changing the size of common RAM is a fundamental system change. It affects all parts of your computer . . . user programs, Basic, Kernal, interrupt routines, and possibly the video chip. It will not be effected by values stored to $FF00 or by BANK commands. It seems dangerous; but in fact, it's relatively safe.

If you feel like experimenting, you may go the machine language monitor and arrange to change the contents of $FD506 to 5. Do it the same way as you'd perform any memory change; note that we need to specify bank 15 with a leading "F". If you do this, you'll quickly discover that all memory locations below $1000 are the same regardless of bank number. In other words, if you display the contents of 00A00 and then of 10A00, you'll get the same values. This was not true before you changed D506. Restore the value in $FD506 to 4 before you leave the monitor.

If you change the common RAM value, I recommend that you put it back when you're finished. Why? There's only one reason I can think of: Basic variables start in RAM 1 at address $0400 (1024). If you're going to use Basic, you'll want to reduce common RAM space to its original value so that Basic variables can go into their proper bank. They'd make a terrible mess if they starting going into RAM 0.

An Example

The following program is based on work done by John Gager. It's written in Basic to allow easy entry.

```
100 BANK 1
110 FOR J=32768 TO 32802
120 READ Y
130 T=T+Y
140 POKE J,Y
150 NEXT J
160 IF T<>4057 THEN STOP
170 BANK 1
180 SYS 32768
190 BANK 15
200 DATA 169,78,141,0,255
210 DATA 169,5,141,6,213
220 DATA 160,0
230 DATA 185,29,128,32,210,255
240 DATA 200,201,13,208,245
250 DATA 169,4,141,6,213
260 DATA 96
270 DATA 72,69,76,76,79,13
```

The program is embedded in the DATA statements: the loop at 110 puts it into RAM 1 at addresses $8000 to $8022. The extra BANK 1 in line 170 isn't really needed; it's just a reminder that the following SYS leaps into RAM 1. The BANK 15 in line 190 is purely for neatness' sake, restoring the machine to its original state.

Let's look at the machine language code:

```
18000  A9 4E     LDA#$4E
18002  8D 00 FF  STA $FF00
```

This sets the configuration to RAM 1 plus Kernal. Until we do this, the computer is in "Bank 1" configuration; that means that the Kernal is not present.

```
18005  A9 05     LDA#$05
18007  8D 06 D5  STA $D506
```

Here's where we expand "common RAM" to allow the Kernal to see addresses in the region of 0A00 in RAM 0. We'll put things back later. By the way, this will work only if we have done the earlier store to $FF00; can you see why?

```
1800A  A0 00     LDY#$00
1800C  B9 1D 80  LDA$801D,Y
1800F  20 D2 FF  JSR $FFD2
18012  C8        INY
18013  C9 0D     CMP#$0D
18015  D0 F5     BNE$800C
```

A straightforward loop to print a simple message to the screen. But it would not work if we hadn't (i) installed the Kernal with our store to $FF00, and (ii) opened up access to RAM 0 with our store to $D506.

```
18017  A9 04     LDA#$04
18019  8D 06 D5  STA $D506
1801C  60        RTS
```

The above code returns the common RAM to 1K and then quits. Note that we don't need to restore the "bank 1" configuration.

The program is followed by a few more bytes containing the message to be printed.

Conclusion

Yes, you can put programs in RAM 1, but it's more complex than for RAM 0. It's useful to see how the architecture can be manipulated. The Commodore 128 has surprising system flexibility.

Thanks go to John Gager who pointed out the nature of the problem and made a significant contribution to its solution.

www.Commodore.ca
May Not Reprint Without Permission

Software On/Off Write Protect for the 1541

William Fossett
San Diego, CA

Write-protect disks of your choice -- with a single command to your 1541!

In the July, 1985 issue of Transactor (Vol. 6, Issue 01), Chris Johnsen introduced the little known '&' (ampersand) command and file structure for the Commodore 1541 disk drive. The following article will explain this DOS feature further, and expand the concept into a method for write protecting diskettes using a software protection scheme. Two programs are listed at the end of the article: one is a source listing, written in standard Commodore assembler format, and the second is a BASIC loader which will create the program "&WP" on a diskette; it, in turn, can be used to write protect (or un-write protect) any diskette. The assembler code source listing is provided for explanation and documentation; only the BASIC loader needs to be typed in.

The Commodore 1541 disk drive contains (among other things) a CPU, 16K of ROM, and 2K of RAM. The 16K of ROM contains the Disk Operating System (DOS) and the 2K of RAM is used by the DOS for a variety of functions. The structure of the RAM is similar to the RAM in the C-64: zero page ($0000-$00FF) is used for frequent and important storage; most of page one ($0100-$01FF) is the stack area for the drive; page two ($0200-$02FF) is used as a work area; the remaining 5 pages of RAM (page three through page seven, or $0300-$07FF) are referred to as buffers 0, 1, 2, 3, and 4. Each buffer is $0100 hex (256 decimal) bytes long - the exact size of one sector on a diskette. As you might have guessed, these buffers are used to transfer blocks of 256 bytes from a diskette to the computer, or vice-versa. The DOS has its own methods of loading and unloading these buffers, depending on the specific operation, and which buffers are already being used. We, as programmers, have the option to use this RAM also, but with the DOS program being so big (16K) and RAM so small (2K), the DOS has a tendency to write over anything we might put in RAM. We actually can use buffers 0, 1, 2, and 3 ($0300-$06FF) quite freely, if we write our program, execute it and then get out. However, buffer 4 is a "special" buffer which contains an exact copy of the Block Availability Map (BAM) of the diskette currently in the drive. As a rule, it's probably best to avoid writing to, or otherwise tampering with, this buffer, as anything you write there may end up on the header (track 18, sector 0) of your diskette. However, knowing this, we can construct a useful tool which will allow us to "soft" write protect a diskette.

An '&' file is usually referenced as a utility loader. As it is used here, and as it has been previously used (Transactor, Vol.6 #1), it is similar to a block execute command. The '&' file is loaded from diskette into disk memory and executed with one command. Used in this fashion, an '&' file may be of any type (USR, PRG, SEQ), and need only include two specific features in its structure: 1) a length byte following the load address (# of bytes after this byte up to the checksum) and, 2) a checksum byte at the end (a sum of all bytes from the load address up to the checksum and all carry bits). If the file meets these 2 criteria, it is a valid '&' file. Executing the '&' file is accomplished with a standard disk command string - I prefer the shortened syntax of:

```
OPEN 15,8,15,"&filename":CLOSE15
```

No colons, drive numbers, or special syntax need be associated with the '&' file on the 1541 (other than it needs the '&' as the first character).

The BASIC program at the end of this article (PROGRAM 1), when run, creates a program named "&WP" on a standard 1541 diskette. This "ampersand" program ("&WP") will allow you to write protect (or un-write protect) the diskette by executing the command:

```
OPEN 15,8,15,"&WP":CLOSE15
```

If the command is executed to a previously unprotected diskette, it will write protect it; if the diskette is already write protected (using this command) then it will un-write protect it. It will "flip-flop", as it were, between the two conditions (protected/unprotected) each time it is executed. A look at the source code (PROGRAM 2) reveals how this is accomplished: the file "&WP" loads and runs in buffer 3 ($0600); the BAM is loaded from the diskette into buffer 4 of the 1541 by initializing the drive; the third byte in the buffer is changed from an 'A' to an 'E' (or back again if un-write protecting) with an Exclusive OR; this change is also reflected in the disk title block ($07A6); the disk version byte ($0101) is set to A (this step is superfluous if we are write protecting, but necessary if we are un-write protecting - see below); the (modified) buffer contents are written back to the BAM; and, finally, the drive is initialized

www.Commodore.ca
May Not Reprint Without Permission

again to update the disk version byte ($0101) so it reflects the current condition. The creation of a visual "flag" in the disk title block is purely cosmetic as far as the DOS is concerned. But it is important to *you:* "2A" in the disk title block indicates a normal (un-write protected) condition; "2E", however, indicates the diskette is write protected – you will not be able to delete, rename, or save files on this diskette.

This write protection is not equivalent to the one you perform when you put a tab over the notch on a diskette. The present scheme changes a byte on the diskette that the DOS checks to find out what type of drive the diskette was formatted on (an 'A' indicates the diskette was formatted on a 1540 or 1541, an 'E' corresponds to a non-existent drive). If the byte does not match the correct format, reading can be performed, but writing is not allowed. Any writing to this diskette will produce a DOS error message (#73). However, this scheme will not prevent a format instruction from working – WARNING: you should still use a write protect tab if you are using disk copy programs or have possible format execution commands to perform. Un-write protecting a previously write protected diskette is a simple matter of fooling the DOS. The DOS checks location $0101 (DSKVER) in memory to see the format version of the diskette it is dealing with. If we change that location from an E to an A (indicating the disk was formatted on a 1541) then we can write to the diskette even if there is an E in the format byte on the diskette. Thus, we can re-execute "&WP" (OPEN 15,8,15,"&WP": CLOSE 15) and the diskette will be un-write protected once more.

About The Author

Bill Fossett is the owner of Hacker's Hardware, a software producer for the C-64 / 1541 equipment line. He has authored a utility package that alters the C-64 computer to operate under RAM control rather than Kernal ROM. Inquiries concerning this product should be directed to P.O. Box 7933, San Diego, CA 92107.

Listing 1: BASIC program to create the "&wp" file on disk.

```
10 rem courtesy of hacker's hardware
15 rem w fossett – l.&wp.050485
20 :
25 rem this program creates a file
30 rem on diskette that will write
35 rem protect or un-write protect
40 rem the diskette. use as follows:
45 :
50 rem   open15,8,15,"&wp":close15
55 :
60 open 8,8,8,"&wp,p,w"
65 for i=1 to 29: read j: print#8,chr$(j);
70 next i: close 8
75 data   0,   6,  25,  32,  66, 208
80 data 173,   2,   7,  73,   4, 141
85 data   2,   7, 141, 166,   7, 169
90 data  65, 141,   1,   1,  32,   7
95 data 239,  76,  66, 208,  25
99 end
```

Listing 2: 6502 Source code for the 1541-resident "wp" program.

```
00001 0000                   ;*******************************
00002 0000                   ;* utility to load and execute *
00003 0000                   ;* a soft on/off write protect *
00004 0000                   ;* for 1541 format – w fossett *
00005 0000                   ;*******************************
00006 0000                   ;
00007 0000                   initdr  =     $d042
00008 0000                   buff4   =     $0700
00009 0000                   dskver  =     $0101
00010 0000                   sb10    =     $ef07
00011 0000                   ;
00012 0000 00 06    .word $0600  ;load addr for pgm
00013 0002 19       .byte 25     ;# bytes *+1 to cksum
00014 0003 20 42 d0 jsr   initdr ;load bam to buff4
00015 0006 ad 02 07 lda   buff4+2 ;dos ver in bam image
00016 0009 49 04    eor   #$04   ;flip a to e / e to a
00017 000b 8d 02 07 sta   buff4+2 ;update bam image
00018 000e 8d a6 07 sta   $07a6  ;and visual flag (2?)
00019 0011 a9 41    lda   #$41   ;changing dskver to
00020 0013 8d 01 01 sta   dskver ;1541 if doing un-wp
00021 0016 20 07 ef jsr   sb10   ;write bam to disk
00022 0019 4c 42 d0 jmp   initdr ;update dskver to new
00023 001c 19       .byte $19    ;checksum 4 thru *-1
00024 001d          ;
00027 001d          .end
```

www.Commodore.ca
May Not Reprint Without Permission

Amiga Dispatches

by Tim Grantham, Toronto, Ontario

It's been a year now since the Amiga, amid great pomp and ceremony, was bestowed upon us like a veritable gift from Mount Olympus. Commodore has just had its first profitable quarter in almost two years and there is every indication that that will continue. Reasonably effective software is available now, at reasonable prices, and some **very** sophisticated hardware has appeared, at equally sophisticated prices. It's time to come up for air and take a hard look at the state of the machine.

There is no question that the machine has established credibility. Even those who worship at the shrine of IBM have deigned to graciously acknowledge the Amiga's graphics power; but asking an IBM clone (I find that the term sometimes applies as much to the users as to their machines) to recognize the Amiga as a serious business or development machine, is like asking an American sports commentator to acknowledge the Toronto Blue Jays as World Series contenders -- the facial expression resembles the gentle puzzlement of an elephant felled by a blowgun.

Despite the excitement over the Sidecar, it's become apparent to me that most of those developing serious applications are coming from anything but a PC background: some are moving up from 8-bit machines, (and finding it hard to climb the steep learning curve); many more are dropping in from a UNIX environment -- not surprising, considering the many functional and design similarities between the Amiga OS and UNIX. It is much easier to port programs from the multitasking UNIX OS than from MS-DOS.

As befits the nature of the machine, Amiga users are an eclectic bunch. They range from engineers who find that a Turboed Amiga (see hardware news) is faster and cheaper than a MicroVAX; to former 64 and Atari owners who want to play the very best computer games; to artists and musicians for whom the Amiga was the first computer they felt any affinity for. It's the engineers and the artists who I feel will become the major users of the Amiga -- many former 8-bitters have felt intimidated by the complexity of AmigaDOS and the CLI, and underwhelmed by Intuition and multitasking. (More about multitasking later.)

Sensing this trend, CBM got smart and started pitching the Amiga to vertical markets in advertising, engineering, and media production houses. (I still wince when I see the ads saying "Give your child an unfair advantage." Aside from the emotional blackmail aspect, the idea is a perfect example of yuppie overkill. An 8-bit computer is perfectly adequate and a heckuva lot cheaper.) The promotion appears to be working: my sources at SIGGRAPH (*the* computer graphics conference) told me the Amiga was the hit of the show, evoking tremendous interest from engineers, artists, and, oddly enough, the military. The last may be because the US Department of Defense has granted the Amiga a 'no bid' status: this means simply that a department within the DOD may simply go out and purchase one -- they are not required to put out a call for bids from competing manufacturers.

Which leaves us with best guesses of between 60,000 and 100,000 machines sold and a solid core of professionals for users -- not at all the scenario CBM envisioned, I believe, but one they would do well to capitalize on. The popular acceptance will come later when the machine and extra memory are cheaper, when the Amiga can be integrated with CD-ROMs and VCRs, and when both users and developers learn to take full advantage of that great concept in the sky, 'multitasking'.

Taking multitasking to multitask

"When I was working with mainframes, multitasking made the invention of BASIC possible and practical." says Jim Butterfield. "Here was something wonderful -- instead of waiting a day to have your program keypunched, two days waiting for it to be processed, and another day to get the output back, twenty people could simultaneously, at their own terminal, bash away at their programs, get immediate results, and make immediate changes. I imagined that, at least with BASIC, multitasking on the Amiga would be the same sort of thing: you could have one BASIC program running in the background, perhaps comparing two files, while you worked on another in the foreground. The interpreter would be handling both programs on an interrupt basis. It turned out that if you want to run two BASIC programs simultaneously on the Amiga, you have to run two BASIC interpreters."

www.Commodore.ca
May Not Reprint Without Permission

That, in a nutshell, is the 'problem' with multitasking on the Amiga -- it's still essentially a single-user machine. I really like the Amiga because I'm a multitaskin' kinda guy: I may switch many times a day between a word processing program, a terminal, a text editor, a C compiler, a BBS program, and (whispering) a game. It's really nice to have some or all of these going on the same machine at the same time -- I've been able to retire the multitasking desk chair I was using to roll from one computer to another.

But I don't believe I'm a typical user. Most people do one thing at a time on their machines. Using Sidekick or a print spooler is about all they need in the way of multitasking.

That certainly doesn't mean that multitasking on the Amiga can't provide some definite advantages to the ordinary user: it's just that developers must change their traditional view of multitasking. Instead of seeing it as a way to provide completely self-contained, incorruptible environments for several programs running under one MPU, they should try to see it as a way to provide a *communal* environment. As Jim says, "It would be great to able to have a spelling checker program, for example, running as a separate task, that did its job *as you entered the text into your word processor*." This sort of thing is already available in so-called integrated software on other machines. However, typically only one or two modules work really well in these programs (usually because of memory restrictions), and they are not intended to work in a strictly concurrent fashion.

The Amiga could provide the environment for this type of sharing of data, though it would not be a task for careless programmers. The multitasking EXEC still has its roots in mainframe-style multitasking. Messages can be sent to and from tasks, but it's a dodgy business having two programs operating on the same data. Perhaps the best approach is that exemplified by Mimetics's SoundScape MIDI software. Here, the various modules are independent programs that can mesh with the other modules if they are run simultaneously. Mimetics is making available to other developers the structures and formats used by their modules so that these developers can create modules that will also mesh with the Mimetics series.

It is precisely this sort of cooperation between software houses, CBM, and the informed user that resulted in the adoption of IFF (Interchange File Format) for the Amiga. It permits the use by one program of files created by another. In the case of Deluxe Paint and Aegis Images, it has resulted in many artists buying **both** programs -- IFF allows them to take advantage of features one program has that the other lacks. Nobody loses, everybody gains.

Others who gain are the makers of expansion RAM for the Amiga -- Comspec Communications, Allegra, Skyles, RS Data Systems, et cetera. Add-on RAM is fast becoming *the* most popular peripheral for the Amiga, ahead of hard drives and printers and not far behind external floppy drives. Aside from the fact that, until recently, **fast** hard drives were not available, you just couldn't take advantage of multitasking because 512K simply wasn't enough memory. (It's still hard for this C64 user to say that without experiencing a peculiar feeling of vertigo.) Comspec kindly lent me an evaluation unit of their 2 Meg RAM, and let me tell you, it was returned with great reluctance.

This board allowed me to try the following experiment: After booting with the supplied version of Workbench v1.1 (1.2 will autoconfigure the RAM), I used the run command to get BBS-PC!, a bulletin board program for the Amiga, going; then after loading Workbench, I ran Online!, a terminal program, Scribble!, a word processor, and finally, Mind Walker, a wonderful arcade-style game. They all worked, with the following provisos: All the programs loaded after BBS-PC! (v4.04) were drastically slowed. Mind Walker's score sounded like a tape recording played back at slow speed. Scribble's screen updates were eons apart.

That was minor however, compared to the fact that all of Scribbles icons, gadgets, and pointers disappeared! They worked, if you could (by trial and error) find them. You just couldn't see them!

The reason for this is quite simple. The graphics chips can only 'see' the first 512K of memory, or 'chip' memory, as it is called. Shape data for the pointers, gadgets, and indeed any graphic, must reside in chip memory. However, unless otherwise told, the Amiga will load a program into 'fast' memory (if it is available) -- that is, memory above chip memory. To cope with this, programs written with the Lattice C compiler, must be ATOMized: ATOM is a utility that marks which part of a program must be loaded into chip memory, and which can be loaded into fast memory. There is a free upgrade available now for Scribble! that adds spell-checking and mail merge, and Micro-Systems Software tells me that the gadgets are back where they're supposed to be.

Pushing Mind Walker behind the Workbench screen demonstrated that Intuition knows nothing about sprites -- I sat and watched helplessly as a 'bad thought' popped into my CLI window and zapped my current persona as he stood innocently in the Online! window.

Although, BBS-PC! monitors the serial port for a carrier detect, I was still able to dial out with Online!. BBS-PC! appeared to freeze while Online! was using the serial port, but unfroze as soon as the serial port was free.

BBS-PC! also monitors the keyboard, however, and I'm guessing that this is the cause of the drastic slowdown in the other programs. BBS-PC! probably puts itself into a busy loop while waiting for a key to be pressed, rather than calling the EXEC Wait() function. The Wait() function puts the process to sleep until a significant event happens. This would mean that, until a

www.Commodore.ca
May Not Reprint Without Permission

key was pressed, or until it detected a carrier, BBS–PC! would take up almost none of the 68000's processing time, instead of the 30 or 40 per cent it appears to be grabbing now. I'm eager to check the new version of BBS–PC! to see if this has changed.

Expansion RAM is almost a necessity for anyone programming in compiled languages. The speedup offered by the ability to compile and link in RAM is phenomenal compared to floppy disk speeds. Even Alink becomes almost livable with.

And now the news

The hottest news right now is, of course, the imminent arrival of Kickstart and Workbench 1.2. The folks at Commodore–Amiga in Los Gatos had held a wrap party upon the completion of beta 7 -- too early, it seems, for rumour indicates that HQ in West Chester sent it back for some minor cleanup before release in late September or early October.

From what I've seen of 1.2, though, I'm very impressed with the improvements. Using the mount command, I was able to not only use an Amiga 5 1/4 inch drive under AmigaDOS, I was able to partition it into three separate 145K drives! Mount works by looking in the devs directory for a text file called mountlist. Here you specify such things as the number of tracks, the sector interleave, and so on. (The version of 1.2 I played with provided a template mountlist for the 5 1/4 inch drive.) This approach permits the Amiga to use non–standard devices.

Readers of the very first edition of Amiga Dispatches will recall mention of a 68020/68881 board produced by Computer Systems Associates. For $1500 dollars (US) you could pull out your 68000, plug this board into the empty socket, and get a tremendous boost in speed. CSA is now making an expansion chassis for the Amiga called the Turbo Amiga. Inside the box is the 68020/68881 board, a 512K–byte, 32–bit static RAM board, a 20 Meg. hard drive, two empty sockets and a power supply. The price is $5475 (US). The internal single–board version is still available.

The August 4 issue of Infoworld carried an article containing interviews with two users of the Turbo Amiga: one, a materials scientist at MIT, said that programs ran as fast as or faster than those on a VAX 11/780, with virtually the same precision. Likewise, a company in California engaged in 'Star Wars' research for the DOD, has found the Turbo Amiga combination to be more cost effective than a PC–AT to "perform complex graphics transformations for analysis of a jet fighter simulation running on a Harris mainframe."

The Turbo Amiga would really come in handy with the next item. I have received two reports of a product called Caligari, from Octree in New York City. This is a 3D solid–model animation program that apparently produces output equivalent to that of a $50,000 Cubicomp system. The company also has a hardware unit called a frame controller (for a VCR) that permits the recording of a computed frame of animation to one frame on 3/4 inch video tape. Once a sequence has been recorded it can be played back at the appropriate speed. Such a system might be useful for such organizations as film production: expensive or dangerous stunt sequences, for instance, could be envisioned on the Amiga first, before attempting to produce. For further information, contact Roman Ormandy at (212) 921–2119.

I've seen True BASIC in the stores. This latest version was written by the original authors of BASIC, John Kemeny and Thomas Kurtz, and has been ported to the Amiga and the Mac, among others. Rumour indicates that it is faster than AmigaBASIC, easier to edit, has structured programming features including local variables, and the source code is highly compatible with True BASIC on other machines. There are extensions available for 3D graphics and string manipulation, among others, that can be purchased separately, or as part of a package.

The next upgrade of the assembler that comes with the Manx Aztec C compiler will apparently fully support the Metacomco assembler directives and labels. . . The two ROM Kernel Manuals, Libraries & Devices ($52.95 Can.) and The EXEC ($37.75 Can.) are now available from Addison–Wesley. . . A fine programmer–oriented newsletter (the best periodical of its type, in my opinion) called The Amigan: Journeyman and Apprentice is put out by Dick Barnes, who is also editor of the SuperPET Gazette. In particular are two excellent columns written by John Toebes VIII (of Hack! fame) and Joe Bostic (author of Aedit) on C and assembly language respectively. You can become a member of The Amigans by sending $24 (US) if you live in the US or Canada, or $34 (US) if you don't, to The Amigans, P.O. Box 411, Hatteras, North Carolina, US 27943. . . Marble Madness is a lot easier to play with a trackball than a joystick or a mouse, it seems. The Wico trackball is recommended. . .

Finally, I've had a look at two audio digitizers: Futuresound and the Mimetics SoundScape sampler. Both produce high fidelity if brief recordings from either a microphone or line inputs. Futuresound is more expensive at $299.95 (Can.), but it comes with a microphone, a very nice sound editing program (play it backward, forwards, at any speed!), and, to my ears, a lower signal–to–noise ratio. The Mimetics device, $219.95 (Can.), comes with sequencing software, can be used in combination with a MIDI interface, and turns the Amiga keyboard into a musical keyboard. Both products produce IFF sounds for use as instruments in other programs, such as Electronic Arts' Instant Music and Deluxe Music Construction Set. I was impressed by the quality of both products.

I appreciate any comments or questions you may have about topics I have discussed. You can reach me c/o The Transactor, or on Compuserve (71426,1646) or on PeopleLink (AMTAG).

www.Commodore.ca
May Not Reprint Without Permission

Exploring The World Of MFM On The 1571 Disk Drive

M. Garamszeghy
Toronto, Ontario

. . .a combination BASIC and machine language program which allows you to examine virtually any type of MFM disk. . .

The 1571 disk drive is capable of reading a wide variety of foreign disk formats. Unfortunately, custom machine language code is required to access this feature and Commodore did not upgrade the "Display Track and Sector" program on the demo disk to allow you to examine these disks. Listing 1, Display MFM, is a combination BASIC and machine language program which allows you to examine virtually any type of MFM disk. The machine language is POKEd into the cassette and RS-232 buffers. The routines contain several entry points:

Hex	Dec	Function
0B00	2816	Write SEQ Binary File
0B03	2819	Read 256, 512 or 1024 Byte MFM Sector
0B06	2822	Read 128 Byte MFM Sector
0B09	2825	Analyze Disk Format
0C45	3141	Write SEQ File, Convert ASCII to PETSCII

For those who are interested, the assembler source code follows the BASIC listing below. The code follows the routines explained in detail in a series of articles by this author published in TPUG magazine under the title of "A Layman's Guide to Burst Mode" from May to August 1986.

Display MFM will automatically determine the number of sides (1 or 2), the number of bytes per sector (128, 256, 512, or 1024) and the number of sectors per track and the sector numbering system. After a brief pause while the ML is being POKEd into memory, you will be asked to insert the disk to be examined. A few whirs, buzzes and clicks later and the format will be analyzed and displayed on the screen. If the disk is a Commodore GCR disk or an unreadable format (such as APPLE), an error message will be displayed.

For a single sided diskette, you will be asked to enter a track and sector number to examine. The track number must be in the range of 0 to 39 (MFM tracks are numbered starting from 0) and the sector number must be in the range specified by the format analysis. For a double sided disk you will also be asked to enter a side number (1 or 2).

The data will be displayed on the screen in chunks of 128 bytes. Thus a 512 byte sector will require 4 screens to display completely. The 128 byte segment is displayed in 16 lines of the following format:

```
XXXX: FF FF FF FF FF FF FF FF :ABCDEFGH
```

Where XXXX is a hexadecimal number representing the offset from 0 where the data are located on the sector. FF, etc., are the hex values of the data bytes, and ABC, etc., are the ASCII characters associated with each byte. Unprintable characters are represented by a period ("."). The data display is followed by the the message "press any key to continue".

At this point there are several special keys you can press. These are:

<escape>	to abort the current sector and return to the select (side), track and sector screen.
<cursor up>	increment the track# (sector & side stay same)
<cursor down>	decrement the track# (sector & side stay same)
<cursor right>	increment the sector# (track & side stay same)
<cursor left>	decrement the sector# (track & side stay same)
c	capture the contents of the sector to the 50k byte capture buffer
k	kill the contents of the capture buffer
s	switch sides (double sided disks only) (track & sector stay the same)
w	write capture buffer to a C-128 GCR data file (SEQ type). You will be asked to enter a file name. A null file name (i.e just <return>) will return to the select (side), track, sector screen. You then put the C-128 disk in the drive. Just before the file is written, you will be asked to select either a PETSCII or ASCII file. The write operation does not automatically kill the buffer. This must be done manually, if desired with the k command outlined above.

Any other key (including s for single sided disks) will display the next 128 byte segment. When the entire segment has been displayed, the program will return to the select (side), track, sector screen. The c and k keys return to the select (side), track, sector screen. The w key returns to the initial "insert disk to be examined" screen.

Some words of caution:

1. The captured sectors will be in ASCII not PETSCII. If they are text files, you should use the <P>ETSCII option for the write. This will create a standard PETSCII file from the ASCII data. The <A>SCII option will give you exactly what you see on the screen with no conversion.

2. Be careful with what you are doing. The techniques required to read MFM disks are NOT very tolerant of stupid errors such as removing the disk during a read, etc. Follow the prompts on the screen and do not insert a new disk unless it tells you to. These types of errors may cause the C-128 to lock up in such a fashion that <run-stop>-<restore> may not work. (Keyboard interrupts are temporarily disabled during certain segments of the ML code.)

Table 1 is a summary of some of the common MFM disk formats. The list is by no means complete, but can be used as a guide when exploring various types of MFM disks. It is worth noting that many other brands of computers use formats similar to those outlined in the table.

www.Commodore.ca
May Not Reprint Without Permission

Table 1: Summary Of 1571 Supported CP/M MFM Disk Formats

Format Name	# Sides	Sector Size	Sector# Range	AU Size	Total Capacity	Data Capacity	# Directory Entries	Directory Starts At Side	Track	Sector	Data Area Starts At Side	Track	Sector
CP/M Formats:													
OSBORNE DD	1	1024	1 – 5	1K	200K	183K	64	0	3	1	0	3	3
SLICER	2	512	1 – 8	2K	320K	314K	64	0	1	1	0	1	5
EPSON EURO (SD)	2	256	1 – 16	2K	320K	284K	128	0	4	1	1	4	1
EPSON QX-10 (SD)	2	256	1 – 16	2K	320K	300K	128	0	2	1	1	2	1
EPSON QX-10 I (DD)	2	512	1 – 10	2K	400K	376K	128	0	2	1	0	2	9
IBM CP/M-86 SS	1	512	1 – 8	1K	160K	154K	64	0	1	1	0	1	5
IBM CP/M-86 DS	2	512	1 – 8	2K	320K	314K	64	0	1	1	0	1	5
KAYPRO II	1	512	0 – 9	1K	200K	193K	64	0	1	0	0	1	4
KAYPRO IV (* = "side#")	2	512	*0: 0 – 9 *1: 10 – 19	2K	400K	390K	128	1	0	10	1	0	19
Other MFM Formats:													
IBM-PC-DOS:													
1 side; 8 sector	1	512	1 – 8	.5K	160K	157K	64	0	0	4	0	0	8
2 side; 8 sector	2	512	1 – 8	1K	320K	314K	112	0	0	4	1	0	4
1 side; 9 sector	1	512	1 – 9	.5K	180K	175K	64	0	0	6	0	1	1
2 side; 9 sector	2	512	1 – 9	1K	360K	354K	112	0	0	6	1	0	4
TRS-80 DD	1	256	1 – 18		180K		64	0	17	1	0	0	2
TRS-80 SD	1	256	1 – 10		100K		64	0	17	1	0	0	2

Listing 1: Display MFM Disks

```
FO 1000 rem save"0:display mfm",8
DB 1010 rem ** written by m. garamszeghy, toronto, ontario
NC 1020 rem ** for use with the commodore c128 and
          1571 drive
BE 1030 rem ** will determine disk format and display data
          if mfm
EI 1040 :
II 1050 e1=2816: rem write seq binary file
GN 1060 e2=e1+3: rem read 256, 512 or 1024 byte mfm
          sector
HP 1070 e3=e2+3: rem read 128 byte mfm sector
JO 1080 e4=e3+3: rem analyse disk format
MA 1090 e5=3141: rem write seq file, convert ascii to petscii
JB 1100 d2$=chr$(13)+chr$(13): hd$=chr$(19)+
          d2$+d2$
HM 1110 cp$=chr$(17)+chr$(27)+chr$(29)+"cksw"+
          chr$(145)+chr$(157): rem ctrl chars
EN 1120 :
KD 1130 bank 15: color 0,7: color 4,7: color 5,2
          : print chr$(14)chr$(11)
PI 1140 gosub1610: bp=13056
CM 1150 print d2$ tab(7)"* please wait *": ca=0
GM 1160 gosub 1810: gosub 1610: rem move in code then
          display intro
IG 1170 print d2$"insert disk to examine then"
          : gosub1540: sd=2: io=1: gosub1560
FF 1180 x=peek(3072): if x<2 then print d2$"gcr disk"
          : gosub1540: goto1170
GL 1190 ss=0: bs=x and 48: if bs=0 then ss=128: else if
          bs=16 then ss=256
FP 1200 if bs=32 then ss=512: else if bs=48 then
          ss=1024
HN 1210 print d2$"mfm disk: ";sd"side(s)"
          : print d2$;ss"bytes / sector";
PB 1220 ts=peek(3074): print "; "ts"sectors / track"
FE 1230 sl(1)=peek(3076): sh(1)=peek(3077)
KK 1240 print "side 1: min sector #"sl(1)" max sector
          #"sh(1): if sd=1 then 1270
LF 1250 sl(2)=peek(3084): sh(2)=peek(3085)
NI 1260 print "side 2: min sector #"sl(2)" max sector
          #"sh(2)
JH 1270 si=1: poke 208,0: print d2$
NA 1280 if sd=1 then input "track,sector";t,s: b1=64
          : goto1300
IH 1290 input "side,track,sector";si,t,s: b1=64: if si=2
          then b1=80
GM 1300 if si<1 or si>sd or t>39 or t<0 or s<sl(si) or s>sh(si)
          then 1650
FJ 1310 open 15,8,15,"u0"+chr$(b1)+chr$(t)+
          chr$(s)+chr$(1)
IM 1320 bl=bp-int(bp/256)*256
CM 1330 if ss=128 then sys e3,bl,bp/256: else sys e2,
          ss/256,bl,bp/256
DO 1340 dclose: gosub1560
MF 1350 gosub1610
DG 1360 for i=bp to bp-10+ss step128: gosub1630
NN 1370 for j=0to127 step8: s$="": ad=i+j-bp
          : ah$=hex$(ad): print ah$": ";
LK 1380 for k=0to7: z=i+j+k: z$=right$(hex$(peek(z)),2)
          : print z$" ";
HL 1390 if peek(z)>31 and peek(z)<128 then
          a$=chr$(peek(z)): else a$="."
IK 1400 s$=s$+a$: next: print ":"s$: next: gosub1540
FH 1410 a=asc(a$): if instr(cp$,a$,1)=0 then 1530
          : rem mask out non-control chars
DH 1420 if a=27  then 1210: rem <esc>
HH 1430 if a=87  then 1660: rem 'w' (write)
FJ 1440 if a=145 then t=t+1: if t>39 then t=0
          : rem <cursor up>
GB 1450 if a=17  then t=t-1: if t<0  then t=39
          : rem <cursor down>
```

www.Commodore.ca
May Not Reprint Without Permission

```
PA  1460 if a=29  then s=s+1: if s>ts then s=1
         : rem <cursor right>
BE  1470 if a=157 then s=s-1: if s<1 then s=ts
         : rem <cursor left>
PP  1480 if a=67  then 1730: rem 'c' (capture)
LK  1490 if a=75  then 1770: rem 'k' (kill)
EF  1500 if a=83 and sd=2 then si=si-1: b1=64
         : if si<1 then si=2: b1=80: rem 's'
PD  1510 if a=83 and sd=1 then 1530: rem 's' (side)
JH  1520 dclose: goto1310
IL  1530 next: goto1210
DN  1540 print d2$"press a key to continue": poke 208,0
         : getkeya$: goto1610
FG  1550 print d2$chr$(18)"disk error >> "ds$: goto1540
JI  1560 close15: open15,8,15: if ds then gosub1550
         : goto1170
LO  1570 print#15,"u0"+chr$(10): sys e4,0
EB  1580 if io then close15: open15,8,15,"u0"+chr$(26)
         : sys e4,8: dclose: io=0
EG  1590 if ds then sd=1: close15: open15,8,15,"u0"
         +chr$(10): dclose
MF  1600 return
HA  1610 print chr$(147)" ** 1571 display mfm t&s v2 **"
ME  1620 print "  by M. Garamszeghy 86-05-01": print
         : return
IK  1630 char,1,24,"side >"+str$(si)+" track >"
         +str$(t)+" sector >"+str$(s)+hd$
EI  1640 return
AK  1650 print d2$ "illegal sector": gosub1540: goto1210
HC  1660 f$="": print d2$: input"file name to save ";f$
FC  1670 if f$="" then 1210: else print d2$"insert c-128
         disk then ": gosub1540
BC  1680 input "<p>etscii or <a>scii";ft$
KK  1690 print d2$"writing file >> "f$
         : open 8,8,8,"0:"+f$+",s,w"
EB  1700 if ds then print d2$"disk error >> "ds$
         : gosub1540: dclose: goto1660
EO  1710 if ft$="p" then poke 252,bp/256+1
         : sys e5,8,0,51: dclose: goto1170
OJ  1720 poke 252,bp/256+1: sys e1,8,0,51: dclose
         : goto1170
PE  1730 ca=ca+1: bp=bp+ss
GK  1740 if bp>65024 then print: print "buffer full"
         : gosub1540: goto1210
BE  1750 print d2$"side"si" track"t" sector"s: print
DF  1760 print "captured": print d2$;ca"sectors captured
         total": sleep3: goto1210
ME  1770 print: input"kill buffer (y/n)";kb$: if kb$<>"y"
         then 1350
BB  1780 bp=13056: print d2$"buffer killed": sleep3
         : ca=0: goto1210
CH  1790 :
CF  1800 rem ** code for mfm disk procedures **
MD  1810 ch=0: for j=2816 to 3045: read x: poke j,x
         : ch=ch+x: next
ON  1820 if ch<>27222 then print"checksum error!": stop
ML  1830 goto 2160: rem move the balance of the code also
EK  1840 :
BJ  1850 data  76, 185,  11,  76, 141,  11,  76,  70
OD  1860 data  11, 133, 250, 162,  12, 134, 251, 160
IE  1870 data   0, 120,  44,  13, 220,  32, 129,  11
IN  1880 data  32,  97,  11, 201,   2, 144,  23,  41
JA  1890 data  14, 208,  19,  32,  97,  11,  41,  14
NO  1900 data 208,  22,  32,  97,  11,  32,  97,  11
PL  1910 data  32,  97,  11,  32,  97,  11,  88,  32
AL  1920 data 204, 255, 169,   8,  32,  74, 255,  96
PI  1930 data 142,   1,  12,  76,  54,  11, 133, 250
GK  1940 data 134, 251, 160, 255, 162,   0, 142,   0
DJ  1950 data 255, 120,  44,  13, 220,  32, 129,  11
ON  1960 data  32,  97,  11, 192, 128, 208, 249,  88
PO  1970 data  96, 169,   8,  44,  13, 220, 240, 251
NJ  1980 data 173,   0, 221,  73,  16, 141,   0, 221
GO  1990 data 173,  12, 220, 162,  63, 142,   0, 255
BN  2000 data 145, 250, 162,   0, 142,   0, 255, 200
OJ  2010 data  96, 173,   0, 221,  73,  16, 141,   0
BP  2020 data 221, 173,  12, 220,  96, 133, 252, 134
FP  2030 data 250, 132, 251, 160,   0, 162,   0, 142
ML  2040 data   0, 255, 120,  44,  13, 220,  32, 129
OG  2050 data  11,  32,  97,  11,  41,  14, 208,  15
DK  2060 data 160,   0,  32,  97,  11, 192,   0, 208
KE  2070 data 249, 230, 251, 198, 252, 208, 243,  88
HF  2080 data  96, 133, 253, 134, 250, 132, 251, 162
BA  2090 data   0, 142,   0, 255, 166, 253,  32, 201
KA  2100 data 255, 160,   0, 162,  63, 142,   0, 255
IC  2110 data 177, 250, 162,   0, 142,   0, 255,  32
KF  2120 data 210, 255, 200, 208, 238, 230, 251, 165
IJ  2130 data 252, 197, 251, 208, 230,  96
AN  2140 :
--  2150 rem code to write seq file, convert ascii to petscii
KJ  2160 ch=0: for j=3141 to 3228: read x: poke j,x
         : ch=ch+x: next
AD  2170 if ch<>12780 then print "checksum error!": stop
AK  2180 return
CA  2190 :
MM  2200 data 133, 253, 134, 250, 132, 251, 166, 253
KM  2210 data  32, 201, 255, 160,   0, 162,  63, 142
FJ  2220 data   0, 255, 177, 250, 162,   0, 142,   0
NL  2230 data 255, 133, 254, 201,  10, 240,  37, 201
JJ  2240 data  26, 240,  48, 201,  64, 240,  26,  41
CH  2250 data 192, 240,  20, 165, 254,  41,  32, 208
HB  2260 data   7, 165, 254,   9, 128,  76, 134,  12
HB  2270 data 165, 254,  41,  95,  76, 134,  12, 165
NA  2280 data 254,  32, 210, 255, 200, 208, 198, 230
PP  2290 data 251, 166, 251, 228, 252, 208, 190,  32
CD  2300 data 204, 255,  96, 169,   0,  76, 134,  12
```

Display MFM: PAL Source Code

```
GG  1000 rem save"0:1571 mfm 1.pal",8
IL  1010 rem ** m. garamszeghy - toronto, ontario
IF  1020 rem ** allows access to most mfm diskettes by
KE  1030 rem ** using the commodore c128 with 1571 drive
EI  1040 :
FE  1050 open 8,8,1,"0:1571 mfm 1.obj"
FP  1060 sys 700
BF  1070 .opt o8
KD  1080 *=$0b00
IL  1090 ;
AL  1100 clkout   =   %00010000 ;to change state of clock
PN  1110 ptr      =   $fa       ;(pointer) for storage/
                                 retrieval of data in ram
DJ  1120 count    =   $fc       ;block count
PG  1130 logadd   =   $fd       ;logical address
FP  1140 flag     =   $0c01
EM  1150 dlsdr    =   $dc0c     ;serial data register
```

www.Commodore.ca
May Not Reprint Without Permission

```
JE 1160 dlicr    =   $dc0d    ;interrupt control register
LG 1170 d2pra    =   $dd00    ;serial port 6526 cia 2
FL 1180 mmucon   =   $ff00    ;mmu control port
MG 1190 eainit   =   $ff4a    ;set standrd i/o devices
FK 1200 chkout   =   $ffc9    ;set output device
OP 1210 clrchn   =   $ffcc    ;clear all channels
KN 1220 chrout   =   $ffd2    ;output a char
EE 1230 ;
BB 1240 ;** jump table to keep it simple **
FM 1250          jmp wrtseq   ;write seq binary file
AA 1260          jmp rd256    ;read 256, 512 or 1024
                              byte mfm sector
MK 1270          jmp rd128    ;read 128 byte mfm sector
GH 1280 ;
NN 1290 ;** analyse disk format **
DO 1300          sta ptr      ;retain .a
NP 1310          ldx #12
BP 1320          stx ptr+1
NH 1330          ldy #0
FN 1340          sei
HE 1350          bit dlicr    ;clear interrupt control
                              register
AH 1360          jsr chkmod   ;check mode (gcr/mfm)
EE 1370          jsr readit
IJ 1380          cmp #2
FF 1390          bcc return
OO 1400 ;
FP 1410          and #%00001110 ;$e
MJ 1420          bne return
MA 1430 ;
KI 1440          jsr readit
NB 1450          and #%00001110 ;$e
JO 1460          bne setflg   ;set flag then return
ED 1470 ;
CL 1480          jsr readit
ML 1490          jsr readit
GM 1500          jsr readit
AN 1510          jsr readit
GG 1520 ;
MN 1530 return   =   *
CJ 1540          cli
AJ 1550          jsr clrchn   ;clear all channels
DB 1560          lda #8
EN 1570          jsr eainit   ;set standard i/o devices
IB 1580          rts
MK 1590 ;
MC 1600 setflg   =   *
JA 1610          stx flag
NJ 1620          jmp return
EN 1630 ;
EA 1640 ;** read 128 byte mfm sector **
OA 1650 rd128    =   *
AB 1660          sta ptr
PE 1670          stx ptr+1
NK 1680          ldy #$ff
BO 1690          ldx #0
FM 1700          stx mmucon   ;set to normal config
HE 1710          sei
AE 1720          bit dlicr    ;interrupt control register
CO 1730          jsr chkmod   ;check mode (gcr/mfm)
CE 1740 ;
JM 1750 getmor   =   *
KM 1760          jsr readit
PL 1770          cpy #128
BM 1780          bne getmor
EH 1790 ;
GJ 1800          cli
OP 1810          rts
CJ 1820 ;
JG 1830 readit   =   *
LC 1840          lda #8
AL 1850 ;
PG 1860 wait1    =   *
GE 1870          bit dlicr    ;wait for byte
DO 1880          beq wait1
IN 1890 ;
AG 1900          lda d2pra    ;read serial port
LA 1910          eor #clkout  ;change state of clock
FM 1920          sta d2pra    ;store back
EC 1930          lda dlsdr    ;get data from serial data
                              register
OI 1940          ldx #%00111111 ;ram 0 and kernal
FJ 1950          stx mmucon   ;set as config
AI 1960          sta (ptr),y  ;store status
JP 1970          ldx #0
NF 1980          stx mmucon   ;back to normal config
GK 1990          iny
ML 2000          rts
AF 2010 ;
JJ 2020 chkmod   =   *
CO 2030          lda d2pra    ;read serial port
NI 2040          eor #clkout  ;change state of clock
HE 2050          sta d2pra    ;store back
GK 2060          lda dlsdr    ;get data from serial data
                              register
CA 2070          rts
GJ 2080 ;
NO 2090 ;** read 256, 512 or 1024 byte mfm sector **
DN 2100 rd256    =   *
HE 2110          sta count    ;max # blocks
JF 2120          stx ptr      ;(ptr)
PB 2130          sty ptr+1
HK 2140          ldy #0
NK 2150          ldx #0
BJ 2160          stx mmucon   ;set to normal config
DB 2170          sei
MA 2180          bit dlicr    ;interrupt control register
OK 2190          jsr chkmod   ;check mode (gcr/mfm)
CI 2200          jsr readit
FB 2210          and #%00001110 ;$e
GA 2220          bne nomore
MC 2230 ;
LA 2240          ldy #0
AE 2250 ;
MJ 2260 more     =   *
IM 2270          jsr readit
FE 2280          cpy #0
AP 2290          bne more
CH 2300 ;
IG 2310          inc ptr+1    ;high byte+1
GL 2320          dec count    ;decrease count of blocks
IB 2330          bne more
KJ 2340 ;
LO 2350 nomore   =   *
GM 2360          cli
OC 2370          rts
```

www.Commodore.ca
May Not Reprint Without Permission

```
CM  2380 ;
KO  2390 ;** write seq binary file **
IJ  2400 wrtseq    =    *
NH  2410           sta  logadd  ;logical write address
FI  2420           stx  ptr     ;(ptr)
LE  2430           sty  ptr+1
PM  2440           ldx  #0
DL  2450           stx  mmucon  ;set to normal config
AB  2460           ldx  logadd  ;la
HJ  2470           jsr  chkout  ;set output device
LP  2480           ldy  #0
AD  2490 ;
LC  2500 wrtmor    =    *
IM  2510           ldx  #%00111111 ;ram 0 and kernal
PM  2520           stx  mmucon  ;set as config
PL  2530           lda  (ptr),y ;get data from ram
DD  2540           ldx  #0
HJ  2550           stx  mmucon  ;back to normal config
LF  2560           jsr  chrout  ;write data
KO  2570           iny
BH  2580           bne  wrtmor
EJ  2590 ;
HH  2600           inc  ptr+1
GA  2610           lda  count
EM  2620           cmp  ptr+1   ;have we hit the end yet
OK  2630           bne  wrtmor  ;more to go
GM  2640 ;
GE  2650           rts
KN  2660 ;
KE  2670 .end
```

Display MFM: ASCII to PETSCII Conversion Source

```
HG  1000 rem save"0:1571 mfm 2.pal",8
IL  1010 rem ** m. garamszeghy - toronto, ontario
CB  1020 rem ** ascii to petscii conversion routine
KH  1030 :
OD  1040 open 8,8,1,"0:1571 mfm 2.obj"
LO  1050 sys 700
HE  1060 .opt o8
CF  1070 *=$0c45
OK  1080 ;
OP  1090 ptr       =    $fa     ;(pointer) to data in ram
EB  1100 count     =    $fc     ;count of blocks to print
LF  1110 logadd    =    $fd     ;logical address
MD  1120 work      =    $fe     ;keep data byte
HN  1130 mmucon    =    $ff00   ;mmu control
JG  1140 chkout    =    $ffc9   ;set output device
CM  1150 clrchn    =    $ffcc   ;clear all channels
OJ  1160 chrout    =    $ffd2   ;output a char
IA  1170 ;
LH  1180 ;** write seq file - convert ascii to petscii **
JL  1190           sta  logadd  ;logical write address
EB  1200           stx  ptr     ;(pointer) through ram
HI  1210           sty  ptr+1
CK  1220           ldx  logadd
PL  1230           jsr  chkout  ;set output device
DC  1240           ldy  #0
IF  1250 ;
FG  1260 loop      =    *
AP  1270           ldx  #%00111111 ;ram 0 and kernal
HP  1280           stx  mmucon  ;set as config
KA  1290           lda  (ptr),y
LF  1300           ldx  #0
NN  1310           stx  mmucon  ;set back to normal config
JA  1320           sta  work    ;retain data in work area
LG  1330           cmp  #10     ;line feed
BL  1340           beq  noshow  ;skip display
ML  1350 ;
NA  1360           cmp  #26     ;'sub'
ON  1370           beq  sndnul  ;send null instead
KN  1380 ;
OE  1390           cmp  #64     ;'@'
NP  1400           beq  showit  ;ok - print it
IP  1410 ;
CE  1420           and  #%11000000 ;test bits 7+6
DI  1430           beq  flash   ;no prob - just display
GB  1440 ;
KK  1450           lda  work
OM  1460           and  #%00100000 ;test for bit 5
CN  1470           bne  maskit  ;needs conversion before
                                 display
OD  1480 ;
CN  1490           lda  work
HM  1500           ora  #%10000000 ;set reverse flag
OL  1510           jmp  showit
GG  1520 ;
AP  1530 maskit    =    *
EA  1540           lda  work
GC  1550           and  #%01011111 ;display mask
AP  1560           jmp  showit
IJ  1570 ;
JE  1580 flash     =    *
GD  1590           lda  work
GL  1600 ;
NF  1610 showit    =    *
NP  1620           jsr  chrout  ;output a char
EN  1630 ;
MJ  1640 noshow    =    *
CF  1650           iny
LN  1660           bne  loop    ;go for some more
MP  1670 ;
PN  1680           inc  ptr+1
FC  1690           ldx  ptr+1
GO  1700           cpx  count
KH  1710           bne  loop    ;more to go
OC  1720 ;
EE  1730           jsr  clrchn  ;clear all channels
IL  1740           rts
ME  1750 ;
AN  1760 sndnul    =    *
FN  1770           lda  #0
MM  1780           jmp  showit
EH  1790 ;
EO  1800 .end
```

www.Commodore.ca
May Not Reprint Without Permission

C64 Mini–Tracer

Jim Frost
La Mesa, California

A Trace Utility For The C64 That Works In Low and High Res Mode

Mini–Tracer is a short machine language wedge utility that allows single step operation of conventional and HIRES BASIC programs. The current line number is displayed in the lower right-hand corner of the screen. The trace routine is located at $CB80 (52096) as this area of memory is seldom used by the short machine language routines often included with BASIC programs. Mini–Tracer is not compatible with most (if any) of the DOS wedge programs, and should not be loaded when a wedge is in use.

Mini–Tracer was first written several years ago, when magazines did not publish verifizer programs. In those days, even a minor typing error could lead to a system crash that took days to locate. With Mini–Tracer, the cause of a crash can usually be located in minutes.

As my computing skills increased and published programs became more complex, Mini–Tracer was rewritten to include single–step and HIRES trace modes. The current version is most useful in finding out how a BASIC program works or why it doesn't. Program logic flow can be traced for various input conditions and the effect of each BASIC line on screen action can be easily observed. In programming or debugging, there is no substitute for planning and logical thinking; however, Mini–Tracer provides a useful tool that allows you to concentrate your thinking on an isolated subroutine, an incorrect variable, or a few faulty lines of code.

To start Mini–Tracer, load and run the loader program. In about 20 seconds, the machine language will be poked into place and BASIC line numbers will start displaying. BASIC programs can then be loaded and run normally, except that program flow is traced. To toggle single–step on, press the Commodore key. BASIC will now execute one line each time any key is pressed. Normally, you should press the shift or control keys to prevent filling the keyboard buffer with gibberish. When the program requires an input, use the standard keyboard. Keys can be held down for a very slow execution of BASIC lines. Single–step can also be controlled from within a program. Just add POKE 52232,1 to any BASIC line to start single–step. Pressing the Commodore key a second time (or POKE 52232,0) will disable single–step. STOP is sluggish when single–step is enabled but the computer will respond if the stop key is held for a few seconds. When a BASIC program is stopped while in single–step, the first command in direct mode must be followed by pressing an additional key after return. Additional direct mode commands work normally.

Programs with custom characters present a problem since the line number may consist of alien pieces of dragon tails instead of readable numbers. To prevent this, locate the line which selects the new character set and temporarily replace the POKE to 53272 with POKE 53272,21. The aliens will look like ones and twos but that can be fixed when the bugs are squashed.

Technical Details

The remaining text describes the operation of the program. If you are interested only in using Mini–Tracer to understand and debug BASIC programs, stop reading at this point. If you are interested in studying machine language or modifying Mini–Tracer to suit your needs, then the assembly listing and the remaining text will be of interest.

Mini–Tracer consists of five main modules: Initialization, Control, Formatting, Standard Display and Bit Map Display. The initialization routine sets up the wedge, then pokes screen and color memory with the title page and instructions. The control module checks line numbers, flags, and key presses to direct program flow. Conversion of the line numbers from HEX to screen display characters is handled by the formatting module. Each module will be functionally described. All addresses are given in hexadecimal. Those addresses that can be used from BASIC have the corresponding decimal address following in parenthesis.

Initialization

A routine called CHRGET is used by BASIC to gather individual characters from the BASIC program. The characters are interpreted and commands are then executed. Mini–Tracer (and many other wedge utilities) works by placing a jump in the middle of CHRGET to divert the program to the new code. When the new job is done, the program jumps back to finish CHRGET. The main loop of Mini–Tracer starts at $CC7E, so the initialization routine pokes CHRGET with JMP $CC7E.

Control

Since CHRGET is entered for each byte of BASIC program, executing a long wedge slows BASIC considerably. To keep BASIC as fast as possible, Mini–Tracer first checks the BASIC line number at $39 and $3A (57 and 58) against the previous line number at $CC09 and $CC0A. When the line numbers are different, the present line number is saved and the remainder of the trace routine is executed.

Single Step is controlled by a flag at $CC08. If the flag is off ($CC08=0), the program will execute at maximum speed. Before

www.Commodore.ca
May Not Reprint Without Permission

testing flags, the status of special keys determined by testing SHFLAG at $028D. If the Commodore key is pressed ($0280=2) the single step flag is toggled. When the single step flag is off, the program jumps directly to the number formatting. When the single step flag is on, the program checks for standard or special keys pressed (standard key memory is $C5 (197)). If no keys are pressed, the program keeps looping until a key is pressed. When a key press is found, a delay timer is started (ML is so fast that without a delay, several lines will execute before you can lift your fingers). The delay is timed by counting zero transitions of the raster position at $DO12 (53266). The raster completes a full scan in 1/60 second so it is changing much too fast to be very useful with BASIC, yet there is time for several thousand machine language instructions. Waiting 96 raster scans provides approximately 1.5 seconds delay.

Formatting

Converting the line number from binary to decimal utilizes a technique described by Jim Butterfield in Compute! (July 83). The method involves alternately adding (in decimal mode), then multiplying by the base. Converting this way will work with any number system, as long as you remember to multiply by the correct base.

Prior to handling the details of screen printing, the formatting module checks the screen location. Usually, the screen is at $0400 (1024), but the program being traced might have a different screen location or use screen flipping for animation. The screen location is calculated by adding the selected bank (determined by the lowest 3 bits of $DDOO (56576) to the screen base address at $0288 (648). An additional 3 is added to the high byte of the screen address to place the line number at the bottom of the screen.

Display of the standard screen line numbers is handled by the subroutine labeled NOBIT on the listing. Each byte of a BCD number contains two decimal numbers, one each in the high and low nibble. These are separated and $30 (48) is added to convert the numbers to screen codes. The converted numbers are then poked on the screen. When this is finished, Mini-Tracer pulls the original A and X registers from the stack and goes back to CHRGET.

The bit map output was a bit trickier. With bit map, individual pixels produce the display, so characters cannot be poked directly to the screen. To display the numbers, I could have gathered the required 8 bits from the standard character set and poked them on the screen. Since I had to handle 8 bits per number anyway, I decided to design a custom set of numbers which would work with multicolor also. The data for these is given (in HEX) in the assembly listing, should you want to use them with your own multicolor programs.

The next obstacle was locating the 8K bitmap screen. The bitmap screen can be located at the beginning or middle of four different 16K banks. The eight possible screen addresses are found in a table called BANKTAB. The bank is determined by placing the low 3 bits of $DDOO in the X register, then checking bit 8 of $DO18 to find the bank half in use and adding $4 to x when the screen is in the high half of the bank. With the screen located, another $1F (31) is added to position the line numbers at the bottom of the screen.

The 8 bits for a desired number are found by multiplying the number by 8 then using the product as an index to the correct position in the character table. Each byte is then poked to the bit mapped screen. After printing the line number on the hires screen, Mini-Tracer returns to the Basic interpreter by jumping back to CHRGET.

Mini-Tracer: BASIC Loader

```
KB  1000 rem save "0:trace44.ldr",8
EK  1010 rem ** minitracer - trace/single step routine for
NP  1020 rem ** basic programs and bit map - c64
ME  1030 rem ** written by: jim frost - rev. 12/12/85
NJ  1040 for j=52096 to 52904: read x: poke j,x
        : ch=ch+x: next
NA  1050 if ch<>89485 then print "checksum error!"
        : stop
CA  1060 print "sys(52096): rem to enable": end
HJ  1070 data 141, 137, 142, 137,  32,  32,  32, 160
PF  1080 data  32,  98,  95, 160, 105,  98,  95, 160
NJ  1090 data 105,  98,  95, 160,  32,  98, 254, 160
CG  1100 data  32,  98,  95,   0, 160,  32, 160, 160
GG  1110 data  32, 226,  32, 160,  32, 226,  32, 160
BG  1120 data  32, 160, 160, 160,  32, 226, 160, 160
DG  1130 data  32, 226,  32,   0, 160,  32, 160, 160
FN  1140 data  32, 160, 123, 160,  32, 160,  32, 160
LH  1150 data 223, 226, 233, 160,  32, 226, 251, 160
LL  1160 data  32, 160, 123,   0,  16,  18,   5,  19
IH  1170 data  19,  32,   3,  61,  32,  20,  15,  32
BC  1180 data  20,  15,   7,   7,  12,   5,  32,  19
ME  1190 data   9,  14,   7,  12,   5,  45,  19,  20
JP  1200 data   5,  16,  16,  18,   5,  19,  19,  32
GI  1210 data  19,   8,   9,   6,  20,  32,  15,  18
JO  1220 data  32,   1,  14,  25,  32,  11,   5,  25
JL  1230 data  32,  20,  15,  32,  19,  20,   5,  16
GK  1240 data   0,   0,   0,   0,   0,   0,   0,   0
DL  1250 data   0,   0,   0,   0,   0,   0,   1,   0
IM  1260 data   1,   0,   1,   1,   1,   0,   1,   1
IP  1270 data   2,   1,   1,   0,   1,   0,  63,  51
LC  1280 data  51,  51,  51,  51,  63,  63,  60,  60
KB  1290 data  12,  12,  12,  12,  63,  63,  63,  51
NA  1300 data   3,  63,  48,  51,  63,  63,  63,  51
AE  1310 data   3,  15,   3,  51,  63,  63,  51,  51
CE  1320 data  51,  63,   3,   3,   3,   3,  63,  48
KE  1330 data  48,  63,   3,  51,  63,  63,  63,  51
LI  1340 data  48,  63,  51,  51,  63,  63,  63,  51
PF  1350 data   3,   3,   3,   3,   3,   3,  63,  51
GI  1360 data  51,  63,  51,  51,  63,  63,  63,  51
ON  1370 data  51,  63,   3,  51,  63,  63, 192, 128
CK  1380 data  64,   0, 224, 160,  96,  32,  72, 138
AK  1390 data  72, 162,   0, 165,  57, 205,   9, 204
MF  1400 data 240,   4, 232, 141,   9, 204, 165,  58
HG  1410 data 205,  10, 204, 240,   4, 232, 141,  10
EH  1420 data 204, 224,   0, 208,   3,  76, 123, 205
AP  1430 data 162,   3, 181, 251, 157,  17, 204, 202
EM  1440 data  16, 248, 173, 141,   2, 201,   2, 208
AN  1450 data  13, 173,   8, 204,  73,   1, 141,   8
BL  1460 data 204, 173, 141,   2, 208, 251, 173,   8
PB  1470 data 204, 240,  45, 165, 197, 201,  64, 208
AL  1480 data   5, 173, 141,   2, 240, 220, 162,  48
HF  1490 data 173,  18, 208, 208, 251, 173,  18, 208
```

www.Commodore.ca
May Not Reprint Without Permission

```
MN 1500 data 240, 251, 173, 141,   2, 201,   2, 208
GA 1510 data  12, 169,   0, 141,   8, 204, 173, 141
CM 1520 data   2, 208, 251, 240,   3, 202, 208, 224
FA 1530 data 162,   2, 181,  56, 157,  14, 204, 169
BE 1540 data   0, 157,  11, 204, 202, 208, 243, 141
EP 1550 data  11, 204, 162,  15,  14,  15, 204,  46
ME 1560 data  16, 204, 120, 248, 160,   2, 185,  11
GB 1570 data 204, 121,  11, 204, 153,  11, 204, 136
CN 1580 data  16, 244, 216,  88, 202,  16, 229, 173
IO 1590 data 136,   2, 133, 252, 173,   0, 221,  41
MI 1600 data   3, 170, 189, 118, 204,  24, 101, 252
EC 1610 data 105,   3, 133, 252, 169, 224, 133, 251
NJ 1620 data 173,  17, 208,  41,  32, 240,   3,  76
JC 1630 data 136, 205, 162,   0, 160,   0, 189,  11
NM 1640 data 204,  72,  74,  74,  74,  74,   9,  48
CG 1650 data 145, 251, 200, 104,  41,  15,   9,  48
EH 1660 data 145, 251, 232, 200, 224,   3, 208, 230
EJ 1670 data 162,   6, 173,  33, 208,  41,  15, 168
OM 1680 data 185,  22, 204, 157, 223, 219, 202, 208
AC 1690 data 250, 162,   3, 189,  17, 204, 149, 251
CP 1700 data 202,  16, 248, 104, 170, 104, 201,  58
GN 1710 data 176,   3,  76, 128,   0,  76, 138,   0
NO 1720 data 173,  24, 208,  41,   8, 240,   4, 232
BL 1730 data 232, 232, 232, 189, 118, 204,  24, 105
JH 1740 data  31, 133, 254, 169,   0, 133, 253, 169
IL 1750 data   0, 141,  21, 204, 160,   0, 174,  21
NP 1760 data 204, 189,  11, 204,  72,  41, 240,  74
OB 1770 data 170, 189,  38, 204, 145, 253, 232, 200
LA 1780 data 192,   8, 240,  10, 192,  24, 240,   6
HB 1790 data 192,  40, 240,   2, 208, 235, 104,  41
IA 1800 data  15,  10,  10,  10, 170, 189,  38, 204
ND 1810 data 145, 253, 232, 200, 192,  16, 240,  10
EG 1820 data 192,  32, 240,   6, 192,  48, 240,   7
KC 1830 data 208, 235, 238,  21, 204, 208, 191, 160
LD 1840 data   5, 169,  16, 145, 251, 136,  16, 251
MN 1850 data  76,  96, 205, 169,  76, 133, 124, 169
DF 1860 data 126, 133, 125, 169, 204, 133, 126, 162
KL 1870 data   1, 173,  33, 208,  41,  15, 201,   1
PD 1880 data 208,   1, 202, 138, 162,   0, 157,   0
OG 1890 data 216, 157,   0, 217, 232, 208, 247, 169
BB 1900 data 147,  32, 210, 255, 162,  13, 169,  17
OJ 1910 data  32, 210, 255, 202, 208, 250, 169, 160
NI 1920 data 162, 240, 157, 255,   3, 202, 208, 250
JH 1930 data 162,   4, 189, 127, 203, 157,  57,   4
PJ 1940 data 202, 208, 247, 160,   0, 169,   3, 141
BO 1950 data  21, 204, 169,  88, 133, 251, 169,   4
CN 1960 data 133, 252, 189, 132, 203, 240,   6, 145
AL 1970 data 251, 232, 200, 208, 245,  32, 157, 206
AB 1980 data 232, 160,   0, 206,  21, 204, 208, 234
ID 1990 data 160,  29, 185, 204, 203, 153,  29,   5
AF 2000 data 185, 234, 203, 153, 109,   5, 136,  16
OA 2010 data 241, 169, 240, 133, 251, 169,   4, 133
EP 2020 data 252, 162,   4, 160,   0, 169, 101, 145
OG 2030 data 251, 160,  39, 169, 103, 145, 251,  32
EB 2040 data 157, 206, 202, 208, 238, 160,  39, 169
OB 2050 data 122, 145, 251, 169, 111, 136, 208, 249
BN 2060 data 169,  76, 145, 251,  96,  24, 165, 251
MI 2070 data 105,  40, 133, 251, 144,   2, 230, 252
JK 2080 data  96
```

Mini–Tracer: PAL Source Code

```
BA 1000 rem save "0:trace44.pal" ,8
EK 1010 rem ** minitracer – trace/single step routine for
NP 1020 rem ** basic programs and bit map – c64
ME 1030 rem ** written by: jim frost – rev. 12/12/85
EI 1040 :
CF 1050 open 8,8,1, "0:trace44.obj"
FP 1060 sys700
BF 1070 .opt o8
FG 1080 *        =    $cb80
IL 1090 ;
CE 1100 curlin   =    $39        ; current line #
GG 1110 keyflg   =    $c5        ; which key pressed
LM 1120 shflag   =    $028d
JL 1130 hibase   =    $0288
DH 1140 raster   =    $d012      ; raster position
OG 1150 bgcol    =    $d021
OJ 1160 chrout   =    $ffd2      ; output a char
IA 1170 ;
FO 1180 ;** screen data **
EA 1190 mini     =    *
HE 1200 .byte $8d, $89, $8e, $89
AD 1210 ;
NJ 1220 tracer   =    *
GG 1230 .byte $20, $20, $20, $a0, $20, $62, $5f, $a0
GN 1240 .byte $69, $62, $5f, $a0, $69, $62, $5f, $a0
NK 1250 .byte $20, $62, $fe, $a0, $20, $62, $5f, $00
GB 1260 .byte $a0, $20, $a0, $a0, $20, $e2, $20, $a0
AC 1270 .byte $20, $e2, $20, $a0, $20, $a0, $a0, $a0
IP 1280 .byte $20, $e2, $a0, $a0, $20, $e2, $20, $00
IE 1290 .byte $a0, $20, $a0, $a0, $20, $a0, $7b, $a0
FL 1300 .byte $20, $a0, $20, $a0, $df, $e2, $e9, $a0
AF 1310 .byte $20, $e2, $fb, $a0, $20, $a0, $7b, $00
OJ 1320 ;
KH 1330 msg1     =    *
AD 1340 .byte $10, $12, $05, $13, $13, $20, $03, $3d
ME 1350 .byte $20, $14, $0f, $20, $14, $0f, $07, $07
LF 1360 .byte $0c, $05, $20, $13, $09, $0e, $07, $0c
HN 1370 .byte $05, $2d, $13, $14, $05, $10
KN 1380 ;
HL 1390 msg2     =    *
IF 1400 .byte $10, $12, $05, $13, $13, $20, $13, $08
CH 1410 .byte $09, $06, $14, $20, $0f, $12, $20, $01
GJ 1420 .byte $0e, $19, $20, $0b, $05, $19, $20, $14
AB 1430 .byte $0f, $20, $13, $14, $05, $10
GB 1440 ;
CD 1450 ;** variables **
HG 1460 ssflg   .byte 0
FH 1470 linlo   .byte 0
PG 1480 linhi   .byte 0
IE 1490 ;
JN 1500 bcdhi   =    *
CI 1510 .byte 0,  0,  0,  0
GG 1520 ;
KM 1530 tlnlo   .byte 0
EM 1540 tlnhi   .byte 0
EI 1550 ;
PI 1560 ztemp   =    *
OL 1570 .byte 0,  0,  0,  0
CK 1580 ;
FB 1590 count   .byte 0
GL 1600 ;
GC 1610 ;** colors compatable with background **
HA 1620 coltab  =    *
PA 1630 .byte $01, $00, $01, $00, $01, $01, $01, $00
LB 1640 .byte $01, $01, $02, $01, $01, $00, $01, $00
IO 1650 ;
NL 1660 ;** character set for bit map **
MD 1670 chrtab  =    *
FH 1680 .byte $3f, $33, $33, $33, $33, $33, $3f, $3f  ;zero
BF 1690 .byte $3c, $3c, $0c, $0c, $0c, $0c, $3f, $3f  ;one
CC 1700 .byte $3f, $33, $03, $3f, $30, $33, $3f, $3f  ;two
HD 1710 .byte $3f, $33, $03, $0f, $03, $33, $3f, $3f  ;three
MD 1720 .byte $33, $33, $33, $3f, $03, $03, $03, $03  ;four
MG 1730 .byte $3f, $30, $30, $3f, $03, $33, $3f, $3f  ;five
EC 1740 .byte $3f, $33, $30, $3f, $33, $33, $3f, $3f  ;six
CB 1750 .byte $3f, $33, $03, $03, $03, $03, $03, $03  ;seven
MI 1760 .byte $3f, $33, $33, $3f, $33, $33, $3f, $3f  ;eight
KK 1770 .byte $3f, $33, $33, $3f, $03, $33, $3f, $3f  ;nine
```

www.Commodore.ca
May Not Reprint Without Permission

```
KG 1780 ;
BG 1790 ;** table of bank addresses **
PE 1800 banktab=  *
CP 1810 .byte $c0, $80, $40, $00, $e0, $a0, $60, $20
CJ 1820 ;
AD 1830 ;** start of wedge **
GJ 1840 start     =     *
DA 1850           pha           ;save a and x on stack
IP 1860           txa
KM 1870           pha
MO 1880           ldx  #0       ;clear temp flag in x
FC 1890           lda  curlin   ;low byte of current line #
AG 1900           cmp linlo
HM 1910           beq samelo
GP 1920 ;
IC 1930           inx           ;set temp flag
BI 1940           sta  linlo
EB 1950 ;
DI 1960 samelo =  *
EO 1970           lda  curlin+1 ;high byte current line #
KJ 1980           cmp linhi
HA 1990           beq samehi
GE 2000 ;
GL 2010           inx
LL 2020           sta  linhi
EG 2030 ;
JL 2040 samehi =  *
GG 2050           cpx  #0       ;if x still 0 then
GD 2060           bne  trace    ;then we are on same line
MI 2070 ;
FP 2080           jmp  quickout
AK 2090 ;
JG 2100 trace     =     *
LI 2110           ldx  #3
OL 2120 ;
DN 2130 savzp     =     *
HL 2140           lda  $fb,x    ;save user zero page
OC 2150           sta  ztemp,x  ;so trace can share
HC 2160           dex
NK 2170           bpl  savzp
KP 2180 ;
HK 2190 nokeys  =   *
FK 2200           lda  shflag   ;get special keypress
OM 2210           cmp #2        ;c="?"
HK 2220           bne  tstflg   ;no. jump to flag test
MC 2230 ;
PC 2240           lda  ssflg    ;else toggle the flag
OA 2250           eor  #1
AB 2260           sta  ssflg    ;and store the new flag
EF 2270 ;
CM 2280 finger    =     *
OA 2290           lda  shflag
LO 2300           bne  finger   ;wait until fingers are
                                 lifted
MH 2310 ;
IC 2320 tstflg    =     *
EM 2330           lda  ssflg
OL 2340           beq  nopause  ;if ssflg=0 then skip ss
EK 2350 ;
GN 2360           lda  keyflg   ;check key
JI 2370           cmp #64       ;if 64 then no keys
                                 pressed
KJ 2380           bne  keyprs   ;else keys pressed so
                                 continue
MM 2390 ;
MH 2400           lda  shflag
DB 2410           beq  nokeys   ;repeat until keys
                                 pressed
KO 2420 ;
IK 2430 keyprs    =     *
AB 2440           ldx  #$30
IA 2450 ;
NB 2460 delay1    =     *
OF 2470           lda  raster   ;raster position
GB 2480           bne  delay1   ;repeat until raster=0
AD 2490 ;
IE 2500 delay2    =     *
MD 2510           lda  raster
ON 2520           beq  delay2   ;repeat until raster not
                                 zero
IF 2530 ;
LN 2540           lda  shflag   ;check for request to
GF 2550           cmp #2        ;exit single step
FG 2560           bne  delay3   ;if no request,continue
                                 wait
AI 2570 ;
PC 2580           lda  #0       ;else clear flag
GA 2590           sta  ssflg
OJ 2600 ;
NJ 2610 thumb     =     *
JH 2620           lda  shflag   ;wait for fingers up
BC 2630           bne  thumb
GM 2640 ;
KE 2650           beq  nopause  ;and resume trace
KN 2660 ;
FP 2670 delay3    =     *
PC 2680           dex
LG 2690           bne  delay1   ;repeat until x=0
CA 2700 ;
BG 2710 nopause=  *
LO 2720           ldx  #2
AC 2730 ;
MJ 2740 clrmem =  *
KB 2750           lda  curlin-1,x ;save basic line #
LK 2760           sta  tlnlo-1,x ;clear mem for new
                                 bcd numbers
NL 2770           lda  #0
PN 2780           sta  bcdhi,x
NJ 2790           dex
GF 2800           bne  clrmem
AH 2810 ;
HK 2820           sta  bcdhi
FO 2830           ldx  #$0f
OI 2840 ;
IL 2850 htod      =     *
LG 2860           asl  tlnlo    ;get one bit at a time
CH 2870           rol  tlnhi    ;from the basic
IN 2880           sei           ;line # and add it
LP 2890           sed           ;to the bcd # being
                                 formed
DK 2900           ldy  #2
EN 2910 ;
EN 2920 decadd =  *
KD 2930           lda  bcdhi,y
GD 2940           adc  bcdhi,y
MI 2950           sta  bcdhi,y
LE 2960           dey
FK 2970           bpl  decadd
KB 2980 ;
IC 2990           cld
GE 3000           cli
JH 3010           dex
NF 3020           bpl  htod
ME 3030 ;
GO 3040           lda  hibase   ;high byte of screen
                                 address
JA 3050           sta  $fc
FB 3060           lda  $dd00    ;video bank in low two
                                 bits
DA 3070           and  #3
DN 3080           tax
IP 3090           lda  banktab,x
CJ 3100           clc
JP 3110           adc  $fc
DB 3120           adc  #3
JF 3130           sta  $fc
FN 3140           lda  #$e0     ;offset to screen
                                 bottom
KG 3150           sta  $fb
BP 3160           lda  $d011
PJ 3170           and  #$20
NC 3180           beq  nobit
MO 3190 ;
EE 3200           jmp  bitout
AA 3210 ;
GC 3220 ;** lo-res line number display **
GO 3230 nobit     =     *
PO 3240           ldx  #0
NP 3250           ldy  #0
CD 3260 ;
OA 3270 gethi     =     *
PE 3280           lda  bcdhi,x  ;get bcd number
OC 3290           pha           ;save it on stack
AJ 3300           lsr           ;shift high nibble to low
HM 3310           lsr
BN 3320           lsr
LN 3330           lsr
OE 3340           ora  #$30     ;convert to screen
                                 code
CH 3350           sta  ($fb),y  ;and poke on screen
AA 3360           iny
MG 3370           pla           ;get save bcd number
EH 3380           and  #$0f     ;throw away high
                                 nibble
BE 3390           ora  #$30     ;convert to screen
                                 code
NG 3400           sta  ($fb),y  ;poke it on screen
OC 3410           inx
MD 3420           iny
NG 3430           cpx  #3       ;repeat until six digits
IC 3440           bne  gethi
AP 3450 ;
BM 3460 col2      =     *
BO 3470           ldx  #6
JP 3480           lda  bgcol    ;check background
                                 color
BD 3490           and  #$0f
LH 3500           tay
LD 3510           lda  coltab,y ;get compatable color
                                 from table
GD 3520 ;
AM 3530 cmem1 =   *
PM 3540           sta  $dbdf,x  ;and poke color
                                 memory
FJ 3550           dex
LH 3560           bne  cmem1
IG 3570 ;
JE 3580           ldx  #3
MH 3590 ;
PI 3600 zrest     =     *
HD 3610           lda  ztemp,x
IE 3620           sta  $fb,x
FO 3630           dex
JE 3640           bpl  zrest
IL 3650 ;
HO 3660 quickout= *
KF 3670           pla           ;finish chrget
LC 3680           tax
CP 3690           pla
JC 3700           cmp #$3a
LH 3710           bcs  cg1
OP 3720 ;
BH 3730           jmp  $80
CB 3740 ;
PM 3750 cg1       =     *
CM 3760           jmp  $8a
AD 3770 ;
EA 3780 ;* hires line number display *
AI 3790 bitout    =     *
PD 3800           lda  $d018    ;bit 8 set puts
DH 3810           and  #8       ;bit map in upper half
HO 3820           beq lowbank   ;mask unwanted
MG 3830 ;
MN 3840           inx
GO 3850           inx
AP 3860           inx
KP 3870           inx
OJ 3880 ;
HP 3890 lowbank=  *
CC 3900           lda  banktab,x
ML 3910           clc
FI 3920           adc  #$1f     ;offset to bottom of
                                 bitmap
PH 3930           sta  $fe
```

www.Commodore.ca
May Not Reprint Without Permission

```
PE 3940          lda  #0
AJ 3950          sta  $fd
DG 3960          lda  #0
EJ 3970          sta  count
HN 3980          ldy  #0
MA 3990 ;
AG 4000 getbcd   =    *
KN 4010          ldx  count
LC 4020          lda  bcdhi,x   ;get bcd number
NO 4030          pha            ;save on stack for low
                                 nibble
FN 4040          and  #$f0      ;mask low nibble
FF 4050          lsr            ;high nibble is
                                 16*value
EJ 4060          tax            ;divide by 2 for 8*value
MF 4070 ;
FO 4080 nextrow =    *
HA 4090          lda  chrtab,x  ;and get indexed
                                 character
NB 4100          sta  ($fd),y   ;poke on bitmap
KO 4110          inx
IP 4120          iny
NK 4130          cpy  #8        ;done with character 1
JH 4140          beq  low       ;print box right and left
MK 4150 ;
EP 4160          cpy  #$18      ;done with character 3
AN 4170          beq  low
KM 4180 ;
JB 4190          cpy  #$28      ;done with character 5
OO 4200          beq  low
IO 4210 ;
DC 4220          bne  nextrow
MP 4230 ;
LH 4240 low      =    *
CJ 4250          pla            ;fetch bcd for low
                                 nibble
DH 4260          and  #$0f      ;mask high nibble
LC 4270          asl            ;multiply by 8
DG 4280          asl
NG 4290          asl
HJ 4300          tax
ME 4310 ;
DM 4320 nextlow =    *
KA 4330          lda  chrtab,x  ;get indexed character
NA 4340          sta  ($fd),y   ;poke on bitmap
KN 4350          inx
IO 4360          iny
EK 4370          cpy  #$10      ;done with character 2
JL 4380          beq  countup
MJ 4390 ;
JM 4400          cpy  #$20      ;done with character 4
HN 4410          beq  countup
KL 4420 ;
OO 4430          cpy  #$30      ;done with character 6
IN 4440          beq  scolor
IN 4450 ;
NA 4460          bne  nextlow
MO 4470 ;
GI 4480 countup=    *
OH 4490          inc  count
GL 4500          bne  getbcd
EB 4510 ;
FM 4520 scolor   =    *
HA 4530          ldy  #5
CO 4540          lda  #$10
MD 4550 ;
LA 4560 cm1      =    *
GF 4570          sta  ($fb),y
PJ 4580          dey
CA 4590          bpl  cm1
OG 4600 ;
KE 4610          jmp  col2
CI 4620 ;
ID 4630 ;** initialize chrget **
PF 4640          lda  #$4c      ;insert the wedge
AC 4650          sta  $7c       ;by poking chrget with
                                 jmp $cd78
ND 4660          lda  #<start   ;low byte start address
CE 4670          sta  $7d
ND 4680          lda  #>start   ;high byte of start
JF 4690          sta  $7e
MJ 4700          ldx  #1        ;white
PO 4710          lda  bgcol
FC 4720          and  #$0f      ;mask high nibble
FD 4730          cmp  #1        ;is background white
FG 4740          bne  white     ;no, leave text white
EA 4750 ;
AH 4760          dex            ;else change color to
                                 black (0)
IB 4770 ;
KP 4780 white    =    *
KG 4790          txa
HA 4800          ldx  #0
AE 4810 ;
PI 4820 color    =    *
IC 4830          sta  $d800,x
FD 4840          sta  $d900,x
OM 4850          inx
BN 4860          bne  color
MH 4870 ;
EM 4880 ;** print initial screen **
KO 4890          lda  #$93      ;clear screen
LP 4900          jsr  chrout
NP 4910          ldx  #$0d
CG 4920          lda  #$11
IL 4930 ;
AL 4940 cdwn     =    *
BE 4950          jsr  chrout    ;print 13 cursor downs
HB 4960          dex
EP 4970          bne  cdwn
KO 4980 ;
EN 4990          lda  #$a0
JE 5000          ldx  #$f0
IA 5010 ;
HC 5020 rvs1     =    *
DF 5030          sta  $03ff,x   ;print 6 rows of reverse
                                 spaces
HG 5040          dex
HP 5050          bne  rvs1
KD 5060 ;
NB 5070          ldx  #4
OE 5080 ;
FC 5090 mi1      =    *
KI 5100          lda  mini-1,x  ;print mini
BC 5110          sta  $0439,x
HL 5120          dex
OP 5130          bne  mi1
KI 5140 ;
IC 5150          ldy  #0        ;print tracer
JB 5160          lda  #3
EE 5170          sta  count
OI 5180          lda  #$58
CG 5190          sta  $fb
DE 5200          lda  #4
JH 5210          sta  $fc
KN 5220 ;
KN 5230 tr1      =    *
EH 5240          lda  tracer,x
IF 5250          beq  nxtrow
CA 5260 ;
CB 5270          sta  ($fb),y
MH 5280          inx
KI 5290          iny
BM 5300          bne  tr1
ED 5310 ;
FC 5320 nxtrow   =    *
LO 5330          jsr  pl40
IL 5340          inx
BD 5350          ldy  #0
PL 5360          dec  count
HA 5370          bne  tr1
KH 5380 ;
EO 5390          ldy  #$1d
OI 5400 ;
DI 5410 ms1      =    *
PG 5420          lda  msg1,y    ;print top message
KG 5430          sta  $051d,y
KH 5440          lda  msg2,y    ;and bottom message
CJ 5450          sta  $056d,y
PA 5460          dey
II 5470          bpl  ms1
ON 5480 ;
LM 5490          lda  #$f0      ;set $fb for printing box
IJ 5500          sta  $fb
JH 5510          lda  #4
PK 5520          sta  $fc
JO 5530          ldx  #4
KB 5540 ;
MO 5550 side     =    *
DA 5560          ldy  #0
PO 5570          lda  #$65      ;left side
IE 5580          sta  ($fb),y
LH 5590          ldy  #$27
CO 5600          lda  #$67      ;right side
GG 5610          sta  ($fb),y
EE 5620          jsr  pl40      ;add to $fb for next row
NL 5630          dex            ;finished when x=0
PE 5640          bne  side
II 5650 ;
AK 5660          ldy  #$27      ;print box bottom
DF 5670          lda  #$7a      ;right side
GK 5680 ;
HL 5690 bott     =    *
AM 5700          sta  ($fb),y
AL 5710          lda  #$6f      ;bottom
DB 5720          dey
AB 5730          bne  bott
CO 5740 ;
FN 5750          lda  #$4c      ;left side
MP 5760          sta  ($fb),y
KL 5770          rts            ;back to basic
KA 5780 ;
IE 5790 ;** add 40 to $fb for next screen row **
EJ 5800 pl40     =    *
IC 5810          clc
KJ 5820          lda  $fb
BA 5830          adc  #$28
MO 5840          sta  $fb
OK 5850          bcc  pl1
KF 5860 ;
PO 5870          inc  $fc
OG 5880 ;
EF 5890 pl1      =    *
IP 5900          rts
MI 5910 ;
MP 5920 .end
```

www.Commodore.ca
May Not Reprint Without Permission

Shiloh's Raid: 1541 Relative File Bug Spray

David Shiloh
Eugene, Oregon

© 1986 by David Shiloh

First we squashed the SAVE@ bug with Phillip Slaymaker's article. . . now David Shiloh kills the dastardly relative file bug -- right at its roots!

It appears that there has not previously appeared in print a dissection of the huge relative file bug in the DOS, although the save "@0:bug" was a major controversy for years: the reason of this escapes me somehow, since relative files seem more major in relation to practical uses of the 1541. . .how have the gurus been distracted from such a serious problem with the DOS?

Dr. Gerald Neufeld, whose Inside Commodore DOS has proved to be indispensable, mentions the bug in his 1541 User's Guide, correctly locating it in the "position" command and offering an effective fix that exacts a 30%–40% access–time penalty. While his fix reaches two of the specific DOS failures that are involved, his discussion does not define the conditions under which problems occur, and his test program yields results that establish the existence of the bug but are otherwise almost completely misleading. Until now, this has been the most comprehensive mention of this bug.

The Position Command

The actual write to a relative file uses the same PRINT# command as any other write operation. With relative files, however, the write goes to a specific record within the file: DOS has to be positioned to the record you want to write to, and to the spot within that record where you want to begin writing. This is done with the "position" command, sent on the DOS command channel; the actual information to be written to that record is sent to the relative file following the position command. The position command is sent with the syntax:

```
print#FN, " p " chr$(96 + SA)chr$(lo)chr$(hi)chr$(po);
```

where "p" is the actual "position" instruction, followed by three parameters and a final semicolon (";") to suppress the sending of a carriage return after the command string.

The chr$(96 + SA) sends DOS the secondary address (SA) of the relative file OPEN command, which is used by DOS to assign internal channels and buffers for the relative file operations: this value is OR'd with 96 ($60) to form the byte sent to the DOS.

The chr$(lo) and chr$(hi) are one parameter, the record number (nu): lo is the low byte of the record number in low–byte/ high–byte format, "hi" is the high byte, taken by

```
hi = int(nu/256):lo = nu–hi*256
```

The chr$(po) is the exact position within the relative record where the write is to begin, and is an optional parameter. However, unless you suppress the carriage return that follows the command string, this parameter chr$(po) must be included: otherwise, DOS will read the chr$(13) carriage return as the parameter and point there.

When the position command is sent, DOS retrieves the record sector you have addressed into its RAM buffers and sets the relative file channel to the selected position in the record. The same "position" command is used to position the relative file channel for reading from the file.

The Bug

Theoretically, the "position" command will allow you to position to any character in any record. In fact, this is true only for reading the file: for writing, it is 100% reliable except under certain conditions in which it is 100% unreliable.

When DOS receives a position command, it checks to see whether the desired bytes are already in one of the two buffers allocated for records. If the necessary sector is not in the "active" buffer, but the immediately preceding file sector is, then DOS simply "toggles" the buffers and makes the one containing the necessary sector active: unless it just toggled during the last access for that reason. This convenience also sets up the bug: the fatal sequence is as follows:

1. A write is performed that runs from one sector (A, in buffer a) to the next (B, in buffer b). During the write, DOS toggles from buffer a to buffer b and makes a note of the toggle.
2. A second write is performed to a record that is entirely contained on sector B in the now–active buffer b. This write does NOT toggle, and DOS makes a note of the no–toggle. Now the bug is waiting.

www.Commodore.ca
May Not Reprint Without Permission

3. A third write is directed to the sector following B; and instead of fetching sector C, DOS toggles from buffer b to buffer a since no toggle was performed during the last access.

Unfortunately, sector A is still in buffer a and this third write goes to exactly the same place on sector A that it should have gone to on sector C -- and often overwrites two records, the last characters of one and the first characters of the next. Thus three records are in jeopardy: these two and the one that did not get written to sector C.

The program listing below demonstrates the bug, then sprays it with Shiloh's Raid.

The program creates a relative file of 100 records for each record size from 42 through 88, spending about 10 minutes with each (6 minutes compiled). Since the entire program runs over 8 hours, I set it up to rotate among my three 1541 drives, which are hardware set to device numbers 8, 9 and 10. The program will rotate among any number of drives by changing the 'nd = 1' in line 1140; the lowest drive number used can be changed from 8 by modifying the 'sd = 8' in line 1130. If you are using just one drive, you may want to use a cooling fan, or run the test for fewer trials (reduce the value of 'el' in line 1120). If you are using the program with a non-Commodore printer, check the control codes in lines 1660, 1830 and 2010 (control-j, chr$(10) for a line-feed) for compatibility with your interface.

Also, in line 1090-1120, "nr=100" determines the size of the relative file (number of records); "nt=15" is the number of test strings written to the file (it must be a multiple of 15); sl=41 is the record length of the first test file; and el=88 is the record length of the last test file (the entire test is performed using files with record lengths from 'sl' to 'el')

Lines 1880-2050 reset the drive, short new the disk, open a relative file, force creation of 'nr' empty records, and then write a unique identifying string to each 8-character field of every record, in the format

```
nnnn/ff*
```

where nnnn is a four-digit record number (with any leading zeroes) and ff is a two-digit field number (with any leading zero). Thus every record looks like this:

```
0123/01*0123/02*0123/03*0123/04*0123/05*012
```

(this is 43-character record #123), with a longer final field if the record length is not a multiple of 8.

Then the fun begins. . .three passes are made through the file.

Pass 1 selects a random field of a random record and tests to insure that the write (which goes to the end of the record) spans two sectors, then constructs a string to overwrite the selected record fields with the identifying string already there. (In literally over a million trials, we found that the initial write to the records always works. If you're skeptical, put a 'GOSUB 1600' in line 2060 to verify the contents of all records.) This pass then calls the position routine at line 1420, and the write is sent to the disk. A second write is sent to the next record, which lies entirely in the sector where the first write ended; and a third to a record lying entirely in the next sector in the file.

Pass 1 will produce an error on every third write, corrupting one or two records and leaving the "updated" record untouched. It may write the same series of three more than once during the pass: a detailed report is sent to the printer for study.

The first (identifying) field of each re-write, the number of the sector (in file sequence) and the initial byte (2–255) of the write, are stored in an array in the order written. On completion of nt/3 sets, the entire file is read by the subroutine at line 1610; and on detection of a variance, this array is sent to the printer from line 1510 followed by a report on the corrupted record (its number within the file, the starting sector and byte) and the actual contents from the disk. Subsequent variances are also printed with their identifying data: this information enables you to see exactly what was overwritten, by which write in which set of three; as well as what might have been restored by a later write and any duplicated sets (duplication confuses the error count). The printer output is formatted to produce a one-page report on each record size (two if needed).

Shiloh's Raid

We have been able to develop a short subroutine to anticipate the bug and apply a fix only when it is needed — less than 1% of the time — and otherwise use the position command as already described, without the 30%-40% time penalty. This subroutine is situated in lines 1380 through 1470 and includes the usual position routine and a variation on Dr. Neufeld's "point twice and wait" fix, which it selectively incorporates.

Line 1380 is the write entry point: if the immediately previous call to the position routine spanned two sectors, then it identifies the second and jeopardized sectors arising from that call and sets a counter to be active during the next two accesses. Line 1390 (the read entry point since reads do not need protection but do need to set a flag) calculates the end position of the current record within the record sector and, if a split record, the start position; and flags a split-write condition when the current access spans two sectors. This is the flag detected during the next position call in Line 1380. Line 1420 (the "index search" entry point, when a single character is to be retrieved for a search comparison, since a single-character retrieval cannot span two sectors) calculates the high and low bytes of the record number; and if a jeopardy flag has been set up by one of the two previous calls to the position routine, checks the sector of the current access against the sectors identified in line 1380; pointing once and setting up the wait

www.Commodore.ca
May Not Reprint Without Permission

flag when an endangered sector is being accessed. Line 1450 sends the position command and, if the wait flag is set, waits 30 jiffies before returning from Line 1470.

Pass 2 performs exactly like Pass 1 except that it calls Shiloh's Raid at line 1380 and produces no errors.

Pass 3 makes 20*nt random selections, not writing a sequence of records unless they occur as a result of the random selection, and counts the number of times (1) that a flagged condition arises and (2) that a full fix is required. Although actual relative file use is not usually as random as this, the 1–2–3 sequence of passes 1 and 2 is just as untypical in the opposite direction. Pass 3 does, however, give some idea of how often Shiloh's Raid calls the delay fix, sending the count to the printer at the end of the pass. Our results depended on the size of the file: fewer waits with larger files, 0.08% in half a million accesses of disk-sized (664-block) files.

The time involved in the flagging algorithm also varied with the size of the file. Calls to Shiloh's Raid cost from 0.039 seconds per call for larger files to 0.048 seconds per call for smaller files: smaller files more often randomly encountered the flag conditions. Enlarge the file and change the subroutine call for Pass 3 in line 2210, and you will get an idea of how often C-64/1541 users encounter this bug: since it bites on 100% of these occasions, the two-jiffy price of reliability is low.

Dr. Neufeld's fix — point a second time and wait half a second — forces DOS to look at the active buffer, where it finds the wrong sector, writes that (previously changed) sector back to the disk, and then fetches the correct sector. The wait is necessary because without it, an immediately following PRINT# command causes an ATN interrupt that is waiting (with a higher IRQ priority than the fetch job) to take over when the DOS comes back from writing the old sector, before the fetch job is put in either the job queue or the buffer's track and sector pointers. The write is performed to the buffer, the buffer dirty flag is set, the poisoned sector is written over the last write-to-disk with the mis-directed information, and then the correct sector is fetched from the disk into the buffer. . . but too late.

Although the position command is entirely reliable for reading from the file, the bug may bite on a write that follows a read access, making the detection algorithm necessary on read accesses since it flags a condition about to arise. Shiloh's Raid still allows retrieval to the screen of an 85-character record in an average 1.17 seconds from a disk-sized file.

With Shiloh's Raid in place, the position command is 100% reliable. Now, perhaps CBM will consider an upgrade chip, since the 1541 outsold their wildest expectations and is still selling: I'd prefer that to a shiny new plastic face. I need three. . . just send them to me at PO Box 10976, Eugene OR 97440, and I'll express my complete surprise and profound astonishment in an appropriate fashion. . .

Shiloh's Raid: The Program

```
CN  1000 rem*********************************
JN  1010 rem*        " Shiloh's Raid "         *
DH  1020 rem*    this program demontrates      *
MJ  1030 rem*    the 1541 relative file bug,   *
II  1040 rem*    and gives an efficient way    *
GE  1050 rem*       to work around it.         *
GH  1060 rem*      (c) 1986 david shiloh       *
IB  1070 rem*********************************
MK  1080 :
NM  1090 nr  = 100:rem* number of records
NA  1100 nt  = 15  :rem* number of writes
DK  1110 sl  = 41  :rem* start record length
BI  1120 el  = 88  :rem* end record length
LL  1130 sd  = 8   :rem* first drive number
JH  1140 nd  = 1   :rem* number of drives
KN  1150 ed=sd+nd-1
MP  1160 :
PD  1170 gosub 1710: rem* initial prompts
CH  1180 goto 1810: rem* continue main routine
PM  1190 rem* subroutines follow
EC  1200 :
AP  1210 rem** create formatted output **
LP  1220 r$(ct)=left$(r$,7)+";"
IH  1230 r$(ct)=r$(ct)+right$("  "
          +str$(q%+1+(l>q)),3)+":"
EL  1240 r$(ct)=r$(ct)+left$(mid$(str$(q-l+p
          +1-(q-l+p<1)*254),2)+"[3 spcs]",4)
OP  1250 return
AG  1260 :
FI  1270 rem** create record contents **
JJ  1280 r$="": n$=right$(z$+mid$(str$(n),2),4)
JL  1290 for fs=f to nf+1
EE  1300 fs$=z$+mid$(str$(fs),2)
BF  1310 r$=r$+n$+"/"+right$(fs$,2)+"*"
MC  1320 next
FM  1330 r$=left$(r$,l-8*(f-1))
IF  1340 return
KL  1350 :
DA  1360 rem** shiloh's raid subroutine **
GD  1370 rem (write relative record)
PM  1380 if sr then r1=sr+1: r2=sr+2: r=2
LO  1390 q=n*l: q%=q/254: q=q-q%*254
          : sr=q%*-(l>q)
AH  1400 if sr then sr=q%*-(q-l+p<1)
JC  1410 rem* entry point for no-fix write
PL  1420 h%=n/pg: lo=n-h%*pg
FH  1430 rem point twice & wait if needed
IC  1440 if r then r=r-1: rs=rs+r: if q%=r1 or q%=r2
          then gosub 1450: w=162
CP  1450 print#1,"pB"chr$(lo)chr$(h%)chr$(p);
GH  1460 if w then poke w,2: wait w,32: w=0: c=c+1
KN  1470 return
MD  1480 :
```

www.Commodore.ca
May Not Reprint Without Permission

```
NG  1490 rem** print bad record message **
MN  1500 if e goto 1540
GG  1510 print#7,r$(0)
NA  1520 for t=1 to nt+1: print#7,r$(t);: next
EA  1530 print#7: x=x+nt/5+3
LK  1540 e=e+1: q=(n-1)*l+1: q%=q/254
         : q=q-q%*254
BP  1550 if n<>sn then print#7, "record"n"sector"
         q%+1"byte"q+1: te=te+1: x=x+1
OJ  1560 sn=n+1: print#7,ck$: x=x+1-(l>80)
BA  1570 if ps<3 then gosub 1420: print#2,r$;: n=n-1
IE  1580 return
KK  1590 :
OJ  1600 rem** read and check all records
DG  1610 print: p=1: f=1: e=0: te=0
AA  1620 for n=1 to nr: print"    reading";n
MD  1630 gosub 1280: gosub 1420
IL  1640 input#2,ck$: if ck$<>r$ then gosub 1500
GH  1650 next
MK  1660 print#7,"j"r$(0)te"errors in"e"records,"
         rs"calls,"c"to wait routine"
AG  1670 print"qpass";ps;":";te;"bad to";e;
         "records";rs;"calls";c
MK  1680 return
OA  1690 :
EJ  1700 rem** print initial prompts   **
HO  1710 print"qnhOutput to (S)creen or (P)rinter ?"
KH  1720 get a$: if a$<>"p" and a$<>"s" goto 1720
NF  1730 sp=3: if a$="p" then sp=4
KG  1740 print"Insert a scratch disk and
         press RETURN."
ME  1750 get a$: if a$<>chr$(13) goto 1750
MP  1760 return
OF  1770 :
IG  1780 rem*************************
CN  1790 rem**  mainline follows: ***
MH  1800 rem*************************
HH  1810 pg=256: l$=chr$(157): s=rnd(-ti): d=sd
IH  1820 open 7,sp,7: rem printout file
GB  1830 z$="000": dim r$(nt+1)
         : r$(nt+1)="jerrors:"
EK  1840 :
HC  1850 rem- do for all record lengths -
JC  1860 for l=sl to el
OM  1870 kn=254/l
JA  1880 rem- reset drive -
AB  1890 close1: open1,d,15,"ui": for t=1 to 500: next t
ID  1900 b=int(nr*l/254)+1: n=nr: nf=int(l/8): f=1: p=1
KO  1910 :
DB  1920 rem- new disk & open rel file -
CO  1930 x$="0:test"+str$(l): print#1,"n"x$
LH  1940 close2: open 2,d,2,x$+",l,"+chr$(l): ps=0
         : x=0
GO  1950 print "SqqqShiloh's Raid: Relative File
         Bug Spray"
GF  1960 print "         (c) 1986 by David Shiloh"
DJ  1970 print "qqtest"l;l$"x"mid$(str$(nf),2)nr;b
         "sectors"nt"testsq"
AD  1980 :
OO  1990 rem- initialize all records -
CJ  2000 for t=0 to nt: r$(t)="": next
PB  2010 print#7,"jtest"l;nr"records"nf"fields"b
         "sectors"nt"re-writesj"
CI  2020 print"setting up the file...": gosub 1420
         : print#2
EJ  2030 for n=1 to nr: gosub 1280
AF  2040 print"    writing "left$(r$,20)"...Q"
         : gosub 1420
NC  2050 print#2,r$;: next
OD  2060 print
AG  2070 rem- write random records -
AN  2080 for ps=1 to 3: rem  three passes
CJ  2090 r$(0) = "qpass" + str$(ps) + " re-writes:"
HI  2100 ne=0: c=0: rs=0: sr=0: print r$(0)
HM  2110 rem- write nt records -
AF  2120 for ct=1 to nt-(ps=3)*19*nt
EH  2130 if ne then n=n+1-(ne=2)*int(kn): goto 2180
LC  2140 n=int(rnd(1)*(nr-kn)+1): f=int(rnd(1)*nf+1)
         : p=8*f-7
GG  2150 if ps=3 goto 2190
JK  2160 gosub 1390: if sr=0 goto 2140
OA  2170 sr=0
KB  2180 ne=ne+1: if ne>2 then ne=0
HH  2190 gosub 1280: print"    writing "left$(r$,7);ct
II  2200 rem* write rec with or w/o "raid"
HJ  2210 on ps gosub 1420, 1380, 1380: print#2,r$;
LC  2220 if ps<3 then gosub 1220
DN  2230 next ct
IN  2240 gosub 1610:rem verify written records
KA  2250 next ps
IE  2260 :
CP  2270 r$="full wait in" + str$(int(50*c/nt)/10)
ID  2280 r$=r$+"%"+str$(nt*20)+" pass 3
         accesses"
DA  2290 print r$: print#7,r$
MO  2300 rem -page printer & do next file-
MB  2310 for t=x to 55-66*(x>54): print#7: next t
OG  2320 d=d+1: if d>ed then d=sd: rem for
         multiple drives
CA  2330 next l
NO  2340 close 1: close 7
OC  2350 end
```

www.Commodore.ca
May Not Reprint Without Permission

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 350 so far. If you would like one, they're free for the asking. Call or write the office in Milton, Ontario.

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 at least 20 times now, and it still shows virtually no sign of wear due to "washing machine punishment".

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.

Oh No!

Some Transactor readers have noticed a problem with the last issue, i.e. duplicate pages. The real problem, however, is that the duplicates caused other pages to go missing. The following is an excerpt from a letter received from our printer, Maclean-Hunter.

> *We have investigated the problem and found that a press problem resulted in the printing of one 16 page signature as two 8 page signatures for part of the run. A duplicate signature must have been placed in the wrong pocket on the binder. Each pocket holds 200 to 300 sheets, and we hope that would limit the extent of the problem. This is backed up by the fact that we did not run short of any pages at the end of the pressing.*

Since then we have received several calls and letters concerning this unfortunate mishap, and new copies have been sent out. It's still possible that more exist and we will replace them. Simply return the bad copy, and another will be sent to you at no charge.

Transactor Mail Order News

Our mail-order department is expanding, 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.

■ Volksmodem 12, w/cable, and CIN Intro-Pack, $299.00 Cdn., $169 U.S.
The Volksmodem 12 is now available from Transactor Publishing, and check out the price! This is an introductory offer ONLY. The price goes up to at or near suggested retail by next issue! Not only do you get the Volksmodem 12 at this incredible price, but you get the cable at no extra charge (the C64 cable goes directly onto the User Port, and the RS232 cable is for any standard RS232 DB-25 female connector) Plus you'll receive a free CompuServe Intro-Pak which contains a User ID, a Password, and $15.00 of connect time! The Volksmodem 12 will work at 300 or 1200 baud, and is "Hayes compatible" so it will work with virtually any terminal software because the commands are controlled by you from the keyboard – just type "AT" (for ATtention) and follow with any of several easy-to-remember commands – no special POKing or elaborate dialing routines necessary! (I've been using a Hayes for almost 3 years, and my Volks for over a year - I love them both! - KJH) It comes with (get this) a 5 year manufacturer's warranty on parts and labour! The modem is shipped insured via UPS at no extra charge! But it won't last long so order soon.

■ Intelligent I/O Interface Cards
■ BH100 I/O Interface Card w/documentation $129 U.S., $199 Cdn
■ BH100-AD8 8-Channel A to D Conversion Module $45 U.S., $69 Cdn
■ BH100 Beginners Course $159 U.S., $239 Cdn
■ BH100-S Security System $25 U.S., $39 Cdn
These products from Intelligent I/O will make great Christmas gifts! And if you've been wondering what to do with that VIC 20 that doesn't get much attention anymore, they're perfect! If you've ever wanted to start doing some real world interfacing, real easy, and inexpensively, then these items are ideal. The boards they sent us for evaluation are currently watching for floods in my basement (see editorial). Too bad I didn't think of it before the flood – it only took about an hour using spare parts I had lying around – no resistors, no capacitors, just two strips of metal, a piece of styrofoam, a brick, and about 20 feet of wire that was also collecting dust. Once I get time, I intend to make it do some more surveillance since only one channel is currently in use. And the program to do it? A quick and messy 5 lines! Since the boards are memory mapped through the cartridge port, a PEEK is all you need! The 22 page manual is clear and concise. All products come with a 90 day manufacturer's warranty. Shipped insured via UPS at no extra charge. See the News BRK item for more information.

■ 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 usally buy. The design is screened using a "super-opaquing" process so they wear much longer than your ordinary screens and iron-ons.

www.Commodore.ca
May Not Reprint Without Permission

■ 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. Occassionally 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, $59.95
You can burn your own using the ROM dump file on Transactor Disk #13, or you can get a set from us. 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'a 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 600 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, Renumber, 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

www.Commodore.ca
May Not Reprint Without Permission

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. 4, Issue 04 – MF only (■ Disk 1)
- ■ Vol. 4, Issue 05 – MF only (■ Disk 1)
- ■ Vol. 4, Issue 06 – MF only (■ 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. 7, Issue 04 – Gizmos and Gadgets (■ Disk 15)

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 still NOT ready. Our new CompuServe duties have forced some projects to the back burner. However, our experience with CompuServe will no doubt help us make this item even better when it's done. We intend to make this "the complete telecomputing package", but please stand by. . . when it's ready, you'll hear about it.

Industry News

MARCA 1986

The first New England "All-Commodore" Computer Fair will be held Saturday, November 15, 1986 at the Best Western Hotel in Marlboro, Massachusetts (just outside Boston at the intersection of Interstate 495 and Route 20), from 10 a.m. to 8 p.m.

The event is being sponsored by the New England member groups of MARCA (Mid-Atlantic Region Commodore Association). MARCA is the largest association of Commodore User Groups in the country.

The Fair will feature vendor exhibits, seminars for beginners through advanced users, and information resource tables. A large collection of public domain software will be available for purchase. Special emphasis will be places on telecommunications, computer graphics, music, and home utility uses for the Commodore machines. Instructional seminars will be scheduled throughout the day. One of the highlights will be a concert of computer-assisted music by Al Hospers.

This show will be of interest to all C-64, C-128 and Amiga users. For additional information, contact:

Frank Ordway, President of MARCA
6 Flagg Road
Marlboro, Massachusetts 01752
(617) 485-4677

Interfacing via the Cartridge Port

Intelligent I/O, Inc. has recently announced the release of its new version of the BH100 General Purpose Input/Output Interface Card for the Commodore 64 and VIC 20 (also the Apple II+ and Apple IIe). This card provides a total of 32 digital input lines, and 32 digital and buffered output lines. Since the ports are memory-mapped, data is sent and retrieved by single POKE and PEEK commands (or their ML equivalents). The BH100 User Manual includes complete instructions, sample programs (including simple BASIC subroutines for all I/O) and diagrams of typical hookups. Knowledge of advanced programming techniques is not needed.

A Complete Beginner's I/O Interface Course is designed for beginners, and includes the BH100 I/O Interface, a Beginner's Module, and an easy-to-read, illustrated Course Manual. The Beginner's Module is a circuit board that "piggy-backs" onto the top of the BH100 I/O Interface and has 8 LEDs, 8 switches and a relay for general switching applications. The Course Manual and Beginner's Module are also available separately.

For those who want to use their computer for a practical application, Intelligent I/O offers the BH100-S Security System Module, which plugs into the BH100 I/O Interface Card and includes everything needed for an eight "zone" advanced security system, including a 120 dB siren. Complete instructions, switches and software round out the package. Any normally closed sensor can optionally be used as a switch (for fire, motion, heat sensors, etc.).

Also available are two models of an Analog-to-Digital Conversion Module (1 channel and 8 channel). These 8-bit A/D converters plug into one of the input ports on the BH100 I/O Interface and automatically digitize an analog input signal (0-5VDC) and read it into memory.

Possible BH100 applications include controlling lights, appliances, relays, motors, heating/cooling systems and other electrical devices; laboratory data acquisition, automated testing/experimentation and security systems; monitoring temperature, pressure, light intensity, humidity, moisture, smoke, heat and fluid levels.

Prices: The BH100 General Purpose Input/Output Interface Card, $129.00; The Complete Beginner's I/O Interface Course, $159.00; the Course Manual alone, $15.00; The BH100-S Security System Module, $25.00; the Analog-to-Digital Conversion Module, $30.00 (1 channel) and $45.00 (8 channels); VIC 20 adapter, $10.00. All prices are in U.S. dollars. A free brochure is available by calling (315) 265-6350, or write to:

Intelligent I/O, Inc.
P.O. Box 70
Potsdam, NY
13676 (315) 265-6350

www.Commodore.ca
May Not Reprint Without Permission

Extending BASIC for Telecommunicating

SoftTools of Montreal has announced the release of its first product, The Boss, a BASIC extension for the Commodore 64 that adds over 40 new commands and functions to BASIC V2. Most of the added commands are designed to facilitate data communications programming.

Originally designed to provide an electronic bulletin board system with machine language speed, The Boss includes commands to perform input/output operations with a modem, and also provides disk support. Among the former group are commands such as SEND, GETLN, HANGUP, CARRIER and DIAL, with which you can send lines to a modem, get user inputs of specified lengths from the other end, turn a modem on or off, check for carrier, and dial a phone number on 1650-compatible or Mitey Mo modems. The Boss handles all ASCII translation, and also provides for accurate time-keeping by using the built-in system timers. Among the disk commands are DEVICE, SEARCH and DISKIN#, to set the disk device number, search the directory for a certain type of file, and get lines from a disk file including commas, colons and quotation marks.

Sample programs on the disk include a small terminal program, a bulletin board system and a disk management system, all written in BASIC using The Boss. The Boss is documented with a reference guide that explains each keyword in detail. The Boss may be ordered directly from SoftTools for $35.00, which includes postage and handling. Address all inquiries and orders to:

SoftTools
Snowdon P.O. Box 1205
Montreal, Quebec
H3X 3Y3 (514) 793-3046

Digital Sound, Digital Drums

Micro Arts Products is now shipping two new digital sound sampling products for the Commodore 64: the SAMPLER-64 digital sound sampler/editor and the COM-DRUM sampled digital drum software.

The SAMPLER-64 lets you do things like record your dog's bark, then mix in your own voice, add a little echo or reverb, mix the sound further, then play your new sound over two octaves from the computer's keyboard in any melody or non-melody you'd like. The melodies can be recorded into the sequencer and stored on disk along with your sound samples.

The SAMPLER-64 comes with a small hardware unit that plugs into the user port of the Commodore 64 (the SID chip is not used), a microphone (sounds can also be recorded from line level signals), a cable, and menu-driven software on disk.

The COM-DRUM software turns the SAMPLER-64 hardware unit and the Commodore 64 into an eight piece drum kit using pre-recorded drum sound samples supplied on the COM-DRUM disk. The COM-DRUM has two sequencers: a real-time sequencer for sounding out a rhythm on the computer keyboard and storing it to disk, and a step-time sequencer for extensive on-screen composition and editing of a rhythm track. The COM-DRUM allows for any 3 percussive samples to be sounded simultaneously. Included with the software are 3 different 8-piece drum kit samples: rock, latin, and what the manufacturer describes as "something that sounds like a Tupperware party".

The SAMPLER-64 is sold by mail for $89.95 US plus $3.50 shipping and handling. The COM-DRUM sells for $14.95 when purchased with the SAMPLER-64 (Philadelphia residents must add 6 per cent sales tax). Visa and Mastercard are accepted. Contact:

Micro Arts Products
P.O. Box 2522
Philadelphia, PA
19147 (215) 336-1199.

Do-it-yourself Amiga Calculator

If you've always wanted to own your own calculator but went and blew the money on an Amiga instead, you might want to check out Quicksilver Software's debut product: Calculator Construction Kit, designed to let you replace the Workbench calculator with the customized number-cruncher of your dreams. The program lets you build your own calculator by dragging buttons into place to suit your taste. More than 80 functions are available to choose from. A new and different calculator can be built at any time.

Among the options are different number bases (binary, octal, hex and decimal) and a print capability for hardcopy printouts. Quicksilver says their product will serve special needs such as financial and surveying calculations, and reverse Polish notation.

The price of the non-protected program is $49.95 (US) plus $3.00 handling, plus $4.00 for C.O.D. Call (712) 258-2018 or write to:

Quicksilver Software
418 West 7th Street
Sioux City, Iowa 51103

Interrogate, Modify and Trace

I/M (Interrogator/Modifier) is a new Commodore 64 product from Innovative Software that shares some of the main features of a regular machine language monitor, such as a disassembler and hex/ASCII dumps.

One feature that sets it apart is its Hunt command. In an ML monitor, a Hunt lets you search for a string of hex bytes or ASCII characters. I/M lets you search instead for a 6502 opcode (entered as a mnemonic) or an addressing mode. This approach avoids the ambiguity between opcode and operand bytes that in a standard monitor can result in you finding many false matches for a particular Hunt.

The Modifier portion of the program lets you replace old addresses and/or opcodes with new ones. This is useful for patching machine code for which you do not have the source.

The package also includes three separate tracers (command, floating and single step), each of which comes in multiple version for different locations in memory. These provide an incorruptible address display in the upper left corner of the screen. Source code for the tracers, along with a few other auxiliary utilities, is included on the disk.

The price for I/M is $24.00 (US), plus $2.00 postage and handling. Make your check or money order payable to:

Innovative Software
530 North 9th Street
Reading, PA, 19604 (216) 372-5438

BusMate from ICS

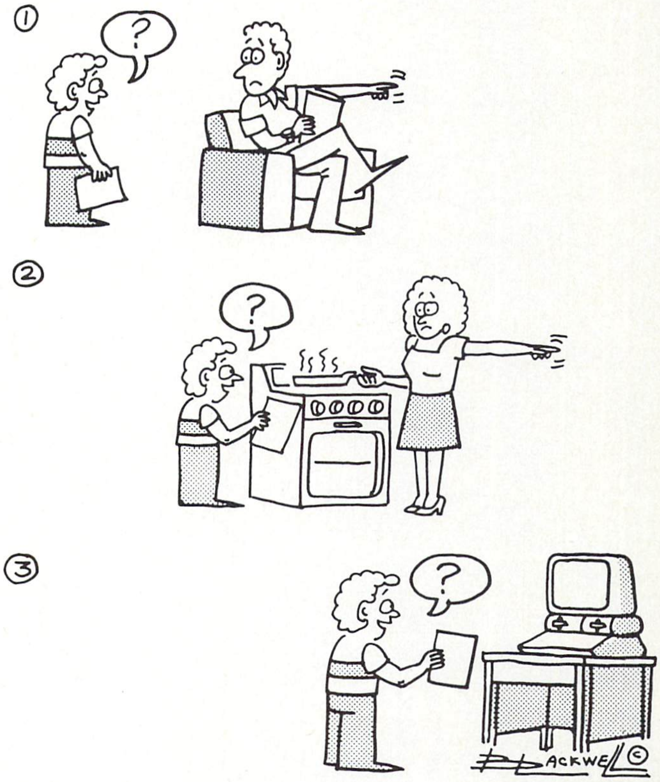
San Jose, CA -- ICS Electronics Corporation has introduced BusMate, a plug-on addition that turns any personal computer with an RS-232 serial port into a full-featured IEEE 488 Bus controller capable of operating up to 14 independant devices. (The IEEE 488 is a bus standard used extensively for scientific instruments; several Commodore floppy disk drives also use IEEE 488 communications.) BusMate is self-contained and self-powered, and provides full control of instruments connected to the 488 bus without taking any control of the personal computer; it is operated completely through the serial port.

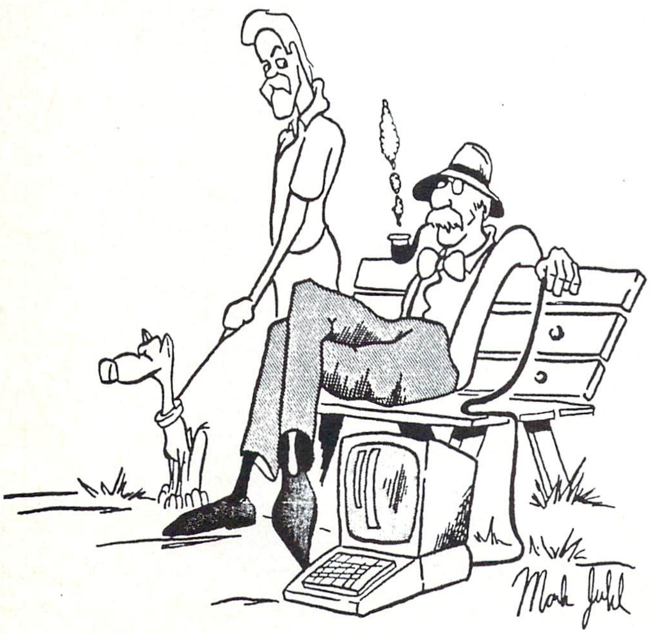
Price is $695 (U.S.) in unit quantities and delivery is from stock to 45 days. Rack mounting kits and various lengths and type of interconnection cables are available as options. For more information, contact:

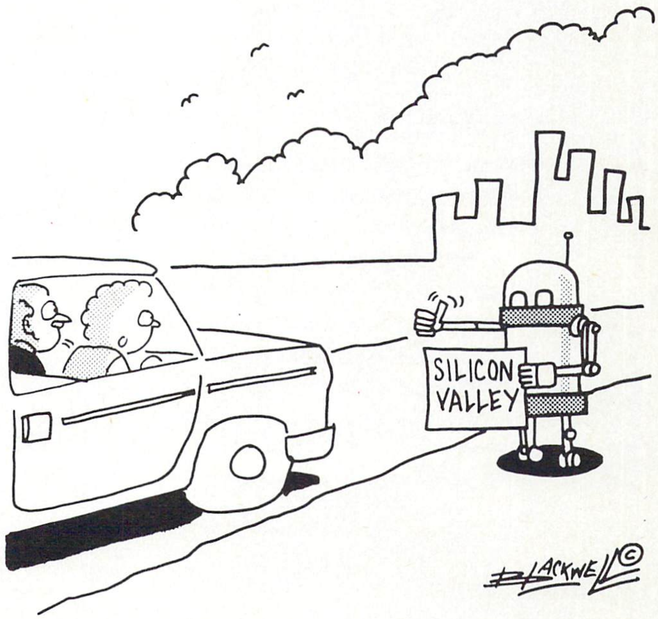
ICS Electronics Corporation
2185 Old Oakland Road
San Jose, CA 95131

www.Commodore.ca
May Not Reprint Without Permission

Computoons

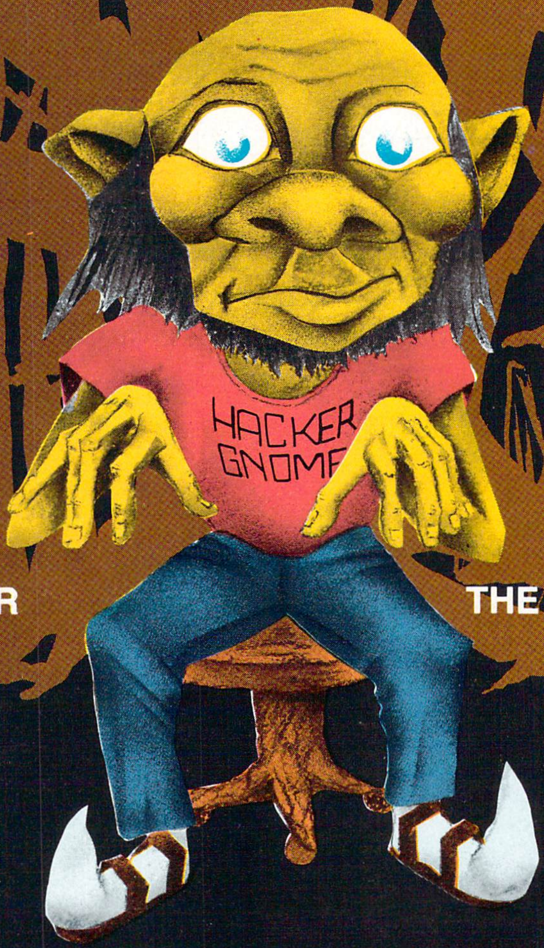






www.commodore.ca
May Not Reprint Without Permission

We don't need to name Gnomes—every Gnome knows that it's Hacker Gnome's wizardry that will not only transform your programs into super fast and compact Gnome Code, but will also cut your programming time in half (leaving you time for more gnomely things).



GNOME SPEED

THE BASIC 7.0 COMPILER

Compile virtually any BASIC program into super fast, compact Pseudo-Code. Simple to use. Easy error correction and powerful directives for compacting code, optimizing speed and producing indispensible programming testing and aids. Whether developing games or serious applications for your own use—or to sell—no gnome should be without this compiler.

C-128 **GNOME SPEED** $59.95 U.S.

KIRA CORP.

GNOME KIT

THE PROGRAMMING TOOL KI

The programming tool kit is a comprehensive set of utilities that provides an unmatched range of features for BASIC 7.0, 2.0 and Machine Language programming and Direct Access DOS manipulation. Full Merge, Find, Selective Line Renumbering, Extended DOS Wedge, Extended Machine Language Monitor and Disk Editor are just some of the features in this transparent programmer's utility. Another must for serious gnomes.

C-64, C-128 **GNOME KIT** $39.95 U.S.

KIRA CORP.

NO COPY PROTECTION

U.S Mail Orders:
BRIWALL
P.O. Box 129-Dept. 87
Kutztown, PA 19530
(215) 683-5433 (24 hr. service)

Canadian Orders:
THE TRANSACTOR
(416) 878-8438

See Order Card

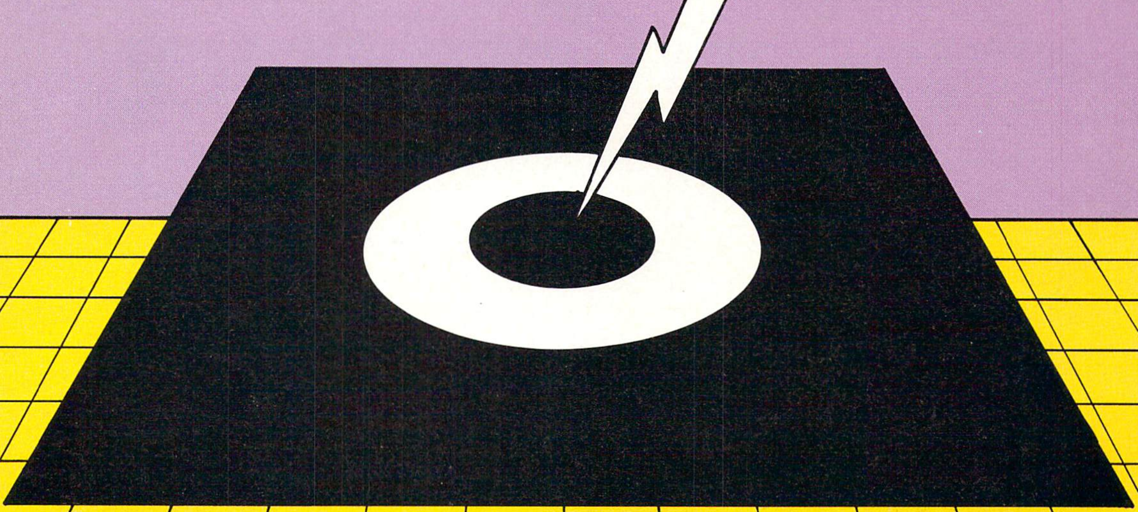
Dealer Inquiries:
Micro-Pace, Inc.
(217) 356-1884

www.Commodore.ca
May Not Reprint Without Permission

INTRODUCING

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

SUPER KIT/1541 is for archival use only! We do not condone nor encourage piracy of any kind.

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.

www.Commodore.ca
May Not Reprint Without Permission

THE WORLD OF COMMODORE

DECEMBER 4, 5, 6, 7, 1986

INTERNATIONAL CENTRE

TORONTO

• Seminars • New Products • Savings