The Transactor

The Tech/News Journal For Commodore Computers

95% Advertising Free! Sept. 1985: Volume 6, Issue 02. $2.95

Networking and Communications

- Commodore 64 Remote Control Program!
- Accessing The TDD Network: Software & Hardware
- Hardware: Easy Intercomputer Connection
- More On The Save With Replace Bug

- Tele-Tone 64: A Touchtone Synthesizer
- The Commodore RS232 Bus Demystified
- Worldwide Telecomputing via HAM Radio
- A Review of 3 Popular Multi-User Systems
- More 1541 Fun: Recovering from the Error Blues
- Flexible Vectors: An Intelligent Approach to Vectoring

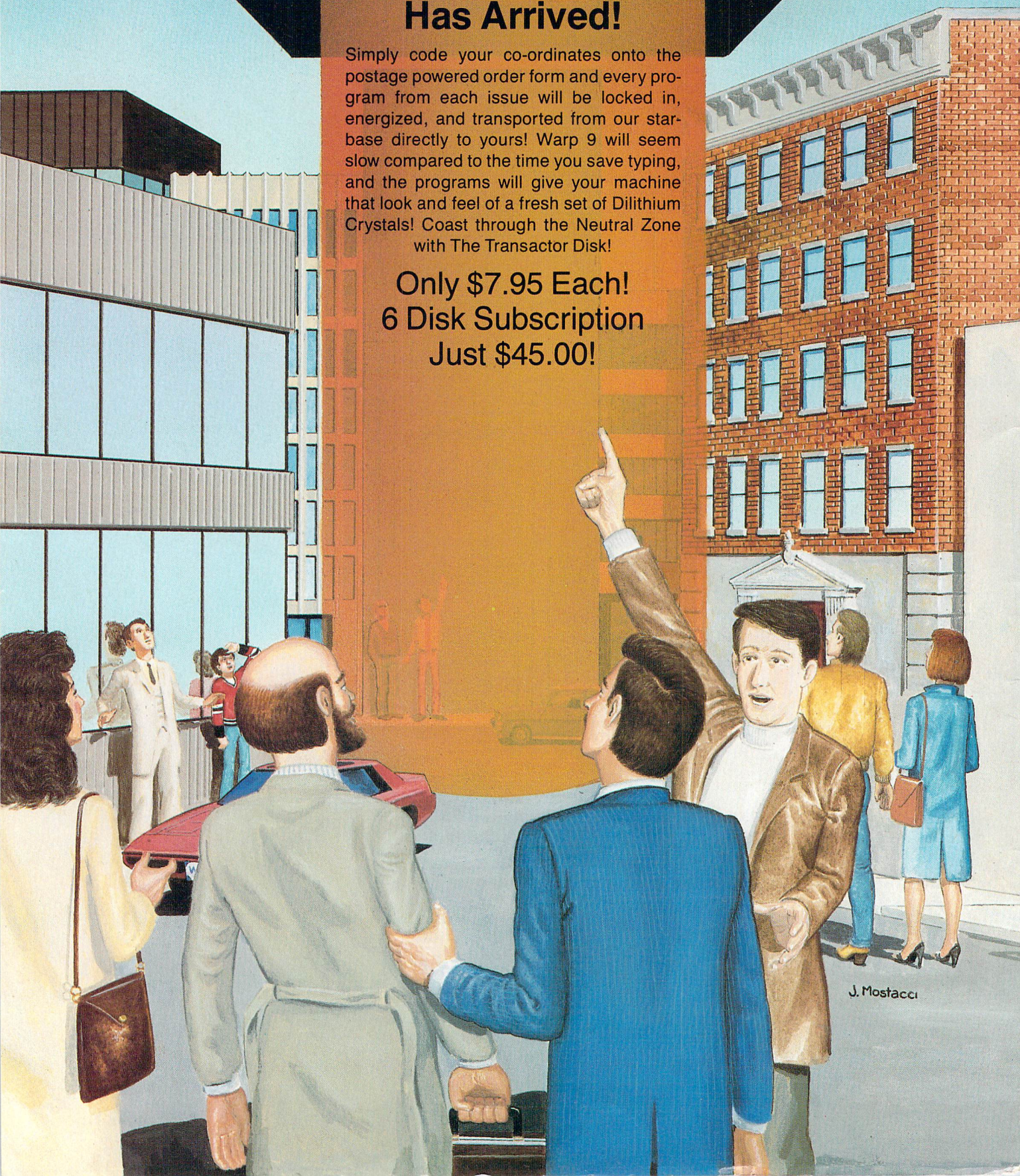
0 72246 00594 09

The Transactor

Disk Has Arrived!

Simply code your co-ordinates onto the postage powered order form and every program from each issue will be locked in, energized, and transported from our starbase directly to yours! Warp 9 will seem slow compared to the time you save typing, and the programs will give your machine that look and feel of a fresh set of Dilithium Crystals! Coast through the Neutral Zone with The Transactor Disk!

Only $7.95 Each!
6 Disk Subscription
Just $45.00!



Volume 6
Issue 02
Circulation 64,000

The Transactor

Start Address Editorial 3

News BRK . . . 69
Delivery Information
Submitting NEWS BRK Press Releases
Transactor on Microfiche
Inner Space Anthology Discounts
Dry Mail
C-64 Users Group Of Canada
And The Winner Is. . .
C Experts Gather For Fall Seminar
International Communications and Computer Exhibition
Paperback Writer
Sixth Sense
C64 & APPLE II+ Compatibility
'C POWER' For The 64
PROMAL - New Structured Programming Language
Freedom Assembler/Monitor
Sight & Sound Expands Floppies To Flippies
Keyboard Chord/Scale Master
Light Pen Reading Series from MicroEd
Master Software Releases Reset Master
ABL-64 (Automatic Boot-Loader Cartridge for the 64)
Studio 64 Gets New Keyboard
Black Box Modem 1200's at Half Price
New Markers Write Safely On Diskettes

Bits and Pieces . . . 4
C64 Keyboard Joystick Simulation
1-Line SEQ file read
C-64 Character Flash Mode
Plus 4/16 Pretty Patterns
C-64: Text on a Hi-Res Screen
"Someone's coming" or "Boss" mode
Fast Key Repeat
Modem Speed-Up
1200 Baud Fallacy
B to PET/CBM Program Converter
C64 Screen Sizzle
C64 Simple Banner Program
Break Box Baffler

Letters 8
Available deVICes?
Plotter Plight
B Users Group
1541 Alignment Notes
Brown Bibles For Sale
Piracy vs. The Software Publishers Association
Public Right vs. Copyright
The Copyright And You

Book Review: Using Compuserve 11
TransBASIC Installment #4 12
Telecomputing: From Concept To Connect 18
A Comedy of Errors The trials and tribulations of connecting a Modem . . 24
World Communications Computer meets Shortwave Radio 26
The Electronic Mailbox The wired office 28
Networking Systems Specifications on 3 popular networks 30
Helping to Communicate – The TDD Network . 34
Easy Intercomputer Connection Let two C-64s talk to each other 40
Remote 64 Control your 64 from a remote terminal 42
Riding the RS-232 Bus All About Commodore RS-232 45
The BBS Link A Database For Bulletin Boards 50
Tele-Tone 64 A Synthetic Telephone Using Your SID Chip 52
Fun With Your 1541 Programming the Disk Drive CPU 55
Simulating a Dual Drive Make two 1541s Behave as a Dual Drive . . . 60
In Defence Of The Frontal Assault An Editorial 62
Vectoring Vector Programming Techniques 63
Save @ Exposed: The Debate Continues 68
Compu-toons 76

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

The Transactor
The Tech/News Journal For Commodore Computers

Managing Editor
Karl J. H. Hildon

Editor
Richard Evers

Technical Editor
Chris Zamara

Art Director
John Mostacci

Administration & Subscriptions
Lana Humphries

Contributing Writers
Gary Anderson
Don Bell
Daniel Bingamon
Anthony Bryant
Jim Butterfield
F. Arthur Cochrane
Gary Cobb
Elizabeth Deal
Domenic DeFrancesco
Tony Doty
Robert Dray
Mike Forani
Jeff Goebel
Jim Grubbs
Gary Gunderson
Bob Hayes
Thomas Henry
David A. Hook
Chris Johnsen
Garry Kiziak
Scott Maclean
Mario Marrello
Chris Miller
Brian Munshaw
Gerald Neufeld
Michael Quigley
Howard Rotenberg
Louis F. Sander
K. Murray Smith
Darren J. Spruyt
Aubrey Stanley
Nick Sullivan
Tony Voleri
Charles Whittern

Production
Attic Typesetting Ltd.

Printing
Printed in Canada by
MacLean Hunter Printing

The Transactor is published bi-monthly by Transactor Publishing Inc., 500 Steeles Avenue, Milton, Ontario, L9T 3P7. Canadian Second Class mail registration number **6342.** USPS **725-050,** Second Class postage paid at Buffalo, NY, for U.S. subscribers. U.S. Postmasters: send address changes to The Transactor, 277 Linwood Avenue, Buffalo, NY, 14209, 716-884-0630. **ISSN# 0827-2530.**

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

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

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

SOLD OUT: The Best of The Transactor Volumes 1 & 2 & 3; Vol 4, Issues 04, 05, 06, Vol 5 Issue 03
Still Available: Vol. 4: 01, 02, 03. Vol. 5: 01, 02, 04, 05

Program Listings In The Transactor

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

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

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

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

Cursor Characters For PET / CBM / VIC / 64

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

Colour Characters For VIC / 64

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

Function Keys For VIC / 64

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

Quantity Orders:

CompuLit
PO Box 352
Port Coquitlam, BC
V5C 4K6
604 438 8854

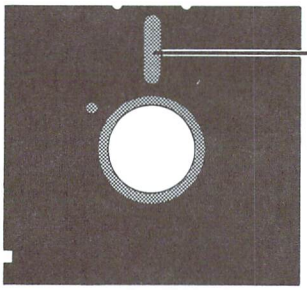
Micron Distributing
409 Queen Street West
Toronto, Ontario, M5V 2A5
(416) 593 9862
Dealer Inquiries ONLY:
1 800 268 9052
Subscription related inquiries are handled ONLY at Milton HQ

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)

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.



Start Address

June 1 three years ago was a Monday, the day I pushed The Transactor from its nest at Commodore Canada. Exactly one year ago we were signing the contract for international distribution. And today we ship to over 35 countries outside North America. Although our print run has stabilized at about 64,000, The Transactor is by no means coasting. Our third year-end figures indicate that now more than ever is the time to forge ahead. There are other reasons too.

Another way too cold Ontario winter has passed, and another way too short Ontario summer is upon us. As this third summer picks up speed, an expected trend is failing to make itself apparent. Computing, in all its many facets, is typically a winter activity, or so we thought until this summer. Being subjected to vast quantities of crystallized and solid water tends to make one more appreciative of the fact that it also comes in two other states; liquid and gaseous. Words like 'hot', 'humid' and 'sticky' are music to the ears. Even a dozen SID chips could not compete. The computer hobby would not completely grind to a halt, but the click of a driver could be heard more often than the clack of a drive. It seems that the seasonal ingredient, however, is becoming less influential. While others are reporting decreases, we're happy to announce that both the temperature AND subscriptions are still on the rise.

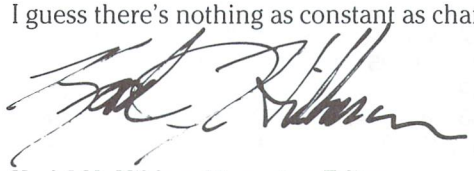
I suppose there are other factors involved. Now that we send Transactors to places that have no *real* winter season may be partially responsible. The Transactor Disk might also take some credit. But the general trend of the market just cannot be ignored. The number of home computer sales could probably be compared to the number of new home sales. The impulse buying trend of three years ago has been replaced by the 'educated acquisition'. Those purchasing computer systems today are more serious about computing from the start, as opposed to building towards a serious level. In our own humble opinion, we believe this is our domain, and though we can't claim first or second spot on the charts, we will not let it slip from our sights.

Telecomputing is also becoming less seasonal. Bulletin Board Systems that greet you with a busy signal all winter are busy all summer too. Many readers have asked why The Transactor has not installed a BBS of our own. Reasons include the unavoidable problem of contention for that one single line and the long distance aspect. We think we may now have a solution. Currently we are investigating the possibility of our own section on the Delphi Network. Unfortunately, the idea of an article detailing Delphi came to us too late for this issue, but by this time next issue we'll have plenty more to say. I met briefly with John W. Gibney, Delphi's key proponent and formerly of CompuServe, who is at this moment preparing a package for us to peruse. Several angles will be studied before we 'take the plunge', but based on Delphi's connect-time rates and their geographic availability, a Transactor SIG is a promising outlook.

Last issue I feel I may have been a little hasty with the demise-of-Atari overtones. At the Computer Fair in Toronto, Atari was out in full force. It's now obvious to me that Jack and crew are as genuine in their intentions as any new outfit. Besides that, the ST looks like a fabulous machine. The 68000 based Atari has a 24 bit address buss allowing for over 16 million bytes of addressable memory without the hassles of bank switching and the like. RS232 and Centronics parallel are there as well as plenty of other peripheral I/O. But their most impressive announcement had to be a thing they presently call the 'C.D.ROM'. According to J.T., this mysterious name will adorn a unit with over 500 megs of storage for about $500, sometime this fall.

Although we have no plans for regular coverage of the new Atari, a not too distant Transactor will feature a technical overview of the ST. We thought if we were interested, you might be too. If that doesn't quench your curiosity, contact Atari. Neil Harris, formerly of Commodore's publishing department, is now producing 'The Atari Explorer', and doing a fine job I might add. Atari HQ is in Sunnyvale, CA, and they also have a Toronto office. On a final note, I can't help wondering if the ST wouldn't have been a new Commodore product if Jack were still skipper.

I guess there's nothing as constant as change. But I remain,



Karl J.H. Hildon, Managing Editor

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 two versions of VERIFIZER on this page; one is for the PET, the other for the VIC or 64. Enter the applicable program and RUN it. If you get the message, "***** data 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, enter NEW, then turn VERIFIZER on with:

SYS 828 to enable the C64/VIC version (turn it off with SYS 831)
or SYS 634 to enable the PET version (turn it off with SYS 637)

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 it means we've editted that line at the last minute which changes the report code. However, this will only happen occasionally and 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 (eg. POKE 52381,0 instead of POKE 53281,0), but 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: 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.

Listing 1a: VERIFIZER for C64 and VIC-20

```
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
```

Listing 1b: PET/CBM VERIFIZER (BASIC 2.0 or 4.0)

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

Bits and Pieces

C64 Keyboard Joystick Simulation

If you ever need to try out a joystick-driven program but you don't have a joystick plugged in, you can simulate the stick by using the keyboard. The keyboard can be used instead of the joystick in port 2 by holding down the space bar while pressing C,Z,B,M or F1 to perform the functions in the table below. The port 1 joystick can't be simulated in some video games, like those which use the keyboard as well as the joysticks, but in most programs you can get joystick 1 functions by just pressing a single key – refer to the table below.

JOY2:

space + C = left
space + Z = down
space + F1 = up
space + B = right
space + M = fire

JOY 1:

CTRL = left
← = down
1 = up
2 = right
spc = fire

1-Line SEQ file read

You've probably typed in a little sequential file-read program many times. Although utilities such as BASIC AID and POWER have such a feature built in, the utility isn't always installed when you need to look at a file. To save typing in several program lines whenever you wish to view a sequential CBM ASCII file, here's a short program to do it. It will print the file to the screen and stop and close the file when the end is reached. You can just tack this line to the beginning of the program in memory and delete it when you don't need it anymore.

```
1 open8,8,8,"filename":fori=0to1:get#8,a$:i=st
  :printa$;:next:close8:end
```

C-64 Character Flash Mode

One of the many features of the plus 4 and 16 machines is a "flash" mode, which operates like reverse on/off, but causes all characters printed in that mode to continuously flash at the rate of the cursor. Flashing is a great way to highlight important text, signal an error condition, etc. Below is a program to simulate flash mode on the C-64. One of the 16 text colours becomes the "flash" colour; anything printed in the flash colour (default green) or the current background colour will blink at approximately the same speed as the cursor. Line 65 sets up the flash colour as 5 for green – change it to whatever you wish.

```
NN 10 rem* data loader for "flash" *
LI 20 cs=0
CG 30 for i=49152 to 49245:read a:poke i,a
DH 40 cs=cs+a:next i
GK 50 :
HI 60 if cs<>12150 then print"!! data error": end
JI 65 poke 49152+19,5 :rem flash colour=5 (green)
DD 70 sys 49152
AF 80 end
IN 100 :
NA 1000 data 173, 21, 192, 141, 22, 192, 120, 169
KO 1010 data 24, 141, 20, 3, 169, 192, 141, 21
GO 1020 data 3, 88, 96, 5, 0, 20, 0, 0
LI 1030 data 206, 22, 192, 208, 61, 173, 21, 192
ID 1040 data 141, 22, 192, 173, 33, 208, 41, 15
BJ 1050 data 141, 20, 192, 160, 0, 132, 251, 169
NG 1060 data 216, 133, 252, 238, 23, 192, 173, 23
GB 1070 data 192, 41, 1, 170, 177, 251, 41, 15
OH 1080 data 205, 19, 192, 240, 5, 205, 20, 192
GK 1090 data 208, 5, 189, 19, 192, 145, 251, 200
NF 1100 data 208, 234, 230, 252, 165, 252, 201, 220
AL 1110 data 208, 226, 76, 49, 234, 252
```

Plus 4/16 Pretty Patterns

Here's a short one. Try changing the step value for different effects, and the values of 'B' and 'E' for different sizes.

```
10 graphic1,1 : b=20 : e=190
20 for i=b to e step7 : draw 1,b,i to i,e : next
```

This next one gives a different pattern each time. After a pattern is drawn, press any key for a new one. Try a few – some are pretty incredible. It works by drawing boxes of different sizes rotated at different angles, thanks to the flexible BOX command in BASIC 3.5.

```
100 rem* +4 boxspiral -cz *
110 graphic 1,1: color 1,1
120 x1=0:y1=0:x2=100:y2=100
130 n1=rnd(0)*10: n2=rnd(0)*10
150 for angle=0 to 180 step 5
160 box 1,x1,y1,x2,y2,angle
170 x1=x1+n1: y1=y1+n2
190 next angle
200 rem* run again when key pressed
210 getkey a$: run
```

C–64: Text on a Hi–Res Screen

The hi–res screen is so much more fun than just plain, boring old text. You know, a picture is worth... But we work with words and numbers so much that it's sometimes hard to give meaning to a diagram such as a bar graph without words of explanation and numbers for scales. The subroutine below lets you label your creations by displaying a given ASCII character on a hi–res screen. The character must lie in one of the usual character cells (25 down by 40 across). Before calling the routine, specify the column (0 to 24) in 'CY' and the row (0–39) in 'CX'. The character itself must be in the variable 'CC$'. The program copies the eight–byte character definition from ROM into hi–res screen memory addressed at $2000.

```
1000 rem* put text on hi-res screen *
1010 rem* character rom
1020 rom = 13*4096 + 1024*(peek(53272)and2)
1030 c = asc(cc$): print " S ";
1040 rem*  convert ascii to screen code
1050 cc = c + 64*(c>64andc<192) + 128*(c>191)
1060 rem turn off irqs and select character rom
1070 poke56334,peek(56334)and254
1080 poke1,peek(1)and251
1090 rem* copy from character rom to hires screen
1100 br = rom + cc*8:bs = 8192 + cy*320 + cx*8
1110 for i = br to br + 7:poke bs,peek(i)
1120 bs = bs + 1:next
1130 rem* switch back i/o in place of char rom
1140 poke1,peek(1)or4
1150 poke56334,peek(56334)or1
1160 return
```

Subroutine notes:

1) Line 1020 chooses upper/lowercase or uppercase/graphics mode for displaying the character, depending on the current mode.
2) If the hi–res screen is located in memory somewhere other than $2000, change the '8192' in line 1100 to the actual location.

"Someone's coming" or "Boss" mode

For those of us who work with computers as an occupation, it's hard to load up a game for a bit of stress–relief without feeling some guilt. If you work in an office, you may find yourself looking over your shoulder between blasting meanies in space – some stress relief.

To let you play at ease, several games for the IBM PC (which are primarily used for business – no having fun allowed) have a "someone's coming" mode. When you hit the "boss" key, the game instantly disappears from the screen and is replaced by a fake spreadsheet, word processor or bar graph display. When the big guy once again leaves the room, you can continue your game right from where you left off with another strike of the boss button.

Sounds like a good idea. Might be good for the home computer in case you're killing klingons when you should be cutting the grass. When your wife looks in on your progress, just hit the button and, "just a minute dear, have to balance last month's budget first." To cover all bases, maybe every game should have "boss", "spouse", and "parent" functions built in. Well, game developers? How about it?

Fast Key Repeat — David Jankowski, Manoora Australia

David (age 11) writes:
"I would like to submit this small interrupt–driven routine. Its purpose is to speed up the keyboard repeat (about 74% faster) for game programs that use the GET command to receive instructions. The program sits in the cassette buffer and runs on the C64.

```
5 rem c64 fast key repeat
10 for i = 828 to 847: read a: poke i,a: next
20 data 120, 169,  3, 141, 21,   3, 169, 73, 141,  20
30 data   3,  88, 96, 169,  0, 133, 197, 76,  49, 234
```

For the VIC, change the second last value in line 30 (49) to 191.

Modem Speed–Up

Daniel Bingamon of Batavia, Ohio gives this command to speed up a 1600, 1650 or 1660 modem to 450 baud:

```
open 5,2,3,chr$(0) + chr$(0) + chr$(12) + chr$(4)
```

Now you might be asking, "Who would you connect to at 450 baud? Most are either 300 or 1200." Well, the sequence above has been around just long enough for some authors, like Steve Punter, to make provisions in their bulletin board software. Once connected, you have the option of changing to the higher speed.

1200 Baud Fallacy

Have you ever been told that 1200 baud transmission is too fast for normal telephone lines? The Phone Company, and their gullible subscribers, will rhyme off a rather technically believable line like, "the bandwidth of the signal encoding equipment is not wide enough to handle some frequencies at 1200 bits per second - you need a special line installed to avoid dropouts", which costs you more, of course.

Don't believe it! I have talked to bulletin board systems as far as 3000 miles away with absolutely no trouble. In fact, the entire Transactor magazine is sent over a regular garden variety phone at none other than 1200 baud - no sweat.

True, most 1200 baud modems are somewhat overpriced but there are some deals to be had. A Milton based firm offers one for about $400 CDN. (Contact The Personal Computer Store on Steeles Avenue , Milton Ontario.) And once you start downloading at 4 times the speed you're familiar with, you'll be spoiled for life. Actually, 1200 is becoming quite popular. Some systems will even detect your transmission speed at connect time and automatically adjust themselves to suit.

B to PET/CBM Program Converter

A quick B fact. Basic programs SAVEd by a B machine have a start address of $0003 in zero page. The B machine accepts and relocates PET/CBM Basic programs as if they were its own, just as the Vic and C64 do. But just try to LOAD the Basic program back into the PET or CBM. It will destroy zero page, the stack, and whatever else lies in its wake depending on the size. A horrible awakening for PET people, until now. The program listed below will take your Basic B program

and relocate it for the PET or CBM. It will re-create a new Basic program on diskette starting at $0401, with each link address also correctly relocated. A pretty terrific utility. Our special thanks to Jack Weaver of Input Systems Inc. in Florida for this one.

```
PH  100 rem ** change b-128 program to run on 80/4032
FL  110 rem ** jack weaver  input systems, inc.
LF  120 rem ** 15600 palmetto lake dr. miami fl 33157
CD  130 rem ** phone (305) 252-1550
AA  140 :
IA  150 cu$=chr$(145): cd$=chr$(17): cl$=chr$(157)
        : ry$=chr$(18): rn$=chr$(146)
FK  160 c$=chr$(0)
KH  170 open 15,8,15,"i0
AM  180 def fn r(x)=(b2*256+b1)+1022
IP  190 if d0
JD  200 print cu$;ry$;" name of b-128 prog to
        change "rn$"  ";
FA  210 input of$
HO  220 print cd$" new name of the 80/4032 program  ";
GB  230 input nf$
AO  240 print "b-128 prg =  "ry$;of$
        : print "80/4032 prg =  "ry$;nf$
GD  250 print cd$;cd$"OK  (y/n)  n"cl$;cl$;cl$;
FH  260 input yn$
OF  270 if yn$<>"y"  then stop
HM  280 a=1025: open 4,8,4,of$: if ds then print ds$: stop
ED  290 get#4,a$,b$: a1=asc(a$+c$): a2=asc(b$+c$)
        : if a1<>3 then next: stop
OI  300 open 5,8,5,"@0:"+nf$+",p,w": if ds then
        print ds$: stop
FJ  310 print#5,chr$(1)chr$(4);
AC  320 if s then 390
HK  330 get#4,a$,b$: b1=asc(a$+c$): b2=asc(b$+c$)
BL  340 x=fn r(x): if x=1022 then s=1: goto 390
        : rem check for end of prg
OD  350 hi=int(x/256): lo=x-hi*256
        : print ry$;a;rn$;b1;b2;x;lo;hi
FJ  360 a=a+2: print#5,chr$(lo)chr$(hi);
KE  370 if a=x then 320
EC  380 a=a+1
PM  390 if s then print#5,c$;c$;c$;: close5: close4
        : print "converted!": end
PD  400 get#4,a$: s=st: print#5,chr$(asc(a$+c$));
BF  410 goto 370
```

C64 Screen Sizzle — **James Cashin, Corner Brook, Newfoundland**

For the Bits & Pieces obligatory screen blitz, we are proud to present this little two-liner for the 64 from James, alias the 'Happy Hacker':

```
10 poke53280,0:poke53281,1:printchr$(147):poke53281,0
   :poke53272,18:cs=2304
20 fori=0to1step0:b=rnd(1)*256:forj=1to7:pokecs+j,b:next j,i
```

The program fills the screen with spaces, and continually changes the character definition for a space. If you've never thought that staring at a CRT can be nasty to your eyeballs, try this one out. A starving optometrists' delight!

C64 Simple Banner Program — **Jeremy Stewart, North Bay, Ontario**

Here's a short banner program for the C-64 and a printer. The message can be up to 255 characters long, and will reproduce any character including graphics.

```
100 rem**  banner by jeremy stewart  **
101 rem** for c64 and 80-col printer **
105 :
110 l=53248: open1,4
120 as$="*********":sp$="                "
130 rem 9 asterisks, 9 spaces -reduce for shorter characters
135 :
140 input"■input message";m$:print"■"m$
150 for y=1024 to 1023+len(m$):n=peek(y)
160 for z=1 to 8:a$(z)="":next z
170 poke 56334,peek(56334)and254
180 poke 1,peek(1)and251
190 for a=7 to 0 step-1:b=2↑a
200 for c=(l+(n*8)+7)to(l+(n*8))step-1
210 p=peek(c): x=abs(a-8)
220 if(p and b)=b then a$(x)=a$(x)+as$: goto 240
230 a$(x)=a$(x)+sp$
240 next c,a
250 poke 1,peek(1)or4
260 poke 56334,peek(56334)or1
270 for j=1 to 8:for k=1 to 4:print#1,a$(j):next k,j
280 next y:close1
```

Notes:
1) Alter the height of the letters as indicated in line 130.
2) Use a character other than asterisks in line 30 for different effects.
3) Change the 'for k=1 to 4' loop in line 270 for wider or narrower characters.

Break Box Baffler — **Tom Johnson, Jefferson, MO**

Here's a terrific bit of code to retard the code buster blues. Written for the Commodore 64, you will find great benefits in locating it high up at $8000 in RAM. This is the area looked at during system reset for the presence of a cartridge. If the correct code is present, ie. CBM 80 etc., then it will be executed. Tom's code takes advantage of this trick. Upon an NMI break in, the code will be executed, thus throwing the 64 into an endless loop. Pretty bad news all packed into 50 bytes.

```
63000 for i=32768 to 32818 : read j : poke i,j : next i : return
63010 data   9, 128, 216, 128, 195, 194, 205,  56
63020 data  48, 120, 169, 128, 162,   9, 141,   3
63030 data   3, 142,   2,   3, 141,  21,   3, 142
63040 data  20,   3, 141,  23,   3, 142,  22,   3
63050 data 141,  25,   3, 142,  24,   3, 141,  41
63060 data   3, 142,  40,   3, 169,  48, 133,   1
63070 data  76,   9, 128
```

Letters

Available deVICes?: I have read with great pleasure the fifth issue of your magazine because I am a VIC 20 owner. I was particularly interested by the paper signed by A.J. Barlaan on page 70 about the Audio/Video cable adapter. This author is talking about a 40/80 column adapter for the VIC. Since the accessories for this microcomputer seem to become harder to find, I would greatly appreciate you sending me the address of the company "Data 20" which is distributing that device.

Real Gagnon, Neufchatel, Quebec

Unfortunately for VIC owners, many companies with VIC accessories have "gone away". Before sending any money for VIC products (or any item for that matter) please check that the company exists. You may also want to check with a local computer club. Data 20 may still be in business but we've seen very little activity under that name and strongly doubt it. Their last address was in Laguna Hills, CA. Call long distance information (1-area code-555 1212) - it costs you nothing and may save you much.

Plotter Plight: I have purchased the last issue of your magazine and admired it very much. I am impressed with your reviews. Under your editorial Letters I wish to express my problem. I have recently purchased a Commodore 1520 Plotter/Printer but, alas, I am not a programmer. I need the address of any person or firm that produces software that will allow me to use the unit as a simple printer or to use it as a plotter. I am well aware of its drawbacks, but I thought it would be nice as a specialty printer. Any assistance you can give me would be most appreciated.

Dan E. Hedgpeth, 10036 Ben Nevis Blvd., Riverside, CA 92509

Off the top of our noggins we can produce little to nothing in the form of plotter software. With luck this letter will be read by many, of which a few may have the answer. If this is the case, we would appreciate if you could either send us or Mr. Hedgpeth a letter directly. In all probability, there is more than one person looking for some plotter software. Drop us a line and we'll spread the word.

B Notes: In reply to the note concerning the B128 on page 10 of V5, #6, there are two manuals, not very technical, available for the machine. Actually, they are the same, except for one difference.

Commodore Business Machines (UK), 1 Hunters Lane, Corby, Northants, England has one available for 12 pounds, 60 pence, air postage included, and Protecto has what appears to be a copy of the UK manual. This version costs $25.00 US, but it also includes circuit diagrams, which the UK version does not, so it may be the better buy of the two, even at the higher price.

When I ordered the UK version, I asked if they had anything else for the 700 (as they style the B128), but received no reply to that question.

Lawrence Williams, San Antonio, Texas

It's nice to know a few B people are reading. Thanks for the tip. The B buffs will appreciate it.

B Users Group: There are over 9,000 B's in the USA, and well over 25,000 in Europe. Since Protectos sale of this machine, a B128 Users Group has been established. As of last week, the membership was well over 500 B128 owners. The Users Group charge is $20.00 per year (includes a newsletter) and the address is:

B128 Users Group
701 E. North Avenue – Suite C
Lompoc, CA 93436

The initial issue described such items such as the parts and procedures involved to bump the B to 256K. Future topics/projects include CP/M and Intel's 8088 co-processors, various software reviews, etc.

I had one of the first proto–types and feel that for the bank switching capabilities and as an 8 bit machine, it's tough to beat. Most C–64 sound programs will easily adapt to the B as well. James L. White, Rapid City, South Dakota

1541 Alignment Notes: I read your article "Aligning the Commodore 1541 Disk Drive" in issue 06. It's good to make the alignment information available to users, but some of the article is not altogether right. For instance, formatting a blank disk is NOT a good way to check the alignment of the 1541. I can take a 1541 that is far enough out of alignment that it will not read a known good disk and format a blank disk fine (with a steady red light). Also, I am able to read and write to the newly formatted disk with no errors and a steady red light. The reason for this is: I formatted a disk with a 1541 out of alignment and I wrote to the disk with the same unaligned 1541. The disk and 1541 think everything is fine until I try to read a known good disk.

This same reasoning disqualifies using the PERFORMANCE TEST on the TEST DEMO disk as a 1541 alignment check. The PERFORMANCE TEST has you put a scratch disk in the drive and then formats and writes to the disk. The program then reads the same disk with the same unaligned drive. The 1541 sees nothing wrong with the disk it formatted and wrote to. Don't get me wrong, the PERFORMANCE TEST has its place. It is checking all functions of your drive but NOT your alignment.

The short program enclosed is a far better alignment check. It first bumps the head to it's mechanical track 00 stop. This makes sure the 1541's head is where your drive thinks it should be. I say thinks it should be, because if your drive is out of alignment it may not be in the right position. The program then reads (Reads Only) every track 35 to 1 to 34 to 1 to 33 to 1 and so on. Keep in mind to use only your TEST DEMO disk or a known good disk (NOT a disk formatted by your unaligned 1541). Using a disk of unknown credibility or a copy protected disk would be useless. The enclosed program reads only and therefore will not harm the TEST DEMO disk.

If the tested 1541 reads the whole disk with no flickers (I mean not only the slightest flicker) of the red light and makes no tapping noises while coming back to track 1 each time it is reasonably safe to assume the 1541 is in alignment. If the 1541 makes a tap every time it reads track one, it's possible that the track 00 adjustment is out (see drawing). If the red light flickers while reading, the stepper motor

adjustment is probably to blame. You MAY be able to adjust the stepper motor slightly and re-run the enclosed program checking for no flickers or taps.

At the very start of the program you should hear a rapping noise, the same noise you hear at the start of a disk NEW command. This rapping noise should be a sharp and fast rap, like a machine gun. If it sounds dull and uneven it could be the beginning of alignment problems.

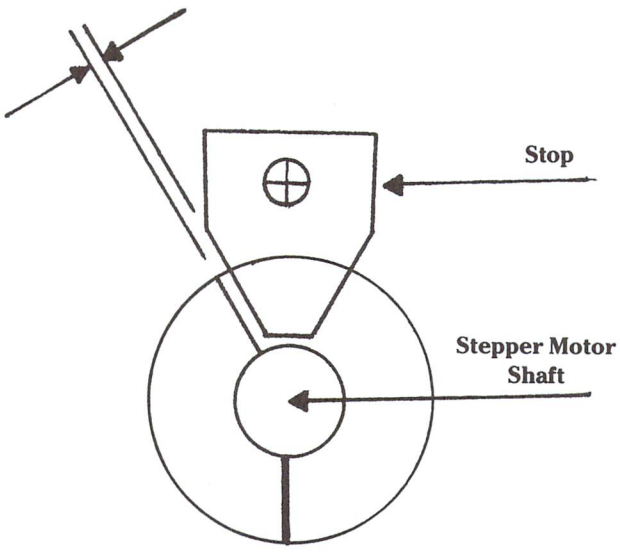
Ed Clutter, Tucson, Arizona

```
100 d=8: rem d = device number
105 open 15,d,15: open 2,d,2,"#"
110 print#15,"m-w" chr$(0) chr$(0) chr$(1) chr$(192)
115 t=35: h$="-"
120 t$=str$(t)
125 print#15,"b-r 2 0 "t$" 9"
130 gosub 160
135 print#15,"b-r 2 0 1 9"
140 t$=str$(1)
145 gosub 160
150 t=t-1: if t>0 then 120
155 close2: close15: end
160 print: print "reading track "h$;t$,
165 input#15,a$,b$,c$,d$
170 print a$;h$;b$;h$;c$;h$;d$
175 if val(a$)<2 then return
180 print "drive has failed alignment check"
185 goto 155
```

Track 00 Adjustment: First step head to track 1 with a program like this:

```
10 open 15,8,15
20 open 2,8,2,"#"
30 print#15,"b-r 2 0 1 0"
```

Next, adjust stop for a clearance of less than 0.25 mm (0.01 inches).



It seems that the alignment article we ran a while back caused quite a stir all over. Some people were pleased, others were not. Thanks to your thoughts on the matter, 1541 users everywhere now have an alternate method to accurately check their 1541's. Thanks for the help. The program works just right.

Brown Bibles For Sale: *In Volume 5, Issue 06 of The Transactor Magazine, on Page 5 ("Bits and Pieces"), you show a one line Decimal to Binary Conversion. I believe I have a better way:*

```
for i=7 to 0 step -1: print sgn( x and 2↑i );: next i
```

I feel this is much less confusing, and obviously requires less statements. It also automatically spaces the bit values, making the binary equivalent easier to read.

Incidently, we still have about 50 copies of Volume 4, Issue 05, "The Reference Issue". If you have readers requesting this issue, please give them our name, address, and telephone number.

Philip Strapp
Computer House, c/o 97 Sandys Street
Chatham, Ontario, N7L 3P5, (519) 354-7882

Thanks for the better binary conversion. Although the Reference Issue has since been superceded by The Complete Commodore Inner Space Anthology, the "Brown Bible" has become somewhat a collectors item - we have absolutely none left.

Piracy vs. The Software Publishers Association: I have just completed reading the open letter to user group presidents, etc. from the executive director of the Software Publishers Association published in Compute! of March, 1985. I am not a president or sysop but this is my letter in response.

I am the newsletter editor and corresponding secretary of our Commodore users group and I was perhaps the most active person in the organization of our group, TRACE, The Richmond Area Commodore Enthusiasts, and one of the most interested and dedicated members.

Piracy is something that I personally am constantly fighting within our group, and I am always crusading against it. I am a firm believer in the free enterprise system and support any efforts to maintain it. The fight against piracy is one of those efforts. The reason we have good software available to us is that those persons with the genius and talent to write software can and will be rewarded for their efforts. If they can do the job, I hope that they make a million! I would like to see all commercial software protected in the manner of Paper Clip from Batteries Included. You need a key to make the program work. I love it!

Persons who make copies for their own personal use or give copies to others are not profiting monetarily from the act of piracy, but are as guilty as if they had. They are stealing potential profits from the rightful recipients. I believe that this group of people is the biggest threat to the industry. They have little respect for the system of free enterprise and if they were the software authors themselves, they would holler the loudest.

One concept blatantly missing from the open letter was reference to the responsibility of publishers in contributing to piracy by encouraging it. We get disks for our group at less than $1.50 each in large quantities so we are aware of the actual cost of the media. The question that comes to mind of many buyers is; Why is the cost of software so high? The cost of copying a disk is slight compared to the price of the finished product. Many can not justify the tremendous mark-up. If prices were more reasonable, piracy would be less tempting.

The thing that is my pet peeve is that once software is no longer available for sale it is legally no longer available at all. This stinks! New owners do not have access to software that is no longer available at retailers. A lot of software that was written for the Vic 20 and Atari computers, for instance, is no longer available legally.

New owners are denied full use of their machines and enjoyment of a lot of terrific software without piracy. Programs published in magazines are not available if the issues no longer exist. This also stinks! I recently bought a PET computer and I would dearly love to be able to get software for it legally. I have been offered copies by owners of copy–righted software which I have reluctantly refused.

Recently I was developing a program for use by members of our group and wanted to incorporate a program that I got from a magazine into my program. In order to remain legal I had to completely re–write the program using only the basic idea behind it. New owners of Commodore computers can not legally obtain this particular program because it is no longer in print. I suppose that I could get permission from somewhere to use it but the time and effort to do so would be prohibitive. It is easier to re–write it.

What I suggest is that when a piece of software is no longer of realistic potential profit to the authors and publishers, that it come under a different status other than 'limbo'. I would love to see some way of finding out what software has been available for my VIC and PET and would be willing to pay whatever cost was necessary to obtain such material. This, in turn, would provide additional profit to the correct persons. Quick Brown Fox is a great word processor. I am using it to write this letter. The company is out of business but the software is still protected by law from being copied. Think about it. The company no longer exists. The author can no longer receive any profits from the sale of the package. Who does the law protect? Nobody! It only prohibits people from benefiting from the particular software in question. There should be an agency to act as a clearing house for such software. I realise that it would require a lot of work but I feel that with each publisher contributing, it could be done to the benefit of everyone concerned. Perhaps efforts could be made to make changes in the copy right laws to cover this problem. If the publishers would aid the buyers in this respect it would be a great improvement.

T.R.A.C.E. (The Richmond Area Commodore Enthusiasts) support all Commodore computers and their friends. We teach introductory BASIC for new computerists, have an imaginative newsletter, and are operating our own exciting bulletin board. Contact us by writing or calling:

Bill Uhler, 2316 LaFayette Ave,
Richmond VA. 23228, (804) 266–0601

Public Right vs. Copyright: I am offended by the affrontery of commercial software vendors who presume to know every detail of what I, as a customer, want a program to do, including even the optimum colour selections for best legibility on my monochrome display, and by their "protection" schemes making it difficult to alter routines I purchase, or to concatenate them for my particular purposes. When I purchase software and find it is undocumented, my personal feeling is that the creator is UNDESERVING OF ANY PROTECTION, COPYRIGHT, OR OTHER. If accompanied by a proper assembly listing, or a least enough information that I can find and modify major routines, it is reasonable to consider that the supplier has met a contractual obligation, and that I should respond in kind. Those who deliberately hide the code of their routines so that I cannot modify them oft–times inspire me to deliberately break their "protection" schemes and disseminate their programmes as widely as I can!

In many respects, I consider the so called "protection" schemes for commercial software in the same vein as the damnable failure on the part of some vendors to provide a schematic with an electronic item, a TV receiver, amateur receiver, microwave oven, computer, or the like. Commercially produced software, like commercially made hardware, must be a compromise to attempt to address the largest market. I have no quarrel with that. But, why in Hell should I not be able to modify the product I buy to best suit MY needs or desires, and be provided with reasonably–expected data with which so to do. The "protectionist", I believe, deserve to be done in – horribly!!!!! This is, by the way, one of my PET peeves (pun intended) with Commodore – they even deleted a part of their own CP/M book after preparation, deleting not only the schematics, but critically needed code as well. BAH

I suppose that this should be expected, though, from a company that publishes such lies as claiming an RS232 port which isn't, ASCII that fails to meet the American Standard Code for Information Interchange, etc.

B. Chandler Shaw, Granada Hills, California
Senior Staff Engineer with Allied Bendix Aerospace

I seem to recall the possibly of a Canadian law that states "a manufacturer is obligated to supply the schematics of an electronic item on request". Perhaps a similar law could be effective for software provided they don't legislate it prior to laws that safely protect manufacturers from being clobbered by those with less than honourable reasons for obtaining documents like commented source code, etc. Feeling 'safe' is a nice feeling no matter what business you're in. Once programmers feel safe about releasing tender information, you'll no doubt see a lot less reluctance.

The Copyright And You: Is there any possible way in which a group such as ours (Fellowship Baptist Church, Racine, Wisconsin) can share your programs legally? For instance, I own The Transactor Volume 5 Issue 04, which contains "Autoswap". It is quite possible that another member of our group would be interested in this program, and yet as we understand the law, there is no legal way for me to share it with him. I realize that it would be possible to apply to you for specific permission to use the program, or even to purchase a back issue of the magazine if it is available. Both of these options seem quite cumbersome and prohibitive, however. Is there an alternative?

Irwin J. Cobb, Union Grove, Wisconsin

There is an alternative. Our standard policy on a program printed in our magazine that we control copyright on is that the programs are free to copy (give away), but not to sell (see page 2). If you want to give a copy of our programs to every friend you have, we won't stop you. If you want to sell a copy to someone, we ask that you don't. Besides, it's sleazy. The respect you feel for us will ultimately determine the future of this policy. More importantly it will mean that we continue to bring you programs worthy of disseminating.

Although we do sell The Transactor Disk at a slight profit, it's only enough to keep the service alive. There is no protection on Transactor Disks and the same policy as above still applies. Giving away diskettes can become a noticeable expense, but that's what we expect from you in return for our permissiveness. Even justifying the cost of giving the programs away constitutes selling the programs and it's only natural that we'll be unhappy. What's worse is if we find ourselves competing against parasites, our enthusiasm will change to discouragement, and the service will be discontinued.

How To Get The Most Out of CompuServe

A review by Robert Adler, Montreal, Quebec

Authors : Charles Bowen
Publisher : Bantam Books, Inc.
666 Fifth Avenue
New York, NY 10103
1984, 278 pages
Price : $12.95 US ($14.95 Canadian)
Audience : Modem users

Question: How do you get the Most Out of CompuServe?
Answer: Buy the book entitled; How to Get the Most Out of CompuServe.

I have used CompuServe on many occasions. When I first bought my modem, I used up my free hour of connect time in what seemed to be 5 minutes! Then, anxious to see what this vast information center actually contains, I signed up with my VISA card. One day, I spent hours looking for a certain program I had read was available in the public domain databases of CompuServe. When I finally found it, I spent hours trying to download it. This cost me $20 an hour! For this reason, I retired early as a CompuServe subscriber.

Many months later, a friend came over to my house and demanded that I show him CompuServe so that he could leave his friend, who lives in the United States, a message using electronic mail. We went through many menus. We found the electronic mail section. We could not understand what the menu options were all about. The help files were a waste of time. I remembered that the best way to get an answer to a problem on CompuServe is to use the feedback option. Using this option, you can enter a question to the employees at CompuServe and they usually respond, in plain English, within two days.

Two days later, I logged onto the service and was told that I had mail waiting. I read the mail with delight and had all my questions briefly answered. One of my questions was about getting some kind of documentation on how to use the various features of CompuServe. It was suggested that I buy a book entitled How to Get the Most Out of CompuServe. Using the CompuServe employee's directions, I quickly found the Electronic Mail (tm) where I gladly purchased the book.

It arrived a few weeks later, very well packaged. I digested all the information in the book within a few days. I was sorry that I had never bought this book before.

How to Get the Most Out of CompuServe serves as a tutorial manual and a reference guide. The authors, Charles Bowen and David Peyton, take readers by the hand and guide them through the various services of CompuServe on four on-line tours. In the first tour of CompuServe, the reader learns how to log on, use the menus, and play some on-line games. The reader also learns how to search for a particular service and a few very important commands.

The authors do not waste much time while the reader is on-line. They wait until the reader has logged off for elaboration on certain topics.

In the second on-line tour, the reader is introduced to on-line electronic conferencing. This is a feature whereby any number of computer users can type messages to each other as if they were on an enormous party line. The authors talk about the procedures, the ethics, the commands, and the different sections of the conferencing area.

In the third on-line tour, the reader is shown the National Bulletin Board. The reader is shown how to read messages selectively, as well as how to leave messages to others who use the service. In this tour, the reader also learns how to use the CompuServe line editor. The reader is also taken into his or her personal file area, a private 128k storage section which is allocated to every CompuServe subscriber The reader is told how to manage his or her personal disk space. In this on-line tutorial, the reader is also taken through public-access, a database of public domain programs, and shown how to download a program. Many commands are learned in this on-line tutorial as is the case with all the others.

In addition to explaining the many commands and features found on CompuServe, the authors discuss shortcuts, and ways to customize CompuServe to each user's likings. Readers are introduced to one of the many Special Interest Groups (SIGs) in CompuServe and shown many ways of finding or reading the messages contained in the SIG's database(s).

What makes CompuServe so useful is not the ability to leave messages and read messages but rather services such as shop at home, or bank at home. The reader is shown such things as how to read stock quotes, how to order a briefcase, and a demonstration of home banking.

Approaching the end of the book, the reader is given a short description of many of the on-line games. This chapter should serve as a reference to those looking for a fun game to play alone or with others while on CompuServe.

The end of the book has a large appendix called the On-line Survival Kit. It serves as a quick reference to those having trouble logging on, or remembering commands. It also answers many commonly asked questions from CompuServe subscribers. Among the many other things compended into this section of the book is a way to contact the authors on the service itself.

The main objective of the authors of this book are to teach the reader how to use and get the most out of CompuServe. The authors do not attempt to take the reader into every nook and cranny of CompuServe, but rather to train the reader how to find and efficiently utilize the services offered on CompuServe. Because the book was written in late 1983, some menu options may have changed slightly. The authors have noted may points where CompuServe may have changed a particular menu or added a new service. The reader is taught to a level where he or she would not have to be concerned about minor changes in the service.

The price of $12.95 (U.S.) is a bargain considering the time that will be saved while on CompuServe. Each of the on-line tutorials should take up to no more than one hour's time. I, personally, cheated and saved four hours of connect time by using the simulated screen displays in the book, since I was already familiar with a few of CompuServe's services.

The book's fine writing, short chapters, and many simulated screen displays make the book a pleasure to read. Using the detailed table of contents, the seven page index, the "on-line survival kit", and an easy way to contact the authors does not allow for questions to be left unanswered. If you have a modem, and have never tried CompuServe, or have tried CompuServe but gave it up because of its complex structure, buy this book!

TransBASIC Installment #4

Nick Sullivan
Scarborough, Ont.

TransBASIC Parts 1 to 3 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 burried inside the Commodore 64. Part 3 explains how to use these routines when creating new modules.

To take advantage of the TransBASIC command system, one must first obtain a copy of the TransBASIC Kernel. The Kernel is only about 500 bytes long, but the source listing of the Kernel is quite long and can't be printed each time. Volume 5, Issue 05 (Hardware & Peripherals) contains the printed listing, however The Transactor Disk for every issue will include this file, plus files from the current and all previous TransBASIC articles.

The next issue, our Languages issue, will have a bumper TransBASIC Column. One of Nick's original goals for TransBASIC was to generate user response. Well the response has been fabulous! Letters have come in from around the world with new TransBASIC commands for the library and next issue seems like the best possible place for them. Keep them coming too - it won't be long and there will be as many TransBASIC commands as Commodore BASIC commands. Also, next issue we'll give a brief re-hash of the procedure for building a typical TransBASIC from the commands published so far. – M.Ed

Using Numeric Expressions

The most powerful set of routines in Commodore BASIC is the one responsible for the evaluation of numeric expressions. Are you a machine language programmer? Try sitting down for an hour or two and whip up a little code to evaluate an expression such as this (don't forget to include complete error–checking):

```
peek(u + v/2) + sqr(abs(fnq(sin(c)))/a(u))–len("YIPE!")
```

You will find that this is not an easy task, yet there are many programming situations in which one would like to have an evaluation routine that would allow such complex expressions. It is nice, in those situations, to make use of the following instruction, which does all the work for you:

```
JSR $AD8A
```

The routine entered at $AD8A in the Commodore 64 BASIC ROM does several things. It reads a BASIC expression from the input stream (whether in program or direct mode), checks that it is of numeric type (or a SYNTAX ERROR ensues), and evaluates it. The result is left in the area of zero page called Floating Point Accumulator #1 (FPA #1), where it is accessible to other routines. Arithmetic operators, function calls, defined functions and nesting are automatically dealt with along the way.

Evaluating the expression is only a first step. If you have floating point arithmetic to do, now is the time, since numeric functions and operations act upon the value in FPA #1. If you want to reduce the floating point value to a 16–bit or 8–bit integer, that is also possible by calling the appropriate routine

(for an extensive list see TransBASIC Installment #3 in the last issue).

Suppose, for instance, that we decided to write a TransBASIC statement called HEX? (pronounced hex-print) to print the hexadecimal equivalent of a 16-bit integer in decimal form. If we typed 'HEX? 49152', the computer would respond '$C000'; if we typed 'HEX? ABS(U*10)', the expression 'ABS(U*10)' would be evaluated, and the corresponding hex value given. How would we go about writing such a statement?

The first step would be to put the HEX? keyword on an available line in the TransBASIC statement keyword list (for a discussion of available line ranges in source code see TransBASIC Installment #2). Let's use line 500. We have to remember that the last character of the keyword has to be shifted (high bit set). The last character of HEX? is the question-mark with an ASCII value of $3F. Setting the high bit gives a value of $BF, thus:

```
500 .ASC "HEX": .BYTE $BF
```

The address of the execution routine should be on a line whose number is greater by one thousand than the keyword line. We'll call the execution routine 'HEX'. Remember that we have to subtract one from the address owing to the way that TransBASIC routines are invoked:

```
1500 .WORD HEX-1
```

All that remains is to write the execution routine itself. When we enter the routine, the CHRGET pointer will be pointing to the first non-blank character following the keyword - the first byte of our numeric expression. To evaluate that expression, and store the result in FPA #1, we call BASIC's expression evaluator as described above:

```
50000 HEX JSR $AD8A
```

Following this call the CHRGET pointer is directed to the first character following the evaluated expression. This will presumably be a statement terminator — either a colon or a zero-byte — since no further parameters are expected.

Now we want to turn the floating point value into a 16-bit integer. Looking through the list of ROM routines in the last issue we find the following entry:

```
47095 $B7F7
```

Convert the number in FPA #1 to an unsigned integer in .Y (low byte) and .A (high byte), and in $14/15.

This appears to be exactly what we need. Let's do it.

```
50002 JSR $B7F7
```

Now all we need is a stock routine to convert a 16-bit number into hex form. We begin by outputting the dollar sign:

```
50004       LDA  #"$"
50006       JSR  $FFD2
50008       LDA  $15
50010       JSR  HE1
50012       LDA  $14
50014  HE1  PHA
50016       LSR
50018       LSR
50020       LSR
50022       LSR
50024       JSR  HE2
50026       PLA
50028       AND  #$0F
50030  HE2   CMP  #$0A
50032       BCC  HE3
50034       ADC  #6
50036  HE3   ADC  #$30
50038       JMP  $FFD2
50040  ;
```

That's all there is to it. The HEX? statement can be written up as a module and added to your TransBASIC library, if you like, although my preference would be to rewrite it as part of a module of numeric conversion functions like those provided in BASIC 3.5 (Plus/4 and C-16). The best part of it is that the full power of the BASIC expression evaluator will be effortlessly applied to the expression part of the command.

Postscripts

As was pointed out in Transbloopers in the last issue, the listings for the CURSOR POSITION module were not given (as promised) in Installment #2, but did make it into Installment #3. Also, some readers have written to point out that the shifted bracket character ($A8) was omitted from the functions CHECK and AWAIT in the 'CHECK & AWAIT' module in Installment #2. If you haven't caught it yet, change line 602 to read:

```
602 .asc "check": .byte $a8: .asc "await": .byte $A8
```

One final error occurred not in the magazine but on at least some copies of the disk, in the short boot program that loads "TB/ADD.OBJ" and "TB/KERNEL". The error is that "TB/ADD.OBJ" is referred to as "TB/ADD.M", leading to a 'FILE NOT FOUND' error when the program is run.

Lastly, I want to thank those readers who have sent me their letters with comments on and ideas for TransBASIC. Several modules contributed by readers are now being edited into the TransBASIC library and will appear in future columns. More contributions are always welcome — please don't hesitate to write.

New Commands

This part of the TransBASIC column is devoted to describing the new commands that will be added each issue. The descriptions follow a standard format.

The first line gives the commands keyword, the type (statement or function), and a three digit serial number.

The second line gives the line range allotted to the execution routine for the command.

The third line gives the module in which the command is included.

The fourth line (and the following lines, if necessary) demonstrates the command syntax.

The remaining lines describe the command.

STRIP$((Type: Function Cat #: 045)
Line Range: 3984–4090
Module: STRIP & CLEAN
Example:

```
A$=STRIP$("TWO BURGHERS, THREE SHEIKS")
```

The argument string is returned stripped of everything except alphanumerics; all alphabetic characters are converted to lower case: "twoburghersthreesheiks".

CLEAN$((Type: Function Cat #: 046)
Line Range: 3988–4090
Module: STRIP & CLEAN
Example:

```
Q$=CLEAN$("TRANSIENT'S      DENTAL MEDITATION?!")
```

The argument string is returned stripped of everything except alphanumerics and blanks. Alphabetic characters are converted to lower case, and multiple blanks are reduced to single blanks: "transients dental meditation".

USCROL (Type: Statement Cat #: 067)
Line Range: 5260–5644
In Module: SCROLLS
Example: USCROL 12,18,6,30
Example: USCROL/0,24,0,39
A rectangular area on the screen, defined by four numerical parameters, is scrolled upwards one character row. If a slash character is present, as in the second example, the top row of the scroll area is moved into the space occupied by the bottom row before the scroll; that is, the scroll includes 'wraparound'. If the slash character is absent, the top row vanishes and the bottom row is filled with blanks. The significance of the numerical parameters is explained under DEFWDW below. Both colour memory and character memory (the video matrix) are scrolled.

DSCROL (Type: Statement Cat #: 068)
Line Range: 5264–5644
In Module: SCROLLS
Example: DSCROL 3,17,11;39
A rectangular area is scrolled downwards on the screen one character row. For the significance of the the numerical parameters and of the optional slash character, consult 067/USCROL.

LSCROL (Type: Statement Cat #: 069)
Line Range: 5268–5644
In Module: SCROLLS
Example: LSCROL/0;1,0;20
A rectangular area is scrolled leftwards on the screen one character row. For the significance of the the numerical parameters and of the optional slash character, consult 067/USCROL.

RSCROL (Type: Statement Cat #: 070)
Line Range: 5272–5644
In Module: SCROLLS
Example: RSCROL/0;11,5,25
A rectangular area is scrolled rightwards on the screen one character row. For the significance of the numerical parameters and of the optional slash character, consult 067/USCROL.

DEFWDW is a routine used by the scroll commands; it is also available for use by other commands if required. Four parameters in the form of numerical expressions are fetched from the input stream to define a rectangular area on the screen. The first and third parameters specify the row and column of the top left corner of the rectangle, numbering from 0 to 24 and 0 to 39 respectively. If the first parameter is followed by a comma, then the second parameter is taken to be the bottom row of the rectangle. If the third parameter is followed by a comma, then the fourth parameter is taken to be the rightmost column of the rectangle. If the first parameter is followed by a semicolon instead of a comma, the second parameter is taken to be the depth of the area to be scrolled, numbering from 1 to 25. If the third parameter is followed by a semicolon instead of a comma, the fourth parameter is taken to be the width of the area to be scrolled, numbering from 1 to 40. The memory address of the top left corner of the rectangle is contained in a word labelled CORNER. The corresponding address in colour memory is in a word labelled COLCOR. The depth and width of the area, now counting from 0 to 24 and from 0 to 39, are returned in the byte-size locations DEPTH and WIDTH respectively.

Program 1

```
LE 0 rem strip & clean (aug 29/84)    :
FH 1 :
DH 2 rem 0 statements, 2 functions
HH 3 :
JE 4 rem keyword characters: 14
JH 5 :
NJ 6 rem keyword     routine    line     ser #
LI 7 rem f/strip$(   strip      3984     045
EL 8 rem f/clean$(   clean      3988     046
NH 9 :
MH 10 rem u/cifchr (2560/003)
PM 11 rem u/cifnum (4092/047)
AI 12 :
LD 13 rem ==============================
CI 14 :
OA 609 .asc "strip$": .byte $a8: .asc "clean$": .byte $a8
CJ 1609 .word strip-1,clean-1
IH 2560 cifchr  cmp #$5b      ;return carry set
HA 2562         bcc cic1      ;if accumulator
EM 2564         clc           ;contains
JP 2566         bcc cic2      ;alphabetic
OB 2568 cic1    cmp #$41
FJ 2570 cic2    rts
CI 2572 ;
OB 3984 strip   lda #$80      ;strip - bit 7 set
GM 3986 .byte $2c             ;'bit'
IL 3988 clean   lda #0        ;clean - bit 7 clr
GE 3990         pha           ;save flag
HP 3992         jsr $aef4     ;eval string expr
DG 3994         jsr $b6a3     ;get str len, addr
KG 3996         sta t3        ;save length
EH 3998         pla           ;get flag
OI 4000         sta t4        ;store it
DE 4002         txa           ;push address
AC 4004         pha
NF 4006         tya
EC 4008         pha
OG 4010         lda t3        ;get length
DN 4012         jsr $b47d     ;adjust b-o-s ptr
LL 4014         pla           ;pointer to input
BN 4016         sta $23       ; string at $22/23
KD 4018         pla
IH 4020         sta $22
HA 4022         stx $24       ;pointer to output
JE 4024         sty $25       ; string at $24/25
KK 4026         ldx #0        ;init len counter
JC 4028         stx t5        ; - result string
LN 4030         ldy #$ff
FM 4032 cln1    iny           ;check if done
GF 4034         cpy t3
AP 4036         beq cln4
NO 4038         lda ($22),y   ;get input char
LC 4040         and #$7f      ;..unshift it
PP 4042         jsr cifchr    ;..branch if it's
GO 4044         bcs cln2      ; alphabetic or
PG 4046         jsr cifnum    ; numeric
GP 4048         bcs cln2
AO 4050         bit t4        ;test flag
AO 4052         bmi cln1      ;strip - skip char
EO 4054         bvs cln1      ;skip mult spaces
GE 4056         cmp #$20      ;skip char if
OC 4058         bne cln1      ; not a space
MN 4060         clc           ;space - carry clr
BN 4062 cln2    sta ($24,x)   ;char to outstring
GP 4064         lda t4        ;get flag
FD 4066         and #$80      ;keep high bit
FP 4068         bcs cln3      ;set bit 6 if char
KE 4070         ora #$40      ; is a space
PB 4072 cln3    sta t4
CC 4074         inc t5        ;bump outstr len
CF 4076         inc $24       ;bump pointer to
FL 4078         bne cln1      ; output string
PE 4080         inc $25       ; then go for a
NO 4082         bne cln1      ; new character
HK 4084 cln4    lda t5        ;get outstr len
PL 4086         sta $61
ON 4088         jmp $b4ca     ;set up descriptor
AH 4090 ;
BJ 4092 cifnum  cmp #":"      ;return carry set
EC 4094         bcc cin1      ;if accumulator
CD 4096         clc           ;contains numeric
CO 4098         bcc cin2
KP 4100 cin1    cmp #"0"
NL 4102 cin2    rts
OH 4104 ;
```

Program 2

```
OL 0 rem scrolls (mar 30/85)    :
FH 1 :
JH 2 rem 4 statements, 0 functions
HH 3 :
KE 4 rem keyword characters: 24
JH 5 :
NJ 6 rem keyword     routine    line     ser #
CF 7 rem s/uscrol    uscro      5260     067
NB 8 rem s/dscrol    dscro      5264     068
ME 9 rem s/lscrol    lscro      5268     069
EE 10 rem s/rscrol   rscro      5272     070
PH 11 :
ML 12 rem u/by40  (5646/071)
OP 13 rem u/defwdw (5668/072)
CI 14 :
ND 15 rem ==============================
EI 16 :
NN 118 .asc "uscroLdscroL"
PP 119 .asc "lscroLrscroL"
FJ 1118 .word uscro-1,dscro-1
AK 1119 .word lscro-1,rscro-1
JC 5260 uscro   ldx #$40
CM 5262 .byte $2c             ;'bit'
HE 5264 dscro   ldx #$c0
NG 5266 .byte $2c
PA 5268 lscro   ldx #$00
BH 5270 .byte $2c
PD 5272 rscro   ldx #$80
AB 5274 ;
FF 5276         stx scrdir    ;save direction
HB 5278         ldx #0        ;flag - no wrap
NG 5280         cmp #$ad      ;slash / means wrap
AG 5282         bne scro1
OO 5284         jsr $73       ;move past slash
HL 5286         ldx #$80      ;flag - wrap
KL 5288 scro1   txa           ;save wrap flag
GC 5290         pha
NK 5292         jsr defwdw    ;fetch window params
GD 5294         pla
OA 5296         sta t2
OK 5298         lda corner    ;screen address of
BE 5300         ldx corner+1  ;top left corner of
```

```
IC 5302          ldy  colcor+1  ;scroll area to t3/t4
IB 5304          sta  t3
EF 5306          sta  t5        ;colour memory
NJ 5308          stx  t4        ;address to t5/t6
EI 5310          sty  t6
LD 5312          bit  scrdir
BI 5314          bvc  scro2     ;horizontal scroll
KH 5316          jmp  scro14    ;vertical scroll
PJ 5318 scro2    ldx  #0        ;init buffer ptr
FG 5320          ldy  scrdir    ;init column ptr
PN 5322          beq  scro3     ;  for scroll dir
BO 5324          ldy  width
CB 5326 scro3    lda  (t3),y    ;put rt col char
BP 5328          sta  buffer,x  ;  in buffer
NA 5330          lda  (t5),y    ;put rt col colr
EO 5332          sta  colbuf,x  ;  in buffer
NI 5334          inx            ;bump buffer ptr
LH 5336          lda  t3        ;set screen and
AI 5338          clc            ;  colour pointers
FG 5340          adc  #$28      ;  to next row
OD 5342          sta  t3
EE 5344          sta  t5
KH 5346          bcc  scro4
IC 5348          inc  t4
OC 5350          inc  t6
JF 5352 scro4    dec  depth     ;count down depth
KD 5354          bpl  scro3     ;save whole column
PK 5356 scro5    sec            ;set screen and
KI 5358          lda  t3        ;  colour pointers
AJ 5360          sbc  #$28      ;  to previous row
CF 5362          sta  t3
IF 5364          sta  t5
AN 5366          bcs  scro6
HB 5368          dec  t4
NB 5370          dec  t6
IN 5372 scro6    dex            ;dec buffer ptr
EK 5374          bmi  scro13    ;to count columns
NA 5376          ldy  scrdir    ;scroll right if
OO 5378          bne  scro8     ;scrdir non-zero
GJ 5380 scro7    cpy  width     ;skip if whole
FA 5382          beq  scro10    ;  row scrolled
BF 5384          iny            ;get screen byte
OF 5386          lda  (t3),y    ;get screen byte
II 5388          pha
OJ 5390          lda  (t5),y    ;get colour byte
NP 5392          dey            ;move left
BN 5394          sta  (t5),y    ;put colour byte
MJ 5396          pla
BI 5398          sta  (t3),y    ;put screen byte
IP 5400          iny
LC 5402          bne  scro7     ;do another
PP 5404 scro8    ldy  width     ;skip if width 0
BF 5406          beq  scro10
BN 5408 scro9    dey
GH 5410          lda  (t3),y    ;get screen byte
AK 5412          pha
GL 5414          lda  (t5),y    ;get colour byte
CP 5416          iny            ;move right
JO 5418          sta  (t5),y    ;put colour byte
EL 5420          pla
JJ 5422          sta  (t3),y    ;put screen byte
LO 5424          dey
FE 5426          bne  scro9     ;do another
AO 5428 scro10   lda  buffer,x  ;recover right
IN 5430          pha            ;  col screen,
DE 5432          lda  colbuf,x  ;  colr, data
EG 5434          bit  t2        ;test wrap flag
KK 5436          bmi  scro11    ;branch to wrap
FF 5438          pla            ;clear stack
JJ 5440          lda  #$20      ;load space
EG 5442          bne  scro12
CF 5444 scro11   sta  (t5),y    ;put colr data
OM 5446          pla
BF 5448 scro12   sta  (t3),y    ;put scrn data
AM 5450          clc
HH 5452          bcc  scro5     ;handle prev row
BJ 5454 scro13   rts
JB 5456 scro14   lda  scrdir    ;get direction
BE 5458          and  #$80      ;test hi bit
DC 5460          tay
HG 5462          beq  scro15    ;upward scroll
CH 5464          ldy  depth     ;calculate
OF 5466 scro15   jsr  by40      ;  offset
MM 5468          clc            ;convert to bottom
BD 5470          adc  corner    ;  left corner address
AM 5472          sta  t3
GM 5474          sta  t5
PH 5476          php            ;save carry state
KH 5478          txa            ;get offset high byte
MJ 5480          adc  corner+1
MM 5482          sta  t4
FE 5484          plp            ;recall carry state
CI 5486          txa            ;get offset high byte
CD 5488          adc  colcor+1
IN 5490          sta  t6
KN 5492          ldy  width     ;copy row to buffer
EM 5494 scro16   lda  (t3),y    ;  for later wrap
AM 5496          sta  buffer,y  ;  if needed
DN 5498          lda  (t5),y
IL 5500          sta  colbuf,y
JD 5502          dey
MM 5504          bpl  scro16
MJ 5506          ldx  depth
DK 5508 scro17   dex            ;skip when done
EL 5510          bmi  scro21
KK 5512          lda  t3
EC 5514          bit  scrdir    ;get direction
NO 5516          bpl  scro18    ;branch to scroll up
CC 5518          sec            ;calculate addresses
EB 5520          sbc  #$28      ;  for row above -
DI 5522          sta  $22       ;  screen in $22/23
LM 5524          sta  $24       ;  colour in $24/25
KL 5526          lda  t4
PF 5528          sta  $23
CM 5530          lda  t6
JG 5532          sta  $25
FO 5534          bcs  scro19
EC 5536          dec  $23
MC 5538          dec  $25
EN 5540          bne  scro19
NE 5542 scro18   clc            ;calculate addresses
JC 5544          adc  #$28      ;  for row below
OG 5546          sta  $22
GH 5548          sta  $24
CN 5550          lda  t4
HH 5552          sta  $23
KN 5554          lda  t6
BI 5556          sta  $25
NL 5558          bcc  scro19
BG 5560          inc  $23
JG 5562          inc  $25
OE 5564 scro19   ldy  width     ;no scroll if
```

```
GA 5566          beq  scro22     ;  width zero
BF 5568 scro20   lda  ($22),y    ;scroll
CJ 5570          sta  (t3),y     ;  current row
OI 5572          lda  ($24),y
NF 5574          sta  (t5),y
DI 5576          dey
LA 5578          bpl  scro20
DI 5580          lda  $22        ;target row
OL 5582          sta  t3         ;  becomes source row
ED 5584          sta  t5
LF 5586          lda  $23
GD 5588          sta  t4
FG 5590          lda  $25
OD 5592          sta  t6
GA 5594          bne  scro17
BA 5596 scro21   ldy  width      ;fetch data from
HK 5598 scro22   lda  buffer,y   ;  wrap buffer
DH 5600          pha             ;set aside
IJ 5602          lda  colbuf,y   ;colour data too
FD 5604          bit  t2         ;test for wrap
DO 5606          bmi  scro23     ;wrap required
IA 5608          pla             ;otherwise leave
GN 5610          lda  #$20       ;  colour same
LN 5612          sta  (t3),y     ;  and put space
OM 5614          bne  scro24     ;  on screen
OG 5616 scro23   sta  (t5),y     ;put wrap colour
KH 5618          pla
PG 5620          sta  (t3),y     ;put wrap char
HL 5622 scro24   dey             ;handle next col
ND 5624          bpl  scro22
GO 5626          rts
CH 5628 ;
EK 5630 buffer   =    *          ;area to save screen row
GO 5632     *    =    *+$28      ;or col to wrap around
IH 5634 ;
AO 5636 colbuf   =    *          ;area to save colour row
MO 5638     *    =    *+$28      ;or col to wrap around
OH 5640 ;
CO 5642 scrdir   .byte 0         ;scroll direction
CI 5644 ;
LL 5646 by40     lda  #0         ;16-bit accumulator
JJ 5648          tax             ;  set to zero
FD 5650 byf1     cpy  #0         ;exit when done
JD 5652          beq  byf2
EM 5654          dey             ;count down
BO 5656          adc  #$27       ;add 40 (carry is set)
NP 5658          bcc  byf1
IP 5660          inx
KC 5662          bne  byf1
FO 5664 byf2     rts
IJ 5666 ;
EM 5668 defwdw   jsr  $b79e      ;get corner row
LB 5670          stx  corner
KD 5672          jsr  $79
OB 5674          cmp  #";"       ;check separator
LE 5676          bne  dfw2
KO 5678          jsr  $b79b      ;get window depth
PI 5680          dex             ;adjust
OD 5682          bmi  dfw3       ;must be some depth
DH 5684 dfw1     stx  depth
OE 5686          lda  corner     ;get corner row
OK 5688          clc
KO 5690          adc  depth
IE 5692          bcs  dfw3       ;8-bit overflow
DA 5694          cmp  #$19       ;check depth
IF 5696          bcs  dfw3       ;deeper than screen
AE 5698          bcc  dfw4
FJ 5700 dfw2     jsr  $aefd      ;must be comma
OP 5702          jsr  $b79e      ;get bottom row
MP 5704          txa
LM 5706          sec
FL 5708          sbc  corner     ;check >= top row
JB 5710          tax
CI 5712          bcs  dfw1
GF 5714 dfw3     jsr  $b248
HB 5716 dfw4     jsr  $aefd      ;skip comma
LG 5718          jsr  $b79e      ;get corner column
ED 5720          stx  corner+1
MG 5722          jsr  $79
AF 5724          cmp  #";"       ;check separator
NI 5726          bne  dfw6
OB 5728          jsr  $b79b      ;get window width
BM 5730          dex             ;adjust
IK 5732          bmi  dfw3       ;must be some width
AK 5734 dfw5     stx  width
PD 5736          lda  corner+1   ;get corner col
AO 5738          clc
HG 5740          adc  width      ;check width
KH 5742          bcs  dfw3       ;8-bit overflow
OP 5744          cmp  #$28
KE 5746          bcs  dfw3       ;wider than screen
OH 5748          bcc  dfw7
LM 5750 dfw6     jsr  $aefd      ;must be comma
FG 5752          jsr  $b79e      ;get right column
OC 5754          txa
NP 5756          sec
NB 5758          sbc  corner+1   ;check >= left col
LE 5760          tax
EM 5762          bcs  dfw5
OH 5764          bcc  dfw3
LO 5766 dfw7     ldy  corner     ;get corner row
KA 5768          jsr  by40       ;calc scrn offset
LJ 5770          clc             ;add column
AM 5772          adc  corner+1
MJ 5774          bcc  dfw8
MG 5776          inx
BK 5778 dfw8     sta  corner     ;sav offset lo byt
KB 5780          sta  colcor     ;  scrn and colr
KE 5782          txa
DD 5784          pha             ;hold high byte
AB 5786          clc
MC 5788          adc  #$d8       ;add colr ram base
EI 5790          sta  colcor+1   ;save colour addr
HJ 5792          pla             ;get offset hi byte
IB 5794          clc
BA 5796          adc  648        ;add screen base
MN 5798          sta  corner+1   ;save screen addr
EJ 5800          rts
AC 5802 ;
GP 5804 depth    .byte 0         ;registers to
MO 5806 width    .byte 0         ;  return window
IJ 5808 corner   .word 0         ;  parameters to
OF 5810 colcor   .word 0         ;  calling routine
KC 5812 ;
```

Telecomputing:
From Concept to Connect.

This article is actually four articles combined into one. Some facts are repeated from one to the next, but by the time you get to the end you will have assimilated the facts enough times that when you hear them again, you will say to yourself "I knew that", which is the nicest part about learning.

The first part, by Jeff Goebel, conceptualizes telecomputing and gives you and idea of what to expect as you progress toward your individual interests.

In the second part, Geoffrey Welsh describes the BBS in a little more detail, particularly the "Punter" type BBS written by Steve Punter, author of the WordPro series of wordprocessor programs and a well-known pioneer of the BBS for Commodore users.

Part three offers still more terminology with tips for spotting communications problems if and/or when you encounter them. It seems our Robert Dray has experienced all the most common misfortunes, and his contribution should indeed save even more serious head-scratching.

Mario Marrello winds it up with part four and a study of the RS232C standard of communicating serial data.

An Introduction to MODEMS

Jeff Goebel, Georgetown, Ontario

In this article, I look into some of the myths of owning and using a modem. If you have just bought a modem, and don't know what it does, or if you are still toying with the idea, then read on.

First of all; what is a modem? Simply put, a modem is an interface. You have an interface to your disk drive and you may have an interface to your printer. A modem is an interface to another computer. It allows you to connect your computer up to someone elses computer and transfer information between the two. The main difference between a modem, and some other interface, is that instead of a simple cable; with a modem, the telephone lines are used.

With a modem, you will discover a whole new use for your computer. It is unlike any other accessory you can buy. The modem is FUN, HELPFUL and useful. Can you say your printer is FUN, or your joystick is HELPFUL? Using a modem allows you to use other people's computers from your own home, and to benefit from their knowledge. It also allows many people to operate their business more effectively by opening up the possibility to work from their own den, or on location.

As stated above, the modem is "just another interface". It's actually very similar to operating your disk drive. You open up a channel to your modem and input data and output data just like you would to a sequential file on the disk drive or cassette. The main thing to remember is that you'll be interfacing two computers together, not a computer to an accessory. What I mean by this is, the computer at the OTHER END of your connection must WANT to communicate, and it must be set up so that telecommunication is possible. Don't believe all these TV shows you see where kids with modems can call up all the computers around the world and print out messages on their screens. This is fiction. In order to establish any connection at all, the computer on the other end of the line (referred to as the HOST COMPUTER) must be expecting your call. If you bought your modem so you could link up with your office computer and do your work at home, I hope you first checked that your office has already set up a system that allows that type of transacting.

So what kind of things can I do with my modem? In nearly every city there is at least one "BULLETIN BOARD SYSTEM" (often abbrieviated to BBS) which is either free to use or asks a small yearly membership fee. These systems are simply computers that have been set up to answer the phone and provide a message service. In the Toronto local dialing area, for example, there are over 40 numbers, and more and more people are opening up their own BBS systems all the time. You dial up

these computers and they allow you to read and send messages or private letters.

If you need to know something but you don't know who to ask, you can leave the request on a public BBS and call back later. . . you'll probably be flooded by several messages from other modem owners who know the answer. An example: You are playing an adventure game and you've come to a standstill. You know that you've got to get a box open, but you've tried everything and nothing seems to work. All you have to do is pick up your phone; dial a BBS and leave your name and question; "How do I get the blue box open?". Later that same day you call back and see if anyone has left you the answer. At the same time, you may see some questions you know the answer to. You may even get involved in a detailed political debate with some of the other users.

Most BBS systems offer other features as well as message reading and posting. Some systems in Toronto even have the provision to actually LOAD and SAVE programs to and from your disk drive over the phone. Most offer a variety of Bulletins. Everything from IN STORE SPECIALS to movie and restaurant reviews. There are BBS's that specialize in GAMES, MEDICINE, HUMOUR, COMPUTERS, HAM RADIOS and just about any other specific interests.

Apart from this type of computer service, there are also some BIG systems which charge you for the time you spend on them. Systems like these do MUCH MUCH more. COMPUSERVE, one of the most popular, has literally HUNDREDS of things to do on line. Larger systems like this one offer the unique opportunity to actually CHAT with other users because COMPUSERVE allows more than one user to access at once. You can be ON LINE with thousands of other users all across Canada and the United states. There's even a C.B. simulation that allows you to "talk" to people as far away as Hawaii or Alaska. Systems like these run on huge mainframe computers.

These are just some of the things you can do with your modem TONIGHT! Modems have other uses too. If you want more ideas on what to use your modem for, ask the people on the BBS systems what they use their modems for. I'm sure you'll get lots of answers back. Many owners have really specific uses for their modems, others use them JUST for the BBS systems. In Toronto, most of the universities allow their computer science students to access the computers from home using modems.

Now you'll have to decide whether the modem life is the life for you. Some people love it, some people never use it. With modem prices starting as low as $149 (for the Commodore VICMODEM), you really don't have much to lose. If you do buy a modem and find little use for it; call up a BBS and place a classified ad on it: MODEM FOR SALE! It'll sell in no time. If you like, you can leave messages for me on many of the local Toronto BBS systems. Just look for JEFF GOEBEL in the users list. HAPPY MODEMING! •

Using A Modem and The "Punter BBS"

Geoffrey Welsh, Islington, Ontario

Commodore has made using a modem easy and inexpensive with their 1600 line of modems. Still, however, many Commodore users are blind to the advantages using a modem can give you. Once you have a modem, not only can you use it for electronic mail (to ask questions, trade ideas, or advertise that used 1541 you'd like to sell), but also to get the latest computer news and to trade public domain programs. Many of the programs in the TPUG library (especially the C-64 library) are available through local bulletin board systems ("BBS's").

What IS a Modem? A Non-Technical Explanation

MODEM or MOdulator - DEModulator is simply a device that "translates" electronic signals to sound and back again. Those of you who study digital electronics know that there are two logic "levels", 0 and 1. The modem transmits one frequency for 0, and another for 1. Similarly, when it receives certain tones, it sends a corresponding logic signal to the computer. In this way, digital signals can be sent between computers without cables - using only sound. Fortunately, we all have access to a sound network: the telephone.

Responsibilities

I know that some of you will try some of the things I will be discussing here, so I thought I'd better discuss responsibilites now. Using a modem also involves using the telephone, so you must be prepared for wrong numbers, prank numbers, and numerous other mishaps. As much as I hate to think that readers need lessons in basic telephone manners, my experience has shown that the nicest people can forget their manners.

Make sure you call during the hours that the BBS is up. If no hours are given, assume it operates from 7 PM on. Never trust numbers given to you. If you have an autodialler, do not use it the first time you call, just in case the number was wrong or misprinted, or the bulletin board has moved. First time, call at a reasonable time (before 10 PM) so you won't disturb anyone any more than you have to. If a person does answer, have the courtesy to say hello, and excuse yourself for bothering them. There is nothing more annoying than answering the telephone and having the person on the other end hang up as soon as they know a computer didn't answer! (And besides, even at long distance, the minimum charge is for one whole minute whether you use it up or not. - M.Ed.)

There is a misconception that bulletin boards are a great place to swap commercial software and otherwise contribute to piracy. Most SYSOPs (SYStem OPerators) go out of their way to make sure that no copyright programs are sent to or taken from the bulletin board, and that no one uses the message system to

arrange meetings for the purpose of piracy. If the program is worth trading and it isn't copyrighted, upload (send) it to the bulletin board!

Getting down to Business: How to Use BBSes

How to dial and connect to a BBS varies from modem to modem, so I cannot go into it here. Read carefully the documentation that comes with your modem. Once you've done it once, though, it will become second nature.

For the sake of simplicity, I will now discuss how to use PET and Commodore–64 based bulletin board systems written by Steve Punter. There are a lot of these around, and they are all similar. I will discuss all the latest features (including those now being developed), but be warned that most BBSes do not have the latest versions of the program, and some of these features may not yet be available by the time you read this.

Once you and the BBS have been connected, you must tell the BBS that you are there and are ready to sign on. To do this, simply press RETURN. The BBS will introduce itself – it will display its name, the names of those running it, the name of the programs's author, and the time. It will ask you for your first name. Enter it and press RETURN (capital letters are NOT important). It will then ask for your last name, which you enter in the same way. If you make a mistake entering your first name, just type RETURN without entering anything when the BBS askas for your last name, and it will go back to your first name. Once it has your name, the BBS will search for you in its user list. Of course, if you have never used that BBS before, it won't find you. NOTE: If the board is private, it may ask for a password, or it may tactfully tell you that the BBS is private, and hang up on you.

If the BBS finds your name (because you've been on before), it will ask for your user code. Enter it and skip the next two paragraphs.

If the BBS can't find your name, it will ask you if this is your first time on the BBS. If you press 'n' and RETURN, it will ask for your name again (if you have been on the BBS before and it can't find you, it figures that you mispelled your name). Because it's your first time, press 'y' and RETURN. The BBS will ask you for your city & province and a user code so that, in the future, when someone signs on under your name the BBS will know it's you and not an imposter. Remember your user code! You cannot sign onto that BBS again for a while if you forget it! I say "for a while" because most SYSOPs will routinely delete inactive users from the user list.

After you are finished entering your city and code and you have verified that these are correct, the BBS will show you a list of the commands that work on that BBS. This will vary, depending on how old the BBS program version is, and how many custom modifications the system operator has made to it.

Once you have "logged on", the BBS will tell you how many active messages there are, and its hours of operation. Then, what is called an "opening bulletin" is displayed. This is like the operator's "column", and is usually dated and contains important information about what's going on. Make sure you read it every time a new one comes out. To stop the transmission, hit 's'. To resume it, hit 'c' for continue. If this doesn't work, hit 's' again. If, after stopping the transmission, you decide not to read it all, hit 'a' for abort.

After you have read (or aborted) the opening bulletin, the BBS will tell you when you last signed on, and how many messages there are waiting for you. Then the BBS will prompt for "Command" and wait for your instruction.

Editor's Note

At this point an whole new realm lies ahead of you. To explain the details would take all the fun away. Instead, you should try one first-hand. Follow the steps that Geoff has given here and your initiation will be much less intimidating – the main reason I find most computerists are reluctant to approach the local BBS. One thing is certain. . . your first call will determine how many more you make. Nine times out of ten, a new addiction will develope. "Telecomputeritis" afflicts computer users of all ages. In its early stages the symptoms are signing on to as many local boards as possible, followed by checking the phone company for the hours of reduced long distance charges. In its advanced stages you'll notice an urge to start your own BBS and exploring the possibilities for improvements. Seriously, though, it is a lot of fun, something everyone can't try just once. •

Data Communications

Robert W. Dray, Peterborough, Ontario

One of the more powerful uses of a microcomputer is for data communications. Your humble home computer can "talk with", and use the facilities of the large mainframe computers, or it can access some of the large data bases to find answers to almost any question.

There must be a link between two computers if they are to communicate with one another. This link may be one of the following: a series of copper wires joining the computers directly, the local telephone wires, a coaxial cable, a fiber optics cable, microwave networks or even satellites. If the communications is over long distances, it is likely that the signal is carried by several of these.

The digital signal produced by the computer is in the form of a series of discrete on or off signals, while the telephones are set up to carry sounds, which consist of continually varying tones. To convert the computer's digital signal to a form that can be carried by the telephone lines, requires a MODEM.

This device MODulates or modifies a carrier tone that can be carried by the phone lines. It then DEModulates the signal it receives from the phone, back to a digital form that can be used

by the computer. If the modem is an acoustic coupler, it actually has a little speaker that creates the modified tone and a microphone that hears the incoming tone. The hand–set of the phone fits into two rubber cups so that the modem and the phone can "talk" to each other.

Another type of modem is the direct connect variety in which the modem plugs into the wall jack, and the phone plugs into the modem. A switch on the modem allows you to select the normal phone use, or connect the computer to the phone line.

The direct connect modems are less susceptible to interference from such sources as noise in the room, but they are a little less portable since they require jacks or some other means of direct connection, as opposed to simply resting the phone in the rubber cups.

You can get modems that will automatically answer the phone, or even dial for you, but all modems take a digital signal from the computer and convert it to a signal that can be sent over the phone lines, and then convert it back.

The rate at which information is sent can be varied from less than 50 bits per second, (baud) to greater than 200,000 baud. Very high baud rates, would require very good quality communication links, such as fiber optics or microwave systems.

Most small systems use 300 or 1200 baud modems. The higher speed modems are more expensive, but if you are paying for the time that you are connected to another computer, it is nice to have the data transferred very quickly. You would usually dump the data directly to disk, then sign off. Once you are no longer paying, you can look at the data at your leisure.

When computers send signals, each character usually consists of: 1 start bit (this says that data is coming), the 7 bit ASCII code (the actual data), a parity bit (more later), and finally 1 or 2 stop bits (these say that "that's all"). Thus it may take 10 or 11 bits to send one character, which means that 300 baud is about 30 characters per second.

The parity is set at one of the following: odd, even, mark, space, or none. If parity is set at odd, then the parity bit is set so that the total number of "on" bits is odd. If the signal sent is 1100101, so that there are 4 "on" bits, then the parity bit will be '1' to make an odd total. If the parity had been set at even in this example, the parity bit would be left as '0', since the number of bits is already even.

The computer can use this as a check that the data wasn't scrambled in transmission. Of course, if two bits were switched, the parity would still check out and the mistake would be missed, although there are other ways of watching for this problem.

If parity is set at mark, the bit is always '1', and space is always '0'. Sometimes the parity is set at none, and that bit is included to make an 8 bit word, rather than a 7 bit word.

If your computer is working on even parity, and the one you are talking to is working on odd parity, both machines will think that everything they receive is wrong, and communication will break down. In a similar manner, the length of the word may be altered, and if the machines do not work under the same rules, they will mix up stop, start and parity bits with the data bits. Again, there is a breakdown in communications.

The links between the computers are classified as: simplex, half duplex or full duplex. Simplex lines allow data to flow one way only, while the other two allow two way flow of data. While half duplex allows only one computer to talk at a time, a full duplex line allows both to talk at the same time.

Most lines we use are full duplex, although there are reasons why you may want to set your modem to half duplex. Most mainframe computers assume full duplex lines, and they will echo back whatever you send them. If your computer is set at full duplex, it "expects" this echo, and therefore it does not print to the screen when you press a key. Typing the letter Z, will cause your computer to send a Z down the line, where it will be echoed back by the "big guy". Your machine will then print the echo on the screen. If the distances are not too great, and you are not a speed typist, you will never notice the delay.

If the other computer does not echo back the signal, you will see nothing on the screen when you type. You will see only what the other computer sends. To solve this problem, set your computer to half duplex. Now your machine does not expect the echo, and so it prints directly to your screen whatever key you press.

If the other computer is echoing, and you work on half duplex, you will see the character on the screen that was printed when you pressed the key, as well as the one printed when the echo is received. Everything you type appears double on the screen.

When sending files, you have the option of sending an "end of record" signal at the end of each record or not. If one of the computers is expecting the signal, and the other is not sending, there will be communication problems.

As you can see, there are many factors that must be in agreement between two computers before they can successfully communicate. For this reason, one should not be too upset when your first attempt at talking with another computer doesn't work.

Along with the modem, one needs a terminal program to enable the computer to send and receive data. Some of these program do not allow many of the items mentioned in this article to be varied, and unless the other person has the same program, you may have problems getting together.

There are many excellent communication programs available such as PETCOM for the SuperPet, which allow you to vary many of the settings and thus communicate with a wide variety of other computers. If you haven't tried it yet, you have missed a very enjoyable application of your computer. •

The MODEM and RS–232C

Mario Marrello, Rexdale, Ontario

The purpose of this article is to provide an introduction to communications using a remote terminal and a modem. Modem operation, proper set up procedures, and software considerations will be discussed.

In this article, the word terminal will represent the data transmitter; i.e. the terminal or terminal emulating computer. The word computer will refer to the host computer that the terminal or terminal emulating computer is talking to.

This article covers two topics. These are: 1) A discussion of the theory behind modem operations; and 2) The most common form of interfacing for data communications.(i.e. the RS232 standard).

Modems

The word MODEM is a contraction of two words; MOdulator and DEModulater. To modulate means to adjust or vary tone or pitch (as in a speaking voice), or to alter amplitude, frequency or phase of a wave using a wave of a lower frequency to convey a signal. Demodulation is the separation of the modulated signal into the modulating wave and the wave carrying the information for the terminal to process. As the names imply, demodulation is just the opposite of modulation. For most telephone modems, a form of frequency modulation is used.

But why bother with modulation at all?

The telephone lines have been set up to handle voice communications. Therefore they can only handle a limited frequency range. The high speed clicks that a computer uses to communicate will not carry over these lines, so it is necessary to convert these clicks into something that the telephone lines can handle. The modulation process mixes in a lower frequency audio signal, creating a signal that can be reliably transmitted over phone lines. When the signal reaches its destination, it is demodulated and the desired signal is separated from the modulating frequency.

There are many different types of modems and communication 'protocols' but the one used most often by people with home computers is a 300 baud modem with the BELL 103A type protocol. This type of modem is designed to be used over standard telephone lines (i.e. those that are used to transmit voice). A discussion of the BELL 103A protocol follows.

With this type of telephone modem, there are two sets of signals used. Each signal has two different tones designated F1 and F2. Each of these tones is broken up into two more individual tones – these are called 'mark' and 'space' and represent the logic values 1 and 0 respectively.

In the originate mode, (originate meaning that this particular modem is doing the calling) the transmitted tone is designated as F1M (mark) and has a frequency of 1270 Hz. The other tone is F1S (space) and has a frequency of 1070 Hz. In answer mode, (the modem being called should be in this mode) the frequencies are F2M (2225 Hz) and F2S (2025 Hz). This type of modulation is called 'frequency shift keying' (FSK), a specialized form of FM (frequency modulation) which is similar to the technique used for FM radios.

The BELL 103A type modem can be operated in half duplex mode or full duplex mode. In half duplex mode, the signal is sent but not returned by the host computer, therefore all characters that appear on your screen must come directly from your terminal. In full duplex mode, a character is first sent to the host computer then returned to your terminal which finally displays it on the screen. Thus the information you see on the screen has already been seen and processed by the remote computer. This is useful for error checking and implementing special features, for example "hiding" passwords by not echoing back the characters.

(Editor's Note: There may be some confusion regarding use of the terms "half duplex" and "full duplex". The definitions above are commonly used, since data is usually echoed back from the host computer in a full duplex system and echoed locally by the modem or terminal in a half duplex system. Usually, but not necessarily. The terms actually refer to the communications circuits themselves, not the communications protocols. The true definition of a full–duplex system is one which allows data to be sent and received simultaneously; BELL 103A type modems are full duplex, since they use separate frequencies for send and receive. In half–duplex communications, the modem can either send or receive, but not at the same time. (A third system exists called simplex, which allows one way transmission only.) The system of printing characters as they are echoed back from the host computer is called "echoplexing". What's called "half duplex" mode on most modems actually enables modem local copy, where the modem echos everything back to the terminal locally. Alternatively, the terminal itself can print each character as the keys are struck, and the modem need not echo locally – and thus can be used in its "full duplex" mode. It all boils down to this: If the host computer echos back everything it receives, your modem doesn't have to echo – it should be set to "full duplex mode". If the host computer doesn't echo back each character it receives, the terminal itself must be set up for local copy, or the modem may be set to "half duplex" mode where it echos back locally.)

The RS232 Interface

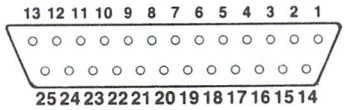
The standard interface used in data communications is the serial RS232C. This interface has an operating range of 0 to 20,000 baud synchronous and asynchronous. Synchronous communications is used with systems that have special 'protocols'. Most IBM equipment uses this form of communication. It is most suitable for sending large quantities of information at once. Asynchronous communications is suited to sending single characters. This is the type of communications available with most 300 baud host computers such as a BBS.

The RS232C standard involves two connectors, a male and a female. In general, (but not always) the female connector is used on the DCE (data communications equipment, i.e. the modem) and the male connector is on the DTE (data terminal equipment i.e. the terminal)

Some other important information on the RS232C standard follows:

- The recommended distance for an RS232 connection is about 50 feet, however, in practice much longer distances have been used with no apparent problems.
- The signal level ranges from –25 V to +25 V.
- +3 V to +25 V is considered a '1' or mark.
- –3 V to –25 V is considered a '0' or space.
- For the control lines (see table), –3 to –25 is off, and +3 to +25 is on. The voltage levels of +25 or –25 volts are considered extremes and a level of +/–15 V is more commonly used.

The DB25 Connector



Female Connector

The chart following is the pin-out configuration for the RS232C standard using DB25 male and female connectors. Anyone constructing a cable for an asynchronous modem should follow these guidelines. DB25 type connectors are available everywhere, but make sure your cable will fit when it's finished – some cables will be male at one end, female at the other, but cables with both ends identical are not unheard of.

It is not necessary to use all these pins when connecting an RS232 line. The most important lines are pins 2,3,7 and 20. Commonly, an RS232 cable has wires connecting pins 1 through 8 and 20, with pins 2 and 3 reversed between the two ends. Thus pin 2, Transmit, goes to pin 3, Receive, from either end, naturally. However, as with every rule there is an exception; some manufacturers attempt to do you a favour by reversing pins 2 and 3 before installing the connector. Although very uncommon, it can happen, which means you'll need a cable that runs pin 2 to pin 2, pin 3 to pin 3. Of course the end result is the same.

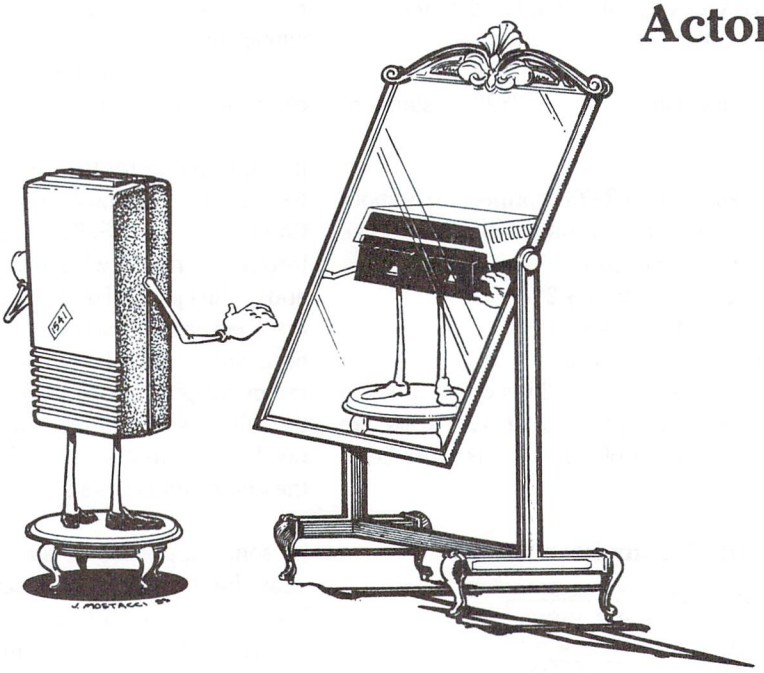
In some applications, such as localized transmissions (see "Easy Intercomputer Connection" this issue), only pins 2, 3 and 7, Signal Ground, are used. As you require more accuracy and flexibility, the other pins will enter the picture. Just how many you need to connect will depend on the demands you make of your particular communications link.

RS232C PIN CONFIGURATION

PIN	LABEL	COMMENT
1	Protective ground	Common physical equipment ground.
2	Transmitted data (TxD)	Data sent to DCE.
3	Received data (RxD)	Data received from the DCE.
4	Request to send (RTS)	Turns on modem's transmit carrier.
5	Clear to send (CTS)	Indicates the modems transmit carrier is on.
6	Data set ready (DSR)	Indicates the modem is on.
7	Signal ground	Ground for signal carrying circuit.
8	Received line signal detector	Asserted when the modem hears a carrier.
9	Not assigned	
10	Not assigned	
11	Not assigned	
12	Secondary Received line signal detector	Used by some high speed equipment.
13	Secondary clear to send channel.	Used by systems with a secondary channel.
14	Secondary transmitted data	Secondary transmitted data.
15	Transmit Clock	For external transmit frequency generator
16	Secondary received data	Secondary received data.
17	Receiver signal element timing.	Timing signal, synchronous modems only.
18	Not assigned	
19	Secondary request to send	Turns on secondary carrier.
20	Data terminal ready (DTR)	Asserted by terminal when ready to use modem.
21	Signal quality	Used by some fancy modems.
22	Ring indicator (RI)	For electrically detecting a phone call (auto–answer modems)
23	Data signal rate selector. (DCE)	Make a connection in high speed mode.
24	Transmitter signal element timing (DTE)	Timing signal sent to the DCE.
25	Busy	May be used by auto–originate modem.

A Comedy Of Errors

David Sale
Acton, Ontario



Introduction

The idea of communicating with other computers over the telephone lines has fascinated me for years. There are many traps, however, for the unwary explorer.

The tale that follows is a true account of my first experiences with a modem. Perhaps my misadventures can forewarn others and make their first attempts more satisfying.

Prologue

My love affair with the modem began innocently enough. It looked like a good idea to save some time while preparing university assignments at home. It would open my door to the great memory banks at "The Source" and allow me to send electronic mail.

The only problem was the modem and I seemed to have incompatible personalities (we hadn't visited a computer dating service) and it wasn't long before the marriage went sour and a divorce was brewing.

I was totally to blame. I was overly demanding and insensitive to her needs. I didn't appreciate her finer points. But maybe I should start this tale at the beginning.

Error 1: You Want A What?

I visited my local computer store armed with all the buzz words. Parity, baud rate and half-duplex rolled off my tongue effortlessly. The salesman was suitably impressed until I mentioned that I had a Pet 4032 computer.

It took me nearly five minutes to calm his hysterical laughter and between chortles he was only able to plutter, "But that's been out of production for months. You don't expect me to STILL carry peripherals for it, do you?"

I muttered an apology for being so presumptuous and slipped out the back door.

Error 2: Direct Connect To What?

Thumbing through back issues of computer magazines I located a supplier of PET modems. My money order was in the mail within an hour and for the next weeks I waited, fantasizing about the fun I would have when my modem unlocked the door to the real world of the mainframes.

When the simple grey box finally arrived, I fondled it lovingly then attempted to connect it to my computer and telephone. But wait, something was wrong... The picture in the box showed the modem neatly connected BETWEEN the hand set and the base of the telephone but my hand set was not removable.

Error 3: There's More Than One Size?

The local electronic supply store provided the necessary plugs and jacks to solve the problem and within a mere three hours I had a telephone with a removable hand set. The hand set lead was now encased in black electrical tape and plastic shielding and would never be caught dead in the finer homes of suburban Mudville.

It was a small sacrifice, one that we must be prepared to make if we are to enter the communication revolution. I quickly forgot its sorry state and began once again to attach the modem. First, you detach the hand set and plug it into the modem. What a time to find out that telephone jacks come in slightly different sizes and the ones I had so carefully attached were the wrong size.

Error 4: Which Software For Which Port?

Another hour in the basement workshop cut the plugs down to size and all the parts went together. I turned on the modem and began to explore her software. Although it was beautifully written, there was no provision for saving incoming messages.

I loaded another modem program that I had acquired some time before. The screen went blank. I typed furiously but the distant computer failed to answer and my long-distance call was cut off.

Disassembly of the machine code revealed that the modem program was directing the output to the PET's parallel port instead of through the IEEE where the modem was attached. I reluctantly reloaded the original driver and redialed.

Error 5: Half, Full or Empty?

Merrily I typed along, pecking my way through the access codes. When I looked up from my pecking, the screen was once more totally blank.

My poor modem, my exasperating modem was a full-duplex model, while the computer I had contacted was half-duplex. My signals were not being echoed back to the screen correctly.

The only solution was to tear apart ten pages of machine code to insert the necessary echoing. Otherwise, I would have to explore this strange new world with a white cane.

Error 6: Password Please!

Now I was on a roll. Nothing could stop me. The great gates were about to open. I reached the trunk line, raced past the computer access codes and entered my password. The distant computer politely burped and printed "Invalid Password".

I re-entered it with the same result before my connection was cut. It appeared that only Ma Bell was going to come out a winner that day.

A short call by conventional telephone to the computer center (once again long distance) revealed that a secretary had inadvertently reversed two letters when informing me of my password. 'Aliment' became 'Ailment'. Computers are notoriously unsympathetic to perceptual handicaps.

Error 7: Cursed Cursor!

What else could possibly go wrong to spoil that budding relationship with the outside world? I didn't want to ask so the computer told me immediately. . .

No sooner was I on line and shaking hands with my new mainframe buddy than my screen began to fill with garbage. 'Enter' became 'Edqndqtdqedqrdq' and 'Print' was 'Pdqrdqidqndqtdq'.

Steam was beginning to escape from my ears but fortunately not in sufficient amounts to blind me to the obvious pattern. The regularly recurring 'dq' had to be some kind of cursor prompt from the host computer. My dumb terminal could not be translating from standard ASCII to PET ASCII correctly.

More machine code disassembly and another added loop trapped the incoming 'dq's and converted them into a block and backspace to simulate a cursor.

Full screen editing didn't work, however, and any errors required that the entire line be re-typed. But that wasn't going to stop me.

Error 8: It's My Party!

Now I was running. I was talking with the real world. I could feel the power flowing from my fingertips as I carefully typed line after line of letter-perfect code. I lost all track of time (but Ma Bell didn't. . .) and an hour later I had successfully written my first Fortran program on a distant computer.

All I had to do was enter the Run commands. . . but disaster struck. My screen display jumped, the phone connection died and my program was lost. Someone on our party line had picked up the phone and attempted to dial, not recognizing the weird buzzes as earth-shattering electronic communication.

Epilogue

More than fifteen hours had been spent to "save time" on a one hour assignment for my computer course. My telephone had been butchered and the telephone company could erect their new office building with my next month's bill. I sat limply staring at the screen then turned off the computer and went to bed, licking my wounds.

As for the modem, perhaps I was too harsh. That subscription to "The Source" was still good and it certainly was a lot cheaper than an encyclopedia. Perhaps I could investigate the local bulletin board or talk to other computer nuts.

Maybe I should reconsider and give her another chance. After all, it was my fault, my ignorance and my stupidity. I think I really would like to try again. I should be able to find that sleek grey box. I couldn't have thrown it that far. . .

World Connection

Jim Grubbs
Springfield, Illinois

Telecompute Without The 'Phone'

With your personal computer and a modem, you are able to access thousands of computerized data bases around the world. Everything from the local bulletin boards, to research computers at universities, and special "added value networks" like to Source and Compuserve are no further away than a telephone connection.

The prices for these services vary of course. Many local BBS operations are free, some require a small yearly membership fee, other services charge an hourly connect rate. In addition to these charges our old friend the phone company gets a very large piece of the action.

Would you be interested to know that there are many data services that are swirling around you at this very moment that don't cost a thing, not even a phone charge? A special version of bulletin boards, nearly all the major press services, international weather information, military services and a lot more enter your home twenty four hours of every day. There are even computer hobbyists exchanging programs over thousands of miles without it costing one penny for telephone charges.

No, this is not an article about cheating the phone company. It is a perfectly legal way of connecting your computer to the outside world. All of the services mentioned are available for the asking on the international short wave bands!

Let's take a moment to consider normal data communication. The first ingredient is a computer or terminal. Most of us these days use our home computers rather than a dedicated terminal. In order to make our computer act like a terminal, software is required. The software can be very simple. Many computers come with a terminal program built into them, or at least provide a simple BASIC listing for a terminal program.

Terminal programs can become very complex, allowing for features like auto dialing, built in text buffers that act like mini–word processors, and other bells and whistles. It is then necessary to take the digital data formatted by the terminal program and convert it into audio frequency tones that can be sent over regular telephone lines.

There are hundreds of modems available to accomplish this conversion. The simplest of modems handle only the data to tone conversions necessary. More advanced units may contain some or even all of the necessary terminal software on chips inside the modem itself. These units are known as smart modems.

The final link in the chain is the telephone line. It serves as a "carrier" for the data much as it serves as a "carrier" for voice transmission.

Data communications via radio follows these steps too. First you need a computer or terminal. Next, you need terminal software for your machine. Since radio transmission of data is not as universal as telephone transmission, the software must be more versatile than the usual terminal package. Many different protocols are used. In order to get the maximum use of a radio connection your software must be able to handle most of the commonly used formats.

It is still true that the digital data from the computer must be transformed at some point into audio tones. This can be done at several points, but with modern day technology, it is usually done once again in a specialized modem called a terminal unit. Functionally a terminal unit or TU as it is abbreviated operates much like a modem, but selects different frequency pairs for the audio tones than are normal for "land line" communications.

The final connection is to the radio transmitter if you are the sending station. Most of you are interested in being on the receiving end, so your final connection will be to a short–wave receiver.

Becoming an SWL (short wave listener) can be a lot of fun, whether your interest is in intercepting data transmissions, or simply listening to broadcasters from around the world. Several international short wave broadcasters have regular features spotlighting computer hobbyists. Radio Nederland has even experimented with direct transmission of computer programs to its listeners!

The selection of a short wave receiver does not have to be difficult or unnecessarily expensive. You should look for a unit that covers the full frequency range from about 3 to 30 megahertz. The unit must be stable, so that the signals don't drift after you tune them in, and it must have a BFO (beat frequency oscillator). On some units this may show up only as the unit having SSB (single side band) capabilities. Acceptable units range from a cost of about $200(US) new at your local discount store, to $700(US) or more for the Cadillac of short wave receivers. Still sound like too much? Particularly during the summer months, amateur radio clubs sponsor "hamfests". This is nothing more than a big electronic flea market. Typically you can find older short wave receivers that are somewhat bulky by today's standards but still perform quite well in the $50 to $100 range. Try to enlist the aid of either an experienced SWL or amateur radio operator before making your purchase.

After you have the receiver, you will need a terminal unit. There is a fairly wide selection of terminal units available, ranging in price from about $69.95 to over $500. For your purposes (unless you insist on driving a Cadillac) the lower range units will work quite well. Several units are designed for the SWL specifically, while many others include the hardware necessary for transmitting. Unless you are a licensed amateur radio operator you really don't need this ability. This feature will go unused for the SWL.

The final consideration is the software. There are many software packages available ranging from $13.95 to over $100. The old adage of you get what you pay for applies here. Many of the inexpensive programs are written in BASIC. That's OK for some applications, but a

real hindrance in others. Not all software will support all modes of data reception. Let's look at what is available as you tune across the bands.

The earliest form of data communications was a simple on and off keying of a signal. Samuel Morse created a system of codes for sending messages that is still used. You will still find numerous morse code or CW (continuous wave) stations transmitting today. Commercially, this type of transmission is mainly used for ship to shore messages and of course can be found on the amateur radio bands. Most morse code transmission takes place in the under twenty word–per–minute range. With the proper software and a terminal unit, you will be able to have morse code printed directly on your monitor screen at well over sixty wpm with no knowledge of the code on your part!

One of the next forms of data communications was the teletypewriter. When connected by radio it is called RTTY for radio teletype. Most commercial services use a somewhat slow 50 baud or 67 wpm speed. Teletype code is structured differently than computer ASCII code. It consists of only five bits of data and one start and stop bit. The number of different characters is therefore limited. Many South American countries as well as African nations still use this form of transmission. The Navy still uses it too. You'll find press transmissions (frequently in Spanish) and lots of weather information. At 45 baud or 60 wpm you will find amateur operators exchanging information via RTTY.

Perhaps of the most interest to the compter hobbyist are the MSO (message storage operations) that are available on the ham bands. An MSO serves as an international BBS where messages about equipment for sale, new computer programs, and other related information can be found.

RTTY is also transmitted at 75 wpm and 100 wpm. Most modern organizations use 100 wpm. Included in this category is the US Information Service. I recently copied the entire text of a speech President Reagan hadn't given yet!

Transmissions also occur at different shifts. Amateurs and some commercial stations use narrow shift at 170 hertz. Typically, most press transmissions use that more standard 450 hertz shift, while the Navy and some older systems still hang on to 850 hertz shift. This can be an important consideration when choosing a terminal unit. Not all units will handle all the different shifts.

Why, you may ask, doesn't anyone use ASCII transmission. Although such transmissions are still the exception rather than the rule they are beginning to show up. The protocol is virtually identical to land line communications, so almost any terminal software can be used. I have even been successful at connecting the audio from my short wave receiver into my modem (not recommended unless you know what you're doing!) and copying radio ASCII transmissions that way!

Perhaps the final form of data communications you will encounter, particularly on the marine channels is TOR (teletype over radio). This form of transmission uses the Moore code, a specialized seven bit code. It is structured to minimize transmission errors. Software to decode this method of transmission has become available to the home computerist in the last year or so. It is a fascinating system, but somewhat beyond the scope of this overview.

Generalizations are nice, but specific examples often help to make new ideas clear. Let's look at a unique software/hardware package that is currently available from Kantronics for the Commodore computers.

Appropriately named, SUPERTAP comes with everything you need to connect your computer to your short wave receiver. The package consists of specialized terminal software on cartridge, a receive only terminal unit, a power supply to run the TU and a guide to what you can find on the bands. Thoughtfully, an audio cassette tape has been included to help guide the novice SWL in tuning morse code and teletype signals properly.

The overall performance of SUPERTAP is quite good. The software allows reception of morse code, radio teletype, TOR and ASCII transmissions. It also contains several features unique to the SUPERTAP package.

It can often be difficult to determine the speed of transmission of an RTTY signal. The "scope" function allows you to use your computer monitor as a storage oscilloscope to help determine the speed of the sending station.

In addition to the "standard" Baudot RTTY code, some stations "scramble" the normal code for security purposes. The SUPERTAP package allows you to try different combinations of bit inversion in order to break these codes. It can be fascinating watching the software "do its thing".

The weak link in the package would have to be the terminal unit. The circuitry used does not have the interference and weak signal abilities of some more expensive units. It is still a good dollar value however, and more than adequate for the beginning SWL. I've just been spoiled by units such as the Kantronics Interface II that do a superb job as terminal units!

There is another category of packages available that combine the terminal software and terminal unit on one plug in module. The AEA Micropatch falls into this category. Such units are particularly attractive for someone wishing to minimize the amount of space and the number of wires that must be connected.

So why not think about linking up with some new forms of data communications via short wave. I've included a listing of some of the more popular suppliers of software and terminal units.

Suppliers of SWL Hardware and Software

AEA, Inc.
P.O. Box C2160
Lynnwood, Washington 98036

MFJ
P.O. Box 494
Mississippi State, MS 39762

Kantronics
1202 East 23rd Street
Lawrence, Kansas 66044

Microlog Corporation
18713 Mooney Drive
Gaithersburg, Maryland 20879

Interested in more? Send Author correspondence to:

Jim Grubbs
P.O. Box 3042
Springfield, IL
62708 217 753–1995

Electronic Mail In The Office

Aubrey Stanley
Mississauga, Ontario

Office work involves a major proportion of the total work force in one way or another. So it is not surprising that a great deal of effort is expended in automating the office, especially in this decade of sharply reducing hardware costs. Electronic Mail will account for a large chunk of this automation process simply because mail is the life breath of the office. If the mail suddenly stopped to move, most of our office activities would come to a standstill.

If we had to choose an alternative to the traditional forms of paper–based mail, what exactly would we go for?

Before we begin to answer this question, we should consider carefully just what it is that we are replacing! Faster delivery on its own cannot make up for convenience and flexibility. An Electronic Mail system is not just one word processor talking to another over a point–to–point link or via some store and forward facility. This may be a quick way to send a letter, but it has little effect on the overall conservation of time in the office. Any move towards Electronic Mail should be a step closer to conserving this most valuable resource and avoiding repetition of effort.

If we use the Post Office to mail a document, we first of all have to prepare the contents on paper, place it in an envelope, address the envelope and then deposit the envelope in a postbox. From that point on we rely on the carrier to deliver the envelope to the mailbox of the receiver. Then its up to the receiver to check the mailbox for received mail.

The key factor is the use of an external agency to deliver the mail. The carrier will automatically forward the mail if the receiver has changed address. We need not be aware of this fact. And if we have requested recorded delivery or registered the envelope, then the carrier will perform these functions as well. Electronic Mail should first of all provide the basic functionality of the traditional service.

Electronic Mail is similar to paper based mail, only the media is electronic. Envelopes of electronic data are moved between the sender's postbox and the receiver's mailbox by some sort of computer based distribution process which replaces the traditional carrier. The improved performance of electronic distribution gives Electronic Mail its great advantage over traditional mail, and the data, in the form of electronic files, can easily be integrated into the automated office system.

In the office, the IN TRAY is used to store incoming mail prior to processing. In the electronic office, the In Tray stores incoming mail in electronic form. Associated with the In Tray is an IN LOG which holds IN NOTES for all incoming mail in chronological order. Each In Note contains a brief description of the item in the In Tray, including things like sender, date, reference/subject headings of a memo, urgent status, reply requested status, etc. The receiver can rapidly scan the In Log and select particular items from the In Tray for display or printing. Mail may be forwarded to other persons without having to be rekeyed. Automatic Reply facilities enable a reply to be generated without the tedium of redoing the address list. Text from the received document may be extracted and merged into the reply with comments, etc. The Follow–up facility enables you to remove a message from your In Tray for a specified period of time – useful for sending reminders to yourself on upcoming appointments.

The electronic OUT TRAY is obsolete in the traditional sense, as mail is immediately posted. Instead the Out Tray becomes a means of tracking mail in progress. A copy of each item sent is held in the Out Tray and the OUT LOG can be scanned for status on a particular item. The sender can determine whether mail was delivered to the receiver's mailbox, if the receiver had seen it, whether a reply has arrived for it, etc.

Each user of the system has a MAILBOX. This is really a combination of the In and Out Tray facilities. Review of mail in the Mailbox may be further enhanced by selective search facilities. For example we may review mail by sender or receiver name, date, subject string, etc.

When an item in the Mailbox is no longer required, it may be deposited into the electronic WASTEBASKET. It remains there for some specified period before it is disposed of completely. Until then, the item may be retrieved, much like fishing out a letter you may have crumpled into a wastebasket by mistake.

Just as you may fill in a formatted sheet on your desk, so also in Electronic Mail you enter appropriate fields on your terminal screen. These fields define the address list, subject, reference, date, message content, etc. Lines of text from some other document (from your In Tray, for example), may be extracted and merged, perhaps with annotations. A pre–defined distribution list file may be copied into the address list. A report created on your word processor or the listings from your Cobol compilation may be attached to the document being prepared.

Attributes may be specified, such as urgent, confidential, reply requested, delivery confirmation, postdated delivery, etc.

The Electronic Mail system maintains a personal PROFILE on each user. Each profile contains user and system modifiable parameters. A user may record a forwarding address for automatic forwarding of his or her mail; a secretary may be authorized to view and/or send mail through her manager's mailbox; the system may maintain statistics on each individual's use of the system, etc.

The backbone of the Electronic Mail system is its DIRECTORY. This is normally on–line to users for interrogation of names, locations, departments, telephone numbers, titles, etc. This DIRECTORY ASSISTANCE facility may be used for automatic filling in of addressee details in the receiver list. For example you may open the Directory at a particular user initial, scan the contents, and mark the required name for transfer to the receiver list.

The above are but a few examples of the potential of an Electronic Mail system.

When a document is prepared for posting, it is passed over to the Distribution facility for mailing. A copy is maintained in the Out Tray of the sender and the document is mailed to each person specified in the address list, using information supplied in the Directory. In fact, the destination need not be a person. You may wish to send mail to the "mailroom" ("hot printer") at a particular location, or to a department printer. For example, your boss may be visiting another location where he or she does not have mailbox facilities. Knowing the department he or she will be visiting, you may address the document to the department printer at that location. Or alternatively, you may specify the department secretary as the recipient. You need not be aware of details like printer number, secretary name, etc. All this information will be extracted from the Directory.

There are several possible implementations for an Electronic Mail system. We will describe one. In this case, a computer (mainframe, mini or super micro) serves the needs of one or more offices. A user at any office ("location" to the computer) accesses the computer from his or her terminal. This may be through dial–up facilities or maybe the terminals are connected through a Local Area Network (LAN). The terminal could be any suitable one – an IBM 3270 or PC, even a Commodore 64. The computer will format the screen for the terminal being used. The user is known to the computer by the special sign-on procedure used. The user's profile is fetched and the name, location, department details, etc, recorded. These will be used to automatically insert the sender details in any document mailed. The user is then given access to his or her mailbox and informed if any new mail has arrived. A Master Menu screen permits the use of any supported features – In/Out Tray, Create mail, etc. Say the user prepares a document for mailing. When the "send"key is pressed, the computer will verify the input, and if everything is in order, a unique serial number is assigned and the mail item is written to the DISTRIBUTION DATABASE. This database serves as the POSTBOX for users of the local computer.

The Distribution program services its Database. Running in background mode, it does not interrupt the user who has just posted an item. The user may continue with other work in foreground. Mail items are processed one at a time from the database. An In note is generated for each receiver who has mailbox facilities on the local computer, and this is written to the receiver's In Log in the receiver's LOCATION DATABASE. Each office has its own Location database to store all mail generated or received for that office. The In Note will record the item serial number and serve as an index to the mailed item. The body of the mail item is itself written to the Location Database (but only once) in the form of Addressee and Text records, each keyed with the serial number of the mailed item. Thus only one copy of the mail item is retained in a Location Database with the In Notes recording receive status for each receiver at that location. When a user scans his or her In Log, the In Note will point to this copy which is then used to provide the user with the required information.

As well as creating the In Note, the Distribution task creates an Out Note in the sender's Location Database.

If a receiver is recorded in the Directory as existing at some other office not connected to the local computer, then steps are taken to ship the item to the remote computer handling that office. This is done via a suitable communication link. When the item arrives at the remote computer, it is stored in the Distribution database from where it will be processed by the Distribution program in exactly the same manner as described above, except that because it has arrived from another computer, only local processing will be done, i.e. no further remote shipping will take place.

The Distribution program also services status records. These are automatically generated and written to the Distribution database whenever certain specified events occur. For example, delivery confirmation has been requested and an In Note has just been written to the receiver's In Log. Or the receiver has seen an In note for the first time. Or a reply has been posted to a particular received item. When the status record reaches the sender's end, the serial number will be used to locate the sender's Out Note and the status will be updated.

In this manner, an Electronic Mail system may be built up with one or more computers as necessary, in order to meet the requirements of any organization, large or small

As Electronic Mail systems begin to proliferate, we should bear in mind that the media is not the message. In other words, the media should not be allowed to adversely affect the way in which we communicate. If it is to serve the office well, Electronic Mail should give us more time and greater freedom to communicate more effectively with our fellow human beings.

Computer Networking Systems

Richard Evers, Editor

A Look at 3 Popular Multi-User Networks

In a multi-computer environment such as a classroom or lab, the ability to share disk drives and other devices among computers can save money and give a means of inter-computer communication. There are several systems giving that ability on the market today, each one approaching the same problem in different ways. These systems are not true LANs (Local Area Networks), but are often referred to as "multi-user networks" or "queuing systems". In this article, we will call them networks for the sake of simplicity.

In most educational institutions that use Commodore computers within their course structure, computer networking systems are, or could be, a very worthwhile investment. Through the use of a good networking system, a great cost savings on hardware such as printers and disk drives can be attained, as can a better control over the usage of the equipment by the students. As a final financial bonus, disk protected software can often be utilized by an entire classroom through the use of one disk drive. Higher education at a lower cost per student. A controllers dream.

A network, in very simple terms, is a method of connecting a number of computers together to allow for shared access of peripheral devices. The network is responsible for allowing each user access to the peripherals, usually one at a time. (An ideal network would allow simultaneous access by timesharing, but no such system seems to exist for Commodore equipment.) All users waiting for time are placed on a 'waiting list' (job queue) until time becomes available. The job queue should be accessed on a first in, first out basis.

Some networks give bonuses to users of the system. Extra commands in BASIC, special password schemes, and file protection tricks, just to list a few. A bonus is a nice present, if you need one. Some people may choose to get the extras, and love everything about them. Others may just want simplicity, and therefore go without the extras. One point to note about a networking bonus. In order to supply the extra favours, modifications often have to be made to the computer. Simplicity verses favours. A big decision.

Below will be found the names of the main manufacturers of networking systems in the Toronto area. The descriptions of the networking systems following the names is purely based on manufacturers claims through their literature and manuals. Though this may seem like a very odd way in which to supply you with information, we felt that it was the best. Of the three systems described, each have their good and not so good points, depending on whom you speak to. Everybody has their own likes and dislikes regarding all facets of human existence. A review of each networking system by one person would probably give the reader an inaccurate view. Numerous reviews of each system would serve little purpose but to waste an entire issue. Needless to say, the best judge will be you. Read through each description and look for the good and bad points. The systems have all been tested extensively in the classroom for years, so actual flaws should not exist. The only not-so-hot points will come from your ideas on what a network should do.

1) The Arbiter by Batteries Included
2) The Microshare by Comspec Communications Inc.
3) The Mupet II by BMB Compuscience Canada Ltd.

Another popular network is the Multi-Link from Richvale Telecommunications. We requested information on this system from Richvale, but since we didn't receive it before our deadline, The Multi-Link was not included in this article.

The Arbiter

Manufacturer : Batteries Included
30 Mural Street
Richmond Hill, Ontario
L4B 1B5
(416) 881-9941

Applications : Pet and CBM

The Arbiter system is one which performs special favours for the users. The price for these favours comes in the form of a new Arbiter E ROM for each computer in the network. Each new Arbiter E ROM comes with an ID number which is contained within the ROM itself, and printed on the top of the chip. Since there are a maximum of 32 computers allowed in the network, the numbers range from 0-31. Although this may seem odd to you, there is logic in assigning numbers. The Arbiter system requires one computer in the network to act as the Controller. The Controller gets a special device installed in it called a Master Control Unit, which is the control mechanism of who gets what device when. Every computer in the chain has its User port and IEEE port tied up by the Arbiter system, with each computer being connected together in series through 4.5 foot ribbon cables. Through a bit of bit manipulation at the User port, the computer with the Master Control Unit attached detects any activity on the bus, and determines which computer, 0-31, originated it. In this way the path to single user access to the peripheral devices for the time required is cleared.

The system comes complete with Arbiter ROMs, Bus–Boards with 4.5 foot ribbon cables, and one Master Control Unit. The Master Control Unit also requires a bit of installation work before it is operational. There is a connector hooked up to the Master Control Unit that is to be passed through the IEEE port opening and connected to J–11 on the expansion port. To hook up any IEEE peripheral device to the system, little trouble will be encountered. Each Bus–Board in the system has the capacity to accept up to 2 Pet to IEEE cables, Pet side connecting to the Bus–Board. "Any reasonable number" of IEEE devices can be attached to the system, as stated in the owners manual supplied. By talking to Paul McCourt of Batteries, I was able to ascertain that 'reasonable' meant as many IEEE devices as the Pet normally can handle, ie. 12 devices (device numbers 4 through 15).

All Basic commands remain the same excepting those working with the extra peripheral devices. In direct mode, the bus will be released immediately after access to the device has ceased. In program mode, a 2 second delay has been implemented in releasing the bus.

As mentioned in the beginning, extra favours, ie. commands, have been written into the system for the user. They are as follows:

Extra Basic Commands

LISTC : lists Basic program in memory to attached printer
LISTP : prints user–defined heading line then lists Basic program in memory to printer (will also translate cursor control characters into mnemonic representations)
@S"filename",YYY,d#N : saves prg "filename" to drive #N with user code YYY
* once @S started, 5 character password asked for to assign to protected program
@L"filename",YYY,d#N : loads prg "filename" from drive #N, user code YYY
* once @L complete, 5 character password asked for before access can be had
@G : grab the bus and do not allow any other access to it
@R : release the bus to normal user access

A few points to ponder regarding the Arbiter. The new E–ROM taps into CHRGET to extend the BASIC command set and allow queuing up for bus activity. Although this does work, it may also slow down Basic execution a tad and play havoc with the use of CHRGET–powered Basic utility programs. The manual states that the Arbiter works just fine with Waterloo Basic, but also notes that if Waterloo Basic is deactivated via sys(9*4096+3), the Arbiter will also become disabled. Re–enabling the Arbiter is the only cure. Problem number two is rather minor. The scanning of the keys has been messed up a bit, so when pressing (Shift) (Run/Stop) to dL"* / Run from disk, release the (Run/Stop) before the (Shift) key, else a break will occur. Not too heart breaking, but an oddity to remember.

One other point to consider with the arbiter is the fact that it uses the user port. This may be a problem in an electronics lab or such where external circuitry is to be connected to the PETs.

When you buy the system, a utility diskette is also included for the teachers. On this diskette will be found "Load and Run", "One Shot", and "Two Shot". The purpose of each is as follows

Load and Run – allows viewing and marking of student programs without using the assigned password
One Shot – de–protects password scheme if protected once
Two Shot – de–protects password scheme is protected twice by mistake

That's everything that could be deduced through their literature and manual. Although I have not given you a blow by blow users description of the system, you do have a knowledge of what it is supposed to do, and can now place it beside the following systems for comparison.

The Microshare

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

The IEEE Microshare
Applications : Pet, CBM, 8296, B Machine

The Microshare does not change the computer's operating system in any way. It simply provides a transparent networking system for any IEEE computers on line. Transparent in this sense means that the computer has not been altered in the least, hardware or software, therefore any software package in any language will work as with a normal computer. You can have a maximum set–up of 16 IEEE computers Microshared at any time, with a maximum distance from the Microshare by any one computer of 100 feet. This is a unique feature. The computers are not linked to each other in series. They are connected directly to the Microshare via separate ribbon cables. The ribbon cables connect directly to the computer through the IEEE interface on the back, just as the standard Pet to IEEE cable does.

Another special bonus with the Microshare is the 'first come, first served' basis in which access to the bus is given. A job queue is present that is constantly refreshed due to any request for activity on the bus. Even while a computer is using the bus, polling of the computers is still active, thus giving a true networking environment. To also supply assistance to the system supervisor, the Microshare has a series of LEDs across the front panel to show which computer is accessing the peripherals at any time. By watching this display, a constant reminder will be in place as to the top users of the peripherals in the class or office.

On Microshare power up, a series of diagnostic routines are performed to check the system out. If any trouble is encountered, the LEDs across the front will display the error condition. Once the system has passed the test, immediate access to the peripherals can begin. The peripherals are hooked up to the Microshare through two edge-card connectors using Pet to IEEE cables. On the left side, up to two devices may be hooked up that have the ability to talk and listen. This will allow up to two disk drives, or one drive with any other IEEE device. On the right side is another edge card connector, of which up to 5 listen only devices can be connected. Listen only means devices that can accept commands but not reply back, like a printer.

According to the manual and literature supplied, the system in uncrashable. Even if a user crashes his or her computer, taking a peripheral along for the ride, the system will recover after 16 seconds if the peripheral is down, or 2 seconds if the computer is down but the peripherals are still alive. It depends on the crash encountered. Whatever happens, life will continue for the balance of the users. The defaults of 16 seconds and 2 seconds can be changed through the use of DIP switches located on the bottom of the unit. The 16 second bus active with computer activity delay can be sped up to 8 seconds, and the 2 second user non-access delay can be sped up to 1 second.

With the system being as simple to use as it is, no other information is required. No special tricks, therefore no special procedures. Now, on to the serial version.

The Serial Microshare

Applications: Vic 20, C-64, Plus 4, C-16, and C-128

If you were to read the description above, most of it would apply. The exceptions are as follows.

This system allows networking of Commodore computers through their serial port, serial as it applies to the 1541 drive. In this way, networking the C-64, Plus 4, C-16, and C-128 can be achieved without additional hardware. There is a maximum of 8 serial computers on the network at any time. The system is still transparent, no hardware or software modifications to the computers, but a few special favours are supplied. The favours: a built in 14K print buffer with software selectable device number and de-spooling feature; individual disk error status reports for each user; individually controllable channel switching delay; and a 'group load' option which allows any number of users to simultaneously load a program. Pretty impressive for a transparent system.

The system has been designed a little differently than its IEEE brother. It allows either 7 users or 8 users, a user-selectable option. The advantage of the 7 channel set-up is that it supports both serial and IEEE parallel activity, using the 8th unused port as the serial output port. The eight channel set-up only allows for IEEE activity.

The IEEE output port can support up to ten IEEE devices daisy-chained together at any time, with the serial port being capable of handling up to 5 serial devices daisy-chained together. The system allows talk and listen devices all around, therefore whatever peripheral devices you have at your access can always find a home. One special point to mention here. The system allows the user to specify which device number is to receive its printed output via the print buffer. A pretty sharp trick to change device numbers while using a print buffer.

The diagnostics are about the same as with the IEEE Microshare, but have a built-in Beeper to let you know when all is Ok. The beeper is also used for a particularly useful trick. If a user encounters a disk error while using a drive unit, the Microshare beeps to inform of the trouble. Here's another useful favour performed. Disk activity always updates the error status of the drive. Normally this status will be altered as new users access the drive. The Microshare handles this in an interesting way. Each user can read his or her disk error status via OPEN 1,8,15: INPUT#1,a,b$,c,d . . . and be assured that it is correct, even if new disk activity has been performed by another user. The reason for this is because the Microshare retains each user's disk status within Microshare RAM.

Most of the special tricks performed by the system are instigated by talking to device number 15, the Microshare. The statement OPEN 1,15,8,"filename" will inform the Microshare that a group load of the program "filename" is about to take place. From here, each user who wants to load the program should type in and execute LOAD"*",15. Once the first computer who issued the group load command actually starts the load, the group load begins, and all computers in the network LOAD the desired program simultaneously. Another smart feature incorporated is the de-spooling option. By either transmitting the request to device #15, or pressing the de-spooling button on the Microshare, the current user's printer output can be terminated prematurely, without harm to any other data in the buffer. The other favours supplied have equally interesting effects, and most can be instigated with just a little conversation with device number 15.

This article could go on much further extolling the manufacturers claims to virtue, but not today. It's time to move on to the final competitor in this networking free for all.

The Mupet II

Manufacturer : BMB Compuscience Canada Ltd.
500 Steeles Avenue
Milton, Ontario
L9T 3P7
(416) 876-4741

Applications : Pet, CBM, 8296, B Machine, and Commodore 64

The original Mupet has been around for quite some time, with the Mupet II being an expanded version of the same. Part of this expansion is a Commodore 64 Mupet module to allow for a networked system of IEEE computers and C64s together. The

system is transparent, meaning it doesn't change the computers' personality in any way, but it does perform a few favours for the user as a bonus. The IEEE Mupet module connects directly to the Pets, CBMs, and other IEEE computers through the IEEE port on the back. The C64 Mupet module connects through the cartridge port of the C64, and has a jumper that is to be connected to a chip within the C64 for software transparency. Each computer in the system has a Mupet module and a length of ribbon cable to daisy chain itself to the next computer in line. There is a maximum of 16 computers in the networking chain at any time.

The system has a few special components. One Mupet module for each computer in line, ribbon cable sufficient to network the system together, a power supply, a terminator to mark the last Mupet module used, and one Mupet II Controller. Recommendations are that not more than 100 feet is spanned between the Mupet II Controller and the furthest computer in the network. According to their literature, 200 feet can be attained under the absolute best of conditions.

The output from the Mupet II Controller is an IEEE port, a parallel port, and an RS–232C port. The recommendations by the manufacturer are to not connect more than 8 IEEE devices to the IEEE port. No maximum was stated regarding the RS–232C port or access to the parallel port. One very novel point to mention regarding the output ports: the ability exists to re–direct output to any device chosen. This means that the Mupet II Controller can re–route output generated by software packages to device numbers of your choosing, without the software realising it. Another neat trick is the ability of the system to translate the printed output either to Pet Ascii or True Ascii, depending on how you set it up.

The RS–232C serial port is a handy creature to have around if you are debating telecommunications work. It has a default setting of 1200 baud, with 1 stop bit, a 7 bit word, and no parity. Quite simply, high–speed telecommunications at a glance with correct parameter set–up of the Mupet II Controller. It appears that this system was designed with excellent telecommunications facilities in mind. Perhaps this is due to BMB's involvement in the design of the SuperPET, which Commodore later bought the rights to.

As with all networking systems described so far, a polling process is carried out that checks for activity on the bus, and queues up for single user access of the peripherals. A pretty useful favour this system performs is a print spooling routine that will list a Basic program from disk directly to the printer, or print a sequential file as it appears on diskette. The spooling procedure is unique in that it will un–tokenize the Basic program as it prints, and that the spooling process in general does not interfere with the Mupet's polling of the bus feature. In operation, one line is printed out, a poll is taken of all computers, then the next line is printed. If access to the bus is requested, the spooling is stopped till activity is no longer required, then the spooling is continued.

The Mupet II Controller has a built–in diagnostic routine that shows, through the use of a series of LEDs, if any errors were encountered on system power–up. The manual has a section which deals with trouble–shooting if so required. On power up, the system checks to see if a diskette is in drive 0. There does not have to be, but if so then one file, "MUPET II.SETUP", is checked for. If you do not want to use the Mupet II Controller's system defaults, then this custom file should be used. Whenever you re–set up your own parameters, it's best if you also write it to disk for future use. One other file that should be on disk if you intend to spool Basic programs to a printer is called "KEYWORD". As the name implies, it contains a list of Basic keywords and their token values. With Commodore's ever–present habit of changing token values for keywords, and inventing new and improved keywords, there should be a few versions of this file around. The system comes with a keyword file on disk when you buy it.

Special commands for special favours are sent to device #15, the Mupet II Controller. To spool a file from disk, as mentioned prior, the technique is simple: OPEN 1,15,4," name " : CLOSE 1 . To set up the Controller to non–default parameters, a string is created of the parameter settings in a special pre–defined order, then they are sent as OPEN 15,15,15,A$: CLOSE 15 . . . where A$ holds the parameter data. In order to set up the Controller correctly, a thorough knowledge of their manual must be attained. It's too complex to cover here, so this is where talking to the Controller ends.

Final Note

The systems described above, although developed and manufactured in Canada, are presently being sold throughout the world. Although I have tried my best to keep to the facts with each manufacturers system, it was difficult not to interject with personel observations once in a while. The run down of each system though, was based on literature and manuals supplied, therefore the systems should operate as stated. If you feel that networking is the answer to your long–awaited dreams, my suggestion is to drop each company a letter asking for a little more information and prices. Although the information may be redundant, pricing is always valuable, and could be used to help you convince the powers above that networking is a cost effective method to battle decreasing budgets. Hope you win.

Helping to Communicate – The TDD Network

Jim Grubbs
Springfield, Illinois

Dr. Robert Weitbrecht (1921-1983) saw the potential for using radio teletype techniques to allow the hearing impaired to communicate. . .

The Commodore VIC–20 and C–64 are responsible for many people being introduced to the fun and excitement of personal computing. Most of you are familiar with COMPUSERVE, the SOURCE and other data bases that can be accessed using an inexpensive modem. There are many bulletin boards, both for special interest groups and general discussions. One area of telecomputing seems to have been overlooked by many of us. It is in this area that your VIC or 64 can perform a very useful service. I am talking about the TDD network, or Telecommunications Device for the Deaf.

In the 1960s, an active amateur radio operator, Doctor Robert Weitbrecht, whose amateur radio call sign was W6NRM saw the potential for using amateur radio teletype techniques (abbreviated RTTY) along with surplus teletypewriter machines to allow hearing impaired individuals to communicate by telephone. Weitbrecht, although deaf himself, became interested in amateur radio as a teenager. In 1948 he was quite taken by an article he had read about radio teletype and set out to assemble a functioning station. Ultimately he became a pioneer of this mode of transmission, communicating internationally with friends in Japan, the Phillipines, Australia and several South American countries.

It was in 1963 that Doctor Weitbrecht's proposal for a TDD system became known to several individuals in the deaf community. With their encouragement and support, he refined the acoustic modem and developed an audio frequency shift keying technique specifically designed to overcome problems involved in telephone data transmission.

By mid 1964 Weitbrecht caught the attention of educators and successfully demonstrated a functioning teletypewriter system for the deaf. Finally, in 1968, AT&T agreed to release surplus machines for the project. Weitbrecht had by now perfected the modem that to this day carries his name. In recognition of the unique nature of his system he was awarded several patents.

Few of us knew in the sixties that the eighties would bring us such low cost and versatile machines as the Commodore 64 and VIC–20. I'm sure that if Doctor Weitbrecht were alive today, he would be a proponent of telecomputing. In mid 1983, however, he was hit by an automobile and rendered comatose. He died on May 30, 1983 at the age of 63.

The TDD system developed by Weitbrecht will live for a long time to come. Although your computer does not speak the TDD language, it is a quick learner and can with the help of some simple software be made to converse in this universal language. Some additional hardware will be necessary, but if you are moderately adept at reading a simple schematic diagram and have used a soldering iron, the hardware should present no problem.

The Parameters of TDD

The TDD code is based on standard Baudot signaling employed in early teletype machines and still in use by amateurs and developing nations around the world. In structure it consists of only five bits compared to the usual eight bits found in ASCII code. The major difference is that with only five bits to work with, only 32 distinct combinations can be created. This barely gives us enough to cover the alphabet. By employing a little trick and designating one of our 32 codes as a "shift" character we can almost double the number of characters. This gives us just enought room for all 26 letters, the ten cardinal numbers and some select punctuation marks.

The Commodore RS–232 port, the place where your regular modem normally connects, is very versatile. By properly addressing this port through software we can exercise precise control of many of the factors involved in data communications. The only thing not built into our Commodore machines is the Baudot code. Your computer speaks ASCII only. A little bit of BASIC to the rescue and we can fool our machines into

translating from ASCII to Baudot and back again.

Another problem presents itself in that the speed used for the TDD network and amateur radio communications is the "non-standard" 45.45 baud (so called sixty words-per-minute) and is not implemented on the VIC and 64. Although it takes a few POKEs and some calculations, it is possible to implement virtually any speed with your machine.

Unlike normal computer communications which are most frequently done in a full duplex mode, the TDD system requires that half-duplex be used. Full duplex implies that both parties may both send and receive at the same time. This has some hardware implications as well, which I'll discuss in part two. Half-duplex transmissions allow only one party to send while the other receives. Our software includes the capability to switch from transmit to receive and back at the touch of the F1 function key. Much of the protocol used in the TDD network originated with amateur radio operators. One abbreviation in wide use is GA, the old telegrapher's abbreviation for go ahead. It is customary when you are finished typing and want the other party to respond to send GA and then wait for the response. The software included both sends the GA and toggles the program back to receive for you with the touch of the F1 key.

The program also provides for sending some pre-programmed messages of your choosing by simply pressing the other special function keys (F3, F5, F7) while you are in transmit mode. Substitute your own messages in line 360, 370 and 380.

It is possible to use commercially available software sold for use on the amateur bands, although it will not specifically be designed for use in a TDD system. The hardware is quite a different story.

When Doctor Weitbrecht was developing the TDD system, the Bell System was reluctant to release information to him on how they accomplished data transmission using audio tones. Left to his own imagination and talents, Weitbrecht carefully developed his transmission system using tones most suited to the telephone technology of the time. His system calls for two tones, one at 1400 hertz and another at 1800 hertz. Standard modem tones consists of two pairs of tones, 1270 and 1070 hertz, and 2225 and 2025 hertz. As you can see TDD tones fall just about in the middle of the voice frequency band that a regular telephone circuit normally carries. The Weitbrecht modem using these tones does a fine job. The problem is that until recently it was very difficult if not impossible to find a modem designed to interface to personal computers. At least one is now available, but it costs approximately $150. Another model provides both the Weitbrecht TDD system and standard data tones. This one retails for about $300. One wonders why a person would spend this kind of money for a modem only, when small portable TDDs are available in the same price range.

Fortunately, some simple construction will give us what we need in the way of a modulator and demodulator. One more trick with a relatively inexpensive telephone amplifier available from your favorite corner electronics store, and you are connected to the telephone line and ready to communicate with others in the TDD network.

In later years Doctor Weitbrecht was criticized for staying with what some consider an antiquated system. Why not convert everyone to ASCII? Weitbrecht's thinking was that there were loads of surplus Baudot machines available for minimal costs. ASCII machines were and still are very expensive. The present situation is that neither the deaf community nor supporters of the TDD system, such as the Bell System who provide directory assistance and other operator services via TDD, have any desire to make sweeping changes. Certainly there are many deaf persons who have an interest in telecomputing and have already discovered that they can do TDD type things using ASCII communications. There is at least one national bulletin board for the handicapped. "HEX" is the "Handicapped Exchange" and operates in standard computer bulletin board fashion at 300 baud.

The telephone number is 301 593-7033.

HEX is run by a group called AMRAD, the Amateur Radio Research and Development Corporation based in the Washington, DC area. Yet another amateur, Dick Barth, (W3HWN) is the system operator. If you are a COMPUSERVE subscriber, the Clark School for the Deaf provides an online system to support deaf persons. Type go SCD at any COMPUSERVE prompt to access this service. My thanks to Dave Manning of CSD for his help in researching my material on TDD.

Telecommunications for the Deaf, Incorporated, is a non profit organization established in 1968 to provide a nationwide directory of TDD services and users and to assist with the distribution of donated surplus machines. My thanks to Doctor H. Latham Breunig, co-founder of TDD and TDI's original chief executive officer, and a very special thanks to Barry Strassler, currently executive director of TDI for their personal insights into Doctor Weitbrecht. More information on TDI is available from; Telecommunications for the Deaf, Inc., 814 Thayer Avenue, Silver Spring, Maryland 20910.

Take some time and enter the TDD program I have included with this article. Note the minor changes indicated in lines 110 & 130 for the C-64. Next time, details on building a modem for TDD communications and getting all of this connected to the phone line, along with what kinds of services are available for deaf persons on the TDD network.

Editors Note: Good news - Part 2 of Jims' article has been printed following the program on the next page.

The TDD Communications Program

```
BC  100 close 2:open 2,2,0,chr$(96+1)+chr$(0)
OG  110 rem c-64 only:poke 53280,0:poke 53281,0
        :print" e "
IL  120 print " Sr tdd term 60 wpm baudot"
        :print " rrxR ";
FJ  130 poke 665,236:poke 666,87:d=37136
        :rem if c-64 then d=56577
CP  140 ls=-1
OI  150 lf$=chr$(10)
NK  160 cr$=chr$(13)
GG  170 l$=" e"+lf$+" a siu"+cr$
        +" drjnfcktzlwhypqobg*mxv* "
CB  180 f$=" 3"+lf$+"-'87"+cr$+
        "$4',!:(5')2#6019?&*./;* "
PH  190 l1$=" cyniamztfkor\lxvwjepg↑s]uq "
HB  200 f1$=" mdtidzqorddlc\]vwsajpugfxn↑dddyd "
KE  210 get#2,c$:if c$="" then 270
ED  220 c=asc(c$):if c<1 or c>31 then 210
OD  230 if ls then c$=mid$(l$,c,1)
GH  240 if not ls then c$=mid$(f$,c,1)
PG  250 if c$<>"*" then print c$;:goto 270
AH  260 ls=(c=31)
CG  270 get a$:if a$="" then 210
JH  280 if a$=" E " then poke d,96:goto 300
OL  290 goto 210
JI  300 print " rtxR ";
FB  310 get x$:if x$="" then 310
KO  320 if x$=chr$(34) then x$=chr$(39)
OO  330 ifx$=" E "thenprint" ga ";:print#2,"dzcd";
        :goto 350
HA  340 goto 360
FN  350 print" rrxR ";:gosub680:poked,100:goto 210
HE  360 if x$=" F " then m1$=" ryryryryryryryryryry "
        :goto 520
CP  370 if x$=" G " then m1$=" listing for j smith
        springfield il pse ":goto 520
IC  380 if x$=" H " then m1$=" thank you and good
        evening ":goto 520
LI  390 print x$;
GP  400 if x$=cr$ then print#2,"h";:goto 310
CO  410 if x$=lf$ then print#2,"b";:goto 310
OD  420 if x$=chr$(32) then print#2,"d";:goto 310
EH  430 x=asc(x$)
JH  440 if x<33 then goto 310
IH  450 if x<65 then x=x-32:goto 470
KI  460 goto 480
EF  470 x$=chr$(91)+mid$(f1$,x,1):print#2,x$+chr$(95);
        :goto 310
DK  480 if x>95 then goto 310
JL  490 x=x-64:x$=mid$(l1$,x,1)
LL  500 print#2,x$;
NJ  510 goto 310
MO  520 for s=1 to len(m1$)
DG  530 x$=mid$(m1$,s,1)
BC  540 print x$;
JK  550 if x$=cr$ then print#2,"h";:goto 660
FJ  560 if x$=lf$ then print#2,"b";:goto 660
BO  570 if x$=chr$(32) then print#2,"d";:goto 660
KA  580 x=asc(x$)
IE  590 if x<33 then print#2,"d
BB  600 if x<65 then x=x-32:goto 620
CB  610 goto 630
LP  620 x$=chr$(91)+mid$(f1$,x,1):print#2,x$+chr$(95);
        :goto 660
IO  630 if x>95 then print#2,"d";
PE  640 x=x-64:x$=mid$(l1$,x,1)
BF  650 print#2,x$;
BJ  660 next s
ND  670 goto 310
OO  680 for x=1to1300:next:return
KM  710 rem jim grubbs
MP  720 rem po box 3042
GD  730 rem springfield
HN  740 rem ill 62708
```

Part Two: The Hardware

In part one, we discussed the origin of the TDD system and how to make the Commodore VIC-20 and C-64 converse in this new language. Hopefully, you have entered the TDD communications software and have it working error free. This time I'll show you how to build the special Weitbrecht modem and get it hooked up to the phone line.

There are three areas that need to be explored; Receiving incoming TDD signals, transmitting your own TDD messages, and hooking up your computer to the phone line. Finally, we will take a quick look at TDD operating procedures with a brief overview of what's available on the TDD network.

TDD Reception

Just like the data communications systems you are familiar with that use ASCII transmission, the TDD system uses audio frequency shift keying. In simple terms, the output of your computer has two states, on and off. In data communications we translate binary codes into a rapidly changing on and off, or high and low signals, with each character being sent in a very precise time period. When using ASCII transmission, the idle or off state is referred to as the mark signal. When you send nothing, the mark signal will be present. As the computer starts sending information, this signal changes rapidly between the mark tone and a space tone. The untrained ear may not hear the difference between 300 baud ASCII and our 45 baud (approximately) TDD code.

In normal data communications it is possible to have a real "two-way" talking path. This is called full duplex and is made possible through the use of two distinct sets of mark and space tones. One set is used by the originating station, the other by the answering station. That's what the switch on the VIC Modem does when you change it from answer to originate. With the TDD system, only one set of tones is used. They lie approximately midway between normal data tones. This was done to make the best use of the voice band frequency response of the telephone lines. It does impose one restriction. With both stations using the same set of tones, interference

results if both stations try to transmit at the same time. It is therefore necessary to drop your transmitting tones when you are receiving and to turn them back on when you want to transmit.

Since TDD tones are different than regular data tones, you can not use a regular modem to receive a TDD signal. A relatively simple circuit however will work for most TDD applications. It uses two inexpensive and readily available integrated circuits. At the heart of the receive circuit is a 456 phase lock loop. This device can detect the presence or absence of a particular tone. With it, we can detect the absence or presence of the space tone and send it through the RS–232 port to our computer for decoding. The only problem with this method of detection is that it is not immune to noise. Some long distance circuits may not provide a "clean" enough signal to make the decoder work reliably. The schematic for the receive decoder is included in figure one along with a complete parts list. Additionally, a printed circuit board along with some of the harder to find components is available directly from John Duke, 1441 Pleasant Drive, Dallas, Texas 75217. John may also be able to provide complete kits and assembled units. Contact him directly for additional information.

Connection of the receive unit is made to the computer through the user port. A 24 pin edge connector will be necessary to accomplish this. A source for this connector is included in the parts list. If you are not familiar with the layout of the pins on the user port connector, consult the users manual or the programmer's reference guide for details. Layout of the circuit is not critical. With a TDD signal fed to the unit, R3 should be adjusted until the LED indicator flickers in step with the signal, and with the TDD terminal software loaded, the computer prints the incoming signal. There are several ways to obtain a test signal. I'll discuss that in the final hook up section.

TDD Transmission

As simple as TDD receive was, transmission is equally as simple! A single integrated circuit will do the job. Figure two shows the schematic of the TDD modulator. An Exar Corporation XR–2206 function generator chip is used. At this writing, the XR–2206 is just being made available through Radio Shack stores. Another reliable source is included in the parts list. Once again, construction and layout are not critical. I built mine on a perf board. The mark and space frequencies are set using R7 and R8. It is recommended that you use a frequency counter for these adjustments. The modulator also connects to the user port. Although the XR–2206 should work when powered off of a 5 volt supply, I found operation at that voltage a bit unstable. You must also take into account that the power supply in the VIC can only supply a very limited amount of power. Therefore, I recommend using an external power source for the XR–2206. In my system, a standard 9 volt transistor battery does a nice job. For testing, you can connect a small earphone across the output. You should hear a tone. With the modulator hooked up to the computer and the TDD terminal software loaded, place the computer into the transmit mode by pressing the F1 special function key. As you type on the keyboard you should hear the tone shift repidly back and forth. Pressing the F1 key again sends the "GA" or go ahead prompt and returns the terminal to the receive mode.

Remember that the transmit tones must be removed from the phone line when you are receiving. The software provides a toggle signal on pin E of the user port. Feeding this signal to a simple one transistor keying circuit and connecting the circuits as shown will turn the tones on and off as you switch between transmit and receive. Make sure that you have used the proper value for "D" as indicated in line 130 of the TDD software listed in Part One.

TDD Meets The Telephone

It is necessary to get all of this hooked up to the phone line, and do it in such a manner that it will not interfere with normal use of the telephone line. This can be done with an inexpensive telephone amplifier, such as the Radio Shack #43–278. All of the actual interfacing to the phone line has been accomplished for you. By disconnecting the leads going to the speaker of the amplifier and connecting them to a simple interface (figure three), you can connect the receive demodulator. A similar connection between the modulator and the microphone connections on the amplifier will take care of the transmit side of the circuit.

I suggest trying each part of the circuit separately. For final adjustment and testing you will need a TDD signal to test the receive converter, and someone with a TDD machine to make sure that your transmitted signal is OK. In the early stages of development I used the automatic message that our local telephone company puts on the line after normal business hours on their TDD assistance line. I recorded it on cassette tape and then played the tape back into the receive converter while I adjusted it.

If you are able to adjust your transmit frequencies with a counter, you can record your own test tape, then use it to adjust your receive decoder. If all else fails, send me a blank cassette tape along with a mailer with sufficient postage and I'll make you a TDD test tape.

Other than adjusting the frequencies of the transmit modulator and setting the level on the modulator, there is nothing else to do. You just have to checkout the transmit portion "on line" with another TDD operator. If it doesn't appear to be working at all, but you are getting the proper tones out to the line, you have probably reversed the two tones when you adjusted R7 and R8. Simply retune them and you should be in business.

You connect the amplifier in a normal fashion to the phone line. Just follow the instructions that come with the amplifier for connection to the line, and the dialing instructions. Adjusting the telephone amplifier interface should be straight forward. Set the volume on the unit so the received signals will consistently print. The microphone input to the amplifier is made for a very low level signal, so set the sensitivity control on the amplifier to low, and adjust the volume on the transmit

modulator (R6) for a very low level—and I do mean low. That's the only problem I experienced in designing this unit. I mistakenly kept increasing the transmit level when it didn't seem to be working properly. My thanks to Rick Myers and Mike Apsey of Journal/20 for the basic demodulator circuit (originally designed for morse code demodulation), and John Spaulding of the HEX organization for the idea of using a telephone amplifier as a connection device for the phone line.

Using The TDD

Once you have the system operational, you will be able to communicate with anyone equipped with another TDD unit. Through a cooperative effort, the telephone companies provide a toll free directory assistance line for deaf TDD users. You reach the TDD operator at 800 855–1155. This same number will allow you to place operator assisted TDD calls so that you may make credit card calls, person to person calls, or other calls needing operator assistance. When the operator comes on the line, your screen should print "OPR MAY I HELP U Q GA", which says, "this is the operator, may I help you, (Q for question), GA (for go ahead)." At this point you go into the transmit portion and give the operator your request. Remember to always use area codes, the TDD operator won't know which one unless you include it. The phone company also provides a service number for you to use in reporting trouble with your telephone line, or for ordering additonal services via TDD. Check the front of your phone book for this number.

There are many other services available via TDD. Many government agencies, including Federal, State, and local offices have TDD machines. Particularly in metropolitan areas, such things as AAA Motor Club and other large companies have TDD, or as they are sometimes called TTY (teletype) numbers. Check your local phone book.

The Commodore VIC–20 and C–64 are extremely versatile machines. Their low cost makes them very attractive and affordable to a wide range of individuals. The avenues of communication that they open up for handicapped individuals is phenomenal. With our TDD program and this simple interface, deaf individuals can have the advantages of a TDD device and a home computer all rolled into one. Anyone interested in a talking modem program for the visually impaired?

Author correspondence to:
Jim Grubbs
P.O. Box 3042
Springfield, IL
62708 217 753–1995

Parts List for Figure One – TDD Receive Decoder (Note: All resistors are 1/4 watt)

C1 – .47 µf tantalum capacitor
C2 – .001 µf capacitor
C3 – .47 µf tantalum capacitor
C4 – .1 µf tantalum capacitor
C5 – .01 µf capacitor
C6 – .1 µf tantalum capacitor

Q1 – 2N2222 NPN transistor
D1 – 1N4148 Diode
IC1 – 567 PLL Integrated Circuit
IC2 – 74LS00 Integrated Circuit
RL1 – 5v DC relay (Radio Shack 275–243 ok)

R1 – 100K resistor
R2 – 390 ohm resistor
R3 – 10K resistor
R4 – 10K 10 turn variable resistor
R5 – 2.2K resistor
R6 – 470 ohm resistor
R7 – 1K resistor
R8 – 4.7K resistor

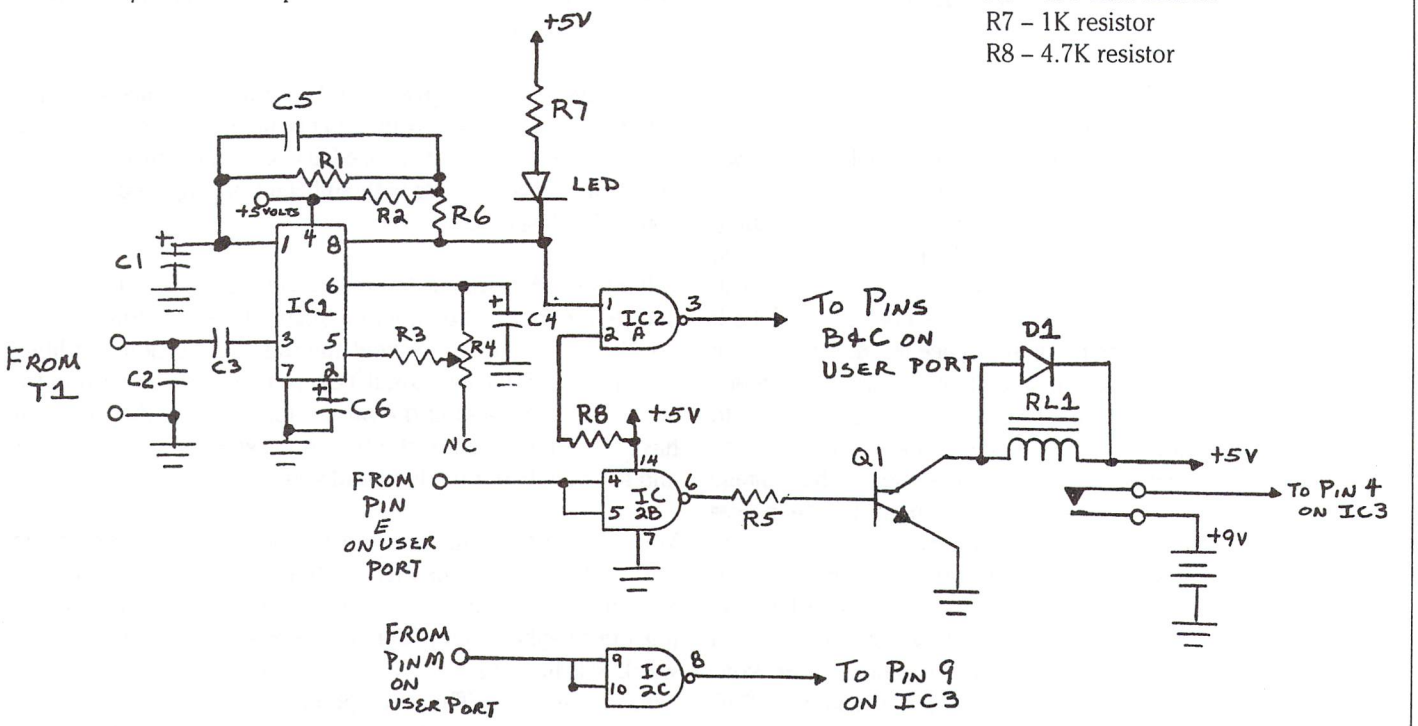


Figure One

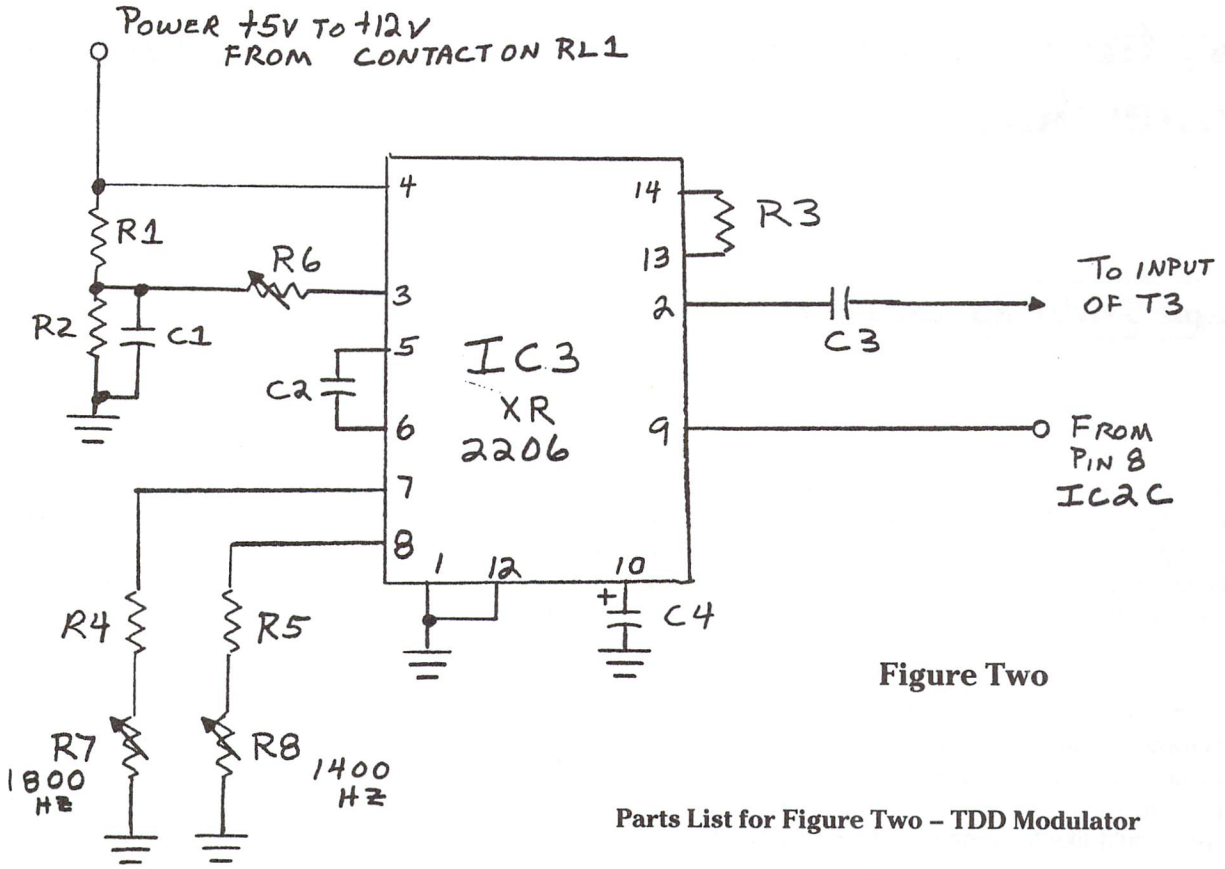


Figure Two

Parts List for Figure Two – TDD Modulator

R1, R2 – 5.1K resistor
R3 – 220 ohm resistor
R4, R5 – 7.5K resistor
R6 – 50K single turn trim pot
R7, R8 – 5K 10 turn trim pots

C1 – 10 µf 25v tantalum capacitor
C2 – 0.047 µf Mylar capacitor
C3 – 0.1 µf 50v disc capacitor
C4 – 1 µf 25v tantalum capacitor

IC3 – XR–2206 Exar integrated circuit

12/24 pin user port connector: Sullins 06SUL1224E5 available from: Priority One Electronics, 9161 Deering Avenue, Chatsworth, CA 91311. XR2206 and most other parts available from: JAMECO Electronics, 1355 Shorewy Road, Belmot, CA 94002

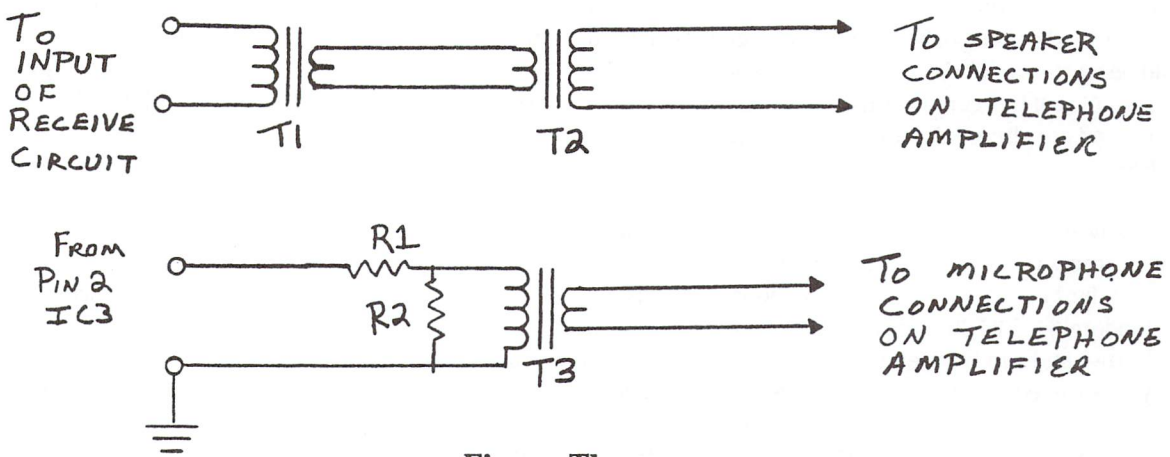


Figure Three

Parts List for Figure Three Telephone Interface

R1 – 10K resistor
R2 – 1K resistor

T1, T2, T3 – 1K to 8 ohm audio transformer Radio Shack #273–1381
1 Inexpensive telephone amplifier such as the Radio Shack #43–278

Easy Intercomputer Connection

Simon Fodale
Montreal, Quebec

A simple 3–wire RS–232 interface

This article describes how to connect together two COMMODORE 64s, VIC 20s , +4s or any combination of two of them. Implementing this connection requires only a user–built cable and some BASIC programming; with the cable in place, any kind of data communication between the machines can take place.

Today the price of a Commodore 64 or VIC 20 is very affordable and many owners of one machine are tempted to buy a second one. If you are attracted by this possibility or you already took advantage of it, I am sure that one of the most interesting projects you would like to accomplish is to connect the two machines together and run somehow in a dual processor environment.

There are actually several ways to connect two micros. The fastest and more versatile would be to use the cartridge expansion port; this would require an interface cable and a program or subroutine to handle the protocol. This protocol naturally should be designed around the signals available on the port and would be almost impossible to write in BASIC because of the speed needed to handle these signals.

Another connection could be made through the user port. This would not be as powerful as the first one and would need an interface cable and a protocol handler. This program or subroutine could be written in BASIC, but the speed would suffer a lot. Several examples of this connection can be found in computer magazines, and are almost all based on the concept of parallel data transfer.

The third way of connecting two machines is using a serial connection, known also as RS–232. This line discipline is an industry standard and can be used with various protocols. On the Commodore 64, VIC 20 and +4, RS–232 protocol is built into the Kernel and can be used with an adapter via the user port. Why do we need an adapter? The answer is that to save on cost for an option not always used, Commodore decided not to use true RS–232 signal levels, providing instead an optional extension on the user port. This means that in theory we cannot use the RS–232 line discipline without extra hardware, but actually that hardware is needed only to translate the TTL signals generated by the motherboard into the RS–232 signals accepted by some serial equipment. The cable described in this article does not need any translation, because we use all signals at their TTL level, allowing connection of the Commodore machines using this feature. In addition, it uses the simpler '3 LINE' handshaking, requiring only Ground, Received Data and Transmitted Data lines.

To build the cable, you need to get two connectors compatible with the user port. I got mine at a local electronics store and I am sure that, being a pretty standard product, they are available everywhere. Another thing you need is any three–wire cable. I used a flat cable just because it happened to be in my junk drawer; a handy telephone cable will do a fine job too.

Referring to figure 1 you can make the few connections needed. While the connections are not many, if you are not familiar with a soldering iron and small projects, I suggest you rely on a friend to avoid a possible waste of time and money. Here is a brief description of the cable functions; for complete information, page 355 of the C–64 Programmer's Reference Guide gives all the user–port pin–outs with the names of RS–232 signal lines. If you hold the connector with the protruding pins facing you, place the side with the letter 'A' on the left top. If the connector does not have letters and numbers, just assume that the leftmost top pin is 'A'. As you can observe from the connector or from the User's Guide page 143, not all letters are used; the exact lettering is A B C D E F H J K L M N.

Connect pins A and N with a wire bridge; those are both ground and their connection is a good practice. Make another bridge between pin B and C; they handle both the Received Data signal, but pin B is internally connected to a detection circuit, able to generate an interrupt when a character is coming from the line. Do the same bridges to the other connector and now you can link the two by the three–wire cable. If the wires are different colours there is no problem, else better make sure with a tester or some other means which wire is which on each end.

Connect first the ground, pin A or N, to both sides; then connect pin M on one side to pins B and C on the other side;

this is the Transmitted Data connected to the Received Data. Do the same connection for the other wire in the reverse direction. Now both connectors should look the same, with pins A, B, C, M, and N soldered somewhere. To complete a durable connection, the connectors should be put in a headshell of some type which covers the soldered connections.

I chose to make my own headshell from a piece of plastic of 1" by 3" by 1/4" (25mm by 75mm by 5mm). In figure 2 there is a sample of this home-made assembly. first I filed down one edge on both sides to fit between the top and bottom rows of connector pins. Then I drilled two holes, corresponding to those on the connector, to fit the holding screws. Next, with the width of the cable to be used in mind, I drilled a hole about 1/2" or 15mm deep into the edge opposite the connector pins. Last, with a very large drill tip and a gouge, I carefully opened a hollow into the 15mm cable hole, through which the cable will be brought to the surface of the headshell. The cable can be inserted into the hole with the wires coming out from the hollow and connected to the pins. A couple of pieces of insulating tape on each side of the headshell to cover the pins should finish the job. Do not forget to pass the wire through the hole before making the connections if you choose to make this housing.

It is very important to mark the connector with a label indicating the side that must be up; inserting the cable in the wrong position will mean connecting to +5V and +9V power sources. It is easy determine the inserting position looking at the finished connector; the side with no soldered pins is the top.

With one end of this cable plugged into the port on each of the two machines, it is now possible to transfer any kind of data directly from BASIC. The built-in RS-232 routines allow communication by simply OPENing a file to device number 2 and using GET# or PRINT# to receive or send. Using a data word length of 8 will allow you to transfer not only alphanumeric but the full Commodore character set (graphics, tokens, etc,) making it possible to transfer BASIC or machine language programs, graphics, portions of memory, etc.

To test the cable you can use your favorite terminal emulator program on both computers, or if you're using C64s or VICs, you can use this simple one:

```
10 open 2,2,2,chr$(8)
15 rem above sets: 1200 baud, 8 bit, no parity
20 get#2,a$: get b$: print a$;b$;
30 if b$<>"" then print#2,b$;
40 goto 20
```

With the above set-up, you can communicate between computers simply by typing on either keyboard.

As a practical use of this communication channel, you could set up a distributed processing system. For example, in a database system, one computer could accept input from the user while the other maintained the database on disk. That way, a record could be sorted and stored on one computer while the user is entering the next record or typing in a query on the other. This would result in major time savings in most circumstances.

Editor's Note: another good application of the RS-232 cable is to run a terminal program on one computer and REMOTE 64, from elsewhere in this issue, on the other. See the article on REMOTE 64 for details. - CZ

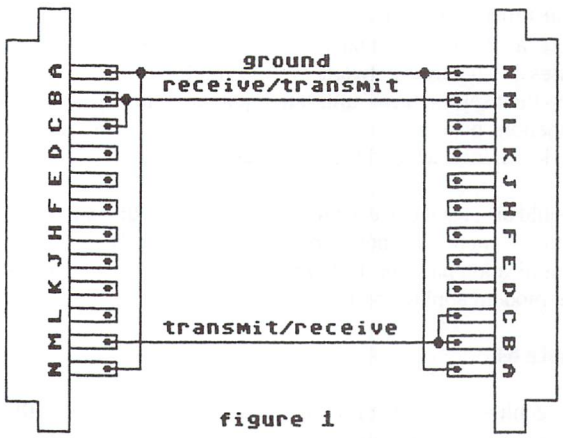
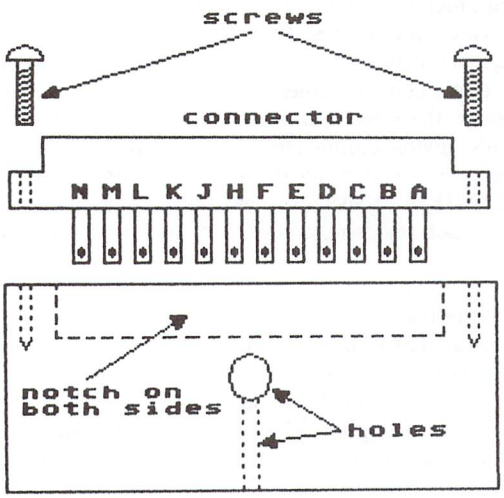


figure 1

Connectors are 24 position .156 × .200 edge connectors
Use any 3-wire cable such as telephone cable



piece of plastic
1/4 x 1 x 3

figure 2

Modems and REMOTE 64

Chris Zamara, Technical Editor

LOAD, LIST, SAVE, even edit programs. . . all from remote control!

The Modem

There are two things that can expand your computer limitlessly. One is software. The other is a modem. Both of these things have unlimited potential, since they take the computer and transform it into. . . well, anything you like. Just as a stereo system's enjoyment comes not from what the system itself does, but from the music that is played on it, a computer that "plays" a piece of software becomes just a medium for the programmer or artist to use. Similarly, a computer with a modem, acting as a terminal, is just a medium through which the user accesses his favorite network or BBS (bulletin board system). Networks themselves have unlimited potential for communications, information, business transactions; a whole new way for people to work, learn, interact, and communicate.

his would be a good place to talk about networks, BBSs, what they are, how to use them, and how they're changing man's work habits and communication patterns. But to get to the topic at hand, let's get on to a neat modem application that might even be useful to some of you.

Remote 64

The machine-language program presented here is called "REMOTE 64", and as its name implies, it allows you to use your 64 (equipped with an RS-232 modem) from a terminal somewhere else in the world. This is the idea: you call your home from a terminal, which could be another 64 running a terminal emulator. Your modem at home answers the phone – now you're in complete control of your 64. Whatever you type on the terminal appears on the screen of the 64 (specifically, it enters the 64's keyboard buffer). Whatever the 64 prints on its screen gets sent to the terminal. This means that you can do anything from the terminal that you could do on the 64, including LOADing programs, editing, running them and seeing the results, even playing some games. Of course, you *won't* be able to see anything that is POKEd directly into screen memory, or any hardware stuff like that. But many games, such as adventure games, only print characters to the screen normally, and can be used from remote. Technically, anything printed through the CHROUT routine at $FFD2 will be sent to the modem. Incidentally, the 64 can also be used locally while REMOTE is installed; it receives input from its keyboard AND the RS-232 port (modem), and prints to its screen AND the RS-232 port.

The remote terminal can also BREAK a program or listing in progress by sending a CTRL-C (ASCII value 3). A CTRL-C can be sent from most terminal programs by holding the CTRL key and striking "C", or by simply pressing the STOP key. After sending the STOP, be prepared to receive up to 256 characters before seeing the 64 stop, since there is a 256 byte RS-232 transmit buffer in the 64.

Applications

Using REMOTE, you could call your 64 from work and check the disk in the drive for certain files. You could load in a program and LIST it to view a programming technique you need to know. Or you could load a BASIC program and make a change as a result of a spontaneous brainstorm which you'd surely forget by the time you got home. REMOTE also acts as a primitive BBS (but only for careful users: you don't want someone to reset the machine from the remote terminal). You could call the REMOTE-equipped 64 and put some REMs in a program for the next user to read, or write ad SAVE a program for him to see. By typing LIST, you can see a program that may have been left by someone else calling in. A person at the 64 end can communicate with anyone calling in, or he can call out to someone with a terminal. The person at the terminal end can just type messages, using SHIFT/RETURN to start a new line, to avoid causing a syntax error – these messages will be seen on the 64 as they are typed, character by character. The person at the 64 end can just type **?"** and then the message he wishes to send, followed by RETURN – these messages will be sent out to the modem after RETURN is pressed. The advantage to communicating this way as opposed to a normal terminal program is that programming ideas can be transmitted: the terminal user can enter programs and RUN them to show the 64 user his idea.

Since REMOTE installs itself unobtrusively in the system (it lives at $C000 and works through the output and IRQ vectors), BASIC and even machine language can still be run with it in place. BBS operators might want to put an option in the BBS program, "Escape to system". This option would be restricted only to certain users, in fact probably only to the SYSOP (SYStem OPerator) himself. Choosing the option would exit from the BBS and install REMOTE. Suppose you're the SYSOP. The "ESCAPE" option would allow you to modify the BBS program, scratch certain files, validate the disk, etc., while you're away, perhaps on vacation (well, a working vacation, anyway). If the error vector in the 64 (location $0300-$0301) was changed so that an error would initialize REMOTE before continuing with the error process, then you could repair the BBS and re-run it if it dies for any reason. (They do crash sometimes – it seems that some "pirates" have figured out ways to confuse certain BBS programs.) Of course there's no way to change disks from remote, so you'll have to come back from vacation sometime. (Maybe some day in the future we'll publish an article on a do-it-yourself robot. Then you could make the 64 say "insert disk x", and he'd do it. Of course, he'd also have to understand "get me a beer" or maybe even "write me a program").

The one drawback that REMOTE has is that it won't allow any disk operations while it's active other than LOAD and SAVE. Due to the critical timing involved in both RS-232 and serial communications, there's just no way to have them both going on at the same time. Well, no easy way, anyway – any ideas out there? If you do need to do some kind of disk access while remote is running, you can temporarily disable part of remote by fixing up the IRQ vector:

```
POKE 781,12 : SYS 64701
```

After the disk operation, you can re-enable remote with SYS 49152.

Usage and Notes

After you put remote into memory by running the basic loader in listing 1, initialize it with SYS 49152. At this point, any characters received over the RS-232 port (probably your modem) will appear on the 64's screen as if typed in from the keyboard. All output will go to the RS-232 port.

Remote is set up to communicate at 300 baud with no parity; the RS-232 parameter used are the values 6 and 16. These are the numbers supplied when remote opens the RS-232 file, using a logical file number of 100. In other words, when remote opens the RS-232 file, it performs the equivalent of:

```
OPEN 100,2,2,chr$(6)+chr$(16)
```

You may change these parameters by altering the fourth and fifth bytes in the BASIC loader (listing 1).

To test out remote, connect two 64s via modem or user port connection, and use the following simple terminal program on the remote 64 – that's the 64 *not* running the remote program.

```
10 open 1,2,2,chr$(6)+chr$(16)
20 get a$: get#1,b$
30 if a$<>" "then print#1,a$;: print a$;
40 if b$<>" "then print b$;
50 goto 20
```

While running this program, you will be able to control the other 64 by "remote control". To simulate the STOP key, just hold down CTRL while pressing C.

How it works

Initializing REMOTE with SYS 49152 opens an RS-232 file (file #2, control and command bytes 6 and 16 respectively), and changes the output vector at $0326/7 and the IRQ vector at $0314/5. It also changes the load and save links at $0330-0333, the abort i/o vector at $$032C/D, and the test-STOP vector at $0328/9. (For more on vectors see the article in this issue).

The new IRQ routine gets a character from the RS-232 file. If it's not a null, it puts it in the keyboard buffer at $0277 and increments the pointer at $00C6 (198 decimal). It then jumps to the system IRQ routine at $EA31, which scans the keyboard.

The new output routine sends the output character to the screen AND to the RS-232 file, then returns back to the calling routine.

The "abort i/o" vector is trapped to keep the RS232 file open when a program line is entered or BASIC wishes to close all files for any reason.

The load and save link vectors are used to restore the IRQ and output vectors during LOADs and SAVEs and restore them after the load or save is complete. This is necessary to prevent death of the disk I/O routines due to timing delays introduced by the new vector-driven code.

So if you have a modem and are exploring the limitless world of telecommunications, you now have the option of computing by remote-control.

Listing 1: BASIC loader for REMOTE.
Just RUN this, then SYS 49152 to activate REMOTE 64

```
HJ 10 rem* data loader for "remote" *
LI 20 cs=0
PF 30 for i=49152 to 49434:read a:poke i,a
DH 40 cs=cs+a:next i
JH 60 if cs<>33357 then print"***** data error *****": end
EI 70 rem sys 49152
AF 80 end
```

```
DI 1000 data  76,   5, 192,   6,  16, 162,  27, 160
DH 1010 data 193,  56,  32, 141, 255,  32, 231, 255
BI 1020 data  32, 157, 192, 120, 169, 185, 141,  40
MO 1030 data   3, 169, 192, 141,  41,   3, 169, 200
BE 1040 data 141,  48,   3, 169, 192, 141,  49,   3
DJ 1050 data 169, 209, 141,  50,   3, 169, 192, 141
PL 1060 data  51,   3, 169,  22, 141,  44,   3, 169
BH 1070 data 193, 141,  45,   3, 120, 169,  83, 141
OG 1080 data  38,   3, 169, 192, 141,  39,   3, 169
OC 1090 data 113, 141,  20,   3, 169, 192, 141,  21
FF 1100 data   3,  88,  96, 120, 133, 251, 138,  72
CO 1110 data 152,  72, 165, 251,  32,  22, 231, 162
AI 1120 data 100,  32, 201, 255, 165, 251,  32, 202
CI 1130 data 241, 104, 168, 104, 170, 165, 251,  88
GB 1140 data  96, 162, 100,  32, 198, 255,  32, 228
MD 1150 data 255, 201,   0, 240,  24, 201,   3, 208
LA 1160 data   8, 169, 127, 141, 184, 192,  76, 149
NL 1170 data 192, 162,   0, 142, 184, 192, 166, 198
FG 1180 data 157, 119,   2, 230, 198, 162,   0,  32
MG 1190 data 198, 255, 108,  27, 193, 162,   3, 160
FD 1200 data 192, 169,   2,  32, 189, 255, 169, 100
MP 1210 data 162,   2, 160,   2,  32, 186, 255,  32
KB 1220 data 192, 255, 162, 100,  32, 201, 255,  96
NL 1230 data   0, 173, 184, 192, 240,   7, 133, 145
OD 1240 data 169,   0, 141, 184, 192, 108,  47, 193
PG 1250 data   8,  72, 138,  72, 162,   0,  76, 215
HC 1260 data 192,   8,  72, 138,  72, 162,   2, 152
GP 1270 data  72, 173,  27, 193, 141,  20,   3, 173
MD 1280 data  28, 193, 141,  21,   3, 173,  45, 193
JD 1290 data 141,  38,   3, 173,  46, 193, 141,  39
GD 1300 data   3, 189,  55, 193, 141,   4, 193, 189
JL 1310 data  56, 193, 141,   5, 193, 104, 168, 104
NP 1320 data 170, 104,  40,  32,   0,   0,   8,  72
MG 1330 data 138,  72, 152,  72,  32,  60, 192, 104
HG 1340 data 168, 104, 170, 104,  40,  96, 169,   1
LO 1350 data  76,  49, 243
```

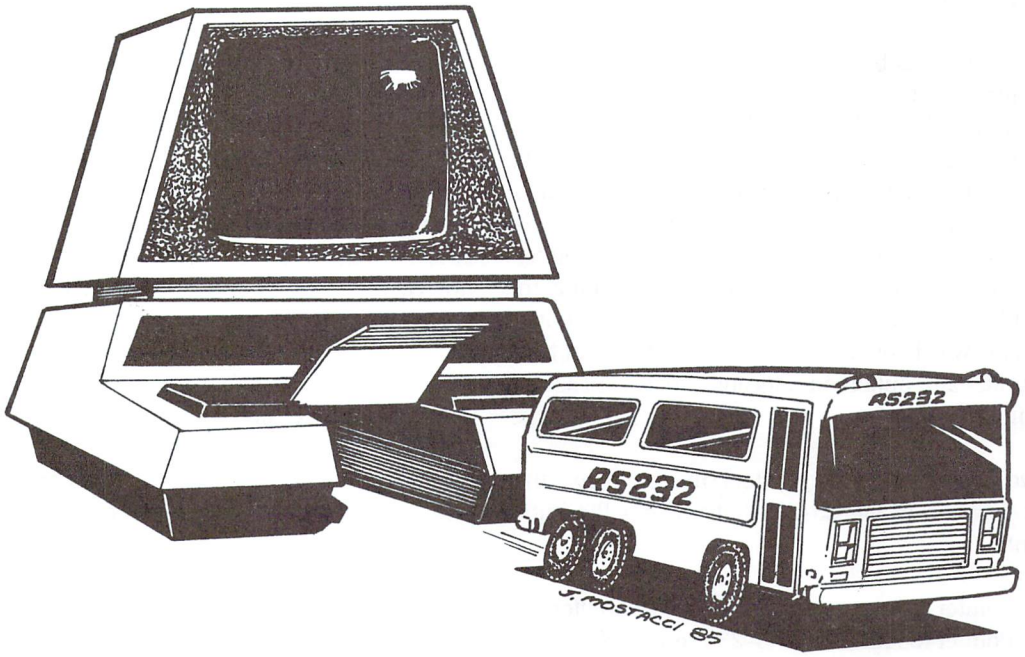
Listing 2: The PAL Source code for REMOTE 64

```
FD 100  sys700
GN 110  .opt oo
JG 120      *     =    $c000
KL 130            jmp  setup
CA 140  ;
LL 150  ;kernal entries:
IC 160  chkin     =    $ffc6
IL 170  chkout    =    $ffc9
JC 180  clall     =    $ffe7
OK 200  getin     =    $ffe4
BB 210  open      =    $ffc0
FA 220  setlfs    =    $ffba
IP 230  setnam    =    $ffbd
PO 240  vector    =    $ff8d
AH 250  ;
AG 260  scrnout   =    $e716      ;chrout for screen
EG 270  normout   =    $f1ca      ;chrout vector
OI 280  ;
MH 290  asave     =    $fb        ;temp storage for a
FC 300  bufptr    =    $c6        ;# chrs in kbd buffr
OD 320  outvec    =    $0326      ;output vector
OC 330  stopvec   =    $0328      ;check stop vector
KN 340  loadvec   =    $0330      ;load vector
NO 350  savevec   =    $0332      ;save vector
DI 370  ioabort   =    $032c
ME 380  filenum   =    100        ;rs232 file #
MP 390  ;
FC 400  fn .byte 6,16             ;rs232 ctrl/command
KB 420  ;
OC 430  setup     =    *
OC 440  ;
HO 450  ;save old vectors
KJ 460            ldx  #<vecsave
```

```
EK 470              ldy   #>vecsave
BG 480              sec
KE 490              jsr   vector
KG 500 ;
AP 510              jsr   clall         ;close all files
DC 550              jsr   rsopen        ;open rs232 file
GK 560 ;
NN 580              sei
MN 600 ;change stop vector
JH 610              lda   #<newstop
JC 620              sta   stopvec
JI 630              lda   #>newstop
JE 640              sta   stopvec+1
AA 650 ;
LB 660 ;change load vector
BM 670              lda   #<newload
IG 680              sta   loadvec
BN 690              lda   #>newload
II 700              sta   loadvec+1
MD 710 ;
IF 720 ;change save vector
BA 730              lda   #<newsave
FK 740              sta   savevec
BB 750              lda   #>newsave
FM 760              sta   savevec+1
IH 770 ;
CJ 780 ;change "abort i/o" vector
AG 790              lda   #<newio
AN 800              sta   ioabort
AH 810              lda   #>newio
GK 820              sta   ioabort+1
EL 830 ;
HI 831 aftersav     =     *
JN 832              sei
GD 840 ;change output vector
OE 850              lda   #<newout: sta outvec
AG 860              lda   #>newout: sta outvec+1
MN 870 ;
MM 880 ;change irq vector
PO 890              lda   #<intrtn: sta $0314
IP 900              lda   #>intrtn: sta $0315
GC 920              cli
OI 930              rts
CC 940 ;
BA 960 newout       =     *
GG 970 ;this is the new output routine
DP 980 ;which sends to rs232 and screen
CF 990 ;the vector at $0326 points here
OF 1000 ;
BA 1010             sta   asave
OA 1020             txa: pha: tya: pha ;save x & y!!
HN 1030             lda   asave
BJ 1040             jsr   scrnout       ;screen
DG 1050             ldx   #filenum
GO 1060             jsr   chkout
PP 1070             lda   asave
MO 1080             jsr   normout       ;rs232
KJ 1090             pla: tay: pla: tax
PD 1100             lda   asave         ;restore a
CE 1110             rts
GN 1120 ;
KN 1140 intrtn      =     *
GJ 1150 ;puts char from rs232 into
OJ 1160 ;keyboard buffer
IA 1170 ;
FO 1180             ldx   #filenum
MO 1190             jsr   chkin         ;connect rs232 channel
HG 1200             jsr   getin         ;get character
EK 1210             cmp   #0            ;null
PL 1220             beq   out           ;ignore nulls
GD 1230             cmp   #3            ;ctrl-c (break)
PJ 1240             bne   nobrk
FM 1250             lda   #$7f          ;indicate break
ED 1260             sta   brkflg        ;..to new stop rtn
GK 1270             jmp   out
PD 1280 nobrk       =     *
PH 1290             ldx   #0            ;clear stop
BC 1300             stx   brkflg        ;..flag
DO 1310             ldx   bufptr        ;# chars in buffer
FM 1320             sta   $0277,x       ;keyboard buffer
MC 1330             inc   bufptr        ;point to next char
DD 1340 out         =     *
JK 1350             ldx   #0            ;switch back to
CM 1360             jsr   chkin         ;..keyboard
HD 1370             jmp   (vecsave)     ;system irq routine
KN 1380 ;
EO 1390 ;
LJ 1400 rsopen      =     *
FA 1410 ;open rs232 file
BF 1420             ldx   #<fn          ;point to
FP 1430             ldy   #>fn          ;. . .filename
GF 1440             lda   #2            ;filename length
LI 1450             jsr   setnam        ;set filename
BK 1460             lda   #filenum
JA 1470             ldx   #2
HB 1480             ldy   #2
MJ 1490             jsr   setlfs
JE 1500             jsr   open          ;open file#,2,2,fn
PC 1510             ldx   #filenum
GB 1520             jsr   chkout        ;connect channel
GO 1530             rts
JE 1540 brkflg      .byte 0
EI 1550 ;
AI 1560 newstop     =     *
GI 1570 ;check stop key routine
NE 1580             lda   brkflg
JG 1590             beq   nostop
PA 1600             sta   $91
FD 1610             lda   #0
DL 1620             sta   brkflg
BD 1630 nostop      =     *
JD 1640             jmp   (vecsave+20)
IO 1650 ;
CP 1660 ;
HP 1670 newload     =     *
LE 1680 ;load vector points here
MK 1690 ;must disable stuff before load
GL 1700             php: pha: txa: pha
OD 1710             ldx   #0            ;0=load
BF 1730             jmp   ld
CE 1740 ;
KE 1750 newsave     =     *
CP 1760             php: pha: txa: pha
II 1770             ldx   #2            ;2=save
EH 1790 ;
AO 1800 ld          =     *
NN 1830             tya
MK 1840             pha
AL 1850 ;
FK 1860             lda   vecsave
CM 1870             sta   $0314         ;irq vector
FM 1880             lda   vecsave+1
OB 1890             sta   $0315
CO 1900 ;
LI 1910             lda   vecsave+18
PI 1920             sta   outvec        ;output vector
CK 1930             lda   vecsave+19
EB 1940             sta   outvec+1
EB 1950 ;
IM 1970             lda   vecsave+28     ;load/save adr lo
DJ 1980             sta   ldsv+1
PM 1990             lda   vecsave+29     ;load/save adr hi
JK 2000             sta   ldsv+2
CP 2010             pla:tay:pla:tax:pla:plp
JJ 2020 ldsv        jsr   *-*           ;load or save routine
HK 2030             php:pha:txa:pha:tya:pha
NL 2040             jsr   aftersav
KB 2050             pla:tay:pla:tax:pla:plp
IP 2060             rts
MI 2070 ;
FI 2080 newio       =     *
PA 2085             lda   #1            ;always keep 1 file open
LC 2090             jmp   $f331
KK 2100 ;
HC 2110 vecsave     *=*+26
```

Riding The RS–232 Bus Lines

Tony Valeri
Hamilton, Ontario



In the beginning, Commodore introduced the Vic–20 along with the philosophy that computers should be affordable. In the design of the Vic (and later the C–64) Commodore decided not to include a full parallel IEEE–488 bus, but to implement a lower cost serial version. This advented the new line of inexpensive, easy to use(?), and reliable(?!?) peripherals, such as the 1541 and the 1525. Also included in the new design was an implementation of a semi–standard RS–232 interface (standard except for the fact that the voltages were wrong and the standard connector was non–existent). Before this it was a rare PET that enjoyed the luxury of telecommunications, being that IEEE modems were prohibitively expensive. With the introduction of the Vic to general consumers, a modem was only a partial paycheque away (as soon as somebody started producing them – modems, not paycheques).

One of Commodore's major strengths in the market place is that the availability of every item used in the manufacture of their products is virtually guaranteed. This may mean that they own the chip manufacturing company (MOS technology), or just that they have a binding contract with suppliers. So, when an RS–232 port was being designed into the Vic–20, a dilemma cropped up. It seems that at this time UARTs (Universal Asynchronous Receiver Transmitter) were in short supply and, rather that risk production delays waiting for more to arrive, they decided to fake it by re–writing the operating system to perform the UART's duties. This has lead to a few minor programming problems that I'll cover a little later on. But for now let's dig into the meat and potatoes of the RS–232 bus line.

Opening an RS–232 channel.

Open lf,2,sa,CHR$(a)CHR$(b)CHR$(c)CHR$(d)

lf – Logical file number. This number must be in the range of 0–255 and is the operating systems way of distinguishing between files.

sa –Secondary address. The same restrictions apply as in the above but in this case it's not really used for anything (but put it in anyway).

a – Control Register. (see fig 1.1) This number controls the number of stop bits, word length, and the parity of the RS–232 channel.

b – Command Register. (see fig 1.2) This number controls the parity, duplex, and the handshaking of the RS–232 channel, and is optional.

c/d – These numbers are needed only if the leftmost four bits of the control register are zero (signifying a user designated baud rate). (see fig 1.3)

Using fig's 1.1 , 1.2 , and if necessary 1.3 , compute the values in binary, convert to decimal and then insert into the appropriate CHR$ statement.

This open command allows the user to decide the exact manner and speed in which the data transfer is to take place. If you wish to open an RS-232 channel from machine language, just follow the normal open procedure for a file from m/l and set the filename routine to point at the one to four characters that are to be used for control purposes.

When an RS-232 file has been opened, the operating system immediately allocates two 256 byte buffers for storage of incoming and outgoing data. These buffers are placed adjacent to the top of basic memory, overwriting any previous data. When used from within a basic program the open statement for the RS-232 file should occur before the defining of any variables or arrays. This will avoid the problem of having the buffers overwrite the variable storage area and thus bombing the program. Machine language programmers are also cautioned to keep a careful watch over the locations of these buffers. Always put the RS-232 buffers in a location that will NOT be used for anything else.

There are two main two-byte pointers in zero page that determine the location of the RS-232 buffers after the initial open statement.

$F7-$F8 Pointer to location of RS-232 receive buffer.
$F9-$FA Pointer to location of RS-232 send buffer.

Once an RS-232 file has been opened and the buffers have been created, the two buffers may be moved around at will. Moving either buffer becomes a simple matter of changing the value stored in the pointers to whatever your little heart (or picky program) desires. Another word of caution; when moving a buffer, any data in the old buffer will become information non-gratis (or as Orwell would say it becomes an Undata. ie Pfffsstt BANG! gone bye bye). So either make sure that the buffer is empty prior to a move, move the data to the new buffer yourself, or kiss it goodbye.

There are also four single byte pointers which control positioning within the two buffers. These pointers determine which location a data byte is to be written to and which location a data byte is to be taken from.

Input Buffer

$029B Where to store the next character from the RS-232 channel
$029C Where in buffer to get next character (for GET#, and INPUT#)

Output Buffer

$029D Where in buffer to get next char to send to RS-232.
$029E Where in buffer to put next character (from PRINT#)

Say, for example, that we have a C-64 and modem and are running a simple terminal program. When the operating system senses a character is about to come in over the RS-232 port (kinda like ESP) it looks at the pointer in $029B, takes that value, adds it to the pointer at $F7, and comes up with a memory location in which to store that character. The same thing happens when a character is being sent from the program (or keyboard) to the modem, and when reading characters from the buffer. Each pointer keeps track of the next available memory location in their respective buffers. When the two pointers for a particular buffer are equal that signifies either that a) the buffer is completely empty, or b) that the buffer was completely filled and the one pointer overtook the other. In effect wiping out whatever was still there (it IS still there but it's not easy for a program to take that kind of situation into account).

Most modems allow features such as automatic dialing and automatic answering capabilities. These are implemented using three of the lines of the RS-232 connector (a nice way of saying 'user port'). These three lines are The Ring Indicator line, The Carrier Detect line, and the Phone Hook Line.

On the C-64, RS-232 operations are controlled through the CIA2 chip at location $DD00. The data register can be found at location $DD01 (56577). This register is where the status of an I/O line can be read or changed, according the the information present in the data direction register ($DD03-56579). The Data register and the Data Direction register each contain eight bits. The purpose of each bit in the Dr is to reflect or change the status of an individual I/O line (as in the user port). Each bit in the DDr corresponds to a bit in the Dr, and is present for the sole purpose of determining whether the Dr will read the status of an I/O line or will change the status of that line.

When a particular bit in the Data Direction Reg is set to a '1', then the corresponding bit in the Data Reg is set to WRITE data to an I/O line. When the same bit in the Direction Reg is set to a '0', then the corresponding bit in the Data Reg is changed so as to reflect the current status of that particular I/O line (1 = write, 0 = read. "NOT I/O" is a helpful reminder)

Ring Detection

The fourth bit of the Data Reg (PB3-loc $DD01) is the ring indicator. When a ring is detected this bit changes from a zero to a one. For Example. . .

```
10 poke 56579,38: rem 00100110
20 if peek (56577) and 8 = 0 then 20
30 rem body of program
```

10-Initial I/O lines for standard operation.
20-Loop until ring is detected
30-Upon ring detection execute program

Answering the Phone/Carrier Detection

```
00 poke 56579,38: rem 00100110
20 if peek(56577) and 8 = 0 then 20
30 poke 56577,peek(56577) or 32
40 ti = 0: rem set timer to zero
50 if peek(56577) and 16 = 1 then 100
60 if ti<1000 then 50
70 poke 56577,peek(56577) and 32
80 goto 10
100 rem  main program
```

10–Initialize I/O lines
20–Wait for ring
30–When ring detected make connection (pick up phone)
40–Clear timer
50–Check for a carrier
60–Has time run out?
70–YES! hang up.
100–carrier found. enter main program

Auto Dialing

Virtually all modems designed for the 64 use the PULSE method of dialing. Dialing a particular number is accomplished by opening and closing a switch a specific number of times and with a timed delay. Tone dialing is different in as much as the actual dialing is accomplished through the sending of tones across the line instead of opening and closing a switch

In pulse dialing the pulse consists of an ON period (making the connection) and an OFF period (breaking the connection)). Each pulse is exactly 100 milliseconds long (1/10 sec) and has a make/break (on/off) ratio of 60/40. In a single pulse connection is made for 60 milliseconds then the connection is broken for 40 milliseconds. When a digit has been dialed (each digit is made up of the proper number of pulses 1 = 1 pulse,2 = 2pulses,....etc...0 = 10 pulses) there is a pause of approximately 700 ms (7/10 sec) (actually it can vary from between 1/2 sec to over 1 sec, and most exchanges will allow up to 10 seconds) until the next digit is sent.

To dial the area code (312), the hook bit in the Dr would be controlled as shown in the diagram below. (1 = make contact, 0 = break contact)

Touch Tone Dialing

The new 1660 modem from Commodore has an Audio In jack which allows touch tone dialing (through program control). This could also be accomplished with a 1650 (or compatible type) modem. All that would be needed would be a patch between the hook relay (inside the modem) and the user port connector. Then you just wire up a connector to the audio output from your 64 and away you go (if this doesn't make COMPLETE sense to you then talk it over with someone familiar with both Ma Bell and electronics. If you make a mistake you could be liable for any damage to the phone lines).

When touch tone dialing each DIGIT must last for a minimum of 100 ms (1/10 sec). The tone for each digit is a combination of two separate and distinct tones. This allows the representation of 12 digits with only 7 different tones.

	1209 hz	1336 hz	1477 hz
697 hz	1	2	3
770 hz	4	5	6
852 hz	7	8	9
941 hz	*	0	#

Thus the digit 1 consists of two tones; one at 697hz and another at 1209hz

and so on and so forth. . . ad nauseum..

Thanks to the SID chip, playing these tones is a simple matter of setting up the right frequencies. There should be two voices used. One for the row tones and one for the column tones.

Some of the autodial modems currently on the market have a problem in distinguishing between a busy signal and a carrier. This can be quite frustrating, especially if your software won't take this problem into account. Don't despair, remember the saying "If you want something done right, do it yourself". The problem arises from the modem manufactures attempt to make his product more sensitive (or to cut his production costs) and allow for bad phone lines, poor carrier quality, etc. . . The easiest way to get around this is to understand the way that Ma Bell organises HER operation and then design your program to work with the system (instead of against).

```
1111111111111111000000111100000011110000001111111111111111000000111111111111111110000001111000000111
```

(3 sec - wait for dial tone) — 60ms 40ms 60ms 40ms 60ms 40ms digit '3' — 700 ms (pause) — 60ms 40ms '1' — 700 ms (pause) — 60ms 40ms 60ms 40ms digit '2' —

equals 4.91 sec

Any method of pulse dialing MUST conform to this standard in order to be acceptable to North American telephone exchanges.

The quickest fix is to wait for 3 seconds of UN–interrupted carrier. Both the ring and the busy signal are intermittent (ie. on..off..on..off..etc..). For those of you who are interested in the possibilities of completely computerized telecommunication ('Pushing The Envelope' as another high tech field says) you might want to chew over these items.

Dial Tone	350 + 440 hz continuous
Busy Signal	480 + 620 hz 0.5 sec on 0.5 sec off
Receiver Off Hook	1400+2060+2450+2600 hz 0.1 sec on 0.1 sec off
No Such Number	between 200 to 400 hz continuous tone

The above info 'could' lead you to the development of a super–computerized answering system. Somebody calls while you are out, the computer answers (via voice synthesis) and stores their message (tape, or digitized voice). The computer then tries to phone you (from a list of numbers). It would then ask whoever answers to press the '1' or '0' button according to whether you are there or not. If you are there ('1') then you would press the '2' button (touch tone systems only). At which point the computer would play the appropriate message, and if you weren't there the computer would just keep trying.

(It was just a thought. I'd be willing to bet that some of you reading this will come up with some applications that are REALLY mind bending)

Compiling RS–232 programs

BASIC terminal programs can usually handle 300 baud with little or no difficulty (as long as there are not too many operations between characters). But when attempting to communicate at 1200 baud or greater your choices become fewer and fewer. When speed becomes vital you will either write a VERY short basic program, write it in a faster language (forth, assembler, etc..), or compile the basic program. Compiling has become one of my favourites in cases where you need extra speed but not as much as you'd get from assembler. And unless you really need a terminal program running at 9600 baud, compiled BASIC should work nicely (and it's a lot easier to write).

Traditionally, compiling a BASIC program has a few pitfalls such as the fact that things like timing loops have to be worked out on a trial and error basis.

After trying to use Petspeed to compile a simple terminal program I was faced with a problem. It wouldn't work. It finally dawned on me that the compiled program set aside memory for variable storage as soon as it was executed. Then when the RS–232 buffers were opened they wrote all over the variable storage area.

I now realize that there are probably quite a few ways to overcome this problem but I'll leave them as an exercise and explain the first method I used.

When I found that I couldn't open an RS–232 channel from BASIC (after compiling) my first thought was to try a machine language routine. A few minutes later I changed one line in my BASIC program.

```
      100 open 2,2,2,chr$(6)
became 100 open 2,8,15:sys12*4096)
```

If I couldn't open an RS–232 channel then I'd open a disk channel and change it with a ml routine. The machine language routine was quite simple. It closed file number two then reopened it as an RS–232 file. But before it returned to the main program it moved the buffers to $CE00 and $CF00, out of harms way. Then it returned to the main program. The whole thing worked like a charm.

The other methods (which I haven't tried) are changing the locations of the buffers with a few pokes immediately after opening the RS–232 file or lowering the top of memory before running the program then temporarily moving it back up just before you open the RS–22 file.

Using RS–232 and the Serial Bus

When an RS–232 file is active, an interrupt is generated each time a character is about to appear over the RS–232 bus line. This type of interrupt is called NMI (Non Maskable Interrupt) and it takes precedence over any other kind of interrupt or activity. In other words, if you happen to be using either the disk drive, cassette, or printer, and a character comes in over the RS–232 bus then odds are that part of your operation (reading, writing, or printing) will be corrupted. Under normal circumstances this is not much of a problem, but in certain applications (such as a bbs program) the affect could be disastrous. If you feel that this accurately describes a problem that's been plaguing you then remember the following hints. Before using the disk drive, printer or cassette remember to close the RS–232 channel. If you don't want to lose anything that's been stored in the buffers you should save the various pointers before closing, and replace them when re–opening the RS–232 channel. This way the only information you lose is whatever tries to come in while you are printing, reading, or writing, instead of everything in both buffers.

With any luck you've found 'something' of interest to you in this article. If you have any questions or comments please feel free to drop me a note.

Tony Valeri
40 Barnesdale Avenue North
Hamilton, Ontario
L8L 6R8

Figure 1.1

Control Register

CHR$(7 6 5 4 3 2 1 0) + CHR$(7 6 5 4 3 2 1 0)

Baud

0	0	0	0	User see Fig 1.3
0	0	0	1	50
0	0	1	0	75
0	0	1	1	110
0	1	0	0	134.5
0	1	0	1	150
0	1	1	0	300
0	1	1	1	600
1	0	0	0	1200
1	0	0	1	2400
1	0	1	0	2400
1	0	1	1	3600*
1	1	0	0	4800*
1	1	0	1	7200*
1	1	1	0	9600*
1	1	1	1	19200*

* VIC/64: not implemented

RCVR Clock
VIC/64

X	Not Used

Word Length

0	0	8 Bits
0	1	7 Bits
1	0	6 Bits
1	1	5 Bits

Stop Bits

0	1 Stop Bit
1	2 Stop Bits

Figure 1.2

Command Register

7 6 5 4 3 2 1 0

VIC/64 Handshake

0	3 Line
1	X Line

VIC/64

X	X	X	Not Used

Duplex

0	Full
1	Half

Parity

X	X	0	Disabled
0	0	1	Odd
0	1	1	Even
1	0	1	Mark
1	1	1	Space

Figure 1.3

If, when OPENing an RS232 channel, the optional User Baud Rate is selected, an additional 2 CHR$ values must be specified after the Control Register CHR$ and the Command Register CHR$. ie.

OPEN LF, 2, SA, CHR$(Control Reg) CHR$(Command Reg) CHR$(X) CHR$(Y)

$X = (Z / \text{Baud Rate}/2 - 100) - (Y * 256)$
$Y = \text{INT}((Z / \text{Baud Rate}/2 - 100) / 256)$
$Z = 1.02273 \times 10^6$ (North America)
$= 0.98525 \times 10^6$ (U.K./Europe)

Data Register (Dr)

7 6 5 4 3 2 1 0 location 56577

PB0 – Received Data (IN)
PB1 – Request To Send (OUT)
PB2 – Data Terminal Ready (OUT)
PB3 – Ring Indicator (IN)
PB4 – Received Line Signal (Carrier Detect) (IN)
PB5 – Phone Hook Relay (1 = ON / 0 = OFF) (OUT)
PB6 – Clear To Send (IN)
PB7 – Data Set Ready (IN)

Data Direction Register (DDr)

7 6 5 4 3 2 1 0 location 56579

0 0 1 0 0 1 1 0 location 56579

1 = Data Out
0 = Data In

C64 BBS Link

Bob Hayes
Winnipeg, Manitoba

How many times have you misplaced important information regarding your favourite Bulletin Board System?

It's not unlikely that you will forget your password, handle or even the phone number of the BBS, especially if you have not logged on for a while. Writing the vital information on paper sometimes helps, but where does that paper go when you need it?!

I decided to create a program that would keep track of this important information and print a list whenever I needed one. The info I decided on was quite simple:

1) Name of the system
2) Phone number
3) Your Name or Handle
4) Password

The list could also be used by a SYSOP to store the Names and Phone numbers of users of the BBS. I suppose that using a database for this purpose seems a good idea, but a database can be cumbersome, particularly when it comes to printing out the list.

The program as shown in Listing 1 uses the excellent sorting routine found in the January '85 issue of The TRANSACTOR (Volume 5, Issue 04, pg.34) so if you haven't typed it in, I suggest you do so. *(In that case, the Transactor Disk #7 for this issue will have Gary Kiziak's sort included. - M.Ed.)* If you don't need an alphabetically sorted list, you can do without the sort routine; just replace lines 840 to 880 in listing 1 with the new lines in listing 2, and delete lines 890 to 910.

I tried to keep fussy programmers happy by using chr$() for the various colour and control characters. If you do not like the idea of having a nicely formatted, colour program then leave out what you don't need in lines 205 to 215.

When your program is ready, make sure the sort routine is loaded in (if required), then RUN it. The first option of the four-choice menu, "create file", is what you will need to do first. By typing in a number in response to the prompt 'APPROX SIZE:' you will have the program create the relative file in which your data will be stored. Enter the MAXIMUM number of records you think you'll ever need, not the number of records you have now!

You can now choose 2> ENTER DATA. The MAXIMUM number of characters for each field will be displayed and you will be prompted for four replies. By typing 'q' at the end to the question SURE?, you will escape back to the menu.

When modifying an entry, the RECORD # must be known, so printing out a list before editing is advised (The record numbers are listed before the entry on the printer). You will then be sent back to the ENTER DATA screen. By typing an '*' (asterisk) to a prompt, you will leave the data in that field unchanged. Entering anything other than the asterisk will change the data. Deleting a record must be done in this option by typing an '@' ('at' symbol) for all prompts. The printer section will check for the @ and omit that record when it comes time for it to be printed.

The printing routine asks for printer type and case before it goes to work. If you are using the program in listing 1, all data will be read into memory from the drive and then sorted/printed. (The name of the BBS should be in lowercase for sorting purposes.) If you have made the listing 2 modifications, data will not be sorted. Hitting a key while the printer is going will halt the printout. This is so that you can pause the printer to make adjustments at the end of a page, if necessary.

If at any time you need to break out of the program, then be sure to type CLOSE2:CLOSE15. If you break out of the printer routine, then also type PRINT#4:CLOSE4.

Listing 1

```
      1 rem c64 bbs link
      2 rem a simple database for your bbs numbers
      3 rem the program " sort64 " must be in memory at $c100
      4 :
NP    5 dim n$(500):goto200
NP    6 save " @0:64 bbs link ",8:verify " 64 bbs link ",8:stop
GM   10 hi = int(n/256):lo = n-hi*256:return
MG   20 n = rc:gosub10:print#15, " p " chr$(2 + 96)
        chr$(lo)chr$(hi)chr$(ps)
FD   25 return
HG   30 input#15,a,z$,c,d:print:printwt$;a;yl$; "   ";z$;
        gy$;c; "   ";d:return
LJ   40 print:printtab(11)gy$ " insert master disk "
HH   41 geta$:ifa$ = " " goto41
GE   42 return
MP   50 poke53265,peek(53265)and239:return
KE   60 poke53265,peek(53265)or16:return
LK  200 open15,8,15:printchr$(14)
EJ  205 cl$ = chr$(147):wt$ = chr$(5):gy$ = chr$(155)
        :cy$ = chr$(159):cu$ = chr$(145)
NA  210 rt$ = chr$(13):yl$ = chr$(158)
JC  215 poke53281,0:poke53280,6
LO  220 printcl$:printtab(14)gy$ " 64 bbs link "
        :fora = 1to40:printyl$ " - " ;:next:print
LE  225 printtab(16)wt$;cu$; " bob hayes "
        :fora = 1to40:print " - " ;:next:print
```

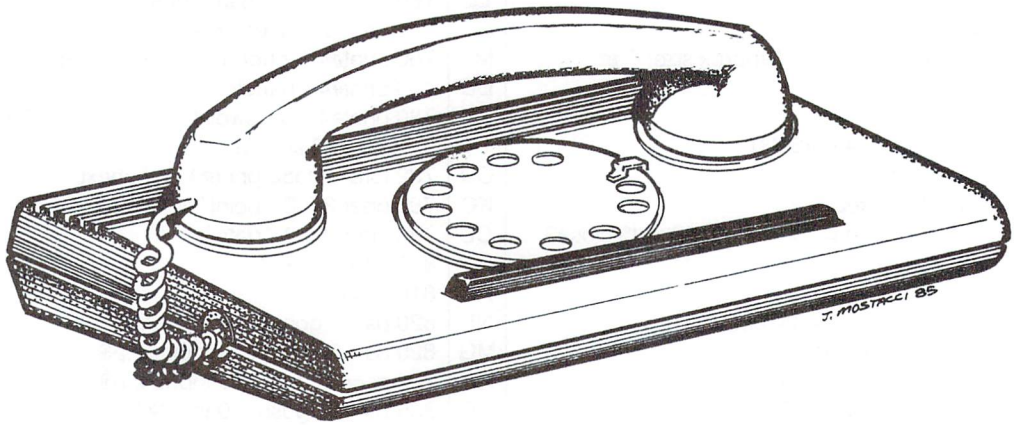
```
II 230 x=7
LE 240 printtab(x)wt$"1>  "gy$"create file"
IL 250 printtab(x)wt$"2>  "gy$"enter new record"
JJ 260 printtab(x)wt$"3>  "gy$"modify existing record"
KN 280 printtab(x)wt$"4>  "gy$"print list"wt$
KC 290 print:printtab(x);:poke19,64:input"select: ";s$
     :print:poke19,0
OC 300 s=val(s$):ifs>5ors<1goto290
NB 310 on s goto 320,400,600,700
FF 320 printcl$;cr$;:poke19,64:input"approx size:  ";sz
     :print:poke19,0
LL 325 gosub40
DL 330 open2,8,2,"0:data,l,"+chr$(77)
MC 335 rc=sz:ps=1:gosub20
MB 336 print#2,"last";rt$:close2
AL 340 open2,8,2,"@0:point,p,w":x=1:print#2,x;rt$:close2
CH 350 run
GP 400 printcl$:gosub30
LI 405 print:printwt$"name/bbs = 20 chars"
     :print"phone # = 12 chars"
OD 410 print"handle/name = 20 chars"
     :print"password = 20 chars"
KN 420 print:poke19,64:printcy$"name/bbs:"wt$;
     :input x$:print:poke19,0:rem white
OA 425 ifx$<>"*"thenb$=x$
GP 430 poke19,64:printcy$"phone #:"wt$;:input x$
     :print:poke19,0:rem cyan/white
CE 435 ifx$<>"*"thenp$=x$
MD 440 poke19,64:printcy$"handle :"wt$;:input x$
     :print:poke19,0
ED 445 ifx$<>"*"thenh$=x$
KH 450 poke19,64:printcy$"password:"wt$;:input x$
     :print::poke19,0
MC 452 ifx$<>"*"thenc$=x$
HP 455 print:print:input"sure (y/n/q)";a$:ifa$="q"thenrun
ON 460 ifa$="n"goto400
OB 465 ifqq=0thenqq=1:gosub40
LE 466 gosub50
PO 470 open2,8,2,"point":input#2,k:close2
AE 480 if r<>0 then k=r
BI 485 open2,8,2,"data"
KG 490 rc=k
EL 500 ps=1:gosub20:print#2,left$(b$,20);rt$
CP 510 ps=22:gosub20:print#2,left$(p$,12);rt$
NP 520 ps=35:gosub20:print#2,left$(h$,20);rt$
GA 530 ps=56:gosub20:print#2,left$(c$,20);rt$
CP 540 close2
EK 550 if r<>0 then gosub60:run
DF 560 k=k+1:open2,8,2,"@0:point,p,w":print#2,k;rt$
     :close2:gosub60
IN 570 goto400
NE 600 printcl$;yl$"entering a * will leave data same":print
DN 605 ch=0:printwt$;:input"input the entry # for
     changes";ch
AJ 610 ifch=0thenrun
LF 620 r=ch:rc=ch:qa=1:open2,8,2,"data":gosub820
     :close2:goto400
NE 700 printcl$"enter printer type:":print
DB 710 printgy$"a) 1525/mps 801,  b) 1526/mps 802"
BM 720 print:input"type";a$:ifa$<>"a"anda$<>"b"
     goto700
OP 725 printcl$;wt$:input"<u>pper/<l>owercase";c$
     :ifc$<>"u"andc$<>"l"goto725
JO 730 ifa$="a"andc$="l"then open4,4
     :print#4,chr$(17):goto750
KI 732 ifa$="b"andc$="l"thenopen4,4,7
     :print#4,chr$(17):goto750
MJ 734 open4,4
LE 750 print#4,"      name/bbs                    ";
     :rem 3 spaces/14 spaces
MF 760 print#4,"  phone number    ";:rem 2 spaces
DB 770 print#4,"handle                    ";:rem 15 spaces
GE 780 print#4,"password                  ";rt$
     :rem 12 spaces
OD 782 fora=1to80:print#4,"-";:next
KC 785 open2,8,2,"point":input#2,k:close2
CL 790 open2,8,2,"data"
EO 800 fora=1tok-1
AK 810 rc=a
NL 820 ps=1:gosub20:input#2,b$
MG 825 ps=22:gosub20:input#2,p$
AG 830 ps=35:gosub20:input#2,h$
KF 835 ps=56:gosub20:input#2,c$
JE 836 ifqa=1thenqa=0:return
GG 840 rem
DI 845 iflen(b$)<20thenb$=b$+".":goto845
GJ 850 iflen(p$)<12thenp$=p$+"  ":goto850
MG 855 iflen(h$)<20thenh$=h$+"  ":goto855
ID 860 iflen(c$)<20thenc$=c$+"  ":goto860
LH 861 rem
MH 862 rem
KL 863 ifa<10thenn$(a)=str$(a)+"  "+b$+".."+p$
     +"  "+h$+"  "+c$:goto870
JI 865 n$(a)=str$(a)+"  "+b$+".."+p$+"  "+h$
     +"  "+c$
KM 870 next:close2
JI 875 rem
CF 880 srt=12*4096+256
CD 890 sys(srt),n$,1,k-1,4,23,a
MG 900 fora=1tok-1:ifmid$(n$(a),5,1)="@"goto902
MD 901 print#4,n$(a)
JH 902 geta$:ifa$=""goto906
FH 904 geta$:ifa$=""goto904
OI 906 next
AC 910 print#4:close4:run
IB 920 rem *** delete lines 890- for listing 2 ***
```

Listing 2: Lines to add for non-sort version

```
LH 840 print#4,a;
BF 845 iflen(b$)<>22thenb$=b$+".":goto845
JM 850 iflen(p$)<>13thenp$=p$+"  ":goto855
OG 855 iflen(h$)<21thenh$=h$+"  ":goto855
NH 860 iflen(c$)<>20thenc$=c$+"  ":goto860
NB 861 ifleft$(b$,1)="@"goto875
ME 862 geta$:ifa$=""goto870
BF 863 geta$:ifa$=""goto862
PH 865 rem
MI 870 print#4,b$;p$;h$;c$
FO 875 next:print#4:close4:close2
BB 880 goto215
```

Tele-Tone 64:
A Synthetic Model Telephone

Richard Evers, Editor



Recently, the Commodore 64 has been making big headlines in most of the major newspapers worldwide. The Commodore 64 verses Bell Telephone. Due to a mixture of over enthusiastic programmers, and the high quality sound reproduction of the SID chip, synthetic telephone systems are appearing everywhere. These systems not only reproduce the standard touch tone dialing system, but also incorporate methods to bypassing Ma Bell, allowing entry into some pretty tender areas. Without pushing the point too far, Bell isn't too happy. People are being arrested, equipment and supplies are being seized, and the courts are starting to get pretty busy trying to sort this mess out.

Due to a very archaic law on the books in Canada, it appears that anyone caught with a "phone freak" type program in their possession can be arrested. You don't have to use it, don't even have to know you have it. It just has to be there. A pretty sad state of affairs when a vast corporation has unlimited power. In most cases, the police won't break your door down and start ripping through your possessions immediately. They have to be tipped off about any supposed illegal telephone activities before starting action. Can you say 'phone tap'. You know, those equally illegal things that governments are always doing, without getting caught. Can you imagine how simple it would be to tap a telephone line that you own and control. The shadow of Big Brother looms on the horizon.

To get back on line, this article is not about phone freaking. It's about telephone conventions and how to synthetically reproduce them using your SID chip. The sounds we will reproduce today are the ones that Bell won't arrest you for, we hope. They are the simple ones, like touch tone dialing, the ringing of the bell, a busy signal, plus a few others. All tricks learned thanks to a few commercially available books, of which the Radio Amateurs Handbook was the primary source.

To start, your telephone normally has two states of operation, the Idle state, phone on the hook, and the Busy state, phone off the hook. The only exception to this rule is when pulse type dialing systems are used. The pulse type system works in a rather simple manner. It simply pulses the line back and forth between the Idle and Busy states. These pulses occur at a rate of 10 pulses per second, with a tolerance of plus or minus 10%. Every digit dialed, 1–9, produces its own value in pulses during the cycle. The exception to this is the digit 0. It pulses 10 times. There is a minimum time between digits of 600 milliseconds, to allow the telephone system time to calculate the digit dialed. The shape of the pulse is a square wave of which the wave is in an Open state between 58% to 64% of the time. Enough said about pulse type dialing systems.

Dial tone systems are quite a bit more logical. For all digits dialed out, two frequencies are used, a high and a low. This helps the telephone system better differentiate between digits, and also cuts down inaccuracies due to noise on the line. The frequencies are as follows.

Frequencies Measured In Hz.

		High Tone		
Low	1209	1336	1477	1633
697	1	2	3	Fo
770	4	5	6	F
852	7	8	9	I
941	*	0	#	P

As the chart shows, the digit '1' can be reproduced using two frequencies, 697 and 1209 Hz. To the extreme right on the chart you will notice a column with a high frequency of 1633 Hz., represented by Fo, F, I, and P. They were not explained in

the books, but were shown all the same. These freqencies do not deviate from the normal pattern of the rest, they just have a high frequency different than the others. Without proof, I will not venture a guess as to the reason for this extra range of digits.

The frequencies listed above can have a maximum frequency deviation of plus or minus 1.5%, a pretty small margin for error. The minimum duration of ON time for the two frequency tones is 50 milliseconds, with a minimum time between digits of 45 milliseconds. There are a few other parameters to follow, but they really have no valence on this article. The program below does not have to worry about the rise time to get up to the correct amplitude of the frequency, or a few other equally useless details that would probably bother a less versatile system.

To generate a few more interesting tones, the frequencies listed below are used.

350 and 440 Hz. Dial Tone
440 and 480 Hz. Audible Ringing: 2 seconds on, 4 seconds off.
480 and 620 Hz. Line Busy: Interrupted 60 times per minute
480 and 620 Hz. Re-Order (all trunks busy): Interrupted 120 times per minute

As can be quickly deduced, these tones will never really come in handy to the average user. The dial tones shown above, as used in the program listed below, are the useful ones. With a few easy mods to the 'tele-tone 64' program, your standard modem could be quickly transformed into a pseudo auto dial unit. Through the use of a soldering iron, a bit of wire, and an old acoustic coupler speaker, the output from the SID could be sent directly to the microphone of an extra telephone on your telecommunications circuit. Not a bad trick to perform with a few old parts floating around your computer room.

To quickly finish up, the program 'Tele-Tone 64', as listed below, performs quite a few tricks that won't upset Ma Bell. It will generate all the standard tones that your phone does, plus a few extras. The extra frequencies Fo, F, I, and P are represented in the program by pressing the keys a, b, c, and d. They seemed to be a logical extension of the 0-9 set. For the Busy Signal, Re-Order Signal, and Audible Ringing, a timing loop was needed, therefore TI$ was used. To keep the program simple, TI$ was set to zero before each timed loop, and TI was checked to see when the correct duration had elapsed. If you need to use this program in an application that requires the correct setting of TI$, consider the TOD clock. Though more of a pain than TI$ to figure out, it's more accurate, powerful, and generally more satisfying.

One final point to ponder while using 'Tele-Tone 64'. The setting of the volume control is critical. Set the volume on your monitor, amplifier, or whatever, to a comfortable listening level. Too loud or too soft will result in no response from your Bell unit. Placing the microphone of the telephone approximately one half inch away from the speaker, then keying in the digits seems to work most of the time. Although this is a pretty vague way in which to work, it's the best we can do considering the limitations of communication through a magazine. Our only advice is for you to play around with the volume setting and placement of the telephone handset until it gives the desired results.

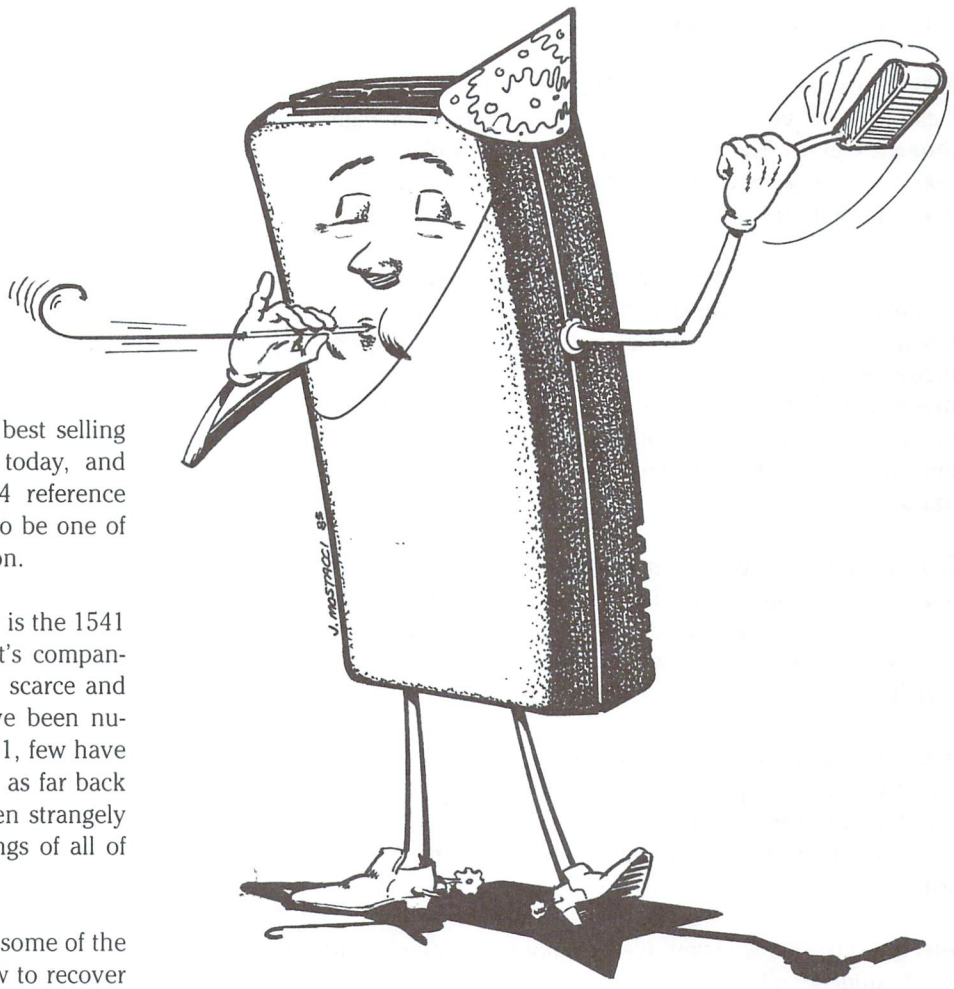
Tele-Tone 64

```
EJ  100 rem save"@0:tele-tone 64",8
DJ  105 rem ** rte/85 - synthetic touch tone sounds
        via the 64's sid chip
CO  110 :
IO  115 key=203: num=198: rem ** which key
        / # chars in keybuf
CH  120 zp$="000000": rem ** for timed loop later
BP  125 :
NM  130 dim pd$(2,3)
BE  135 pd$(0,0)="697"+"1209": rem 1
FF  140 pd$(1,0)="697"+"1336": rem 2
BH  145 pd$(2,0)="697"+"1477": rem 3
LE  150 pd$(0,1)="770"+"1209": rem 4
PF  155 pd$(1,1)="770"+"1336": rem 5
LH  160 pd$(2,1)="770"+"1477": rem 6
HG  165 pd$(0,2)="852"+"1209": rem 7
LH  170 pd$(1,2)="852"+"1336": rem 8
HJ  175 pd$(2,2)="852"+"1477": rem 9
HG  180 pd$(0,3)="941"+"1209": rem #
HI  185 pd$(1,3)="941"+"1336": rem 0
MJ  190 pd$(2,3)="941"+"1477": rem *
HD  195 :
KI  200 xt$(0)="697"+"1633": rem fo - extra
AI  205 xt$(1)="770"+"1633": rem f  - extra
EJ  210 xt$(2)="852"+"1633": rem i  - extra
LK  215 xt$(3)="941"+"1633": rem p  - extra
AF  220 :
HD  225 dt$="350"+"440": rem dial tone
DJ  230 rg$="440"+"480": rem audible ringing:
        2 seconds on, 4 seconds off
HL  235 lb$="480"+"620": rem line busy:
        interrupted at 60 int/minute
PL  240 ro$="480"+"620": rem re-order:
        interrupted at 120 int/minute
JG  245 :
AM  250 sd=54272:  rem start address of sid chip
HI  255 fc=16.40426: rem frequency multiplying
        constant
BM  260 pt=64:     rem voice type: pulse
IJ  265 for i=0 to 24: poke sd+i,0: next: rem initialise
        sid chip
IL  270 poke sd+2,0: poke sd+3,8: rem 50/50 square
        wave voice #1
```

```
IN 275 poke sd+9,0: poke sd+10,8: rem 50/50
       square wave voice #2
AG 280 poke sd+5,0:  rem attack=0: decay=0
       voice #1
NA 285 poke sd+6,240: rem sustain=infinite,
       release=immediate
DK 290 poke sd+12,0:  rem attack=0: decay=0
       voice #2
IO 295 poke sd+13,240: rem sustain=infinite,
       release=immediate
MI 300 poke sd+24,15: rem set volume to max
FK 305 :
BE 310 print chr$(147)chr$(14)"** Touch Tone 64 **"
MJ 315 print "Press 0-9  Normal Dial Tones"
EG 320 print "Press #, *  Special For Business"
MN 325 print "Press a-d  Extra Tones Fo, F, I, P"
EO 330 print "Press (B)  Line Busy"
EM 335 print "Press (D)  Dial Tone"
GH 340 print "Press (O)  Re-Order"
HE 345 print "Press (R)  Audible Ringing"
GE 350 print "Press (Shift) (Clr/Home) For Clear"
HN 355 :
PE 360 get p$: if p$="" then 360: rem get request
BO 365 :
NM 370 sid$="": spec=0: rem init sid string + set
       condition flag
LE 375 if p$<"0" or p$>"9" then 395: rem leave
       room for numeric check
AL 380 p=val(p$): if p=0 then sid$=pd$(1,3)
       : goto 450
DC 385 p=p-1: x%=p/3: y=p-3*x%
       : sid$=pd$(y,x%)
KP 390 :
BA 395 if p$="#" then sid$=pd$(0,3)
BC 400 if p$="*" then sid$=pd$(2,3)
DC 405 if p$=>"a" and p$<="d" then
       sid$=xt$(asc(p$)-65)
NI 410 if p$="B" then  sid$=lb$: spec=1: rem busy
JO 415 if p$="D" then  sid$=dt$      : rem dial tone
NK 420 if p$="O" then  sid$=ro$: spec=2
       : rem re-order
LC 425 if p$="R" then  sid$=rg$: spec=3
       : rem ringing
HJ 430 if p$=chr$(147) then 310: rem clear the display
FF 435 if len(sid$)=0 then poke num,0: goto 360
       : rem wrong key, try again
MC 440 :
CD 445 rem ** time to start touch toning **
PK 450 print p$;
DP 455 lf=val(mid$(sid$,1,3))*fc
       : hf=val(mid$(sid$,4))*fc: rem set freq/voice
MG 460 poke sd+4,pt: poke sd+11,pt: rem gate voice
       #1 and #2 off
JE 465 hi%=lf/256: lo=lf-hi%*256:  rem calc lo/hi
       freq voice #1
AJ 470 poke sd,lo: poke sd+1,hi%:  rem set up freq
       voice #1
OD 475 hi%=hf/256: lo=hf-hi%*256:  rem calc lo/hi
       freq voice #2
ML 480 poke sd+7,lo: poke sd+8,hi%: rem set up
       freq voice #2
LF 485 poke sd+4,pt+1: poke sd+11,pt+1
       : rem gate voice #1 and #2 on
GL 490 ct=0: ti$=zp$: rem set for timed loop if needed
DG 495 :
CI 500 if spec=0 then 570: rem nothing special to do
NG 505 :
LC 510 if spec<3 then 530: rem #3  = audible ringing
NO 515 if ct=0 and ti=>120 then gosub 595
       : rem flip ct, re-set ti$, gate off
GH 520 if ct and ti=>240 then gosub 600
       : rem flip ct, re-set ti$, gate on
BI 525 :
MD 530 if spec<>2 then 550: rem #2  = re-order signal
KN 535 if ct=0 and ti=>15 then gosub 595
MN 540 if ct and ti=>15 then gosub 600
FJ 545 :
IM 550 if spec>1 then 570: rem #1  = busy signal
BP 555 if ct=0 and ti=>30 then gosub 595
FO 560 if ct and ti=>30 then gosub 600
JK 565 :
JJ 570 wch=peek(key): if wch<>64 then 500
       : rem wait till no key pressed
DN 575 poke sd+4,pt: poke sd+11,pt: rem gate both
       voices off
OE 580 poke sd,0: poke sd+1,0: poke sd+7,0
       : poke sd+8,0: rem set freq's low
KL 585 goto 360: rem go for next key press
CM 590 :
HF 595 ti$=zp$: ct=not ct: poke sd+4,pt
       : poke sd+11,pt: return:   rem gate off
GA 600 ti$=zp$: ct=not ct: poke sd+4,pt+1
       : poke sd+11,pt+1: return: rem gate on
```

Fun with the 1541

Tony Valeri
Hamilton, Ontario



The Commodore 64 is one of the best selling micro-computers on the market today, and since the publication of the C-64 reference manual it can also be considered to be one of the best in the area of documentation.

The companion accessory to the 64 is the 1541 disk drive. Unfortunately, unlike it's companion, documentation has been both scarce and somewhat spotty. While there have been numerous articles concerning the 1541, few have gone beyond information available as far back as 1980. Commodore itself has been strangely silent concerning the inner workings of all of their disk drives, not just the 1541.

In this article I'll explain the use of some of the operating system routines, and how to recover from certain disk errors without losing the program or data effected.

Our first step will be an overview of the actual operation of the disk drive itself and the ways in which we can manipulate it. Typically the owner of a C-64 system actually has TWO computers. This is not to say that they also own another brand of computer but that the 1541 is actually a full-fledged stand alone computer all by itself. The 1541 uses a 6502 microprocessor, has two kilo-bytes of Random Access Memory (RAM), sixteen kilo-bytes of Read Only Memory (ROM - This is where the instructions needed for the operation of the drive are stored), and two Input/Output chips (I/O - used to control the drive motor, R/W head, and send info back and forth). This arrangement frees up a large portion of the C-64's memory since the instructions used to store/retrieve data are built into the disk drive and do not have to be loaded and executed by the computer. All the 64 has to do is remember a handfull of commands and the drive takes care of the rest.

If you are one of the fortunate few who have been able to understand and digest the information in the 1541's manual, then you can skip ahead to the section on 'Recovering From Disk Errors'. Right now I'll try and cover some of the commands and techniques that we'll need in order to have the disk drive do something other than just loading and saving.

There are two ways of sending data to the disk drive. The first way is used to write information onto the diskette itself (programs, files, etc..). After an open command 'PRINT# a' is used to write information to a file and both 'INPUT# a' and 'GET# a' can be used to retrieve the data from the diskette. 'a' is the file number which can range from 1-255. A file number greater than 127 will cause a linefeed to be transmitted after every carriage return via PRINT#. An appropriate open command is required before a file command is executed for the first time (by

'file command' I am referring to PRINT#, INPUT#, and GET#). The open command used for file access follows the following format.

OPEN a,b,c," d:name,e,f "

Where:

(a) Represents the file number. This number is used by the file access commands (see above) and is used to determine which file is being accessed.

(b) Represents the device number. This is usually an eight (the factory set drive number) but could range from eight through eleven. You can have as many as 4 disk drives hooked up to your computer at the same time and each would have it's own drive number.

(c) Represents a value known as the secondary address. This number is used to signal a device to perform a certain function. Commodore printers use this to change print modes etc.. For file access, two through fourteen are commonly used. Zero and one are reserved by the computer for program load and save operations and fifteen is used to talk directly to the disk drives computer (we'll get to that in a minute).

(d) Represents the drive number, which can be either a zero or a one. This feature is provided for the users of dual drives. Since a dual drive has only one device number there has to be a way of determining which drive is to be used at a certain time. This is it.

This number, along with the colon, can be omitted if you are using a 1541. If an '@' precedes the colon, and the file is being opened for a write operation (see below), then the new data will be written over the old file, completely destroying the old information.

(name) is just the name of your file (see, not everything is hard to understand)

(e) This is a letter representing the type of file that you wish to work with. This can be either an 'S', 'P', or 'U'. These letters stand for 'Sequential', 'Program', and 'USER. We'll go through a few examples in just a second. Relative files require a more complex format to access, and such, this article will not be going into depth on the subject.

(f) This determines the kind of operation that will be performed on the file to be opened. An 'R' means that we want to Read data from this file. A 'W' indicates that we want to Write data to this file. And finally, an 'A' means that we want to write information onto the end of the file without destroying any information already written there.

O.K. now for some examples.

1) open 1,8,2," testfile,s,w "
2) open 3,9,3," 1:phone numbers,s,r "
3) open 2,8,2," tax data,s,a "
4) open 1,8,9," star trek,p,r "
5) open 7,8,2," @:masterpiece,p,w "

1) This opens a file on disk device eight, as file number one. The file is named " testfile ", it is a sequential file and is going to be written to.

2) This opens a sequential file on disk device number nine, as file number three and using drive number one. The file to be read is named " phone numbers ".

3) This opens a sequential file on disk device number eight as file number two. Any data sent to this file, named " tax data " , will be added to the end of the file.

4) This opens a program file for reading as file number one on device number eight.

5) This erases the old program named " masterpiece " and opens a new program file, as file number seven, for writing with the same name on device eight.

Note: If you try to open a new file with the same file number as a file that is currently open, the computer will respond with an error message. Each file must have a unique number. This is the only way that the computer knows where to send a certain piece of data.

The next type of open command is used to communicate directly with the computer inside of the disk drive. The format looks like this:

open a,b,15

(a) and (b) are the same as before.

The secondary address of fifteen signals the disk drive's computer and informs it that you want to give it a direct command.

Now we can start examining some of the more simple disk commands and work our way slowly to the more advanced stuff.

When you buy a brand new diskette it has to be properly prepared before you can store any data on it. (for the following command and for all of the others we'll assume that you have entered the command " open 15,8,15 ". This will open the command channel prior to sending any command. After the command type in " close 15 " to let the disk drive know that you have finished talking.)

Preparing a new disk is called FORMATTING. This fills the disk with information that the drive needs to know in order to read/write it's own data and is similar to the cutting of grooves in a phonograph record.

To format a disk you can issue the following command:

```
print#15,"n:diskname,id"
```

Disk name is (you guessed it!) the name that will appear on your disk whenever you ask for a catalog of the disks contents.

"id" this is a unique two character (can be both letters and numbers) code that is used by the disk drive to identify that particular disk. If you use two disks with the same id your drive could possibly become confused and store information in a place that could cause you a great deal of anguish.

After you've just saved the fifteenth version of that great program you're working on, you realize that your rapidly running out of space on your diskette. You reach for another diskette to work with and find that it is full also. Since Murphy is the patron saint of all programmers and hackers, it just happens that it's either too late to run to the nearest computer store to buy another box of disks, or you've just spent your last couple of dollars on some neat gadget, or the nearest computer store is seventy miles away, or all of the above. So what do you do now? No, you don't field test your 1541 as a wheel block for your car (besides it the wrong shape. Try a Timex/Sinclair. Their wedge shaped 'computer' could keep a car from rolling down even the steepest of hills). All you have to do is erase some of those old files that have outlived their usefulness. How? you may ask (and I knew you would). Easy.

```
print#15,"s:program name"
```

This kills the file called "program name" from your disk and allows you access to the more room on the disk.
(Oh yea, the "n" stands for "new" and the "s" stands for "scratch")

These two commands give you an idea of the structure used in talking to the disk drive. As we go along I'll be introducing others which I will then explain (I promise).

Recovering From Disk Errors

Whenever you write to a diskette, the disk drive automatically reads back each part, and if the two don't match exactly then it tries to write it again (up to five times). If the drive still can't write the data to the disk the drive then signals an error condition by flashing a red light located at the front of the disk drive.

Errors can also occur at other times, like when you are trying to read back a piece of data. After you have successfully written to a disk a number of things can then happen. Part of the disk may have gone bad for various reasons such as dirt, dust, cigarette smoke, heat, or stray magnetic fields (DON'T store diskettes near a telephone. The bell operates using an electro-magnet. One phone call from a well meaning friend could sabotage your whole library). When this happens the value of one or more bytes have been either changed or corrupted. Most errors are detected through a method known as 'checksumming'. The way that this works is that all of the bytes in a certain section are added together and the resulting number is then chopped up, leaving the sixteen rightmost binary digits. This leaves a value between 0 and 255 which is then stored on the disk along with the data. When the data is read back into the disk drive, a new checksum is calculated and is then matched to the one originally stored on the disk. If the two checksums don't match then the disk drive tries again (five times). If they still don't match the drive stops whatever it's doing, sulks, and flashes that stupid light again. In computer jargon this is what's known as a boo-boo (an error condition). The most common boo-boo is an error number 23. This stands for a 'checsum error in data', which means that somewhere in the actual data written to the disk one or more of the bytes have gone belly up. The checksum calculated for the data block and stored on the disk does not match the checksum calculated at the time the information was read into the disk drive.

A few months back I was doing some thinking about what the various error messages actually mean. I realized that with an error 23 the data had to be read into the disk drive before a checksum could be calculated. The data was there, but because the disk drive had decided that the information was not 100% pure and unsullied it could not be read by the computer. Well if the drive can read the information why can't we? The answer is that we can but it has to be done in a manner that is both sneaky and underhanded.

This is where we get to some of the more interesting disk commands. The first one is known as the 'Block-Read' command. This command tells the disk drive to read a specific sector on a certain track into memory which can then be accessed and manipulated by the computer. In order to do this we must tell the disk drive to reserve a part of memory for our exclusive use. This is done by opening a drive 'buffer'. After opening the command channel (OPEN 15,8,15 - remember?) you them open the buffer like so:

```
OPEN 2,8,2,"#"
```

The file number is used later on by the actual disk commands and the number sign tells the disk drive to look for the first available buffer and to reserve it for our exclusive use as file number two.

Try this little program.

```
10 open 2,8,2,"#"
20 open 15,8,15
30 print#15,"u1:"2;0;18;1
40 get#2,a$: if a$ = "" then a$ = chr$(0)
50 print a$;
60 if st = 0 then 40
70 close 2: close 15
```

Now I'll see if I can explain it.

Line 10 - opens a channel and reserves a buffer.
Line 20 - opens the command channel (Tell the disk drive to wait for a command)

Line 30 – 'u1' is a substitute for the Block–Read command. The general format for the command is

PRINT#15, "u1:" a;b;t;s

Where :

(a) corresponds to the file number used in the buffer open command (line 10)
(b) corresponds to the drive number (always a zero on the 1541)
(t) stands for the track number to be read
(s) stands for the sector number to be read

Line 40 – Read a byte of data from the buffer. If we don't get a byte back then we let a$ equal the character equivalent of a zero. This has to be done since the C–64 will not recognize an ASCII value of zero in data transmissions. The zero is used as a signal between different devices and thus is not useable when reading data (writing is a different matter though).

Line 50 – Print the character to the screen. This includes cursor codes and certain character codes that won't print properly to the screen.

Line 60 – ST is the status variable. This tells us the status of the last operation performed. If it is not a zero then either an error has occurred or we have reached the end of the data block (256 bytes)

Line 70 – Close file number 2 and file number 15.

ASC(A$); A$

This program will read the first block of your diskette's directory into the disk drive's memory, where it will then be read byte by byte into the computer and be displayed onto the screen. Since quite a few of the possible values will not print to the screen properly you might want to change line 50 to 'print asc(a);a$' or just 'print asc(a$);'. This will print the character number and the character, and just the character number respectively. You can change the values for T and S to anything that you wish (T goes from 1 to 35 and S ranges from 0–20 to 0–16 depending on the track number). This would let you examine the whole disk a block at a time.

The Block–Write command "u2" is almost identical to the Block–Read command except that you are writing information to the disk instead of reading it. This program will write 256 a's to a block located at track one and sector one. You can use the first program to examine this block before and after it has been written. All you have to do is change the track and sector values in the first program to match this one (namely one and one respectively).

```
10 open 2,8,2,"#"
20 open 15,8,15
30 for i=1 to 256
40 print#2,"a";
50 next i
60 print#15,"u2:"2;0;1;1
70 close 2: close 15
```

Line 10 – open a buffer
Line 20 – open the command channel
Line 30 to Line 50 – Send 256 A's to the buffer that we've opened
Line 60 – Write the buffer to the disk
Line 70 – close all files

Now that we've covered the necessary background information, we can proceed to try and eliminate some errors.

I call this program 'error 23 fix', and what it does is read a block that contains an error 23 and then re–write that block back to the disk (but without the error)

```
10 input "what track is the error on ";t
20 input "what sector is the error on ";s
30 open 2,8,2,"#"
40 open 15,8,15
50 print#15,"u1:"2;0;t;s
60 print#15,"u2:"2;0;t;s
70 input#15,a,s$,d,f
80 print "status — ";a;" ";s$;" ";d;" ";f
90 if a=0 then 200
100 print "it didn't work–could be a damaged disk"
110 print " try again?"
120 get x$: if x$="" then 120
130 if x$="n" then 200
140 goto 50
200 print:print "done!!"
210 close 2: close 15
```

There is only one warning that I should tell you about before you use this program. If the error occurred due to the fact that a byte of data was corrupted instead of the checksum byte then when the block is re–written that bad byte will also be re–written. After using this program to recover your own program or file it would be smart to copy it over to a new disk. Next you should examine your program or file carefully to find the bad byte of information. In a program this could turn up as a print statement with a weird character or as a transformed command (such as a 'for' statement becoming a 'gosub' statement. To copy a file to another spot on the same disk use the command 'print#15,"c:newname=oldname" ' where oldname is the current name of the file and newname is the newname of the file.

This next program is of specific interest to the hacker community in general. Track 18, sector 0 of a disk contain certain information that is vital to the proper operation of the disk. This block is the first block of the directory and also contains a map telling the drive which blocks have been used and which are still available. It also contains the name of the disk and the type of disk drive that was used to format the disk. This last piece of information is of specific interest to us at the moment. If this byte is changed to a value other than what it is supposed to be then the next time the disk drive reads that disk a number of things happen. 1) the disk becomes unwriteable. 2) the disk becomes impossible to back up (using a dual drive).

This means that the information on the disk cannot be changed or altered (barring an act of God such as little sisters). Occasionally a piece of software has this little gem added to it's various other protection schemes to keep people from altering codes contained on the disk. Another nasty trick has turned up in high school computer classes. Some smart alek changes the DOS version of all his friends(?) disks. Now that poor person can't write to his disks unless he re-formats them, and if he does then he loses all the data on that particular disk. Nasty. Now it should be possible to change the DOS version number back to what it originally was. The only problem is that normal methods of reading/writing have to go through various error detection routines, all designed to make life easier for a programmer but a real pain until you learn about ways to get around them.

The way that we can do this is to get directly inside of the drive (figuratively speaking) and force it into doing what we want by giving it some false information (usually impossible using normal methods).

Here's the program that will Re-write track 18, sector 0 and eliminate the error caused by a wrong DOS version number.

```
10 open 15,8,15
20 print#15, " uj " : rem disk drive reset command
30 open 2,8,2, " #0 " : rem reserve buffer number zero
40 print#15, " u1: " 2;0;18;0
50 print#15, " m-w " chr$(2)chr$(3)chr$(1)chr$(42)
   : rem write a $2a to buffer pos 3
60 print#15, " m-w " chr$(6)chr$(0)chr$(2)chr$(18)chr$(0)
   : rem set trk/sector #'s
70 print#15, " m-w " chr$(0)chr$(0)chr$(1)chr$(144)
   : rem re-write the block
80 print#15, " uj " : rem clean and reset memory
90 close 2: close15
```

After running this program your disk should again be writeable in a normal fashion. In this program we've used the Memory-Write command. This command allows us to write a single byte or number of bytes anywhere in the disk drives memory. In actual practice we can only write to either the 2k of RAM (which extends from $0000 to $0800) or to the registers in the I/O chips (from $1800 to $180F and from $1C00 to $1C0F). The companion command to Memory-Write is the Memory-Read command (M-R). This command allows us to read any memory location in the disk drive, including the 16k of ROM located from $C000 to $FFFF.

Memory-Write:

```
print#15, " m-w " chr$(a)chr$(b)chr$(c)chr$(d)chr$(d)chr$(d). . .
```

(a) Low byte of address to be written to
(b) High byte of address to be written to
(c) Number of bytes to write
(d) Actual Values to be written

Memory-Read:

```
10 print#15, " m-r " chr$(a)chr$(b)
20 get#15,z$: if z$ = " " then z$ = chr$(0)
30 z = asc(z$)
```

(a) Low byte of address to be read
(b) High byte of address to be read
(z$) This will contain a character equal to the value contained at that memory location
(z) This will contain the actual value stored at that memory location

This little program will blink the little light on the front of the disk drive.

```
10 open 15,8,15
20 print#15, " m-r " chr$(0)chr$(28)
30 get#15,a$: if a$ = " " then a$ = chr$(0)
40 a = asc(a$)
50 light = (a or 8)
60 nolight = (light and 247): rem (255-8)
70 for i = 1 to 500: rem number of flashes
80 gosub 1000: rem see if key is pressed
90 gosub 2000: rem turn light on
100 for j = 1 to 40: rem delay
110 gosub 1000: next j
120 gosub 3000: rem turn light off
130 for j = 1 to 40
140 gosub 1000: next j
150 next i
160 goto 9999
1000 get b$
1010 if b$<>" " then 9999
1020 return
2000 print#15, " m-w " chr$(0)chr$(28)chr$(1)chr$(light)
2010 return
3000 print#15, " m-w " chr$(0)chr$(28)chr$(1)chr$(nolight)
3010 return
9999 close 15
```

Well, I guess we've covered quite a bit of material that might prove useful or at least help light the spark of creativity among a few of you.

Simulating a Dual Drive With Two 1541's

Gerald Neufeld
Brandon, Manitoba

Would you like a dual drive for your Commodore 64 or VIC–20 but can't afford an MSD drive or an IEEE interface and a used 4040 drive? In most situations you don't really need a dual drive — you can get away with two 1541's, one as device #8 and one as device #9. However, some commercial programs such as Easy Script and Oracle require a dual drive (device #8 with drive #0 and drive #1) to perform certain operations. This article describes how to use two 1541 drives to simulate a dual drive.

Possible Solutions

The most obvious solution to our problem would be to convert one of the 1541's so that it becomes drive #1, device #8. Unfortunately, you can't change a 1541's drive number as easily as you can its device number. The only way to do this would be to rewrite and replace the 1541's ROM.

Since we can't change a 1541's drive number, how do we simulate drive #1? The trick is to leave the drive alone and modify the commands issued by the computer. Although this may sound difficult, it turns out to be quite simple. The only common commands that specify a drive are LOAD, SAVE, and OPEN. If we can intercept these commands and redirect those that refer to drive #1, the PRINT#, INPUT#, GET#, and CLOSE commands will all work properly.

A Solution That Works

Our simulation uses to standard 1541 drives. One of the drives will not be modified and will remain as device #8, drive #0. The device number of the other drive must be changed so that it becomes device #9, drive #0. This may be done temporarily (see below) or permanently (see your 1541 Users Manual). Once the drives have been set up, the short machine language routine given below is loaded into the computer's cassette buffer and activated. This routine intercepts and checks all LOAD, SAVE, and OPEN commands. If the file name indicates that the command refers to drive #1 on device #8, the file name and device number are changed to redirect the command to drive #0 of device #9. As a result, the BASIC or machine language program thinks it is accessing drive #1 of device #8, but in actual fact it is accessing drive #0 of device #9.

Commands That Work

Most of the commonly used disk commands will be intercepted and modified by the machine language routine. The table below lists some of the commands that have been tested and found to work correctly.

Original Command	Modified Command
LOAD "$1",8	LOAD "$0",9
LOAD "1:TEST PRG",8	LOAD "0:TEST PRG",9
SAVE "1:TEST PRG",8	SAVE "0:TEST PRG",9
SAVE "@1:TEST PRG",8	SAVE "@0:TEST PRG",9
OPEN 1,8,5,"1:FILE,S,W"	OPEN 1,9,5,"0:FILE,S,W"
OPEN 1,8,5,"@1:FILE,S,W"	OPEN 1,9,5,"@0:FILE,S,W"
OPEN 1,8,5,"1:FILE,S,R"	OPEN 1,9,5,"0:FILE,S,R"

Disk commands such as these will always be intercepted and modified properly when they are part of a BASIC program. Commands issued from a machine language program will be handled correctly provided the normal KERNAL entry points are used for LOAD, SAVE, and OPEN functions.

Note that an OPEN statement set the device number for all subsequent accesses of the file. As a result, PRINT#, INPUT#, GET#, and CLOSE commands that refer to that file will be redirected to the appropriate drive.

Limitations

1. Disk file names must not contain a "1" as either of the first two characters in the name. A "1" in either of these positions will cause the command to be redirected to device #9.

2. Since this is just a "simulated" double drive, special dual–drive commands such as DUPLICATE a diskette and COPY from one drive to another will not work. In addition, direct–access commands that reference drive #1 (such as "U1:5 1 18 0") will not be redirected to the appropriate drive.

3. Since the machine language routine resides in the cassette buffer, you may not use your cassette tape recorder while using this routine. If you need to use the cassette buffer for other purposes, you will have to assemble the source code to reside elsewhere in RAM (such as $C000 in the C–64).

Instructions

If one of your drives has been permanently modified to be device #9, skip steps 1 to 3. Simply turn on your system and proceed to step 4.

1. Turn on your Commodore 64 (or VIC–20) and the 1541 that is to be changed to device #9. This is the one that will appear to be device #8, drive #1

2. Enter and RUN the following program to convert the 1541 to be device #9.

```
10 OPEN 15,8,15
20 PRINT#15,"M-W"CHR$(119)CHR$(0)CHR$(2)
   CHR$(9+32)CHR$(9+64)
30 CLOSE 15
```

3. Turn on the 1541 that will remain as device #8, drive #0.

4. Load and RUN the BASIC loader program given below.

5. The simulated dual drive is now in operation.

6. The routine may be toggled off and on with SYS 828.

```
KG  100 rem   using device #9 drive #0 to
BL  110 rem   simulate device #8, drive #1
GK  120 rem   copyright: g. neufeld, 1984
GP  130 :
JH  140 k=828:sum=0
LP  150 read x:if x<0 goto 180
IN  160 poke k,x:sum=sum+x:k=k+1:goto 150
OB  170 :
NE  180 if sum<>13992 then print " bad data " :stop
CK  190 sys 828
GI  200 print " routine activated "
GE  210 :
MB  220 data 160,   1, 185,  48,   3,  72, 185
KN  230 data 123,   3, 153,  48,   3, 104, 153
KB  240 data 123,   3, 185,  50,   3,  72, 185
CO  250 data 132,   3, 153,  50,   3, 104, 153
KD  260 data 132,   3, 185,  26,   3,  72, 185
HP  270 data 112,   3, 153,  26,   3, 104, 153
EA  280 data 112,   3, 136,  16, 211,  96,  32
LE  290 data 137,   3,  32, 108,   3,  76, 180
NG  300 data   3,  32, 137,   3, 169,   0,  32
CF  310 data 117,   3,  76, 180,   3,  32, 137
AG  320 data   3,  32, 128,   3,  76, 180,   3
GJ  330 data 160, 255, 132, 251, 165, 186, 201
IM  340 data   8, 208,  32, 200, 177, 187, 201
PP  350 data  49, 240,   7, 200, 177, 187, 201
EE  360 data  49, 208,  18, 169,  48, 145, 187
KJ  370 data 169,   9, 133, 186, 132, 251, 165
EO  380 data 187, 133, 252, 165, 188, 133, 253
GP  390 data  96, 164, 251,  48,   4, 169,  49
CB  400 data 145, 252,  96, -99
```

Machine Language Source

For those of you interested in how the routine works, here is the PAL TM assembler output listing. The routine is surprisingly short and simple. However, it does illustrate how you can intercept commands without having to wedge into the 64's CHARGET routine. Note that the BOOT routine puts the actual C-64 or VIC-20 LOAD, SAVE, and OPEN addresses into the JSR commands in lines 490, 540, and 580. Infinite recursion does not occur!

```
DL  100 ;     gerald neufeld, december, 1984
OB  110 ; using two 1541's to simulate a dual drive
OO  120 ;
KH  130 ; routine to redirect load, save, and open
GM  140 ; commands referring to drive #1, device #8
HC  150 ; to drive #0, device #9
GB  160 ;
KD  170 ; uses cassette buffer of c-64 or vic-20
KC  180 ;
HP  190     *    =    $033c
OD  200 ;
AE  210 .opt p,o1
CF  220 ;
OG  230 boot  ldy   #$01
HK  240 boot1 lda   $0330,y
GH  250       pha
OP  260       lda   load+6,y
AD  270       sta   $0330,y
AK  280       pla
KF  290       sta   load+6,y
CB  300       lda   $0332,y
CL  310       pha
FD  320       lda   save+4,y
OG  330       sta   $0332,y
MN  340       pla
BJ  350       sta   save+4,y
FF  360       lda   $031a,y
OO  370       pha
MH  380       lda   open+4,y
BL  390       sta   $031a,y
IB  400       pla
IN  410       sta   open+4,y
PF  420       dey
AK  430       bpl   boot1
EK  440       rts
ID  450 ;
OA  460 ; modified routines
ME  470 ;
BG  480 open  jsr   fix
OF  490       jsr   open
OO  500       jmp   unfix
EH  510 ;
BH  520 load  jsr   fix
FD  530       lda   #$00
EI  540       jsr   load
AC  550       jmp   unfix
GK  560 ;
FK  570 save  jsr   fix
NK  580       jsr   save
IE  590       jmp   unfix
OM  600 ;
GB  610 fix   ldy   #$ff
EE  620       sty   $fb       ; change flag
MO  630 ;
GC  640       lda   $ba       ; device number
BG  650       cmp   #$08      ; is it device #8
DH  660       bne   fix2      ; not for us
EB  670 ;
II  680       iny
EO  690       lda   ($bb),y
BD  700       cmp   #$31
BA  710       beq   fix1
AL  720       iny
MA  730       lda   ($bb),y
JF  740       cmp   #$31
IB  750       bne   fix2
OG  760 ;
IH  770 fix1  lda   #$30      ; set drive# to 0
MH  780       sta   ($bb),y
AF  790       lda   #$09      ; set device# to 9
BD  800       sta   $ba
JG  810       sty   $fb       ; save pointers
KA  820       lda   $bb
NF  830       sta   $fc
BC  840       lda   $bc
EH  850       sta   $fd
CN  860 ;
LE  870 fix2  rts
GO  880 ;
OI  890 unfix ldy   $fb
FP  900       bmi   unfix1    ; no change was made
JP  910       lda   #$31      ; undo change
IB  920       sta   ($fc),y
IB  930 ;
GC  940 unfix1 rts
```

In Defense of The Frontal Assault

Chris Zamara, Technical Editor

The frontal assault: crude, messy, inelegant, but usually an effective method of attack — the stuff of which legends and heros are made. Attacking a problem can be thought of as a battle, planning our strategies and calculating maneuvers. When the problem is computer-oriented, we write a program. But apply the frontal assault to programming, and you're likely to get bad press, not a hero's welcome. Attack a program directly on the machine, and you're also fighting a battle against conventional wisdom.

When a hacker writes a program, he typically takes the frontal assault approach, turning on the computer before he even knows what he's going to write. As he enters code, ideas take shape, and he builds on these ideas while testing, sparking new concepts and more efficient solutions all the time. Dynamic and creative, this is true art: a pure flow of ideas from the mind to the medium, with no pollution from physical limitations.

The hacker described above could be any one of you. The hacker described above is also condemned as a "bad programmer" by instructors and experts of computer programming. As taught in schools, programming is a strict science in which there is no room for personal creativity. What you will hear from the wise programmers of our age is that you should have your program 100% planned and coded before you turn on the machine. Wear out pencils rather than use electricity. Don't test things on the machine, look them up in manuals. The frontal assault is reserved for the undisciplined masses or the lunatic fringe of hard-core hackers. Either way, *good* programmers don't do it.

Now, the main thrust of current programming techniques makes sense: programs should be modular, structured, easy to understand, flexible, and should break a problem down into successively smaller, simple problems. No one can deny that following those guidelines results in "better" programs, ones which will have a long and useful life. But the frontal assault technique doesn't preclude structured programming or top-down coding. It just means that sections of code are developed on the machine instead of on paper. The interactive, personal programming environment is perhaps the greatest benefit that microcomputers have given to the programming industry, and the educators and industry are reluctant to take advantage. Oh, sure, interaction is highly touted for *users*; heck, they even get pretty pictures and moveable "mice" to entertain them. But programmers? It seems that they are not included in the interactive-age picture. Interactive programming is somehow mixed up with non-structured programming techniques, and thus labelled as bad.

When I studied computer science at Ryerson Polytechnic Institute in Toronto, desk-debugging — working through a program on paper before running it — was encouraged. It made sense, too, because each program line was punched onto a card using ancient, temperamental keypunch machines. The stack of cards representing your program was run through equally ancient card readers, and if you were lucky, you'd get your output about an hour later, make that four hours when it was busy. No one ever saw the computer or even knew where it was or who made it. The point is, you wanted to be pretty darn sure that your program worked before waiting another hour for the output. For a programmer to debug in such an environment is like an artist having to wait until each brushstroke dries before painting the next. I wonder if Mona Lisa would have such a mysterious smile if that were the case.

Fortunately, programming environments such as the above have been pretty much eliminated by now, replaced with terminals allowing much more interaction. Unfortunately, the prescribed programming techniques haven't changed accordingly, and the old plan-in-pencil school of thought has been dragged into the microcomputer scene. We Commodore users have it very good, since the standard editor and interpreter lend themselves so well to frontal-assault, type-while-you-think programming. With software packages for the Commodore 64 like the new COMAL cartridge, the interactive environment works so well that you can build good, structured, readable, maintainable programs directly on the computer. The natural urge with such a system is to build program modules one at a time, testing each as you go along, without spending three days grinding pencils and "playing computer" by walking through your untested code. To suppress that urge in the name of proper programming form is about as counter-productive as using masking tape to "correct" the behaviour of a left-handed child.

Perhaps I am confusing the issue here by mixing up programming as a recreational hobby, and programming as a business-serious profession. Perhaps, but I don't think so. Look at the brains behind any commercially successful piece of software and you'll probably find a frontal-assaulter. And that's good news, because it supports the premise that programming is as much of an art as it is a science, and as such, the excellence of a work will depend on the creativity of its author. If that's true, the last perpetrators of the frontal assault can proudly ignore the derision of their contemporaries, and continue their craft with conviction. And once the current fervor dies down, maybe some day, they'll get a hero's welcome after all.

Flexible Vector Management

Chris Zamara, Technical Editor

Warning: This article contains advanced machine-language programming concepts!

A vector is simply the address of an important Kernel routine stored as two bytes in memory. Its purpose is to give the user more control of the computer's operating system. Important routines in the operating system (the Kernel) are not entered directly, but their addresses are stored in RAM locations, and the Kernel jumps to the destination of this RAM vector. This allows the user to intercept the flow of control by changing the vector, and sneak in some operation before the Kernel routine is executed, or skip the normal routine altogether. This capability allows for great flexibility, since you can use the main parts of the Kernel and just add or substitute the code that you need. Your own programs can be "linked" or "connected" directly to the system so that they don't need an explicit SYS command to execute. However, to properly use vectors so that your programs are themselves flexible and don't mess up the system, there are some special programming techniques you should use.

Examples of Vectors

Routing your routines through vectors allows you to do things like define function keys, add BASIC commands, trap errors, disable the STOP or RESTORE keys, use standard I/O commands with custom I/O routines for non-Commodore peripherals, or execute any machine language program 60 times a second. Here's some examples of vectors in the 64 and what you can use them for.

The IRQ vector (at $0314-$0315 on the VIC/64/16/+4) is the most popular. This vector points to the Kernel routine which scans the keyboard and updates the keyboard buffer, and updates the clock for the BASIC TIme function. An IRQ is a hardware interrupt (Interrupt ReQuest), and is triggered by a timer 60 times a second. You can intercept the IRQ vector whenever you want some operation to be performed repeatedly, for example constantly displaying something on the screen. On PETs, performing some action when a certain key is pressed requires use of the IRQ vector.

The output vector points to the standard output routine at $FFD2. By intercepting the output vector (at $0326-$0327 on the VIC and 64), you can force the computer to send everything it prints to the screen somewhere else as well, like the RS-232 port or a disk file.

The warm start vector on the 64 at $0302-$0303 is used after a RESTORE, after a command is executed in direct mode (or you press RETURN), or after a BASIC program ends. A few possible applications of intercepting this vector: You can fix up anything that may have been reset by a RESTORE (like border and background colours), execute a program automatically after a LOAD, or re-open a file that may have been closed by entering a BASIC program line.

All Commodore machines use vectors; here's a few useful ones:

Location 64/20	+4/16	Vector	used when
0300	0300	error	any error occurs (error # in .A)
0311	032C	USR	USR(x) function is called
0318	n/a	NMI	RESTORE is struck or rs-232 input
0326	0326	stop	STOP key pressed
032C	032A	abort I/O	after BASIC line entered or CLR
0330	032E	load	used by LOAD command
0332	0330	save	used by SAVE command

Set-Up Programs: Connecting the Vector-Driven Routine

A vector holds the address of the destination routine in low, high order format. As an example, let's say you wanted to point the IRQ vector to a routine at $C000. In this case, you'd put $00 in the first byte of the IRQ vector and $C0 in the second byte. In assembler, you could code:

```
;vector set-up routine
IRQVEC  =   $0314       ;irq vector in VIC/64/16/+4
        LDA #$00        ;dest address low
        STA IRQVEC      ;irq vector low
        LDA #$C0        ;dest address high
        STA IRQVEC+1    ;irq vector high
        RTS             ;end of set-up program
```

The above code is the simplest way to change a vector, but in the case of the IRQ, there are two more instructions required. An SEI instruction must appear before changing the vector to disable any interrupt from occurring during the time the low byte has been changed and the high byte hasn't. After both have been changed (just before the RTS), the IRQs can be enabled again with CLI.

Now, if you wish to intercept the IRQ vector and then continue with the usual routine, the program at $C000 will have to jump to the original destination of the vector once it's finished. The easy way to do this (but not always the best way) is to just look at the IRQ vector in memory and find out where it points normally, which happens to be $EA31 in the 64. The code at $C000 could perform whatever routine you wish and then jump to $EA31 like this:

```
*   =   $C000       ;origin at $c000
    JSR YOURRTN     ;perform your routine,
    JMP $EA31       ;then jump to the normal IRQ destination
```

The above approach will work, but you may wish to put the IRQ routine right after the set-up code, and not care exactly where it lies on memory. You can let the assembler take care of finding the address of the routine. The following listing is the set-up and IRQ routines together. To make this exercise useful, a subroutine for the C64 which is executed by the IRQ routine is also listed. All it does is display a message on the top line of the screen, which will be permanent and un-erasable since it's put there 60 times a second.

```
; irq set-up routine (PAL assembler compatible)
IRQVEC    =   $0314
          SEI               ;disable interrupts
          LDA #<NEWIRQ      ;low byte of destination address
          STA IRQVEC        ;irq vector low
          LDA #>NEWIRQ      ;high byte of destination address
          STA IRQVEC+1      ;irq vector high
          CLI               ;enable interrupts
          RTS               ;back to calling program or basic
;
NEWIRQ    =   *             ;new irq destination
          JSR DISPLIN       ;perform display subroutine
          JMP $EA31         ;normal irq destination
;
; irq vector-driven routine follows
; (displays a message on the top screen line)
DISPLIN   =   *
          LDX #39           ;40 characters
SFIL      =   *
          LDA MESSAGE,X     ;next character from message
          STA $0400,X       ;store in next screen memory byte
          DEX               ;do all 40
          BPL SFIL          ;byte #0 is last one
          RTS
;
MESSAGE .ASC "    *** UNERASABLE STATUS LINE ***    "
; pad message string to 40 characters
.END
```

So far, we've successfully trapped the IRQ vector to execute our routine, and it works, under normal conditions. For short vector-driven routines, the above set-up technique is usually used. But what if another interrupt-driven routine was already executing? Our program would disconnect it. Conversely, if you were to execute another similar IRQ setup routine while the above program was running, it would take complete control. Another problem is that the above program relies on the fact that the normal IRQ entry point is at $EA31; on a different ROM version it may not work. You may wish to write a program which will work on the VIC or the 64, which have their IRQ vectors in the same place, but contain different addresses. And as a final problem, there's no way to turn off our little screen display ditty once it's going, short of a RESTORE sequence – which is always a bit drastic, messing up your screen colours and all.

The first two problems can be solved with one modification. The trick is to make the program jump to the original destination of the vector instead of some fixed address like $EA31. In other words, store the existing vector somewhere, change the original vector to point to the user routine as usual, then have the routine jump to the stored address when it's finished. Managing vector-driven routines that way means that whatever routine was being executed through the vector when your program starts up will continue to be executed. This allows linking more than one program through a single vector, chaining them by jumping from one to the next until the final Kernel routine gets executed. (This may not always be desirable, for example if the new user routine overwrites the old one in memory, but in that case the vector must simply be restored to normal before the new vector set-up routine is executed.) The code to accomplish this flexible vector chaining system is quite simple. The first thing the program should do is save the old vector's contents. Using the above program example, you would place the following code at the start of the program:

```
LDA IRQVEC      ;existing vector, low byte
STA OLDIRQ      ;store
LDA IRQVEC+1    ;high byte
STA OLDIRQ+1
```

(OLDIRQ and OLDIRQ+1 can be any two bytes in the same page)

Then, when the user routine finishes, instead of jumping to a fixed address, it does an indirect jump like this:

```
JMP (OLDIRQ) ;jump to original vector destination
```

That takes care of the linking problem, but introduces a new one: what happens if you execute the set-up routine a second time, i.e. while the vector has already been changed? Bad things happen. Like a system crash. If you think about it, you'll see that executing such a vector set-up routine twice causes a crash because the second time it's executed, the vector is already pointing to the user routine – and that's where the user routine jumps when it's finished. In other words, the routine ends up executing itself over and over again indefinitely. A good, flexible, vector set-up routine has to assume that somewhere along the way, someone is going to be stupid enough to execute it while the vector has already been linked. To do that, it has to make sure that the existing vector doesn't point to the user routine. If it does, it should give some kind of warning ("program X already activated"), or just do nothing. The initial code code could look like this:

```
    LDA  IRQVEC       ;check existing vector, low byte
    CMP #<DISPLIN     ;. . .against user rtn addr low
    BNE  ADDROK       ;if different, continue as usual
    LDA  IRQVEC+1     ;if equal, check high byte too
    CMP #>DISPLIN
    BEQ  RERUN        ;if high byte also equal, re-run
                      error
;
ADDROK  =  *
;address ok, now save old vector as before
    LDA  IRQVEC       ;existing vector, low byte
    STA  OLDIRQ       ;store
    LDA  IRQVEC+1     ;high byte
    STA  OLDIRQ+1
;vector can now be changed to point to user routine
    SEI               ;disable interrupts
    LDA #<NEWIRQ      ;low byte of destination addr
    STA  IRQVEC       ;irq vector low
    LDA #>NEWIRQ      ;high byte of destination addr
    STA  IRQVEC+1     ;irq vector high
    CLI               ;enable interrupts
    RTS               ;back to calling program or basic
;
ERROR  =  *
;the error routine may print an error message,
;or simply exit without doing anything.
    RTS
```

It may seem like more work than necessary to put in all that code to check for re-runs, but if you do, you'll be able to create an expandable vector-driven system. Any number of different vector-driven routines can be "hooked up" to run through the desired vector by simply running the appropriate set-up program. But now we get to the next problem: how to "unhook" them.

It's important that a vector set-up routine can disconnect the routine as well. That's a simple task: just store the saved original vector (OLDVEC) back into the actual vector. In other words:

```
LDA  OLDVEC
STA  IRQVEC
LDA  OLDVEC+1
STA  IRQVEC+1
```

A good place to put the disconnect code is right at the beginning of the set-up routine, with a JMP just before it to skip to the usual entry point. Then to connect the user routine through the vector, just SYS to the usual start address, and to disconnect it, SYS to the address plus 3. The code becomes:

```
        JMP  SETUP        ;set up the vector normally
;vector disconnect code follows
        LDA  OLDVEC       ;get stored vector low
        STA  IRQVEC       ;put back in actual vector low
        LDA  OLDVEC+1     ;stored vector high
        STA  IRQVEC+1     ;put into actual vector high
;
SETUP =    *
;the rest of the setup code is as above.
```

Sharp readers may have noticed a potential problem here: what if you tried disconnecting a routine that hadn't been connected yet? Well, whatever was contained in OLDVEC would be stored in the vector, which could again be bad news.

If our multiple-vector-driven system is to be as flexible as possible, the exit feature should also check the vector address against the address of the vector-driven user routine. We can easily do that by incorporating the section of code which compares the addresses into a subroutine, and calling the subroutine from both the vector connect and disconnect portions. Here's the subroutine, which simply compares the vector address with the routine's address and exits with the Z flag clear if they're equal:

```
DDCHECK =    *
         LDA  IRQVEC
         CMP #<DISPLIN
         BNE  COUT
         LDA  IRQVEC+1
         CMP #>DISPLIN
COUT     =    *
         RTS
```

Now the connect and disconnect code looks like this:

```
            JMP  CONNECT
;
;disconnect code follows
;
            JSR  ADDCHECK   ;see if vector already set up (connected)
            BNE  DERR       ;hasn't been set up, can't disconnect
            SEI             ;ok, set vector back to previous state
            LDA  OLDIRQ     ;..low
            STA  IRQVEC
            LDA  OLDIRQ+1   ;..high
            STA  IRQVEC+1
            CLI
            RTS             ;vector now set back to pre-connect address
;
CONNECT =    *
            JSR  ADDCHECK   ;see if already set up
            BEQ  CERR       ;already connected, error
            LDA  IRQVEC     ;ok, save old vector address
            STA  OLDIRQ     ;..low
            LDA  IRQVEC+1   ;..high
            STA  OLDIRQ+1
            SEI             ;now change actual vector (connect)
            LDA  #<DISPLIN  ;routine address low
            STA  IRQVEC     ;vector low
            LDA  #>DISPLIN  ;routine address high
            STA  IRQVEC+1   ;vector high
            CLI
            RTS             ;successfully connected
;
DERR        =    *          ;disconnect error entry
;
;"routine not connected yet" message or do nothing
;
            RTS
;
CERR        =    *          ;connect error
;
;"routine already connected" msg or nothing
;
            RTS
```

All that code is a lot of overhead, and is probably not worth using on a little program like DISPLIN in the example. But if you have several utilities that you wish to be able to install and remove at will, you may want to use it. It's still not a completely flexible vector-management system, though; you can only disconnect programs in the reverse order that they were connected, starting with the last one. For example, suppose you had connected four programs. That means that instead of the Kernal going directly to its usual program through the default vector contents, it first jumps to the start of program 1, which ends by jumping to program 2, etc. until program 4 finishes and jumps to the Kernal where the vector normally points. Once all four programs are connected in this way, you'd have to disconnect them in the order 4, 3, 2, then 1. It works a bit like a Last-in-first-out stack, where each newly-installed program goes on top, and only the top program can be disconnected from the chain.

The main problem with this stack-oriented vector management system is the problems that result when you try to connect or disconnect a program that isn't at the top of the stack, i.e. the last one connected. To avoid a crash in either of these instances, one further refinement can be added to the connect and disconnect portions: use a "connect flag" to indicate whether the routine is connected or not. Coupled with the address check, we now have a foolproof system. Attempting to connect an unconnected program will give an error, as will attempting to disconnect a program that isn't at the top of our "stack".

With the connect flag logic in place, the connect routine looks like this:

```
CONNECT =    *
            LDA  CONFLAG    ;see if already set up
            BNE  CERR       ;already connected, error
            LDA  #255       ;set connect flag
            STA  CONFLAG
            LDA  IRQVEC     ;ok, save old vector address
            STA  OLDIRQ     ;..low
            LDA  IRQVEC+1   ;..high
            STA  OLDIRQ+1
            SEI             ;now change actual vector (connect)
            LDA  #<DISPLIN  ;routine address low
            STA  IRQVEC     ;vector low
            LDA  #>DISPLIN  ;routine address high
            STA  IRQVEC+1   ;vector high
            CLI
            RTS             ;successfully connected
CONFLAG .BYTE 0             ;connect flag starts off at zero
```

The disconnect routine must still compare the vector–driven routine's address with the vector in case the routine is active, but not at the top of the "stack". It now looks like this:

```
JSR  ADDCHECK    ;vector pointing to routine"?
BNE  DERR        ;no, can't disconnect
LDA  CONFLAG     ;make sure previously connected
BEQ  EXIT        ;no, already disconnected
LDA  #0          ;clear connect flag
STA  CONFLAG     ;..before disconnecting
SEI              ;ok, set vector back to previous state
LDA  OLDIRQ      ;..low
STA  IRQVEC
LDA  OLDIRQ+1    ;..high
STA  IRQVEC+1
CLI
RTS              ;vector now set back to pre–connect address
```

More Housekeeping

In the above IRQ driven example, we took advantage of the fact that the Kernal saves and restores the A,X,Y and processor status registers, so the program didn't have to bother. In general, when intercepting most vectors these registers must be saved at the beginning of the routine and restored at the end. You can use the stacking sequence:

PHP: PHA: TAX: PHA: TAY: PHA

To save them, and

PLA: TAY: PLA: TAX: PLA: PLP

To restore.

The rule for an interrupt–driven routine is the same as when working with the stack: "Please leave these premises as clean as you found them." That's called good housekeeping, and it's essential when intercepting the Kernel.

If you want to skip the normal kernel routine altogether after your vector code completes, an easy way to do this is by pointing the final JMP in your vector program to the end of the normal kernel routine, right to the RTS or RTI instruction. In some cases, you may have to fix up the stack to a certain state when doing this, but that will of course depend upon the Kernel routine concerned.

The VECTOR Kernal Routine

On the VIC and 64, there is a special Kernal routine for vector management. It's called (appropriately) VECTOR, and is at $FF8D. This routine will perform one of two functions:

1) copy the system RAM vectors from $0314 to $032E into a user–specified memory area, or
2) copy vectors from a user–specified area into the system RAM vectors.

Calling VECTOR with the carry flag set (SEC: JSR VECTOR) causes the vectors to be copied to the address specified by the X and Y registers (X = address low, Y = address high), and calling it with carry clear (CLC: JSR VECTOR) writes from memory to the vectors. 26 bytes are affected by the VECTOR routine.

Using VECTOR gives you an easy way to save the original contents of the system vectors. This is especially handy if your program is going to be changing several vectors; first read the vectors, then use the VECTORS routine to put them somewhere in memory. Commodore recommends that you then change the copy of the vectors in memory to what you desire and copy them back to actually change the system vectors. If that was all VECTORS was good for, I wouldn't even mention it here, but it has a higher purpose in life.

Let's say you are setting up a major system which will change many vectors, and you want to be able to connect and disconnect them as in the above example. To save the old vectors' contents, call VECTOR and save them in some area of memory called, say, "OLDVECS". Your new IRQ routine would end with a JMP (OLDVECS), since the old IRQ vector is stored at that address. Going through the vectors in order, the "BRK" routine would end with a JMP (OLDVECS+2), the NMI routine with a JMP (OLDVECS+4), etc., up to the warm–start routine (vectored through $032E) which would end with a JMP (OLDVECS+26). When you want to disconnect the entire system, just call VECTOR with the carry flag clear to copy the old vectors back into place again. The whole messy process becomes simple and elegant.

Conclusion

Most of the more difficult programming challenges that involve changing the behaviour of the computer in some way can be accomplished by using vectors. Whether you use the above vector management system or not, you should keep your program flexible so that it can be expanded, and removable so that it can be turned off if desired. Most important, it should "clean up" after itself to avoid any ill side effects on the system. The more vector–driven programs you write, the more you'll find that nothing's impossible when it comes to programming your Commodore.

SAVE @ Exposed: The Debate Continues

SAVE@ Composed

I have run two tests using the SAVE@ EXPOSED!!! program. The first test was with the program as written, using a disk about 25% full. The program ran for one hour and was then stopped. An examination of the files indicated that nothing was disturbed.

Test two involved a disk with 94 free blocks and was run with 10 filenames as you indicated in the article. This was allowed to run for 1.5 hours. Again, there were no files disturbed.

I am using the newer 1541 disk drive with the lever that closes over the disk slot. The drive serial number is AJ1032352. Could there be something different in the ROMs of the new drives?

One final note: I made sure that the program names in the data statements normally loaded with the LOAD"name",8 command and not the LOAD"name",8,1 command. This could cause problems, could it not?

Ray E. Striker, Lincoln, Nebraska

A program that normally loads to BASIC text space (start address = $0800) can be loaded either way. In fact, a rule of thumb around the Transactor bullpen is "if in doubt use comma one". This way programs will always load at the start address specified by the file. If the address is $0800 it will load there; if it isn't 0800, chances are it wasn't supposed to load there anyway.

SAVE@ Exhausted!!

One can hardly ignore a development as important as the demonstration of the save-with-replace bug. You have certainly stirred the pot. While I cannot comment on Mr. Whittern's 1541 version of the program, I have tried to duplicate the Transactor results on my 4040. The program, as presented, did no damage to my two test disks. One was a fairly new disk which had only the test programs saved on it. The other disk was a controlled mess: barely enough room to replace a file with, and the available sectors were split into several areas; furthermore, some of the available sectors fell on a "zone" boundary (the place where the sector count changes from one number to another). What more can I do short of having asterisk files?

Various versions of the program, including one using BSAVE and BLOAD on the B-128 machine (much easier to work with), also failed to damage the disk in several hours of trying. Not only were the test files intact, but all the original files that were on the floppy compared correctly, byte for byte, with its duplicate made prior to the tests.

While, by definition, I CANNOT PROVE NON-EXISTENCE of a bug, I am unable to duplicate your results based on the scant data presented in the editor's notes. The burden to prove the bug is with the Transactor. We need more information, the most important being the history of the floppy you used, the exact files that were used, and the exact order of doing things. Unless we can duplicate your results, the bug remains a mystery.

Liz Deal, Malvern, PA.

SCRATCH & SAVE Exposed!!

I have just finished reading "Save With Replace Exposed!!" in your latest issue. Of course I had to try it immediately. I got pretty much the same results as described in the text.

Then, out of curiosity, I decided to try the alternative (using scratch and save instead of save @). I changed lines 170 and 200 to read:

```
170 print d4$"open 1,8,15,"qt$"s0:"a$(i)qt$"close1
    :save"qt$"0:"a$(i)qt$",8"
```

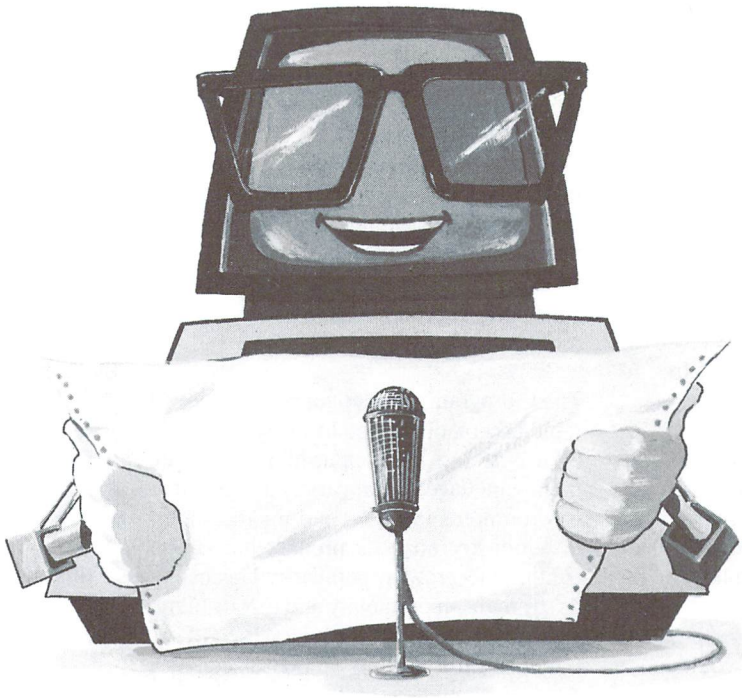
Do the same for line 200, changing a$(i) to a$(j).

Guess what! Results are the same. Where do we go from here? Can you get your champagne back?

Sheldon C. Wotring, Palmerton, PA

Seems the SAVE @ situation is certainly not over. The responses above are typical of several that we have received, written and verbal. Although evidence subtantiating the existence of a bug has been tested true, it now seems we may be hunting the wrong bug. Could the problem be unit dependent? If so, a mechancal discrepancy would have to be the culprit as units are electronically identical, aren't they? The only controlled experiment to test this theory would be to have enough disk drives together running the same test under the same conditions. Hopefully there would be enough machines so that several would show the problem occurring and several would not. Once they're seperated they could be mechanically compared. However this could be a lot of work that, theoretically, might prove nothing. If things like motor speed and head positioning are identical, what then? And who, besides Commodore (hint hint), has enough units to conduct such a test. Until then we'll be sleuthing away and we'll no doubt have more in issues to come. – M.Ed.

News BRK



Transactor News

Delivery Information

Because subscriptions, disks, books, etc. are sent out in batches, it is possible that you won't get your order for six to eight weeks. Disk orders are handled by an outside supplier, so your disks won't necessarily come at the same time as your magazine — you should get both within a few weeks of each other. The same goes for any items ordered (eg. magazine and disk subscriptions, back issues and disks, and the Complete Commodore Inner Space Anthology) — you may not receive them in one shipment. We apologize for any confusion this may have caused.

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.

Transactor on Microfiche

We are looking into the possibility of putting Transactor issues on microfiche. If there is enough interest, we will go ahead, so let us know if you think it's a good idea.

Inner Space Anthology Discounts

The Complete Commodore Inner Space Anthology is now just over 3 months old and closing in on a second printing. Book stores reluctant to take 3 or 4 copies are submitting second orders of 12 or 20, often just days after their first shipment! If your local book retailer could stand an extra sale or two, the CCISA can be ordered from wherever they usually get computer books and magazines. For orders of 64 (32 per carton) or more, they can be ordered directly from us at the following discounts:

64-250 - 44% OFF
250 up - 50% OFF

Suggested retail price is $16.95. Prepayment is required on the initial order, with terms extended accordingly after that, no returns. For orders substantially higher, contact us in Milton.

Dry Mail

In the last issue's article "Save @ Exposed", we promised author Charles Whittern of Hudson, MI, a bottle of champagne for meeting our requirements of proving the existence of the obscure disk drive bug. Unfortunately, in the state of Michigan, it is not legal to send alcohol by mail. Even through a local delivery service we found the request could not be met. It seems the state cannot be sure the sender and/or the receiver is of age. So instead we sent a card and a cheque, and unfortunately the Champagne offer no longer stands.

Computer News

C-64 Users Group Of Canada

Montreal— Commodore 64 computer users can now take advantage of a wide range of unique services offered by the C–64 Users Group Of Canada, established February 1, 1985, in Montreal, Quebec. Services include no–frills software offered monthly for $4.95. Every month, the club will offer diskettes filled with software from the COMAL Users Group USA Ltd, Educational software from Commodore Canada, and a collection of the highest quality public domain (non–commercial) software.

For a yearly membership fee, members will receive a monthly club bulletin. An electronic bulletin board service is accessible to those members with modems. Benefits of becoming a member also include discounts at local retailers and computer repair cen-

ters. The club also offers a hotline for computer questions, and many other useful services to Commodore 64 computer users.

And The Winner Is. . .

While many people dream of writing a hit song, one musician has made his dream come true in an unusual way. He won the first annual Computer Song Writing Contest. His song, "Melting pot", was chosen by a panel of music industry judges as the best in composition, melody, and use of sound. This talented musician received EnTech's grand prize of $1,000 and free recording time at a Hollywood studio.

As part of the awards presentation, Bones jammed with EnTech's new keyboard version of Studio 64. As a synthesizer player, he commented that "the keyboard version is a lot less limiting than typing on the computer". He also remarked that "I was surprised that I can create the same quality of music on Studio 64 as I could on a synthesizer. The versatility is there." EnTech included as part of Bones' prize a keyboard and a copy of the advanced version of Studio 64.

Bones plans to use his winnings to purchase new musical equipment, which he will use for his studio recording session. EnTech will distribute copies of his song to radio and television stations throughout the country. He called EnTech's Studio 64, "A great new advance in the music world" and hopes his contest victory would "allow the public to hear my songs".

This contest was sponsored by EnTech to promote computer music as an art form. EnTech's Public Relations Representative Matthew Stern announced that plans are underway for a second contest in 1985.

People interested in entering should contact:

EnTech Software,
P.O. Box 185,
Sun Valley, CA
91604 818 768-6646

C Experts Gather For Fall Seminar

Many of the world's most prominent C language experts will speak and conduct workshops at a technical seminar in Cambridge, Mass. this fall.

The publishers of COMPUTER LANGUAGE, a leading monthly programmer's magazine, will sponsor this technical event from September 16-18. The seminar will be held at the Sheraton-Commander Hotel in Cambridge, Mass., and will cover a wide variety of practical subjects on programming in C.

"Never before has so many key leaders in the C programming community gathered for one event." said publisher Carl Landau. "The focus of the speaker and workshop sessions will be on the latest practical strategies and techniques used in C programming today."

The C programming language has enjoyed rapid acceptance in the industry as a powerful systems and applications development language for a variety of machine environments in mini- and microcomputers. Industry analysts predict that C will continue to grow in popularity because of its strengths in portability and extensibility.

C has also just become "standardized" by a special committee of the American National Standards Institute (ANSI). The chairman of this committee, Jim Brodie, and his four subchairmen will be speaking at the seminar on the state of the C language today.

Among the list of speakers confirmed for the event are: Jim Brodie, ANSI C Standards committee chairman; P.J. Plauger, co-author of Elements of Programming Style and ANSI C secretary; Leor Zolman, compiler writer and public domain C expert; Heinz Lycklama, chairman of the /usr/ group UNIX Standards group; Robert Ward, coordinator of The C User's Group; Tom Plum, author of Learning to Program in C; and Larry Rosler, ANSI language sub chairman.

In addition to the topics these speakers will address, the seminar will also feature programming workshops on a variety of subjects including "Porting C Across Operating Systems", "Using C Libraries", and "Structured Debugging Techniques."

The attendance fee for the seminar is $695. Early bird registration is being offered at the reduced rate of $595. The deadline for early registration is June 30, 1985.

COMPUTER LANGUAGE is the first and only magazine dedicated to programming languages and software design. It is written for advanced programmers and engineers who write their own software. The monthly publication has a paid circulation of over 30,000 copies around the world.

For more information about the seminar, write to:

Beatrice Blatteis or Cal Landau,
CL Publications,
131 Townsend Street,
San Francisco, CA
94107 415 957-9353

International Communications and Computer Exhibition

Tracon Exhibitions is pleased to announce that planning is well underway for the INTERNATIONAL COMMUNICATIONS AND COMPUTER EXHIBITION. This prestigious exhibition will be taking place during EXPO '86, Vancouver's world fair on transportation and communications. The dates of the exhibition, September 9 - 11, 1986, ensure it to be a featured event of Communications week, one of fourteen specialized periods of EXPO '86.

Of special interest to potential exhibitors in EXPO '86's official endorsement of the INTERNATIONAL COMMUNICATIONS AND COMPUTER EXHIBITION. Tracon Exhibitions, a Vancouver based producer of major trade an consumer exhibitions, has been recognized by EXPO '86 officials as having the expertise and imagination to put on a show of this stature.

The endorsement by EXPO '86 entitles Tracon Exhibitions to use the fair's logo on all promotional material pertaining to the exhibition and more importantly, to benefit from EXPO '86's international high profile. It is expected that visitors with particular interest in communications and computers will visit EXPO during Communications Week and thus attend the exhibition as well.

The dual theme of the exhibition acknowledges the ever increasing relationship between the once separate technologies of communications and computers. Writers and social commentators everywhere are hailing the marriage of these two technologies as being one of the most profound developments of this century. New shifts in the corporate strategies of major manufacturers reinforce the relevance of this theme. No doubt major shake-ups will continue to occur between now and the exhibition in 1986. Because of these developments, the exhibition's audience can be assured of many exciting displays of the latest communications and computer technology.

The stature of the exhibition, its timeliness, and its location in B.C. Place Stadium, just steps away from the site of EXPO '86, provides worldwide manufacturers and distributors of communications and computer products a unique opportunity to unveil new wares to a well qualified international audience. Tracon Exhibitions is planning an extensive advertising and promotion campaign to ensure that both buyers and users of communications attend the show.

Government agencies and trade associations from around the world are currently being contacted to obtain their support and endorsement of the show. Plans are to offer educational show features, displays and seminars in cooperation with these groups.

It is especially fitting that Canada be the location of this special event. As a world leader in satellite technology, videotex, telecommunications and mobile communications, Canadian companies can proudly display their products in an international arena. In addition, Western Canada's proximity to an increasing trade with Pacific Rim countries will no doubt signal active participation by leading edge companies from that region.

For more information contact:

International Communications and Computer Exhibition,
#202 – 535 West 10th Avenue,
Vancouver, B.C.
V5Z 1K9 604 874-5233

General

New Markers Write Safely On Diskettes

Two new types of markers designed specifically for use on computer software have been introduced by Sanford Corporation.

The growing need for these markers is evident in the widespread use of disks and diskettes. Nearly 1 billion floppy disks will be sold in 1985. It is expected that this figure will more than double to 2.1 billion by 1987.

One of the new markers called Disktribe can be used directly on computer software disk sleeves for safe identification and reference.

The ink is quick-drying and permanent. Tests verify that the markings do not affect information on the disk itself.

Availability of two distinct Disktribe colours – one silver and one gold – enables users to colour-code disk markings to identify different kinds of data.

Sanford Corp. is a 128-year-old producer of pens, markers, rubber stamps and adhesives. Its headquarters are Bellwood, IL. For more information contact:

The Philip Lesly Co.
130 East Randolph St.
Chicago, IL
60601 312 565-1900

Software News

Paperback Writer

What you see is what you get! That's the promise of an easy-to-learn, inexpensive word processor released this spring by Digital Solutions Inc. of Toronto.

With Paperback Writer 64, no longer will you have to guess what a page will look like when printed. Set margins and paragraph indents, decide whether to justify or center text, — and see it on the screen. Set page lengths and numbers — and see exactly where new pages begin. Use boldface, italics and underlining — and watch the text change in front of your eyes. No fancy codes to memorize that clutter up the screen; no words broken up at the end of a line.

Neither do you need a bulky, complicated manual typical of software "documentation". Pressing one key gives you in-depth help on the screen, first from memory, and, for more detail, from the program disk. For sophisticated features, an easy-to-read Reference Guide explains everything.
Paperback Writer 64 lets you work in 80 columns — with no additional hardware — instead of the Commodore 64's standard 40 columns. A spelling checker is also included.

In addition to standard word processing features such as search-and-replace, Paperback Writer 64 lets you edit sequential files. Also, files from most other popular word processors can be directly loaded, appearing properly formatted on the screen.

Digital Solutions says Paperback Writer 64, programmed by David Foster, will be adapted for the new Commodore 128 for release in June. The company is also working on database, spreadsheet and communications programs to complete a sophisticated business package. Its philosophy is to produce high-quality software that's easy to use and costs under $50.

The suggested price for Paperback Writer 64 is $39.95 U.S. Contact your local dealer in June for a demonstration. Or write:

Digital Solutions Inc.
P.O. Box 345, Station 'A'
Willowdale, Ontario
Canada M2N 5S9

Sixth Sense

Microtechnic Solutions, Inc. announces SIXTH SENSE, the Thinking Terminal for the Commodore 64. SIXTH SENSE sets new standards for home modem software, allowing users total control of terminal actions, even when they're not home! The software can originate calls to, or answer calls from, remote computers, gathering and dispensing information automatically based on user instruction. SIXTH SENSE uses a macro language that allows it to make decisions and perform actions based on user-programmed parameters such as time of day, an internal counter, and external events – all preset by user-defined macros or with one of the Macro Templates included.

Other significant features include: a 700 line virtual screen with bi-directional scrolling, a standard screen for use with the

built-in screen editor, split-screen line input, flexible data routing between devices - including simultaneous downloading to screen, printer and disk at 1200 baud, independently controlled transmit and receive translation, and error-free file tranfers with XMODEM and CompuServe B protocols. SIXTH SENSE contains over 150 functions available to the user.

The internal sophistication of SIXTH SENSE makes it easy to operate while providing the user with greater control and better performance than other modem programs. SIXTH SENSE is supplied on disk with a comprehensive user guide, and you can back-up the program. Suggested retail is $89.95. A 20% discount is offered to registered owners of Smart 64 Terminal.

Available from:

Microtechnic Solutions, Inc.,
P.O. Box 2940,
New Haven, CT
06515 203 389-8383

'C POWER' For The 64

Pro-Line Software Ltd. is pleased and excited to announce the release of their latest new software product in the Utilities/Languages field. C POWER is a fully implemented Kernihan and Ritchie version of the "C" language for the command interpreter, EDITOR, SYNTAX CHECKING EDITOR, COMPILER, LINKER, MATH LIBRARY, STANDARD LIBRARY and SYSTEM LIBRARY. C POWER compiles directly to native 6510 machine code, and does not require the extra overhead of some C compilers that compile to "P" code or use some other intermediate step. C POWER is a serious cookie.

Documentation for C POWER includes a User's Guide, Library Description, Library Listing and Cross-Index, and the 531 page C PRIMER PLUS "User Friendly Guide to the C Programming Language" by Mitchell Waite, Stephen Prata and Donald Martin.

C POWER joins PAL 64, POWER 64 and TOOLBOX 64 in Pro-Line Software's best selling line of Utility and Language software programs for the Commodore 64.

Pro-Line Software is probably best known for their long running best-seller, the WordPro Series of word processing software programs for Commodore computers written by young Canadian Steve Punter. The WordPro Series was first marketed in 1978, and has since surpassed the 1.6-million-in-use mark despite being the most copied and closely emulated software product since Visicalc. The latest word processing technology for the Commodore 64 is embodied in the all new WordPro 64, which is now reported to be selling 6 to 1 over all comers.

For further information, please contact:

Pro-Line Software Ltd.,
755 The Queensway East,
Unit 8,
Mississauga, ONT.
L4Y 4C5 416 273-6350

Editor's Note: In our "Languages" issue, we will be reviewing this package from a users stand point. To further enlighten our reading audience, the finer points of programming in C will also be discussed at length. Hope you can wait.

PROMAL - New Structured Programming Language

PROMAL (PROgrammer's Micro Application Language), a new high-level structured programming language (similar to "C" and "Pascal") is now available for the Commodore 64.

PROMAL includes a one-pass compiler, a full screen editor, a command executive and a library of pre-defined utility subroutines.

PROMAL was designed for programmers at all levels of expertise. It has a fast compiler and a highly efficient run-time environment that permits applications to be written in a high-level language which prior to PROMAL had to be written in assembler for performance reasons. The Executive (operating system) provides file, memory and program management, and I/O redirection. A full-screen, cursor driven Editor permits rapid source program entry and editing. The Library of machine language subroutines supports the run-time environment with optimized routines for file I/O, string handling, formatted output, cursor control, and data conversion. PROMAL comes with a comprehensive 210 page reference manual.

Benchmark results which show that the Commodore 64 version of PROMAL is from 70% to 2000% faster than BASIC, COMAL, FORTH, and PASCAL are available upon request from SMA.

PROMAL is available direct from SMA at $49.95 retail. The Developer's Version, which includes an unlimited run-time distribution license, is $99.95. Both come with a money-back guarantee.

For further information, please contact:

Jennifer L. Conn, Vice President,
Systems Management Associates,
3700 Computer Drive,
Raleigh, NC
27609 919 787-7703

Freedom Assembler/Monitor

The Freedom Assembler/Monitor is an easy to use, fast and powerful, cartridge based symbolic assembler for 6502/6510 and 65C02 based systems. It is activated only when you need it. It can remain plugged into your computer and inactive (blanked out on the C-64) until you SYS it to attention.

If you are tired of loading and unloading numerous files just to get your machine code operable, this is the assembler for you. A simple SYS command and you can assemble, disassemble, walk through, and execute your assembly code in a flash.

The Freedom Assembler uses the resident editor so source code can be loaded, edited, listed or saved quickly and easily, at any time.

Also included in the Assembler are these features:

Written 100% in Assembly Language for very fast assemblies.

- Cross-assembles 3 other related instruction sets including CMOS 65C02 (disassembly and walk simulation included). Excellent for industrial applications.
- Full assembly listing printout capability including an alphabetical cross-reference list.
- 9 types of assembly error messages.
- Save command for machine code.
- Disassembler that displays opcodes, operands, and raw hex code.

The powerful built in Monitor program allows you to walk through and debug your code.

The Monitor comes with these commands:

Disassemble	Walk machine code
Transfer memory	Set/clear breakpoint
Compare memory	Register display/edit
Fill memory	Quicktrace

Checksum memory Go–execute code
Memory display/edit Save machine code
Renumber source file
Hunt for hex or ASCII
Bank out RAM (C–64 only)
Bank in RAM (C–64 only)
Change instruction set mode
List cross–reference info.
Print cross–reference list
Kill assembler (return to normal operation.)

The Freedom Assembler/Monitor is available for either the VIC–20 or C–64. Both the assembler and monitor are contained in the same cartridge for ease of use and execution. The FREEDOM ASSEMBLER/MONITOR and a 60 page instruction manual with sample program are available for $39.95 (U.S. funds. Shipping is included in price.) Order directly from:

Hughes Associates Software,
45341 Harmony Lane,
Belleville, MI 48111

Sight & Sound Expands Floppies To Flippies

Sight & Sound Music Software, Inc., New Berlin, Wisconsin, is turning two of its popular floppy disk programs into flippies! All the existing features of Kawasaki Rhythm Rocker and the Incredible Musical Keyboard have, in each case, been condensed to one side of the disk and sensational new bonus programs have been added to the other side at no extra cost to the user.

Kawasaki Rhythm Rocker now incorporates a number of additional features which were most frequently requested by users. A new notation system is included that enables everything the user plays to be shown on the screen. Also, to further enhance the ability to create, the bonus program can record and overdub up to 750 notes. To complete the package, the new program also includes a score printer which enables the user to print out his or her original composition.

Other new functions include the ability to change voices while playing, transpose to any of 12 keys and use a multitude of specially created effects. And for those users who aren't perfect, there's even an auto–correct feature. Any notation played out of time will be corrected to the exact beats, visually, on screen.

The new double–sided Rhythm Rocker, containing both original and bonus programs, will retail for $34.95 (the price for just the original, alone). Or, owners of the original Rhythm Rocker can upgrade to the complete new program for just $15.00.

The Incredible Musical Keyboard bonus program now allows for recording songs using up to three voices. A new notation and graphics program has been added and actually displays the notes on the musical staff. There are now five background accompaniments to choose from ranging from reggae to samba, and the user can change these accompaniments both while playing and recording.

The Incredible Musical Keyboard bonus program also allows the user to select from over 20 instrument sounds which can be played in any of six octaves. The instrument sounds, as well as the octaves, can be changed at any time while playing or recording.

The new Incredible Musical Keyboard package, including the new double–sided program, documentation, keyboard overlay and two music books, will retail for $39.95. Owners of the original program may upgrade to the new one for just $7.00.

These new super flippy programs will be available early in May; the price will remain the same as the previous single–sided programs. And for those who own the originals and want to upgrade, simply send proof of ownership and a cheque or money order made out to Sight & Sound Music Software, Inc., P.O. Box 27, Department R2D2, New Berlin, Wisconsin 53151. Ownership of the original may be proven by either returning the disk or cutting out the UPC and ISBN product numbers from the packaging. Please allow two to four weeks for delivery.

Keyboard Chord/Scale Master

Valhala Software of Fernadale, Michigan introduces the Keyboard Chord/Scale Master, "the contemporary approach to learning" through educational music software. Development of this approach has taken considerable time to research and evaluate prior to this release.

The Keyboard Chord/Scale Master is designed to enhance the user's keyboard abilities through sight and sound reinforcements of CHORD and SCALE DISPLAY MODES, CHORD and SCALE REVIEW MODES, and a COMPARE MODE. Each mode allows the user to choose up to twelve key signatures for reviewing.

CHORD DISPLAY and SCALE DISPLAY MODES allows the student/users to view and sound the most commonly used chords and scales. The keyboard's creativity can be enhanced by learning chords, inversions, and scales from these modes.

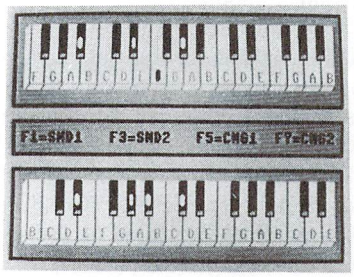
Both the CHORD REVIEW and SCALE REVIEW MODES have three different approaches to learning:

1. DRILL MODE functions by having the computer randomly select any number of chords and/or key signatures to study which were previously chosen by the users.

2. QUIZ MODE functions are the same as the DRILL MODE, with one exception, it displays at the bottom of the screen a continuous changing percentile as determined by the number of correct answers.

3. COMPETE MODE is a competitive game of challenge for the classroom or home. This approach displays team or individual cumulative scores throughout the game.

A COMPARE MODE which displays two keyboards is another feature. This mode allows the user to compare one chord with another in the same or different key signatures. The COMPARE MODE again compliments Valhala's approach to learning the keyboard through sight and sound reinforcements.



The Keyboard Chord/Scale Master, in diskette form is adaptable for colour or monochrome displays and is available for the Commodore 64. The screen displays are crisp and exacting with 3–Dimensional keyboard effects. Coloured indicators show the chord and scale positions on the keyboard; computer audio sounds support the chord or scale being displayed.

This program, written in machine language is Valhala's first Contemporary Educational Software release. Next to follow in the near

future will be the Guitar Chord/Scale Master, another prestigious program written for learning with fun for the users being kept in mind. Valhala is proud to be a part of the software industry working toward the revolution of music learning through the aid of the computer.

Whether you are learning the piano, organ, or the latest electronic synthesizer, be among the first adventurous enough to challenge our new approach to learning. The Keyboard Chord/Scale Master is priced at only $39.95; include $1.50 for postage and handling charges. Michigan Residents include 4% sales tax.

For further information, please contact:

Valhala Software,
205 East Hazelhurst,
Ferndale, MI
48220

Light Pen Reading Series from MicroEd

MicroEd Incorporated, a Minneapolis-based publisher of educational software, has begun marketing a light pen/microcomputer series of 80 program sets designed to provide students with beginning word attack skills that are normally taught in kindergarten through third grade. Entitled Point & Read, the series is also expected to become popular with educators working in the field of special education. Using a light pen as a response device for individuals with learning disabilities of one kind or another.

Initially available for the Commodore 64, the series can be purchased in four packages having a combined total of fourteen disks. The suggested retail prices for these packages are $74.95 for Package One (3 disks), $74.95 for Package Two (3 disks), $99.95 for Package Three (4 disks), and $99.95 for Package Four (4 disks).

Of particular interest to students are the eight hundred full-screen computer-drawn pictures used to illustrate the various problem sentences within the series.

Persons desiring more information may call toll free 1-800-MicroEd, or contact:

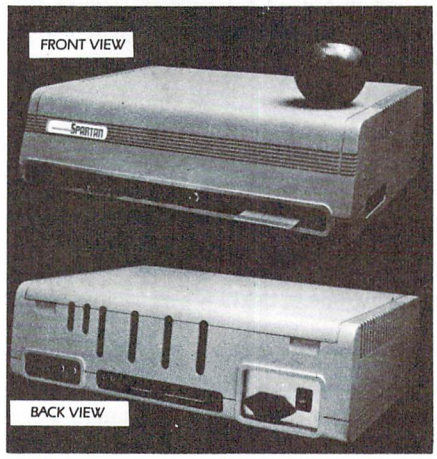
George Esbensen,
National Sales Coordinator,
MicroEd Incorporated,
P.O. Box 444005,
Eden Prairie, MI
55344

Hardware News

C64 & APPLE II+ Compatibility

Mimic Systems Inc. of Victoria, BC has a line of products which give a Commodore 64 Apple II+ compatability.

The "Spartan" gives 100% Apple II+ software and hardware compatability, allowing use of Apple or C-64 peripherals and generating both C-64 and Apple II+ video outputs. Eight Apple II+ slots and four Commodore 64 slots are provided on the Spartan, along with an extra 8-bit parallel I/O port. Suggested retail for the Spartan is $599.00 (US)



The "BUSS Card" supplies ten standard Apple II+ peripheral slots and four software-selectable Commodore 64 cartridge slots, along with an extra 8-bit parallel port. The BUSS Card also contains an Apple II+ switching power supply. The BUSS card retails for $299.00 US.

The "CPU Card" contains 64 K bytes of RAM, a 6502A, and all standard Apple II+ game, cassette and keyboard I/O ports. The Bus card can be upgraded to a 65816 16-bit processor. Suggested list is $199.00 US.

The "DOS Card" allows a Commodore 1541 disk drive to read and write all standard Apple II+ disks. Either Commodore or Apple II+ format may be selected by software. The BUSS Card accepts all standard Apple II+ disk interface cards. Retail is $199.00 US.

Mimic Systems Inc.
1112 Fort Street, 6th Floor
Victoria, B.C.
V8V 4V2 1-800-663 8527

Master Software Releases Reset Master

Master Software of Randallstown, Maryland, has introduced Reset Master, a system reset switch for the Commodore VIC-20 and Commodore C-64 computers.

Reset Master will reset your computer without shutting off the power, and will restore control of the computer to the operator in case of computer lock-up. Four RENEW programs are included to restore the BASIC program that was in memory before using Reset Master.

Reset Master simply plugs into any serial port on your computer system, and therefore is installed without opening the computer case and without any soldering. Rather than just shorting the computer's reset signal line to ground, Reset Master contains electronic circuitry to protect your computer.

Other important features of Reset Master include a two-foot cord, which acts as an extension cord on your serial bus, and two female serial ports, enabling the use of two printers. Suggested Retail $24.95 postpaid.

Master Software,
6 Hillery Court,
Randallstown, MD
21133

ABL-64 (Automatic Boot-Loader Cartridge for the 64)

Computer Bulletin Boards, Security Systems, or other "Constant Use" applications for the Commodore 64 computer will find immediate and handy use for this package. ABL-64 will re-boot and run an essential program after a power failure, even if the computer is left unattended.

ABL-64 is inserted into the Commodore 64 expansion (cartridge) port, and forces the Commodore 64 to re-LOAD and RUN after a power failure.

When ABL-64 is installed and there is a power failure, ABL-64 is activated the in-

stant power is restored. There is a timer aboard which counts up to 15 seconds. During that time, if an operator is present, he may invoke manual control over the program through the keyboard. With no operator present, after 15 seconds, ABL-64 BOOTS and runs a pre-selected program from the disk. It can pick up where it left off before the power failure.

The package includes cartridge, instruction manual, and listings of two usable utility programs. The suggested list price is $39.95.

This is a very inexpensive alternative to battery power backup systems. Available for immediate delivery.

For further information, contact:

Jack Weaver, President,
Input Systems, Inc.,
15600 Palmetto Lake Drive,
Miami, FL
33157 305 252-1550

Studio 64 Gets New Keyboard

One of the road blocks in creating music on the Commodore 64 has been its keyboard, since it is designed for typing, not playing. This has made playing the Commodore 64 like a musical instrument in real time a cumbersome task. However, EnTech Software and Sequential Circuits have teamed up to form what will be the next generation of home computer music.

EnTech Software has made its Studio 64 music synthesizer program compatible with the Sequential Circuits keyboard. This keyboard has thirty-two full sized piano keys and plugs into the Commodore 64's joystick port. According to musician and EnTech chairman Ray Soular, "The keyboard has the same feel and playability of a professional synthesizer."

With the keyboard, Studio 64 turns into a three track recording studio. Music for each of the three voices is played live and recorded in true musical notation. As a person plays one voice, the other voices are played back at the same time. A built in metronome uses a click track and changing border colours to help keep the musician on beat, and the tempo can be adjusted to any of nine settings. An auto correct function rounds off notes to the nearest sixteenth note, so even novice keyboard players play with precision.

After it is played, the music can be edited in true musical notation, including tied notes and sharps and flats. Blocks of music can have their octaves and waveforms changed without reentering notes. Full editing features allow people to duplicate or move sections of music, and enter and delete notes individually. Studio 64 has also added ring modulation and sync to create a variety of sound effects.

The keyboard also gives Studio 64 additional educational value. Studio 64 is already being used in schools for teaching music composition, sight reading, and vocal and keyboard training.

Studio 64 still uses the Commodore 64 keyboard to enter notes or play in real time. The new metronome, block changing of octaves and waveform sounds, and ring and sync sounds are available in both versions.

Soular described the Sequential Circuits keyboard version of Studio 64 as a great advance for computer music. "When I originally designed this program, this was exactly the type of music system I had invisioned. The keyboard makes the Commodore 64 a true musical instrument and permits a degree of creativity that hasn't been possible before. With Studio 64, people can create performance quality music."

Both the Commodore keyboard and the Sequential Circuits keyboard versions are available on the same disk for $39.95. Current Studio 64 users who want the updated version can send $10 and their old disk to Entech (see earlier item).

Black Box Modem 1200's at Half Price

In a modem haze? Black box has made the buy decision easier by offering its new Modem 1200's at half their catalog-listed prices — less than $300/unit. Only $279 for the auto-dial Modem 1200 and $249 for the manual-dial version.

The auto-dial Modem 1200 is easy to use with a "dumb" terminal or PC running a sophisticated communication software program. You can access a broad range of built-in HELP commands and menus in the terminal Command mode. Auto-dial 1200 has built-in, non-volatile memory that stores up to 20 telephone numbers of up to 55 characters each. You can mix tone and pulse dialing with one number, perfect for use with long-distance services like SPRINT and MCI. You can even assign easy-to-remember names like LAB and OFFICE to every stored number.

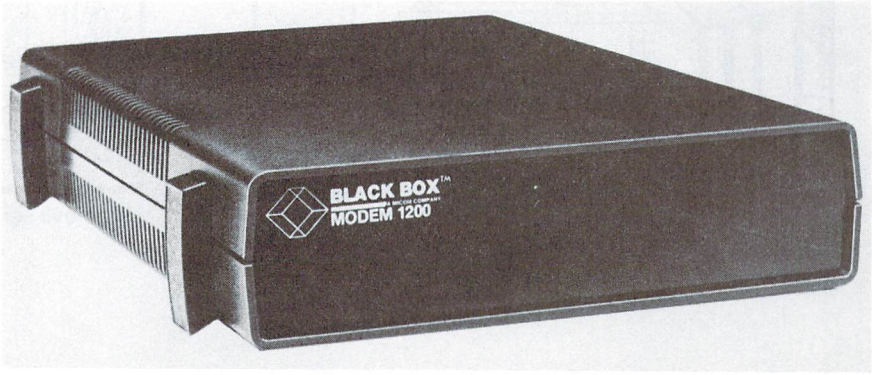
Unlike many other units, Auto-dial 1200 does not require sophisticated command sequences. Most commands are single-key driven.

The Manual-dial Modem 1200 is an economical alternative for users who do not need auto-dial capacity, especially in applications where the modem's primary use is for answering incoming data calls.

Both Black Box modems are Bell 103/212A compatible, and at these low prices are exceptional values. For more information contact:

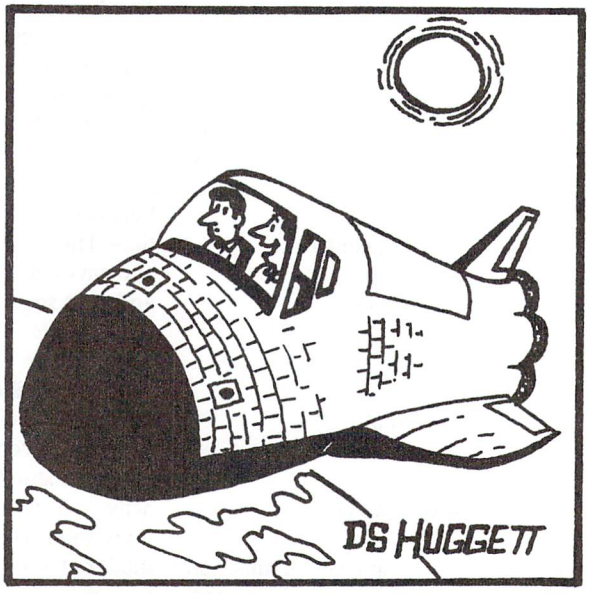
Modem 1200
Black Box Corp.
Box 12800
Pittsburgh, PA
15241 412 746-5500

(Looks like the price of 1200 baud modems is finally starting to come down. - M.Ed)



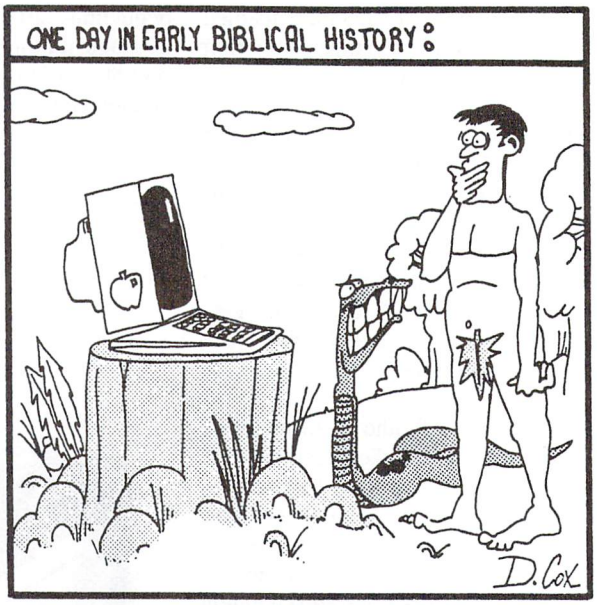
Compu–toons

WARNING: Staring at your computer screen for long periods of time may cause nightmarish hallucinations.



Have you noticed we haven't had any computer malfunctions since the Commodore 64 was installed?

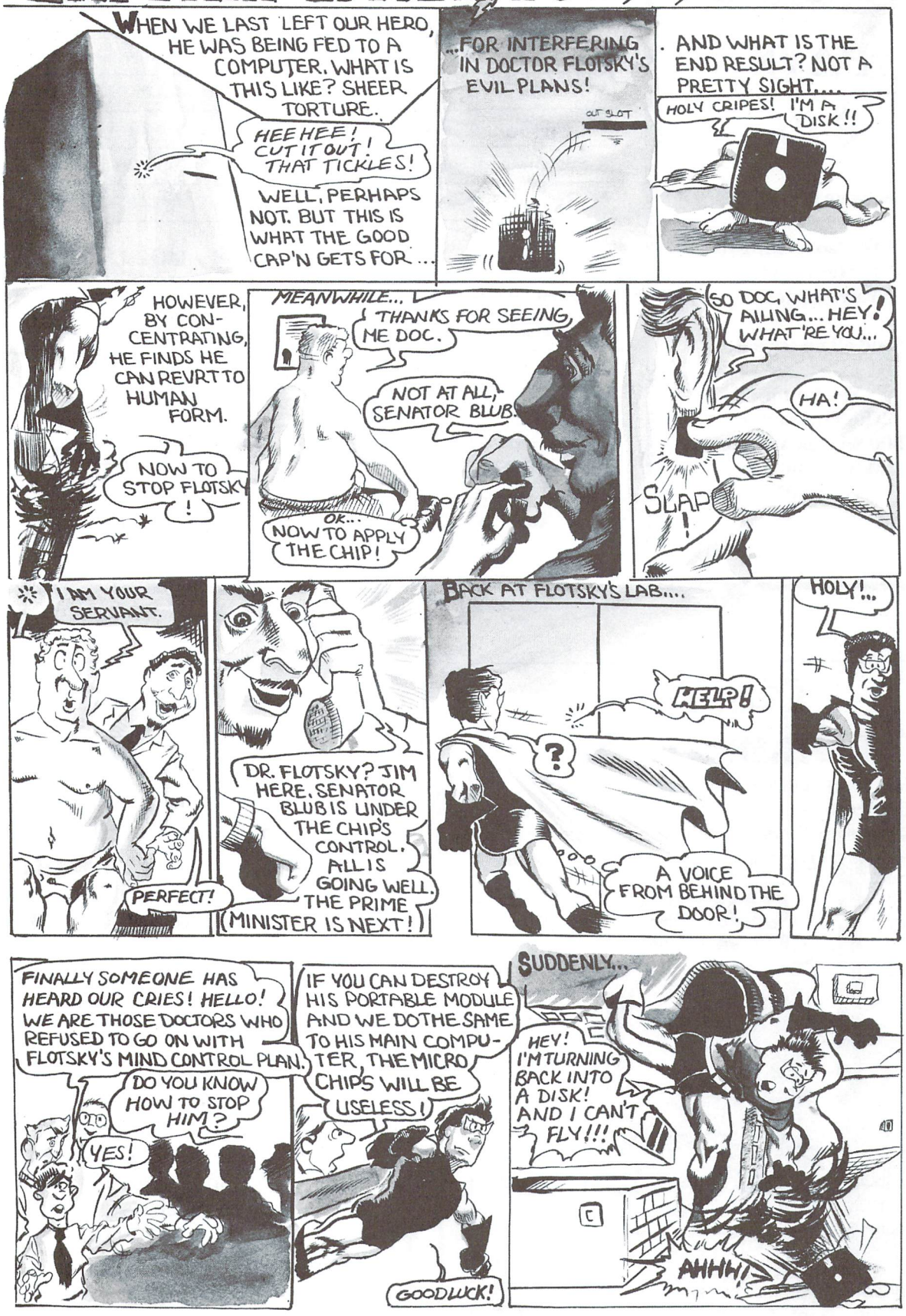
"What's this Bob? Another one of your silly jokes - 'Give me a banana or I'll peel your face off' - very funny Bob.



The evil serpent leads Adam to the forbidden Apple

CAPTAIN SYNTAX

©1985 by DAN SLOAN



The Transactor
The Tech/News Journal For Commodore Computers

PAYS $40

per page for articles

We're also looking for professionally drawn cartoons!

Send all material to:

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

Volume 6 Editorial Schedule

Issue#	Theme	Copy Due	Printed	Release Date
1	More Aids & Utilities	Feb 1	Mar 22	April 1/85
2	Communications & Networking	Apr 1	May 24	June 1
3	Languages	Jun 1	Jul 26	August 1
4	Implementing The Sciences	Aug 1	Sep 20	October 1
5	Hardware & Software Interfacing	Oct 1	Nov 22	December 1
6	Real Life Applications	Dec 1	Jan 24	February 1/86

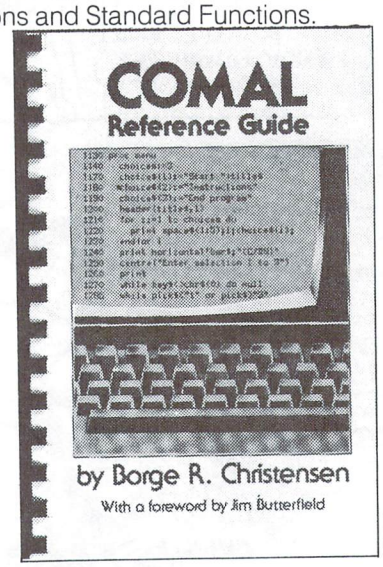
Volume 7 Editorial Schedule

1	ROM Routines / Kernel Routines	Feb 1	Mar 21	April 1
2	Games From The Inside Out	Apr 1	May 23	June 1
3	Programming The Chips	Jun 1	Jul 25	August 1
4	Gadgets and Gizmos	Aug 1	Sep 26	October 1
5	Simulations and Modelling	Oct 1	Nov 21	December 1
6	Programming Techniques	Dec 1	Jan 23	February 1/87

Advertisers and Authors should have material submitted no later than the 'Copy Due' date to be included with the respective issue.

COMAL
REFERENCE GUIDE

Sixty-four pages outlining all the C64 COMAL keywords, with sections on the language's superb String Handling, Procedures and Parameters, Expressions and Standard Functions. **$9.95**



Send cheque or money order to:
TPUG Inc., Dept. A., 1912 Avenue Rd., Ste. 1, Toronto, Canada M5M 4A1

JOIN TPUG
The largest Commodore Users Group

Benefit from:

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

Magazine (10 per year) with advice from

Jim Butterfield
Brad Bjomdahl
Liz Deal

TPUG yearly memberships:

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

FOR FURTHER INFORMATION:
Send $1.00 for an information catalogue
(tell us which machine you use!)

To: TPUG INC.
DEPT. A,
1912A AVENUE RD., SUITE 1
TORONTO, ONTARIO
CANADA M5M 4A1

You're invited to the biggest party at Valley Forge since George brought the boys!

M.A.R.C.A.

The biggest Commodore User Fair in the US.

July 26, 27, 28

Valley Forge Convention Center, Valley Forge PA

- Speakers! • Seminars! • Hanging out!
- Fun! • Vendors! • Great Buys!
- Social Events! • Fun!

Areas Tours available.

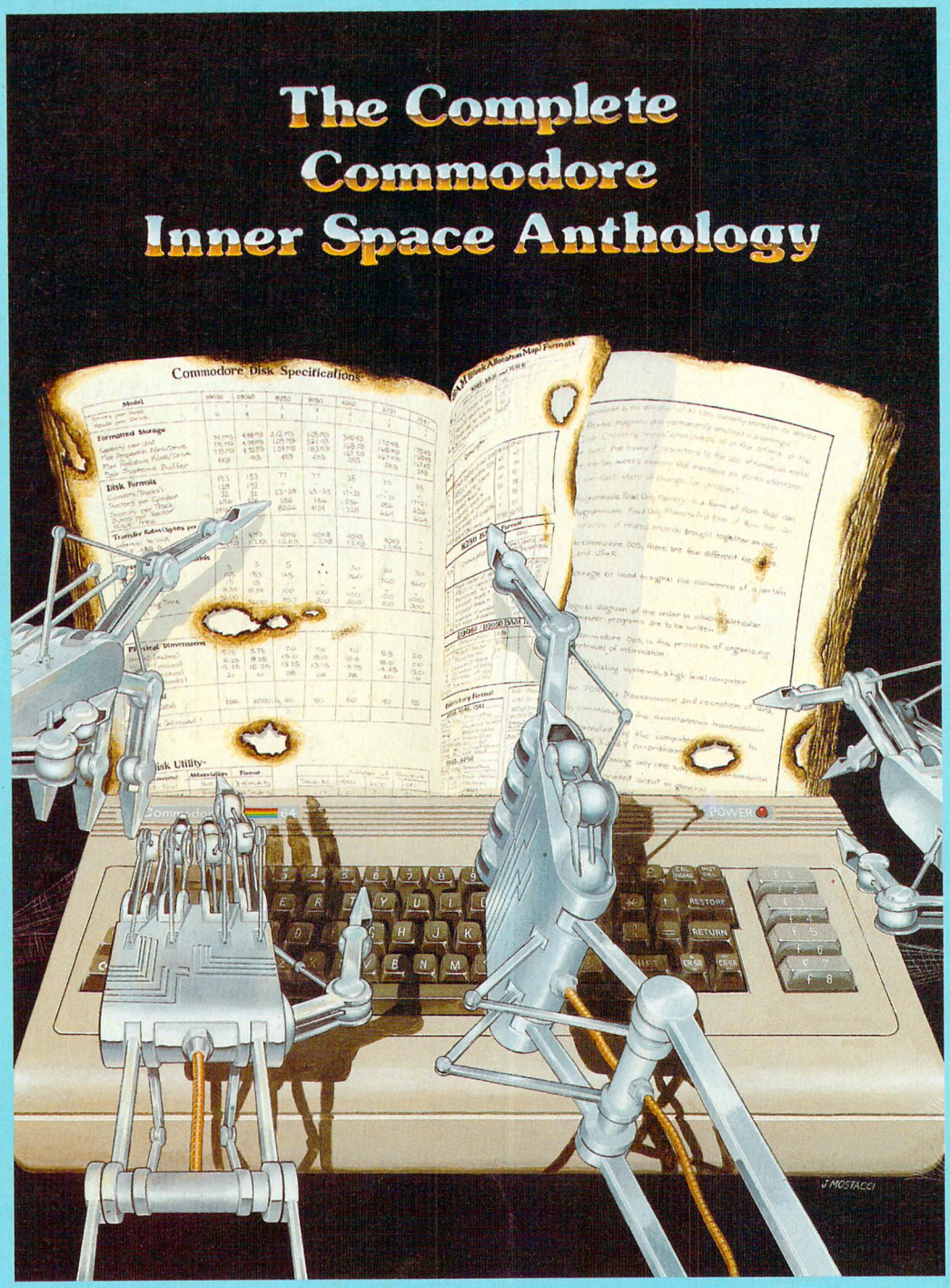
Meet the names you've only read about. Jim Butterfield. Dick Immers. Len Lindsey. Many, many more! Ask the questions you need answers to. Have 2½ days of non-stop Commodore fun! Bring the whole family. Lots to do. See. And buy. Bargains galore!

Pre-registration by July 1: 2½ days $25

For pre-registration information: M.A.R.C.A., P.O. Box 1902, Martinsburg, West VA 25401.

DON'T MISS THE PARTY!

The Transactor presents, The Complete Commodore Inner Space Anthology



Only $14.95

Postage Paid Order Form at Center Page