THE MAGAZINE FOR COMMODORE PROGRAMMERS AUGUST 1989: VOLUME 9, ISSUE 6

THE ONE MEGABYTE C64!

Canada $4.25
USA $3.50

Transactor

- **RAMfinder** - *Identify, stash and fetch*
- **Combiner** - *A handy utility for geoWrite*
- **Encryptor** - *Password protection for the C64*
- **Pop-ASCII** - *A handy pop-up utility for the C64*
- **IEEE-to-Serial Bus Conversion** - *for the 4040*
- **Colour Coordination** - *Making the right choices*
- **The One Megabyte C64!** - *Expand your C64 internally*
- **Clean Machine Language Screens** - *Techniques for text output routines*
- **Plus** Regular columns by Todd Heimarck and Joel Rubin, *Bits,* and more

Whyaduck by Wayne Schmidt

UTILITIES UNLIMITED, Inc.

12305 N.E. 152nd Street
Brush Prairie, Washington 98606

If you wish to place your order by phone, please call **206-254-6530**. Add $3.00 shipping & handling; $3.00 COD on all orders. Visa, M/C accepted. Dealer Inquiries Invited.

WORLD'S BIGGEST PROVIDER OF C64/128 UTILITIES

Software Submissions Invited

We are looking for HACKER STUFF: print utilities, parameters, telecommunications, and the unusual.

We now have over 1,000 parameters in stock!

OVER 5000 UNITS SOLD!!!

Unlike our competitors, we at Utilities Unlimited, Inc. have been concentrating all our efforts in bringing the newest technology. The result of that effort is **SuperCard**. It is far **superior to all the copy utilities out there including: Ramboard/Renegade**, Datel Burst Nibbler, 21Second, Ultrabyte, and any other backup utility on the market. So don't be led astray. **We will give you your money back** if they can back up more of the latest software, will they??? In a word "NO! ALL SALES ARE FINAL!!!" That is their response if you want to return RAMBO.

If you happen to see the ads on RAMBOard (original name huh), they claim to be cheaper. Well, that's partially true, but as is usual, mostly false. First you need to buy their board, then you need to spend another $34.95 for software to run their board. **That makes the cost of Rambo/Renegade to be at least $69.90. But then they claim you can use our software (what does that say about their software?). Well now, that may be just a bit of a white lie as well, while it's true that early, less reliable versions work with THEIR thing, the new more reliable versions of SuperCard software is specifically designed not to work with their RAMBO. For those people that have found out that the RAMBO and Renegade software package are quite inferior to SuperCard we offer the following suggestion. Send in your RAMBO and $24.95 and WE'LL SEND YOU THE REAL THING — SuperCard.** Needless to say you need a pair of hip boots to walk through their claim that they are the best. By the way, their software that backs up an unprotected disk in 50 seconds, well, it doesn't even use the RAMBO to work. **I suppose if you had a choice of an OLDSMOBILE or a Corvette with no engine, you would still pick the Oldsmobile.**

SuperCard 1541/1541c	**$49.95**	2 drive version	**$79.90**
SuperCard 1541-II	**$59.95**	2 drive version	**$99.90**
SuperCard 1571	**$59.95**	2 drive version	**$99.90**

SuperCard 1541-II version will work with most compatible drives.

These prices include software. You don't need to steal anyone else's software to make it work.

SUPER PARAMETERS 500 Pack #1 and #2

500 Pack #1 - $24.95 has the vintage parameters on it that no one else has. This pack comes in a 5-disk set.
500 Pack #2 - $29.95 has all the most current parameters on it. And put together as only Utilities Unltd. can. All Super Parameter Packs are completely menu driven, fast and reliable. Included on both 500 Packs is our state-of-the-art 64/128 Super Nibbler at no extra charge.

SUPER PARAMETERS 1000 Pack #1

Utilities Unltd. has done it again!! We have consolidated and lowered the prices on the most popular parameters on the market . . . Super-Parameters, now you can get 1000 parameters and our 64/128 nibbler package for just **$39.95!!!** This is a complete 10 disk set, that includes every parameter we have produced.

PARAMETERS CONSTRUCTION SET

The company that has **The Most Parameters** is about to do something **Unbelievable.** We are giving you more of our secrets. Using this **Very Easy** program, it will not only **Read, Compare** and **Write Parameters for You;** it will also **Customize** the disk with your name. It will impress you as well as your friends. The **"Parameter Construction Set"** is like nothing you've ever seen. In fact you can even **Read Parameters** that you may have already written; then by using your construction set rewrite it with your new **Customized Menu. $24.95**

LOCK PICK - THE BOOKS - for the C64 and C128

Lock Pik 64/128 was put together by our crack team, as a tool for those who have a desire to see the **Internal Workings** of a parameter. The books give you **Step-By-Step Instructions** on breaking protection for backup of 100 popular program titles. Uses **Hesmon** and **Superedit.** Instructions are so clear and precise that anyone can use it.

• OUR BOOK TWO IS NOW AVAILABLE •

BOOK 1: Includes Hesmon and a disk with many utilities such as: **KERNAL SAVE, I/O SAVE, DISK LOG FILE** and lots more, all with instructions on disk. Along-time favorite.

BOOK 2: 100 NEW EXAMPLES, Hesmon on disk and cartridge plus more utilities to include: **A General Overview on How to Make Parameters** and a **Disk Scanner.** **$19.95 each** OR **BUY BOTH FOR ONLY $29.95**
Now with FREE Hesmon Cartridge.

THE 128 SUPERCHIP - A, B or C (another first)

A — There is an empty socket inside your 128 just waiting for our **Super Chip** to give you 32K worth of great **Built-in Utilities,** all at just the **Touch of a Finger.** You get built-in features: **File Copier, Nibbler, Track & Sector Editor, Screen Dump,** and even a 300/1200 baud **Terminal Program** that's 1650, 1670 and Hayes compatible. Best of all, it doesn't use up any memory. To use, simply touch a function key, and it responds to your command.

B — HAS SUPER 81 UTILITIES, a complete utility package for the 1581. Copy whole disks from 1541 or 1571 format to 1581. Many options include 1581 disk editor, drive monitor, Ram writer and will also perform many CP/M & MS-DOS utility functions.

C — "C" IS FOR COMBO and that's what you get. A super combination of both chips A and B in one chip, switchable at a great savings to you. **All Chips Include 100 Parameters FREE!**
Chips A or B: **$29.95 ea.** Chip C: **$44.95 ea.**

SUPER GRAPHICS 1000 PACK

That's right! Over 1000 graphics in a 10-disk set for only **$29.95.** There are graphics for virtually everything in this package. These graphics work with Print Shop and Print Master.

ADULT GAME & GRAPHICS DATA DISKS

GAME: A very unusual game to be played by a very **Open Minded** adult. It includes a **Casino** and **House of Ill Repute.** Please, you **Must be 18** to order **Either One.**
DATA ★: This Popular disk works with **Print Shop** and **Print Master.**
Now Version 1 + 2 . . . **$24.95 ea.**

NEW! SUPER CARTRIDGE EXPLODE! V4.1 w/COLOR DUMP $44.95

Introducing the **World's First Color Screen Dump** in a cartridge. Explode! V4.1 will now **Support Directly** from the screen. FULL COLOR PRINTING for the **Rainbow Star NX-100** and also the **Okidata 10 & 20** printers.

The **Most Powerful Disk Drive** and **Printer Cartridge** produced for the COMMODORE USER. **Super Friendly** with the features most asked for.

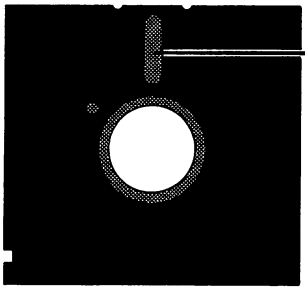
- **SUPER FAST** built-in single drive 8 or 9 FILE COPY, copy files of up to 235 BLOCKS in length, in less than 13 seconds!
- **SUPER SCREEN CAPTURE.** Capture and Convert **Any Screen** to KOALA or DOODLE.
- **SUPER FAST FORMAT** (8 SEC'S) - plus FULL D.O.S. WEDGE w/standard format!
- **SUPER FASTLOAD** and **SAVE** (50k in 9 SEC'S) works with all **C-64 or C-128's No Matter What Vintage!** And with most after market drives EXCEPT the 1581, M.S.D. 1 or 2.
- **SUPER PRINTER FEATURES** allows ANY DOT MATRIX PRINTER even 1526/802 to print HI-RES SCREENS (using 16 shade GRAY SCALE). **Any Printer or Interface Combination** can be used with SUPER EXPLODE! V4.1 or V3.0.
- **NEW and IMPROVED CONVERT** feature allows anybody to convert (even TEXT) **Screens into DOODLE or KOALA Type Pictures w/Full Color!**
- **SUPER FAST SAVE** of EXPLODE! SCREENS as KOALA or DOODLE FILES w/COLOR.
- **SUPER FAST LOADING** with **Color Re-Display of DOODLE or KOALA** files.
- **SUPER FAST LOAD or SAVE** can be TURNED OFF or ON without AFFECTING the REST of SUPER EXPLODE'S FEATURES. The rest of Explode V4.1 is still active.
- **SUPER EASY LOADING and RUNNING** of ALL PROGRAMS from the DISK DIRECTORY.
- **SUPER BUILT-IN TWO-WAY SEQ. or PRG. file READER** using the DISK DIRECTORY.
- **NEVER TYPE A FILE NAME AGAIN** when you use SUPER EXPLODE'S unique LOADERS.
- **CAPTURE** 40 COLUMN C or D-128 SCREENS! (with optional DISABLE SWITCH). Add $5.

ALL THE ABOVE FEATURES, AND MUCH MORE!
PLUS A FREE UTILITY DISK w/SUPER EXPLODE! V4.1.
MAKE YOUR C-64, 64-C or C-128*, D-128* SUPER FAST and EASY to use.

SUPER TRACKER

Utilities Unlimited has done it again. At last an easy way to find out where the protection really is. **Super Tracker** will display the location of your drive head while you are loading a piece of software. This information will be very useful, to find where the protection is. **Super Tracker** has other useful options such as: track and half-track display, 8 and 9 switch, density display, write protect on/off. This incredible little tool is encased in a handsome box that sits on top of your drive. Works with all C/64/128 and most C/64 compatible drives. Some minor soldering will be required.

Introductory Priced at Just **$69.95**



Start Address

Hack this editorial

It's that time again. *Transactor* is pleased to introduce a new assistant editor. By the time you read this, the editorial staff will include Paul Bosacki. Readers will probably recall that it was Paul who introduced us to the C256 in Volume 9, Issue 2 and showed us how to expand the 1764 with RAM and an EPROM in Volume 9, Issue 5. In this issue, you'll find that *The One Megabyte C64* has been added to Paul's list of credits. As you would imagine, the presence of a hardware hacker in the *Transactor* offices could make for some interesting developments in the matter of 'pushing the limits' in the pages of *Transactor*. Stay tuned! There are more limits that need pushing....

* * *

If you haven't sent in your Reader Survey yet, please do. They've just started coming in and have made for interesting reading. Although no space on the page was allotted for your name and address, feel free to include that information or your CompuServe PPN or Q-Link handle if you wish. I spend my on-line time on CompuServe (76703,4243) but Paul is on Q-Link (PaulB109).

You are encouraged not only to participate in the Reader Survey but also to write letters or to send electronic mail. We want to establish a dialogue. Now that there are fewer large companies supporting the 8-bit machines, it has become increasingly important that we support each other. This can only come about when such a dialogue becomes established. The on-line networks are an excellent way to keep in touch. Another is our exchange subscriptions with user groups. I read all the user group newsletters that come into *Transactor* and that has been a very valuable indicator of what's happening in the 8-bit world. So don't hold back, tell us what's on your mind.

* * *

We are distressed to find that the new edition of the Oxford dictionary gives the follow (informal) meaning to the term "hack": to gain unauthorized access to (computer files). This is somewhat puzzling considering that they give the (informal) meaning of "hacking" as: using a computer for the satisfaction that it gives. Do they mean to suggest that there's no satisfaction in gaining *authorized* access to computer files? Does this make no sense at all, or is it me?

* * *

In addition to Paul's Mega64, this issue features: a pop-up utility by Peter Lottrup for the 64 (runs in 64K machines!), some tips from Bill Brier on creating ML text display routines, a nifty IEEE-to-serial coversion project for the 4040, an encryption program from Jim Frost, a utility by Nick Vrtis that will combine *geoWrite* files (regardless of version).

The prolific Jim Butterfield explains exactly why some colour combinations work and others don't. You'll save a lot of trial and error by using the chart that Jim has included with this article. Add to this the columns, *bits*, reviews and other articles and I'd say you're in for some interesting reading.

Malcolm D. O'Brien

Transactor

The Magazine for Commodore Programmers

Volume 9, Issue 6

Publisher
Antony Jacobson

Vice-President Operations
Jeannie Lawrence

Assistant Advertising Manager
Mike Grantham

Editors
Malcolm O'Brien
Nick Sullivan
Chris Zamara
Assistant Editor
Paul Bosacki
Contributing Writers
Ian Adam
Paul Bosacki
Bill Brier
Anthony Bryant
Joseph Buckley
Jim Butterfield
William Coleman
James Cook
Richard Curcio
Jim Frost
Miklos Garamszeghy
Larry Gaynier
Michael Gilsdorf
Kerry Gray
Todd Heimarck
Adam Herst
Robert Huehn
George Hug
Dennis Jarvis
Francis Kostella
Jean-Yves Lemieux
Peter Lottrup
Mike Mohilo
D.J. Morriss
Noel Nyman
Steve Punter
Robert Rockefeller
Joel Rubin
Anton Treuenfels
Nicholas Vrtis

Cover Artist
Wayne Schmidt

Departments and Columns

Letters 6

Bits 10

Debug 128
Don't Assume Device 8!
Shortest directory in BASIC 2.0?
Partition

The ML Column 14

by Todd Heimarck

More on big numbers including a primes program. Requires an REU.

The Edge Connection 19

by Joel Rubin

Societies, anti-rental laws, shows and disk drive voodoo.

News BRK 78

About the cover: *Whyaduck* by Wayne Schmidt:

Quite a different source of inspiration this time around. This issue's cover has an old comedy routine as its source. This colourful picture of a duck is a reference to a humourous routine by the Marx Brothers concerning a viaduct. This picture was created with *Artist 64*, modified for the 1351 mouse. - Wayne Schmidt

Transactor is published bimonthly by Croftward Publishing Inc., 85-10 West Wilmot Street, Richmond Hill, Ontario, L4B 1K7. ISSN# 0838-0163. Canadian Second Class Mail Registration No. 7680, Gateway-Mississauga, Ont. USPS Postmasters: send address changes to: Transactor, PO Box 338, Station C, Buffalo, NY, 14209.

Croftward publishing Inc. is in no way connected with Commodore Business Machines Ltd. or Commodore Incorporated. Commodore and Commodore product names are registered trademarks of Commodore Inc.

Subscriptions:

Canada $19 Cdn.
USA $15 US
All others $21 US
Air Mail (Overseas only) $40 US

Send all subscriptions to: Transactor., Subscriptions Department, 85 West Wilmot Street, Unit 10, Richmond Hill, Ontario, Canada, L4B 1K7, (416) 764-5273. For best results, use the postage paid card at the centre of the magazine.

Quantity Orders: In **Canada**: Ingram Software Ltd., 141 Adesso Drive, Concord, Ontario, L4K 2W7, (416) 738-1700. In the **USA**: IPD (International Periodical Distributors), 11760-B Sorrento Valley Road, San Diego, California, 92121, (619) 481-5928; ask for Dave Buescher.

Editorial contributions are welcome. Only original, previously unpublished material will be considered. Program listings and articles, including BITS submissions, of more than a few lines, should be provided on disk. Preferred format is 1541-format with ASCII text files. Manuscripts should be typewritten, double-spaced, with special characters or formats clearly marked. Photos should be glossy black and white prints. Illustrations should be on white paper with black ink only. Hi-res graphics files on disk are preferred to hardcopy illustrations when possible. Write to Transactor's Richmond Hill office to obtain a writer's guide.

Production
In-house with Amiga 2000 and
Professional Page and Digiview

Final output by Vellum Print &
Graphic Services, Inc., Toronto

Printing
Printed in Canada by
Bowne of Canada Inc.

Using "VERIFIZER"

Transactor's foolproof program entry method

VERIFIZER should be run before typing in any long program from the pages of *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; you can then check this code against the corresponding one in the printed program listing.

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

SYS 634 to enable the PET/CBM version (off: SYS 637)
SYS 828 to enable the C64/VIC version (off: SYS 831)
SYS 3072,1 to enable the C128 version (off: SYS 3072,0)

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

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

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

VERIFIZER will catch transposition errors like POKE 52381,0 instead of POKE 53281,0. However, VERIFIZER uses a "weighted checksum technique" that can be fooled if you try hard enough: transposing two sets of four characters will produce the same report code, but this will rarely happen. (VERIFIZER could have been designed to be more complex, but the report codes would need to be longer, and using it would be more trouble than checking the program manually). VERIFIZER ignores spaces so you may add or omit spaces from the listed program at will (providing you don't split up keywords!) Standard keyword abbreviations (like nE instead of next) will not affect the VERIFIZER report code.

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

PET/CBM VERIFIZER (BASIC 2.0 or 4.0)

```
CI  10 rem* data loader for "verifizer 4.0" *
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
```

VIC/C64 VERIFIZER

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

***NEW* C128 VERIFIZER (40 or 80 column mode)**

```
KL  100 rem save"0:c128 vfz.ldr",8
OI  110 rem c-128 verifizer
MO  120 rem bugs fixed:  1) works in 80 column mode.
DG  130 rem              2) sys 3072,0 now works.
KK  140 rem
GH  150 rem by joel m. rubin
HG  160 rem * data loader for "verifizer c128"
IF  170 rem * commodore c128 version
DG  180 rem * works in 40 or 80 column mode!!!
EB  190 ch=0
GC  200 for j=3072 to 3220: read x: poke j,x: ch=ch+x: next
NK  210 if ch<>18602 then print "checksum error": stop
BL  220 print "sys 3072,1 to enable
DP  230 print "sys 3072,0 to disable
AP  240 end
BA  250 data 170, 208,  11, 165, 253, 141,   2,   3
MM  260 data 165, 254, 141,   3,   3,  96, 173,   3
AA  270 data   3, 201,  12, 240,  17, 133, 254, 173
FM  280 data   2,   3, 133, 253, 169,  39, 141,   2
IF  290 data   3, 169,  12, 141,   3,   3,  96, 169
FA  300 data   0, 141,   0, 255, 165,  22, 133, 250
LC  310 data 162,   0, 160,   0, 189,   0,   2, 201
AJ  320 data  48, 144,   7, 201,  58, 176,   3, 232
EC  330 data 208, 242, 189,   0,   2, 240,  22, 201
PI  340 data  32, 240,  15, 133, 252, 200, 152,  41
FF  350 data   3, 133, 251,  32, 141,  12, 198, 251
DE  360 data  16, 249, 232, 208, 229,  56,  32, 240
CB  370 data 255, 169,  19,  32, 210, 255, 169,  18
OK  380 data  32, 210, 255, 165, 250,  41,  15,  24
ON  390 data 105, 193,  32, 210, 255, 165, 250,  74
OI  400 data  74,  74,  74,  24, 105, 193,  32, 210
OD  410 data 255, 169, 146,  32, 210, 255,  24,  32
PA  420 data 240, 255, 108, 253,   0, 165, 252,  24
BO  430 data 101, 250, 133, 250,  96
```

The Standard Transactor Program Generator

If you type in programs from the magazine, you might be able to save yourself some work with the program listed on this page. Since many programs are printed in the form of a BASIC "program generator" which creates a machine language (or BASIC) program on disk, we have created a "standard generator" program that contains code common to all program generators. Just type this in once, and save all that typing for every other program generator you enter!

Once the program is typed in (check the Verifizer codes as usual when entering it), save it on a disk for future use. Whenever you type in a program generator, the listing will refer to the standard generator. Load the standard generator *first*, then type the lines from the listing as shown. The resulting program will include the generator code and be ready to run.

When you run the new generator, it will create a program on disk (the one described in the related article). The generator program is just an easy way for you to put a machine language program on disk, using the standard BASIC editor at your disposal. After the file has been created, the generator is no longer needed. The standard generator, however, should be kept handy for future program generators.

The standard generator listed here will appear in every issue from now on (when necessary) as a standard *Transactor* utility like Verifizer.

```
MG  100 rem  transactor standard program generator
EE  110 n$="filename": rem  name of program
LK  120 nd=000: sa=00000: ch=00000
KO  130 for i=1 to nd: read x
EC  140 ch=ch-x: next
FB  150 if ch then print "data error": stop
DE  160 print "data ok, now creating file."
CM  170 restore
CH  180 open 1,8,1,"0:"+n$
HM  190 hi=int(sa/256): lo=sa-256*hi
NA  200 print#1,chr$(lo)chr$(hi);
KD  210 for i=1 to nd: read x
HE  220 print#1,chr$(x);: next
JL  230 close 1
MP  240 print"prg file ’";n$;"’ created..."
MH  250 print"this generator no longer needed."
IH  260 :
```

LeTteRs

Another view of DevPak: This letter is a comment concerning Joel Rubin's remarks in Volume 9, Issue 3 about the **DevPak128** package from Commodore.

I have made extensive use of this package in the development of a multi-user, online truck leasing and billing system. The total amount of code written for this system (it is 100 percent machine language) is about 100,000 lines. The software runs on a group of C128D computers multiplexed to an 80MB Xetec Lt. Kernal hard disk subsystem. I used a separate C128D and 40MB Lt. Kernal as the development system, using a homebrew text editor to write the source code and the DevPak assembler and loader to create executable object code.

It is true that the DevPak assembler is disk-intensive. So is just about any assembler that must make two passes through ten files totalling nearly 400 kilobytes of source code. As for the procedure of having to use the hex file loader to actually place your program into RAM, that procedure has existed with all assembler packages that have been marketed by Commodore (the **C64 Macro Assembler Development System** or MADS uses the identical procedure).

The limitations on open files and speed on a 1541 or 1571 drive are limitations that any assembler must contend with. As Mr. Rubin mentioned, these limitations are clearly explained in the DevPak documentation and can be alleviated by using multiple drives, as the assembler can read source code from one unit and write object code to another. Additional gains in speed can be achieved by utilizing the SFD-1001 drive and a Skyles Quicksilver IEEE interface or if the user is intent on doing some heavy-duty programming, the Lt. Kernal system (the Lt. Kernal DOS allows up to seven files to be opened at the same time).

Because I do my development on a Lt. Kernal-based system, I do not experience the problems Mr. Rubin mentions about speed and open files. Even my largest program assembles at a rapid rate. Smaller programs (those with less than 50K of source code) assemble in under three minutes if no listing output is required. So, while the disk-intensive nature of DevPak might be a problem on a 1541 or 1571 system, it probably would not be a problem on a system with greater disk capacity (for example, the SFD-1001 allows a larger number of files to be simultaneously opened because of more available drive RAM).

The advantages of the DevPak assembler, in my opinion, outweigh the disadvantages. For one thing, the assembler's parsing routine is not case-sensitive for non-quoted strings. Quoted strings may include shifted or PET graphics characters (something which is not allowed by many assemblers). Another point to consider is that DevPak supports local labels (real handy for patching existing programs). The macro facility works flawlessly and allows nesting of macros (macros can call other macros). The printed output listing is more informative than that of most other assemblers. The symbol table is structured in RAM 1 and has over 60 kilobytes available in which to deposit data.

The need to use the loader to place the hex image file into RAM is a minor nuisance in some cases. However, the use of the hex loader allows me to assemble for an area which can't be conveniently used as a location from which to execute a binary save (such as the hardware stack) and load the program into a free area of RAM from which it may be saved. This feature is complemented by the ability of the Lt. Kernal DOS to change the load address of a binary file after it has been stored on the drive.

I cannot recommend the EDT text editor that is supplied with DevPak, both for the reasons mentioned by Mr. Rubin (the use of the numeric keypad to issue commands to the editor) and because the editor is actually quite unfriendly and cumbersome. However, as he mentions, almost any editor can be used in its place.

In summary, the DevPak assembler is gross overkill for the casual programmer that is interested in writing only a few lines of code. I can't recommend it for the user that has only a 1541 or 1571 on his system. This assembler is really meant for a serious machine language programmer who has the proper hardware to go with it.

Bill Brier, Bensenville, IL

A letter to Francis Kostella: I am writing you in a somewhat desperate attempt to get some reliable information on how to obtain a copy of Alexander Boyce's GEOS manual. I realize it's not your job to answer questions like this (sorry) but I couldn't think of anyone else to ask. I'm a bit at my wits' end.

I have been trying to obtain a copy of the manual for several months. Through what seemed a stroke of good fortune, Nicholas Vrtis published Alexander Boyce's address in *Transactor*, Volume 9, Issue 4. However, a letter to that address was returned to me only yesterday, unopened - that address does not seem to exist. My final plan of attack is to get in touch with people who have the manual already, to see if they can give me a lead on how to get a copy. Hence my letter to you.

Can you please send me any hints or suggestions you might have on how to get a copy of Boyce's book? Even a photocopy, I don't care. I really do want this manual. Thanks very much for your trouble.

David Kotchan, Toronto, Ontario

We managed to contact Alex. Here's his new address:

Alexander Boyce
63 Chamberlain Ave.
Elmwood Park, NJ 07407

Incompatible 1541C?: I am writing to you in hopes that you may help me with a problem which has plagued the technicians here in Ottawa and at Commodore in Toronto for some time.

The problem began when I bought a second disk drive model 1541 and added it to my collection of 1541s... This is my setup: 64, 1701, three 1541s, Epyx *Fastload* cartridge, Aprotek RS-232 interface, Star NX-1000 and a Datagram modem.

After many years of being interested in Commodore equipment, I have never heard of this problem. When I connect my recently purchased 1541 as device 8, it locks up the 64. I have made many trips to my local service depot and spent many hours in frustration, so I decided to troubleshoot this problem myself.

In the beginning, I had everything connected to a power bar so all I had to do was hit the switch and go... (not by the book, but has been effective in the past).

To make a long story short, after I put my new drive on as device 8, it locks up the 64. The screen will say, for example, "searching for $" and that's it. The read LED never lights and my keyboard is now frozen. The only way to access it, is by resetting the drive and then it will work, but this only happens on the very first time, then it's somewhat OK for the rest of the evening.

Now it gets even more interesting. If I only leave device 8 on and turn on the power bar everything is fine, but as soon as I turn on device 9, that's it! - the keyboard is frozen. The only way to get back to normal is to reset all drives. Now this may not be a bad solution; however, as my system has grown I have gotten squeezed out of my office and forced to build a custom computer hutch that contains all my equipment. The hutch is virtually useless to me now, because every time I go to use it I have to consistently start pulling equipment out of it to reset it. This is not very practical and so I have abandoned this drive.

So you say, how can we help? Well, I'm going to tell you. After some research I believe that it has something to do with the priority of how the 64 recognizes the 1541C.

After closer inspection of the situation, I have concluded that the logic PCB in this new drive is not compatible with the others. As I had previously stated, everything was in perfect working order until I installed this new drive.

What I have done is taken the version number from each drive, hoping that you will be able to help me...

My question is: Can I make them compatible with the same type of software?

My new 1541 is a PCB #251830 Rev. A. My old 1541 is a PCB #1540050 Rev. C. The service people have been co-operative and have said that if it is possible to make them compatible, then they would do so. I hope that the solution is a simple software upgrade or downgrade, whichever makes it work!

Terry Golding, address unknown

First of all, troubleshooting by mail rarely works... However, it sounds like a 'serial bus loading problem'. These tend to be more common as devices are added to the bus and some devices are more likely than others to cause such problems. For example, one revision of the 1526 is notorious in this regard. Of course, you may be right about the 1541C. This is one piece of equipment with which Transactor *has no experience. (We don't have any 1541-IIs either.) If anyone can supply more information on this subject, please send it in.*

Transblooperz in *Programming GEOS Icons*: First, let me say that this is the first letter to an editor that I have ever written. I have been reading *Transactor* for several years and I believe it is the finest Commodore-specific magazine existing.

I am writing regarding the article *Programming GEOS Icons* on page 56 of Volume 9, Issue 5. I am a GEOS enthusiast and enjoyed the article very much. The program works well, but there are a couple of errors; one in the article and one in the program. Also, the program (*geoKeyboard*) can be shortened considerably, as I will show.

Firstly, in the fifth paragraph at the top of the right-hand column on page 56, it is stated: "When *DoIcons* is called, the GEOS Kernal expects the two-byte **.word** following the JSR in memory to contain the pointer to the icon table." This is not correct. The pointer to the icon table must be loaded into **r0L/r0H** (using the macro, **LoadW r0,IconTable**) before the JSR to *DoIcons*, as is done in the program on page 59. There is no in-line form of the *DoIcons* routine.

Secondly, in the *geoKeyboard* program (page 59, left hand column), the following sequence is printed:

```
lda     #0              ;Put mouse on geos menu item
LoadW   r0,GeosMenu     ;Put address of menu table in r0
jsr     DoMenu
```

I must point out that if this routine is coded as above, the **LoadW** macro will change the value of the A register, and the mouse will not be put in the right place. The **LoadW** macro and the **lda** should change places, as follows:

```
    LoadW   r0,GeosMenu
    lda     #0
    jsr     DoMenu
    rts
```

Now the A register will contain 0 on entry to the *DoMenu* routine, and the mouse will be placed on the first menu item.

Now for the change to make the program shorter. The following is based upon the fact that, after an icon is clicked, its number (based on its position in the icon table, starting with 0) is returned in **r0L**. It is simple then to use this value to index into a table of frequency values, instead of having a separate action routine for each note.

1) In the Keyboard icon table (page 59), change all the action routine pointers (such as **.word DoCN4**, **DoCS4** etc.) to **.word Play**

2) Eliminate all the routines on page 60/61 for loading the frequency values into **a0L/a0H** (**DoCN4** to **DoCN6**)

3) Change the routine **Play** on page 61 (left hand column) to:

```
    jsr InitForIO
    lda #$40
    sta vlcntrl
    ldx r0L       ;put icon number into index register
    lda lofreq,x ;get low frequency value from table
    sta vlfreqlo ;put it in the sid register
    lda hifreq,x ;get high frequency value from table
    sta vlfreqhi ;put it in the sid register
    lda #$41
    sta vlcntrl
    <rest same>
```

4) Add the following data table to the program at the end (after **jmp EnterDeskTop**)

```
lofreq:
    .byte 195, 195, 209, 239,  31,  96, 181,  30
    .byte 156,  49, 223, 165, 135, 134, 162, 223
    .byte  62, 193, 107,  60,  57,  99, 190,  75,  15
hifreq:
    .byte  16,  17,  18,  19,  21,  22,  23,  25
    .byte  26,  28,  29,  31,  33,  35,  37,  39
    .byte  42,  44,  47,  50,  53,  56,  59,  63,  67
```

There is just one more thing. If the **lda #$01/sta vlsusrel** in the *LoadSIDRegisters* routine is changed to **lda #$0c/sta vlsusrel**, the note lasts longer and seems to sound better.

Roy Longworth, Trenton, ON

Right on all fronts, Roy. Thanks for pointing out the errors in the text and code. And thanks for the tip on shortening the code. Keep on writing letters to editors. We do appreciate it when readers find (and correct) our mistakes.

Back to Forth: Friends, I am looking for documentation for Scott Ballantyne's *Blazin' Forth* implementation of the Forth language. He wrote an article in *Transactor*, Vol. 7, Iss. 5 and it was on your disk. It seems to assume we all know the program well! I'm trying to learn Forth.

I would also like disk I/O routines for HESForth cartridges. C64 and VIC-20 disk operations crash on mine. Thanks.

Premena
P.O. Box 1038
Boulder, CO 80306-1038

Your best bet for 8-bit Forth support is CompuServe. LIB 5 of our Commodore Programming Forum (GO CBMPRG) is devoted to the Forth language. In addition to the complete source code for Blazin' Forth, *LIB 5 contains a number of helpful text files. What follows is a list of the files in the Forth library:*

```
Filename legend:

 /A = ASCII text file
 /B = Xmodem upload
 /I = B-protocol (Vidtex) upload
```

```
 /R = RLE graphic file

NOTE:  Size is rounded to the nearest full K (1K = 1024 bytes)

System            Upload
Filename    Size   date     Brief description
------------+----+---------+-------------------------------------------------
INTRO.4TH/A  19K  12-Oct-88 Overview of the Forth programming language
LIB5.DIR/A   5K   17-Jul-88 Directory of all files in LIB 5 to date
BVT100.BIN/B 28K  27-Sep-87 Blazin'Forth VT52 terminal emulator
FORTH.TXT/A  4K   15-Apr-87 A review of Steve Burnap's FORTH tutorial book
SIDEXP.IMG/I 38K  06-Feb-87 Forth program to exercise SID chip
PRTFIL.BIN/B 2K   11-Dec-86 Blazin'Forth sequential file printer
BFCDEM.IMG/B 18K  12-Nov-86 Fport source to demos described in BFHSRC.BIN
FSP.BFT/A    4K   12-Nov-86 Structured programming constructs in bforth83
FSP.TXT/A    80K  12-Nov-86 Text by George Hawkins on structured programming
BFCSRC.TUT/A 36K  22-Sep-86 Explains the inner workings of Blazin'Forth
BFC1.ASM/A   19K  18-Sep-86 First assembler source for Blazin'Forth Compiler
BFC10.ASM/A  1K   18-Sep-86 Support file for Blazin'Forth (Macros)
BFC11.ASM/A  10K  18-Sep-86 Support file for BForth (global declarations)
BFC12.ASM/A  2K   18-Sep-86 Support file for BForth (constant declarations)
BFC2.ASM/A   26K  18-Sep-86 Second source file for Blazin'Forth Compiler
BFC3.ASM/A   22K  18-Sep-86 Third source file for Blazin'Forth Compiler
BFC4.ASM/A   24K  18-Sep-86 Fourth source file for Blazin'Forth Compiler
BFC5.ASM/A   28K  18-Sep-86 Fifth source file for Blazin'Forth Compiler
BFC6.ASM/A   13K  18-Sep-86 Sixth source file for Blazin'Forth Compiler
BFC7.ASM/A   17K  18-Sep-86 Seventh source file for Blazin'Forth Compiler
BFC8.ASM/A   17K  18-Sep-86 Eighth source file for Blazin'Forth Compiler
BFC9.ASM/A   12K  18-Sep-86 Ninth source file for Blazin'Forth Compiler
DYNAM.FTH/A  5K   09-Sep-86 BForth code to do dynamic memory management
ESTAC2.DOC/A 5K   31-Aug-86 Documentation for ESTAC2.IMG
ESTAC2.IMG/I 17K  31-Aug-86 BForth floating point math in FPORT file
FTHSTR.BIN/A 2K   29-May-86 64FORTH string handling program
FTHSTR.DOC/A 1K   29-May-86 Documentation for FTHSTR.BIN
RELSEQ.BIN/B 1K   29-May-86 Converts 64FORTH REL to SEQ file
SEQREL.BIN/B 1K   29-May-86 Converts SEQ file to 64FORTH REL file
FILES.BIN/B  4K   12-May-86 Gives BForth C like files (fopen, fclose, etc.)
CBMDIR.IMG/I 1K   11-May-86 Directory using CBM's DOS directory, FPORT file
VBACK.BIN/B  2K   09-Mar-86 Backup for files created with VFILE.BIN
SF2BLZ.IMG/I 5K   07-Mar-86 Translate screens between Super Forth and BForth
VFILE.BIN/B  3K   05-Mar-86 Save BForth code as commodore REL files
FLOAT.BIN/B  11K  26-Feb-86 Forth 83 floating point math words
MULTI.BIN/B  4K   26-Feb-86 Add background tasks to BForth
BACKUP.BIN/B 2K   18-Feb-86 Utility to backup screens
REALCL.BIN/B 2K   18-Feb-86 Forth words to access the c64's hardware clock
BFASM.DOC/A  29K  10-Dec-85 Blazin'Forth Assembler tutorial
BFVDTE.TXT/A 6K   09-Dec-85 Blazin'Forth terminal program example
FPORT.IMG/I  1K   25-Nov-85 Upgraded FPORT file transfer utility
HFGFC0.DOC/A 12K  20-Oct-85 Documentation for HFGFC0.IMG
HFGFC0.IMG/I 6K   20-Oct-85 HES 64FORTH graphics program
BFCYAD.IMG/I 4K   23-Sep-85 Decompiler for Blazin'Forth
DECOMP.IMG/I 3K   16-Sep-85 Decompiler for Blazin'Forth
DIR.IMG/I    2K   13-Sep-85 Disk Directory for Blazin'Forth Command = DIR
BFEDIT.DOC/A 3K   12-Sep-85 Procedure for adding full screen editor to BForth
BFORTH.IMG/I 23K  08-Sep-85 Scott Ballantyne's Blazin' Forth Compiler system
ARTHUR.IMG/I 7K   04-Sep-85 Arthurs Theme, Blazin'Forth music
EXAMPL.FTH/A 4K   27-Aug-85 Help for Forth-83 changes to 'Starting Forth'
SIEV83.SRC/A 1K   27-Aug-85 Forth-83 Sieve of Eratosthenes
BFHSRC.BIN/B 61K  25-Aug-85 Squeezed source code for BFORTH.IMG
SRCWRT.DOC/A 1K   25-Aug-85 Documentation for SRCWRT.IMG and BFHSRC.BIN
SRCWRT.IMG/I 1K   25-Aug-85 Convert squeezed format source to Forth screens
BFDEMO.SRC/A 2K   24-Aug-85 BForth Turtle graphics demo
BFRTH1.IMG/I 1K   07-Aug-85 Readme file for BFORTH.IMG
BFRTH2.DOC/A 20K  07-Aug-85 Documentation for BFORTH.IMG (part 1)
BFRTH3.DOC/A 11K  07-Aug-85 Documentation for BFORTH.IMG (part 2)
BFRTH4.DOC/A 8K   07-Aug-85 Documentation for BFORTH.IMG (info on string pkg)
BFRTH5.DOC/A 10K  07-Aug-85 Documentation for BFORTH.IMG (sound extensions)
BFRTH6.DOC/A 18K  07-Aug-85 Documentation for BFORTH.IMG (turtle graphics)
BFRTH7.DOC/A 9K   07-Aug-85 Documentation for BFORTH.IMG (misc. info)
BFRTH8.DOC/A 19K  07-Aug-85 BFORTH.IMG help file 1 for 'Starting Forth' text
BFRTH9.DOC/A 11K  07-Aug-85 BFORTH.IMG help file 2 for 'Starting Forth' text
MON.IMG/I    4K   29-Oct-84 Monitor for HES 64FORTH (only)
NPOWER.SCR/A 1K   15-Apr-84 Forth power arguments
CONCAT.SCR/A 2K   28-Mar-84 Takes PMP screens and creates file for uploading
DECRYP.SCR/A 1K   28-Mar-84 Takes downloaded file and converts to PMP screen
MACROS.SCR/A 1K   28-Mar-84 Updated macros for Performance Micro (PMP)
QX.SCR/A     1K   28-Mar-84 Prints out screen headers, for PMP
TABLE.SCR/A  1K   28-Mar-84 PMP C64FORTH creates tables
THRU.SCR/A   1K   28-Mar-84 PMP C64FORTH word for fetching several screens
ASK.SCR/A    1K   28-Mar-84 Defining word create daughters numeric input
CASE.SCR/A   1K   28-Mar-84 Forth79 CASE statements
GOES.SCR/A   1K   28-Mar-84 Forth recursive decompiler
LIFE.SCR/A   2K   28-Mar-84 Forth mathematical/graphic Game of LIFE
LSCR.SCR/A   1K   28-Mar-84 Screens contain example life screens
TIME.SCR/A   1K   28-Mar-84 Forth79 words to support clock on 6526 chip
CANON.DOC/A  9K   28-Mar-84 Documentation file describing .SCR format
SCMSCR.SCR/A 1K   28-Mar-84 Simple data encrypter for forth screens
BOXES.SCR/A  1K   28-Mar-84 Draws random size and color boxes
SIEVE.SCR/A  1K   28-Mar-84 Sieve of Eratosthenes benchmark
SQROOT.SCR/A 1K   28-Mar-84 Returns square root, PMP assembler format
```

That empty REU socket: Is it possible to put the 28-pin chip from the Epyx *Fast Load* cartridge into the 1764? The *Fast Load* cartridge also has one other chip on it. It is a SN7407N DIP.

This info would be greatly appreciated. My *Fast Load* collects dust now because I don't want to keep plugging and unplugging the 1764, and I have no room for an expander board to plug both in. I hope that my *Fast Load* can be put back into action soon. There are probably quite a few people with the same need.

Frank Liuzzi, Broomall, Pennsylvania

Great idea, Frank, but unfortunately, it's just not possible. The Epyx Fast Load *cartridge is a transparent cartridge. The cartridge is visible only at particular times, specifically at a hard reset and on an access to $de00. This magic is achieved through the 7407N and a discharge capacitor on the cartridge board. Although the code maps in at $8000, the command parser maps in at $df00. (More magic, because the code for the parser is found in the $9f00 range of the EPROM.)*

Placing the EPROM in the REU socket would result in a hung machine on power-up because the BASIC IERROR vector is left pointing to somewhere in the $df00 block by the code that initializes the cartridge. Result: on an error (i.e., $, /, %, etc.), control is passed over to non-existent code at $df00 and the machine most likely crashes.

Also, the EPROM would grab the $8000 to $9fff block. Because the transparency was achieved through the support circuitry in the cartridge, we would always be out 8K of BASIC RAM. A terrible waste! Especially when you consider that the cartridge used to be 'invisible' in normal use.

So, what to use the REU EPROM socket for? Mostly home-brewed code, I would think. Today's cartridges are a lot more sophisticated than those of a few years ago. Not uncommonly now, we find kilobytes of bank-switched EPROM, DRAM and even microprocessors. Off hand, I can't think of any cartridge EPROM that could be plugged into that slot. Anybody know different?

bits

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

Debug Utility

Jean-Yves Lemieux, Rimouski, PQ

Debug is a programming utility for the C128 that can help a machine language programmer in a number of ways. It can provide a controlled testing environment for assembler programmers: avoid a system crash, detect endless loops, and so on. It is an interrupt-driven program that uses NMI and BREAK vectors and a CIA 2 timer to perform a 'trace' function. It lets you see, step by step, each instruction that your C128 executes, displaying register contents, PC address and disassembly of the next instruction to be executed.

This version is loaded at $03000. You'll need to reassemble to relocate it. Enable it with **sys12288** from BASIC or **jf3000** from your monitor. Now you're ready to use *Debug*'s two commands: Walk and Quick.

W <start address> (eg. **w 2000**): The first instruction is executed and you are then presented with a register display, PC address and the disassembled next instruction. Debug is waiting for your next command. Pressing a key will result in the execution of the next instruction. RUN/STOP will stop walking.

Q <routine address>: This command only works during a walk and at the beginning of a subroutine. Following instructions will be executed at nearly full speed until an RTS or BRK is encountered. No display is provided during this process. You should use the Quick command for normal system subroutines (BASIC or Kernal) since Walking through these will probably cause unpredictable results.

You can disable *Debug* with RUN/STOP-RESTORE. *Debug* generates system interruptions via Timer A of CIA 2 ($dd00). During a Walk or a Quick command a timer is set to generate an NMI. The registers are then pulled from the stack and are saved with the program counter for future use. Since the timers of CIA 1 are often used for system tasks (I/O), Timer A of CIA 2 (which generates only NMI) has been used. Because of the timer's involvement with RS-232 operations, you should not try to use *Debug* for RS-232 routines.

Listing 1: *debug.gen*

```
OC  100 rem prg. gen. for debug.obj
EK  110 n$="debug.obj"
DD  120 nd=376:sa=12288:ch=39305
KO  130 fori=1tond:readx
EC  140 ch=ch-x:next
FB  150 if chthenprint"data error":stop
DE  160 print"data ok, now creating file"
CM  170 restore
CH  180 open1,8,1,"0:"+n$
HM  190 hi=int(sa/256):lo=sa-256*hi
NA  200 print#1,chr$(lo)chr$(hi);
KD  210 fori=1tond:readx
HE  220 print#1,chr$(x);:next
JL  230 close1
MP  240 print"prg file '";n$;"' created..."
MH  250 print"this generator no longer needed."
LE  12288 data 120, 169, 185, 160,  48, 141,  22,   3
NB  12296 data 140,  23,   3, 169,  53, 160,  48, 141
MJ  12304 data  46,   3, 140,  47,   3, 169,  64, 162
ID  12312 data 250, 141, 157,   2, 142, 158,   2, 169
BI  12320 data   0, 141, 154,   2, 141, 155,   2,  88
EC  12328 data   0, 198,   4, 208,   2, 198,   3,  76
BO  12336 data  70, 176,  76, 178, 176, 201,  87, 208
DP  12344 data 249,  32, 167, 183, 176, 244, 166,  96
PK  12352 data 164,  97, 165,  98, 133,   2, 134,   4
BC  12360 data 132,   3, 186, 142, 156,   2, 169,  40
EI  12368 data 162,  49, 133, 250, 134, 251, 108, 250
IJ  12376 data   0,  32, 152,  85,  32, 125, 255,  83
MK  12384 data  82,  32,  65,  67,  32,  88,  82,  32
NM  12392 data  89,  82,  32,  83,  80,  32,  80,  67
LJ  12400 data  13,   0, 166,   2, 165,   3, 201,  64
DL  12408 data 144,   2, 162,  15, 134, 104, 133, 103
FI  12416 data 165,   4, 133, 102, 160,   0, 185,   5
IE  12424 data   0,  32, 165, 184, 200, 192,   5, 144
ED  12432 data 245,  32, 146, 184, 160,   0, 174, 170
FE  12440 data   2, 134,  77,  32,  26, 177,  32,  89
JO  12448 data 182,  32,   8, 182, 166,  77, 142, 170
EN  12456 data   2,  76, 152,  85, 173, 157,   2, 172
FL  12464 data 158,   2, 141,  24,   3, 140,  25,   3
JH  12472 data  96, 169, 128, 141,  14, 221, 173,  13
KM  12480 data 221,  32, 172,  48, 216, 104, 133,   2
MK  12488 data 104, 133,   8, 104, 133,   7, 104, 133
EL  12496 data   6, 104, 133,   5, 104, 133,   4, 104
KJ  12504 data 133,   3, 186, 134,   9,  88, 165,   5
LA  12512 data  41,  16, 240,   3,  76,  41,  48,  44
```

```
FB  12520 data 154,   2,  48,  12, 166,   9, 236, 155
OH  12528 data   2, 208,  58, 169,   0, 141, 155,   2
LD  12536 data  32, 196, 119, 166,   9, 236, 156,   2
PD  12544 data 240,   5, 144,   3,  76,  47,  48,  32
AG  12552 data  89,  48,  32,  18, 192,  32, 228, 255
NE  12560 data 240, 248, 201,   3, 208,   3,  76,  47
NI  12568 data  48, 201,  81, 208,  11, 186, 142, 155
BD  12576 data   2, 169,   1, 141, 154,   2, 208,   5
HE  12584 data 169, 128, 141, 154,   2, 162,   0, 173
IH  12592 data  17, 208, 168,  41,  16, 240,  16, 152
LF  12600 data  41, 239, 141,  17, 208, 234, 234, 160
GF  12608 data  12, 202, 208, 253, 136, 208, 250, 120
JE  12616 data 169,  57, 141,   4, 221, 142,   5, 221
FH  12624 data 169, 129, 141,  13, 221, 173,  14, 221
OK  12632 data   9,   1, 141,  14, 221, 169,  48, 162
OI  12640 data 185, 141,  25,   3, 142,  24,   3, 166
OL  12648 data   9, 154, 165,   3,  72, 165,   4,  72
KI  12656 data 165,   5,  72, 165,   2,  76, 242,   2
```

Listing 2: *debug.pal*

```
GL  9 open2,8,1,"0:debug.o"
LN  10 sys700
OF  20 ; * debug source code    *
IE  30 ; * for the c128         *
GA  40 ; * by jean-yves lemieux *
NC  50 ; * rimouski, quebec     *
JO  60 ; * feb. 1989            *
MA  70 ; ***********************
GM  80 ;
LG  90 .opt o2
KN  100 ;
CC  110 bkby    =$02        ;bank byte
AL  120 pchi    =$03        ;prg counter hi
MH  130 pclo    =$04        ; "   "      lo
EN  140 sreg    =$05        ;cpu status reg
AM  150 areg    =$06        ;acc. reg.
OG  160 xreg    =$07        ;x    "
OH  170 yreg    =$08        ;y    "
MC  180 sptr    =$09        ;stack pointer
CB  190 hinmi   =$298       ;nmi ptrs
DN  200 lonmi   =$299
KC  210 cmdflg  =$29a       ;walk flag
PA  220 qflg    =$29b       ;quick "
AN  230 rflg    =$29c       ;return flag
KM  240 oldnmi  =$29d       ;storage for nmi
DD  250 brvec   =$316       ;break vector
BK  260 nmivc   =$318       ;nmi    "
PK  270 exmon   =$32e       ;exmon  "
OF  280 talo    =$dd04      ;timer a low byte
BE  290 tahi    =talo+1     ;
JN  300 icr     =$dd0d      ;int. cntl reg.
MN  310 cra     =$dd0e      ;control reg. a
JE  320 prcr    =$5598      ;bas. print <cr>
IM  330 gslow   =$77c4      ; "   slow cmd
JO  340 prsp    =$5604      ; "   print space
JN  350 meval   =$b7a7      ;mon eval entry
GD  360 primm   =$ff7d      ;kernal print
OF  370 getin   =$ffe4      ; "      get
CP  380 ;
DN  390 *=$3000 ;'sys12288'
GA  400 ;
GP  410 init    =*
KB  420 ;
HE  430         sei
OI  440         lda #<newbrk     ;break vector
MJ  450         ldy #>newbrk     ;will point to
EI  460         sta brvec        ;newbrk routine
CA  470         sty brvec+1
BK  480         lda #<wtwalk     ;exmon
FL  490         ldy #>wtwalk     ;point to
FL  500         sta exmon        ;wtwalk
AD  510         sty exmon+1
LC  520         lda #$40        ;normal
HH  530         ldx #$fa        ;nmi entry
MI  540         sta oldnmi
EL  550         stx oldnmi+1
EM  560         lda #0          ;init 'walk' &
FD  570         sta cmdflg      ;'quick' flags
HM  580         sta qflg
MN  590         cli
OG  600         brk             ;jump to newbrk
IN  610 ;
LI  620 rmon =* ;return to monitor
MO  630 ;
JN  640         dec pclo
MB  650         bne inmon
JM  660         dec pchi
EB  670 ;
BF  680 inmon =*
IC  690 ;
AF  700         jmp $b046       ;init mon. entry
MD  710 ;
KL  720 norm =*
AF  730 ;
CI  740         jmp $b0b2       ;exmon norm. entry
EG  750 ;
FG  760         wtwalk =*       ;read keyword for
HN  770 ;                       ;walk command
NK  780         cmp #"w"
NC  790         bne norm
OI  800         jsr meval       ;evaluate cmd
IF  810         bcs norm
GI  820         ldx $60         ;store addr
JD  830         ldy $61         ;taken from
EE  840         lda $62         ;opl
KP  850         sta bkby
EF  860         stx pclo
OD  870         sty pchi
MN  880         tsx             ;store stack ptr
IP  890         stx rflg        ;for 'rts' eval.
AG  900         lda #<walk
DP  910         ldx #>walk      ;jump to walk
DL  920         sta $fa         ;routine via
LH  930         stx $fb         ;z-page
IO  940         jmp (!$fa)
MC  950 ;
HD  960 direg =* ;display registers
AE  970 ;
OK  980         jsr prcr
DA  990         jsr primm
EE  1000 .asc "sr ac xr yr sp pc"
OK  1010 .byte $0d,0
KP  1020        ldx bkby        ;is it a basic
KI  1030        lda pchi        ;or kernal call
EI  1040        cmp #$40
JB  1050        bcc d1
IN  1060        ldx #$0f        ;so, set 'bank 15'
FB  1070 d1     stx $68
DB  1080        sta $67
AK  1090        lda pclo
EC  1100        sta $66
JN  1110        ldy #$00
NN  1120 d2     lda !sreg,y
DJ  1130        jsr $b8a5       ;display 2-char
KC  1140        iny             ;ascii for reg
FO  1150        cpy #5
KO  1160        bcc d2          ;& 5-char ascii
JE  1170        jsr $b892       ;for pc
HO  1180        ldy #0
JE  1190        ldx $2aa        ;store 'fetvec'
KA  1200        stx $4d
AD  1210        jsr $b11a       ;mon indfet entry
HO  1220        jsr $b659       ;test code in acc
KD  1230        jsr $b608       ;mon disassembly
EP  1240        ldx $4d
HK  1250        stx $2aa        ;restore 'fetvec'
MK  1260        jmp prcr
MG  1270 ;
GH  1280 ;
NE  1290 rest =* ;restore nmi vec
KI  1300 ;
AF  1310        lda oldnmi
MH  1320        ldy oldnmi+1
GD  1330        sta nmivc
HI  1340        sty nmivc+1
CD  1350        rts
GM  1360 ;
AN  1370 ;
CC  1380 newbrk =* ;break routine
EO  1390 ;
GH  1400        lda #$80        ;regain control
MP  1410        sta cra         ;from timer
JL  1420        lda icr
AJ  1430        jsr rest
KB  1440        cld
FD  1450        pla             ;get reg contents
BC  1460        sta bkby        ;from stack
HP  1470        pla             ;and store in z-p
GF  1480        sta yreg
KF  1490        pla
JG  1500        sta xreg
OG  1510        pla
GG  1520        sta areg
CI  1530        pla
MI  1540        sta sreg
GJ  1550        pla
EL  1560        sta pclo
KK  1570        pla
EK  1580        sta pchi
AD  1590        tsx             ;store stack pntr
JH  1600        stx sptr
IN  1610        cli
GB  1620        lda sreg        ;get cpu status
DN  1630        and #$10        ;break bit set
OO  1640        beq n1          ;no then continue
MM  1650        jmp rmon
PM  1660 n1     bit cmdflg      ;if bit 7 set
LL  1670        bmi ckrfg       ;then do 'walk'
LB  1680        ldx sptr        ;did we reached
MA  1690        cpx qflg        ;the end of
MK  1700        bne delay       ;of subroutine
NN  1710        lda #0          ;yes, stop running
PL  1720        sta qflg        ;and walk
ID  1730 ;
DH  1740 ckrfg =* ;check for last rts
ME  1750 ;
JB  1760        jsr gslow
FO  1770        ldx sptr
IK  1780        cpx rflg
LG  1790        beq wtcmd
HD  1800        bcc wtcmd
NN  1810        jmp inmon
CJ  1820 ;
DO  1830 wtcmd =* ;wait for new cmd
GK  1840 ;
GD  1850        jsr direg
BK  1860 w1     jsr $c012       ;check kbd matrix
BI  1870        jsr getin       ;get char
IK  1880        beq w1
BA  1890        cmp #$03        ;stop key pressed
LB  1900        bne quick
BE  1910        jmp inmon
GP  1920 ;
OH  1930 quick =* ;full speed cmd
KA  1940 ;
NC  1950        cmp #"q"
DB  1960        bne walk
```

```
BK  1970        tsx             ;store 'return'
FF  1980        stx qflg        ;address
FO  1990        lda #1          ;set 'quick' flg
PJ  2000        sta cmdflg
LG  2010        bne delay
KF  2020 ;
NO  2030 walk =* ;walk cmd rout
OG  2040 ;
HO  2050        lda #$80        ;set 'walk' flg
LN  2060        sta cmdflg
MI  2070 ;
JC  2080 delay =* ;delay for raster
AK  2090 ;
LH  2100        ldx #0
HN  2110        lda $d011
HB  2120        tay
KD  2130        and #$10
PJ  2140        beq d4
NB  2150        tya
IJ  2160        and #$ef
BF  2170        sta $d011
AF  2180        nop:nop
NF  2190        ldy #$0c
OL  2200 d3     dex
OM  2210        bne d3
HG  2220        dey
CO  2230        bne d3
HA  2240 d4     sei
JN  2250        lda #$39        ;set clock timer
PM  2260        sta $dd04       ;in cia2
NE  2270        stx $dd05
OA  2280        lda #$81        ;enable timer a
NF  2290        sta icr
EP  2300        lda cra         ;start timer
GO  2310        ora #1
AG  2320        sta cra
NN  2330        lda #>newbrk
HE  2340        ldx #<newbrk
JB  2350        sta nmivc+1
IJ  2360        stx nmivc
ND  2370        ldx sptr
IE  2380        txs
MM  2390 ;
JK  2400 jmpfar =* ;prepare  jmpfar
AO  2410 ;
OK  2420        lda pchi
KP  2430        pha
GO  2440        lda pclo
OA  2450        pha
GO  2460        lda sreg
CC  2470        pha
KB  2480        lda bkby
PK  2490        jmp $2f2        ;jmpfar entry
```

Shortest Catalog in BASIC 2.0?
Michael Gilsdorf, Toledo, OH

Here's a little four-liner for the C64 (or VIC) that will get you an on-screen disk directory in a hurry. It features a pause function that can be toggled on or off by pressing any key. No ML code - so you can easily tailor it to your needs (change device or drive number, display specific files, etc.). It may very well be the shortest, fastest BASIC directory routine with a pause feature.

```
AI   10 t=1:x=12:n$=chr$(0):p=198:q=255:y=13
        :printchr$(147):opent,8,0,"$0":get#t,a$
JJ   20 get#t,a$,a$,a$,a$,b$,c$
        :printasc(b$+n$)*256+asc(a$+n$)c$;:fori=ttox
KD   30 get#t,a$,b$:printa$b$;:next:print
        :ifb$<>""thenx=y:waitp,q,t:pokep,0:goto20
EG   40 closet
```

Don't Assume Device 8!
Michael Gilsdorf, Toledo, OH

If you're writing a program that loads, saves, or otherwise accesses the disk drive, don't assume the default is always device 8, drive 0. Allow users the option to use drive 1 (for dual drives) and devices 8, 9, 10 and 11 as well. Programs which allow the use of multiple devices and drives eliminate the need to have the user swap program and data disks.

So how do you tell which device numbers the user may want to use? Simple! First, PEEK location 186 to tell what device number was used to load the program file (last device number accessed). Use this same number if the program will be loading any additional program files. This location is the same on both the C64 and C128. Second, by opening and closing the device, then reading the STatus, you can tell what devices are present. Here's a short and simple BASIC routine that demonstrates this. It checks the last device number accessed, which device numbers are present, and the type of drive.

```
PC   10 rem device number check -- by michael gilsdorf
LP   20 dn=peek(186):print"device number";dn;
        ": accessed last"
AM   30 for dv=8 to 15:open 1,dv,15:close1
LK   40 print"device number";dv;": ";
        :a$="not present":if st<0 then 70
OP   50 open1,dv,15,"uj":for d=1 to
         1000:next:input#1,a,a$:close1
GC   60 a$=right$(a$,4):if left$(a$,1)<>"1"
        then a$="drive unknown"
JA   70 print a$:next
```

Disk Partitions On The 1571
M. Garamszeghy, Toronto, ON

Being a developer of software for the C128 in both its native mode and CP/M mode, I frequently send out program disks to various people for 'beta testing' (i.e. testing of the programs by others before release to the general public). In order to save on disk and mailing costs, I sometimes send out more than one program on a disk.

Sometimes I even send out CP/M and C128 software on the same disk. Since I do not like using flippy disks, I have developed a method to partition a 1541 or 1571 disk so that it can be used by both CP/M and CBM DOS at the same time. The program listed below gives you just over 70K available to CP/M and about 70K (for a single-sided 1541 disk) or 240K (for a double-sided 1571 disk) for use by normal CBM DOS. (The numbers include the inefficiencies in disk utilization caused by the chosen CP/M format.)

www.Commodore.ca
May Not Reprint Without Permission

The program works by formatting the disk in CBM DOS mode normally, then reserving tracks 1 to 17 with the DOS block-allocate command. You then write a blank CP/M directory (i.e. all bytes set to hex $e5) to track 3, and presto you have a C64 style CP/M disk for use in C128 CP/M mode (or C64 CP/M if you have the CP/M cartridge) occupying the lower half of the disk and a CBM DOS disk in the upper tracks.

It should be noted that there are some limitations to this technique. Firstly, you must not validate or 'collect' the disk in CBM DOS mode. This would de-allocate the reserved CP/M tracks. Secondly, you must not put more than about 70K of stuff in the CP/M area or else you will overwrite the CBM DOS BAM, directory, and data tracks.

"partition.bas"

```
IE  10 rem   *********************************
BJ  20 rem   partition v 1.0
EA  30 rem   <c> 1988 herne data systems ltd.
GG  40 rem   *********************************
GK  50 :
KF  60 dv=8                                   : rem  device#
DA  70 print "{clr}           partition v1.0"
DK  80 print   " <c> 1988 herne data systems ltd."
OJ  90 print : print
DM  100 input "enter disk name,id code ";na$,id$
HE  110 print : print "insert new disk in device .."dv
CI  120 print : print "then press a key to continue ...."
HN  130 getkey a$
AO  140 print : print "formatting disk ==> "na$+","+id$
HF  150 open 15,dv,15,"n0:"+na$+","+id$
JM  160 input#15,ex$ : print
OG  170 print#15,"i0"
CG  180 for t=1 to 17
GJ  190 print chr$(27)"jallocating cp/m space ... track ==>"t;
JF  200 for s=0 to 20
AA  210 print#15,"b-a: 0";t;s
NN  220 next s,t
HA  230 open 2,dv,2,"#"
ED  240 print : print
EB  250 print"creating cp/m directory ..." : print
NK  260 for b=1 to 256
FD  270 print#2,chr$(229);
MB  280 next
FA  290 for s=0 to 8
KI  300 print chr$(27)"jwriting cp/m directory ... sector ==>"s;
PI  310 print#15,"u2: 2 0 3";s
EE  320 next
JK  330 close 2 : close 15
IL  340 print : print "==> done <=="
```

AWARD WINNING*
BIG BLUE READER 128/64
File Transfer Utility

Big Blue Reader 128/64 is ideal for those who use IBM PC compatible MS-DOS computers at work and have the Commodore 128 or 64 at home.
Big Blue Reader 128/64 is not an IBM PC emulator, but rather it is a quick and easy to use program for transferring word processing, text and ASCII files between Commodore and IBM MS-DOS diskettes.
Both C128 and C64 applications are on the same disk. ***1571 or 1581 disk drive is required. Does not work with 1541 type drives.***
BBR transfers 160K-360K 5.25 inch & 720K 3,5 inch MS-DOS disk files.
Big Blue Reader 128 supports: C-128 CP/M files, 17xx RAM exp, 40 & 80 column modes and more.
Big Blue Reader 64 is available separately only **$29.95**

BIG BLUE READER 128/64 only $44.95

Order by check, money order, or COD.
Free shipping and handling. No credit card orders please.
BBR 128/64 is available as an upgrade to current users
for $18 plus original BBR disk. Foreign orders add $4
CALL or WRITE for more information.

NEW – **BIBLE SEARCH** – Complete KJV New Testament with very fast word and verse search capabilities. Complete Concordance. Word(s) in text can be found and displayed in seconds. Includes both C64 and C128 mode programs. Please specify 1541, 1571 or 1581 formatted disk. **only $25.00**

To order Call or write:
SOGWAP Software
115 Bellmont Road; Decatur, IN 46733
Ph (219) 724-3900

**Big Blue Reader was voted the best utility program by RUN's 1988 Reader Choice Awards.*

JASON-RANHEIM
CARTRIDGE MATERIALS
FOR YOUR COMMODORE 64 or 128

Quality Products
from the World Leader!

- Promenade C1 EPROM Programmer
- Game Type Cartridges
- Bank Switching Cartridges
- RAM/ROM Combination Cartridges
- Capture Archival Cartridge System
- Cases, EPROMS, Erasers, Etc.

Call or write for complete information!

Call Toll Free 800-421-7731
from California 916-878-0785
Tech Support 916-878-0785

JASON-RANHEIM
3105 Gayle Lane
Auburn, CA USA 95603

Top-Tech International, Inc.
Advanced Computer Systems

INDUSTRY FIRST — LIFETIME COMPUTER"

Lifetime Warranty—available for any C-64 computer serviced and/or sold by us!!
Flat Service Rates — FAST, Professional Service
Full line of CBM computers, peripherals & parts; **C-64 Power Supply with 3-yr warranty;**
1531 Datasette — $19.95; **Hard-to-find parts** (STR-54041); Service Manuals; VIC-20 and
C-64 Cartridges & Tapes: $3.00 ea.; 10 for $25.00 ("Pot Luck" — No exchanges/returns).
VISA, MASTERCARD, DISCOVER, AMEX
Orders ONLY: FAX — (215) 389-5920 or CALL — (800) 843-9901
No extra charges for our GIs! We want your business!!!
(215) 389-9901 • 1112 S. Delaware Ave., Philadelphia, PA 19147 • **(215) 389-9901**

The ML Column

Two Kinds of Numbers

by Todd Heimarck

I want to start by explaining how I write this column. The kind editor (KE) lets me know two weeks before the deadline that he needs to fill up some pages in the magazine and that I should write another column. I say to myself, "Well, if it were me, topic XYZ would be interesting." Sometimes I suggest the idea to the KE, who usually says either "Fine" or "No, we did that two years ago."

But I'm never sure if the XYZ topic interests *you*. You buy this magazine; you should get a vote. If there's something you want to see, let me know. Send a letter to: *The ML Column, Transactor*, 85 West Wilmot St., Unit 10, Richmond Hill, ON, Canada, L4B 1K7 (they'll forward it to Seattle). Or leave electronic mail on CompuServe to ID 76703,3051.

If you saw the last issue (Volume 9, Issue 5), you saw the letter from Barry Kutner. He wants to read more about input/output routines in machine language. Sounds good to me. We'll look at I/O in the next issue.

This issue we'll finish the big numbers idea from last issue.

Kinds of people

Someone once said that there are two kinds of people in the world: people who think there are two kinds of people and people who don't.

For thousands of years, mathematicians have made a less trivial distinction. They divide whole numbers into primes and composites. Each prime number is divisible only by 1 and itself; it has no other divisors. Every composite number is divisible by two or more primes. For example, 650 breaks down into 2 * 5 * 5 * 13. The numbers 2, 5, and 13 are primes; 650 is a composite.

There's no formula for testing primes and there probably never will be.

In the third century BC, a Greek mathematician named Eratosthenes invented a way to generate prime numbers. His method, The Sieve of Eratosthenes, is still in use because it's simple and it works. You go through a list and cross off all numbers that are composite. Whatever's left is a prime.

Let's say you want the prime numbers betwen 2 and 30. Write down the numbers. The first prime is 2. Now you cross out all multiples of 2 - the even numbers 4, 6, 8, 10, and so on. Next on the list is 3, another prime. Cross out 6, 9, 12, 15, and so on. Although 4 comes after 3, it's been crossed off, being a multiple of 2, so you skip ahead to 5. The remaining primes (after some more crossing out) are 7, 11, 13, 17, 19, 23, and 29.

Running the program

The program *Primes* calculates all prime numbers up to 8,386,549. To run it, just **sys 49152**. It prints them to the screen. If you prefer, you can redirect output to a disk file or the printer (with **open 4,4:cmd 4:sys 49152**, for example).

Be prepared to wait; it takes nearly four hours to print all of the primes.

If you're curious about how I fit 8,000,000+ variables into a program, I'll admit that I cheated a bit. You must have a RAM Expansion Unit (REU) installed. I wrote it for a 1750 REU (512K), but it should work just as well with the 1764 (256K) or the 1700 (128K). If you have less than 512K, change the variable REUTYPE at the end of the program. Putting a smaller number into REUTYPE also makes the program run faster; theoretically, every time you cut the value in half, you get half as many primes, but the program finishes in half as much time.

It runs on a 64, but you can reassemble it to a new location in the range 0-16384 and run it without modification on a 128. I tested it at location 5000. Two notes for 128 users: Enter a **bank 15** command before SYSing to the program (to make the Kernal ROM and REU registers visible) and don't run the program in FAST mode. The RAM Expander doesn't like FAST mode.

Let the data write the program

This is one of those programs that's built around the data structure. Once you figure out how to fit the data in memory, the program almost writes itself.

Begin with the 1750 REU's memory of 512K. That should be enough for 524,288 byte-sized variables. We don't need entire bytes, though, because each variable has only two possible states: prime or not-prime. That amount of information can fit into a bit. We'll arbitrarily decide that 1 means prime and 0 means composite. There are eight bits in a byte, so we have room for about 4,000,000 variables.

There's one more trick to stretch the data. We can ignore all even numbers, which always end with a zero in base two, anyway. We'll only deal with odd numbers. Byte 0 of the REU will hold eight bits representing the odd numbers 1, 3, 5, 7, 9, 11, 13, and 15. In byte 1, the bits are 17-31. In byte 2, the bits are 33-47, and so on.

The program has two primary subroutines named FILLREU and PRIMES. The first fills up memory with $ff bytes (because we start out assuming that all odd numbers are prime until they're crossed off the list). The second prints out the primes, while whittling away at the composites.

Talking to the REU

The RAM Expansion Unit's 11 registers map into the addresses $df00-df0a on both the 64 and 128. The important ones are:

- DMACMD ($df01): a multipurpose command register. When you store a value here, the appropriate command executes. In bits 0- 1, the value 00 means STASH, 01 means FETCH, 10 means SWAP, and 11 means VERIFY. Bit 4 should be 0 if you want the command to execute immediately (if it's a 1, the command waits until a value is stored at $ff00, which is useful on the 128 in some situations). Bit 5 is the load flag. If it's 0, the addresses in DMAADL and DMALO are automatically incremented after a memory access. If it's 1, the addresses are restored to their original values. Bit 7 is the execute flag; it signals the REU to begin the operation specified in bits 0-1.
- DMAADL ($df02): two bytes that specify an address inside the computer. In this and other registers, the low byte is stored before the medium or high bytes.
- DMALO ($df04): three bytes that specify an address inside the REU. Whether or not the addresses in DMAADL and DMALO increment depends on bit 5 of DMACMD (after an operation) and bits 6-7 of $df0A (during an operation).
- DMADAL ($df07): two bytes that specify the number of bytes to transfer. Up to 65,535 bytes can be transferred.
- DMAVER ($df0a): address control register. Bit 6 controls whether the REU memory increments during an operation (0 means yes, 1 means no). Bit 7 controls whether the system memory increments.

The FILLREU routine begins by putting the number $ff into MVAL, which happens to be location $00ff. We want to fill the whole REU with $ff because ones represent prime numbers and we assume that all numbers are prime until proven otherwise. That one byte will fill all 512K because an $80 is stored in DMAVER. Next, we put 4096 into NBYTES (a shadow of DMADAL) and set the addresses in C64MEM and REUMEM (shadows of DMAADL and DMALO). Then copy the shadow registers to the real REU registers in the COPYREGS subroutine. Then loop 128 times (or 64 or 32 times for a 1764 or 1700 RAM Expander).

When FILLREU is finished, the REU should contain nothing but ones.

Skipping over even numbers

We've already decided that we don't need to bother with even numbers. That means the program's outer loop has to count from 1 to 3 to 5 to 7, up to 8 million, two at a time.

In the inner loop where the multiples of x get zapped, we can count $2 * x$ numbers at a time. For example, if we discover that 5 is a prime number, the algorithm says that we cross off every fifth number: 10, 15, 20, 25, etc. But we're ignoring even numbers, so we needn't bother with 10, 20, 30, and the others. Start with 5, add 10 (making 15), add ten (25), add ten (35), and we'll zap only the odd multiples of 5.

The second major subroutine, called PRIMES, contains mostly JSRs to other routines in the program. Start out with the number 1 and clear that bit (meaning that 1 is not prime). Then the main loop (MAIN) begins. Add two to the number in BIG. BIG is similar to BIGSIX from the last column, but it holds only three bytes instead of six. A second three-byte number is TEST (used for the inner loop). A third is DOUBLE, which is just BIG times two.

The TOOBIG subroutine checks the value in TEST to see if the loop (inner or outer) should end because the number has grown too large.

TESTPRIM tests TEST to see if it's prime. When a prime is located, two things happen: PRINTIT prints it out (in ASCII decimal) and the routines in CPLOOP zap all multiples of TEST.

The PRINTIT routine is copied almost exactly from MAKEDEC from the BIG1.SRC program from last issue. It converts a big binary number into printable ASCII characters that provide a decimal (base ten) number. It also adds a comma and a space to separate the numbers.

With a 512K REU, there is a large delay of about 20 minutes between printing the number 3 and the number 5. There are a lot of multiples of 3 between 9 and 8.4 million (in that 20-minute pause, more than a million bits are turned off). Between 5 and 7, the delay is only about 12 minutes. The delay gradually decreases as the primes get bigger. I inserted the **inc 53280** line to increment the border colour on the 64 and on the 128 in 40-column mode. When the border flashes, you know the program is running and not locked up in an endless loop.

If you don't want to wait 20 minutes between 3 and 5, make REUTYPE a smaller number. $80 means 512K, $40 means 256K, and $20 means 128K. But there's no reason you couldn't used a smaller number such as $04 or $02.

Making bricks

In a previous column, I said that if BASIC is a pile of bricks from which you can build a house, then ML is like a pile of clay from which you make the bricks to build a house.

The trick, I think, is to make the bricks small enough. MAKEDEC from last issue printed out a decimal number. It needed only slight modifications to become PRINTIT this issue.

Programming is like musical composition. When you compose music, you have to keep the entire structure of the piece in mind at all times. But you can divide a symphony into movements. Movements break down into parts. Parts break down into phrases. Those are the bricks.

When I wrote the PRIMES subroutine, I divided the program into small modules that did specific tasks. For example, I typed **jsr getmval**, knowing that I would eventually write a routine that would grab a byte from the REU and put it in MVAL. I didn't have the routine written yet, but I knew how to write it.

We've done enough with big three-byte and six-byte numbers. In the next column, we'll look into I/O.

If you'd like to do something with big integers, here's an idea. Set aside a 16K section of memory (in the computer, not the REU). If you store only odd primes, that's enough memory to handle values up to 262,143. Next, ask the user to input a number up to about 68 billion (see the GSTRING routine from BIG1.SRC in Volume 9, Issue 5). Now figure out its factors. If the binary number ends with a zero, it's divisible by two, so print a 2 and shift to the right. If not, take the square root (see BIG2.SRC) and call that MAX. That's the highest possible factor if it is a square. Run through the prime numbers from 3 to MAX and see if they divide into the target number (see Volume 9, Issue 2). If you find a factor, calculate the new value of MAX and repeat the loop until you find all of them.

Listing 1: *primes.src*

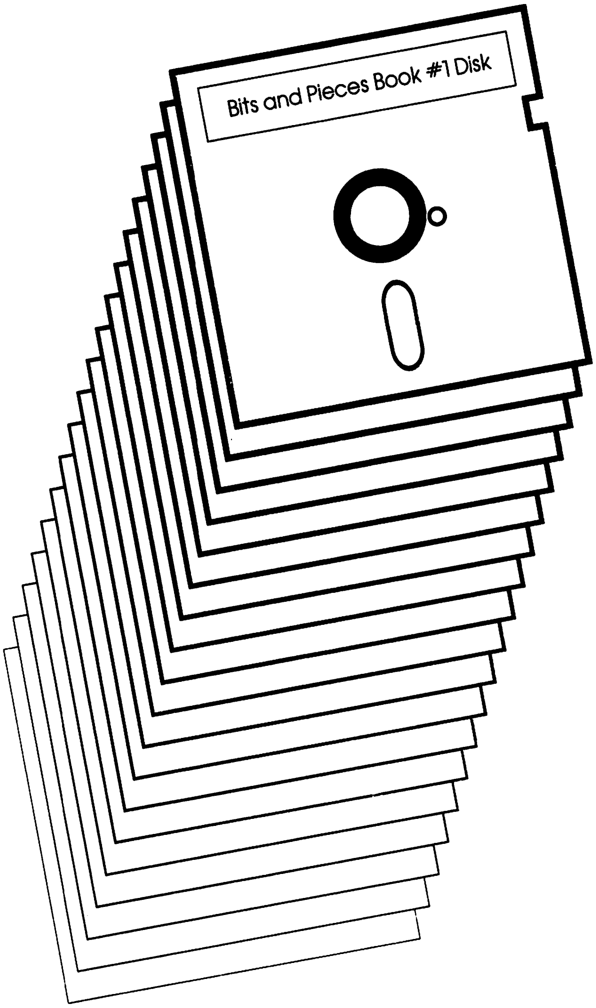
```
FG  10 rem save"primes.src",8
FO  20 sys700
OF  30 *=49152
AJ  40 .opt oo
KP  50  mval   = $ff   ; zero-page location for value to fetch or stash
CM  60  scmnd  = 144   ; stash command
OG  70  fcmnd  = 177   ; fetch command
HH  80  chrout = $ffd2
LH  90  dmacmd = $df01 ; command for reu
PC  100 dmaadl = $df02 ; c64 memory address
JP  110 dmalo  = $df04 ; reu memory address
KE  120 dmadal = $df07 ; number of bytes
BH  130 dmaver = $df0a ; if address increments
CA  140 ;
GO  150        jsr fillreu    ; fill with 1s
HK  160        jsr primes     ; print all primes
GJ  170        rts
KC  180 ;
CH  190 fillreu = *
EE  200        lda #$ff       ;  the fill byte
PD  210        sta mval       ;  the location in 64 memory
GD  220        lda #$80       ;  don't increment 64 memory
LG  230        sta dmaver     ;  reu register
BB  240        lda #<4096     ;  4k at a time
AF  250        sta nbytes     ;  number of bytes
CP  260        lda #>4096
OI  270        sta nbytes+1
GA  280        lda #<mval     ;  location in 64 memory
GC  290        sta c64mem
EB  300        lda #>mval
MP  310        sta c64mem+1
EN  320        lda #0         ;  location in reu memory
ON  330        sta reumem
KK  340        sta reumem+1
IL  350        sta reumem+2
ON  360 ;
NO  370        jsr copyregs            ;  copy to reu registers
DD  380        ldx reutype
KA  390 frloop jsr stash               ;  stash many times
BB  395        lda #>4096:sta dmadal+1
HE  400        dex
FB  410        bne frloop
MP  420        lda #1:sta nbytes       ;  from now on, one byte at a time
DI  430        lda #0:sta nbytes+1
PN  440        lda #$c0:sta dmaver     ;  don't increment any addresses
JM  450        jsr copyregs
IL  460        rts
ME  470 ;
GD  480 copyregs = *
AH  490        ldy #6         ;  seven registers
KO  500 crloop lda c64mem,y   ;  from memory
IL  510        sta dmaadl,y   ;  to the reu
DM  520        dey
MK  530        bpl crloop
IA  540        rts
MJ  550 ;
EH  560 fetch = *
OG  570        lda #fcmnd     ;  fetch command
AL  580        bne doit       ;  branch always
NJ  590 stash = *
DP  600        lda #scmnd     ;  stash command
KA  610 doit   sta dmacmd
IF  620        rts
EH  630 primes = *
LJ  640        jsr number1    ;  start with $000001
HI  650        jsr getmval    ;  fetch bit for 1
NL  660        jsr clbit      ;  clear that bit
MG  670 main   jsr addtwo     ;  add 2 to big
CJ  680        jsr big2test   ;  copy big to test
JF  690        jsr toobig     ;  is it too big
OM  700        bcc more       ;  keep going if ok
AM  710        rts            ;  else get out of primes (because we're done)
GL  720 more   jsr testprim
GD  730        beq main       ;  if equal, not a prime
IL  740        jsr printit    ;  if not equal, we have a prime, so print it
KI  750        jsr times2     ;  multiply by two
```

```
PD  760 cploop jsr composit   ;  add test = test + double
LM  770        inc 53280
EF  780        jsr toobig
HK  790        bcs main       ;  too big, back to the next prime
NF  800        jsr getmval    ;  fetch from reu
DM  810        jsr clbit      ;  clear that bit, it isn't a prime
BO  820        jmp cploop
EL  830 ;
HE  840 number1 = *           ;  start with the number $000001
PD  850        lda #1
LK  860        sta big
BF  870        lda #0
IJ  880        sta big+1
DK  890        sta big+2
BP  900        jsr big2test
KH  910        rts
OA  920 ;
DH  930 big2test = *          ;  copy three bytes from big to test
KJ  940        lda big:sta test
MJ  950        lda big+1:sta test+1
KK  960        lda big+2:sta test+2
GL  970        rts
KE  980 ;
NL  990 getmval = *           ;  get a value from reu and put it in mval
JB  1000       lda test       ;  copy test to reumem
GI  1010       sta reumem
HJ  1020       lda test+1
MF  1030       sta reumem+1
NK  1040       lda test+2
EH  1050       sta reumem+2
GK  1060       jsr rotreu           ;  rotate reumem to right
PH  1070       lda reumem:and #7
IA  1080       sta bitloc           ;  bit location (0-7)
GI  1090       jsr rotreu:jsr rotreu:jsr rotreu
DF  1100       jsr copyregs
OM  1110       jsr fetch            ;  get the byte
PJ  1120       lda mval
GF  1130       rts
KO  1140 ;
HH  1150 rotreu lsr reumem+2:ror reumem+1:ror reumem:rts
OP  1160 ;
EO  1170 clbit = *            ;  clears a bit (call fetch first)
CL  1180       ldx bitloc     ;  bit location 0-7
DH  1190       lda mval       ;  value in memory
OI  1200       and bitoff,x   ;  clear the bit
HD  1210       sta mval
LM  1220       jsr copyregs
BG  1230       jsr stash      ;  store back in reu
EM  1240       rts
IF  1250 ;
AA  1260 addtwo = *           ;  adds two to big
MG  1270        clc
BB  1280        lda big
LO  1290        adc #2
DG  1300        sta big
IA  1310        lda big+1
FA  1320        adc #0
KF  1330        sta big+1
HC  1340        lda big+2
DC  1350        adc #0
JH  1360        sta big+2
GE  1370        rts
KN  1380 ;
```

```
ON  1390 toobig = * ;checks test for out of range (about 8 million for 512k reu)
GA  1400        lda test+2 ; high byte of test
CC  1410        cmp reutype
KN  1420        rts          ; carry set means error/too big, clear means it's ok
MA  1430 ;
GC  1440 testprim = *
JA  1450        jsr getmval
NE  1460        ldx bitloc
KC  1470        and biton,x
EL  1480        rts
IE  1490 ;
DE  1500 printit = *
JI  1510         lda #0:pha
IB  1520 mdlp1   ldx #24                  ;  3 bytes = 24 bits
IG  1530         stx count
ML  1540         lda #0:sta temp
HI  1550 mdlp2   asl test:rol test+1:rol test+2
ON  1560         rol temp
PI  1570         lda temp
KN  1580         cmp #10:bcc mdcool
DM  1590         sbc #10
LO  1600         sta temp
FL  1610 mdcool  php
EF  1620         lsr test
CC  1630         plp
AF  1640         rol test
BE  1650         dec count
JC  1660         bne mdlp2
IA  1670         lda temp:ora #48         ;  make it an ascii number
MA  1680         pha
DF  1690         lda test:ora test+1:ora test+2
AF  1700         bne mdlp1
KC  1710 priloop pla
DM  1720         beq prend
JJ  1730         jsr chrout
DG  1740         jmp priloop
HM  1750 prend   jsr big2test             ;  put test back
PO  1760         lda #44:jsr chrout       ;  comma
BB  1770         lda #32:jsr chrout       ;  space
AO  1780         rts
EH  1790 ;
EL  1800 times2 = *
AM  1810         lda big:asl:sta double
BE  1820         lda big+1:rol:sta double+1
NE  1830         lda big+2:rol:sta double+2
MB  1840         rts
AL  1850 ;
GK  1860 composit = *
EM  1870         clc
FB  1880         lda test:adc double:sta test
LP  1890         lda test+1:adc double+1:sta test+1
NA  1900         lda test+2:adc double+2:sta test+2
CG  1910         rts
OG  1920 reutype .byte $80; $80 means 512k, $40 is 256k, $20 is 128k
PJ  1930 biton   .byte   1,   2,   4,   8,  16,  32,  64, 128
EH  1940 bitoff  .byte 254, 253, 251, 247, 239, 223, 191, 127
OO  1950 e = *
AC  1960 c64mem = e            ; 2 bytes (64k)
NH  1970 reumem = e+2          ; 3 bytes (512k)
BE  1980 nbytes = e+5          ; 2 bytes
JP  1990 big    = e+7          ; 3 bytes
HC  2000 test   = e+10         ; 3 bytes
OK  2010 double = e+13         ; 3 bytes
HL  2020 bitloc = e+16         ; 1 byte
HM  2030 count  = e+17         ; 1 byte
IO  2040 temp   = e+18         ; 1 byte
```

Listing 2: *primes.gen*

```
HO  100 rem  generator for "primes.obj"
FL  110 n$="primes.obj": rem  name of program
KB  120 nd=468: sa=49152: ch=70208

(for lines 130-260, see the standard generator on page 5)

AC  1000 data  32,   7, 192,  32, 105, 192,  96, 169
IO  1010 data 255, 133, 255, 169, 128, 141,  10, 223
CG  1020 data 169,   0, 141, 217, 193, 169,  16, 141
BG  1030 data 218, 193, 169, 255, 141, 212, 193, 169
FP  1040 data   0, 141, 213, 193, 169,   0, 141, 214
OC  1050 data 193, 141, 215, 193, 141, 216, 193,  32
MB  1060 data  83, 192, 174, 195, 193,  32,  99, 192
PL  1070 data 169,  16, 141,   8, 223, 202, 208, 245
CE  1080 data 169,   1, 141, 217, 193, 169,   0, 141
KI  1090 data 218, 193, 169, 192, 141,  10, 223,  32
AL  1100 data  83, 192,  96, 160,   6, 185, 212, 193
BI  1110 data 153,   2, 223, 136,  16, 247,  96, 169
DG  1120 data 177, 208,   2, 169, 144, 141,   1, 223
JI  1130 data  96,  32, 157, 192,  32, 193, 192,  32
OO  1140 data 250, 192,  32,  11, 193,  32, 174, 192
NB  1150 data  32,  37, 193, 144,   1,  96,  32,  44
LB  1160 data 193, 240, 239,  32,  54, 193,  32, 144
KG  1170 data 193,  32, 166, 193, 238,  32, 208,  32
JF  1180 data  37, 193, 176, 222,  32, 193, 192,  32
AK  1190 data 250, 192,  76, 137, 192, 169,   1, 141
JK  1200 data 219, 193, 169,   0, 141, 220, 193, 141
FC  1210 data 221, 193,  32, 174, 192,  96, 173, 219
IB  1220 data 193, 141, 222, 193, 173, 220, 193, 141
CC  1230 data 223, 193, 173, 221, 193, 141, 224, 193
ID  1240 data  96, 173, 222, 193, 141, 214, 193, 173
GD  1250 data 223, 193, 141, 215, 193, 173, 224, 193
IA  1260 data 141, 216, 193,  32, 240, 192, 173, 214
LB  1270 data 193,  41,   7, 141, 228, 193,  32, 240
HI  1280 data 192,  32, 240, 192,  32, 240, 192,  32
GF  1290 data  83, 192,  32,  95, 192, 165, 255,  96
AF  1300 data  78, 216, 193, 110, 215, 193, 110, 214
JO  1310 data 193,  96, 174, 228, 193, 165, 255,  61
JL  1320 data 204, 193, 133, 255,  32,  83, 192,  32
PO  1330 data  99, 192,  96,  24, 173, 219, 193, 105
JB  1340 data   2, 141, 219, 193, 173, 220, 193, 105
HA  1350 data   0, 141, 220, 193, 173, 221, 193, 105
OI  1360 data   0, 141, 221, 193,  96, 173, 224, 193
EL  1370 data 205, 195, 193,  96,  32, 193, 192, 174
OO  1380 data 228, 193,  61, 196, 193,  96, 169,   0
JJ  1390 data  72, 162,  24, 142, 229, 193, 169,   0
BM  1400 data 141, 230, 193,  14, 222, 193,  46, 223
LN  1410 data 193,  46, 224, 193,  46, 230, 193, 173
NJ  1420 data 230, 193, 201,  10, 144,   5, 233,  10
AM  1430 data 141, 230, 193,   8,  78, 222, 193,  40
LB  1440 data  46, 222, 193, 206, 229, 193, 208, 219
JA  1450 data 173, 230, 193,   9,  48,  72, 173, 222
BP  1460 data 193,  13, 223, 193,  13, 224, 193, 208
JN  1470 data 192, 104, 240,   6,  32, 210, 255,  76
EK  1480 data 121, 193,  32, 174, 192, 169,  44,  32
KE  1490 data 210, 255, 169,  32,  32, 210, 255,  96
IA  1500 data 173, 219, 193,  10, 141, 225, 193, 173
CB  1510 data 220, 193,  42, 141, 226, 193, 173, 221
AM  1520 data 193,  42, 141, 227, 193,  96,  24, 173
JD  1530 data 222, 193, 109, 225, 193, 141, 222, 193
OD  1540 data 173, 223, 193, 109, 226, 193, 141, 223
DG  1550 data 193, 173, 224, 193, 109, 227, 193, 141
DG  1560 data 224, 193,  96, 128,   1,   2,   4,   8
EJ  1570 data  16,  32,  64, 128, 254, 253, 251, 247
AK  1580 data 239, 223, 191, 127
```



The Edge Connection

Societies, shows and disk drive voodoo

by Joel Rubin

The Toronto PET Users' Group (5333 Yonge St., Box 116, Willowdale, Ontario, Canada, M2M 6M2, telephone +1 416 733 2933, 1200-1700 Eastern Time), which stretches back to the time when Jack Tramiel was making watches, calculators, and the brand new PET 2001, seems to be gradually coming back after a moribund period. They are getting out newsletters, albeit a few months after the date on them, and the one office worker (formerly three) is working on filling disk orders, and, one of these days, they may even get out renewal notices.

JAMECO Electronics (1355 Shoreway Rd., Belmont, California, USA 94002, telephone +1 415 592 8097) is closing out ICs, including the Commodore custom chips which they were carrying. The chips that they still have are reduced in price, but some are already sold out.

The March, 1988 issue of the newsletter of the Commodore Owners Workshop (c/o Home Computing Center, Tanforan Park, San Bruno, CA), a local users' group just south of San Francisco, warns that Datel's MIDI interface, while it may work with many European programs, will not work with the American programs written to the Passport standard. Conversely, it is presumably the case that those who buy a Passport or Passport-compatible interface will not be able to run European software on it. God must have loved standards because He made so many of them.

Anti-rental law in the States?

According to a blurb in the 24 April *Christian Science Monitor*, Senator Orin Hatch (Republican, Utah) has introduced legislation to prohibit firms from renting or loaning software. The law is based on the Record Rental Act of 1985, which was passed when phonograph record producers complained that stores which rented records were, in fact, encouraging illegal copying and thus costing them money. (One can still borrow recordings from public libraries in the U.S.A., however.)

On the other hand, the loaning of video cassettes is a very big business, with the full co-operation of the recording industry, and many of the video stores also rent Nintendo cartridges which are, technically, software - although cartridges are usually more expensive to pirate than to buy. The Senator did accept an amendment which would exempt libraries at non-profit organizations - for example, a computer lab at a university.

CLONEDEX

The Fourteenth West Coast Computer Faire was held the weekend of March 17, back at its old home at Brooks Hall and the Civic Auditorium, near San Francisco City Hall, instead of the more impersonal and newer Moscone Center where it had been held the past few years. There was a point to this - the theme of the show was "Legends of the West", and the Faire was attempting to regain its glory years, back when the Apple I was sold by its creators from one of the mini-booths or when Adam Osborne introduced the first luggable.

Lee Felsenstein, the designer of the Processor Technology Sol and of the Osborne I, held a meeting of the long defunct Home Brew Computer Club; many of the Silicon Valley giants grew out of their meetings at Stanford. This was, for the most part, an excursion into nostalgia, and 'where are they now'. Jim Warren, the founder of the Computer Faire, and of *Infoworld*, had a seminar on the future. I hope that Jim's visions are more valid than the view one got from the present, which seems to be full of 80x86 clones.

Whereas Jim used to patrol the Faire on roller skates, the head of the company which now owns the Faire (The Interface Group of COMDEX fame, and MACDEX infame) was busy buying the Sands Hotel in Las Vegas. A seminar on older computers, which I thought might be interesting for 8-bit Commodore owners, turned out to be mostly about marketing orphans. Bob Cook, of Sun Remarketing, told how he built a multi-million dollar business selling Apple ///s and Lisas with Mac compatibility enhancements, mostly on Apple's money.

The keynote address was given by Philippe Kahn, of Borland International. Mr. Kahn spent some time bad-mouthing Lotus and, in fact, the newspaper here reported that, a week or so later, he was caught putting copies of an anti-Lotus article beneath the doors of a hotel at a Palm Springs event. Of course, Lotus version 3.0 has been vapourware for so long that a lot of people have been bad-mouthing Lotus, but it's in bad taste for a competitor to do so. Mr. Kahn said he had heard that Lotus 1-2-3 version 3.0 had acquired the internal name "Titanic", and that he hoped his company would make the iceberg.

Mr. Kahn did say one thing which Amiga programmers might want to keep in mind. He said that many programmers were becoming lazy because, faced with faster processors, and huge amounts of memory, they felt that they need not optimize for time or speed the way they would have had to on an older computer. He warned such programmers that multi-tasking would eat much of the speed they counted on; and that new graphics standards would eat much of the RAM; and that, if they aren't careful, their badly written programs will be swapped out to disk faster than one can say "128K Mac" (my phrase).

Of course, there wasn't much there for Commodore owners, or even Amiga owners. For 8-bit Commodores, there was a local store which sells both new and liquidated software, a couple of CP/M users' groups, Softdisk (*Loadstar*), Virgin/Mastertronic (which is going to take its Leisure Genius line back from Electronics Arts in the U.S.), and Elcomp selling its old C64 books and software at a discount, and that was just about it. By the way, Softdisk wants to put out an Amiga version, and is looking for contributors.

There was an Amiga store, a few games available, and a users' group. Poor Person Software (3721 Starr King Circle, Palo Alto, CA 94306, telephone +1 415 493 7234) had an Amiga program called *Thinker*. In essence, *Thinker* is a word processor which allows you to click on a phrase and either reference some more text or a picture. They claim that it's Hypertext. I don't know enough about the definition of Hypertext to decide that. (Speaking of Hypertext, someone ought to port something more or less like *Hypercard* to the Amiga. I'm not sure that it's quite as great as its boosters claim, but what it has done is to allow a lot of people whose expertise is outside the computer field to write programs reflecting their expertise on the Mac. *Hypercard* may have its deficiencies as a programming language, but many of the programs written in it probably wouldn't have been written without it, and some of these are quite useful.)

Humour, probably not intentional

If you can manage, see if you can find a copy of *Transactor*'s cousin magazine, *Commodore Computing International*, for the month of April (Fools'). On page 7, there's an ad for the company well-known for importing American software and hardware into Britain. One of the products being advertised is a nybbler/parameter package. You are, of course, familiar with the disclaimers that follow such ads. "While we don't condone piracy...", or "We strongly condemn piracy..." or some such blurb follows the claim that "Our package copies more copy-protected programs than any other." Well, it appears that there were two versions of this ad, and someone accidently (or because of a Freudian slip, or because they had just gotten fired and wanted to get back at the company or for some other reason) mixed them - leading to the statement: "While we strongly condone piracy..."

Reading 1581 'credit' messages

Also, on the humour front, if you have a 1581 disk drive, try entering the following program at disk RAM address $0300, and executing it:

```
error = $ff3f

org $0300

lda #$79
jmp error
```

Then, read the error channel. You will get the author's credit message. If you substitute **$7a** for **$79**, you will get a dedication to one of the authors' wives. Read the error channel using GET# rather than INPUT#, especially with **$7A**, since the error number gets printed as **7:**, and that colon plays havoc with INPUT#.

```
100 get#15,a$:print a$:if st=0 goto 100
```

Relative files and 96

In recent Commodore disk drive manuals, you have been instructed to give the relative file positioning command, **p**, in the form:

```
print#15,"p"chr$(sa or 96)chr$(reclo)
         chr$(rechi)chr$(ofs)
```

because in BASIC 7.0, RECORD ends up sending the disk drive this message. (This is because Kernal OPEN sets the $60 bits in the secondary address, and RECORD looks up the secondary address from the file number.)

I have looked at 1541, 1571, and 1581 disassemblies, and, with all of these drives, it doesn't matter. On the 1541 and 1571 the **p** command begins at $e207. On the 1581, it is vectored through to $a1a1. All of these routines are the same, except for specific addresses. The beginning looks like this:

```
jsr syntax
lda buf+1     ; this is the secondary address
sta tempsa
jsr getchannel
```

If a secondary address is greater than 18, GETCHANNEL lops off the high nybble, so if you add 96 to the secondary address, not only won't the gods of relative files appreciate your sacrifice, but the disk drive will just subtract it off and they won't even know about it. I use the word "gods" advisedly - I think the source file for Commodore DOS has just gotten too complicated, with too many patches between the olde 2040 and today. And, since no one really knows the whys and wherefores of some of the bugs, Commodore is just trying voodoo debugging. That sounds like programmers' hell - you've got this huge source file with zillions of patches, and half the programmers don't work at Commodore anymore, and you've got to try to maintain it!

I don't really know what's going on with secondary addresses 16, 17, and 18. Since most routines in the disk drive also lop off the high nybble of these three numbers, 16 and 17 yield the load/save channels of 0 and 1, respectively, and 18 usually is equivalent to 2. (Channel 0 is not used for C128 fast loads.) So, you can't use 16 or 17 for relative files, and 18 may confuse the disk drive.

Where you *do* have to 'or' 96 to the secondary address is when you call SECOND and TKSA. Actually, SECOND 'or's $20 to the secondary address, and TKSA 'or's $40, so you don't have to use $60 - just $40 for SECOND and $20 for TKSA. But, who wants to remember that? Doing both ($60) always works. I think that the problem is that, because of handshaking between the computer and the disk drive, the disk drive must be told to be both a talker and a listener whenever you send or receive data.

A neat 1581 trick from West Chester

It turns out that on the 1581, you can have the 128 boot sector wherever you want. Look at an official C128 1581 CP/M disk. (Not one with the Miklos G. format!) You'll notice that there's an autoboot user file on it ("copyright cbm 86"). When you boot, you send the string **ui** to the disk drive, and with the 1581, **ui** forces a search for and (if found) execution of a **&** file called "copyright cbm 86". What is in this mysterious file? In the case of a 1581 CP/M disk, it diverts the sector translation vector so that the first time the disk drive attempts a read, if it attempts to read track 1/sector 0, it actually reads track 40/sector 5. After the first read, even if the disk drive was trying to read another sector, the translation vector is restored. The boot sector is to be found on track 28/sector 5; the real track 1/sector 0 is the first sector of the CP/M directory.

```
****************************
* autoboot file on 1581    *
* cp/m disks, disassembled*
* with merlin's            *
* disassembler             *
****************************

jobs       = 2
hdrs       = $b
vtransts   = $1b8
jcbmbtrtn  = $ff5a

 org $300

             sei
             lda vtransts
             sta savead
             lda vtransts+1
             sta savead+1
             lda #<temptr
             sta vtransts
             lda #>temptr
             sta vtransts+1
             lda #$81
             sta $6d
             jmp jcbmbtrtn
```

```
temptr       ldx $83
             lda jobs,x
             cmp #$80       ;is it a read job?
             bne :no
             ldy $99
             ldx hdrs+1,y   ;is it on track 1?
             bne :no
             ldx hdrs,y
             dex            ;is it on sector 0?
             bne :no
             ldx #$28       ;if so, track 40
             stx hdrs,y
             ldx #$5        ; sector 5
             stx hdrs+1,y

:no          lda savead     ; now, restore translate sector vector
             sta vtransts
             lda savead+1
             sta vtransts+1
             hex 4c         ; jump
savead       ds  2          ; dummy address
```

Further applications of this technique are left to the reader. Of course, this effort is, for the most part, wasted in CP/M, since very few 1581s are hooked up as device 8, and CP/M must be booted from device 8. In the U.S., at least, one can no longer buy a C128 - only a C128D, and the separate 1571, if not officially dead, is almost impossible to find. The 1571 in the C128D has no DIP switches and changing the device number of the built-in 1571 from device 8 involves the old pad-cutting technique. However, the pads are not as accessible as they were in 1541s. If you have a 128D and a 1581, however, you can try booting from the 1581 by shutting off your 1581, flipping the DIP switches to make it device 8, soft-setting the built-in 1571 to device 9 (**open 1,8,15,"u0>"+chr$(9)**), turning on the 1581, and then booting. The 1581 must be set to device 8 by DIP switches, because when it receives the **ui** command from boot, it will read the switches.

Let's look at this real boot sector, track 40, sector 5. What it does is to fill $1000-$feff in bank 0 with NULL's, and then read in the four logical sectors beginning at track 40, sector 6, to $e000. These are the same as the two 512-byte physical sectors on side 0, beginning at track 39, sector 4. It then jumps to the Z-80 code beginning at $e000.

```
fillsp       = $1000
hdc0c        = $dc0c
hdc0d        = $dc0d
hdd00        = $dd00
z80code      = $e000
mmucr        = $ff00
setbnk       = $ff68
ioinit       = $ff84
setlfs       = $ffba
setnam       = $ffbd
open         = $ffc0
chkout       = $ffc9
clrchn       = $ffcc
z80on        = $ffd0
chrout       = $ffd2
z80wake      = $ffee

             org $b00

             txt 'cbm'
             ds  6
```

```
           sei
           jsr ioinit
           lda #$3f
           sta mmucr

********************************
* fill $1000 - $feff w/ 0      *
********************************

           lda #>fillsp
           sta $21
           lda #<fillsp
           sta $20
           ldx #$ef
           tay
:lup       sta ($20),y
           iny
           bne :lup
           inc $21
           dex
           bne :lup
           sta mmucr       ;bank 15

* open 15,8,15,name            *

           lda #$f
           ldx #8
           tay
           jsr setlfs
           lda #0
           tax
           jsr setbnk
           lda #4
           ldx #<name
           ldy #>name
           jsr setnam
           jsr open
           lda #$27        ;these are physical sectors--
           ldx #4          ;logically 40/6,7
           ldy #$e0
           jsr readsec
           lda #$27        ;logically
           ldx #5          ;40/8,9
           jsr readse2
           lda #$c3        ; z-80 jump
           sta z80wake
           lda #<z80code
           sta z80wake+1
           lda #>z80code
           sta z80wake+2
           lda #$3e
           sta mmucr
           jmp z80on

********************************
* read track .a, sector .x     *
* side 0 to .y*256             *
*                              *
* this routine reads physical  *
* sectors, so 512 bytes        *
********************************

readsec    sty $21
readse2    sta track
           stx sect
           ldx #$f
           jsr chkout
           ldy #6
:lup2      lda ecmd-1,y
           jsr chrout
           dey
           bne :lup2
           jsr clrchn
           bit hdc0d
           jsr getbyt
           ldx #2
           ldy #0
           sty $20
:lup3      jsr getbyt
           sta ($20),y
           iny
           bne :lup3
           inc $21
           dex
           bne :lup3
           lda hdd00
           and #$ef
           sta hdd00
           rts

getbyt     sei
           lda hdd00
           eor #$10
           sta hdd00
           lda #8
:wait      bit hdc0d
           beq :wait
           lda hdc0c
           rts

********************************
* this is a burst command      *
* sent to the disk drive       *
* in reverse, beginning with   *
* the first 'u' in name        *
********************************

ecmd       hex 01          ;read 512 bytes--1 physical sector
sect       hex 00
track      hex 00
           hex 00          ;read cmd--physical sector, side 0
           txt '0'

*===============================

name       txt 'u0'4c00    ; set the status byte
```

The four (logical) sectors of Z-80 machine language then partially replace the boot ROM in booting CP/M.

Save time on 1581 partitioning

When you make a partition on a 1581, if you want to make a directory, you have to enter the partition and do a long format on it. Typically, you format the disk, make partitions, and format the partitions - so you end up formatting the disk twice. Since the 1581 does not appear to use the disk or partition ID at the lowest level (the way the Commodore GCR drives do), you can save time by just doing a short format on the partition. But, there are complications.

If, within a partition, you try to write on a partition sector (e.g. doing a short format) you will get error 73 (**dos mismatch**) unless byte 2 of sector 0 of the first track of the partition contains a **d** ($44). You can avoid this by two methods - either write from the root or parent partition, or use the job queue. If you now do a short format in the directory, you will get a directory, but it will look a bit strange because it will have **chr$(0) + chr$(0)** for its ID. So, you should now write the ID to bytes 22-23 of sector 0, and to bytes 4-5 of sectors 1 and 2. See the partition program below for details.

```
BL  100 rem faster 1581 partition--avoids full formatting of partition
OE  110 rem by joel m. rubin
FO  120 rem run from parent directory
FO  130 input"device number";dn
CI  140 open1,dn,15,"m-r"+chr$(252)+chr$(255)
NE  150 get#1,a$:ifasc(a$)<>36thenprint"not a 1581":run
OJ  160 input"name of partition";na$
LN  170 input"first track";ft
PH  180 input"number of tracks (including 1 overhead)";nt
KJ  190 ns=40*nt:nh=int(ns/256):nl=ns-256*nh
CN  200 print#1,"/"na$","chr$(ft)chr$(0)chr$(nl)chr$(nh)",c"
DI  210 input#1,e,e$,t,s:ifethenprinte;e$t;s:stop
FG  220 open2,dn,2,"#0":print#1,"b-p:2,2":print#2,"d";:rem dos version
PN  230 print#1,"u2:2"0;ft;0:input#1,e,e$,t,s:ifethenprinte;e$t;s:stop
MC  240 print"name of directory  ";na$
FC  250 input"";nd$
HB  260 input"id of directory";id$
LO  270 iflen(id$)<>2goto260
LA  280 print#1,"/"na$:input#1,e,e$,t,s:ife<>2thenprinte;e$t;s:stop
AN  290 print#1,"n0:"nd$:input#1,e,e$,t,s:ifethenprinte;e$t;s:stop
NO  300 close2:open2,dn,2,"#0"
JO  310 fori=0to2
PA  320 print#1,"u1:2";0;ft;i:input#1,e,e$,t,s:ifethenprinte;e$t;s:stop
NJ  330 print#1,"b-p:2"(-22*(i=0))+(-4*(i<>0)):print#2,id$;
BL  340 print#1,"u2:2"0;ft;i:input#1,e,e$,t,s:ifethenprinte;e$t;s:stop
CG  350 next
OD  360 close2
FO  370 print"done--in new directory!"
```

Save time on 1571 single-sided formatting

There are two ways to do single-sided formatting on the 1571. First, you can do double-sided formatting, and then tell the BAM that you really did a single-sided format. CP/M does it this way. This isn't too slow, but it destroys flippies. Of course, the Aligner General has determined that flippies may be dangerous to the health of your system, but there are some programs which come on flippies and are inconvenient to use in any other format.

On the other hand, you could do what GEOS does - you can go into 1541 mode before you format. This is less dangerous to flippies. However, 1541 formats are notoriously slow. Here is a third method: you use exactly that part of the 1571 format routine, in the disk ROM, which formats side 0.

```
GI  100 printchr$(147)chr$(14);
OE  110 print"Format a single-sided disk using the 1571 format routine
OP  120 print"(c) 1989 Joel M. Rubin--commercial rights reserved
DD  130 open1,0
OK  140 fori=1to30:e$=" "+e$+chr$(157):next:d$=chr$(17)+chr$(17)
HN  150 u$=chr$(145)+chr$(145)+chr$(145)
OG  160 printd$"Insert disk to be formatted."d$
OM  170 printd$"Drive Number:  "e$"8"chr$(157);:input#1,dn:print
PI  180 if(dn<8)or(dn>11)thenprintd$e$u$u$:goto170
MG  190 open2,dn,15,"m-r"+chr$(103)+chr$(254):rem irq at $fe67 should be jmp ()
NF  200 get#2,a$:ifa$<>chr$(108)thenclose2:printd$"NOT A C'1571!"u$u$:goto170
BL  210 printd$"Disk name:  "1$"new"chr$(157)chr$(157)chr$(157);:input#1,dn$:print
OJ  220 if(len(dn$)=0)or(len(dn$)>16)thenprintu$;:goto210
BA  230 print:print"Insert disk to be formatted!"
EA  240 x=rnd(-ti):id$=chr$(65+26*rnd(0))+chr$(65+26*rnd(0))
FG  250 printd$"Disk id:  "1$id$chr$(157)chr$(157);:input#1,id$:print
IK  260 iflen(id$)<>2thenprintu$;:goto240
JB  270 print#2,"u0"chr$(190)"m1":bu=3:rem 1571 mode, working with buffer 3
CK  280 print#2,"m-w"chr$(18)chr$(0)chr$(2)id$
OP  290 print#2,"m-w"chr$(59)chr$(0)chr$(1)chr$(240):rem format @ $3b
IP  300 print#2,"m-w"chr$(162)chr$(2)chr$(1)chr$(36):rem < 36 tracks
HG  310 print#2,"m-w"chr$(178)chr$(1)chr$(1)chr$(0):rem side 0
DE  320 print#2,"m-w"chr$(bu)chr$(0)chr$(1)chr$(240):rem format
MA  330 gosub470
JI  340 ifc>1thenprintd$"Format error!":goto510
NG  350 print#2,"i0":bu=4:rem i0 reads 18/0 into buffer 4
JO  360 input#2,x,x$,t,s:ifxthenprintx;x$;t;s:stop
MI  370 print#2,"m-w"chr$(2)chr$(7)chr$(2)"a"chr$(0):rem--right dos, single sided
DB  380 tr=1
LK  390 print#2,"m-w"chr$(bu)chr$(0)chr$(1)chr$(144):rem write 18/0 w.o. err 73
BN  400 gosub470:ifc>1thentr=tr+1:iftr<=3goto390:rem try 3 times to write
OA  410 iftr=4goto510
AP  420 print#2,"i0":print#2,"n0:"dn$
JF  430 input#2,x,x$,t,s:ifxthenprintx;x$t;s:stop
GM  440 open3,8,2,"#":print#2,"u1:2,0,1,0"
NG  450 input#2,x,x$,t,s:ifxthenprintx;x$t;s:stop
EN  460 printd$"It worked!":close2:end
BC  470 print#2,"m-r"chr$(bu)chr$(0):get#2,a$:c=asc(a$)
HC  480 ifc>=128goto470:rem not done
GA  490 return
HH  500 rem-convert job code to ds error
BN  510 m$="m-w"+chr$(0)+chr$(3)+chr$(7)+chr$(169)+chr$(c)+chr$(162)+chr$(bu)
HK  520 m$=m$+chr$(76)+chr$(185)+chr$(169)
NB  530 print#2,m$
FP  540 print#2,"m-e"chr$(0)chr$(3)
AD  550 input#2,x,x$,t,s:printx;x$;t;s:stop
```

The One Megabyte C64!

Activities for a rainy afternoon: C512

by Paul Bosacki

In Volume 9, Issue 2, Paul showed us how to expand a C64 to 256K internally and have GEOS recognize the extra RAM as a RAMdisk. At that time we stated that Paul was using a 1MB C64 - 512K internal and 512K in an REU. As you now know, this project generated a lot of interest amongst the readership and the Commodore community at large. The machine became the subject of various speculations and rumours.

Well, the time has come to lay those rumours to rest. The first half of the 1MB C64 was covered last issue when Paul showed how to expand the 1764 to 512K. This is the second part. This article will show how a C64 can be expanded internally to 512K. Et voilà, the 1MB C64.

As you might expect, this project is more complex than the two previous ones - in the software as well as the hardware. If you're not comfortable with a soldering iron in your hand you may want to have someone else do it. The usual disclaimers apply: you undertake this project at your own risk and goodbye the warranty.

On the software side of things, ***GEOS V2.0*** *has made significant changes in the way that the operating system handles drives. Consequently, it was necessary for Paul to modify some of Berkeley Softworks' own code to enable the banked RAM used in the C512. Accordingly:*

Special Note: Portions of *Driver1571.src* Copyright © 1986-1989 by Berkeley Softworks. All rights reserved. Used with permission. Our thanks to Berkeley Softworks for their kind indulgence in this regard and to Matt Loveless at Berkeley for his support and assistance.

* * *

When I claimed a few months ago that an Amiga needed a meg of memory to really show, I never imagined that anyone would want a 512K 64. Nor did I expect the overwhelming response the article generated. So first, before I get into anything, I want to thank all the people who took the time to write. Considering the vagaries of postal offices, you all should have long ago received my reply. Yes, I answered each and every letter and that's why, in part, this update is so late in getting out to you.

Also, I'd like to thank two people in particular: Richard Curcio and George Hug. Although this article might have appeared without their comments, suggestions and interest, writing it wouldn't have been as much fun.

Now, into the meat

As I pointed out in *Care and Feeding of the C256*, the limiting factor in an MPU's addressable memory space is its number of address lines. The C64's 6510 has sixteen, allowing access to 65,000 or so bytes. Adding two pseudo-address lines, as the last project demonstrated, bumped that to 256K. Four banks of 64K were made available through a simple POKE to $01. However, a small amount of memory had to remain 'common' to each bank. Specifically, memory below $0400 was always available. This was necessary because the stack and OS vectors must (within limits) remain constant. Change them without proper setup and the machine crashes.

The 512K project offers significant improvements over the previous design. In order to take our machines to 512K, it's necessary to add a third pseudo-address line. In the previous article, the two needed lines were found at the MPU I/O port. Needing three lines in this case, the MPU I/O port is no longer adequate. I/O is found elsewhere.

Unfortunately, the 64 uses its resources to the fullest, and this necessitates some additional work. But the extra work yields some nice returns. Unlike the 256K version of this project, all options can now be controlled through software. Options like 0K common memory to 16K and control over where the VIC chip finds its data. Well worth the extra effort!

The modification

An eight bit read/write latch is used in the 512K system to allow control over system memory configurations. This latch, called the Bank Control Register (BCR) for lack of a better name, is accessable at $dd80. The astute among us will realize that this space usually contains phantom CIA2 images. But that problem is worked around by remapping the CIA's 256 bytes into four unique and separately selectable 64-byte sections. The first 64 bytes still belong to CIA2 keeping its base address valid. However, the third 64-byte block belongs to the BCR. Read from or write to $dd80 and the system memory configuration will either be returned or set. The second and the fourth 64-byte sections are open for user expansion.

The BCR is the most significant improvement over the previous design. And the most powerful aspect. Through a little bit twiddling, the memory configuration can be changed at will. What follows is the bit function layout of the BCR:

bit0-2: The three pseudo-address lines needed to access the addtional memory. Bank 0 on power-up or hard reset.

bit 3: AEC enable. When this bit is set to 0 (default power status), the Video display matrix is drawn from Bank0.

The following three bits affect the amount of common memory (CRAM) available to the system.

```
bit 4: Mask A10 ($0400)
bit 5: Mask A11 ($0800)
bit 6: Mask A12 & A13 ($1000)
```

This takes a little explaining. When the MPU accesses a particular memory location, a combination of ones and zeros are placed on the address bus corresponding to the desired address. In the case of $03ff, A0 through A9 would have ones while A10 through A15 would be zeros. Decoding CRAM simply becomes a matter of monitoring A9 through A15. If they should equal zero, then CRAM is being accessed and Bank 0 is switched in. However, if any of those lines equal one then the bank selected is enabled.

> *Part of the design philosophy behind the 512K board was that switches were to be done away with altogether and that all options should be controlled through software...*

Each of the above bits masks the corresponding address line. Simply put, if the bit is set, then the address line cannot signal that a selected bank should enabled. CRAM is effectively widened. If all three bits are clear (the default power-on status), CRAM stretches to $3fff. However as each bit is set, CRAM space narrows:

option	bit4	bit5	bit6	CRAM
	a10	a11	a12,13	
i)	0	0	0	=$3fff
ii)	0	0	1	=$0fff
iii)	0	1	1	=$07ff
iv)	1	1	1	=$03ff

Four other CRAM combinations are possible, but some open CRAM 'holes'. For example, %101 has CRAM to $03ff. Then banked memory appears until $0800; there a CRAM hole opens that continues to $0fff. Then banked memory reappears. So, unless you know what you're getting into, stick to the four CRAM configurations above. They are the most useful.

Let's take a closer look. Clear all three bits and CRAM widens to include the bitmap at $2000. Not using the bitmap? Then how about a large (16K) area for machine language or BASIC programs that need to easily take advantage of an additional 384K of memory (that's (64-16)*8). The next option is similar to the first except that CRAM narrows to $0fff. This excludes the bitmap, and the work space is smaller. But some interesting possibilities open here. For example, rapid cycling through up to 64 different bitmaps becomes a reality. Imagine, the REU globe demo done totally from within system RAM!

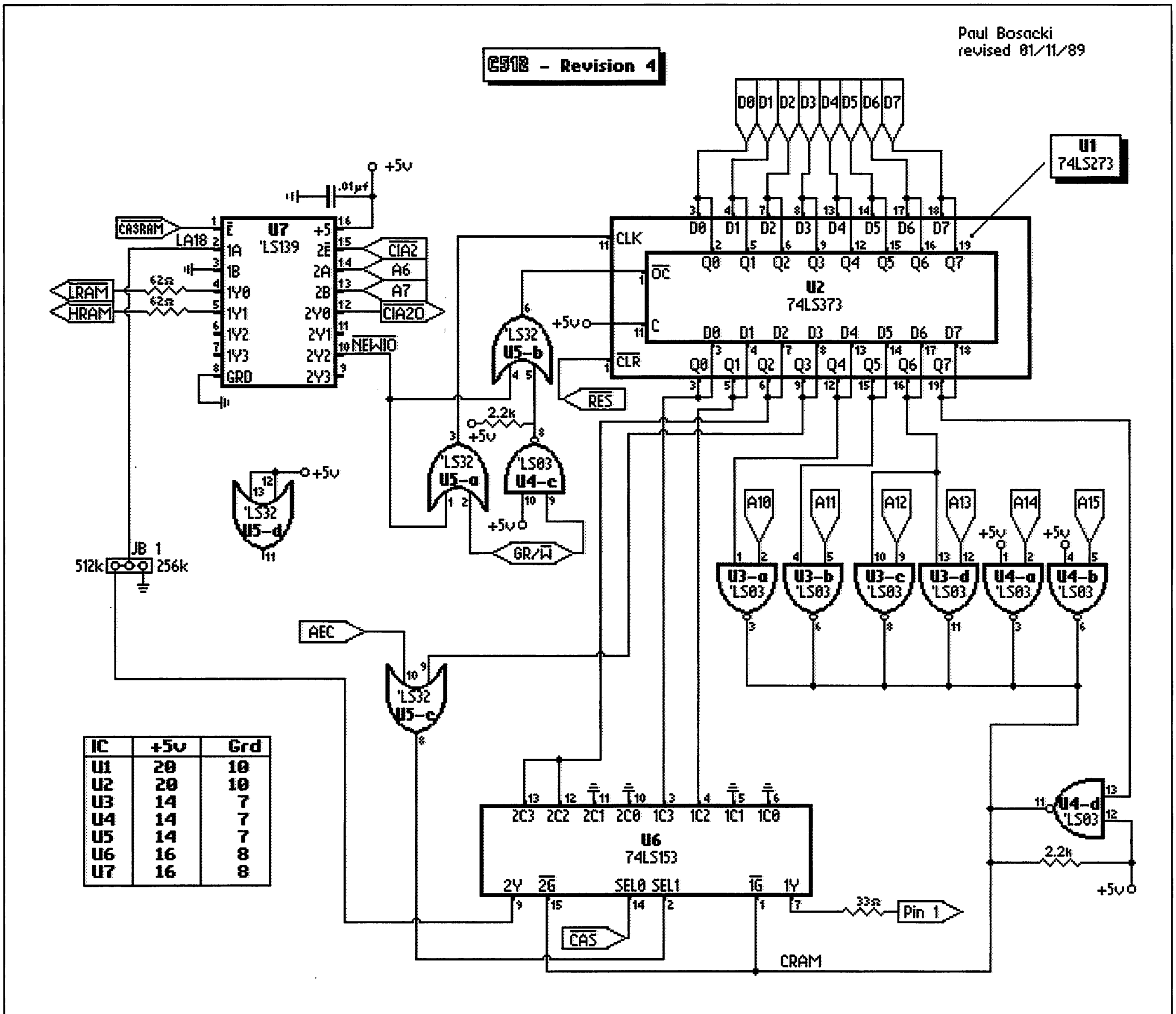
Option 3 narrows CRAM even further, leaving only the default screen matrix within CRAM. All banks could, therefore, draw their character screen matrix from the same place. And the final option banks out the screen matrix as well. All banks now share only OS vectors, the stack, and zpage; in short, anything below $0400 is drawn from Bank 0.

Then there's bit 7. By setting this bit, the CRAM option is disabled. In other words, there is no common memory. On a bank switch, the machine moves into a whole new domain; a place with it's own stack, OS vectors, zpage etc. This option is really exciting because it allows us a kind of task switching. With proper setup, a bank switch might drop us into a radically different machine. More on this later....

Arguably, the BCR is the most difficult aspect of this modification, both from the hardware and software side of things. When it comes time to build it, and later on, to program it, take your time looking over its specifications. It will save a lot of frustration later on.

Unlike the 256K version, the 512K modification has no switches. Two of the three switches have corresponding functions available through the BCR outlined above. The third switch was a 'master disable' switch necessary because some software and hardware is incompatible with the 256K modification. However, because the BCR is mapped into phantom I/O, and because every sane programmer reads from and writes to the BASE address of CIA2, the BCR should never be inadvertently accessed. Consequently, the mod board cannot be disabled; nor, really, should such action be necessary.

The 512K modification requires one step unnecessary to the 256K version: the installation of an additional 256K. Rather than building an additional board with all its attendant difficulties, it is easier to 'piggy back' one bank of DRAM on top of the other. Then all that needs to be done is bend up pin 15 of each chip on the top bank and solder the rest to the corresponding pin below. A somewhat simpler operation with little opportunity for mistakes.



IC	+5v	Grd
U1	20	10
U2	20	10
U3	14	7
U4	14	7
U5	14	7
U6	16	8
U7	16	8

Notes:

This board allows expansion through to 512k. The jumper block (JB1) is not necessary. If 512k is to be installed simply wire the 2Y output of the 'LS153 directly to the 'LS139. If 256k, pull the appropriate A input of the 'LS139 to ground.

On some c64's it may be necessary to replace the following chips as follows:
i) 74LS03's with 74F03's
ii) 74LS153 with 74F153
In banks where bit 1 is set, a "sparkling" effect may be visible in hires mode. The above chip changes solve this problem.

See text for function outline of 8 bit latch.

Circuit theory

The first 512K board was a patch on the orginal 256K board. With a little (read: a lot) of wiring and rewiring and an additional six ICs, it was possible to access even more memory. Twelve chips is a lot of chips. So the board was redesigned and the chip count reduced to seven. Good and bad. Bad because if you built the 256K mod, it's necessary to build and install another board. Good because the fewer chips, the less likely mistakes are, and the easier it is to troubleshoot any problems that may arise along the way. If you did build the 256K board, I offer this consolation: installing the DRAM is the hardest, touchiest part, and you don't have to do that again!

Part of the design philosophy behind the 512K board was that switches were to be done away with altogether and that all options should be controlled through software. When the wish list of options was drawn up, an 8-bit latch was pretty much demanded. The problem became where to map it into system memory. For better or for worse, I chose CIA2. Because CIA2 occupies 256 bytes of memory starting a $dd00, it was necessary to 'remap' I/O in that area.

This was acomplished through the use of an 'LS139 Dual 2-to-4 Line Decoder. The CIA2 select signal is intercepted and used to enable one half of the 'LS139. Address lines A6 and A7 serve to select which of the four 64-byte sections is accessed. If both A6 and A7 are low, then the 'LS139 'selects' CIA2 allowing it to continue its exsistence at $dd00. If however, A6 is low and A7 high (indicating address $dd80), then a low is generated on pin 10 of the 'LS139. This signal, with a little additional qualifying, selects the two components that make up the BCR: an 'LS273 Octal 'D Type' Flip-Flop is the write portion of the register, while an 'LS373 Octal 3-State 'D' Latch forms the read.

That signal, *NEWIO on the schematic, is then OR'd with GR/*W at one gate of an 'LS32, and OR'd with G*R/W at another. On a read operation (GR/*W is high, G*R/W is low)), the output buffer of 'LS373 is enabled and dumps to the data bus. On a write, the 'LS273 is clocked and the contents of the data bus are latched into the chip and immediately present at its outputs.

These outputs serve the variety of functions outlined in the BCR bit function map. Bits 0 and 1, the low address bits, are presented to two inputs of one half of an 'LS153 Dual 4-Line to 1-Line Data Selector/Multiplexor. Depending upon the state of CRAM (a signal whose generation we will discuss shortly), a 2-bit code is then strobed out to pin 1 of the 41256s.

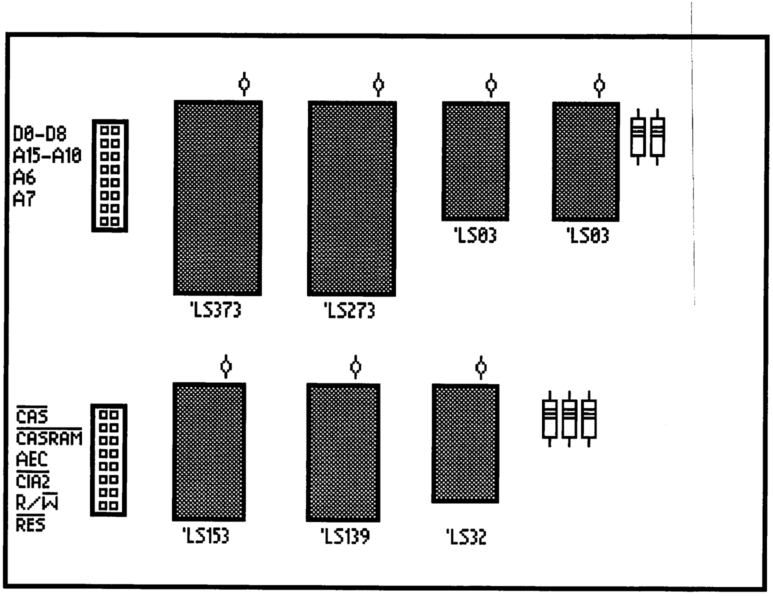
Ignoring bit 2 for the moment, bit 3 is used to qualify the AEC signal from the VIC chip. If bit 3 is high, the output of the 'LS32 OR gate will be high regardless of the state of AEC. If low, the state of AEC is present at the output of the OR gate. The output of the OR gate is the old *VID signal and drives one of the select pins on the 'LS153. The 'LS153 is wired in such a manner that should *VID go low, a pair of lows are strobed out to the 41256's. This occurs on the rise and fall of *CAS which is used to drive the other select pin on the 'LS153. The end result is that when bit 3 is high, AEC cannot force a CRAM call, and the video matrix is drawn from the current bank.

The next three bits serve to mask address lines and function much as bit 3 above. First, each of the address lines A10-A15 is presented to one input of an open-collector dual-input NAND gate (the two 'LS03's of the schematic). *[Each 'LS03 contains four such dual-input NAND gates. - PB]* In the case of A15 and A14, the other input is pulled high, the immediate result being that the inverted state of A14 and A15 is present at the output. The other address lines are handled in a different fashion. To the other input of the gates shared with A10 and A11, bits 4 and 5 respectively are presented. If either bit is high, the corresponding output shows the inverted state of that address line. If either bit is low, a high is generated. Consequently, neither of these lines can affect a CRAM call.

A12 and 13 share bit 6 and are affected as above. The output of the NAND gates are grouped and pulled up with a 2.2K resistor. Should any output go low (indicating that the corresponding address line is high), the grouped output is pulled low enabling the 'LS153 to pass a 2-bit bank select code to the 41256s. However, should all lines go high indicating a CRAM access, the grouped output goes high, forcing a default to Bank 0. This output is the old *CRAM signal, now active high and renamed CRAM on the schematic.

It is grouped with one other NAND output. At this gate, bit 7 and a high are decoded. Bit 7 is, as noted above, the CRAM disable function. If high, the grouped output is forced low, permanently enabling the 'LS153 until that bit is cleared.

All that's left now is to explain how the new address line is handled. The third address line does something wonderful. It



Parts Placement Layout
C512 - Revision 4

Parts List

1: 'HCT373
1: 'HCT273
1: 'HCT139
1: 'LS32
2: 'LS03
1: 'LS153

2: 20 pin sckts
2: 16 pin sckts
3: 14 pin sckts
2: 62Ω resistors
1: 33Ω resistors
2: 2.2kΩ resistors
7: 0.1 μF capacitors

Misc: solder ringed PC board, ribbon cable, connectors, 30 guage wirewrap, 22 guage wire.

allows us to select which 256K bank of DRAM is accessed. Bit 2 is presented to the other half of the 'LS153 dual 4-to-1 multiplexer. Depending again on the state of CRAM and *VID above, the other Y output of the 'LS153 generates either a high or low. A low is generated when either a CRAM call has been generated or when bit 2 is low. A high is found only when bit 2 is high. This signal, LA18 on the schematic, is passed to the other half of the 'LS139. The 'LS139 is enabled whenever *CASRAM goes low. *CASRAM is the same signal used by the VIC to actually select system DRAM over other system resources. Here we are using it to do the same thing; however, dependent on the state of bit 2, either *RAML or *RAMH will go low selecting one of two banks of DRAM. LA18 does not act as an address line in the truest sense of the word, but rather helps to generate a select for one of two banks of DRAM.

That's pretty much it. The key signal here is the CRAM signal. It plays the part of the master controller. When high, the 'LS153 is disabled and default bank 0 is switched in; when low, the contents of the three low bits in the BCR are free to set the bank accessed. *VID when low, has a similar effect, but this is acheived in a slightly different fashion.

Installation

Before you attempt installation, I suggest you read and reread the section above. It's intended to familiarize you with the function of the board and how the various components relate. Knowing how the board works can only help you later on if there are any problems.

Now, if you've already installed the 41256s, all you're really concerned with is the construction of the new mod board. If you have yet to install the DRAM, let's go over it briefly (for a more complete description, see *Transactor*, Volume 9, Issue 2). First, disassemble your computer and locate the eight 4164 DRAMs on the system board. They're located in the lower left hand corner of the system board. If you can't find them, don't worry. Just keep reading. There's interesting news ahead.

Once you've located the chips, turn the board over and carefully note their position. Now, using a combination of desoldering braid and a vacuum desolder, remove them. Another option is to cut the pins away from the chip, heat the pin and remove it with a small pair of pliers (Richard Curcio's *RAMifications* from Volume 9, Issue 4 offers some valuable advice here). Make certain that each of the holes is as free of solder as possible.

With all chips removed, install 16-pin sockets in their places. Once installed, use a fine guage wire to link pin 1 of each socket to the next. Connect the final one to a convenient ground (pin 16 of that socket, for example). Now, install the 41256-15s. Although I've never had a problem here, these chips are static-sensitive, so be certain to ground yourself first. Mistakes with DRAM are expensive and, at this stage, difficult to uncover. Now reassemble as much of your computer as necessary to power up safely. But before you do, check the orientation of each DRAM. An upside down DRAM equals a dead DRAM.

If everything checks out, connect your power supply and your monitor and turn your machine on. Most likely, you'll see the familiar power-up screen. Generally, the only other possiblities are a blank screen or one that changes randomly then 'freezes'. If you're confronted with either, don't panic. Turn your machine off, disconnect everything and examine your work. Check your soldering for bridges, try reseating the chips. Are they installed properly? Did a pin get bent beneath a chip when you installed it earlier? Check your pin 1 work. Is it properly grounded; is each socket in the chain linked? If you have a logic probe, power your machine back up and test each of the pins. Pins 1 and 16 should show low, while pin 8 shows high. All others should pulse between high and low. If a pin does not reflect the proper state, there is probably a problem with the soldering at that point. Resolder that pin of the socket and any other that might show a problem. Check everything and try again. And don't worry: You probably won't have to go through any of this.

With the chips installed, move on to constructing the board. Once again, I used point-to-point soldering and all my suggestions from the previous article still apply. Check the parts layout diagram for the layout I used. Something I did this time round, was use 16-pin connectors for all interfacing. The parts layout shows the male connectors to the left hand side of the board. The result was a board that could be easily removed if troubleshooting indicated a problem. And there *were* problems - an incorrectly wired 'LS273 for one! As Richard Curcio once told me, half jokingly: "If it works right the first time, don't trust it!"

With the board finished, it's time to interface (I've always wanted to say that). There are two distinct places to go for the various signals required by the board: either the cartridge port or the MPU. I suggest the MPU only if it is socketed, and then I suggest you carefully remove it from its socket while doing this. Take a length of ribbon cable 16-conductor wide and make the following connections: A15-A10, A7, A6, D0-D7. Do this by heating the pin, gently pushing it to one side with the tip of your soldering iron and carefully inserting the conductor.

Whether you use the cartridge port or the MPU socket, follow the appropriate diagram. *[See page 50 - MO]* Both diagrams show the layout from the solder side. Determining the final length of the ribbon cable is up to you but keep it short. Now, if you used connectors, attach the other end to the connector. Otherwise, solder A15-A10 to the appropriate inputs of the 2 'LS03 sockets, A6 and A7 go to pins 14 and 13 of the 'LS139 respectively. The data bus is tricky, and if you're going to make any mistakes it's here. Follow the schematic carefully and make the appropriate connections.

With that finished (easier said than done), take another length of ribbon cable, this time six-conductor wide. Now we hunt.

The first signal we want to locate is *CAS. Locate pin 1 of either 'LS257 (U13 or U25) on the system board (they're to the right of the DRAM you removed earlier. Follow the trace away from the pin until you reach a tiny silver dot. This is a pass-through to the other side of the board. Heat the dot and install the first of six conductors. Remember this procedure because we're about to repeat it.

Look to the DRAMs, and locate pin 15 with a trace moving away from it (only one DRAM will show this). You've just found the *CASRAM signal generated by the PLA. Again follow the trace away from the chip until it comes to a resistor. On *my* board, this resistor was labelled R42. It may not be on *your* board, so follow the trace instead. Remove the resistor. Into the opposite solder pad, install the second conductor.

The next two signals are easier to locate. The first, AEC is available at a number of places. The first is pin 16 of the VIC chip (U19), the second is the MPU, pin 5. Both offer pass-through jumpers which can be found by following the trace. Or, if you wish, heat the pin and insert the third conductor. The next signal, *CIA2, offers us a special case. Again, I'll offer you two choices. Locate the 'LS139 on the system board (U15). Pin 11 is the *CIA2 select line. Cut the trace leading away from the pin, scrape away the green insulating material and carefully solder the fourth conductor here. Or, if CIA2 (U2) is in a socket, bend up pin 23 of CIA2. This is the chip select pin. Now, heat the socket's pin and insert the conductor. I did neither. I removed the 'LS139 and installed a socket. When I reinstalled the 'LS139, I bent up pin 11 and soldered the conductor to pin 11 of the socket. Any of the above will work; I leave the method to you.

If you choose to install the second bank of RAM, you will need a stronger power supply than the one that came originally with your C64! ...

Next locate pin 8 of the 2114 Colour RAM (U6). The fifth conductor attaches there. And lastly, pin 40 of the MPU allows us easy access to the *RES signal. Solder the last conductor to this pin.

Now on the mod board, make the following connections:

```
*CAS     to pin 14 of the 'LS153 socket
*CASRAM  to pin 1  of the 'LS139 socket
AEC      to pin 10 of the 'LS32  socket
*CIA2    to pin 15 of the 'LS139 socket
GR/W     to pin 2  of the 'LS32  socket and
         to pin 9  of the 'LS03  socket (U4)
*RES     to pin 1  of the 'LS273 socket
```

Again, if you used connectors, you've had it somewhat easier. Just install the connector to the other end of the cable.

The above are the required signals from the system board. There are four signals that go the other way. Turning our attention again to the system board, go back to the resistor we removed earlier. Into the other hole install one conductor. This will become the new *CASRAM signal, labelled *RAML on the schematic.

Depending on how you dealt with the *CIA2 signal earlier, another conductor connects to the other side of the cut trace, or to either the bent up pin on the CIA or 'LS139. That's the worst of it. Two more lines go out to the system board, but we'll save them for a bit.

Now, you can fix the board into place and install the chips ensuring correct orientation and placement. Connect the +5V source and ground to the mod board. Both are availabe at the cassette port. Again, reassemble as much of your computer as necessary and power up. With any luck, you're staring at the power-up screen! If not, let's go over the possiblities. Check the interface wiring. Are we getting the right signals up to the mod board, is there any sloppy soldering? Using a voltmeter or a logic probe check for the following conditions: pulses on all data and address lines. Fixed highs indicate a crashed bus and the problem is probably in the interface wiring. Pulses on *CAS, *CASRAM, *RAML. Fixed highs or lows are a problem. Check all associated wiring. The same holds true for AEC and GR/W. CIA2 will show high; otherwise, there's a problem either in the interface wiring or the mod board itself.

One trick that might help you locate a problem is turning off your computer, pulling a chip from the mod board and powering back up. If you get a power-on message, then you just narrowed your field of search. The only chips that you can't do this with are the 'LS139 and 'LS153. Pull either of these chips and you will not get the power-on message.

However, if you've been careful and meticulous with this, you were confronted with the power-on message. Great! Now, turn off your computer and install the second bank of RAM. However, if you choose to do so, *you will need a stronger power supply than the one that came originally with your C64!* A 128 power supply or the one that comes with the 1764 will do just fine. But you *will* need a stronger power supply. Things might work fine for a while, but you're courting disaster.

To install the second bank of DRAM, carefully bend up pin 15 of each chip. Then piggy back the second bank atop the first and solder the upper pin to the lower. Again using a fine guage wire, link pin 15 of each chip on the upper bank and run the conductor out to pin 5 of the 'LS139. Now, disconnect pin 1 of the 41256s from the convenient ground and run it out to pin 7 of the 'LS153. Connect the keyboard and again power your machine back up. The power-on message should greet you. Now type this:

```
poke 56704, 124 <cr>
```

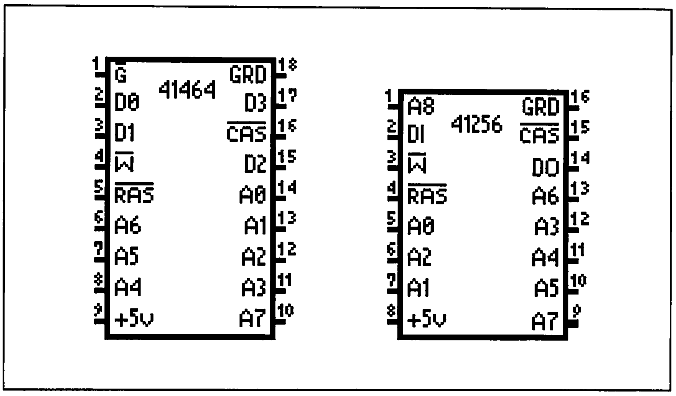
The screen should fill with garbage. If any of the garbage characters are randomly changing, then one or more of the DRAMs on the top bank is not connected properly. Locate the problem and try again until the problem is solved.

If you choose not to install the additional bank of memory at this time, that's fine, just connect the keyboard and power-up as above. Now, instead type:

```
poke56704,121
```

As above, your screen should fill with garbage. You won't have the random charcters problem though. If nothing happened, however, there is a problem. Something is wrong in the new I/O decoding, or the CRAM generation ciruitry. Try PEEKing the above location. If you get 121 or 127 for the 512K machines, then your problem's with CRAM generation. If you get a 0, then check out the write half of the BCR. A value that changes with each PEEK, indicates a problem with both halves of the BCR. Check the wiring carefully and try again.

With that done, it's over. Your machine now contains 512K of user installed banked memory. Give yourself a pat on the back. What you've done is just short of amazing. Congratulations! Now, reassemble your computer. Fair warning, if you installed the extra bank of memory, the top RF shield will no longer fit. Don't bother with it. It's not a problem.



Modifyng the 'E Board'

If you opened your computer and had trouble finding the the eight DRAM chips, there might be a good reason for this: you're the proud of owner of Commodore's latest line of revisions for the C64. The 'E board' has two 41464 DRAM's (also labelled LH2464) rather than eight 4164's. As well, the board layout itself is significantly different from earlier boards. There are two different E revisions that I'm aware of. The first maintains the old layout and logic except for the 41464s. The second one is radically different, with one large chip handling all the select logic and timing. Gone are the PLA and the 'LS139 decoder and 'LS257 multiplexers. But it is not difficult and certainly not impossible to modify either of these revisions to 512K.

All that is mentioned in the above section still applies with the following changes:

i) *CAS is available at pin 19 of the VIC chip.

ii) *CASRAM: Use the same technique as above, follow the trace back from the DRAM, but instead of pin 15, follow pin 16 to the resistor.

iii) *CIA2: use the cut trace method from above. Cut the trace off pin 23. The trace opposite the CIA goes to the 'LS139.

iv) Some chip designations have changed. The 6510 is now an 8500 and in generally socketed. The SID and VIC both have new 85xx prefixes. The VIC is the larger of the two. The pin layouts have not changed, however, just the designations.

All the rest of the signals are available as indicated in the above section and don't present a problem.

Because the DRAM used in the E board revisons are such different beasts from the 4164, the mod board, as presented, is incompatible. This leaves us with two options. The first of course, is to modify the board so that it will work. But attendant with this strategy is laying in 14 additional 41464 DRAMs. The second strategy involves forsaking the 41464's altogether, and installing 41256's. Using this strategy, the mod board does not have to redesigned, although 16 41256's must be 'laid in'.

Of the two strategies, the second makes the most sense. The 41464 DRAM was initially designed for systems that would be using less than 256K of RAM. In that case, their use becomes more economical. When 256K is reached, however, it makes more sense to go with the 41256, and that's what we'll do here.

The method is actually fairly straightfoward, but it requires a lot of additional work on your part. First, remove the 41464s using the techniques outlined earlier and install 18-pin sockets in their place. Now, install eight 16-pin sockets on a small board. Take a look at the attendant pin layout for the 41256-15. For each of the eight sockets, link together each address line to the next in the chain. Do the same for *CAS, *RAS, *W, pin 8 and pin 16. 41256s have two data lines, a data in (DI) and data out (DO). For each socket, link these two pins together (pins 2 and 14).

Now, direct your attention to the pin layout for the 41464. Ignoring pin 1 (*G) and pin 18 (GRD) altogether, each pin corresponds to another on the 41256s. The data pins offer a special case. Each 41464 has four data pins. With two chips, that gives us eight data lines. Each data line goes to one 41256.

The new DRAM board is interfaced to the system using two 16-pin dual-in-line connectors plugged into the sockets installed on the system board earlier. Plug the connector into the lower 16 holes. You can find this item at Radio Shack. It's made up of two connectors joined by an 18-inch stretch of ribbon cable. Cut the cable in half, each piece consisting of a connector and nine inches of cable. At the other end, solder the corresponding line from the 41464 sockets to the 41256s. Do this with one of the pieces. With the other, it is only necessary to connect the data lines. Now make a suitable ground connection between the system board and the DRAM board.

With the signals out to the board, install eight 41256s, and ground pin 1 of the 41256s again paying attention to the suggestions above. Power up your machine and the usual power-on message should appear. If it doesn't, check your wiring, and chip orientation. Check with logic probe or voltmeter that each pin shows a pulse condition (except for pin 8, high, and pin 16, low). If you find a pin that isn't offering the proper condition, check the pin out diagram and that'll give you an idea as to the nature of the problem. Follow the above instructions if you intend to install a second bank of DRAM.

Once you've have the DRAM installed and operating properly, proceed as above. Pay special attention to those areas where signal locations differ. There are two additional notes in this area. The first is that output Y1 from the 'LS153 goes to pin 1 of the 41256s on the new RAM board. The second: *RAML and *RAMH go to pin 15 of bank 1 and pin 15 of bank 2 respectively. And that, as they say, ladies and gentlemen, is that.

Getting aquainted

Saying that your C64 is a radically different beast would be a gross understatement. The Bank Control Register up at $dd80 gives you access to total control over how your system's memory is configured. Study the above tables, and the resultant configurations. Some of the results might be suprising - even disconcerting - if you don't understand the implications of a particular configuration. And, although there are 256 possible configurations, not every one of them useful. However, none of them are tragic.

To make certain our RAMdisk acted like a RAMdisk, Berkeley's RAMdisk driver was unassembled and then rebuilt...

But to get you started, here are some values to try and the resultant configuration. Just POKE the value to $dd80 (56704), when you press Return the new configuration will be in effect.

i) **0-7:** These are the base values. Each corresponds directly to one of the eight banks of memory. In each case memory above $3fff is replaced with the select bank.

ii) **120-127:** These values are interesting in that CRAM has been set to its lowest amount. As well, screen data is drawn from the current bank. This allows eight separate work spaces for BASIC or machine language programs. However since all OS vectors, the stack and zpage are shared, be careful of programs that modify those areas - especially the interrupt vector. On a bank switch, your machine is sure to crash.

iii) **248-255:**These values disable the CRAM option. Poking one of these values without proper set up will cause a crash.

Before poking one of these values, it is necessary to set up the bank's zpage, stack and OS vectors. Key in the program *task switch* to see this option in operation. Call it with **poke2,*b*:sys828** where *b* is the bank you want initialized.

I encourage you to become familiar with the operation of the BCR. POKE away! The results you get may be strange (even confusing) but, I assure you, yours is a 'few of a kind' machine. Have fun!

Of RAMdisks and GEOS

One of the nice things about things about the 256K project was that it allowed the additional memory to be configured as a RAMdisk under GEOS. For those of us lacking REU's, that do-it-yourself RAM expansion project offered an alternative route to a souped-up GEOS. As so many sources have stated, an REU is an absolute necessity for the serious GEOS user. In fact, REU routines are now an integral part of that operating system. Where it was once possible to fool GEOS without consquence that an REU was present, it now seems (at least at this point) impossible. What exactly does this mean? Well, first the good news: The program that follows allows the configuration of a 256K or 512K RAMdisk under GEOS 2.0 and the co-existence of an REU. Now the bad news: you must have an REU (either a 1764 or 1750) for it to work.

Some background. When you boot GEOS, it searches for any programs of the auto-exec file type. *Configure* is one of these. *Configure*, as you know, searches out, and initializes the drives on your system. If you have an REU, the disk drivers are stashed in the REU. When you switch drives, the driver is fetched down from the REU and the other drive is accessed. If you don't have a REU, there are two possibilities. If your drives are the same type (ie., two 1541s), you don't have a problem. If they are different, you must have a copy of *Configure* on the work drive. The reason for this being that the DeskTop will load the appropriate driver and get things going.

This seemed to be a glimmer of hope. It seemed possible to splice a new driver into *Configure*. The idea was this: your system consists of a 1541 and 1571. Great, we'll overwrite the 1581 driver with the new one and then configure a 1581 as part of the system. Now, when we open the RAMdisk, the DeskTop will think it's a 1581, fetch our driver and we'll be in business. No way! Unfortunately, the DeskTop is fairly picky about drivers and the number of tracks and sectors a disk should have. As well, it doesn't always use the standard routines for fetching its information about a particular disk or driver. The DeskTop is smart! It knows a 1581 has its directory at Track 40, Sector 0 and if it isn't found there, the DeskTop tells you this with an error. The bottom line was: the scheme described above didn't work.

But with an REU on the system, we've a different situation altogether. All we need do is make certain that the right driver is in the REU and that our RAMdisk conforms to the expected format. In other words, all that was necessary was to make certain our RAMdisk acted like a RAMdisk. In order to accomplish this, Berkeley's RAMdisk driver was unassembled and then rebuilt to support the banked RAM.

The program that handles installation of the BRAMdisk is *C512Install*. Like *Configure*, it is an auto-exec program. Because auto-exec programs are executed in the order they appear on your boot disk, *C512Install* must be placed after *Configure*. What happens is this: after *Configure* installs the drives on your system, control is returned to what I call the 'BootTop'. It then searches for other auto-execs. Finding *C512Install*, the BootTop executes it. *C512Install* does very little error checking. It simply searches out an empty drive slot and, on finding one, installs itself (uploading the BRAMdriver to the REU). Depending on the amount of expansion DRAM, either a BRAM1571 or '41 is installed. It then exits.

Now you're asking: what use is it. After all, you already have an REU. Imagine this: two shadowed 1541s (if you've a 512K REU and two 1541s) and a 1571 BRAMdisk. Now that's a fast, powerful configuration.

And if you don't have an REU? Well, I'm still working on the problem and you've got a little incentive too. Maybe between the few of us, we can figure it out.

How to get there from here

I received many letters after *Care and Feeding of the C256* appeared in *Transactor*, Volume 9, Issue 2. In letter after letter I had the unique pleasure of reading how people had pushed their C64s into domains that would have been impossible for us to imagine just a few years ago.

Throughout writing this article, a phrase from an old movie has been running through my head. I've been looking for a way to work it in. When I think again of what people are doing to and with their 64s, the phrase becomes suddenly appropriate: Something wonderful is about to happen!

Enjoy it! And thanks!

Listing 1: *C512Inst.hdr*

```
;**********************************
;*          C512Inst.hdr          *
;*                                *
;*                                *
;* This is the header declaration *
;* for C512Install.               *
;**********************************

.header
.byte  3
.byte  21
.byte  $80|USR
.byte  AUTO_EXEC
.byte  VLIR

.word  $0400
.word  $03ff
.word  $0400

.byte  "C512Install V1.1",0,0,0,0
.byte  "Paul J. Bosacki     ",0

.block 43
.byte  "Installs banked RAM as RAMdisk.",0

.endh
```

Listing 2: *C512Install.src*

```
;*********************************
;     InstallRAM
;*********************************

.if Pass1

   .include geosSym
   .include geosMac

.endif

BCR          ==$dd80
DoBankRAMOp  ==$02a7

.psect
      lda    version          ;if not V2.0 then exit
      cmp    #$20
      bne    Quit

      lda    ramExpSize       ;if REU not present then exit
      beq    Quit

      LoadW r10, C512Install  ;get diskdriver
      lda    #1               ;second VLIR record
      sta    whichDriver
      jsr    GetDriver
      txa                     ;exit on error
      bne    Quit

      ldy    #8
2$:   lda    $8486,y
      beq    1$               ;branch on empty drive slot or
      bmi    1$               ;REU drive
      iny                     ;otherwise, continue search.
      cpy    #10
      bcc    2$

1$:   sty    driveNum         ;save drive number
      jsr    InstallDriver    ;and install driver
      txa
      bne    Quit

      jsr    StashDriver

Quit:lda     C_curDrive       ;restore configuration
      jsr    SetDevice
      jmp    EnterDeskTop     ;and return control to BootTop
GetDriver:
      LoadW r6,  fileNmBuf    ;locate filetype auto-exec
      LoadB r7L, $0e          ;with permanent filename "C512Install"
      LoadB r7H, 1
      jsr    FindFTypes
      txa
      bne    1$
```

```
        LoadW r0, fileNmBuf     ;open that file
        jsr   OpenRecordFile

        lda   whichDriver       ;open second record
        jsr   PointRecord

        LoadW r7, diskBuf       ;and load that record into
        LoadB r2L, $ff          ;diskBuf
        sta   r2H
        jsr   ReadRecord
        txa
        pha
        jsr   CloseRecordFile   ;tidy up
        pla

1$:     rts
StashDriver:
        LoadW r0,  $848e        ;update drive vars. in REU
        LoadW r1,  $7900+$048e  ;so system can be RBOOTed
        LoadW r2,  4
        LoadB r3L, 0
        jsr   StashRAM

        ldy   driveNum          ;stash new disk driver to REU
        LoadW r0, diskBuf
        lda   REUHstash-8, y
        sta   r1H
        lda   REULstash-8,y
        sta   r1L
        LoadW r2, $0d80
        LoadB r3L, 0
        jsr   StashRAM
        rts
InstallDrive:
        php
        sei
        jsr   MoveTransfer
        jsr   FindRamSize
        txa
        bne   1$

        jsr   FormatDrive

1$:     plp
        rts

MoveTransfer:
        LoadW r0, moveRoutine   ;move transfer routine to CRAM and
        LoadW r1, DoBankRAMOp   ;GEOS free ram
        LoadW r2, routineSize   ;because of its location, this is the
        jsr   MoveData          ;only free ram under GEOS
        rts

FindRamSize:
        MoveB CPU_DATA, tempA
        LoadW CPU_DATA, IO_IN   ;map in IO

        ldx   DEV_NOT_FOUND
        lda   #0
        sta   size
        lda   BCR               ;if not bank 0, then
        bne   1$                ;error-don't install!

        LoadW r0,  $c006        ;source
        LoadW r1,  buf          ;destination
        LoadW r2,  $0009        ;tlength
        LoadB r3L, #%01100100   ;bank 4
        LoadB r3H, 0            ;bank 0

        jsr   DoBankRAMOp       ;do BankOp
```

```
        ldy   #r1
        ldx   #r0
        lda   #9
        jsr   CmpFString
        bne   2$
        lda   #$40
        sta   size
        bne   3$

2$:     lda   #$80
        sta   size

3$:     ldx   #0                ;return no error
1$:     MoveB tempA, CPU_DATA
        rts

FormatDrive:
        MoveB CPU_DATA, tempA
        LoadB CPU_DATA, #$35

        ldy   driveNum
        lda   #$81              ;drive type=ram1541
        sta   $8486,y
        lda   #0
        sta   $88b7,y
        lda   size
        bvs   1$

        lda   #$80              ;update drive vars. to indicate
        sta   Header+3          ;a ram5171
        sta   $88b7,y
        ora   #2
        sta   $8486,y
        lda   #$37              ;fix header title to reflect change
        sta   driveModel

        LoadW r0,  #head1571    ;dump track $35, sector $00 header
        LoadW r1,  $8800
        LoadW r2,  #$0100
        LoadB r3L, 0
        LoadB r3H, #%01100101   ;bank 5
        jsr   DoBankRAMOp

        LoadW r0, fix1571
        LoadW r1, Header+$dd
        LoadW r2, 35
        jsr   MoveData

1$:     LoadW r0,  #Header      ;dump track $18, sector $00 header
        LoadW r1,  $8800
        LoadW r2,  #$0100
        LoadB r3L, 0
        LoadB r3H, #%01100010   ;bank 2
        jsr   DoBankRAMOp
        ldy   #0
        tya

10$:    sta   diskBlkBuf,y
        dey
        bne   10$
        lda   #$ff
        sta   diskBlkBuf+1
        LoadW r0,  #diskBlkBuf  ;dump offside dir track to $19,$08
        LoadW r1,  $9c00+$0800
        LoadW r2,  #$0100
        LoadB r3L, 0
        LoadB r3H, #%01100010   ;bank 2
        jsr DoBankRAMOp

        LoadW r0,  #diskBlkBuf  ;dump $18,$01 to expansion ram
        LoadW r1,  $8900
        LoadW r2,  #$0100
        LoadB r3L, 0
        LoadB r3H, #%01100010   ;bank 2
        jsr   DoBankRAMOp
```

```
     MoveB tempA, CPU_DATA

     lda   #0
     ldx   #0
     rts
moveRoutine:
     PushB CPU_DATA
     LoadB CPU_DATA, IO_IN
     ldy   r2L
     ldx   r3H
1$:  dey
     lda   r3L
     sta   BCR
     lda   (r0L),y
     stx   BCR
     sta   (r1L),y
     tya
     bne   1$

     beq   Done

     PushB CPU_DATA
     ldy   r2L
2$:  dey
     ldx   #IO_IN
     stx   CPU_DATA
     lda   r3L
     sta   BCR
     lda   #$30
     sta   CPU_DATA
     lda   (r0L),y

     stx   CPU_DATA
     ldx   r3H
     stx   BCR
     beq   3$
     ldx   #$30
     stx   CPU_DATA
3$:  sta   (r1L),y
     tya
     bne   2$
     ldx   #$35
     stx   CPU_DATA

Done:sta   BCR
     PopB  CPU_DATA
     rts

IRQVEC:
     pla
     tay
     pla
     tax
     pla
NMIVEC:
     rti

e_moveRoutine:

routineSize  = e_moveRoutine-moveRoutine

Header:
.byte $12, $01, $41, $00, $15, $ff, $ff, $1f
.byte $15, $ff, $ff, $1f, $15, $ff, $ff, $1f
.byte $15, $ff, $ff, $1f, $15, $ff, $ff, $1f
.byte $15, $ff, $ff, $1f, $15, $ff, $ff, $1f
.byte $15, $ff, $ff, $1f, $15, $ff, $ff, $1f
.byte $15, $ff, $ff, $1f, $15, $ff, $ff, $1f
.byte $15, $ff, $ff, $1f, $15, $ff, $ff, $1f
.byte $15, $ff, $ff, $1f, $15, $ff, $ff, $1f
.byte $15, $ff, $ff, $1f, $15, $ff, $ff, $1f
.byte $11, $fc, $ff, $07, $12, $ff, $fe, $07
.byte $13, $ff, $ff, $07, $13, $ff, $ff, $07
.byte $13, $ff, $ff, $07, $13, $ff, $ff, $07
.byte $13, $ff, $ff, $07, $12, $ff, $ff, $03
.byte $12, $ff, $ff, $03, $12, $ff, $ff, $03
.byte $12, $ff, $ff, $03, $12, $ff, $ff, $03
.byte $12, $ff, $ff, $03, $11, $ff, $ff, $01
.byte $11, $ff, $ff, $01, $11, $ff, $ff, $01
.byte $11, $ff, $ff, $01, $11, $ff, $ff, $01

headerTitle: .byte "BRam 15"
driveModel:  .byte "41"
             .byte 160,160,160,160,160
             .byte 160,160,160,160
             .byte "PJ",160,"2A"
             .byte 160,160,160,160
             .byte 19,8
             .byte "GEOS format V1.0"

             .block 256-188

fix1571:
  .byte $15, $15, $15, $15, $15, $15, $15, $15, $15
  .byte $15, $15, $15, $15, $15, $15, $15, $15, $00
  .byte $13, $13, $13, $13, $13, $13
  .byte $12, $12, $12, $12, $12, $12
  .byte $11, $11, $11, $11, $11
head1571:
  .byte $ff, $ff, $1f, $ff, $ff, $1f, $ff, $ff, $1f, $ff, $ff, $1f
  .byte $ff, $ff, $1f, $ff, $ff, $1f, $ff, $ff, $1f, $ff, $ff, $1f
  .byte $ff, $ff, $1f, $ff, $ff, $1f, $ff, $ff, $1f, $ff, $ff, $1f
  .byte $ff, $ff, $1f, $ff, $ff, $1f, $ff, $ff, $1f, $ff, $ff, $1f
  .byte $ff, $ff, $1f, $00, $00, $00, $ff, $ff, $07, $ff, $ff, $07
  .byte $ff, $ff, $07, $ff, $ff, $07, $ff, $ff, $07, $ff, $ff, $07
  .byte $ff, $ff, $03, $ff, $ff, $03, $ff, $ff, $03, $ff, $ff, $03
  .byte $ff, $ff, $03, $ff, $ff, $03, $ff, $ff, $01, $ff, $ff, $01
  .byte $ff, $ff, $01, $ff, $ff, $01, $ff, $ff, $01

.block 152

;internal variable space trails code

C512Install:  .byte  "C512Install",NULL
fileNmBuf:    .block 16
REUHstash:    .byte  $83,$90,$9e
REULstash:    .byte  $00,$80,$00
driveNum:     .byte  9
whichDriver:  .block 1
size:         .byte  0
buf:          .block 9
tempA:        .block 1
C_curDrive:   .block 1

spacer:       .block 8

diskBuf:      .block 1
```

Listing 3: *C512Install.lnk*

```
;***************************************
;*          C512Install.lnk           *
;*                                     *
;*                                     *
;* These are the link file directives *
;* for C512Install & driver1571.      *
;***************************************

.output  C512Install
.header  C512Inst.hdr.rel

.vlir
.psect   $0400
         C512Install.rel

.mod 1

.psect   $9000
         driver1571.rel
```

Listing 4: *Driver1571.src*

```
;**********************************
;*      RamDisk Driver (geos)      *
;**********************************

.if Pass1
   .noeqin
   .noglbl
      .include geosSym
      .include geosMac
   .eqin
   .glbl
.endif

BCR      =$dd80     ;this is the Bank Control Reg.

; Four banks of 64K are available. The register is laid out like this:

;     bit 0: bank select
;     bit 1: bank select
;     bit 2: bank select
;     bit 3: video access forced bank0=0
;     bit 4: consider a10=1
;     bit 5: consider a11=1
;     bit 6: consider a12 & 13=1
;     bit 7: CRAM Inhibit

; Portions of the following code are Copyright (C) 1986-1989 by
; Berkeley Softworks. All rights reserved. Used with permission.
; Special thanks to Matt Loveless at Berkeley for his support.

.psect

OSJumpTable:
   .byte [C_InitForIO,        ]C_InitForIO
   .byte [C_DoneWithIO,       ]C_DoneWithIO
   .byte [C_ExitTurbo,        ]C_ExitTurbo
   .byte [C_ExitTurbo,        ]C_ExitTurbo
   .byte [C_EnterTurbo,       ]C_EnterTurbo
   .byte [C_ChangeDskDev,     ]C_ChangeDskDev
   .byte [C_NewDisk,          ]C_NewDisk
   .byte [C_ReadBlock,        ]C_ReadBlock
   .byte [C_WriteBlock,       ]C_WriteBlock
   .byte [C_VerWriteBlock,    ]C_VerWriteBlock
   .byte [C_OpenDisk,         ]C_OpenDisk
   .byte [C_GetBlock,         ]C_GetBlock
   .byte [C_PutBlock,         ]C_PutBlock
   .byte [C_GetDirHead,       ]C_GetDirHead
   .byte [C_PutDirHead,       ]C_PutDirHead
   .byte [C_GetFreeDirBlock,  ]C_GetFreeDirBlock
   .byte [C_CalcBlocksFree,   ]C_CalcBlocksFree
   .byte [C_FreeBlock,        ]C_FreeBlock
   .byte [C_SetNextFree,      ]C_SetNextFree
   .byte [C_FindBAMBit,       ]C_FindBAMBit
   .byte [C_NxtBlkAlloc,      ]C_NxtBlkAlloc
   .byte [C_BlkAlloc,         ]C_BlkAlloc
   .byte [C_ChkDkGEOS,        ]C_ChkDkGEOS
   .byte [C_SetGEOSdisk,      ]C_SetGEOSDisk

               jmp C_Get1stDirEntry
               jmp C_GetNxtDirEntry

GetOffPgTS     jmp C_GetOffPgTS
SetLink:       jmp C_SetLink
DskBufRdBlk:   jmp C_RdBlkDskBuf
DskBufWrBlk:   jmp C_WrBlkDskBuf
               nop
               nop
               rts
               nop
               nop
               rts

               jmp   C_AllocateBlock
               jmp   C_ReadLink
MoveTransfer:  jmp   C_MoveTransfer
diskType:      .byte $82, "V1.0",NULL
```

```
C_GetDirHead:  jsr   Dir1Get
               jsr   C_GetBlock
               txa
               bne   1$
               ldy   curDrive
               lda   $8203
               sta   $88b7,y
               bpl   1$
               jsr   Dir2Get
               jsr   C_GetBlock
               lda   #$06
               sta   interleave
               rts

1$:            lda   #$08
               sta   interleave
               rts

C_RdBlkDskBuf: LoadW r4,diskBlkBuf

C_GetBlock:    jsr   EnterTurbo
               txa
               bne   1$
               php
               sei
               jsr   ReadBlock
               plp
1$:            rts

C_PutDirHead:  php
               sei
               jsr   Dir1Get
               jsr   WriteBlock
               txa
               bne   1$
               ldy   curDrive
               lda   curDirHead+3
               sta   $88b7,y
               bpl   1$
               jsr   Dir2Get
               jsr   WriteBlock
1$:            plp
               rts

C_WrBlkDskBuf: LoadW r4, diskBlkBuf

C_PutBlock:    jsr   EnterTurbo
               txa
               bne   1$
               php
               sei
               jsr   WriteBlock
               txa
               bne   2$
               jsr   VerWriteBlock
2$:            plp
1$:            rts

Dir1Get:       lda   #$12
               sta   r1L
               lda   #$00
               sta   r1H
               sta   r4L
               lda   #$82
               sta   r4H
               rts

Dir2Get:       lda   #$35
               sta   r1L
               lda   #$00
               sta   r1H
               sta   r4L
               lda   #$89
               sta   r4H
               rts

CheckTrack:    lda   #$00
               sta   tempc
               ldx   #$02
               lda   r1L
               beq   1$
```

```
               cmp   #$24
               bcc   2$
               ldy   curDrive
               lda   $88b7,y
               bpl   1$
               lda   r1L
               cmp   #$47
               bcs   1$

2$:            sec
               rts

1$:            clc
               rts
C_OpenDisk:    jsr   NewDisk
               txa
               bne   1$
               jsr   GetDirHead
               txa
               bne   1$

               LoadW r5, curDirHead
               jsr   ChkDkGEOS
               LoadW r4, curDirHead+$90
               ldx   #$0c
               jsr   GetPtrCurDkNm
               ldx   #r4L
               ldy   #r5L
               lda   #$12
               jsr   CopyFString
               ldx   #$00
1$:            rts

C_BlkAlloc:    ldy   #$01
               sty   r3L
               dey
               sty   r3H
C_NxtBlkAlloc: PushW r9
               PushW r3
               lda   #$00
               sta   r3H
               lda   #$fe
               sta   r3L
               ldx   #r2L
               ldy   #r3L
               jsr   Ddiv
               lda   r8L
               beq   1$
               inc   r2L
               bne   1$
               inc   r2H

1$:            jsr GetCurDirHd
               jsr CalcBlksFree
               pla
               sta r3L
               pla
               sta r3H
               ldx #$03
               lda r2H
               cmp r4H
               bne 2$
               lda r2L
               cmp r4L

2$:            beq 3$
               bcs 4$

3$:            lda r6H
               sta r4H
               lda r6L
               sta r4L
               lda r2H
               sta r5H
               lda r2L
               sta r5L

7$:            jsr SetNextFree
               txa
               bne 4$
               ldy #$00
```

```
                lda r3L
                sta (r4L),y
                iny
                lda r3H
                sta (r4L),y
                clc
                lda #$02
                adc r4L
                sta r4L
                bcc 5$
                inc r4H

5$:             lda r5L
                bne 6$
                dec r5H

6$:             dec r5L
                lda r5L
                ora r5H
                bne 7$
                ldy #$00
                tya
                sta (r4L),y
                iny
                lda $12
                bne 8$
                lda #$fe

8$:             clc
                adc #$01
                sta (r4L),y
                ldx #$00

4$:             PopW r9
                rts

GetCurDirHd:    LoadW r5, curDirHead
                rts

C_Get1stDirEntry:
                lda    #$12
                sta    r1L
                lda    #$01
                sta    r1H
                jsr    DskBufRdBlk
                LoadW r5, diskBlkBuf+2
                lda    #$00
                sta    tempf
                rts

C_GetNxtDirEntry:
                ldx #$00
                ldy #$00
                clc
                lda #$20
                adc r5L
                sta r5L
                bcc 1$
                inc r5H
1$:             lda r5H
                cmp #$80
                bne 2$
                lda r5L
                cmp #$ff

2$:             bcc 3$
                ldy #$ff
                lda diskBlkBuf+1
                sta r1H
                lda diskBlkBuf
                sta r1L
                bne 4$
                lda tempf
                bne 3$
                lda #$ff
                sta tempf
                jsr GetOffPgTS
                txa
                bne 3$
                tya
                bne 3$
```

```
4$:             jsr   DskBufRdBlk
                ldy   #$00
                LoadW r5, diskBlkBuf+2

3$:             rts

C_GetOffPgTS:   jsr   GetDirHead
                txa
                bne   1$
                LoadW r5, curDirHead
                jsr   ChkDkGEOS
                bne   2$
                ldy   #$ff
                bne   3$

2$:             MoveW curDirHead+$ab, r1
                ldy   #0

3$:             ldx   #0

1$:             rts

C_ChkDkGEOS:    ldy #$ad
                ldx #$00
                lda #$00
                sta isGEOS
2$:             lda (r5L),y
                cmp formatID,x
                bne 1$
                iny
                inx
                cpx #$0b
                bne 2$
                lda #$ff
                sta isGEOS
1$:             lda isGEOS
                rts

formatID:       .byte "GEOS format V1.0",NULL

C_GetFreeDirBlk:
                php
                sei
                lda r6L
                pha
                PushW r2
                ldx   r10L
                inx
                stx r6L
                lda #$12
                sta r1L
                lda #$01
                sta r1H
1$:             jsr DskBufRdBlk
2$:             txa
                bne 7$
                dec r6L
                beq 5$
3$:             lda diskBlkBuf
                bne 4$
                jsr SetLink
                clv
                bvc 2$
4$:             sta r1L
                lda diskBlkBuf+1
                sta r1H
                clv
                bvc 1$
5$:             ldy #$02
                ldx #$00
6$:             lda diskBlkBuf,y
                beq 7$
                tya
                clc
                adc #$20
                tay
                bcc 6$
                lda #$01
                sta r6L
                ldx #$04
                ldy r10L
```

```
                iny
                sty r10L
                cpy #$12
                bcc 3$
7$:             pla
                sta r2L
                pla
                sta r2H
                pla
                sta r6L
                plp
                rts
C_SetLink:      PushW r6
                ldy   #$48
                ldx   #$04
                lda   curDirHead,y
                beq   1$
                MoveW r1L,r3L
                jsr   SetNextFree
                MoveW r3L,diskBlkBuf
                jsr   DskBufWrBlk
                txa
                bne   1$
                MoveW r3L,r1L
                jsr   ClearBlock
1$:             PopW  r6
                rts

ClearBlock:     lda #$00
                tay
1$:             sta diskBlkBuf,y
                iny
                bne 1$
                dey
                sty diskBlkBuf+1
                jmp DskBufWrBlk
C_SetNextFree:  lda r3H
                clc
                adc interleave
                sta r6H
                lda r3L
                sta r6L
                cmp #$12
                beq 2$
                cmp #$35
                beq 2$

1$:             lda r6L
                cmp #$12
                beq 4$
                cmp #$35
                beq 4$

2$:             cmp #$24
                bcc 3$
                clc
                adc #$b9
                tax
                lda curDirHead,x
                bne 5$
                beq 4$

3$:             asl a
                asl a
                tax
                lda curDirHead,x
                beq 4$

5$:             lda r6L
                jsr CheckSector
                lda NumSectors,x
                sta r7L
                tay

7$:             jsr SetAllocBlock
                beq 6$
                inc r6H
                dey
                bne 7$
```

```
4$:             bit $8203
                bpl 8$
                lda r6L
                cmp #$24
                bcs 9$
                clc
                adc #$23
                sta r6L
                bne 10$

9$:             sec
                sbc #$22
                sta r6L
                bne 11$

8$:             inc r6L
                lda r6L

11$:     :      cmp #$24
                bcs 12$

10$:     :      sec
                sbc r3L
                sta r6H
                asl a
                adc #$04
                adc interleave
                sta r6H
                bne 1$

6$:             lda r6L
                sta r3L
                lda r6H
                sta r3H
                ldx #$00
                rts

12$:     :      ldx #$03
                rts

CheckSector:    pha
                cmp #$24
                bcc 1$
                sec
                sbc #$23

1$:             ldx #$00

2$:             cmp SideAScVals,x
                bcc 3$
                inx
                bne 2$

3$:             pla
                rts

SideAScVals:    .byte $12,$19,$1f,$24

NumSectors:     .byte $15,$13,$12,$11

SetAllocBlock:
                lda r6H
1$:             cmp r7L
                bcc 2$
                sec
                sbc r7L
                clv
                bvc 1$
2$:             sta r6H

C_AllocateBlock:
                jsr FindBAMBit
                beq 1$
                lda r6L
                cmp #$24
                bcc 2$
                lda r8H
                eor #$ff
                and dir2Head,x
                sta dir2Head,x
                clv
                bvc 3$
```

```
2$:             lda r8H
                eor #$ff
                and curDirHead,x
                sta curDirHead,x

3$:             ldx r7H
                dec curDirHead,x
                ldx #$00
                rts

1$:             ldx #$06
                rts
C_FindBAMBit:   lda r6H
                and #$07
                tax
                lda bitMask,x
                sta r8H
                lda r6L
                cmp #$24
                bcc 1$
                sec
                sbc #$24
                sta r7H
                lda r6H
                lsr a
                lsr a
                lsr a
                clc
                adc r7H
                asl r7H
                clc
                adc r7H
                tax
                lda r6L
                clc
                adc #$b9
                sta r7H
                lda dir2Head,x
                and r8H
                rts

1$:             asl a
                asl a
                sta r7H
                lda r6H
                lsr a
                lsr a
                lsr a
                sec
                adc r7H
                tax
                lda curDirHead,x
                and r8H
                rts

bitMask:        .byte $01,$02,$04,$08
                .byte $10,$20,$40,$80
C_FreeBlock:    jsr FindBAMBit
                bne 1$
                lda r6L
                cmp #$24
                bcc 2$
                lda r8H
                eor dir2Head,x
                sta dir2Head,x
                clv
                bvc 3$

2$:             lda r8H
                eor curDirHead,x
                sta curDirHead,x

3$:             ldx r7H
                inc curDirHead,x
                ldx #$00
                rts
1$:             ldx #$06
                rts
```

```
C_CalcBlksFree:lda #$00
                sta r4L
                sta r4H
                ldy #$04

2$:             lda (r5L),y
                clc
                adc r4L
                sta r4L
                bcc 1$
                inc r4H

1$:             tya
                clc
                adc #$04
                tay
                cpy #$48
                beq 1$
                cpy #$90
                bne 2$
                lda #$02
                sta r3H
                lda #$98
                sta r3L
                bit $8203
                bpl 3$
                ldy #$dd

5$:             lda (r5L),y
                clc
                adc r4L
                sta r4L
                bcc 4$
                inc r4H

4$:             iny
                bne   5$
                LoadW r3, $0530

3$:             rts
C_SetGEOSDisk:  jsr GetDirHead
                txa
                bne out
                LoadW r5, curDirHead
                jsr CalcBlksFree
                ldx #$03
                lda r4L
                ora r4H
                beq out
                lda #$00
                sta r0L
                lda #$13
                sta r3L

3$:             lda #$00
                sta r3H
                jsr SetNextFree
                txa
                beq 2$
                lda r0L
                bne out
                lda #$01
                sta r3L
                sta r0L
                bne 3$

2$:             lda   r3H
                sta   r1H
                lda   r3L
                sta   r1L
                jsr   ClearBlock
                txa
                bne   out
                MoveW r1L,curDirHead+$ab
                ldy   #$bc
                ldx   #$0f
```

```
PutIDString:    lda formatID,x
                sta curDirHead,y
                dey
                dex
                bpl PutIDString
                jsr PutDirHead

out:            rts
C_InitForIO:    php
                pla
                sta temp1
                sei

                lda $02b0
                bne 1$
                jsr MoveTransfer

1$:             lda   $01
                sta   temp3
                lda   #$36
                sta   $01
                lda   $d01a
                sta   temp2
                lda   $d030
                sta   temp0
                ldy   #$00
                sty   $d030
                sty   $d01a
                lda   #$7f
                sta   $d019
                sta   $dc0d
                sta   $dd0d
                LoadW $0314, $02f6
                LoadW $0318, $02fb
                lda   #$3f
                sta   $dd02
                lda   $d015
                sta   temp4
                sty   $d015
                sty   $dd05
                iny
                sty   $dd04
                lda   #$81
                sta   $dd0d
                lda   #$09
                sta   $dd0e
                ldy   #$2c

2$:             lda $d012
                cmp $8f
                beq 2$
                sta $8f
                dey
                bne 2$
                rts
C_DoneWithIO:   sei
                lda temp0
                sta $d030
                lda temp4
                sta $d015
                lda #$7f
                sta $dd0d
                lda $dd0d
                lda temp2
                sta $d01a
                lda temp3
                sta $01
                lda temp1
                pha
                plp
                rts
```

```
C_EnterTurbo:   lda curDrive
                jsr SetDevice
                ldx #$00
                rts

C_ExitTurbo:    lda #$08
                sta interleave
                rts

C_ChangeDskDev:sta curDrive
                sta $ba
                ldx #$00
                rts

C_NewDisk:      jsr EnterTurbo
                rts
C_ReadBlock:    jsr CheckTrack
                bcc 1$
                jsr Do_Fetch
1$:             ldy #$00
                rts

C_ReadLink:     jsr CheckTrack
                bcc 1$
                ldy #$91
                jsr LoadLink

1$:             rts

C_WriteBlock:   jsr CheckTrack
                bcc 1$
                jsr Do_Stash

1$:             rts

C_VerWriteBlock:
                jsr CheckTrack
                bcc 1$
                ldx #$00

1$:             rts

Do_Fetch:       ldy #$91
                bne LoadPage

Do_Stash:       ldy #$90
                bne LoadPage

;** the code most heavily modified to support the banked ram begins here **


LoadLink:       lda   $02a7        ;quickie is transfer routine installed?
                bne   1$           ;if not, then do so.
                jsr   MoveTransfer ;this routine should be unnecessary, but
                                   ;one never knows.
1$:             PushW r2
                LoadW r2, $0002    ;fetch links only
                bne   SavePs

LoadPage:       lda   $02a7        ;as above
                bne   1$
                jsr   MoveTransfer

1$:             PushW r2
                LoadW r2, $0100    ;fetch page

SavePs:
                PushW r0
                PushW r1
                PushW r3
                tya
                and   #%00000001   ;mask out high bits
                pha                ;and save
                lda   r1L
                cmp   #$24         ;track request>35
                bcc   2$           ;if .cs then do some math to access
                sec                ;correct page values
                sbc   #$23
2$:             tay
                dey
                lda   RamDiskTab,y ;RAM page translation
                clc
                adc   r1H          ;sector
```

```
        sta   r0H
        txa
        ldx   #%01110001 ;base value for BCR=bank 1, CRAM set to $0400
        cpy   #11        ;work out the bank
        bcc   3$         ;value based on
        cpy   #23        ;sector requested
        bcc   4$
        inx
4$:     inx

3$:     lda   r1L         ;if sector>35 then
        cmp   #$24        ;increase bank value
        bcc   5$          ;by 3 the hard way.
        inx
        inx
        inx

5$:     stx   r3L         ;source
        LoadB r3H, 0      ;destination
        sta   r0L
        MoveW r4L, r1L    ;set up for fetch

        pla               ;get back command value
        bne   6$          ;on .ne = fetch

        PushB r3H         ;stash, then do flip
        MoveW r0L, r1L
        MoveW r4L, r0L
        MoveB r3L, r3H
        PopB  r3L

6$:     lda   r3H
        beq   11$

        lda   r1H
        bne   12$

11$:    lda   r0H         ;determine whether page requested
12$:    and   #$f0        ;lies under IO block
        cmp   #$d0        ;if .eq then do slow transfer, otherwise...
        beq   13$

        jsr   $02a7       ;do fast transfer
        bne   14$

13$:    jsr   $02c4       ;do under IO trans.

14$:    PopW  r3          ;restore psreg.'s
        PopW  r1
        PopW  r0
        PopW  r2
        ldx   #0
        lda   #0
        rts               ;and return no errors
RamDiskTab:
        .byte $04, $1a, $30, $46, $5c, $72, $88, $9e
        .byte $b4, $ca, $e0, $04, $1a, $30, $46, $5c
        .byte $72, $88, $9c, $b0, $c4, $d8, $ec, $04
        .byte $18, $2b, $3e, $51, $64, $77, $8a, $9c
        .byte $ae, $c0, $d2, $e4, $00

C_MoveTransfer:
        PushW r0
        PushW r1
        PushW r2
        LoadW r0, mvRoutine
        LoadW r1, $02a7
        LoadW r2, routineSize
        jsr   MoveData
        LoadW $0314, $02f6      ;IRQvector
        LoadW $0318, $02fb      ;NMIvector
        PopW  r2
        PopW  r1
        PopW  r0
        rts
```

```
mvRoutine:
             PushB CPU_DATA
             LoadB CPU_DATA, IO_IN
             ldy   r2L
             ldx   r3H
1$:          dey
             lda   r3L
             sta   BCR
             lda   (r0L),y
             stx   BCR
             sta   (r1L),y
             tya
             bne   1$

             beq   Done

             PushB CPU_DATA
             ldy   r2L
2$:          dey
             ldx   #IO_IN
             stx   CPU_DATA
             lda   r3L
             sta   BCR
             lda   #$30
             sta   CPU_DATA
             lda   (r0L),y

             stx   CPU_DATA
             ldx   r3H
             stx   BCR
             beq   3$
             ldx   #$30
             stx   CPU_DATA
3$:          sta   (r1L),y
             tya
             bne   2$
             ldx   #$35
             stx   CPU_DATA

Done:        sta   BCR
             PopB  CPU_DATA
             rts

IRQVEC:      pla
             tay
             pla
             tax
             pla
NMIVEC:      rti

e_mvRoutine:

routineSize    = e_mvRoutine-mvRoutine

;internal variable space trails code

temp0:     .byte 0
temp1:     .byte 0
temp2:     .byte 0
temp3:     .byte 0
temp4:     .byte 0
temp5:     .byte 0
temp6:     .byte 0
temp7:     .byte 0
temp8:     .byte 0
temp9:     .byte 0
tempa:     .byte 0
tempb:     .byte 0
tempc:     .byte 0
tempd:     .byte 0
tempe:     .byte 0
tempf:     .byte 0

.end
```

Ramfinder

Identify, stash and fetch

by Ian Adam

Introduction

Adding an external RAM cartridge to a Commodore 64 or 128 can greatly increase its power and speed. For example, program overlays and disk files can be held in RAM, for near-instant access. A word processor or spreadsheet can now handle vastly larger documents or tables, rivalling those on any other personal computer. Another of my favourite uses is to prepare a number of graphics images, either high-resolution or low-res, and stash them in the RAM cartridge. When these are fetched rapidly, some pretty good animation can be created. Many other kinds of programs can use that extra capacity for a variety of different purposes, if only they know it's there.

The speed of the RAM cartridges is truly amazing. The RAM Expansion Controller is a special-purpose Direct Memory Access chip; it has a very limited instruction set, and is optimized for just one purpose - moving data. As a result, the data transfer rate is one byte per clock cycle, or one million bytes per second. This is far higher than with any other method, even much higher than you could achieve with hand-crafted machine language (a maximum of 70,000 cycles per second). Compared to loading data from a 1541 disk drive... well, there's just no comparison. When programming animation with the cartridge, I find that it's actually necessary to introduce delay loops in order to keep the animation down to a reasonable speed! The RAM cartridge can load high-resolution images about twice as fast as the video chip can display them, and four times as fast as the human mind can perceive them.

With all of these capabilities at hand, it follows that the thorough programmer will take the time to write programs in such a way that external RAM is taken advantage of. After all, there's no sense in the user buying a cartridge, if programs for the computer don't make use of the facility. Besides, your programs will look so much more impressive when they use all of the power at hand.

Right away, though, you run into the little problem of finding out how much RAM, if any, you have to work with. The standard Commodore operating system doesn't test for external RAM, and the cartridge itself doesn't go out of its way to tell you that it's present, so you have to devise a way to find out for yourself. What's more, while the cartridge does have a status byte to tell you how big it is, unfortunately two of the three available cartridges can have the same status byte!

That's the bad news. The good news is that all three cartridges use the same ten instruction registers, so they can all be controlled with the same commands. Furthermore, they are all located at the same address in the I/O block, at $df00 to $df0a, regardless of what computer they are installed in. Here are the cartridges Commodore has made available for the 64 and 128:

Model	Banks	RAM	Status Byte	For	Bank #S
1764	4	256K	xxx1xxxx	C 64	0 to 3
1700	2	128K	xxx0xxxx	C 128	0 and 1
1750	8	512K	xxx1xxxx	C 128	0 to 7

Check the larger accompanying table for further details on the meaning of the various control registers. In theory at least, the status byte (at $df00) should be a sufficient signature to identify the cartridge uniquely, once you know which model of computer it's installed in. After all, there is no duplication of the byte within each computer model. The 64 is not supposed to use a 128-model cartridge, since its meager power supply is barely capable of powering the computer itself, let alone any RAM expansion. The 1764 comes with an upgraded power supply, and so would not be of interest to an owner of a 128.

In the real world, however, you must remember that hardware could be combined in ways that your program might not have anticipated. For example, a Commodore 128 could be running a C64 program in 64 mode, and still have access to either of the 128-model expansion cartridges. You could also encounter a 64-model cartridge being operated in a 128. Thus, there is no guarantee that the cartridge will be the one you expect from its signature byte.

What's more, there still remains the problem of sorting out whether a cartridge is present at all. A genuine status register can take on many different values at different times, as a glance at the table will illustrate. However, if there is no cartridge present, a read of the address of the non-existent status register gives a random value, which could mimic the status byte of a cartridge. All in all, an interesting programming challenge.

The *Ramfinder* program

To the rescue rides the *Ramfinder* program. The challenge of detecting RAM isn't all that difficult to deal with, and any experienced programmer could tackle it reasonably well. However, I've always felt that the programmer should be freed to deal with important matters like making his or her program work properly, and not have to spend time and energy worrying about little details like what sort of hardware is attached.

To help out with this, I prepared the *Ramfinder* program, which has several useful advantages. This compact program will run in either the 64 or the 128, with no preference for either. As a further advantage, it is fully relocatable to any available start address (SA), so it will be compatible with just about any program you may want to write. What's more, it has three handy entry points:

```
sys sa      identify RAM cartridge & report
sys sa+4    STASH to expansion RAM
sys sa+7    FETCH from expansion RAM
```

All of this usefulness is packed into just over 100 bytes of machine language.

Of the three entry points, the first entry is the key one, because it will check whether or not a RAM cartridge is present. If none is found, it will return a value of zero. If it succeeds in finding external RAM, then the program will perform a couple of additional tests to identify which cartridge is present. It will return a result of 2, 4, or 8, representing the number of banks of memory available. The result is stored in zero-page memory, where it can be retrieved with a simple **lda $fb**, or a **peek(251)** from BASIC. The result is also held in the accumulator on departure.

The second and third entry points will perform very simple STASH and FETCH operations. Because the 64 and 128 manage their memory in such different fashions, these operations will not deal with subtleties like data in hidden memory banks. However, they are ideal for my favourite task, pulling graphic screens in and out of memory. To use these operations, put the number of the external RAM bank that you want to use in $fb (from BASIC: **poke 251, bank#**. For example, if you have a four-bank cartridge, select a bank number of 0 to 3). Load the microprocessor registers as follows:

accumulator	high byte of expansion address
X register	high byte of computer address
Y register	high byte of length of transfer (all low bytes will be set to zero)

If you are working in machine language, this is very straightforward. If you are working in BASIC 2.0 on the 64, just POKE these three values into memory locations 780 through 782, then **sys sa+4**. With BASIC 7.0 on the 128, the values can be transferred directly by the extended SYS command (as an example: **sys sa+4,8,4,4** to stash a low-res screen in the cartridge at $0800), but be sure you are in Bank 15 when you use the program.

If you find you need a more comprehensive STASH and FETCH capability, see Dale Castello's wedge commands for the 64 in *Transactor*, Volume 8 Issue 2, page 38 or use the built-in commands in BASIC 7.0 on the 128.

Starting *Ramfinder*

How you use the *Ramfinder* program is at least partly dependent on what you want to do. If you are doing machine language programming and want to deal with the expansion cartridge issue painlessly, then type in the source code and add it to your library of useful routines. Again, note that the code is fully relocatable, so you should find it most accommodating in getting along with other routines. Its only requirement is for one byte of space in zero page, at $fb. A JSR to the start of the code will identify the expansion RAM available, and on return the accumulator will contain the number of 64K banks available. You can use the stash and fetch commands if suitable to your needs.

For you non-ML programmers, a BASIC loader is also supplied. Type the program in, being especially careful with the DATA statements at the end. Be sure to save a copy of the program before running it. When you do run the program, it gives a brief description of itself, then asks for the address to load the machine language into. Enter the address of any suitable free RAM (in the 128, you must be in Bank 15, so the load address must be less than 16270 in order to stay in non-banked RAM). If you are unsure, just press Return and the code will be loaded into the cassette buffer automatically. The program will then give further instructions for each of its routines.

If you want to incorporate the routine into other BASIC programs that you write, you have my blessings. Of course, you won't need to include all of the detailed instructions - just the DATA statements and their loader.

How it works

The only way to detect a RAM cartridge reliably is actually to command it to work, then find out whether it performed as expected. As I mentioned, the status byte should tell you about the cartridge, but unfortunately it cannot be relied upon. Reading this when a cartridge is not installed may yield a phantom random number, leading to the erroneous conclusion that extra RAM is available.

To get around this problem, the program puts a known byte in zero page (the seed value 1 in its storage location at $fb), then commands the cartridge to save the page in expansion RAM. The value in $fb is changed (to #$b5, a convenient alternative), then the page is fetched back. By checking what value remains, the presence or absence of a cartridge can be deduced. If none, then a value of zero is returned.

With the knowledge that a RAM cartridge is present, the status byte can be read reliably. If bit 4 is clear, then the cartridge must be the 1700, and the task is finished.

Otherwise, there are still two possibilities, so one more test is required. This depends on the characteristic that the bank addresses 'wrap around'; that is to say, access to a bank beyond those in place will be decoded into the existing banks. To make use of this, remember that zero page has already been stashed in bank 0: this page will now be verified against bank 4. In the 1764 (the 256K cartridge) bank 4 is read as bank 0, so the verify operation succeeds. In the 1750, bank 4 is distinct and different from bank 0, so the verify fails. Thus, the detection is complete.

Table of REU Registers

REGISTER	ADDRESS	TYPE	MEANING	
STATUS	$DF00	Read Only	bits 0-3	version
			bit 4	'size'
			bit 5	1 = verify error
			bit 6	1 = complete
			bit 7	interrupt pending
COMMAND	$DF01	R/W	bits 0,1	transfer type
			bit 4	0 = $FF00 trigger
			bit 5	1 = reset parameters
			bit 7	execute
ADDRESS	$DF02	R/W	low byte,	computer address
	$DF03	R/W	high byte	
EXP ADDR	$DF04	R/W	low byte,	expansion RAM address
	$DF05	R/W	high byte	
BANK	$DF06	R/W	RAM bank #,	low bits only
LENGTH	$DF07	R/W	low byte,	length of transfer
	$DF08	R/W	high byte	
IRQ MASK	$DF09	R/W	bit 5	IRQ on verify error
			bit 6	IRQ on completion
			bit 7	enable interrupts
INCREMENT	$DF0A	R/W	bit 0	0 = increment RAM addrs
				1 = fix RAM address
			bit 1	0 = increment host addrs
				1 = fix host address

The benefits are yours

How you use this program is up to you. It is most useful when combined with other programs, whether in BASIC or machine language. *Ramfinder* is compatible with both; its length and transportability make it easy to incorporate with other programs of all types. *[If you've ever plugged in your REU and booted GEOS only to discover that the REU wasn't seated properly and thus was not seen by the system, you'll recognize another use for the program as published. - MO]*

There are two beneficiaries of this process; one is the user, whose investment in an expansion cartridge is rewarded with programs that offer more power and speed. The other beneficiary is you, the programmer - your programs will be slicker and more popular when they take advantage of all the resources available to them. Ultimately, that reflects favourably on your ability as a programmer!

Listing 1: *ramfinder.bas*

```
PK  100 print chr$(147):print"** ramfinder **"
PH  110 print:print"(c) ian adam"
PK  120 print"vancouver bc 1988"
GP  130 :
GK  140 print:print"this short program will identify an"
CA  150 print"external ram cartridge attached to"
GG  160 print"the computer, and indicate its size"
JM  170 print"in 64k banks. the program will operate"
OL  180 print"without modification in either"
II  190 print"the 64 or the 128."
MD  200 :
DK  210 print:print"the program is fully relocatable to"
HL  220 print"any start address, for compatibility."
OE  230 print"good locations are 828 in the 64,"
HK  240 print"and 2810 in the 128."
OG  250 :
FE  260 print:input"your start address";a$
IO  270 sa=val(a$):if sa=0 then sa=828 -2000*(peek(46)>27)
MI  280 :
HP  290 for i=sa to sa+117
GJ  300 read a:poke i,a
KD  310 next
EL  320 ;
CF  330 print chr$(147):print"identifying ram:"
CE  340 print:print"sys"sa
JA  350 print:print"this command will locate a ram"
KL  360 print"cartridge and indicate the number of"
EI  370 print"banks in location $00fb (251)."
BP  380 print"a value of 0 means no expansion ram."
GF  390 print"options are 2, 4, or 8 banks of 64k."
HI  400 print:print"number of banks installed now:"
DN  410 sys sa
DF  420 print:print"peek(251) =" peek(251)
CD  430 print:print"press return to continue"
FB  440 input a$
GD  450 :
HP  460 print chr$(147):print"stash and fetch:"
GC  470 print:print"to start, set these parameters;  all"
PF  480 print"others will be set to zero:"
CC  490 print:print"poke 251, external ram bank #"
ND  500 print"accumulator = msb external ram address"
KB  510 print"x register  = msb computer address"
GM  520 print"y register  = msb length to transfer"
GI  530 :
LN  540 print:print"on the 64, poke these three values"
LL  550 print"into locations 780 to 782, then...":print
AA  560 print"sys"sa+4" to stash"
DK  570 print"sys"sa+7" to fetch"
IL  580 :
NK  590 print:print"on the 128, use the extended sys"
CF  600 print"command.  for example, to save this"
IM  610 print"screen at the start of external ram:"
BG  620 print:print"poke 251,0:sys"sa+4",0,4,4"
KO  630 :
AI  640 end
OP  650 :
FG  660 data169,0,240,6,24,144,80,56,176,77,120,162,10,157,0,223,202,208
DO  670 data250,232,142,8,223,134,251,169,180,141,1,223,169,181,133,251,141,1
CG  680 data223,197,251,240,40,173,0,223,41,16,208,4,169,2,208,31,169,4
GA  690 data141,6,223,169,1,133,251,169,183,141,1,223,173,0,223,41,32,208
NE  700 data4,169,4,208,6,169,8,208,2,169,0,133,251,88,96,141,5,223
ON  710 data142,3,223,140,8,223,166,251,142,6,223,169,0,141,2,223,141,4
BH  720 data223,141,7,223,105,180,141,1,223,96
```

Listing 2: *ramfinder.src*

```
;******************
;*               *
;*               *
;*  external ram  *
;*   identifier   *
;*               *
;*               *
;*  for the c-64  *
;*   and  c-128   *
;*               *
;*               *
;******************
;
;
;
; (c) ian adam
;     may 1988
; vancouver bc
;
;
;
zpbank = $00fb
rec    = $df00
;
;
;
; 'jump table'
;
; start address = test exram
; sa + 4        = stash
; sa + 7        = fetch
;
;
; dummy start address:
;
* = $2000
;
; code is fully
; relocatable,
; and executes
; on either the 64
; or 128 (bank 15)
;
;
      lda #$00     ;entry to test ram
      beq trial
;
      clc          ;entry for stash
      bcc stash
;
      sec          ;entry for fetch
      bcs stash
;
;
;
; *************
; *           *
; *   trial   *
; *   stash   *
; *           *
; *************
;
; move zero page from computer
; to external ram bank 0, as
; a test of cartridge operation:
;
;
;
trial sei
      ldx #$0a
clear sta rec,x    ;clear registers
      dex
      bne clear
;
      inx
      stx $df08    ;move 1 page
      stx zpbank   ;plant seed
;
      lda #$b4     ;control byte = stash
      sta rec+1    ;execute
;
;
; ********************************
;
; the value 1 was saved as a test.
; if the stash was successful,
; then that seed value will be
; restored when the same page is
; fetched back.  thus, this
; sequence will detect a working
; external ram cartridge:
;
;
;
      lda #$b5     ;control byte = fetch
      sta zpbank
      sta rec+1    ;execute
;
      cmp zpbank   ;check it
      beq noram    ;exit if no exram found
;
; external ram located -
; find out how much:
;
      lda rec
      and #$10     ;check # of banks
      bne more
;
;
;
; if bit 4 is clear, then
; there must be 128k of
; external ram, in 2 banks:
;
      lda #$02
      bne exit
;
;
; ******************************
;
; if bit 4 is set, then there
; are either 4 banks (256k) or
; 8 banks (512k). test for this
; by verifying bank 4.  if there
; are only 4 banks, bank 0 will
; read as bank 4, and verify ok.
; if there are 8 banks, a verify
; error will result:
;
;
;
more  lda #$04
      sta $df06    ;set bank 4
      lda #$01
      sta zpbank
      lda #$b7     ;control byte = verify
;
      sta rec+1    ;execute
;
      lda rec      ;check status
      and #$20     ; for error
      bne most
;
      lda #$04     ;no error, 4 banks
      bne exit
;
most  lda #$08     ;error = 8 banks
      bne exit
;
;
; ***************
; *             *
; *  exit with  *
; *   message   *
; *             *
; ***************
;
;
noram lda #$00
exit  sta zpbank   ;leave message
      cli
      rts
;
; the # of expansion banks will
; be left in zpbank ($00fb).
; 0 banks means no external ram.
; options:  2, 4, or 8 banks.
;
;
; ********************
; *                  *
; *  stash and fetch  *
; *                  *
; ********************
;
;
;
; a = high byte expansion address
; x = high byte computer address
; y = high byte of length
; bank number in zpbank
; all other parameters set to 0
;
;
stash sta $df05    ;external ram address
      stx $df03    ;set computer address
      sty $df08    ;set length
      ldx zpbank
      stx $df06    ;set bank
;
      lda #0       ;set low bytes to 0
      sta $df02
      sta $df04
      sta $df07
;
;
; build control byte and execute:
;
; the carry bit will increment the
; control byte by 1, when a fetch
; was specified in the jump table
;
;
      adc #$b4     ;build control byte
      sta rec+1    ;execute
;
      rts          ;all finished
;
.end
```

Encryptor

Password Protection for C64

by Jim Frost

First, let me set the record straight. I believe in neither copy protection nor stealing programs. Why then did I write *Encryptor*? Computing at my house is a family pastime. Mother does word processing and neatens documentation so no one can find it. Jim (Grandpa is James R., I'm James S. and he's James T.) plays games and writes music. My daughter, Summer, writes BASIC games that she definitely does not want her older brother to touch. Jim naturally delights in analyzing, modifying and criticizing Summer's latest effort. With Summer's work encrypted, I spend less time preventing fights and more time writing programs.

If you have similar problems and want to protect BASIC programs from unauthorized use, with *Encryptor*, it's easy! To use *Encryptor*, load and run the BASIC loader. Nothing appears to happen; however, BASIC's LOAD and SAVE vectors are changed to access encryption routines. A password prompt appears when LOAD or SAVE is requested. For normal (plain-text) loading, simply press **return**. To save an encrypted program, enter a password in the spaces immediately following the prompt. Any password will work - provided it does not begin with a space and is not longer than eleven characters.

Loading encrypted programs involves the same procedure as saving them. Type your password, then press **return** and let the computer work. Unless you use the correct password, loaded programs will be hopelessly scrambled, and the operating system may even lock due to confusion while relinking gibberish.

Encryptor works by exclusive-ORing the ninth and eleventh password characters with the first byte of your BASIC program. To provide additional confusion, the password is then rotated and the process repeated byte by byte until the entire program is encrypted. Because XORing zeroes changes nothing, a password consisting of 11 @ characters (screen code 0) will not encrypt. More accurately, the encrypted version will be identical to the plain text. I have slowed the encryption processes so that you can watch it work. If you prefer lightning speed, change the last data element from zero to one.

While *Encryptor* will make breaking into your programs difficult, no encryption method is infallible. With time and effort, any protection can be overcome. For those who savour the challenge of overcoming any obstacle, I have included data statements to create an encrypted BASIC program on disk. The password is my middle name.

Listing 1: encryptor.s

```
*********************************
*                               *
*           ENCRYPTOR           *
*                               *
*   LOADS AND SAVES ENCRYPTED   *
*      FILES. TO USE ENTER      *
*      PASSWORD AT  PROMPT      *
*  A SPACE PASSWORD BYPASSES    *
*         THE PROGRAM           *
*                               *
*  J FROST       rev 4MAY89     *
*                               *
*********************************

ILOAD       = $0330
SAVE        = $F5ED
LOAD        = $F4A5
CHROUT      = $FFD2
STOBUF      = $A560
COUNT       = $FD

            ORG $033C

            LDX #$03

NEWPOINT    LDA VTAB,X      ;change LOAD and SAVE
            STA ILOAD,X     ;pointers to encrypt code
            DEX
            BPL NEWPOINT
            RTS

VTAB        DA  ELOAD       ;encrypt addresses
            DA  ESAVE

* Encrypted load Routine

ELOAD       PHA             ;save load/verify flag (in A)
            JSR PWDMSG      ;get password

            PLA             ;recover load/verify flag
            JSR LOAD        ;do normal load
            BCS LFAULT      ;if load error

            STX $2D         ;else save end of
            STY $2E         ;load address

            JSR ENCRYPT     ;mess things up

            LDX $2D         ;then recover end of
            LDY $2E         ;load address
            CLC             ;carry indicates fault

LFAULT      RTS

* Encrypted save routine
```

```
ESAVE       JSR PWDMSG    ;get password
            JSR ENCRYPT   ;scramble
            JSR SAVE      ;then normal save
            JSR ENCRYPT   ;unscramble
            CLC           ;carry indicates error
            RTS

* print password message then input password

PWDMSG      LDX #$00

PWM1        LDA TEXT,X    ;get text character
            BEQ PASWRD    ;zero flags end of string

            JSR CHROUT    ;non-zero - print
            INX
            BNE PWM1      ;and loop

PASWRD      JSR STOBUF    ;input password
            RTS

TEXT        HEX 93        ;CLR
TXT         'password:',00

* encrypt/decrypt routine

ENCRYPT     LDA $0409     ;if space unencrypted
            CMP #$20      ;LOAD/SAVE requested
            BEQ NOENC     ;skip encryption

            LDA #$00
            STA COUNT     ;count rotations

            LDA $2B       ;copy start of BASIC
            STA $FB       ;address from #43

            LDA $2C
            STA $FC       ;to $FB

* Encrypt loop - One cycle with a password will scramble.
* A second pass with the same password changes encrypted to plaintext.

ELOOP       LDY #$00      ;zero pointer

            LDA ($FB),Y   ;fetch program character

            EOR $0411     ;XOR with 9th
            EOR $0413     ;and 11th password character
            STA ($FB),Y   ;and replace character

            INC $FB       ;advance character pointer
            BNE ROTATE    ;low byte

            INC $FC       ;and high if needed

* scramble password for next pass

ROTATE      LDX $0413     ;save last password character
            LDY #$09

ROT1        LDA $0409,Y   ;rotate 10 password characters
            STA $040A,Y   ;to right one bit
            DEY
            BPL ROT1

            TXA           ;and rotate last
            STA $0409     ;to first

            LDA FSTFLG    ;do it fast?
            BNE QUICK     ;if non-zero, hurry

            LDX #$D0      ;else time delay
            LDY #$00

TIMDEL      INY
            BNE TIMDEL

            INX
            BNE TIMDEL

* advance count and test for end of BASIC

QUICK       LDX COUNT
            INX           ;advance count
            CPX #11       ;last password character?
            BNE TEST      ;test end of BASIC

            LDX #$00

TEST        STX COUNT
            LDA $FC       ;reached end of program?
            CMP $2E       ;high bytes match?
            BNE ELOOP     ;no then keep working

            LDA $FB       ;else test low bytes
            CMP $2D
            BNE ELOOP

            LDA COUNT     ;password centred?
            BNE ROTATE    ;loop until it is

* return to BASIC if no encrypt or when finished

NOENC       RTS

FSTFLG      HEX 00        ;any nonzero speeds encryption
```

Listing 2: encryptor.bas

```
BG  100 rem places encryptor in cassette buffer
EB  120 nd=180: sa=827  : ch=21121
DC  130 for i=1 to nd: read x:pokesa+i,x
EC  140 ch=ch-x:next
FB  150 if ch then print "data error":stop
PM  160 print "data ok, encryptor installed
CI  170 sys 828:end
IH  260 :
BG  270 data 162,  3,189, 72,  3,157, 48,  3
AI  280 data 202, 16,247, 96, 76,  3, 99,  3
EB  290 data  72, 32,113,  3,104, 32,165,244
PE  300 data 176, 12,134, 45,132, 46, 32,141
HM  310 data   3,166, 45,164, 46, 24, 96, 32
ED  320 data 113,  3, 32,141,  3, 32,237,245
CA  330 data  32,141,  3, 24, 96,162,  0,189
KC  340 data 130,  3,240,  6, 32,210,255,232
NJ  350 data 208,245, 32, 96,165, 96,147, 80
LP  360 data  65, 83, 83, 87, 79, 82, 68, 58
BG  370 data   0,173,  9,  4,201, 32,240, 90
NM  380 data 169,  0,133,253,165, 43,133,251
GC  390 data 165, 44,133,252,160,  0,177,251
HA  400 data  77, 17,  4, 77, 19,  4,145,251
BP  410 data 230,251,208,  2,230,252,174, 19
AJ  420 data   4,160,  9,185,  9,  4,153, 10
JD  430 data   4,136, 16,247,138,141,  9,  4
LB  440 data 173,239,  3,208, 10,162,208,160
MI  450 data   0,200,208,253,232,208,250,166
KK  460 data 253,232,224, 11,208,  2,162,  0
ID  470 data 134,253,165,252,197, 46,208,188
EE  480 data 165,251,197, 45,208,182,165,253
KO  490 data 208,196, 96,  0
```

Listing 3: makescram.bas

```
MG  100 rem transactor standard program generator
EJ  110 n$="scrambled.bas"
KF  120 nd=43: sa=2049 : ch=2875
KO  130 for i=1 to nd: read x
EC  140 ch=ch-x:next
FB  150 if ch then print "data error":stop
BM  160 print "data ok, now creating file."
CM  170 restore
CH  180 open 1,8,1,"0:"+n$
HM  190 hi=int(sa/256):lo=sa-256*hi
NA  200 print#1,chr$(lo)chr$(hi);
KD  210 for i=1 to nd:read x
HE  220 print#1,chr$(x);:next
JL  230 close 1
MP  240 print"prg file'";n$;"'created..."
MH  250 print"this generator no longer needed."
MD  260 print "program created will not run"
CJ  265 print "unless you have the password"
CL  270 data  21,  8,253, 48,138, 37, 70, 93
AJ  280 data  64,  5,103, 87, 79,  4,126, 76
NG  290 data  75, 69, 65, 18, 15, 59,111,  3
AL  300 data 189, 18, 39, 76, 84, 53, 65,  5
PK  310 data  96, 69, 65,115,115, 87, 65, 34
AE  320 data  18, 18, 37
```

Pop-ASCII For The Commodore 64

A handy pop-up utility

by Peter M.L. Lottrup

Here's a programming aid that you won't want to be without once you've given it a try. With *Pop-ASCII* installed, you'll have immediate access to a pop-up window, displaying a list of ASCII codes in decimal and hexadecimal, along with the character corresponding to that code (or a three-character code, representing non-printable characters, like colours, reverse, cursors, etc). And you won't even lose the screen beneath *Pop-ASCII*, as the utility will restore it for you once it is done. Say goodbye to programming manuals and charts forever!

Using *Pop-ASCII*

When you call *Pop-ASCII*, using the 'hot-key' combination Commodore-RESTORE, *Pop-ASCII* will spring to life on the centre of your current screen. *Pop-ASCII* is a 3-D window, where ASCII codes are displayed in decimal and hexadecimal, along with the corresponding character codes. Fourteen characters are shown per screen, and you may quickly shift through all characters using the up and down cursor keys to move forwards or back. The program starts by displaying character 32, the default starting point, but you may shift through all 255 characters.

Pop-ASCII is an all-machine language program, which loads in the following address space:

```
Start Address: $C000
End   Address: $C32E
```

Once you have typed in the program, save it. If you plan to use it with the customizing loader program included, use the name **ml-popascii** for the save.

To install the program in memory without using the loader program, type:

```
load "ml-popascii",8,1
new
sys 49152
```

Pop-ASCII will then be active and waiting for you to press the CBM-RESTORE keys. When this happens, you'll see *Pop-ASCII* jump to life. If, for any reason, *Pop-ASCII* ceases to function, you may reactivate it by simply typing **sys 49152**. RUN-STOP/RESTORE will not deactivate *Pop-ASCII*.

The customizing loader

You may customize *Pop-ASCII* to your own preferences. Colours and activation keys may be changed. Any control key (SHIFT, CONTROL, CBM, CBM+SHIFT, etc.) plus RESTORE may be used to activate *Pop-ASCII*.

To make customizing easy, a loader program has been included. It is written in BASIC. Type it in and save it. To customize *Pop-ASCII*, simply change the values in the DATA statements in lines 100-140.

Line 100 selects the background colour of the *Pop-ASCII* window. The current colour is cyan (print code 159). Replace this value for the ASCII print code value of the colour you wish to use.

Line 105 selects the shadow colour of the window (currently black) in the same way.

Line 110 determines what combination of keys (in conjunction with RESTORE) will activate *Pop-ASCII*. A value of one selects a SHIFT key, a value of two the Commodore key, and a value of four the CTRL key. You may combine more than one of these keys, by adding the values. For example, a value of 3 selects the SHIFT+CBM+RESTORE keys to activate *Pop-ASCII*.

Line 120 selects the character used to scroll the list of characters a screen forwards (currently cursor down).

Line 130 selects the backward shift key (currently cursor up).

Line 140 selects the *Pop-ASCII* 'quit' key (currently **q**).

The abbreviations

Pop-ASCII uses a list of three-letter codes for non-printable characters. Most of them are quite direct, like BLK for black or RON for Reverse-On. Here's a list of abbreviations used:

```
WHT - White
DIS - Disable SHIFT-CBM
ENA - Enable SHIFT-CBM
RET - Return
LWR - Lowercase
DWN - Cursor Down
RON - Reverse On
HME - Home
DEL - Delete
RED - Red
RHT - Cursor Right
GRN - Green
BLU - Blue
SPC - Space
ORG - Orange
SRT - Shift-Return
UPP - Uppercase
BLK - Black
CUP - Cursor Up
ROF - Reverse Off
CLS - Clear Screen
INS - Insert
BRN - Brown
LRD - Light Red
GR1 - Gray 1
GR2 - Gray 2
LGR - Light Green
LBL - Light Blue
GR3 - Gray 3
PUR - Purple
LFT - Cursor Left
YEL - Yellow
CYN - Cyan
SPC - Space
```

Programming notes

The NMI interrupt vector was selected to activate *Pop-ASCII*, providing the easiest and shortest way of interrupting a program and activating a memory-resident utility. When *Pop-ASCII* is called, current cursor colour and address are stored, along with screen and colour memory.

This information is restored upon exit from the utility. The memory area below BASIC ROM ($A000-$A800) is used for this storage. Aside from this memory, addresses 820-827 are used for miscellaneous data storage. The program itself resides at memory addresses $C000-$C32E.

I have been using *Pop-ASCII* for quite some time now, and find it incredibly handy. I use it for quick hex-dec conversions, and to find all necessary character codes.

Just think about how often you have found yourself searching for that Commodore manual just to find the code for one of the function keys or some special character code. With *Pop-ASCII* installed, you'll be able to remain at the keyboard instead of rummaging through your bookshelves.

Listing 1: *"popascii.src"*

```
NI  1000 open2,8,1,"0:ml-popascii"
DM  1010 sys 700
PA  1020 .opt p2
HP  1030 *= $c000
NP  1040 ;--- first save screen & color ---
HF  1050             lda #<newer
ME  1060             sta $0318   ;--- new irq low ---
HG  1070             lda #>newer
AF  1080             sta $0319   ;--- new irq high ---
DF  1090             lda #<setter
JL  1100             ldy #>setter
LA  1110             sta $0302
MI  1120             sty $0303   ;--- make sure new vect. stays
GF  1130             rts
KF  1140 setter      lda #<newer
ND  1150             sta $0318
BM  1160             lda #>newer
CF  1170             sta $0319
HI  1180             lda #0
DB  1190             sta active
AM  1200             jmp $a483
IL  1210 newer       pha
FG  1220             lda 653
CA  1230             cmp #2
KK  1240             beq ours
MB  1250 ignore      pla
NC  1260             jmp $fe47
NF  1270 ours        lda active
IE  1280             bne ignore
JF  1290             inc active
KK  1300             lda 204
CA  1310             sta ctemp
ON  1320             inc 204
KN  1330             lda 646
JI  1340             sta tcolor
HM  1350             sec
PO  1360             jsr $fff0
PD  1370             stx cur
GC  1380             sty cur+1
JL  1390             ldy #0
AP  1400 ;--- store screen and color memory ---
KE  1410 loop1       lda $0400,y
LL  1420             sta $a000,y
FK  1430             lda $d800,y
AG  1440             and #15
FO  1450             sta $a400,y
CJ  1460             lda $0500,y
AP  1470             sta $a100,y
KN  1480             lda $d900,y
CJ  1490             and #15
KB  1500             sta $a500,y
HM  1510             lda $0600,y
FC  1520             sta $a200,y
EC  1530             lda $da00,y
EM  1540             and #15
PE  1550             sta $a600,y
MP  1560             lda $0700,y
KF  1570             sta $a300,y
JF  1580             lda $db00,y
GP  1590             and #15
EI  1600             sta $a700,y
KC  1610             iny
OB  1620             bne loop1
BA  1630 ;--- now display the pop-ascii window ---
FL  1640             ldx #3
FF  1650             ldy #12
CP  1660             clc
FC  1670             jsr $fff0
NB  1680             lda #"{rvs}"
DD  1690             jsr $ffd2
```

```
LK  1700            lda #"{black}"
HE  1710            jsr $ffd2
ED  1720            lda #"{logo-y}"
NK  1730            ldx #16
DJ  1740 loop0      jsr $ffd2
NI  1750            dex
JK  1760            bne loop0
JD  1770            ldx #4
EN  1780            ldy #11
EH  1790            clc
HK  1800            jsr $fff0
AC  1810            lda #<prep
GI  1820            ldy #>prep
AH  1830            jsr $able
JB  1840            ldy #14
BN  1850            lda #"{shift-*}"
BB  1860 loop3      jsr $ffd2
JA  1870            dey
EC  1880            bne loop3
DM  1890            lda #"{logo-s}"
FA  1900            jsr $ffd2
PD  1910            jsr black
BN  1920            ldx #5
MD  1930 entry      ldy #11
KA  1940            clc
ND  1950            jsr $fff0
GH  1960            lda #"}"
LE  1970            jsr $ffd2
FK  1980            ldy #14
LP  1990            lda #" "
PJ  2000 loop4      jsr $ffd2
FJ  2010            dey
BL  2020            bne loop4
ML  2030            lda #"}"
BJ  2040            jsr $ffd2
LM  2050            jsr black
IO  2060            inx
GA  2070            cpx #21
HP  2080            bcc entry
KA  2090            ldy #11
KK  2100            clc
NN  2110            jsr $fff0
GK  2120            lda #"{logo-z}"
LO  2130            jsr $ffd2
DP  2140            lda #"{shift-*}"
PE  2150            ldy #14
BE  2160 loop5      jsr $ffd2
FD  2170            dey
CF  2180            bne loop5
MB  2190            lda #"{logo-x}"
BD  2200            jsr $ffd2
HJ  2210 ;--- now fill the window ---
DD  2220            lda #32
FL  2230            sta 2        ;--- initial char ---
NB  2240 again      lda #6
ME  2250            sta line
FK  2260            jsr place
LP  2270 more       lda 2
KK  2280            cmp #10
EC  2290            bcs notone
DO  2300            lda #"0"
PJ  2310            jsr $ffd2
BK  2320 notone     cmp #100
CP  2330            bcs none
LA  2340            lda #"0"
HM  2350            jsr $ffd2
PD  2360 none       ldx 2
NC  2370            lda #0
JJ  2380            jsr $bdcd    ;--- display number ---
IG  2390            jsr twospaces
DE  2400 ;--- now hex number ---
EA  2410            lda 2
BH  2420            and #$f0
HF  2430            lsr
BG  2440            lsr
LG  2450            lsr
FH  2460            lsr
MB  2470            clc
DH  2480            jsr dispnum
EF  2490            lda 2
HN  2500            and #$0f
BJ  2510            jsr dispnum
KO  2520            jsr twospaces
MH  2530            lda 2
IL  2540            cmp #32
AJ  2550            bcc special1
IK  2560            cmp #128
IE  2570            bcc normal
MK  2580            cmp #161
MJ  2590            bcs normal
NE  2600 special2   sec
FK  2610            sbc #128
HO  2620            sta temp
BP  2630            asl
GM  2640            clc
JL  2650            adc temp
DD  2660            tay
LL  2670            ldx #3
KP  2680 lo1        lda table2,y
LB  2690            jsr $ffd2
MG  2700            iny
NE  2710            dex
PJ  2720            bne lo1
OC  2730            jmp finish
CA  2740 special1   asl
ED  2750            clc
EF  2760            adc 2
BK  2770            tay
JC  2780            ldx #3
LG  2790 lo2        lda table1,y
JI  2800            jsr $ffd2
KN  2810            iny
LL  2820            dex
AB  2830            bne lo2
MJ  2840            jmp finish
HL  2850 normal     lda #32
FM  2860            jsr $ffd2
AN  2870            lda 2
JN  2880            jsr $ffd2
PH  2890            lda #" "
NO  2900            jsr $ffd2
KB  2910 finish     lda #13
BA  2920            jsr $ffd2
GN  2930            inc line
GD  2940            inc 2
HF  2950            jsr place
EN  2960            lda line
OF  2970            cmp #20
IJ  2980 bcs        waitkey
FO  2990            jmp more
JL  3000 ;--- wait for 'q' key ---
DN  3010 waitkey    jsr $ffe4
LJ  3020            cmp #"{down}"
LO  3030            beq forward
PC  3040            cmp #"{up}"
KD  3050            beq back
DI  3060            cmp #"q"
LN  3070            bne waitkey
GE  3080 ;--- now restore current screen ---
LK  3090            sei
HG  3100            ldy #0
PL  3110            lda 1
MF  3120            and #254
KJ  3130            sta 1        ;--- switch out basic rom ---
```

```
JI  3140 ;--- restore screen & colors ---
LD  3150 loop2     lda $a000,y
BH  3160           sta $0400,y
PF  3170           lda $a400,y
JL  3180           sta $d800,y
KG  3190           lda $a100,y
MJ  3200           sta $0500,y
KI  3210           lda $a500,y
EO  3220           sta $d900,y
FJ  3230           lda $a200,y
HM  3240           sta $0600,y
FL  3250           lda $a600,y
EC  3260           sta $da00,y
AM  3270           lda $a300,y
CP  3280           sta $0700,y
AO  3290           lda $a700,y
PE  3300           sta $db00,y
OM  3310           iny
DM  3320           bne loop2
EL  3330 ;--- restore basic & interrupts ---
FK  3340           lda 1
GP  3350           ora #1
HP  3360           sta 1
IL  3370           cli
LN  3380           ldx cur
CM  3390           ldy cur+1
OL  3400           clc
BP  3410           jsr $fff0
LG  3420           lda tcolor
ME  3430           sta 646
GB  3440           lda ctemp
OE  3450           sta 204
BP  3460           sta active
LJ  3470           lda #"{rvs off}"
BD  3480           jsr $ffd2
KC  3490           pla
AH  3500           rti
ND  3510 forward   jmp again
IP  3520           back lda 2
LE  3530           sec
LH  3540           sbc #28
GL  3550           sta 2
HK  3560           jmp again
BP  3570 twospaces lda #" "
FJ  3580           jsr $ffd2
PJ  3590           jsr $ffd2
MP  3600           rts
LK  3610 place     ldx line
KA  3620           ldy #13
EK  3630           clc
HN  3640           jsr $fff0
PM  3650           lda #"{rvs}"
FO  3660           jsr $ffd2
CE  3670           rts
IB  3680 dispnum   cmp #10
DG  3690           bcc numeric
KO  3700           clc
EA  3710           adc #55
BC  3720           jsr $ffd2
OH  3730           rts
PN  3740 numeric   clc
DD  3750           adc #48
JE  3760           jsr $ffd2
GK  3770           rts
ME  3780 black     lda #"{black}"
HG  3790           jsr $ffd2
NA  3800           lda #" "
LH  3810           jsr $ffd2
AC  3820           lda #"{cyan}"
PI  3830           jsr $ffd2
MO  3840           rts
CC  3850 prep      .asc "{rvs}{cyan}{logo-a}"
OE  3860           .byt 0
EH  3870 active = 821
FH  3880 cur    = 822
FE  3890 ctemp  = 824
FD  3900 line   = 825
LE  3910 temp   = 826
BD  3920 tcolor = 820
FB  3930 ;--- data for special characters ---
GF  3940 table1 .asc "              wht      disena"
KO  3950        .asc "        retlwr    dwnronhmedel      "
JP  3960        .asc "              redrhtgrnbluspc"
NC  3970 table2 .asc "   org          f1 f3 f5 f7 f2"
BL  3980        .asc " f4 f6 f8srtupp   blkcuprofclsinsbrnlrd"
BC  3990        .asc "gr1gr2lgrlblgr3purlftyelcynspc"
```

Listing 2: *"customizer"*

```
EF  10 rem ---     pop-ascii      ---
IG  20 rem --- customizing loader ---
MD  30 rem
JH  35 ifa=0thena=1:load"ml-popascii",8,1
IC  40 read bc:poke 49760,bc:poke 49766,bc
GI  50 read sc:poke 49750,sc:poke 49309,sc
JG  60 read ky:poke 49196,ky
KE  70 read dw:poke 49585,dw
KH  80 read up:poke 49589,up
MH  90 read qt:poke 49593,qt
ME  95 sys49152
CF  100 data 159:rem --- background color:cyan ---
NC  105 data 144:rem --- shadow color:black ---
FF  110 data 2:rem --- activation key:cbm ---
KC  120 data 17:rem --- forward scroll:cursor down ---
CD  130 data 145:rem --- backward scroll:cursor up ---
EO  140 data 81:rem --- quit key:q ---
```

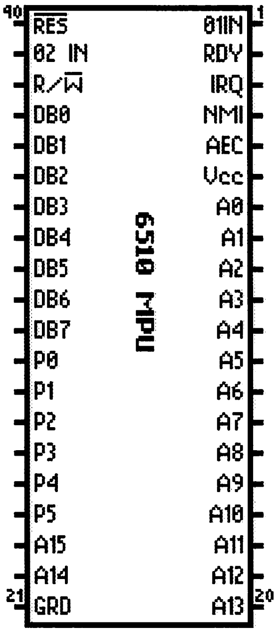
Listing 3: *BASIC generator for "ml-popascii"*

```
EL  100 rem  generator for "ml-popascii"
HK  110 n$="ml-popascii": rem  name of program
FD  120 nd=815: sa=49152: ch=83582

(for lines 130-260, see the standard generator on page 5)

NE  1000 data 169,  39, 141,  24,   3, 169, 192, 141
PC  1010 data  25,   3, 169,  21, 160, 192, 141,   2
CA  1020 data   3, 140,   3,   3,  96, 169,  39, 141
BC  1030 data  24,   3, 169, 192, 141,  25,   3, 169
HG  1040 data   0, 141,  53,   3,  76, 131, 164,  72
CB  1050 data 173, 141,   2, 201,   2, 240,   4, 104
HE  1060 data  76,  71, 254, 173,  53,   3, 208, 247
ME  1070 data 238,  53,   3, 165, 204, 141,  56,   3
KP  1080 data 230, 204, 173, 134,   2, 141,  52,   3
CB  1090 data  56,  32, 240, 255, 142,  54,   3, 140
MF  1100 data  55,   3, 160,   0, 185,   0,   4, 153
KE  1110 data   0, 160, 185,   0, 216,  41,  15, 153
OF  1120 data   0, 164, 185,   0,   5, 153,   0, 161
CH  1130 data 185,   0, 217,  41,  15, 153,   0, 165
NH  1140 data 185,   0,   6, 153,   0, 162, 185,   0
AK  1150 data 218,  41,  15, 153,   0, 166, 185,   0
KA  1160 data   7, 153,   0, 163, 185,   0, 219,  41
JA  1170 data  15, 153,   0, 167, 200, 208, 197, 162
JJ  1180 data   3, 160,  12,  24,  32, 240, 255, 169
HA  1190 data  18,  32, 210, 255, 169, 144,  32, 210
OO  1200 data 255, 169, 183, 162,  16,  32, 210, 255
JK  1210 data 202, 208, 250, 162,   4, 160,  11,  24
BD  1220 data  32, 240, 255, 169, 101, 160, 194,  32
DE  1230 data  30, 171, 160,  14, 169, 192,  32, 210
FN  1240 data 255, 136, 208, 250, 169, 174,  32, 210
EN  1250 data 255,  32,  85, 194, 162,   5, 160,  11
KD  1260 data  24,  32, 240, 255, 169, 221,  32, 210
EJ  1270 data 255, 160,  14, 169,  32,  32, 210, 255
```

```
FP  1280 data 136, 208, 250, 169, 221,  32, 210, 255
FG  1290 data  32,  85, 194, 232, 224,  21, 144, 222
HH  1300 data 160,  11,  24,  32, 240, 255, 169, 173
BO  1310 data  32, 210, 255, 169, 192, 160,  14,  32
NC  1320 data 210, 255, 136, 208, 250, 169, 189,  32
ND  1330 data 210, 255, 169,  32, 133,   2, 169,   6
JB  1340 data 141,  57,   3,  32,  52, 194, 165,   2
HD  1350 data 201,  10, 176,   5, 169,  48,  32, 210
BH  1360 data 255, 201, 100, 176,   5, 169,  48,  32
NE  1370 data 210, 255, 166,   2, 169,   0,  32, 205
NF  1380 data 189,  32,  43, 194, 165,   2,  41, 240
FC  1390 data  74,  74,  74,  74,  24,  32,  67, 194
LP  1400 data 165,   2,  41,  15,  32,  67, 194,  32
BK  1410 data  43, 194, 165,   2, 201,  32, 144,  35
KM  1420 data 201, 128, 144,  51, 201, 161, 176,  47
CP  1430 data  56, 233, 128, 141,  58,   3,  10,  24
FL  1440 data 109,  58,   3, 168, 162,   3, 185, 204
GM  1450 data 194,  32, 210, 255, 200, 202, 208, 246
HM  1460 data  76, 150, 193,  10,  24, 101,   2, 168
AC  1470 data 162,   3, 185, 105, 194,  32, 210, 255
JO  1480 data 200, 202, 208, 246,  76, 150, 193, 169
KI  1490 data  32,  32, 210, 255, 165,   2,  32, 210
MF  1500 data 255, 169,  32,  32, 210, 255, 169,  13
CD  1510 data  32, 210, 255, 238,  57,   3, 230,   2
BA  1520 data  32,  52, 194, 173,  57,   3, 201,  20
LC  1530 data 176,   3,  76,  22, 193,  32, 228, 255
KP  1540 data 201,  17, 240, 106, 201, 145, 240, 105
HJ  1550 data 201,  81, 208, 241, 120, 160,   0, 165
NH  1560 data   1,  41, 254, 133,   1, 185,   0, 160
DC  1570 data 153,   0,   4, 185,   0, 164, 153,   0
FD  1580 data 216, 185,   0, 161, 153,   0,   5, 185
AD  1590 data   0, 165, 153,   0, 217, 185,   0, 162
JE  1600 data 153,   0,   6, 185,   0, 166, 153,   0
BG  1610 data 218, 185,   0, 163, 153,   0,   7, 185
CD  1620 data   0, 167, 153,   0, 219, 200, 208, 205
HG  1630 data 165,   1,   9,   1, 133,   1,  88, 174
KI  1640 data  54,   3, 172,  55,   3,  24,  32, 240
OG  1650 data 255, 173,  52,   3, 141, 134,   2, 173
CK  1660 data  56,   3, 133, 204, 141,  53,   3, 169
FL  1670 data 146,  32, 210, 255, 104,  64,  76,  14
IL  1680 data 193, 165,   2,  56, 233,  28, 133,   2
NM  1690 data  76,  14, 193, 169,  32,  32, 210, 255
PK  1700 data  32, 210, 255,  96, 174,  57,   3, 160
KH  1710 data  13,  24,  32, 240, 255, 169,  18,  32
GK  1720 data 210, 255,  96, 201,  10, 144,   7,  24
HL  1730 data 105,  55,  32, 210, 255,  96,  24, 105
AO  1740 data  48,  32, 210, 255,  96, 169, 144,  32
BP  1750 data 210, 255, 169,  32,  32, 210, 255, 169
AJ  1760 data 159,  32, 210, 255,  96,  18, 159, 176
MK  1770 data   0,  32,  32,  32,  32,  32,  32,  32
GP  1780 data  32,  32,  32,  32,  32,  32,  32,  32
ED  1790 data  87,  72,  84,  32,  32,  32,  32,  32
FI  1800 data  32,  68,  73,  83,  69,  78,  65,  32
EB  1810 data  32,  32,  32,  32,  32,  32,  32,  32
CJ  1820 data  82,  69,  84,  76,  87,  82,  32,  32
GI  1830 data  32,  32,  32,  32,  68,  87,  78,  82
PO  1840 data  79,  78,  72,  77,  69,  68,  69,  76
MD  1850 data  32,  32,  32,  32,  32,  32,  32,  32
GE  1860 data  32,  32,  32,  32,  32,  32,  32,  32
NH  1870 data  32,  32,  32,  32,  32,  82,  69,  68
BN  1880 data  82,  72,  84,  71,  82,  78,  66,  76
FL  1890 data  85,  83,  80,  67,  32,  32,  32,  79
CI  1900 data  82,  71,  32,  32,  32,  32,  32,  32
BK  1910 data  32,  32,  32,  32,  70,  49,  32,  70
HJ  1920 data  51,  32,  70,  53,  32,  70,  55,  32
MK  1930 data  70,  50,  32,  70,  52,  32,  70,  54
DP  1940 data  32,  70,  56,  83,  82,  84,  85,  80
PO  1950 data  80,  32,  32,  32,  66,  76,  75,  67
PB  1960 data  85,  80,  82,  79,  70,  67,  76,  83
OD  1970 data  73,  78,  83,  66,  82,  78,  76,  82
IB  1980 data  68,  71,  82,  49,  71,  82,  50,  76
DC  1990 data  71,  82,  76,  66,  76,  71,  82,  51
PD  2000 data  80,  85,  82,  76,  70,  84,  89,  69
IN  2010 data  76,  67,  89,  78,  83,  80,  67
```

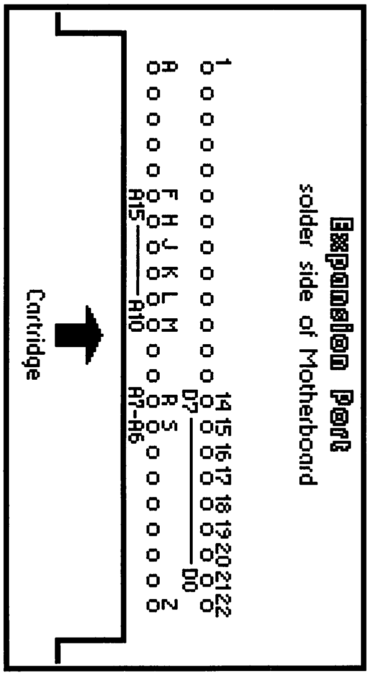
Pinout Diagram for 6510 MPU

(viewed from solder side of motherboard)



Expansion Port Pin Positions

(viewed from solder side of motherboard)



Combiner

A utility for geoWrite files

by Nick Vrtis

Combiner is a program which I wrote to take multiple *geoWrite* files and combine them into a single file. It comes in very handy when you have a number of separate documents and want to combine them so that you can edit and paginate the whole thing. GEOS can be pretty slow if you are working without a RAM expander on a large document, so I found it quicker and easier to work with smaller files. This way I could key, spell check, etc. each file as a small piece, then combine them for final preparation. It also came in handy when someone would write an article for our newsletter and I had to combine that article with the rest of the articles.

When you double click on *Combiner*, you are presented with the Main Menu screen. This Main Menu has four items to select from:

GEOS - This item is a pretty standard GEOS menu item. You can run any Desk Accessories which are on the same disk that *Combiner* was loaded from. You can also get information about *Combiner.*

Done - Select this item when you are finished combining documents. It has two submenu items: **Quit** will quit *Combiner*, and return you to the DeskTop and **geoWrite** will load *geoWrite* and let you edit the last output document you created (as if you had double-clicked on the document icon from the DeskTop). Note that in order for this to work properly, *geoWrite* must be on one of the disks currently in the active drives.

Help - This item is a short series of screens which covers the basic operation of *Combiner*, just in case you have forgotten something and don't have this documentation handy.

Begin - This item is selected to start the process of combining an input file with an output file. After you have combined an input with an output, you will be returned to the Main Menu screen where you can select *Begin* again to repeat the process (either with the same output file, or a new one). If you select *Cancel* from any of the windows in the process, you will be returned to the Main Menu screen.

The first thing that happens after you select *Begin* is that you are presented with a series of windows to identify the output document. The first window gives you a choice to *Create* an new output file, or to *Open* an existing file and add to it. *Create* will ask for the name of the new output file. If that file already exists on the disk, you will be asked to confirm that you want to delete the old version. If you select Open an existing file, you will be presented with the standard GEOS scrolling filename window showing all the *geoWrite* documents on the disk. Highlight the output file you want, and click over the *Open* box.

Once the output file has been determined, the input file needs to be selected. This is done through a standard scrolling filename window (the same way *geoWrite* lets you select an existing file). Click to highlight the name of the file you want to use as input, and click the *Open* box to use that file.

The last thing *Combiner* needs to know is where to put the input in the output file (in terms of pages). *Combiner* can pick off any number of pages from the input file (it doesn't need to be the whole document). It can also put those pages after any page of the output file (or insert them at the beginning).

In order to get this information, *Combiner* puts up a window with five boxes. The first box requests the first page of the input document you want included in the output. The second box requests the last page of the input document you want included. Both the first page and the last page are included, so in order to select a single page, put the same number in both the first page box and the last page box. The third box requests the page number of the output document you want the new pages placed *after.* All the pages specified by the first page number and last page number will be placed after this page of the output.

For example, if you wanted to put the first two pages of an input document after the second page of an existing document, 'first page' would be 1, 'last page' would be 2, and 'after page' would be 2. Page 1 of the input would become page 3 of the output. To insert the input pages at the beginning (before page 1) of the output, use a value of 0 for the 'after page'.

When this window is first opened, the three numbers are initialized to select the whole input document and insert it at the end of the output document. If you are creating a new output document, the 'after page' shows up as page 00. To get from one number box to the next, click in the box you want to go to. You don't have to fill in the boxes in order. You can exit the window by one of three ways. Hitting Return or clicking over the *OK* box will close the window and use the numbers currently in the boxes. Clicking over the *Cancel* box returns you to the Main Menu without processing any input.

There are currently three different document versions produced by various *geoWrite* versions. Version 1.x is from the *geoWrite* shipped with the original GEOS (version 1.0, 1.1, 1.2, 1.3). Version 2.0 is from *Writer's Workshop*, and version 2.1 is from the *Writer's Workshop* upgrade or GEOS version 2.0. *Combiner* will combine different versions of *geoWrite* documents. When you are creating a new document its version is determined by the version of the first input file. Since *Combiner* will combine a version 2.1 document with a version 1.3 and produce a version 1.3 output file, it is conveniently allows owners of 1.x versions of *geoWrite* to edit files that have been produced originally by 2.x versions. You should be aware that there are features within version 2.1 (and 2.0) which are unavailable with version 1.3. *Combiner* drops the unsupported features when combining a higher version file into a lower version.

Any graphics which are referenced in the pages selected from an input document will be copied along with those pages. *Combiner* doesn't bother copying any graphics not referenced by the pages selected.

Version 2 of *geoWrite* allows for a header and footer page. *Combiner* will not select headers or footers from the input as there can only be one set of headers and footers per document. They are not removed from the output document, so if you need the headers and footers, use the DeskTop *duplicate* option to make a copy of the file with the header and footer and use it as the first output file.

Combiner will handle multiple drives if they are present on the system. You can also have the input and output documents on different disks (even with a single drive system). *Combiner* will ask you to insert the necessary disks as they are needed. *Combiner* reads as much of the input document into memory as it can and then writes it out in order to keep disk swapping to a minimum in a single drive system. Desk Accessories are always loaded from the disk that *Combiner* was on when it was loaded from the DeskTop.

Combiner has been tested under versions 1.3 and 2.0 of GEOS, and version 1.4 of GEOS 128. *[In tests here at the Transactor offices, it was necessary to exit to a 40-column version of geoWrite with GEOS 128 v2.0 - MO]*

Programming considerations

In addition to being a useful program to have around, *Combiner* contains a number of examples of how to make use of various features within GEOS. In addition to the normal menus and windows, *Combiner* will run Desk Accessories, handles multiple drives, custom click boxes in a window, and multiple input fields in one window. I've tried to keep the source code well commented, so I will only present an overview in this article. The routine labels I've used are those from Alexander Boyce's GEOS programmer's reference (except for the general purpose page zero locations). *[The BSW labels and hex addresses are provided in square brackets following the first usage of each Boyce label.-MO]*

I'll start with how the *geos* portion of the main menu is setup and handled. What really happens is that the *geos* item is set up as a submenu from the main menu. When the source was coded, the submenu was set up to handle all nine possible items (eight Desk Accessories plus the *info* item). Then as part of the initialization process, *Combiner* uses TABLE *[FindFTypes*, $c23b] to get a list of the names of Desk Accessories.

TABLE returns a list of names which are 17 bytes apart, and zero-terminated, so the addresses don't need to be changed. All that needs to be adjusted in the submenu entry is the number of items in the list, and the height of the menu. The number of entries left is returned by TABLE, so a simple subtract gives the number used. Each entry takes 14 pixels, so the height can be calculated easily. Note that one of the reasons for the *info* option is so that the submenu under *geos* always has at least one entry.

Running desk accessories

Running a Desk Accessory from within a program is really pretty simple. All you need to do is point LOAD *[GetFile*, $c208] to the name you want, and GEOS takes care of saving and restoring the piece of your program which is going to be overlayed. If your screen is not complicated, tell the Desk Accessory not to bother saving and restoring the screen. This can save some disk I/O if the Desk Accessory has to create a temporary file to save the screen.

Most of the GEOS environment is preserved during the running of the Desk Accessory. Unfortunately, Berkeley has never published much about what a Desk Accessory can and cannot trash, so I would be a little careful. Obviously, the general purpose registers (**r0** - **r15**) are not preserved, nor are the disk buffers. Strangely, I haven't found anything in GEOS which indicates an open or a closed VLIR file, so I would not count on this being preserved between calls to a Desk Accessory. If you think about it, these restrictions aren't too bad. Since your program controls when the menu entry is active, just make sure that you aren't in the middle of some complicated update when you activate it.

You might be interested in looking at how the *geoWrite* option of *Done* is implemented. This is an example of how to transfer from one program to another as if it had been double-clicked on from the DeskTop. This avoids the hassle of reloading the DeskTop just to get to *geoWrite* to clean up the file you just created.

Text windows and custom click boxes

There was a little challenge in doing the *Help* windows. The way text is implemented in windows, each line has to be a separate entry in the window definition. This is because the WINDOW routine [*DoDlgBox*, $c256] sets the left margin to zero (instead of the left edge of the window) so a carriage return in the text takes you outside the window. Having each line as a separate entry either means a separate window definition for each help screen, or adjusting a lot of pointers for each line.

I didn't care for either alternative. So I cheated and used the 'set next character position' function code ($16) within the text. The three bytes after this code specify the absolute *x* and *y* coordinates where the following text is to be displayed. The *Help* window definition has only one text pointer (**r12**). This points to one long text string which contains positioning commands to format the text correctly. The bad part of doing windows this way is that if you move or resize the window, you have to go back and recalculate all the positioning.

The *Begin* main menu entry really starts the combining process. The code in this section is pretty straightforward. The way the *Drive* option is implemented is that windows which may or may not have a *Drive* box on them have been designed with that box definition at the end. Unlike menus which start with a count of the number of entries, windows end when they have a function byte of zero. So, by putting the *Drive* box last, it can be included in the window by making the function byte equal to $12 (custom click box), or removed from the window by making it equal to zero.

Combiner checks NUMDRV during initialization, and sets the function bytes appropriately for those windows which may or may not have a *Drive* box. This way, the code which processes the windows doesn't have to worry about whether there are enough drives. The only way that code will be executed if there was a *Drive* box in the window, and there will only be a *Drive* box if there is more than one drive on the system.

Non-standard windows

Probably the hardest part of this program was doing the window asking for the starting page, the ending page, and the page to put them after. It is definitely a non-standard window! The way I wanted to implement it was to have all three numbers on one window, and let the user click on any of the numbers to get into that box and change it. This is similar to the way *geoWrite* implements the search and replace function (in fact that's where I got the idea). There were challenges though.

The trick is displaying *text* within a click box. The window processor doesn't do *any* click boxes until the very end of processing (regardless of where they appear in the window definition). So what I had to do was put up and display *all* the click boxes for the window from within a routine (labeled WHERESET) which gets called after the window is drawn. You have to do all click boxes at once, because GEOS can only handle one set at a time. Any boxes specified in the window definition will replace those you have defined in the setup routine. I wanted to show the default values for each of the page numbers required within a box. But, when CBOXES [*DoIcons*, $c15a] draws a box, it overlays anything under the box area (it makes sense when you think about how GEOS draws graphics). The first thing WHERESET does is put up the click boxes. Then it displays the default values for the pages within the areas where the boxes are by calling WHEREIN.

Handling user text input

In order to start the process, one text input function needs to be defined in the window definition. You don't want to define all three, since INPUT [*GetString*, $c1ba] saves a copy of the carriage return entered vector and replaces it with its own value. If you call INPUT twice, the second call saves the wrong carriage return vector and when you hit return INPUT gets in a loop.

Each page value has a control block associated with it. This block keeps the values which are needed to switch between one input area and the other (the input buffer, the number of characters entered, and the text cursor column). When the user clicks over one of the value boxes, two routines get called. SVWHERE takes the necessary INPUT values and saves them in the current control block (the current block index is saved in WIDXSAVE). Then NXTWHERE is called to move the new values into the areas where INPUT uses, and finally calls PROMPTON to move the text cursor. As far as INPUT is concerned, nothing ever happened.

These routines are not totally general purpose, since there is a lot in common with each area (size, starting column, number of characters, etc.) The whole window is closed by one of three actions: 1) entering a Return in any of the input windows, 2) clicking on the OK box, or 3) clicking on the *Cancel* box. *Cancel* takes you back to the main menu while the other two fall through and process the input from the window. One final bit of cleanup needs to be done before error-checking the user's input. When INPUT is accepting characters, it just keeps track of where the next one will go, but doesn't put the zero at the end of the string until the Return is entered. This works fine if you have only one input area; but with more it is possible to have shortened a field and then moved to another field. The first field would not be properly zero-terminated but the count of the valid characters has been saved, so *Combiner* makes sure all three fields are properly terminated before checking the values.

That is really the end of the major programming challenges (due to the way GEOS works) that I encountered while doing *Combiner*. There are a number of other details which needed to be taken care of. Rulers need special attention when converting between various revisions of *geoWrite* (a ruler is the set of codes which define the margins, tabs, etc.). Version 1.x (any of the original *geoWrite* versions) only has one ruler per page, and they are shorter than version 2 rulers. So whenever *Combiner* goes from version 2 to version 1, it has to shorten the ruler at the start of the page, and discard any rulers found within the page (version 2 allows rulers at the start of any paragraph). Likewise, when going from version 1 to version 2, *Combiner* has to lengthen the ruler (and set up a default value for the paragraph indent).

There are even some conversions which must be done when going between the two different version 2 documents. Version 2.0 was delivered with the original *Writer's Workshop*, and is the default version created by the new C64 versions of *geoWrite*. The difference between version 2.0 and version 2.1 is that version 2.1 margins can go from 0.2 inches to 8.2 inches, and version 2.0 (and version 1.x) goes from 1.2 inches to 7.2 inches. This means that when going between these two versions, the values for the margins and tabs need to be adjusted.

A value of zero for the left margin amounts to the 1.2 inches mark on the page for version 2.0 documents, and 0.2 inches for version 2.1. This is a pretty easy conversion, as all early version tabs are available in 2.1 rulers. But, a tab (or margin setting) of less than 1.2 inches (or greater than 7.2 inches) is meaningless (and impossible) in a 2.0 or 1.x document, so *Combiner* has to correct for these. The only 'gotcha' is that the 7.2 inch value used in versions 1.x & 2.0 is one less than the pixel value needed by 2.1. You will notice an odd check for this in the program.

Combining graphics

Another minor adjustment which needs to be checked for when combining documents is the way graphics are referenced. In GEOS, graphics are not imbedded directly in the document. They are indicated by a graphics escape character, the size of the graphic, and the VLIR record number where the graphic is actually stored. In order to combine documents properly, any graphic escapes found in the input pages need to have the VLIR record number adjusted because graphics from the original document may already be occupying that record.

You also need to keep track of what graphics are actually used, since it would be possible for the user to select pages from the input which don't use all the graphics. The way *Combiner* does this is to have a 64-byte table (PICSUSED). This allows one byte for each possible graphic VLIR. This gets initialized to zeros. OLDPICS starts with the first free VLIR record from the original file. When a page gets read in, it is scanned for the graphics escape character ($10). When one is found, the input VLIR record is saved in the next available table slot (pointed to by OLDPICS). The record number is then changed to the value of OLDPICS, as that is where it will go in the output file. Once all the input pages have been combined, PICSUSED is scanned for any non-zero values. That VLIR record from the input file is read in, and gets written out as a new output VLIR record.

Dealing with end of text markers

One other 'adjustment' needs to be made to text in the process of combining pages. When *geoWrite* stores a document, the last byte on the last page is a zero to indicate end of text. When *Combiner* needs to insert a new page after the last page of the output file, it needs to read the last sector of the last page, and replace the zero with a end of page character (the ASCII form feed - $0C). For the same reasons, if *Combiner* has to insert what was the last page of an input file in between pages of the output file, the zero at the end of the input page needs to be replaced with the form feed.

Maximizing available RAM

Combiner reads in as many pages of input into RAM as it can before writing that data out. This reduces the number of disk changes which occur in a single drive system when the input and output disks aren't the same. In order to increase the size of available RAM for the input buffer, *Combiner* doesn't use the background screen which normally goes from $6000 to $8000.

Whenever GEOS closes a window (or menu) it calls the routine pointed to by IRECVR, with the coordinates of the area to be recovered. Normally, IRECVR points to the routine which transfers a box from the background screen to the foreground.

Combiner changes this vector to REPAT. The screen for *Combiner* is relatively simple, and except for the title area on the bottom, it is simply a pattern fill. So in most cases, REPAT simply sets the pattern to the one *Combiner* used in the title screen and calls PFILL [*Rectangle*, $c124] to replace the pattern.

The only exception is for the *Help* windows which extend into the title area. Whenever this happens, REPAT has to redraw the title area as part of the window recovery. REPAT actually gets called twice when a window is closed. Once for the border, and once for the main window. The position of the *Help* window and the value that REPAT checks for were carefully chosen so that it recovers the title area only once.

The method of buffering deserves some explanation. *Combiner* uses all the RAM from the end of the program to $8000 (the start of GEOS storage) for a big buffer. Each time a page gets read in, the first two available bytes are reserved as a pointer area, and then LCHAIN [*ReadFile*, $c1ff] is called in an attempt to load the VLIR record into the area available. After the page is processed (remember, the size might have changed if going between version 1 and version 2 type documents), the ending address is saved in the pointer field at the start of each page.

The process is then repeated until either LCHAIN says a record would not fit, or we run out of input text pages. In either case, a zero is stored at the end of the last record that has been read in successfully (any record truncated by memory limitations is ignored - it will get picked up in the next pass).

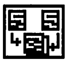
Combiner then begins at the start of the buffer area, and writes out pages until it sees the zero pointer. In order to write a page, you need to know the starting address, and the number of bytes. The starting address of the data portion is two bytes past the current address pointer (RA2), and the length is the difference between that and the ending address from the pointer area.

After the VLIR record is written, the old ending address becomes the new current address and the process is repeated. This continues until *Combiner* finds the zeros as the new current address. This signifies the end of the data in memory. All that needs to be done is a quick check to see if all the requested pages have been processed in order to determine if we are done. Handling the graphics VLIR records is done the same way. The only difference is that the PICSUSED table needs to be checked to determine which graphics pages need to be read in, and to which record they need to be written.

That pretty much covers the high points of the *Combiner* code. The details are commented within the code. I had to break the source into two separate files and an include file because *geoAssembler* couldn't handle it all in one piece.

$HCOMBINER - *Header file for Combiner*

```
; $HCOMBINER - Combine multiple geoWrite files into one - N. Vrtis 1/89
; Header definition file
;
   .header
   .word 0          ;first two always zero
   .byte 3          ;size of ICON always fixed
   .byte 21
```



```
   .byte $80+3      ;CBM filetype is USR
   .byte 6          ;GEOS filetype is Application
   .byte 0          ;GEOS file structure is sequential
   .word start      ;where to load program
   .word patch+30   ;ending address
   .word start      ;begin execution @ load address
;
   .byte "COMBINER   V1.1",0,0,0,$00    ;(40 column only)
   .byte "Nicholas J. Vrtis",0,0,0
   .block   160-117                      ;unused in header
;
   .byte "Combine multiple geoWrite files into a single file.",$0d
   .byte "Nicholas J. Vrtis - 1989",0
   .endh
; end of $HCOMBINER
```

/HCOMBINER - *Include file for Combiner*

```
; /COMBINER - Combine multiple geoWrite documents into one.  - Nick Vrtis 1/89
; include file used to define Page zero locations & GEOS routines
;
r0      ==      $02
r1      ==      $04
r2      ==      $06
r3      ==      $08
r4      ==      $0a
r5      ==      $0c
r6      ==      $0e
r7      ==      $10
r8      ==      $12
r9      ==      $14
r10     ==      $16
r11     ==      $18
r12     ==      $1a
;
string ==       $24     ;input string pointer
pline  ==       $26
scnflg ==       $2f     ;forground/background flag
mousex ==       $3a
mousey ==       $3c
;
ra2     ==      $70     ;pointer to start of a buffer area
ra3     ==      $72     ;pointer to data part of buffer area (ra2+2)
ra0     ==      $fb
ra1     ==      $fe
;
.macro  ldptr   p,pz
        lda     #[(p)
        sta     pz
        lda     #](p)
        sta     pz+1
.endm
.macro  window  p
        ldx     #[(p)
        ldy     #](p)
        jsr     xywindow
.endm
.macro  movew   src,dst
        lda     src+1
        sta     dst+1
        lda     src
        sta     dst
.endm
;
buf0        ==      $8000 ; 1st disk buffer
buf1        ==      $8100 ; 2nd disk buffer
buf2        ==      $8200 ; 3rd disk buffer
tsbuf       ==      $8300 ; Track/Sector buffer
direntry    ==      $8400 ; directory entry after open
dfname      ==      $8442 ; double clicked filename
ddname      ==      $8453 ; double clicked disk name
curdrv      ==      $8489 ; current drive number
numdrv      ==      $848d ; number of drives in system
irecvr      ==      $84b1 ; screen recovery vector
cursx       ==      $84be ; used by INPUT to store info
cursy       ==      $84c0 ;  " "
wincmd      ==      $851d ; command from window close
inplen      ==      $87cf ; used by INPUT to store info
;
pfill       ==      $c124 ; Rectangle        - pattern fill an area
setpat      ==      $c139 ; SetPattern       - set display pattern
dsptxt      ==      $c148 ; UseSystemFont    - display text
menu        ==      $c151 ; DoMenu           - menu processor
eramns      ==      $c157 ; RecoverAllMenus  - erase all menus
cboxes      ==      $c15a ; DoIcons          - draw click boxes
movedata    ==      $c17e ; MoveData         - move a block of data
drwmnu      ==      $c193 ; ReDoMenu         - redraw menu
grphc2      ==      $c1a8 ; i_GraphicsString - inline graphic commands
input       ==      $c1ba ; GetString        - get text input
cmenus      ==      $c1bd ; GoToFirstMenu    - close all menus
read        ==      $c1e4 ; GetBlock         - read in a sector
write       ==      $c1e7 ; PutBlock         - write a T/S
save        ==      $c1ed ; SaveFile         - save data to file
lchain      ==      $c1ff ; ReadFile         - load a file chain
follow      ==      $c205 ; FollowChain      - follow disk chain
load        ==      $c208 ; GetFile          - load desk accessory
lookup      ==      $c20b ; FindFile         - find file entry
dsetup      ==      $c214 ; EnterTurbo       - disk setup
```

```
read2    =     $c21a ; ReadBlock       - disk read
restrt   =     $c22c ; EnterDeskTop    - reload desktop
clrrdy   =     $c232 ; ExitTurbo       - stop turbo code
delete   =     $c238 ; DeleteFile      - delete a file
table    =     $c23b ; FindFTypes      - create file list table
window   =     $c256 ; DoDlgBox        - process window commands
opnser   =     $c25c ; InitForIO       - open channel to disk
clsser   =     $c25f ; DoneWithIO      - done with i/o
vopen    =     $c274 ; OpenRecordFile  - open vlir file
vclose   =     $c277 ; CloseRecordFile - close vlir file
vgoto    =     $c280 ; PointRecord     - set vlir chain #
vappend  =     $c289 ; AppendRecord    - add record to vlir
vsave    =     $c28f ; WriteRecord     - save data to vlir record
prompton =     $c29b ; PromptOn        - turn text prompt on (and position)
opndsk   =     $c2a1 ; OpenDisk        - open disk in drive
drvset   =     $c2b0 ; SetDevice       - set drive #
drvnam   =     $c298 ; GetPtrCurDkNm   - get disk name
clswin   =     $c2bf ; RstrFrmDialog   - close window
;
; end of /COMBINER
```

$1COMBINER - *First source file*

```
; $1COMBINER - Combine multiple geoWrite files into one.
; Nicholas J. Vrtis
; 5863 Pinetree S.E
; Kentwood, MI 49508
;
.include /COMBINER              ;Page zero & GEOS definitions
;
   .psect
start:  lda    #$80           ;I will be using the background screen
        sta    scnflg         ;so tell GEOS not to use it
        ldptr  repat,irecvr   ;set my vector to recover
        jsr    opentitle      ;do opening credits
;
        lda    curdrv         ;save D.A. disk drive
        sta    dadskdrv
        sta    odskdrv        ;also as inital output and input drive
        sta    idskdrv
        ldx    #[r0
        jsr    drvnam
        ldy    #15            ;save D.A. disk name
100$:   lda    (r0),y
        sta    dadsknm,y
        dey
        bpl    100$
        ldptr  dan1,r6        ;find D.A.'S
        lda    #5             ;looking for D.A.'S
        sta    r7
        lda    #8             ;up to 8
        sta    r7+1
        lda    #0             ;no class
        sta    r10
        sta    r10+1
        jsr    table
        sec                   ;calc how many found
        lda    #9             ;8 D.A.'s + 1 for INFO BOX
        sbc    r7+1
        tax                   ;save cnt
        ora    #$80           ;add vertical menu option bit
        sta    gmopts
        lda    #0             ;calc menu hgt
calcgmh: clc                  ;14 for each (+14)
        adc    #14
        dex
        bpl    calcgmh
        sta    gmhgt
;
        lda    numdrv         ;check if 2 drives available
        cmp    #2
        bcc    remenu         ;..drive not available
        lda    #$12           ;else enable -drive- click boxes
        sta    drvopt1
        sta    drvopt3
;
; Here is the common point to put the main menu back up
;
remenu: ldptr  mainmenu,r0    ;do mainmenu
        lda    #3             ;on 'HELP'
        jmp    menu
;
dohelp: ldx    #0
helplp: stx    helpidx
        lda    helpptrs,x
        sta    r12
        lda    helpptrs+1,x
        sta    r12+1
        window helpw
        lda    r0             ;chk option
        cmp    #1
        bne    helprtn        ;..not -ok-
        ldx    helpidx
        inx                   ;nxt help panel
        inx
        cpx    #[(maxhelp-helpptrs)
        bcc    helplp         ;..more to show
helprtn: jmp   drwmnu         ;redraw the menu
;
doinfo: window infow
        jmp    drwmnu         ;redraw the menu
;
dogeowrite:
        ldptr  ddname,r2      ;setup pointers to name/disk of output
        ldptr  dfname,r3
        ldy    #16
115$:   lda    odsknm,y
        sta    (r2),y
        lda    outnm,y
        sta    (r3),y
        dey
        bpl    115$
        ldptr  geowrite,r6    ;load GEOWRITE program
        lda    #$80           ;say 'double clicked'
        sta    r0
        jsr    load
        jsr    error          ;just in case load had an error
        jmp    drwmnu
;
doda1:  lda    #dan1-dan1     ;offset to start of name
   .byte $2c
doda2:  lda    #dan2-dan1
   .byte $2c
doda3:  lda    #dan3-dan1
   .byte $2c
doda4:  lda    #dan4-dan1
   .byte $2c
doda5:  lda    #dan5-dan1
   .byte $2c
doda6:  lda    #dan6-dan1
   .byte $2c
doda7:  lda    #dan7-dan1
   .byte $2c
doda8:  lda    #dan8-dan1
        sta    ra0            ;save offset
        jsr    eramns         ;clear off menu
        lda    dadskdrv       ;make sure D.A.'S available
        ldx    #[dadsknm
        ldy    #]dadsknm
        jsr    chkdsk
        beq    dodaexit       ;didn't want to mount D.A. disk
        clc                   ;calc adr of start of name
        lda    ra0
        adc    #[dan1
        sta    r6
        lda    #]dan1
        adc    #0
        sta    r6+1
        lda    #0
        sta    r0             ;no options
        sta    r10            ;I will restore screen/color
        jsr    load
        jsr    opentitle      ;restore screen
dodaexit:
        jmp    remenu         ;go put menu back up
;
helpw:  .byte $01
        .byte 18,174
        .word 4
        .word 305
        .byte $0c,2,15,r12
        .byte $01,2,135       ;-ok-
        .byte $02,31,135      ;-cancel-
        .byte 0
help1:  .byte $18,"COMBINER is a program to combine multiple geoWrite"
        .byte $16,6,0,43
        .byte "documents into one."
        .byte $16,6,0,63
        .byte "The MainMenu options are:"
        .byte $16,6,0,73
        .byte $12,"GEOS",$13," - Lets you run any Desk Accessories on the"
        .byte $16,6,0,83
        .byte "disk COMBINER was loaded from."
        .byte $16,6,0,93
        .byte $12,"Done",$13," - QUIT and return to the DeskTop, or go to"
        .byte $16,6,0,103
        .byte "geoWrite and edit the last output document."
        .byte $16,6,0,113
        .byte $12,"Begin",$13," - Start the process of combining documents"
        .byte $16,6,0,123
        .byte "(more information to follow)."
        .byte $16,6,0,133
        .byte $12,"Help",$13," - This Help series of screens.",$1b,0
help2:  .byte $18,"After you select BEGIN, you will be presented with a"
        .byte $16,6,0,43
        .byte "window which allows you to:"
        .byte $16,6,0,53
        .byte $12,"CREATE",$13," a new geoWrite output document."
        .byte $16,6,0,63
        .byte "(a follow on window will ask for the document name)."
        .byte $16,6,0,73
        .byte $12,"OPEN",$13," an existing geoWrite document for output."
        .byte $16,6,0,83
        .byte "(a follow on window will present you with the"
        .byte $16,6,0,93
        .byte "standard filename selection window)"
        .byte $16,6,0,103
        .byte $12,"CANCEL",$13," and return to the MainMenu.",$1b,0
help3:  .byte $18,"Once the input and output files have been identified,"
        .byte $16,6,0,43
        .byte "you need to tell COMBINER how many pages of the"
        .byte $16,6,0,53
        .byte "input document you want, and where to put them in"
        .byte $16,6,0,63
        .byte "in the output document.  A window allows you to"
        .byte $16,6,0,73
        .byte "specify the starting and ending pages (inclusive) to"
        .byte $16,6,0,83
        .byte "take from the input, and the page number to place"
        .byte $16,6,0,93
        .byte "those pages AFTER.  Click over the number to move"
        .byte $16,6,0,103
        .byte "the cusor to that value and change it.",$1b,0
help4:  .byte $18,"COMBINER will combine different versions of geoWrite"
        .byte $16,6,0,43
        .byte "documents.  When you create a new document, the"
        .byte $16,6,0,53
```

```
        .byte "version is determined by the first INPUT document."
        .byte $16,6,0,63
        .byte "You can combine a Version 2.1 (from GEOS 2.0 or"
        .byte $16,6,0,73
        .byte "geoPublish) document with a Version 1.3 (from GEOS"
        .byte $16,6,0,83
        .byte "1.3).  The result can either be a Version 1.3 ",$0e,"OR",$0f
        .byte $16,6,0,93
        .byte "Version 2.1.  ",$0e,"Note",$0f," though, that Version 1.3 cannot"
        .byte $16,6,0,103
        .byte "handle some Version 2.x (2.0 or 2.1) options, so"
        .byte $16,6,0,113
        .byte "these are dropped when combining Version 2.x files"
        .byte $16,6,0,123
        .byte "into a Version 1.3 file.",$1b,0
help5:  .byte $18,"Graphics included in any of the input pages are"
        .byte $16,6,0,43
        .byte "copied to the output."
        .byte $16,6,0,53
        .byte "You cannot copy headers or footers from an input"
        .byte $16,6,0,63
        .byte "Version 2.x document."
        .byte $16,6,0,73
        .byte "COMBINER will handle either multiple drives, and/or"
        .byte $16,6,0,83
        .byte "input and output from different disks.  You will be"
        .byte $16,6,0,93
        .byte "asked to insert the required disk when it is needed."
        .byte $16,6,0,103
        .byte "Desk Accessories are always loaded from the disk"
        .byte $16,6,0,113
        .byte "which was in the drive COMBINER was loaded from."
        .byte $16,6,0,133
        .byte $19,"      End of HELP Screens.",$1b,0
;
helpptrs:.word help1          ;Pointers to each help screen
        .word help2
        .word help3
        .word help4
        .word help5
maxhelp:                      ;(maxhelp-helpptrs) is high index for screens
;
   .end
; END of $1COMBINER
```

$2COMBINER - *Second source file*

```
; $2COMBINER - Combine multiple geoWrite documents into one.  - Nick Vrtis 1/89
;
.noeqin                       ;these got defined in the 1st file
.include /COMBINER            ;Page zero & GEOS definitions
.eqin
        .psect
remenu3: jmp    remenu
;
dobegin: jsr    cmenus        ;close menus
        window beginw         ;get options
        lda    r0
        sta    ra0            ;save selected option
        cmp    #2
        beq    remenu3        ;..-cancel-
        lda    odskdrv        ;see if current drive is same as last output
        cmp    curdrv
        beq    savodsk        ;..yes
outdrv: jsr    nextdrv        ;must be other drive (which will be NEXT)
;
savodsk: lda   curdrv         ;save ablum disk info
        sta    odskdrv
        ldx    #[r0           ;get boot drive name
        jsr    drvnam
        ldy    #15
101$:   lda    (r0),y
        sta    odsknm,y
        dey
        bpl    101$
        lda    #0
        sta    oldpages       ;no old pages yet
        sta    oldpics        ;or pictures
        lda    ra0            ;get 'BEGIN' option
        cmp    #5
        beq    oldout         ;..'OPEN' old geoWrite file
;
        lda    #$ff
        sta    ovflag         ;don't know version yet
        sta    oldnew         ;this is new file
        ldptr  outnm,r10
        window newfilew       ;get name for new file
        lda    r0
        cmp    #2
        beq    remenu4        ;..-cancel-
        cmp    #6
        beq    newodsk2       ;..-disk-
        cmp    #$80
        beq    outdrv         ;..-drive-
        lda    outnm
        beq    remenu4        ;..no name selected
        ldptr  outnm,r6
        jsr    lookup         ;see if file aready exists
        cpx    #5
        beq    getidsk2       ;..doesn'T EXIST
        jsr    error
        bne    remenu4        ;..other error
        window replw          ;make sure can replace
        lda    r0
        cmp    #4
        beq    remenu4        ;..-no-
        ldptr  outnm,r0
        jsr    delete         ;get rid of original
getidsk2:
        jmp    initidrv       ;..then proceed
outdrv2: jmp   outdrv
newodsk2:
        jmp    newodisk
remenu4: jmp   remenu
;
oldout: ldptr  fnomsg,fnwmsg1
        ldptr  outnm,r5
        ldptr  writecl,r10
        ldptr  odsknm,fnwmsg2
        lda    #7             ;application data type files
        sta    r7
        window filenw         ;get old output file name
        lda    r0
        cmp    #2
        beq    remenu2        ;..-cancel-
        cmp    #6
        beq    newodisk       ;..-disk-
        cmp    #$80
        beq    outdrv2        ;..-drive-
        ldptr  outnm,r0
        jsr    vopen
        jsr    error
        bne    remenu2        ;..bad open
        jsr    getvsn         ;get version number
        bcs    remenu2        ;..unsupported version
        stx    ovflag         ;save version flag
        jsr    countrecs      ;get # pages & pictures
        stx    oldmax         ;save values
        sty    oldpics
        sta    oldpages
        lda    #0             ;set 'OLD' file flag
        sta    oldnew
        jsr    vclose         ;done for now
initidrv:
        lda    idskdrv        ;make sure last used input drive active
        cmp    curdrv
        beq    getidsk        ;..yes-no change needed
        bne    newidrv        ;..need to make other (NEXT) drive current
remenu2: jmp   remenu
;
newodisk:
        jsr    newdisk        ;tell to insert new disk
        jmp    savodsk        ;..go start process over
;
newidisk:
        jsr    newdisk
        jmp    getidsk
newidrv: jsr   nextdrv        ;setup next drive
getidsk: lda   curdrv         ;get current drive #
        sta    idskdrv        ;save drive #
        ldx    #[r0           ;get disk name of input disk
        jsr    drvnam
        ldy    #15
102$:   lda    (r0),y
        sta    idsknm,y
        dey
        bpl    102$
        ldptr  fnimsg,fnwmsg1
        ldptr  inpnm,r5
        ldptr  writecl,r10
        ldptr  idsknm,fnwmsg2
        lda    #7
        sta    r7
        window filenw         ;get input filename
        lda    r0
        cmp    #2
        beq    remenu2        ;..-cancel-
        cmp    #6
        beq    newidisk       ;..-disk-
        cmp    #$80
        beq    newidrv        ;..-drive-
        ldptr  inpnm,r0       ;open input file
        jsr    vopen
        jsr    error
        bne    remenu2
        jsr    getvsn         ;get input file version
        bcs    getidsk        ;..can't handle this version
        stx    ivflag         ;save input version
        bit    oldnew         ;check if output decided yet
        bpl    202$           ;..yes-everything set
;
        sta    ovsn1          ;else 'OLD' version is same as 1st input
        sty    ovsn1+2
        stx    ovflag
        lsr    oldnew         ;clr 'NEW' bit (still need to create one)
;
202$:   jsr    countrecs
        bit    oldnew
        bvc    203$           ;..old file already
;
        stx    oldmax         ;else set max pages from input type
        lda    #64            ;1st available graphic page is same for either
        sta    oldpics
;
203$:   jsr    vclose
        lda    #1             ;set default start/stop as 1 to # pages
        sta    frstipge
        lda    pages
        sta    lastipge
        lda    oldpages       ;default is after last page
        sta    aftpge
getwhere:
        ldptr  fpblk,r10      ;set pointer for first page
        window wherew         ;get 1st/last/after pages
        lda    r0
        cmp    #2
        bne    300$           ;..not cancel
        jmp    remenu
300$:   lda    #0             ;make sure 0 after last entered digit
        ldx    fpblk+4
        sta    fpblk,x
        ldx    lpblk+4
        sta    lpblk,x
        ldx    apblk+4
        sta    apblk,x
        ldx    fpblk          ;convert values
        lda    fpblk+1
        jsr    binary
        beq    fperr          ;..0 is bad
        bcc    chkfp          ;..go check for max
fperr:  ldx    #$83           ;'invalid first page'
pgerr:  jsr    error
        jmp    getwhere
lperr:  ldx    #$84           ;'invalid last page'
        .byte  $2c            ;BIT
aperr:  ldx    #$85           ;'invalid after page'
        inc    frstipge       ;restore to what was typed in
        bne    pgerr          ;..unconditional
chkfp:  dex
        cpx    pages
        bcs    fperr          ;..page is too big
        stx    frstipge       ;save
        ldx    lpblk          ;same process for last page
        lda    lpblk+1
        jsr    binary
        beq    lperr          ;..0 is invalid last page
        bcs    lperr
        dex
```

```
        cpx     pages
        bcs     lperr
        cpx     frstipge
        bcc     lperr       ;..can't be less than first page
        inx
        stx     lastipge
        ldx     apblk
        lda     apblk+1
        jsr     binary
        beq     303$        ;..inserting at start
        bcs     aperr
        dex
        cpx     oldpages
        bcs     aperr
        inx
303$:   stx     aftpge
        sec
        lda     lastipge    ;calc # pages being added
        sbc     frstipge
        clc
        adc     oldpages    ;calc total resulting pages
        cmp     oldmax
        bcc     roomok      ;..they will fit
        ldx     #$80        ;"Too many pages"
        jsr     error
        jmp     getidsk
roomok: ldx     #63         ;clear used flags for pictures
        lda     #0
206$:   sta     picsused,x
        dex
        bpl     206$
;
; here to get pages from input into buffer area
pagesin: ldptr  bufbeg,ra2  ;buffer area is empty
        lda     idskdrv
        ldx     #[idsknm
        ldy     #]idsknm
        jsr     chkdsk      ;make sure input disk mounted
        beq     remenu5     ;..cancel from disk mount
        ldptr   inpnm,r0    ;open the input file
        jsr     vopen
        jsr     error
        bne     remenu5     ;..error in open
;
; here for the start of each page
pageloop:
        ldx     frstipge
        jsr     vgoto
        jsr     readpage    ;go get the page in
        cpx     #11         ;check for no buffer space error
        bne     211$        ;..not that
        jmp     pagesout    ;else need to output pages in so far
211$:   jsr     error
        bne     remenu5     ;..load failed
        lda     #0
        sta     backlvlf    ;assume no backlevel challenges
        ldx     ivflag
        cpx     ovflag
        beq     torulerok   ;..no ruler escape fix needed
        bcs     tobacklvl   ;..need to backlevel the input file
; going from lower input level to higher output level
        cpx     #2          ;check for V2.0 as input
        bne     vlxv2x      ;..no->V1.x to V2.x
        beq     chkuplvl    ;..go chk for upleveling (2.0 to 2.1)
;
torulerok:
        jmp     rulerok
tobacklvl:
        jmp     backlvl
remenu5: jmp    remenu
;
; must be from V1.x to V2.x
vlxv2x: movew   ra3,r4      ;need to expand starting ruler
        clc
        lda     r4
        adc     #20         ;FROM past 1.x ruler
        sta     r0
        lda     r4+1
        adc     #0
        sta     r0+1
        clc
        lda     r4
        adc     #27         ;TO where 2.x needs
```

```
        sta     r1
        lda     r4+1
        adc     #0
        sta     r1+1
        jsr     perfmove
        ldy     #2+2+16-1   ;RM, LM, + 8 TABS
600$:   lda     (r4),y      ;make room for ruler escape
        iny
        sta     (r4),y
        dey
        dey
        bpl     600$
        lda     #17         ;add ruler escape
        ldy     #0
        sta     (r4),y
        iny
        lda     (r4),y      ;make paragrph indent=left margin
        tax
        iny
        lda     (r4),y
        ldy     #22
        sta     (r4),y
        dey
        txa
        sta     (r4),y
        iny
601$:   iny                 ;0 justification/text color/reserved
        lda     #0
        sta     (r4),y
        cpy     #26
        bcc     601$
        lda     #$10        ;I don't know what this bit does
        ldy     #23         ;but geoWrite sets it on
        sta     (r4),y
        clc                 ;add to end of buffer pointer
        lda     r7
        adc     #27-20
        sta     r7
        bcc     chkuplvl
        inc     r7+1
chkuplvl:
        clc
        lda     ra2         ;set pointer past escape
        adc     #1
        sta     r4
        lda     ra2+1
        adc     #0
        sta     r4+1
        jsr     uplvl       ;see if need to uplevel the ruler
        jmp     rulerok
backlvl: movew  ra3,r4      ;set pointer to start of buffer
        sec
        ror     backlvlf    ;set 1st back level flag bit
        ldx     ovflag
        cpx     #2          ;check output level
        bcs     603$        ;..must be 2.1->2.0
        sec
        ror     backlvlf    ;set flag as 2.x->1.x
        ldx     ivflag      ;check input version
        cpx     #2
        beq     slidetabs   ;..going 2.0 to 1.x (tabs values are OK)
603$:   ldy     #1          ;Never have 1.x input (would be 1.x->1.x)
604$:   jsr     fixinches   ;go adjust inch offsets
        cpy     #23
        bcc     604$
        bit     backlvlf    ;check how far back level
        bvc     rulerok     ;..2.1->2.0 is done
; need to slide tabs up for 2.x to 1.x
slidetabs:
        ldy     #1
607$:   lda     (r4),y
        dey
        sta     (r4),y
        iny
        iny
        cpy     #21         ;don't bother with paragraph margin
        bcc     607$
        clc
        lda     r4          ;move FROM is just past V2.x escape
        adc     #27
        sta     r0
        lda     r4+1
        adc     #0
        sta     r0+1
```

```
        clc                 ;move TO is just past TABS
        lda     r4
        adc     #20
        sta     r1
        lda     r4+1
        adc     #0
        sta     r1+1
        jsr     perfmove
        sec                 ;back up ending address
        lda     r7
        sbc     #27-20
        sta     r7
        bcs     rulerok
        dec     r7+1
rulerok: ldx    ovflag      ;adjust past start of each page
        lda     #31         ;assume V2.x
        cpx     #2
        bcs     300$        ;..OK
        lda     #24         ;must be V1.x
300$:   clc
        adc     ra3
        sta     r4
        lda     ra3+1
        adc     #0
        sta     r4+1
;
chrloop: jsr    getchr      ;here to get input character
        beq     toendpage   ;..end of input
        cmp     #16
        beq     tographic   ;..graphic escape
        cmp     #17
        beq     ruler       ;..ruler escape
        cmp     #23
        bne     chrloop     ;..not newcard escape
        beq     newcard     ;..newcard escape
;
toendpage:
        jmp     endpage
tographic:
        jmp     graphic
;
newcard: lda    #5-1        ;NEWCARD skips 5 chrs
bmpr4:  clc                 ;add offset to r0
        adc     r4
        sta     r4
        bcc     chrloop
        inc     r4+1
        bne     chrloop     ;..unconditional
ruler:  bit     backlvlf    ;check backlevel
        bvs     rulerout    ;.. 2.x to 1.x
        bmi     rulerfix    ;.. 2.1 to 2.0
        lda     ivflag      ;check if same level
        cmp     ovflag
        beq     307$        ;..yes
        jsr     uplvl       ;else may need to uplevel the ruler
307$:   lda     #27-1       ;RULER is 27 characters
        bne     bmpr4       ;..unconditional
rulerfix:
        ldy     #0          ;have already skipped the escape
304$:   jsr     fixinches
        cpy     #23-1       ;see if to the end of tabs/margins/etc.
        bcc     304$        ;.. more to do
        lda     #27-1       ;all fixed
        bne     bmpr4       ;..go skip it
;
rulerout:
        sec                 ;backup to start of ruler escape
        lda     r4
        sbc     #1
        sta     r1          ;that is the TO
        sta     r4          ;will also be next character needed
        lda     r4+1
        sbc     #0
        sta     r1+1
        sta     r4+1
        clc
        lda     r1
        adc     #27         ;start of ruler + length = FROM
        sta     r0
        lda     r1+1
        adc     #0
        sta     r0+1
```

```
        jsr     perfmove
        sec                     ;shorten end of data also
        lda     r7
        sbc     #27
        sta     r7
        bcs     308$
        dec     r7+1
308$:   jmp     chrloop         ;don't skip any more characters
;
graphic: ldy    #3
        lda     (r4),y          ;get graphic record number
        tax
        lda     picsused-64,x  ;see if already referenced
        bne     302$            ;..yes
        lda     oldpics
        cmp     #128
        bcc     301$            ;..room for at least 1 more
        ldx     #$81            ;"Too many pictures"
        jsr     error
        jmp     remenu
301$:   inc     oldpics         ;count record used
        sta     picsused-64,x  ;keep where stored
302$:   sta     (r4),y          ;replace with new record id
        lda     #5-1            ;graphic escape is 5 bytes
        jmp     bmpr4           ;..go skip it
;
; here at the end of each input page
endpage: jsr    nextarea        ;save end of this area & setup next one
        inc     frstipge        ;bump to next input page
        lda     frstipge
        cmp     lastipge        ;see if done yet
        beq     pagesout        ;..yes-write them out
        jmp     pageloop        ;else just go get another input page
;
; here either when buffer area is full, or all input pages read
pagesout:
        jsr     seteob          ;set end of buffer flags
        jsr     vclose          ;done with input file (for now)
        lda     odskdrv
        ldx     #[odsknm
        ldy     #]odsknm
        jsr     chkdsk          ;make sure output disk mounted
        beq     remenu6         ;..CANCEL from mount
        bit     oldnew          ;see if need to create output file
        bvc     openold         ;..no
        ldptr   writeinfo,r9 ;else create one first
        lda     #0
        sta     r10
        sta     oldnew          ;file is now OLD
        jsr     save
        jsr     error
        bne     remenu6         ;..could not create file
        ldptr   outnm,r0
        jsr     vopen
        lda     #0
        sta     xsave
210$:   jsr     vappend         ;extend vlir file to 127 entries
        inc     xsave
        bpl     210$
        jsr     vclose
openold: ldptr  outnm,r0        ;open the output file
        jsr     vopen
        jsr     error
        beq     pageout         ;..opened ok
        jsr     vclose          ;error-close output file
remenu6: jmp    remenu
;
; here for the start of each page to output
pageout: jsr    getarea         ;point to a used area
        bne     500$            ;..area has data in it
;
        jsr     vclose          ;done with output for now
        lda     frstipge        ;else see how we got here
        cmp     lastipge
        beq     501$            ;..we just finished the last input page
        jmp     pagesin         ;..we need to get more pages
501$:   jmp     textdone        ;..we are finished with the text portion
;
500$:   lda     aftpge
        cmp     oldpages        ;check where adding
        beq     toaddlst        ;..1st page added to end
```

```
        bcs     addoth          ;..other pages added to the end
        sec                     ;calc # chains to move
        lda     oldpages
        sbc     aftpge
        tay
        lda     oldpages        ;adding in the middle of the original
        asl     a
        tax                     ;old end *2 is last used chain index
503$:   lda     bufl+0,x        ;slide left 2 bytes
        sta     bufl+2,x
        lda     bufl+1,x
        sta     bufl+3,x
        dex
        dex
        dey
        bne     503$
        lda     #0              ;set this as empty chain
        sta     bufl+2,x
        sta     bufl+3,x
        inc     oldpages        ;+1 to page counter
;
        ldx     frstipge        ;see if this will be the last page to add
        inx
        cpx     lastipge
        bcc     todowrite       ;..no-then OK to write
        sec                     ;else need to make sure last char is non-null
        lda     r7
        sbc     #1
        sta     r1
        lda     r7+1
        sbc     #0
        sta     r1+1
        ldy     #0
        lda     (r1),y
        bne     todowrite       ;..was non-null
        lda     #$0c            ;replace null with page skip
        sta     (r1),y
todowrite:
        jmp     dowrite         ;..now do the write
toaddlst:
        jmp     addlst
;
addoth: ldx     frstipge        ;see if this is last page to be added
        inx
        cpx     lastipge
        bcc     todowrite       ;..no-then OK to write
        sec                     ;must have a null at the end
        lda     r7
        sbc     #1
        sta     r1
        lda     r7+1
        sbc     #0
        sta     r1+1
        ldy     #0
        lda     (r1),y
        beq     dowrite         ;..already ends in null
        cmp     #$0c
        beq     501$            ;..replace ending page skip with null
        inc     r1              ;else extend 1 char
        bne     500$
500$:   inc     r1+1
        inc     r7
        bne     501$
        inc     r7+1
501$:   lda     #0
        sta     (r0),y
        beq     dowrite         ;..unconditional
addlst: ldx     oldpages
        beq     dowrite         ;..no adjustment (no old pages)
        dex                     ;else find last good chain
        txa
        jsr     vgoto           ;set initial T/S
        ldptr   tsbuf,r3        ;set buffer for T/S
        jsr     follow
        jsr     error
        beq     505$
        jsr     vclose          ;close the output file anyway
        jmp     remenu
505$:   ldx     buf0+1          ;get index to last char
        lda     #$0c            ;replace with page skip
```

```
        sta     buf0,x
        ldx     #[(-2)          ;start @ -2 so 2 inx's = 0
502$:   inx                     ;find end of chain
        inx
        lda     tsbuf,x         ;check track
        bne     502$            ;..not zero, so not end of chain
        lda     tsbuf-1,x       ;get last T/S
        sta     r1+1
        lda     tsbuf-2,x
        sta     r1
        ldptr   buf0,r4
        jsr     write           ;rewrite T/S
dowrite: lda    aftpge          ;get page to write to
        jsr     writepage
        jsr     error
        beq     303$
        jsr     vclose
        jmp     remenu          ;..save was bad
303$:   inc     aftpge          ;next page to store to
        jmp     pageout         ;..go do another page
;
textdone:
        ldx     #64-1           ;now process the pictures used from input
        stx     picinx          ;set input & output indexes to (start-1)
        stx     picoutx
;
; here to get input pictures from input
picsin: ldx     picinx          ;see if any pictures needed
305$:   inx
        cpx     #128
        bcs     topicsdone      ;..none used
        lda     picsused,x
        beq     305$            ;..this one not used (end test @ 1st used)
        ldptr   bufbeg,ra2      ;reset to start of buffer area
        lda     idskdrv
        ldx     #[idsknm
        ldy     #]idsknm
        jsr     chkdsk          ;make sure input disk mounted
        beq     topicsdone      ;..CANCELed/not really done
        ldptr   inpnm,r0
        jsr     vopen           ;open input file
        jsr     error
        bne     picerr          ;..error on open
; here to process each graphic picture referenced in the input
picloop: inc    picinx          ;bump for next picture area
        ldx     picinx
        cpx     #128
        bcs     picsout         ;..end of pictures - wrap up last areas
        lda     picsused-64,x
        beq     picloop         ;..this one not used
        txa                     ;X is input VLIR record #
        jsr     vgoto
        jsr     readpage        ;get graphics page in
        cpx     #11             ;check for out of room in buffer
        beq     picsout         ;..out of room, need to write what is there
        jsr     error
        bne     picerr          ;..error writing
        jsr     nextarea        ;setup next area
        jmp     picloop         ;..go do another
topicsdone:
        jmp     picsdone        ;..just passing through
; here when buffer files or end of pictures form input
picsout: jsr    seteob          ;set end of buffer & reset pointer to start
        jsr     vclose          ;done with input file (for now)
        lda     odskdrv
        ldx     #[odsknm
        ldy     #]odsknm
        jsr     chkdsk          ;make sure output disk mounted
        beq     picsdone        ;..CANCELed/not really done
        ldptr   outnm,r0
        jsr     vopen
        jsr     error
        bne     picerr          ;..error
; here to process each picture from buffer
picout: jsr     getarea         ;setup pointers for a used area
        bne     304$            ;..got some data to do
;
        jsr     vclose
        ldx     picoutx         ;else see if done
        cpx     #128
        bcs     picsdone        ;..done with all pictures
        jmp     picsin          ;..more graphics to read in
;
```

```
304$:    inc     picoutx       ;bump output counter
         ldx     picoutx
         lda     picsused-64,x ;get new record #
         beq     304$          ;..skip to one that was used
         jsr     writepage     ;write it out
         jsr     error
         beq     picout        ;..go do another
picerr:  jsr     vclose
picsdone:
         jmp     remenu        ;let the process start again
;
; Start of subroutines
; check to make sure correct disk is mounted
chkdsk:  stx     ra1           ;save ptr to name
         sty     ra1+1
         tax
         clc
         adc     #'A'-8
         sta     swpdskd       ;save drive letter
         txa
         cmp     curdrv        ;check against current drive
         beq     chkdsknm      ;..don't need to open
         jsr     drvset
rechkdsk:
         jsr     opndsk
chkdsknm:
         ldx     #[r0
         jsr     drvnam
         ldy     #15
112$:    lda     (r0),y
         cmp     (ra1),y
         bne     swpdsk        ;..need to swap disks
         dey
         bpl     112$
         rts                   ;correct mounted/return (NE set)
swpdsk:  window  swpdskw       ;put up window to swap the disk
         lda     r0
         cmp     #2
         bne     rechkdsk      ;..not CANCEL/verify disk name
         rts                   ;..return with EQ set if CANCEL
; advance to next drive number (8 or 9)
nextdrv: ldx     curdrv        ;get current drive #
         inx                   ;bump to next
         cpx     #10
         bcc     111$          ;..ok
         ldx     #8            ;back to 8
111$:    txa
         jsr     drvset        ;make current
         jmp     opndsk        ;read in name
; allow user to insert a new disk
newdisk: lda     curdrv        ;get current drive
         clc
         adc     #'A'-8
         sta     nddrive       ;put drive letter in window
         window  newdiskw
         jmp     opndsk        ;open/get disk name
; calc length of data to move and the call MOVEDATA
perfmove:
         sec                   ;calc number of bytes to move
         lda     r7
         sbc     r0
         sta     r2
         lda     r7+1
         sbc     r0+1
         sta     r2+1
         jmp     movedata
; check for error and put up window with decoded message if so
error:   txa                   ;chk for error
         beq     errrts        ;..no error
         sta     miscerr       ;so always finds something
         ldptr   errmtbl,r12
errmlp:  ldy     #0
         lda     miscerr
         cmp     (r12),y
         beq     errfnd        ;..found it
errmskp: iny                   ;not found/find end of message
         lda     (r12),y
         bne     errmskp
         tya
         sec                   ;skip over end also
         adc     r12
         sta     r12
         bcc     errmlp        ;..look some more
         inc     r12+1
         bne     errmlp        ;..unconditional
errfnd:  inc     r12           ;skip past code
         bne     115$
         inc     r12+1
115$:    lda     miscerr       ;in case needed
         jsr     ascii
         sta     miscerrd
         stx     miscerrd+1
         window  errorw
errrts:  rts
; convert binary number in X reg to two ASCII digits
decimal: lda     #0            ;this is not 'elegant', but it is effective
         cpx     #0
         beq     ascii         ;0 is input
117$:    clc
         sed
         adc     #1
         cld
         dex
         bne     117$          ;fall through to ascii
; convert value in A reg to two ASCII hex characters (in X&A)
ascii:   pha
         jsr     toascii
         tax                   ;1st digit in X
         pla
         lsr     a
         lsr     a
         lsr     a
         lsr     a
toascii: and     #$0f
         ora     #$30
         cmp     #$3a
         bcc     116$
         adc     #6
116$:    rts
; set pointer from X/Y and call WINDOW routine
xywindow:
         stx     r0
         sty     r0+1
         jmp     window
;
; get the version number for the directory entry & check if supported
getvsn:  movew   direntry+19,r1  ;get t/s of info sector
         ldptr     buf0,r4       ;set where to read it into
         jsr     read
         ldx     #1            ;assume 1.x
         lda     buf0+90       ;get Va
         ldy     buf0+92       ;get Va.y
         cmp     #'1'
         beq     vsnok         ;..V1.x
         inx                   ;assume 2.?
         cmp     #'2'
         bne     unsvsn        ;..not V1.x or V2.x is error
         cpy     #'1'
         bcc     vsnok         ;..V2.0
         inx                   ;must be V2.1 or higher
vsnok:   clc                   ;set OK flag
         rts
unsvsn:  ldx     #$82          ;tell bad version
         jsr     error
         jsr     vclose        ;close file (can't use it)
         sec                   ;set BAD flag
         rts
; count the number of text pages & picture pages in the vlir file
countrecs:
         lda     #64           ;assume V1.x (64 max)
         cpx     #1
         bcc     200$          ;..V1.x
         lda     #61           ;else 61 max for V2.x
200$:    sta     max
         lda     #0            ;clear counters
         sta     pages
         lda     #64
         sta     pics
cntpages:
         lda     pages
         cmp     max           ;check for full
         bcs     cntpics       ;..full
         jsr     vgoto
         tya
         beq     cntpics       ;..end of text pages
         inc     pages
         bne     cntpages      ;..unconditional
cntpics: lda     pics          ;same process with pictures
         cmp     #128          ;up to 64 pics in either
         beq     endcount
         jsr     vgoto
         tya
         beq     endcount      ;..end of pictures
         inc     pics
         bne     cntpics       ;..unconditional
endcount:
         ldx     max
         ldy     pics
         lda     pages
         rts
; redraw COMBINER screen when WINDOW/MENU needs to restore it
repat:   lda     openpat       ;get pattern used from opening screen
         jsr     setpat        ;make current
         jsr     pfill         ;that will restore screen
         lda     r2+1          ;check bottom
         cmp     #[(174+1)     ;to see if undoing shadow of help window
         bcs     100$          ;..yes-need to redraw title box
         rts                   ;else just return
100$:    jmp     redotitle     ;redraw title box
;
; get next character from input buffer area
getchr:  ldy     #0            ;get next text character
         lda     (r4),y
         inc     r4            ;bump for next time
         bne     400$
         inc     r4+1
400$:    ldx     r4+1          ;see if this was last character
         cpx     r7+1
         bne     401$          ;..can't be, return with NE set
         ldx     r4
         cpx     r7
401$     rts
; upgrade ruler to V2.1
uplvl:   lda     ovflag        ;check for V2.1 output
         cmp     #3
         bcc     611$          ;..no
         ldy     #0
610$:    lda     (r4),y
         cmp     #[(480-1)     ;check for old max right (7.2" was $1df)
         bne     612$          ;..no problem
         lda     #[480         ;fudge to even 7.2 inches
612$:    clc                   ;in V1.x or V2.0, pixel 0 is @ 80 in V2.1
         adc     #80
         sta     (r4),y
         iny
         lda     (r4),y
         adc     #0
         sta     (r4),y
         iny
         cpy     #2+2+16+2+1   ;RM+LM+8 tabs+pi
         bcc     610$
611$:    rts
; setup the INPUT portions of the window asking for page numbers
whereset:
         ldptr     wboxes,r0
         jsr     cboxes
         lda     #[(apblk-wblk) ;setup the INPUT portions of where window
         ldx     aftpge
         jsr     wherein
         lda     #[(lpblk-wblk)
         ldx     lastipge
         jsr     wherein
         lda     #[(fpblk-wblk) ;note that first page is last
         ldx     frstipge
         jmp     wherein
;
; display values in where window & save pointers
wherein: sta     widxsave      ;save for index offset
         clc                   ;calc real address
         adc     #[wblk
         sta     r0
         lda     #]wblk
         adc     #0
         sta     r0+1
         jsr     decimal       ;convert # to ascii
         ldy     widxsave
         sta     wblk,y
```

```
        txa
        sta     wblk+1,y
        clc                     ;calc pixel line to display on
        lda     wblk+3,y
        adc     pline
        sta     r1+1
        ldptr   (64+10),r11     ;DEF-DB-LEFT+8
        jsr     dsptxt
        lda     r11             ;move column to where save will find
        sta     cursx
        lda     r11+1
        sta     cursx+1
        lda     #2              ;have 2 chrs displayed
        sta     inplen
; save pointers from current INPUT area in where window
svwhere: ldx    widxsave        ;get offset into buffer area
        lda     inplen          ;save variable stuff
        sta     wblk+4,x
        lda     cursx
        sta     wblk+5,x
        lda     cursx+1
        sta     wblk+6,x
        rts
; advance to requested input area in where window
nxtwhere:
        sty     widxsave        ;save table index for save
        lda     wblk+4,y        ;restore pointers from the table
        sta     inplen
        lda     wblk+5,y
        sta     cursx
        lda     wblk+6,y
        sta     cursx+1
        lda     wblk+3,y
        sta     cursy
        tya
        clc                     ;calc address of string buffer
        adc     #[wblk
        sta     string
        lda     #]wblk
        adc     #0
        sta     string+1
        jmp     prompton
;
; convert characters in X/Y to a binary number
binary: tay                     ;check if 2 digits
        bne     150$            ;..yes
        txa                     ;else move 1st digit to 2nd
        ldx     #'0'            ;and replace 1st with 0
150$:   sta     r1              ;save 2nd digit for now
        cpx     #'0'
        bcc     bcerr           ;..bad digit
        cpx     #'9'+1
        bcs     bcerr
        txa
        and     #$0f
        asl     a               ;*2
        sta     r0
        asl     a               ;*4
        asl     a               ;*8
        adc     r0              ;+*2=*10
        sta     r0
        lda     r1
        cmp     #'0'
        bcc     bcerr
        cmp     #'9'+1
        bcs     bcerr
        and     #$0f
        adc     r0
        tax
        rts                     ;return w/ carry clear
bcerr:  sec                     ;set error flag
        rts
; handle click on first page block in where window
dopg1st: ldy    #[(fpblk-wblk)
dopg:   jsr     svwhere         ;save current pointers
        jmp     nxtwhere        ;move to new area
; handle click on last page block in where window
dopglst: ldy    #[(lpblk-wblk)
        bne     dopg            ;..unconditional
; handle click on after page block in where window
dopgaft: ldy    #[(apblk-wblk)
        bne     dopg            ;..unconditional
; handle click on OK box in where window
dook:   lda     #1              ;set OK code
        .byte   $2c
docancel:
        lda     #2              ;set CANCEL code
        sta     wincmd
        jmp     clswin          ;close the window
; handle click on DRIVE box in various windows
dodrv:  lda     #$80            ;return $80 from custom box
        sta     wincmd
        jmp     clswin
; handle click on CREATE box in begin window
docreate:
        lda     #$81
        sta     wincmd
        jmp     clswin
;
; fix tabs, etc. when going from V2.1 to other versions
fixinches:
        sec                     ;need to shift tabs down
        lda     (r4),y
        sbc     #[80
        sta     r5
        tax                     ;save copy for testing
        iny
        lda     (r4),y
        and     #$7f            ;in case of decimal tabs
        sbc     #]80
        bcs     605$
        lda     #0              ;if < 0, make 0
        tax
605$:   cmp     #]480           ;check if still too big
        bne     607$
        cpx     #[480
607$:   bcc     608$            ;..size is ok
        ldx     #[479           ;else this is max size
        lda     #]479
608$:   stx     r5
        sta     r5+1
        lda     #$80            ;put back decimal flag if there before
        bit     backlvlf        ;only if result is 2.0
        bvc     606$            ;..2.0
        lda     #$00            ;discard if result=1.x
606$:   and     (r4),y
        ora     r5+1
        sta     (r4),y
        dey
        lda     r5
        sta     (r4),y
        iny
        iny
        rts
; read in a page of text or graphics to next available buffer area
readpage:
        clc                     ;read in a data page
        lda     ra2             ;leave room for pointer to end of area
        adc     #2
        sta     r7              ;where to start loading
        sta     ra3             ;save copy of start of data portion
        lda     ra2+1
        adc     #0
        sta     r7+1
        sta     ra3+1
        sec                     ;calc bytes available
        lda     #[bufend
        sbc     r7
        sta     r2
        lda     #]bufend
        sbc     r7+1
        sta     r2+1
        jmp     lchain          ;load the series of sectors
;
; write a page of text or graphics from buffer area
writepage:
        jsr     vgoto           ;position to write new page
        sec
        lda     r7              ;calc bytes used
        sbc     ra3
        sta     r2
        lda     r7+1
        sbc     ra3+1
        sta     r2+1
        movew   ra3,r7
        jmp     vsave
; advance from one buffer area to the next
nextarea:
        lda     r7              ;get pointer to end of area used
        ldy     #0              ;save at start of buffer area
        sta     (ra2),y
        tax
        lda     r7+1
        iny
        sta     (ra2),y
        sta     ra2+1           ;end becomes start of next area
        stx     ra2
        rts
; set zeros at the end of buffer area used to flag the end
seteob: ldy     #0              ;set end of buffer
        tya
        sta     (ra2),y
        iny
        sta     (ra2),y
        ldptr   bufbeg,ra2      ;reset pointer to start of buffer
        rts
; setup pointers for an area to read data into
getarea: clc                    ;set pointer to data portion
        lda     ra2
        adc     #2
        sta     ra3
        lda     ra2+1
        adc     #0
        sta     ra3+1
        ldy     #0              ;get end of area pointer
        lda     (ra2),y
        sta     r7
        tax
        iny
        lda     (ra2),y
        sta     r7+1
        sta     ra2+1           ;set pointer to start of next area
        stx     ra2
        rts                     ;0=end of areas (else ne)
;
opentitle:
        jsr     grphc2          ;opening screen
        .byte    $05
openpat: .byte   24             ;pattern used to clear the screen
        .byte   $01             ;erase screen
        .word   0
        .byte   0
        .byte   $03
        .word   320
        .byte   199
        .byte   0
redotitle:
        jsr     grphc2          ;done this way so REPAT can redraw title
        .byte   $05,9           ;pattern fill title area
        .byte   $01
        .word   8
        .byte   176
        .byte   $03
        .word   312
        .byte   198
        .byte   $06
        .word   20
        .byte   190
        .byte   $18,$20,$1a,"Combiner V1.1",$1b,$18
        .byte   " Combine geoWrite files. ",$1b,0
        rts
;
mainmenu:
        .byte   0
        .byte   14
        .word   0
        .word   120
        .byte   4               ;4 menu options
        .word   geos
        .byte   $80             ;submenu
        .word   geosmenu
        .word   done
        .byte   $80
        .word   donemenu
        .word   begin
        .byte   $00             ;flash & run
```

```
         .word    dobegin
         .word    help
         .byte    $00
         .word    dohelp
geos:    .byte    "GEOS",0
done:    .byte    "Done",0
begin:   .byte    "Begin",0
help:    .byte    "Help",0
;
geosmenu:
         .byte    14          ;start below mainmenu
gmhgt:   .block    1          ;height depends on # D.A.'S
         .word    0
         .word    75
gmopts:  .block    1          ;# D.A.'S + 1 + $80
         .word    info
         .byte    0
         .word    doinfo
         .word    dan1
         .byte    0
         .word    doda1
         .word    dan2
         .byte    0
         .word    doda2
         .word    dan3
         .byte    0
         .word    doda3
         .word    dan4
         .byte    0
         .word    doda4
         .word    dan5
         .byte    0
         .word    doda5
         .word    dan6
         .byte    0
         .word    doda6
         .word    dan7
         .byte    0
         .word    doda7
         .word    dan8
         .byte    0
         .word    doda8
info:    .byte    "COMBINER Info",0
;
infow:   .byte    $81
         .byte    $0b,10,14
         .word    infom1
         .byte    $0b,10,35
         .word    infom2
         .byte    $0b,10,50
         .word    infom3
         .byte    $0b,10,65
         .word    infom4
         .byte    $0e
         .byte    0
infom1:  .byte    $1a,"COMBINER",$1b,$18," V1.1",0
infom2:  .byte    "Nicholas J. Vrtis",0
infom3:  .byte    "5863 Pinetree S.E.",0
infom4:  .byte    "Kentwood, MI 49508",0
;
donemenu:
         .byte    14
         .byte    14+14+14
         .word    28,75
         .byte    $80+2
         .word    quit
         .byte    0
         .word    restrt     ;reload desktop
         .word    geowrite
         .byte    0
         .word    dogeowrite
quit:    .byte    "Quit",0
geowrite:
         .byte    "GEOWRITE",0
;
beginw:  .byte    $81         ;initial options from 'BEGIN'
         .byte    $0b,60,6+12
         .word    createm
         .byte    $0b,60,40+12
         .word    openm
         .byte    $0b,60,74+12
```

```
         .word    cancelm
         .byte    $12,1,6       ;-create-
         .word    createbox
         .byte    $05,1,40      ;-open-
         .byte    $02,1,74      ;-cancel-
         .byte    0
createm: .byte    $18,"new geoWrite file",0
openm:   .byte    "existing geoWrite file",0
cancelm: .byte    "and go to MainMenu",$1b,0
;
createbox:
         .word    creategph
         .word    0
         .byte    6,16
         .word    docreate
creategph:
         .byte    $05,$ff
         .byte    $82,$fe,$80,$04,$00,$82,$03,$80
         .byte    $04,$00,$b8,$03,$8f,$80,$00,$00
         .byte    $00,$03,$98,$c0,$00,$00,$60,$03
         .byte    $98,$1f,$9e,$3c,$f1,$e3,$98,$1c
         .byte    $33,$66,$63,$33,$98,$18,$33,$3e
         .byte    $63,$33,$98,$18,$3f,$66,$63,$f3
         .byte    $98,$18,$30,$66,$63,$03,$98,$d8
         .byte    $33,$66,$63,$33,$8f,$98,$1e,$3e
         .byte    $39,$e3,$80,$04,$00,$82,$03,$80
         .byte    $04,$00,$81,$03,$06,$ff,$81,$7f
         .byte    $05,$ff
;
drivebox:
         .word    drvgraph
         .word    0
         .byte    6,16
         .word    dodrv
drvgraph:
         .byte    $05,$ff,$82,$fe
         .byte    $80,$04,$00,$82,$03
         .byte    $80,$04,$00,$b7,$03
         .byte    $80,$f8,$00,$c0,$00,$03
         .byte    $80,$cc,$00,$00,$00,$03
         .byte    $80,$c6,$fd,$d9,$9e,$03
         .byte    $80,$c6,$e0,$d9,$b3,$03
         .byte    $80,$c6,$c0,$d9,$b3,$03
         .byte    $80,$c6,$c0,$cf,$3f,$03
         .byte    $80,$c6,$c0,$cf,$30,$03
         .byte    $80,$cc,$c0,$c6,$33,$03
         .byte    $80,$f8,$c0,$c6,$1e,$03
         .byte    $e2,$02,$01
         .byte    $80,$04,$00,$01,$03
         .byte    $06,$ff
         .byte    $01,$7f,$05,$ff
;
filenw:  .byte    $01         ;non-std window
         .byte    40,145      ;std height+10
         .word    72
         .word    263
         .byte    $0b,2,9
fnwmsg1: .block   2
         .byte    $0b,130,19
         .word    fndmsg
         .byte    $0b,130,28
fnwmsg2: .block   2
         .byte    $05,17,34  ;-open-
         .byte    $06,17,52  ;-disk-
         .byte    $02,17,88  ;-cancel-
         .byte    $10,4,14   ;filename list
drvopt1: .byte    0          ;$12=-drive-
         .byte    17,70
         .word    drivebox
         .byte    0
fnomsg:  .byte    $18,$19,"Select geoWrite output file.",$1b,$18,0
fnimsg:  .byte    $18,"Select geoWrite input file.",0
fndmsg:  .byte    "On disk:",$1b,0
;
newdiskw:
         .byte    $81
         .byte    $0b,10,30
         .word    ndmsg
         .byte    $01,17,78  ;-ok-
         .byte    0
ndmsg:   .byte    $18,"Insert new disk in drive "
```

```
nddrive: .byte    0,":",$1b,0
;
newfilew:
         .byte    $81
         .byte    $0b,8,15
         .word    nfmsg1
         .byte    $0b,8,30
         .word    nfmsg2
         .byte    $02,17,78        ;-cancel-
         .byte    $06,17,42        ;-disk-
         .byte    $0d,8,50,r10,16
drvopt3: .byte    0                ;$12=-drive-
         .byte    17,60
         .word    drivebox
         .byte    0
nfmsg1:  .byte    $18,"Enter name of new geoWrite file.",0
nfmsg2:  .byte    "On Disk: ",$1b
odsknm:  .block   16
         .byte    0
;
replw:   .byte    $81
         .byte    $0b,10,30
         .word    replm
         .byte    $0b,10,50
         .word    outnm
         .byte    $03,17,78      ;-yes-
         .byte    $04,1,78       ;-no-
         .byte    0
replm:   .byte    $18,"File exists, OK to replace file:",0
;
errorw:  .byte    $80+2
         .byte    $0c,10,32,r12
         .byte    $01,17,72      ;-ok-
         .byte    0
errmtbl:                         ;table of error codes and messages
         .byte    $80,$18,"Combined file has too many pages.",$1b,0
         .byte    $81,$18,"Combined file has too many graphics.",$1b,0
         .byte    $82,$18,"Unsupported geoWrite Version.",$1b,0
         .byte    $83,$18,"First page number is invalid.",$1b,0
         .byte    $84,$18,"Last page number is invalid.",$1b,0
         .byte    $85,$18,"After page number is invalid.",$1b,0
         .byte    $03,$18,"Disk full",$1b,0
         .byte    $26,$18,"Write protect on",$1b,0
miscerr: .block   1
         .byte    $18,"Disk Error: "
miscerrd:
         .block   2
         .byte    $1b,0
;
swpdskw: .byte    $81
         .byte    $0b,10,32
         .word    swpdskm1
         .byte    $0c,10,47,ra1
         .byte    $0b,10,62
         .word    swpdskm2
         .byte    $01,17,78      ;-ok-
         .byte    $02,1,78       ;-cancel-
         .byte    0
swpdskm1:
         .byte    $18,"Please insert disk:",$1b,0
swpdskm2:
         .byte    $18,"In Drive: "
swpdskd: .block   1
         .byte    $1b,0
;
wherew:  .byte    $81
         .byte    $13            ;call user routine first
         .word    whereset       ;this sets up ICON boxes
         .byte    $0d,10,14-2,r10,2
         .byte    $0b,43,20-2
         .word    wwmsg1
         .byte    $0b,43,43-2
         .word    wwmsg2
         .byte    $0b,43,66-2
         .word    wwmsg3
         .byte    $0b,43,74
         .word    wwmsg4
         .byte    0
wwmsg1:  .byte    $18,"First page of input to use.",0
wwmsg2:  .byte    "Last page of input to use.",0
wwmsg3:  .byte    "Place input after this page.",0
wwmsg4:  .byte    $1b,"(Use '0' to place at start.)",0
;
wboxes:  .byte    5          ;5 boxes in the table
```

```
          .word     64+16      ;put mouse in OK box
          .byte     32+88
          .word     pgbox
          .byte     8+1,32+8
          .byte     3,16
          .word     dopglst
          .word     pgbox
          .byte     8+1,32+32
          .byte     3,16
          .word     dopglst
          .word     pgbox
          .byte     8+1,32+56
          .byte     3,16
          .word     dopgaft
          .word     okbox
          .byte     8+1,32+78
          .byte     6,16
          .word     dook
          .word     cancelbox
          .byte     8+17,32+78
          .byte     6,16
          .word     docancel
;
pgbox:    .byte     3,$ff
          .byte     220+4,14,128+3,$80,$00,$01
          .byte     3,$ff
;
okbox:    .byte     $05,$ff,$82,$fe,$80,$04,$00,$82
          .byte     $03,$80,$04,$00,$b8,$03,$80,$00
          .byte     $f8,$c6,$00,$03,$80,$01,$8c,$cc
          .byte     $00,$03,$80,$01,$8c,$d8,$00,$03
          .byte     $80,$01,$8c,$f0,$00,$03,$80,$01
          .byte     $8c,$e0,$00,$03,$80,$01,$8c,$f0
          .byte     $00,$03,$80,$01,$8c,$d8,$00,$03
          .byte     $80,$01,$8c,$cc,$00,$03,$80,$00
          .byte     $f8,$c6,$00,$03,$80,$04,$00,$82
          .byte     $03,$80,$04,$00,$81,$03,$06,$ff
          .byte     $81,$7f,$05,$ff
;
cancelbox:
          .byte     $05,$ff,$82,$fe,$80,$04,$00,$82
          .byte     $03,$80,$04,$00,$b8,$03,$87,$c0
          .byte     $00,$00,$00,$e3,$8c,$60,$00,$00
          .byte     $00,$63,$8c,$07,$9f,$1e,$3c,$63
          .byte     $8c,$0c,$dd,$b3,$66,$63,$8c,$07
          .byte     $d9,$b0,$66,$63,$8c,$0c,$d9,$b0
          .byte     $7e,$63,$8c,$0c,$d9,$b0,$60,$63
          .byte     $8c,$6c,$d9,$b3,$66,$63,$87,$c7
          .byte     $d9,$9e,$3c,$63,$80,$04,$00,$82
          .byte     $03,$80,$04,$00,$81,$03,$06,$ff
          .byte     $81,$7f,$05,$ff
;
writecl:  .byte     "Write Image",0
;
writeinfo:
          .word     outnm
          .byte     3,21,$80+63
          .byte     $ff,$ff,$ff,$80,$00,$01,$8f,$ff
          .byte     $01,$88,$01,$01,$8b,$ff,$c1,$8a
          .byte     $00,$41,$8a,$ff,$f1,$8a,$80,$11
          .byte     $8a,$8e,$11,$8a,$80,$11,$8a,$bf
          .byte     $91,$8a,$80,$11,$8a,$9f,$11,$8a
          .byte     $80,$11,$8a,$bf,$91,$8e,$80,$11
          .byte     $82,$bf,$91,$83,$80,$11,$80,$80
          .byte     $11,$80,$ff,$f1,$ff,$ff,$ff
          .byte     $83,$07,$01
          .word     0
          .word     $ffff
          .word     0
          .byte     "Write Image V"
ovsnl:    .byte     "?.?",0,0,0,0
          .byte     "by: Combiner V1.0",0,0,0
          .byte     "geoWrite    V?.?",0,0,0,0
          .word     1               ;used by V2.x
          .byte     0
          .word     0,0,768
wblk:                               ;save blocks for each INPUT data
fpblk:    .byte     "xx",0,32+14-2
          .block    1               ;$87cf
          .block    2               ;$84be-84bf
lpblk:    .byte     "xx",0,32+38-2
          .block    3
apblk:    .byte     "xx",0,32+62-2
          .block    3
dmyx:     .block    23-(dmyx-wblk) ;padding to 23 bytes
inpnm:    .byte     0
          .block    16
outnm:    .byte     0
          .block    16
idsknm:   .block    16         ;disk name of input disk
          .byte     0
dadsknm:  .block    16         ;disk name of D.A. disk
          .byte     0
;
patch:    .block    30         ;program patch area
;
picused:  .block      64  ;table to convert input picture # to output
idskdrv:  .block      1   ;drive # of input disk
odskdrv:  .block      1   ;drive # of output disk
dadskdrv: .block      1   ;drive # 0f D.A. disk
xsave:    .block      1   ;temp save area for x reg
picinx:   .block      1   ;index into input picsused
picoutx:  .block      1   ;index into output picsused
ivflag:   .block      1   ;input version flag (0,1,2,3=none,1.x,2.0,2.1)
ovflag:   .block      1   ;output version flag
oldpages: .block      1   ;# chains in output file
oldpics:  .block      1   ;# pictures in output file
oldmax:   .block      1   ;max # pages available in output file
oldnew:   .block      1   ;00=old output/ff=new output/7f=need create
max:      .block      1   ;used by count recs
pages:    .block      1
pics:     .block      1
frstipge: .block      1   ;1st page to start with
lastipge: .block      1   ;last page to merge
aftpge:   .block      1   ;merge after this page
helpidx:  .block      1   ;help screen index
backlvlf: .block      1   ;00=normal; $80=2.1->2.0; $c0=2.x->1.x
widxsave: .block      1   ;save of where window index
dan1:     .block      17
dan2:     .block      17
dan3:     .block      17
dan4:     .block      17
dan5:     .block      17
dan6:     .block      17
dan7:     .block      17
dan8:     .block      17
;
bufbeg:                              ;start of buffer area
bufend       ==      $8000-2-(27-20) ;allow room for eob flags + ruler expansion
;
          .end
; END of $2COMBINER
```

%COMBINER - *Linker statements for Combiner*

```
; %COMBINER - Link statements for geoWrite Combiner - Nick Vrtis - 1/89
;
.output combiner
.header $HCOMBINER.rel
.seq
.psect $0400
$1COMBINER.rel
$2COMBINER.rel
```

Clean Machine Language Screens

Techniques for text output routines

by Bill Brier

Displaying text on the screen is the most common of program activities and is usually one of the first things learned by beginning programmers. Those moving up from BASIC to machine language may initially find it somewhat difficult to print attractive screen displays. The handy PRINT statement and the companion TAB and SPC formatting commands vanish once the realm of the BASIC interpreter has been left behind.

Fortunately, the freedom of expression inherent in machine language makes it possible to write custom versions of PRINT. Using some simple programming techniques, you will be able to handle text output to the screen with ease, and at the same time realize a tremendous increase in display speed. So, if you are ready to learn, please read on!

In describing some of the techniques I've learned, I'll assume that you know what an assembler is and how to use one. My examples will be given in the MOS Technology standard assembler syntax supported by the Commodore 64 assembler and the **C-128 Developer's Package** *HCD65* assembler. Some of this stuff may be old hat to experienced machine language progammers, but even they may find something useful here.

Displaying text strings

Let's start with something simple. In BASIC, you type the command **print "text string"** and the interpreter obediently prints **text string** for you. BASIC figures out where in memory (RAM) the text is located, the number of characters to print and so forth. The price for this convenience is speed (a lack of it).

In machine language, there isn't an interpreter to figure out where the string is at and what to do with it. So, you have to make your own PRINT statement. The first thing to consider in designing your own version of PRINT is how to determine the length of the string. This is readily handled by terminating the string with a zero ($00) byte. Fortunately, the Commodore screen editor considers the null byte to be... well, nothing as far as printing goes. So, to store the text string "text string" in memory, you would code as follows in your assembler:

```
  string  .byt 'text string',0
```

When the assembler parses this line it will generate bytes that are the PETASCII equivalents of the string. Immediately following the string will be the null byte. We can output the string with a simple loop:

```
print   ldy #0            ;starting string offset
                          ;
print1  lda string,y      ;fetch byte from text string
        beq print2      ;the zero byte means we're done
                        ;
        jsr bsout       ;this kernal routine is located
                        ;at $ffd2
        iny             ;point at next character
        bne print1      ;do it again
                        ;
print2  rts             ;exit
```

The Kernal BSOUT routine outputs the character in the microprocessor accumulator (.A register) to the current output device. BSOUT is 'non-destructive' in that it leaves the .A, .X and .Y registers unchanged. It normally exits with the carry bit cleared in the status register. By the way, the above routine works because the act of loading .A with a zero will make the BEQ PRINT2 instruction succeed. If .A contains any other value it will be passed to BSOUT.

The trouble with this PRINT subroutine is that it will only output the text string located at STRING. So, to make the routine a little more generic, we'll modify it to use some zero-page pointers and we'll also modify it to handle text strings that occupy more than 256 bytes of RAM (the above routine will abort when .Y wraps around to zero at the INY instruction). Here's our new, improved PRINT subroutine (funny thing about that name, eh?):

```
print   stx ptr         ;set up zero page pointer to...
        sty ptr+1       ;the text string to output
        ldy #0          ;starting offset
                        ;
print1  lda (ptr),y     ;fetch a character
        beq print2      ;zero byte...we're done
```

```
                    ;
        jsr bsout   ;output via $ffd2
        iny         ;next character
        bne print1  ;loop
                    ;
        inc ptr+1   ;now in the next page of ram
        bne print1  ;loop
                    ;
print2  rts         ;exit
```

To use this routine you would code the following:

```
        ldx #<string;fetch lo byte of string address
        ldy #>string;fetch hi byte of string address
        jsr print   ;output the string
```

Now we can output text by simply giving the subroutine the starting address of the string. Some programmers like to pass such things as text string addresses via the stack. I personally fail to see why. The PLAs and PHAs required to do that are enough to confuse even the most experienced programmers, gain little in execution speed, and are likely to introduce hard-to-solve bugs if everything isn't right.

Ok, now that we have a generic PRINT routine, it's time to introduce some more text string output techniques.

Jockeying for position

Most displays require that text be placed at certain locations on the screen. Such things as menus and columns of numbers look best when neatly aligned top to bottom and centered between margins. Fortunately, the Kernal includes a useful cursor positioning function that can be readily called from your program. Located at $fff0 in the Kernal jump table is the PLOT subroutine, the primary cursor control mechanism built into the screen editor. (One of the truly unique features of the eight-bit Commodore machines is their flexible, easy-to-use full-screen editors. In particular, the C128 editor is amazingly powerful for a machine that purports to be a 'home computer'!)

PLOT actually serves two purposes, depending on whether the carry bit is set or cleared. If carry is cleared when PLOT is called, the cursor will be positioned at the row number given in .X and the column given in .Y (row and column coordinates always start at 0,0 and in the C128 editor, location 0,0 is relative to the currently defined window, not the screen). On the other hand, if the carry bit is set, then the current cursor position will be returned by PLOT, the row in the .X register and the column in the .Y register. I call such an operation "logging the cursor position". Let's write some code to avail ourselves of these functions:

```
;set cursor position
                    ;
plota   clc         ;carry clear to set position
        .byt $24    ;fall thru, call plot routine
                    ;
;log cursor position
                    ;
plotb   sec         ;set carry to log position
        jmp plot    ;located at $fff0
```

The PLOTB subroutine directly follows the PLOTA subroutine (PLOTA falls through into PLOTB). A JSR PLOTA instruction will move the cursor to the row and column specified by the .X and .Y registers. For example:

```
    ldx #row
    ldy #column
    jsr plota
```

This sequence sets the cursor position. If the return from PLOT leaves the carry bit set, then the position command was not executed (usually because the coordinates lie outside the limits of the screen). Calling the subroutine at PLOTB will result in the current cursor position being returned in .X and .Y without disturbing the cursor itself (it won't move). This works like the POS statement in BASIC.

I know you're wondering about that .BYT $24 business in the PLOTA subroutine. It's a simple method of getting the processor to jump past the SEC instruction in PLOTB. Let's suppose that this little routine is assembled at $2000. The resulting code would be disassemble in a machine language monitor like so:

```
.d 2000 2003

.2000  18        clc
.2001  24 38     bit $38
.2003  4c f0 ff  jmp $fff0
```

Hmmm, you say. It doesn't look like what I wrote in my source file. Actually, all of the bytes are there, but appear different because of the $24 byte (which is a zero-page BIT instruction). When the processor executes the instructions starting at $2000 it will clear carry and then execute a harmless BIT operation on location $38 in zero-page RAM (BIT leaves .A, .X and .Y untouched and doesn't affect carry). The byte $38 that acts as the operand for the BIT instruction is really the SEC instruction that was assembled into PLOTB. Now, let's disassemble the routine at $2002:

```
.d 2002 2003

.2002  38        sec
.2003  4c f0 ff  jmp $fff0
```

Do you see how it works? Good! The technique of using a BIT op-code to cover some other code is a commonly used one (take a look at the Kernal itself for some examples of this trick). By the way, even though PLOT is actually a subroutine, the JMP PLOT sequence is correct. When the RTS at the end of PLOT is executed, the processor will return back to the point in your program that called PLOTA or PLOTB.

Now that we can position the cursor to any location on the screen, we will move forward in our quest for clean screens.

Fetching text strings via a look-up table

Ah, yes... that bane of easy programming: the look-up table! A suprisingly large number of machine language programmers avoid the use of look-up tables, probably because they don't understand them or fail to see any practical use for them. Well, there's really no good reason to avoid the use of look-up tables. Once a few simple principles are understood, working with look-up tables becomes as easy as pressing the Return key. As for incorporating a look-up table in your software; well, I'll show you how to use one. Let's first see what a small look-up table looks like:

```
    txttab .wor strng1,strng2,strng3,strng4
```

In MOS assembler syntax, the directive (pseudo-op) .WOR (.WORD) causes the assembler to generate two bytes equal to the low-byte/high-byte address of the referenced locations (data words) following the .WOR directive itself. CBM assemblers allow you to assemble several words on one code line by separating them with commas. Let's suppose that the referenced strings have the following addresses:

```
  strng1 = $2000
  strng2 = $2027
  strng3 = $2641
  strng4 = $316a
```

When the assembler parses the TXTTAB look-up table, it will generate the following output bytes (I've arranged them in two columns to make it more obvious as to what happens):

```
    $00 $20
    $27 $20
    $41 $26
    $6A $31
```

The .WOR directive results in a series of two-byte pointers being assembled in low-byte/high-byte order, each pointing to a location where a text string is lying in wait. The entire table starts at the address location referenced by the TXTTAB label. The distance in memory between successive text string pointers is always two bytes, even though the strings themselves are scattered all over the map. Incidentally, most assemblers have a directive (usually .DBYTE) that assembles such pointers in high-byte/low-byte order. The resulting addresses are usually used as jump vectors.

Ok, big deal, you say. I've got a bunch of addresses in memory that point to text strings. What good is that? The beauty of this arrangement is that you can fetch the string pointer by passing a one-byte index value to a decoding subroutine, which then leaves the other two processor registers free to do something else (such as position the cursor). In effect, you can say, "Give me the pointers to string #14" and the combination of the look-up table and a simple decoding routine will do just that.

Let's write the decoding subroutine:

```
loctxt  sec             ;set carry to subtract
        sbc #1          ;reduce index by one
        asl a           ;double the value in .a
        bcs errext      ;the index is out of range
                        ;
        tay             ;becomes an offset
        lda txttab+1,y  ;fetch pointer hi byte
        ldx txttab,y    ;fetch pointer lo byte
        tay             ;give hi byte to .y
                        ;
errext  rts             ;exit
```

You call this routine by loading .A with the relative position of the string in the table and the routine comes back with the string pointer in .X and .Y (which may then be used in the PRINT subroutine). If you call LOCTXT with 1 in the accumulator (which means you want the first of the four referenced string pointers, STRNG1 in this case), it will be decremented to 0 and then doubled in the ASL A op-code. Since zero times anything is zero, the resulting offset will be zero and the first and second bytes in the look-up table will be fetched. Thus, .X and .Y will contain $00 and $20 respectively.

If you call LOCTXT with 3 in .A (meaning that you want the pointers to STRNG3), the subroutine will return .X = $41 .Y = $26, pointing to STRNG3. Why? Let's 'single step' through the routine to see what happens to the index:

```
                                 Value in
Executed Code                  .A    .X    .Y
----------------------------------------------
loctxt sec                     $03    *     *
       sbc #1                  $02    *     *
       asl a                   $04    *     *
       tay                     $04    *   $04
       lda txttab+1,y          $26    *   $04
       ldx txttab,y            $41   $41  $04
       tay                     $26   $41  $26
       rts                     $26   $41  $26
----------------------------------------------
```

The asterisk (*) indicates that the registers contain indeterminate values. Can you see how we got the correct pointers with a single byte index in .A? If TXTTAB is assembled at $3000, the LDA TXTTAB+1,Y instruction in effect becomes LDA $3000+$01+$04. So we get the low-order byte from $3004 and the high-order byte from $3005.

The above decoding technique will work on a table with a maximum of 128 entries. For that reason, a range check is included to avoid exceeding the 128 entry limitation. If carry is set when the ASL instruction is executed, the supplied index was in the range of 129 to 255 or was zero (an index of zero is undefined and therefore not allowed). Can you figure out why 128 entries is the limit with this technique?

Just to show you that the look-up table is a handy thing to incorporate into your software, we'll utilize TXTAB and LOCTXT together with PLOTA and PRINT and make an all-purpose text string printing function. We'll name this routine PRNTXT and call it as follows:

```
        lda #index   ;1, 2, 3, etc. (text string
                     ;index)
        ldx #row     ;row to print the text on
        ldy #column  ;column to start printing at
        jsr prntxt   ;do it
        bcs error    ;something went wrong
```

Now for the actual PRNTXT subroutine:

```
prntxt  pha          ;stash text string index
        jsr plota    ;position cursor
        pla          ;recover index
        bcs abort    ;coordinates are out of range
                     ;
        jsr loctxt   ;fetch text string pointers
        bcs abort    ;the index was out of range
                     ;
        jmp print    ;output the text
                     ;
abort   rts          ;exit
```

How's that for an efficient, all-purpose print routine? If the routine exits with the carry cleared then everything went as expected (upon exiting from the PRINT subroutine the carry is cleared). By the way, because we directly control the cursor position by using PLOT (rather than by utilizing cursor position characters and spaces) the speed at which the screen display is brought up is dramatically improved. On a C64, the screens will seem to appear like magic.

Now for C128 speed tip number one: The 128's screen editor has its own jump table starting at $c000. Rather than outputting to the screen via the BSOUT subroutine at $ffd2, it's much faster to go directly to the editor at $c00c. BSOUT performs time-consuming checks to determine which peripheral is the designated output device. If you already know where you want the display to appear, why get tangled up in a bunch of output device tests? And, while you're busy bypassing the Kernal jump table, you might want to consider using PLOT at $c018, rather than at $fff0 (the code at $fff0 simply jumps to $c018). These are legitimate jump table entry points (they're documented in the *C128 Programmer's Reference Guide*) and thus should be stable even though Commodore may revise the Kernal at a future date.

How to order up menus

It's my humble opinion that menu-driven programs are much friendlier than those that make the user remember operating codes and strange keypress sequences. So, let's look at some menu-handling techniques.

Menus give you an opportunity to make attractive and easy-to-read screens. However, in machine language you don't have those convenient TAB, SPC and cursor movement commands that BASIC provides. Fortunately, the PLOT subroutine (more specifically, our PLOTA subroutine) can be used to provide the necessary cursor movements.

Let's design a simple "do-nothing" menu:

```
1. Do Operation #1

2. Do Operation #2

3. Do Operation #3

4. Do Operation #4
```

We have several approaches available to us in coding this menu and outputting it to the screen. The first approach would be to assemble four separate text strings (you could call them MENUA, MENUB and so forth), enter them into the look-up table TXTTAB and then repetitively call the PRNTXT subroutine to output them. Each call to PRNTXT would require that you supply the string index and the row and column coordinates. If you had a menu with eight lines, you'd have to make eight calls to PRNTXT. That's a lot of code just to output a menu.

A second approach, if you have a C128, is to define a window whose top left corner is in the same location as the top left corner where the menu is expected to start. Then, each line would end with a carriage return and extra linefeed, with only the last line ending with a zero byte terminator. However, not everybody has a C128, so not everybody has a window command to work with. Besides, the 128's editor is relatively slow when it is expected to figure out the cursor position via the use of carriage returns, cursor movement characters and linefeeds.

A better way (and the one that I like to use) is to imbed the cursor position coordinates within the text itself. Obviously, that means that we can't use our PRNTXT subroutine for such a menu, as it wouldn't know what to do with the cursor coordinates. We can, however, use PLOTA to position the cursor and LOCTXT to look up the text pointers.

The nice thing about imbedding the coordinates in the text is that when examining the source code, it is easy to determine just where on the screen the menu is expected to appear. Knowing that makes it easier to change the menu's location if desired. If we were to utilize PRNTXT and a bunch of individual text strings, we'd have a lot more work cut out for us. So, let's implement the third method by imbedding cursor position coordinates into the text itself. Here is the menu again, but this time with the row and column positions noted:

ROW	COL	
6	35	Menu Title
9	31	1. Do Operation #1
11	31	2. Do Operation #2
13	31	3. Do Operation #3
15	31	4. Do Operation #4

The coordinates shown above center the menu on an 80-column display (side to side, as well as top to bottom). The title appears centered on row 6. On a 40-column screen, just subtract 20 from the above column values to center the display.

To imbed the cursor coordinates into the text you would code as follows:

```
mentxt  .byt 0,6,35,'Menu Title'
        .byt 0,9,31,'1. Do Operation #1'
        .byt 0,11,31,'2. Do Operation #2'
        .byt 0,13,31,'3. Do Operation #3'
        .byt 0,15,31,'4. Do Operation #4',128
```

Let's look a little closer at this gibberish. Each line is started with a zero byte, followed by the row and column coordinates, which are then followed by the text itself. The value 128 acts as the terminator for the entire menu (128 is not a printing character nor is 0). The trick to this technique is that when we encounter a zero byte we know that the next two bytes are row and column coordinates, not text. When we encounter the 128 byte, we know that the entire menu has been displayed.

You are not limited to just text and coordinates with this technique. You can also imbed colour values at any point in the text. You can imbed characters to turn reverse video on or off. In fact, any charcter may be imbedded as long as it will not be recognized as one of the two possible delimiting values (0 or 128).

To work with this type of format, we need a character-fetching subroutine and a subroutine to take care of outputting characters and positioning the cursor. First the character fetching subroutine:

```
chrgt   lda chrgt+1  ;fetch a character
        inc chrgt+1  ;step text pointer lo
        bne chrgt1
                     ;
        inc chrgt+2  ;step text pointer hi
            ;
chrgt1  pha  ;stash on stack
        pla  ;recover character
            ;
        rts
```

Yeah, I know...it's a strange-looking routine, but it does work. This is an example of self-modifying code. The "text pointer" is actually the operand of the LDA instruction at CHRGT. When this subroutine is called, the text pointers will have been stored at CHRGT+1 (lo byte) and CHRGT+2 (hi byte). After fetching the character the text pointer is incremented. This means that the next call to CHRGT will fetch the next character. Finally, the character is pushed on the stack and then immediately pulled back off the stack. The purpose of pushing and then pulling the character to and from the stack is simply to condition the Z flag in the status register before exiting.

Now for the main routine, which we'll call TXTP. It is invoked as follows:

```
lda #index   ;index to text string
jsr txtp
bcs error    ;something is not right
```

If you wanted to display the menu and the look-up table TXTTAB had the following entries:

```
txttab  .wor strng1,strng2,strng3,strng4
        .wor mentxt,strng5,strng6,strng7
```

You would code:

```
lda #5
jsr txtp
bcs error
```

MENTXT (the pointer to the menu) is the fifth item in the look-up table. Note that TXTP is called in the same manner as the PRNTXT routine except that you don't pass any cursor coordinates. Here's the TXTP subroutine:

```
txtp    jsr loctxt ;convert index into text pointers
        bcs txtp04;index out of range
        ;
        stx chrgt+1  ;set up beginning text pointers
        sty chrgt+2
        ;
txtp01  jsr chrgt    ;fetch a character
        beq txtp02   ;zero byte, cursor coordinates
                     ;follow
        ;
        cmp #128     ;final terminator
        beq txtp03   ;exit
        ;
        jsr bsout    ;output the character
        bcc txtp01   ;go get the next one
        ;
txtp02  jsr chrgt    ;fetch row
        tax
        jsr chrgt    ;fetch column
        tay
        jsr plota    ;position cursor
        bcc txtp01   ;resume
```

www.Commodore.ca
May Not Reprint Without Permission

```
        ;
        .byt $24   ;cursor coordinates out of range
        ;
txtp03  clc        ;no error
        ;
txtp04  rts        ;exit
```

Looks like a lot of code just to print menus? Not really! Don't forget, this routine takes care of cursor positioning and everything. All you have to do is enter the menu into the look-up table and pass the index in the accumulator. If you come out of TXTP with the carry set, then something was amiss (either the index or the cursor coordinates were out of range).

Make it easy for yourself

One of the things I do when planning a program where a lot of menus and such will be used is to lay out the screen on graph paper (TOPS form 3304 is perfect for this purpose). I position the text just as I want it to appear on the screen, and I include notes about color, reverse video and so forth. This allows me to visualize the finished product and avoid such contretemps as off-center titles or text running off the edge of the screen.

Once I have worked out my screen I can read the text right off the paper version and type it into the program, cursor coordinates, colours and all. I suggest that you adopt this method when writing your software. I call it easy programming!

NOTHING LOADS YOUR PROGRAMS FASTER

THE QUICK BROWN BOX
A NEW CONCEPT IN COMMODORE CARTRIDGES

Store up to 30 of your favorite programs — Basic & M/L, Games & Utilities, Word Processors & Terminals — in a single battery-backed cartridge. READY TO RUN AT THE TOUCH OF A KEY. HUNDREDS OF TIMES FASTER THAN DISK. Change contents as often as you wish. The QBB accepts most unprotected programs including "The Write Stuff" the only word processor that stores your text as you type. Use as a permanent RAM-DISK, a protected work area, an autoboot utility. Includes utilities for C64 and C-128 mode.

Packages available with "The Write Stuff," "Ultraterm III," "QDisk" (CP/M RAM Disk), or QBB Utilities Disk. Price: 32K $99; 64K $129. (+$3 S/H; $5 overseas air; Mass residents add 5%). 1 Year Warranty. Brown Boxes, Inc, 26 Concord Rd, Bedford, MA 01730: (617) 275-0090; 862-3675

Quick Brown Box

NEW!

VIDEO BYTE the first FULL COLOR! video digitizer for the C-64, C-128

Introducing the world's first **FULL COLOR!** video digitizer for the Commodore C-64, C-128 & 128-D computer.
VIDEO BYTE can give you digitized video from your V.C.R., B/W or COLOR CAMERA or LIVE VIDEO (thanks to a fast! 2.2 sec. scan time).

- **FULL COLORIZING!** Is possible, due to a unique SELECT and INSERT color process, where you can select one of 15 COLORS and insert that color into one of 4 GRAY SCALES. This process will give you over 32,000 different color combinations to use in your video pictures.
- **SAVES as KOALAS!** Video Byte allows you to save all your pictures to disk as FULL COLOR KOALA'S. After which (using Koala or suitable program) you can go in and redraw or recolor your Video Byte pic's.
- **LOAD and RE-DISPLAY!** Video Byte allows you to load and re-display all Video Byte pictures from inside Video Byte's menu.
- **MENU DRIVEN!** Video Byte comes with an easy to use menu driven UTILITY DISK and digitizer program.*
- **COMPACT!** Video Byte's hardware is compact! In fact no bigger than your average cartridge! Video Byte comes with its own cable.
- **INTEGRATED!** Video Byte is designed to be used with or without EXPLODE! V4.1 color cartridge. Explode! V4.1 is the perfect companion.
- **FREE!** Video Byte users are automatically sent FREE SOFTWARE updates along with new documentation, when it becomes available.
- **PRINT!** Video Byte will printout pictures to most printers. However when used with Explode! V4.1 your printout's can be done in FULL COLOR on the **RAINBOW NX-1000, RAINBOW NX-1000 C, JX-80 and the OKIDATA 10 / 20.**

Why DRAW a car, airplane, person or for that matter . . . anything when you can BYTE it . . . Video Byte it instead.

VIDEO BYTE $79.95

SUPER EXPLODE! V4.1 w/COLOR DUMP

If your looking for a CARTRIDGE which can CAPTURE ANY SCREEN, PRINTS ALL HI-RES and TEXT SCREENS in FULL COLOR to the RAINBOW NX-1000, RAINBOW NX-1000 C, EPSON JX-80 and the OKIDATA 10 or 20. Prints in 16 gray scale to all other printers. Comes with the world's FASTEST SAVE and LOAD routines in a cartridge or a dual SEQ., PRG. file reader. Plus a built-in 8 SECOND format and MUCH, MUCH MORE! Than Explode! V4.1 is for you.
PRICE? $44.95 + S/H or $49.95 w/optional disable switch.

* IN 64 MODE ONLY

VIDEO BYTE only $79.95
SUPER EXPLODE! V4.1 $44.95
PLUS $1.50 S/H C.O.D.'S ADD $4.00
IL RESIDENTS ADD 6% SALES TAX

TO ORDER CALL 1-312-851-6667
Personal Checks 10 Days to Clear

THE SOFT GROUP, P.O. BOX 111, MONTGOMERY, IL 60538

Notice to Subscribers

Renew early to avoid missing an issue! Check the expiry volume and issue on your mailing label, and make sure you renew your subscription a couple of issues in advance. For faster processing of your order, please pay by cheque or money order.

Notice to Non-Subscribers

Fill in the subscription card in the centre of the magazine and subscribe now! Get the most authoritative technical journal for the Commodore 8-bit computers delivered right to your door.

Ride Your 4040 On The Serial Bus

IEEE-to-serial bus conversion for the 4040

by Michael Gilsdorf

Ever since I bought my C128 I always wished I could use all of BASIC 4.0 commands. Top on my wish list was being able to copy files from one drive to another with the COPY and BACKUP commands. However, since neither the 1541, 1571, or 1581 support drive 1, copying files required loading and running a copy program. A few months ago I had the opportunity to acquire a 4040 dual drive with an IEEE interface. The price was right so I took the plunge.

Unfortunately the IEEE interface had some problems. It worked fine in C64 mode, but refused to operate in C128 native mode. Also, it would not work with the SX-64 or the VIC-20. Since the interface patched into the computer's operating system via the cartridge port, some programs would neither run nor support it. I was constantly plugging and unplugging the interface, and found myself unable to use the drive when I most wanted to.

What I needed was another interface - one that would offer the maximum compatibility and operate with my other Commodore computers. Since speed was a secondary concern, the best interface, I thought, would be one that connected the 4040's IEEE connector to the Commodore's serial bus. However after contacting several Commodore suppliers and second party vendors, it appeared no one made this type of interface. I finally decided to interface the drive to the serial bus myself.

My goal was to connect the 4040 to the serial bus using the minimum amount of hardware. It seemed to me that since the 4040 was an intelligent drive, it ought to be able to be programmed to respond to the serial bus signals. If so, then there would be no need of an external interface equipped with its own CPU, ROM, RAM, I/O, and other support chips. I hoped that I could connect the 4040 to the serial bus by having to replace only a ROM chip and the connecting cable.

As it turned out, an additional NAND chip was required since a direct pin-for-pin EPROM replacement was not available for the 4040 ROM. Once having completed the conversion process, I did find the 4040 a little faster than the 1541. But don't expect the 4040 to work with disk speed-up cartridges or some commercial software. They generally are not written to operate with the 4040 disk operating system (DOS).

The following is a list of the items you'll need to make the conversion - but first, a word of caution. If you are not technically inclined (or don't know what end of a solding iron to hold) I strongly recommend you do *not* attempt the modification yourself! Acquire the skills of an electronic technician experienced with digital hardware. Should you damage your 4040 through carelessness, you may find it difficult to find an technician experienced in repairing the drive. So be careful and double check all your work!

Parts List

Quantity	Description
1	2532 EPROM chip
1	74LS00 NAND chip
1	6-pin male DIN plug

The next material list is recommended, but not absolutely essential to perform the conversion. (More about these items later.)

1	14-pin IC socket for the 74LS00 NAND chip
1	IEEE-488 connector and cable
1	6-pin female DIN plug

(Note: All parts for this project are available at Radio Shack with the exception of the 2532 EPROM and IEEE connector cable.)

One nice thing about this modification is you can still revert back to the original IEEE interface, if you so desire. Simply swap the EPROM chip with the original CBM ROM (along with a cable change-out) and your drive is back to its IEEE configuration. Removing the NAND chip is not necessary. By the way, if you're interested in complete, documented source code for the 4040 drive, *The Anatomy of the 4040* by Hilaire Gagne is an excellent book. *[Hilaire Gagne, 4501 Carl St., P.O. Box 278, Hanmer, ON, Canada, POM 1Y0. For Canadian residents, $39.95 (Cdn) plus $3 shipping and handling; for U.S. residents, $31.95 (US) plus $9 shipping and handling.]* Discussions with Hilaire strongly hint that a similiar modification may be possible for the 8050, 8250, and 9060 drives since their source code is quite similiar to the 4040. The ROM chips used in these other drives appear to be 8K. If true, modifying these drives for serial operation may involve only a ROM and cable swap!

Step I - Software Modification

We'll begin the modification by programming the EPROM. For this you'll need an EPROM programmer (e.g., **Promenade**). Begin by unplugging the 4040's AC power cord, and disconnect the IEEE connector/cable on the back of the drive. Next, locate and remove the two screws on either side of the drive. You'll find them situated at the bottom near the front face. Now open the top cover to the drive by lifting and swinging up the cover from the front to the back. The motherboard, which we'll be working on, is mounted to the underside of the top cover. Carefully unplug the four cables on the motherboard. Notice that three of the cables are keyed to prevent them from being connected improperly. However, the two-wire cable for the error LED could be reconnected improperly. So be sure to label/mark it to show its orientation. Next locate and remove the six Phillips screws used to mount the motherboard to the top cover. Carefully remove the motherboard.

Now find the 24-pin socketed ROM chip located near the bottom of the motherboard at position UJ1 (part no. 901468-14). Most chips have their locations labeled on the motherboard, but (as luck would have it) UJ1 is not marked (see Fig. 2). The ROM at UJ1 contains 4K of DOS code ($d000-$dfff), and is located between the other two socketed ROM chips at UL1 ($e000-$efff) and UH1 ($f000-$ffff). Take careful note of the position of the key on the ROM chip. It should be oriented towards the bottom of the motherboard. (The same direction as the other two socketed 24-pin ROM chips on either side.) It indicates which way the ROM is to be reinserted into its socket. You'll need this information when installing the EPROM. After removing the ROM chip, set the motherboard aside for now. We'll make the hardware modifications to it later.

Next, using an EPROM programmer, read and copy the ROM code into the computer's memory. We'll be patching our serial bus routines into the original 4040 code. You might want to store the code on disk for future reference. If you're using a Promenade the set-up for the 2532 EPROM will allow you to read the ROM chip. The main difference between the 2532 EPROM and the ROM chip is that the ROM chip has two chip select (CS) lines (pin nos. 20 and 21) as opposed to one CS line (pin no. 20) for the 2532 EPROM. The ROM chip is selected when pin 21 is low and pin 20 is high. After reading the ROM, we are now ready to make the patch.

Using the C128 built-in monitor (or other means) replace sections of the ROM code with the new serial bus code (see listing). After making the patches, run the code check program and verify that the code is error free. If the code is correct, then program the EPROM with the new code. This completes the software portion of the modification.

Step II - Hardware Modification

First, we'll make up the IEEE-to-serial cable. This cable will be constructed with an IEEE-488 connector on one end and a six-pin (male) serial bus connector on the other end. For this you can use the original Commodore IEEE cable, but you'll need to cut off the flat edge 24-pin connector. Since this may not be desirable (if you ever intend to use this cable again), I recommend you use a different cable. If you're lucky you might be able to pick up a used one at a HAM fest or surplus store. However, if all else fails, you can purchase a new IEEE-488 connector and cable for about $50. (One possible source is L-com Inc. located in N. Andover, MA).

Once you've decided on an IEEE cable, solder a 6-pin DIN (male) plug to the end. I also recommend that you solder a 6-pin (female) DIN plug alongside (parallel to) the male DIN plug. This will allow you to connect an extra device (e.g., printer) to the serial bus. The table below shows how the pins are to be connected. When finished it's a good idea to check each connection with an ohmmeter (or DVM). Pay special attention when soldering the wires to the DIN plug. It's easy to bridge solder across a couple of pins and cause a short!

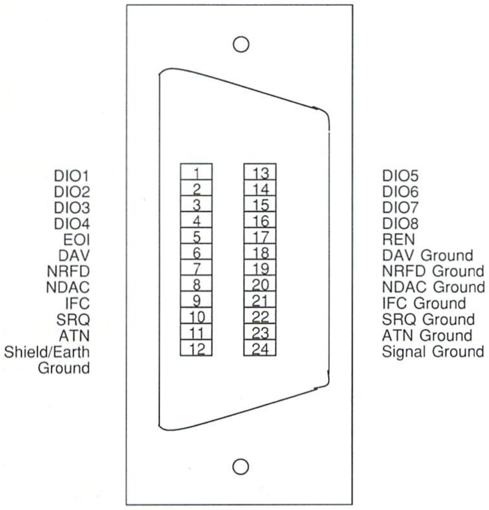
IEEE Connector		6-pin DIN Connector(s)
Pin 5 (EOI)	to	Pin 5 (Data line)
Pin 6 (DAV)	to	Pin 4 (Clock line)
Pin 9 (IFC)	to	Pin 6 (Reset line)
Pin 11 (ATN)	to	Pin 3 (Attention line)
Pin 12 (Shield)	to	Shield of DIN plug
Pins 18,23,24 (GND)	to	Pin 2 (Ground)

(Note: Pin numbers are usually stamped on the connectors. See Figure 1)

Next install the 74LS00 NAND chip. I recommend that you install a 14-pin IC socket for this chip. This will make it easy to remove if the need ever arises. There are few places on the motherboard that can accommodate an additional chip - none have their locations labeled. I chose UA5 located between locations UA4 and UA6 which should be marked (see Fig. 3). Notice that UA5 is designed to accept a 16-pin chip, while the 74LS00 is a 14-pin chip. Be sure the key locator on the chip is oriented in the proper direction, and lines up with the chip key printed on the motherboard. This will position the chip towards the bottom of the motherboard leaving two empty IC pin holes at the top.

Before starting the final wiring, a quick review in pin numbering is in order. To determine pin numbers, look at the top side of the chip with the key up and facing you and pins pointing away. Pin 1 is located directly on the left at the top. The other pins are numbered counter-clockwise around the chip. This information is commonly pictured in many IC reference books. Also a word of caution - remember that IC pin numbers are numbered clockwise when viewed from the bottom of the motherboard - so be careful! Armed with this knowledge, you're ready to proceed with the final wiring.

IEEE Connector Pins



Serial Bus Connector Pins

Pin #	Name	Description
1	SRQ	Serial SRQ in (active low)
2	GND	System Ground
3	ATN	Serial ATN In/Out
4	CLK	Serial Clock In/Out
5	DATA	Serial Data In/Out
6	RESET	Resets all devices on serial bus (active low)

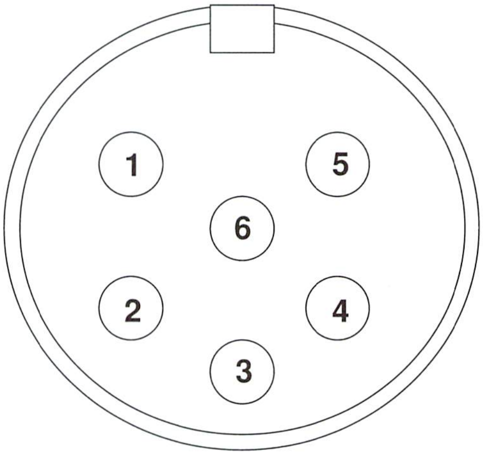


Figure 1: *Connector pinout diagrams*

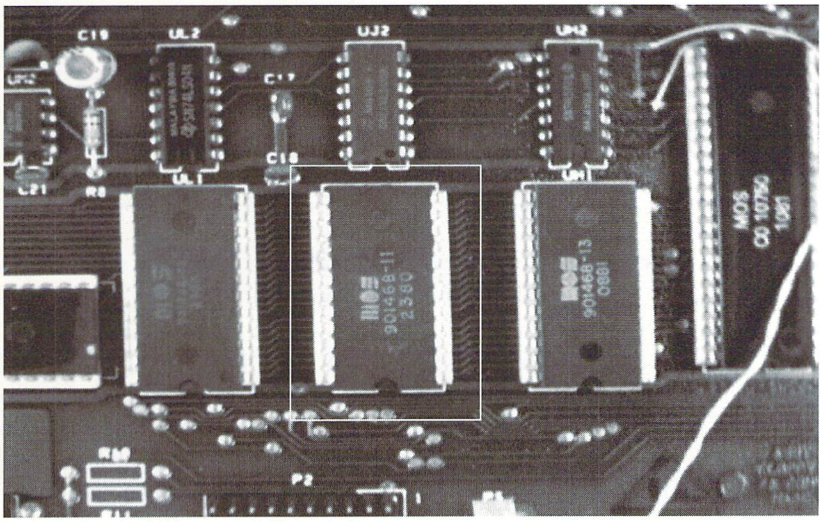


Figure 2: *Locating UJ1*

Locate the three ROM chips at the bottom of the motherboard. The one in the middle is UJ1. Note that although the article refers to this ROM as part no. 901468-14, the chip is designated as 901468-11 in the 4040 used by *Transactor* for this illustration. The white wire is a device number switch.

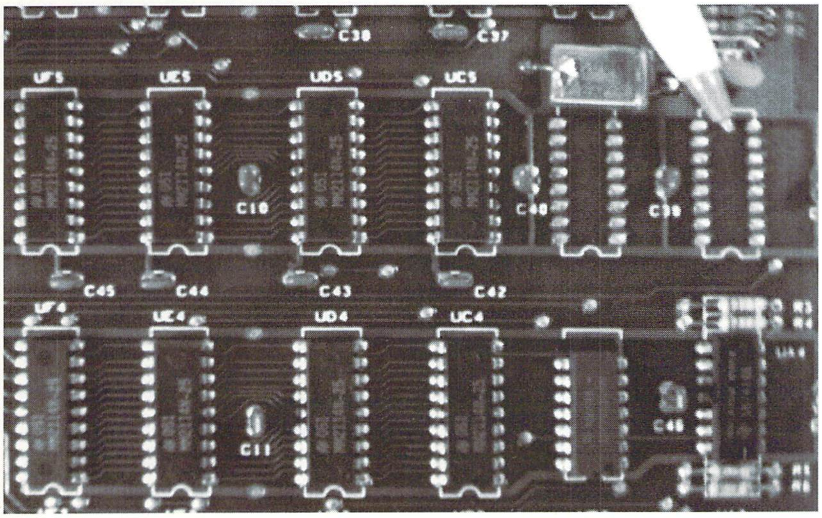


Figure 3: *Chip Location*

Install a 14-pin IC socket in UA5 (indicated by the pen). This socket will hold the 74LS00 NAND chip. Ensure that the key locator on the chip is oriented in the proper direction and lines up with the chip key printed on the motherboard.

Once the NAND chip is in place, jumper pins 1 and 2, then separately jumper pins 3 and 4. Next, jumper pin 7 of the 74LS00 to the adjacent empty IC pin hole. This hole, which would normally be pin 8 for a 16-pin socket, connects to the ground bus etching on the motherboard. Now all that is left is to connect the NAND chip to the 2532 chip and socket with wires. Thirty gauge insulated wire-wrap (or hook-up) wire works well for this.

Run one wire and connect the 74LS00 pins 1 and 2 to pin 20 of the empty 24-pin ROM socket (UJ1). (This is the socket from which we removed the ROM chip.) Run a second wire and connect pin 5 of the 74LS00 to pin 21 on the same empty ROM socket. After double checking your wiring and connections, you are now ready for the EPROM.

Before inserting the EPROM chip, bend out pin 20. This pin will not be inserted into its socket. Carefully insert the EPROM into the empty ROM socket and check to be sure the chip key is positioned in the same direction as it was for the ROM. (Should be the same as the keys on the other two 24-pin ROM chips on either side.) After inserting the EPROM, solder one wire on pin 20 of the EPROM and connect it to pin 6 on the 74LS00.

Before final reassembly, you may want to change the 4040's device number. The device number is determined by whether pins 22, 23, and 24 on the 6532 chip (located at UE1) are grounded or open-circuited (floating). For device 8, the pins are all grounded by a tiny etched tracing connecting a pair of adjacent 'half moon' etchings. These 'half moons' are located between UE1 and UH2. The device number is changed by open-circuiting the pins. This can be accomplished by cutting the trace(s), or bending out pins 22, 23, and/or 24 (preferred). Another alternative, if you think you'll be changing the device number again in the future, is to install switches across the cut half moons. The table below shows how the device number relates to the various pin combinations.

Device Number	Pin 22	Pin 23	Pin 24
8	Ground	Ground	Ground
9	Ground	Ground	Open
10	Ground	Open	Ground
11	Ground	Open	Open
12	Open	Ground	Ground
13	Open	Ground	Open
14	Open	Open	Ground
15	Open	Open	Open

Now, reinstall the motherboard to the top cover using the six Phillips screws. Do *not* overtighten the screws or you may crack the motherboard! Carefully plug in the four cables on the motherboard, and install the IEEE-to-serial cable on the back of the drive. Close the top cover and reinstall the two remaining screws on either side of the drive. This completes the hardware modification.

Final check-out

With the DIN plug *not* connected, insert the power cord and power-up the 4040 drive. If you performed the modification correctly, you should see the drive's LEDs light momentarily. You won't hear the familiar head chatter on power-up. That has been eliminated. If the LEDs continue to flash, or didn't momentarily light, remove the drive's power cord immediately and double check all your wiring. Flashing LEDs could indicate a hardware problem with the EPROM circuit (wiring, connections, etc.). The most likely cause is connections made to the wrong pin numbers. Closely re-check all wiring and correct as necessary. Another possible cause is an error in the EPROM code or wrong checksum.

If the drive initialized correctly, turn off the drive, connect the DIN plug to the serial bus, and power up the computer and the 4040 drive. For now, disconnect any other devices you have from the serial bus. Following initialization, try loading the directory and loading/saving a program. If this works correctly, then reconnect any other devices you have to the bus and check them all for proper operation. (Note: Some devices may cause problems with serial bus operation if they are left connected to the bus while turned off.)

Well, thats it! Hopefully everything went smoothly and your 4040 is riding the serial bus. If you have any questions or comments, I can be reached on QLink as "Mike All". Until then...easy DOS it!

Patch code for serial bus 4040

```
; Serial Bus Conversion Code for 4040 Drive
; By Michael Gilsdorf - Copyright (c) Mar 89


0d2a0  e0       ???           ;checksum byte

0d2ee  08       ???           ;gap same as 1541

; You may wish to keep the original gap of $09.
; If so, change checksum byte to $df.

0d339  a9 18    lda #$18      ;reset bus lines

0d468  20 73 d6 jsr $d673     ;eliminate head chatter on error

0d492  a9 00    lda #$00      ;initialize track

0d49a  a2 c1    ldx #$c1      ;eliminate head chatter on power-up,
0d49c  4c a3 d4 jmp $d4a3     ;reset, and "UJ" command


; Idle Loop Patch

0d4a6  ad 47 43 lda $4347     ;command waiting?
0d4a9  f0 07    beq $d4b2     ;no
0d4ab  78       sei           ;disable interrupts
0d4ac  20 a6 d6 jsr $d6a6     ;set Data low for future ATN ack
0d4af  4c 79 d6 jmp $d679     ;DOS command execution patch
0d4b2  20 50 d6 jsr $d650     ;if ATN pending then service bus
0d4b5  20 94 d6 jsr $d694     ;no ATN - reset bus lines

0d507  4c b2 d4 jmp $d4b2     ;continue idle looping
```

```
; Main Serial Bus Routine

0d50a  78        sei         ;disable interrupts
0d50b  20 a6 d6  jsr $d6a6   ;set Data line low - ATN ack
0d50e  ad 87 02  lda $0287   ;clear interrupt register
0d511  a2 ff     ldx #$ff    ;reset the
0d513  9a        txs         ;stack pointer
0d514  20 97 d6  jsr $d697   ;set Clock line high - release
0d517  2c 80 02  bit $0280   ;read serial bus
0d51a  10 66     bpl $d582   ;jump if ATN gone (high)
0d51c  50 f9     bvc $d517   ;wait for Clock line high
0d51e  20 c0 d5  jsr $d5c0   ;read bus command byte
0d521  aa        tax         ;save command byte
0d522  c5 0c     cmp $0c     ;LISTEN?
0d524  d0 06     bne $d52c   ;no
0d526  85 0e     sta $0e     ;set listen flag active
0d528  84 0f     sty $0f     ;set talk flag inactive
0d52a  f0 08     beq $d534   ;jump (always)
0d52c  c5 0d     cmp $0d     ;TALK?
0d52e  d0 0e     bne $d53e   ;no
0d530  85 0f     sta $0f     ;set talk flag active
0d532  84 0e     sty $0e     ;set listen flag inactive
0d534  a9 20     lda #$20    ;set for internal (default)
0d536  85 16     sta $16     ;secondary address and
0d538  85 17     sta $17     ;original secondary address
0d53a  85 10     sta $10     ;set primary address flag active
0d53c  d0 1b     bne $d559   ;jump (always)
0d53e  29 60     and #$60    ;mask command byte
0d540  c9 60     cmp #$60    ;SECONDARY ADDRESS?
0d542  d0 23     bne $d567   ;no
0d544  a5 10     lda $10     ;is primary address flag active?
0d546  f0 1f     beq $d567   ;no
0d548  8a        txa         ;retrieve command byte
0d549  85 17     sta $17     ;save byte as original secondary address
0d54b  29 0f     and #$0f    ;mask command byte
0d54d  85 16     sta $16     ;save as secondary address
0d54f  8a        txa         ;retrieve command byte
0d550  29 f0     and #$f0    ;mask command byte
0d552  c9 e0     cmp #$e0    ;CLOSE?
0d554  d0 27     bne $d57d   ;no
0d556  20 8d f5  jsr $f58d   ;execute Close
0d559  2c 80 02  bit $0280   ;read serial bus
0d55c  30 c0     bmi $d51e   ;jump if ATN line low - get next byte
0d55e  10 22     bpl $d582   ;jump if ATN line high

; Enable Interrupts - Check for Pending ATN

0d560  58        cli         ;enable interrupts
0d561  2c 80 02  bit $0280   ;is ATN line low?
0d564  30 a4     bmi $d50a   ;yes - service bus
0d566  60        rts

; Main Serial Bus Routine (continued)

0d567  8a        txa         ;retrieve command byte
0d568  c9 3f     cmp #$3f    ;UNLISTEN?
0d56a  d0 04     bne $d570   ;no
0d56c  84 0e     sty $0e     ;set listen flag inactive
0d56e  f0 06     beq $d576   ;jump (always)
0d570  c9 5f     cmp #$5f    ;UNTALK?
0d572  d0 06     bne $d57a   ;no
0d574  84 0f     sty $0f     ;set talk flag inactive
0d576  84 10     sty $10     ;set primary address flag inactive
0d578  f0 03     beq $d57d   ;jump (always)
0d57a  20 94 d6  jsr $d694   ;reset serial bus
0d57d  2c 80 02  bit $0280   ;read serial bus
0d580  30 fb     bmi $d57d   ;wait for ATN line high
0d582  58        cli         ;enable interrupts
0d583  a5 0e     lda $0e     ;is listen flag active?
0d585  f0 06     beq $d58d   ;no
0d587  20 a7 d5  jsr $d5a7   ;execute Listen
0d58a  18        clc
0d58b  90 0d     bcc $d59a   ;jump (always)
0d58d  a5 0f     lda $0f     ;is talk flag active?
0d58f  f0 09     beq $d59a   ;no
0d591  20 9a d6  jsr $d69a   ;set Data line high
0d594  20 a3 d6  jsr $d6a3   ;set Clock line low
0d597  20 04 d6  jsr $d604   ;execute Talk
0d59a  4c a6 d4  jmp $d4a6   ;execute DOS command/idle loop

; Main Listen Routine

0d59d  20 c0 d5  jsr $d5c0   ;read byte from serial bus
0d5a0  78        sei         ;disable interrupts
0d5a1  20 f8 eb  jsr $ebf8   ;write byte to buffer/disk
0d5a4  20 60 d5  jsr $d560   ;enable interrupts - check ATN
0d5a7  20 84 ed  jsr $ed84   ;open channel for writing
0d5aa  b0 05     bcs $d5b1   ;jump if channel not open
0d5ac  b5 98     lda $98,x   ;check channel status
0d5ae  6a        ror         ;is channel set for writing?
0d5af  b0 ec     bcs $d59d   ;yes
0d5b1  a5 17     lda $17     ;retrieve original secondary address
0d5b3  29 f0     and #$f0    ;mask original secondary address
0d5b5  c9 f0     cmp #$f0    ;OPEN?
0d5b7  f0 e4     beq $d59d   ;yes
0d5b9  a5 16     lda $16     ;retrieve secondary address
0d5bb  c5 01     cmp $01     ;is secondary address set for Save?
0d5bd  f0 de     beq $d59d   ;yes
0d5bf  60        rts

; Read Byte from Bus

0d5c0  20 89 d6  jsr $d689   ;read bus
0d5c3  90 fb     bcc $d5c0   ;wait for Clock line high
0d5c5  a9 ff     lda #$ff    ;set the
0d5c7  aa        tax         ;timer/counter, and
0d5c8  a8        tay         ;set EOI status to no
0d5c9  20 9a d6  jsr $d69a   ;set Data line high
0d5cc  20 89 d6  jsr $d689   ;read bus
0d5cf  90 14     bcc $d5e5   ;wait for Clock line high
0d5d1  ca        dex         ;is timer/counter still running?
0d5d2  d0 f8     bne $d5cc   ;yes - no EOI yet
0d5d4  20 a6 d6  jsr $d6a6   ;set Data line low
0d5d7  c8        iny         ;set EOI status to yes
0d5d8  a2 0a     ldx #$0a    ;set timer/counter
0d5da  ca        dex         ;is timer/counter still running?
0d5db  d0 fd     bne $d5da   ;yes
0d5dd  20 9a d6  jsr $d69a   ;set Data line high
0d5e0  20 89 d6  jsr $d689   ;read bus
0d5e3  b0 fb     bcs $d5e0   ;wait for Clock line low
0d5e5  84 a0     sty $a0     ;set EOI flag
0d5e7  a0 08     ldy #$08    ;set for 8 bits per byte
0d5e9  2c 80 02  bit $0280   ;read bus
0d5ec  50 fb     bvc $d5e9   ;wait for Data line high
0d5ee  ad 80 02  lda $0280   ;read bus for data
0d5f1  0a        asl
0d5f2  0a        asl
0d5f3  0a        asl
0d5f4  66 18     ror $18     ;save data bit
0d5f6  20 89 d6  jsr $d689   ;read bus
0d5f9  b0 fb     bcs $d5f6   ;wait for Clock line low
0d5fb  88        dey         ;read bus for more bits?
0d5fc  d0 eb     bne $d5e9   ;yes
0d5fe  20 a6 d6  jsr $d6a6   ;set Data line low
0d601  a5 18     lda $18     ;retrieve data byte sent
0d603  60        rts

; Main Talk Routine

0d604  20 69 ed  jsr $ed69   ;open channel for reading
0d607  90 63     bcc $d66c   ;jump if channel open
0d609  60        rts
```

```
; Send Byte to Bus

0d60a  20 89 d6 jsr $d689     ;read bus
0d60d  08       php           ;save status
0d60e  20 97 d6 jsr $d697     ;set Clock line high
0d611  28       plp           ;retrieve status - is Data line high?
0d612  30 0d    bmi $d621     ;yes - byte not sent (EOI)
0d614  20 89 d6 jsr $d689     ;read bus
0d617  10 fb    bpl $d614     ;wait for Data line high
0d619  a6 15    ldx $15       ;retrieve channel index
0d61b  b5 98    lda $98,x     ;get channel status
0d61d  29 08    and #$08      ;is channel status EOI?
0d61f  d0 0a    bne $d62b     ;no
0d621  20 89 d6 jsr $d689     ;read bus
0d624  10 fb    bpl $d621     ;wait for Data line high
0d626  20 89 d6 jsr $d689     ;read bus
0d629  30 fb    bmi $d626     ;wait for Data line low
0d62b  20 a3 d6 jsr $d6a3     ;set Clock line low
0d62e  20 89 d6 jsr $d689     ;read bus
0d631  10 fb    bpl $d62e     ;wait for Data line high
0d633  a0 08    ldy #$08      ;set for 8 bits per byte
0d635  20 89 d6 jsr $d689     ;read bus
0d638  10 38    bpl $d672     ;jump if Data line low - abort
0d63a  a6 15    ldx $15       ;get channel index
0d63c  76 b5    ror $b5,x     ;fetch data bit to send
0d63e  90 06    bcc $d646     ;is data bit 0?
0d640  20 9a d6 jsr $d69a     ;no - send data bit 1
0d643  18       clc
0d644  90 03    bcc $d649     ;jump (always)
0d646  20 a6 d6 jsr $d6a6     ;send data bit 0
0d649  20 97 d6 jsr $d697     ;set Clock line high - data ready
0d64c  a2 0f    ldx #$0f      ;set timer/counter (delay)
0d64e  ca       dex           ;is timer/counter running?
0d64f  d0 fd    bne $d64e     ;yes
0d651  20 a3 d6 jsr $d6a3     ;set Clock line low
0d654  20 9a d6 jsr $d69a     ;set Data line high
0d657  88       dey           ;more data bits to send?
0d658  d0 db    bne $d635     ;yes
0d65a  20 89 d6 jsr $d689     ;read bus
0d65d  30 fb    bmi $d65a     ;wait for Data line low
0d65f  78       sei           ;disable interrupts
0d660  20 a6 d6 jsr $d6a6     ;set Data line low - for future ATN ack
0d663  20 a3 ef jsr $efa3     ;get next data byte from buffer/disk
0d666  20 60 d5 jsr $d560     ;enable interrupts - check ATN
0d669  20 9a d6 jsr $d69a     ;set Data line high
0d66c  a6 15    ldx $15       ;get channel index
0d66e  b5 98    lda $98,x     ;is channel set for reading?
0d670  30 98    bmi $d60a     ;yes
0d672  60       rts

; No Head Chatter on Error Patch

0d673  09 80    ora #$80
0d675  8d 5c 43 sta $435c
0d678  60       rts

; DOS Command Execution Patch

0d679  a9 00    lda #$00      ;clear the
0d67b  8d 47 43 sta $4347     ;command waiting flag and the
0d67e  8d f2 10 sta $10f2     ;NMI flag
0d681  b8       clv
0d682  18       clc
0d683  20 55 db jsr $db55     ;execute DOS command in command buffer
0d686  4c b2 d4 jmp $d4b2     ;back to idle loop

; Read Serial Bus

0d689  ad 80 02 lda $0280     ;read serial bus
0d68c  cd 80 02 cmp $0280     ;has bus settled?
0d68f  d0 f8    bne $d689     ;no
0d691  0a       asl
0d692  0a       asl           ;Clock bit in Carry - Data bit in bit #7
0d693  60       rts
```

```
; Set Bus Line(s) High

0d694  a9 18    lda #$18      ;reset bus
0d696  2c a9 10 bit $10a9     ;set CLOCK line high
0d699  2c a9 08 bit $08a9     ;set DATA line high
0d69c  0d 80 02 ora $0280     ;read bus
0d69f  8d 80 02 sta $0280     ;set bus
0d6a2  60       rts

; Set Bus Line(s) Low

0d6a3  a9 ef    lda #$ef      ;set CLOCK line low
0d6a5  2c a9 f7 bit $f7a9     ;set DATA line low
0d6a8  2d 80 02 and $0280     ;read bus
0d6ab  8d 80 02 sta $0280     ;set bus
0d6ae  60       rts


0d8a5  b3       ???           ;Change version no. to 3

; Error Patch

;This patch allows the 4040 to be used with CP/M+. Unlike the 1541, the 4040
;DOS does not auto-initialize on a disk swap or on receiving a "#" command.
;The error patch initializes the disk (reads BAM & ID) once, when an error is
;encountered.


0d949  ae 00 43 ldx $4300     ;is cmd buffer & disk initialized?
0d94c  f0 52    beq $d9a0     ;yes
0d94e  48       pha           ;save error number
0d94f  4c 96 d9 jmp $d996     ;patch
0d952  68       pla
0d953  4c 49 d9 jmp $d949     ;patch

; Error Recovery Code
  (same as original code, but has IEEE code removed)

0d978  f0 19    beq $d993
0d97a  a5 0e    lda $0e       ;Listen flag active?
0d97c  d0 0a    bne $d988     ;yes
0d97e  a5 0f    lda $0f       ;Talk flag active?
0d980  f0 11    beq $d993     ;no
0d982  20 69 ed jsr $ed69
0d985  4c 8b d9 jmp $d98b
0d988  20 84 ed jsr $ed84
0d98b  20 a1 ed jsr $eda1
0d98e  b0 03    bcs $d993
0d990  20 9f ee jsr $ee9f
0d993  4c a6 d4 jmp $d4a6     ;back to idle loop

; Error Patch (continued)

0d996  20 b8 db jsr $dbb8     ;initialize (clear) cmd buffer
0d999  68       pla           ;retrieve error number
0d99a  8d 01 43 sta $4301     ;save error number
0d99d  20 fa ec jsr $ecfa     ;initialize disk - read BAM & ID
0d9a0  ad 01 43 lda $4301     ;retrieve original error number
0d9a3  20 d4 d9 jsr $d9d4     ;write error no. and msg into buffer
0d9a6  d0 ae    bne $d956     ;jump (always)
```

Code Check Program - written to check code residing at $d000 to $dfff.

```
10 bank0: rem for the c128 only - change bank no. as required
20 for i=53248 to 57343: rem $d000 to $dfff - change if needed
30 b=b+peek(i): if b>255 then b=b-255
40 next: if b=208 then print "Code OK - Program EPROM": end
50 print "Error in code!  Correct and recheck the code before
          programming the EPROM."
```

Colour Coordination

Why some combinations work while others don't

by Jim Butterfield

Computers that use TV sets, or monitors with a "composite video" connection, are often accused of rendering some colours poorly. The accusation is often unfounded: it's usually the video system itself that's at fault. In this article, I'll try to give you a quick run-down on why this is. An accompanying chart may help you choose colour schemes with good readability.

The Video Concept

Whether your video system is NTSC, as used in North America, or PAL, as used in much of the rest of the world, it has a built-in anomaly: there is no detail in colour. Any detail you need on your screen must be created with a change in luminance (brightness) rather than a change in chrominance (colour).

When television systems were being designed, there were sound reasons for this. Tests showed that people can not see colour within detail; only the broad areas of a picture convey colour information to the eyes. In order to save channel space, the television system was designed to drop colour information from intricate parts of the picture.

To be more precise: the brightness part of a picture (the black-and-white portion, if you like) is sent complete with sharp detail. The colour information of a picture is sent with much less sharpness. It's easy to demonstrate this on a Commodore Plus/4 or C-16 computer; the HUE command will change the brightness level of any selected colour, except black. No matter what basic colours you choose on these machines, you can also pick a hue that will make the text unreadable against the background colour.

Here's the trick in setting up a good, readable screen: choose colours with good luminance differences. The chart will help.

Horrible Examples

Look at the chart, and note that a Commodore 64 starts up with a background colour of blue (128 code 7, POKE value 6). Brown happens to have exactly the same luminance level as blue; on your Commodore 64, type **print chr$(149)** (or select brown on the keyboard with Commodore-2) and then try to type a readable line. Horrible, isn't it? Yet you can rescue that colour by putting it against a background that has a contrasting luminance level. Let's pick green, and **poke 53281,5**. Even though brown and green are not considered harmonious, except in trees, the washed-out brown characters suddenly become crisp and readable. Simultaneously, the earlier light blue text that may be on the screen becomes washed out and virtually unreadable; it no longer has enough contrast against the green background.

The same thing may be found on the 128. When you turn on the power, the dark grey background is actually a little brighter than 64 blue. **Print chr$(28)** (or select red on the keyboardwith CTRL-3 and then try to type something legible. Next switch the background to yellow with **color 0,8**; the red text will now be fine, but the startup-message (in light green) will wash away.

Refer to the chart and pick a colour that has a luminance level one group away from whatever background you are using. Try typing; you'll find the characters are readable, but not as crisp as you might like.

Practical Applications

If you write a program in which you pick one or more character colours, you should be sure that such colours are separated from the background by at least two groups. That way, you should get good readability.

Can you trust the background colours to be the "default" values? Probably not. If you're going to set colours at all, you might as well set the whole thing: background, border, and character. Your program might follow on from somebody else's masterpiece, in which the background colour has been set to something completely incompatible to your colour selection.

Then again, you can leave colours alone completely, on the assumption that the user will have set colours to a personally pleasing palette. Side Issues.

Keep in mind that the table and the above description apply only to the TV-like signals: television itself, of course, and monitors taking composite video signals. If you hook up an 80-column display to your 128 using the RGBI cable, the problem will not arise. Colour will be delivered to the same sharpness as black-and-white.

All colour combinations will work together. Except, of course, such combinations as blue on blue, which is, as always, very hard to see.

It's interesting to note how we tend to blame the computer for such problems, when the problem is in the video methodology. Questions such as "How can I fix my video chip?" can't be answered easily, since the problem is not in the chip.

In the same way, interlace pictures as seen with some 128 programs and with the Amiga have an unsolvable flicker problem. The flicker is not in the computer: it comes about as a result of the nature of television signals.

To get rid of the flicker, you must pursue the same drastic solution as for 'fuzzy colours': you would abandon the standard TV signal and go to a special monitor.

		128Color	POKE	Character	CHR$()
	White	2	1	E	5
	Yellow	8	7	π	158
	L.Green	14	13	▐	153
	Cyan	4	3	◣	159
	L. Grey	16	15	✚	155
	Green	6	5	↑	30
	L. Red	11	10	✕	150
	M. Grey	13	12	♣	152
	L. Blue	15	14	◆	154
	Purple	5	4	▚	156
	Orange	9	8	♠	129
	Red	3	2	£	28
*128	D. Grey	12	11	○	151
*64	Blue	7	6	←	31
	Brown	10	9	╭	149
	Black	1	0	┐	144

*Shows default background colour of computer.

The Table

The table is a convenient thing to keep on hand. Colours are grouped in descending order of luminance, from white to black. The 128 COLOR command numeric value is given, plus the POKE value which is valid for both 128 and 64. If you're looking at a listing, the symbols that you'll see when colours are selected via 'programmed cursor' are shown. And finally, the CHR$() values are given; I like to use these when setting colour within a program, since they are easier to typeset (and read, and enter) than the reverse-character equivalents. I don't mind if your colours clash. But if you're going to fly that multi-coloured sprite against a split screen with both hi-res and text in various colours, I'd like it all to be sharp and visible. The table may help. ◘

NewsBRK

CompuServe expanding to Europe: We're pleased to see that CompuServe will soon be available to European users. CompuServe has entered into an agreement with Tele Columbus of Baden, Switzerland. This extension to CIS service will begin in the U.K. and Switzerland with other European countries to follow. European users will be able to tap CIS' vast resources in the fall of this year.

We would be remiss if we failed to note that the Commodore Programming Forum (GO CBMPRG) and the Commodore Communications Forum (GO CBMCOM) are a part of those vast resources. CBMPRG's data libraries contain a large number of public domain, freely redistributable and *Transactor* programs which are provided to support Commodore programmers. CBMCOM is directed more to users of Commodore applications programs, especially terminals. CBMCOM also features an on-line conference each Sunday at 9:00 PM Eastern time.

The Commodore Arts Forum (GO CBMART) is directed to users interested in games, graphics and music. Commodore itself is also accessible via CIS (GO CBMSERV).

Minitel comes to North America: Also on the communications front, North American microcomputer users will be able to reach Minitel, the French information network, for only a local call, using Minitel's free terminal software.

The software connects users with the information network used daily by more than four million people in France. Minitel offers a wide range of services - everything from financial and business transactions to electronic chatlines and the French National Phone Directory. By the end of 1989, Minitel will also become the gateway to similar networks in Belgium, Italy, Spain, Germany, and Finland.

To get the free software, use your modem to call Minitel's toll-free BBS number: 1 (800) 999-6163. Set your parameters to 1200 8N1 and enter "Minitel" at the login prompt. You'll also receive a Directory of Minitel's services at no charge. *[Yes, there is Minitel software for the C64. Prospective users should note that charges amount to 17 cents per minute for some services and 35 cents per minute for others. There is no sign-up fee and no minimum monthly charge. The free software will undoubtedly be available all over the continent soon via local BBSs as well. - MO]*

Free Spirit to market VizaStar & VizaWrite 128: Viza Software and Free Spirit Software have entered into an agreement whereby Free Spirit shall exclusively market *VizaWrite Classic* and *VizaStar 128* in North America. *VizaWrite Classic* is described as a high performance, easy-to-use, word processing program for the C128. *VizaStar 128* is the integrated spreadsheet, database and business graphics program for the C128.

VizaWrite Classic uses page-based WYSIWYG format - word wraps and formats text, instantly, as you type. Editing features include: copy, move, delete text by highlighting a character, word, sentence, paragraph, page or by searching; find and replace any sequence of characters; full screen and document scrolling, up to 240 character page width; go to any page instantly; merge almost any other word processing file directly into a document; glossary area for frequently used words or phrases. Mail merge, a full function calculator and a 30,000 word Spelling Checker are among its many features. An 80-column monitor is required. Free Spirit will market *VizaWrite Classic* at a new suggested retail price of $59.95.

The spreadsheet for *VizaStar 128* contains a ruled worksheet display, a 1000 row by 64 column worksheet, variable column widths, multiple worksheet windows, copy, move, erase functions and more. The database allows full screen design of records (up to nine screens can make up an individual record), up to 8,000 characters per record, unlimited number of records per file and more. The Business Graphics function uses data from the spreadsheet or database to draw two- or three-dimensional full colour graphs and charts. Free Spirit will market *VizaStar 128* at a new suggested retail price of $69.95. For further information, contact: Free Spirit Software, P.O. Box 128 - 58 Noble St., Kutztown, PA, 19530, (215) 683-5609.

Psygnosis moves into the C64 market: Psygnosis - already firmly established in the games market for the Amiga and ST - is now seeking a major slice of the action at the top and bottom ends of the computer entertainment marketplace. The company, whose titles regularly feature in the international Amiga and Atari ST charts, has launched a simultaneous two-pronged attack on the PC and 8-bit games areas. Three Psygnosis games will soon be available for the C64. The new versions are to be released under the Psyclapse label. The following material was taken from the Psygnosis press release:

- **Baal** - "An addictive mixture of strategy and arcade action, it features eight way ultra smooth scrolling through three distinctive domains containing multiple levels, over 250 highly detailed screens, superb graphics and sound effects, and more than 100 monsters and 400 traps."

- **Captain Fizz Meets The Blaster-trons** - "A gripping mixture of high speed play and deep strategy, the game offers simultaneous two player action, split screen view, 20 Blastertron infested levels and a pounding soundtrack."

- **Ballistix** - "Considered the ultimate ball game, it is played on 130 different pitches, with splitters filling the screen with dozens of balls, tunnels to hide them from view, red arows to increase their velocity and magnets to take them out of control. All this and a reverberating soundtrack complemented by crowd applause for every goal."

These new versions will have new style packaging with cover illustrations involving lettering from top British artist Roger Dean. The C64 titles carry a suggested list price of £9.99 (cassette) and £12.99 (disk). *[Sorry, no dollar amounts were given in the press release. - MO]* Jonathan Ellis, Managing Director of Psygnosis, states: "As far as the 8-bit scene is concerned, we are convinced there is a great need for good quality fames. The trouble has been that 8-bit users have tended to be treated as the poor relations of late and so the product they have been offered has not been of a sufficiently high standard. Psygnosis intends to change all that by breathing new life into the market."

The ICT Mini-Chief hard drive returns!: Owners of 1571 disk drives will probably be tickled pink to discover that the Mini-Chief hard drive, originally marketed by the now-defunct InConTrol, is available once again. The hard drive is installed *inside* the 1571 case. Manufacturing rights for the Mini-Chief have been obtained by The Computer Bar, P.O. Box 436, Hagerstown, MD, 21741, (301) 293-7005.

Star Micronics offers 14 resident fonts: Star Micronics America, Inc. has begun shipping the first 24-wire dot matrix printers offering 14 resident fonts, claimed to be the greatest number of internal fonts available in a single dot matrix model. These printers also produce multi-colour output with an optional colour-printing kit. Additionally, the manufacturer contends that the XB-2415 Multi-Font (15") and the XB-2410 (10") Multi-Font printers are the quietest models in their price/performance categories operating at 49 and 50 decibels respectively.

According to Star, these printers offer exceptional speed, print quality, memory capacity, paper-handling capabilities and an easy-to-use front control panel for optimal functionality in business and home office applications. They offer a super letter quality mode (SLQ) in addition to the standard LQ mode featured on today's 24-wire printers. Each model prints at 240 characters per second in draft elite mode and 80 cps in LQ elite mode.

The 14 resident fonts are TMS Roman, TW-Light, Courier, Prestige, Script, Letter G, Orator, Helvet, Optimo, Cinema, Blippo, OCR A, OCR B and Code 39. The SLQ mode is available in two fonts: TMS Roman and TW-Light. In addition, users can expand their font library with optional font cards that will soon be available. The printers also offer superior graphics output by producing 360 by 360 dpi graphics resolution, which surpasses that of most laser printers.

The XB-2415 Multi-Font has an exceptionally large 41K buffer which allows the printer to store up to 20 pages. The large memory capacity frees the computer to handle other processing tasks. The XB-2410 Multi-Font has a 27K buffer and holds up to 13 pages. To expand the memory capacity, users can add an optional 128K parallel board and a 32K *ram* card.

The XB-2415 Multi-Font printer incorporates Epson LQ-1050, IBM Proprinter XL24 and NEC graphics emulations. The XB-2410 Multi-Font incorporates Epson LQ-850, IBM Proprinter X24 and NEC graphics emulations. The printers come standard with a Centronics parallel interface, and an optional 8K serial board with RS-232C and RS-422A interfaces is available.

The front control panel allows users to conveniently select from 21 frequently-used print functions, virtually eliminating the need for DIP switches. By pressing a button, users can engage the paper parking feature, which permits feeding of single sheets without removing tractor-fed fanfold paper. In addition, users can choose fonts, print quality, print pitch, condensed print, italic print, quiet mode, graphics printing direction, among other functions.

When producing output on pre-printed forms and fanfold paper, users can program the printers to skip over the perforation and position the page for a short tear-off. Other printer functions such as page length, lines per inch, automatic line feed and automatic carriage return functions can also be set from the front control panel. In addition, an optional pull tractor and a single-bin cut sheet feeder are available.

The XB-2415 carries a suggested retail price of $999, the XB-2410 is $749. The colour printing kit has a MSRP of $50. Star Micronics Inc., 200 Park Avenue, Suite 3510, New York, NY, 10166, (212) 986-6770.

Clip art for your Commodore: Parsec, Inc. is distributing a new series of Public Domain clip art. This package consists of ten disks filled with over 1,000 pieces of clip art. The clip art is available in either *Basic 8*, *Newsroom*, *PrintMaster*, *Printshop* (Side A) or *Printshop* (Side B) formats. Also included is a booklet, catalogued by disk, for quick location of the graphics.

The clip art series includes everything from hi-tech and cars to nature and sports. The delivered price, including the shipping and handling, is: $13.40 for the 48 states (with a street address); $14.30 for POB addresses, AK, HI; and $16.80 for Canadian orders. The mailing address is: Parsec, Inc., POB 111, Salem, MA, 01920. Parsec can be reached on-line at: CompuServe: 76456,3667; Q-Link: Parsec; GEnie: JBEE

Fourth Annual Commodore Showcase: C.A.S.E (Commodore Association of the South/East) has selected the dates of September 16-17, 1989 to hold the Fourth Annual Commodore Showcase in Nashville, Tennessee. This year's show will be held at the Nashville Convention Center. Educational and fun seminars are given throughout the two days of the show. Last year, Jim Butterfield, R.J. Mical, Jim Oldfield, Pete Baczor and Andre Frech were some of the personalities that presented topics. Tickets for the Showcase will be sold by member clubs at $7.50 each (prior to August 15th, $10.00 thereafter), good for both days of the show.

C.A.S.E. is a consortium of user groups formed to better serve the souteastern community of Commodore computer users by providing education, communication, product information and fellowship to the members. Currently, there are 35 user groups who are members of C.A.S.E. and these groups consist of well over 5,000 Commodore owners and users. For more information: C.A.S.E., P.O. Box 2745, Clarksville, TN, 37042-2745.

***Bible Search* from SOGWAP Software:** *Bible Search* contains the complete King James Version New Testament text. According to SOGWAP, the program is equipped with very fast word search and verse display capabilities. *Bible Search* includes the full text with a complete Concordance on two Commodore disks. The package comes with two versions of the program: a C64 version (40 columns, 64K) and a C128 version (40 or 80, 128K).

The Concordance references every word to every verse in the New Testament, thereby eliminating fruitless searching of text. With *Bible Search*, the user can perform single or multiple word searches and then display the full text of those verses where the word(s) are used together or separately. The manufacturer states that complete verse usage of each search word is returned in about five seconds or less on a C64/1541 and that faster times are possible for less used words and for C128/1571 users.

The text is provided with: book, chapter and verse markings; upper and lower case; full punctuation; italics, and the words of Christ in colour. Display colours and drive usage are configurable - works with one or two drives. Printer output is available for any verse(s).

Bible Search comes complete with User's Guide and is normally supplied on two 1541 flippy disks (1571 and 1581 format available on request; specify when ordering). Both programs are on one disk with the Concordance on the back. The full text of the New Testament has been compressed onto both sides of a second disk. The four Gospels are on one side; Acts through Revelation on the other. *Bible Search* is written by Michael R. Miller (*Big Blue Reader*) and is not copy protected. The package is available through Commodore dealers or direct from SOGWAP at a cost of $25.00. Send cheque or money order to: SOGWAP Software, 115 Bellmont Road, Decatur, IN, 46733, (219) 724-3900.

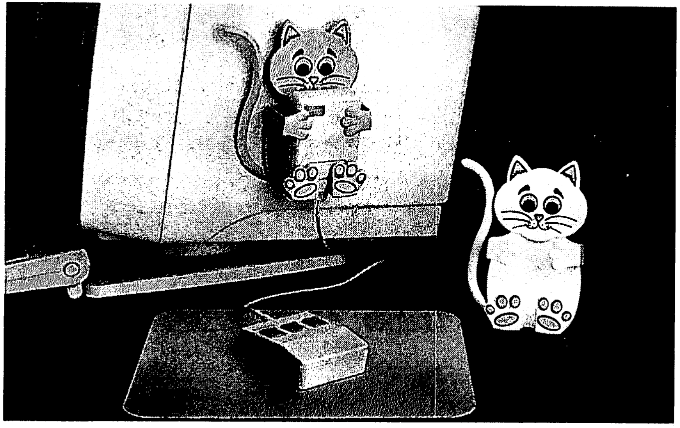
New PD disks for C64/C128: Public Domain Solutions is pleased to announce the release of several new Public Domain disks.The first four new disks (E004-E007) are only for the 128 in 128 mode and comprise 12 Physics lessons. There are three lessons per disk (each lesson is about 180 blocks). This collection sells for $12.00 (US).

The next group of five disks provides telecommunications capabilities to C64 users and supports a variety of modems. Each of the five disks contains the terminal program itself (PCGTERM), a wide variety of fonts and 282 blocks of documentation. Choose the disk that supports your modem: TO52 is for the 1650; TO53 for the 1660; TO54 for the 1670;TO55 for the Volks 6480; and TO56 for the Mitey Mo. The disks are $4.00 each or purchase all five as a set for $15.00.

PDS's 'disks of the month' sell for $4.00 each and include: *April '89/C64:* calculate Social Security benefits, view the pictures that are provided with the commercial game *Strip Poker* (user must have the game to use this program), small SEQ file reader, count number of files on a disk, C64 pictures with a fade in/out slideshow program, and a disk cataloging program. *May '89/C64:* powerful sprite editor from Germany, more pictures, Print Shop graphics (3-block), C64 terminal with VT100 (yes, 80-column) and Kermit protocol. *June '89/C64:* PCGTERM with support for 1650, 1660, 1670, Mitey Mo and Volks 6480 all on one disk. This version doesn't do everything that the version on the individual disks does since the disk had insufficient space for the extra fonts and some other support files. *April '89/C128:* music menu program with many song files, a Star Trek demo, some 1571 utilities, a side 2 recovery program and some menu programs. *May '89/C128:* 203-block checkbook program that operates in 80-column mode, the shareware terminal program DESTERM.SDA (which runs in 80-column mode and supports ANSI graphics).

All prices are USD and include shipping and handling within the USA. Public Domain Solutions, P.O. Box 832, Tallevast, FL, 342701, (813) 378-2394 help and information line, (800) 634-5546 toll free order line.

Purrrfect mouse holder unleashed: H&H Enterprises has introduced the MouseCAT mouse device holder. The mouse holder looks like a kitten and holds the mouse in its lap with its front paws wrapped around the mouse. MouseCAT comes in either light grey or white with pink ears, nose and paws, green eyes and a curling tail. MouseCAT attaches to the computer monitor or other flat surface with a velcro-type fastener. MouseCAT retails for $6.95 (US). For more information, contact H&H Enterprises, 4069 Renate Dr., Las Vegas, NV, 89103. Phone (702) 876-6292.



The Potpourri Disk

Help!

This HELPful utility gives you instant menu-driven access to text files at the touch of a key – while any program is running!

Loan Helper

How much is that loan really going to cost you? Which interest rate can you afford? With Loan Helper, the answers are as close as your friendly 64!

Keyboard

Learning how to play the piano? This handy educational program makes it easy and fun to learn the notes on the keyboard.

Filedump

Examine your disk files FAST with this machine language utility. Handles six formats, including hex, decimal, CBM and true ASCII, WordPro and SpeedScript.

Anagrams

Anagrams lets you unscramble words for crossword puzzles and the like. The program uses a recursive ML subroutine for maximum speed and efficiency.

Life

A FAST machine language version of mathematician John Horton Conway's classic simulation. Set up your own 'colonies' and watch them grow!

War Balloons

Shoot down those evil Nazi War Balloons with your handy Acme Cannon! Don't let them get away!

Von Googol

At last! The mad philosopher, Helga von Googol, brings her own brand of wisdom to the small screen! If this is 'AI', then it just ain't natural!

News

Save the money you spend on those supermarket tabloids – this program will generate equally convincing headline copy – for free!

Wrd

The ultimate in easy-to-use data base programs. WRD lets you quickly and simply create, examine and edit just about any data. Comes with sample file.

Quiz

Trivia fanatics and students alike will have fun with this program, which gives you multiple choice tests on material you have entered with the WRD program.

AHA! Lander

AHA!'s great lunar lander program. Use either joystick or keyboard to compete against yourself or up to 8 other players. Watch out for space mines!

Bag the Elves

A cute little arcade-style game; capture the elves in the bag as quickly as you can – but don't get the good elf!

Blackjack

The most flexible blackjack simulation you'll find anywhere. Set up your favourite rule variations for doubling, surrendering and splitting the deck.

File Compare

Which of those two files you just created is the most recent version? With this great utility you'll never be left wondering.

Ghoul Dogs

Arcade maniacs look out! You'll need all your dexterity to handle this wicked joystick-buster! These mad dog-monsters from space are not for novices!

Octagons

Just the thing for you Mensa types. Octagons is a challenging puzzle of the mind. Four levels of play, and a tough 'memory' variation for real experts!

Backstreets

A nifty arcade game, 100% machine language, that helps you learn the typewriter keyboard while you play! Unlike any typing program you've seen!

All the above programs, just $17.95 US, $19.95 Canadian. No, not EACH of the above programs, ALL of the above programs, on a single disk, accessed independently or from a menu, with built-in menu-driven help and fast-loader.

**The ENTIRE POTPOURRI COLLECTION
JUST $17.95 US!!**

See Order Card at Center

THE WORLD OF COMMODORE

The Computer Show For Everyone!

Featuring

AMIGA • C-64 • C-128 • PCs

September 22, 23 and 24, 1989

Valley Forge Convention Center
Valley Forge, Pennsylvania

Welcome to a spectacular world of Commodore computing — a world devoted to the **Amiga, C-64, C-128 and Commodore PCs.** You'll discover the software you've always wanted to try, plus amazing, new programs. You'll find printers and plotters. Modems and monitors. Disk drives and joy sticks. Lasers and light pens. MIDI and mice. All the big and little stuff that make computing more productive, more creative — more fun!
And some of the best bargains you'll find anywhere!

It's all in one place — at the 2nd annual North-Eastern World of Commodore. Whether you compute for business or fun, at home or school, you can't miss this computer show!

THE WORLD OF COMMODORE
September 22, 23 and 24, 1989
Fri. noon–8pm/Sat.&Sun. 10am-5pm
Admission $10 Students/Seniors $8
Includes seminars & stage demonstrations

Exhibitors contact: The Hunter Group **(416) 595-5906** Fax (416) 595-5093. Produced in association with Commodore Business Machines.