THE MAGAZINE FOR COMMODORE PROGRAMMERS FEBRUARY 1989: VOLUME 9, ISSUE 5

Canada $4.25
USA $3.50

Transactor

- Non-destructive windows on the C128
- Trouble-free 2400 Baud serial communication on the C64
- A DOS wedge for the C64 in ROM
- Far-Sys: execute machine code in *any* RAM area
- An easy-to-build parallel printer in terface for the C128
- Special centrespread bonus: GEOS label reference chart
- Machine language routines for game programming
- Serial I/O routines in Power C
- All about the C128's BANK command
- **Product Reviews**: Z3PLUS for CP/M, JiffyDOS, SWL shortwave decoder, The ZR2 Hardware interfacing chip
- **Plus** Regular columns by Todd Heimarck and Joel Rubin, Programming tips in *Bits,* and more

Dreamtime by Wayne Schmidt

02
0 29074 12121

DATEL COMPUTERS

FINAL CARTRIDGE III

- Give your C64/128 an "Amiga Look-Alike" environment. Pull down windows/icons.
- PLUS a huge range of utilities all built into this one cartridge . . . including:
- Freezer option to freeze and copy most any memory resident program.
- Full machine code monitor with all the usual commands and more.
- Unique picture print dump – compressed/enlarged Plus other effects.
- Fast loader – loads commercial software at up to 15 times normal speed.
- Range of desk-top type tools – clock, calculator, desktools etc. etc.
- Final Cartridge 3 is a superb utilities cartridge with an added bonus of a freezer function.

ONLY $54.99

TOOLKIT IV

The ultimate disk toolkit for the 1540/1541

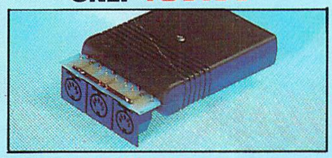
- A disk toolkit is an absolute must for the serious disk hacker. Toolkit IV has more features than most for less.
- DISK DOCTOR V2 – Read and write any track and sector including extra and renumbered tracks. Repair damaged sectors. Look underneath read errors.
- HEADER/GAP EDITOR – Decodes and displays ALL header information including off bytes and header gap. Rewrite the entire header and header gap. Renumber sectors. Also edit any sector tail gap.
- DISK LOOK – Sort directory. Recover lost files. Display file start/end addresses. Disassemble any file program directly from the disk to SCREEN or PRINTER including undocumented opcodes. Edit Bam. Much, much more.
- FILE COMPACTOR – Can compact machine programs by up to 50%. Saves disk space. Compacted programs run as normal.
- FAST DISK COPY – Copy an entire disk in 2 minutes or less using single 1541.
- FAST FILE COPY – Selective file copy. Works at up to 6 times normal speed.
- FORMATTER – 10 second format an entire disk or format any individual track or half track 0 to 41. Redefine any of 30 parameters to create or recreate unique disk formats.
- ERROR EDIT – Quickly find and recreate all read errors including extra and renumbered tracks or sectors and half tracks from 0 to 41. Even recreates data under errors and allows you to redefine any necessary parameters.

ONLY $14.99

DIGITAL SOUND SAMPLER

- The new sampler allows you to record any sound digitally into memory and then replay it with astounding effects.
- Playback forwards/backwards with echo/reverb/ring modulation.
- Full 8 bit D to A & ADC conversion.
- Midi compatible with suitable Midi interface. (see Ad')
- Full sound editing and realtime effects menu with waveforms
- Powerful sequencer plus load/save functions
- Line In/Mic In/Line Out/Feedback controls.
- Complete hardware/software package

ONLY $89.99



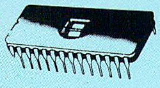
MIDI 64

- Full specification MIDI interface at a realistic Price.
- MIDI IN
- MIDI OUT
- MIDI THRU

Compatible with most leading MIDI software.

ONLY $49.99

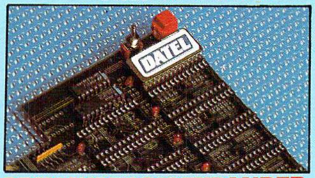
TURBO ROM II



Turbo Rom II is a replacement for the actual kernal inside your 64. It provides superfast load/save routines.

- Loads/Saves most programs at 5-6 times normal speed.
- Improved DOS support including 10 sec format.
- Programmed function keys:- load, directory, old, etc.
- Return to normal kernal at flick of a switch.
- FCOPY – 250 block file copier.
- FLOAD – special I/O loader.
- Plus lots more.
- Fitted in minutes – no soldering usually required. (On some 64's the old ROM may have be desoldered).

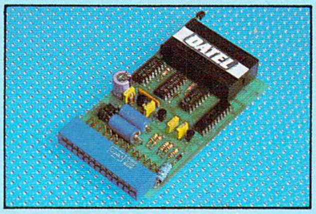
ONLY $24.99



256K SUPEROM EXPANDER

- Now you can select from any of 8 32K EPROMs
- 8 sockets to accept upto a 32K EPROM in each.
- On board operating system – no programs to load.
- Program your own EPROMs using our EPROM programmer.
- No need to have loads of cartridges – just make a selection from the Superom menu.
- Directory of utilities on power up.
- Fully menu driven on power up.
- Select any slot under software controls.
- Unique EPROM generator feature will take your own programs – basic or m/c and turn them into autostart EPROMs. (EPROM burner required).
- Accepts 2764/27128/27256 EPROMs.

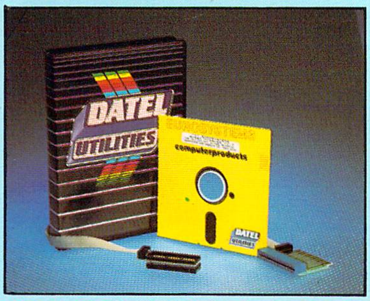
ONLY $49.99



EPROMMER 64™

- A top quality, easy to use EPROM programmer for the 64/128.
- Fully menu driven software/hardware package makes programming/reading/ verifying/copying EPROM's simplicity itself.
- Will program 27256 chips. 12.5, 21 or 25 volts.
- Fits into user port for maximum compatibility with cartridges/Superom Board etc.
- Full feature system – all functions covered including device check/verify.
- We believe Eprommer 64 is the most comprehensive, most friendly and best value for money programmer available for the 64/128.
- Ideal companion for Superom Board. Cartridge development System, our kernal expanders or indeed any ERPOM base project.
- Comes complete with instructions – plus the cartridge handbook.

ONLY $69.99 COMPLETE



DEEP SCAN BURST NIBBLER™

- The most powerful disk nibbler available anywhere, at any price!
- Burst Nibbler is actually a two part system – a software package **and** a parallel cable to connect the 1541/1570/1571 to 64/128 (state type).
- What gives Burst Nibbler its power? Conventional nibblers have to decode the data from the disk before it can transfer it using the serial bus – when non standard data is encountered they are beat. Burst Nibbler transfers data as raw GCR code via the parallel cable without the need to decode it so you get a perfect copy of the original.

- Will nibble upto 41 tracks.
- Copy a whole disk in under 2 minutes.
- Full instructions.
- Regular updates – we always ship the latest.
- Fitted in minutes – no soldering usually required.
- Full or $^1/_2$ tracks.
- No need to buy parallel cable if you have Professional DOS etc.
- Cable has throughbus extension for other add ons.

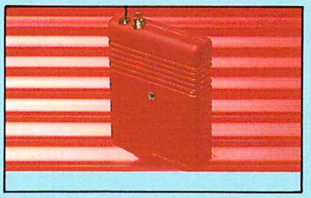
ONLY $39.99 COMPLETE

SOFTWARE ONLY $19.99 CABLE ONLY $19.99

BURST NIBBLER PARAMETER DISK

- Burst Nibbler is the most powerful Nibbler around – but even the best can be better
- This disk contains dozens of parameters specially written for the protection schemes impossible to Nibble. These are mainly 'V-max' and 'Rapidlock' and include titles such as 'Defender of the Crown', Gauntlet etc.
- Regularly updated to include any new titles.
- This disk is by no means a necessity – the standard Burst Nibbler will copy 98% of software as is – if you want to go to the limits, this is the product for you!

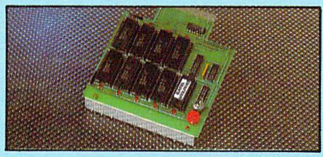
ONLY $14.99



SMART CART

- NOW YOU CAN HAVE A 32K CARTRIDGE THAT YOU PROGRAM LIKE RAM THEN ACTS LIKE ROM!
- 32K pseudo ROM – battery backed to last up to 2 years (lithium battery).
- Simply load the program you require – then flick the switch. The cartridge then acts like a ROM cartridge and can even be removed.
- Make your own cartridges – even auto start types, without the need for an Eprom burner.
- Can be switched in/out via software.
- I/02 slot open for special programming techniques.

ONLY $59.99



DUPLIKATOR

- Duplikator is a mass disk copier ideal for clubs, user groups etc.
- Comes with 256K on board RAM and it's own built-in operating system – no software to load.
- Makes entire backups in 8 seconds!!
- Multiple copies from one original – over 250 disks per hour. Full verify option.
- Copies upto 40 tracks and can cope with standard disk errors 21-29.
- The most price effective method of mass duplication available.

ONLY $189.99

EXTERNAL 3.5″ DISK DRIVE

- Slimline extra low profile unit – only 6″ long!
- Top quality NEC drive mechanism.
- Throughport allows daisychaining other drives.
- A superbly styled case finished in amiga colours.
- Fully compatible.
- 1 meg unformatted capacity.
- Good length cable for positioning on your desk etc.
- Complete – no more to buy.

ONLY $169.99 SINGLE
ONLY $299.99 DUAL

MIDIMASTER

- Full Midi Interface for A500/2000/1000 (Please state model).
- Compatible with most leading Midi packages (inc. D/Music).
- Midi In – Midi Out x 3 – Midi Thru.
- Fully Opto Isolated/Full MIDI.

ONLY $59.99

VISA® MasterCard

Circle 154 on Reader Service card.
Add $4.00 Shipping/Handling
CHECKS/MONEY ORDER/CODS ACCEPTED

DATEL COMPUTERS
3430 E. TROPICANA AVENUE, UNIT #67, LAS VEGAS, NV 89121

TOTAL BACK UP POWER TAKES A QUANTUM LEAP!

ACTION REPLAY IV THE ULTIMATE UTILITIES/BACK-UP CARTRIDGE FOR THE 64/128

• Action Replay allows you to Freeze the action of any memory Resident Program and make a complete back-up to disk or tape – but thats not all . . . . Just compare these features

- Simple Operation: Just press the button at any point and make a complete back-up of any* memory resident program to tape or disk.
- Turbo Reload: All back-ups reload completely independantly of the cartridge at Turbo speed.
- Sprite Monitor: View the Sprite set from the frozen program – save the Sprite – transfer Sprites from one game to another. Wipe out Sprites, view the animation on screen. Create custom programs.
- Sprite Killer: Effective on most programs – make yourself invincible. Disable Sprite/Sprite/Backround collisions.
- Unique Picture Save: Freeze and save any Hires Screen to tape or disk. Saved in Koala, Blazing Paddles Format.

ONLY $59.99

UNIQUE FEATURES:

WARP 25

Loads 50K in under 5 seconds!!

Yes it true action replay features 'Warp 25' the world's fastest disk serial Turbo. A typical backup will reload in under 5 seconds.

● No additional hardware required ● No special formats ● Super reliable ● Remember this is not a system where files have to be converted – with action replay you simply save directly into Warp 25 status – reload in seconds. ● Backup all your existing programs to load at unbelievable speed!! ● Because AR4 has on board ram it can also load commercial disks directly at 25 times speed.

This is unique to Action Replay!!

● Warp Save/Load Available straight from Basic.

Plus unique Code Cracker Monitor

● Freeze any program and enter a full machine code monitor ● Full Monitor Features – Dissasemble, Hex, Find, Jump, Compare, Replace, Two Way Scroll, Full Disk Load/Save, Printer Support etc. In fact all usual monitor commands plus a few others. ● Because of Action Replay's on Board Ram the Frozen program can be looked at in it's ENTIRETY: – In it's Frozen state. That means Video Ram, Zero Page etc. and remember you see the code in it's Frozen state not in a Reset state as with competitors products. ● Restart the program at any point. ● No corruption ● An absolute must for the program Hacker – make changes and see your result's instantly!

POWERFUL DESIGN WITH ON BOARD LSI LOGIC PROCESSOR CHIP PLUS 40K OF ON BOARD ROM/RAM!

- Compactor: Efficient compacting techniques – 3 programs per disk side – 6 programs if you use both sides.
- Single File: All programs saved as a single file for maximum compatibility.
- Utility Commands: A host of additional commands: – Autonum, Append, Old, Delete, Linesave, Printor lister
- Screen Dump: Print out any Screen to Host or Printers. 16 Gray Scales, Double size print options.
- Unstoppable Reset: Reset button to Rectrieve System and Reset even so called Unstoppable Programs
- Fully Compatible: Works with 1541/C, 1581, 1571 and Datacassette with C64,128, 128D (in 64 Mode)
- Compatible: With Fast DOS and Turbo Rom Systems
- Disk Utilities: Fast Format, Directory, List, Run and many other key commands are operated by Function Keys
- Unique Restart: Remember all of these utilities are available at one time from an integrated operating system. A running programme can be Frozen to enter any Utility and the program is restarted at the touch of a key – without corruption.

LS1 Logic Processor: This is where Action Replay IV gets it's power. A special custom LS1 chip designed to process the logic necessary for this unmatched Freeze/Restart power. No other cartridge has this power!

Action Replay Graphics Support Disk

To help take advantage of action replay's unique power to freeze any screen and save it into your favourite graphics package we have prepared a suite of graphic support facilities.

● **Screen Viewer** View screens in a 'slide show' sequence – Joystick control simple to use. ● **Message Maker** Add scrolling messages to your saved screens with music, very easy to use. ● **Prosprite** A full sprite editor, modify/save/load feature, full edit facilities. ● **Zoon Lens** Explode sections of any saved screen to full size including Border – Superb fun and very useful.

ONLY $19.99

CENTRONICS PRINTER LEAD

- Take full advantage of Action Replay or Final Cartridge screen dump features with this centronics lead.
- Connects user port to any full size Epson compatible printer.

ONLY $18.99

SUPERCRUNCHER

- A utility to turn your Action Replay into a super powerful program compactor. Reduce programs by up to 50%! Save disk space and loading times. Can even further compact programs already crunched by Action Replay's own compactor.

ONLY $9.99

NOW WITH RAMLOADER™

- Not only can Action Replay load your backups at 25 times normal speed . . . Now with the new RAMLOADER feature you can also load most of your commercial originals also at 25 times faster!!
- Making use of our special LSI logic processor and 8K of onboard Ram Action Replay now offers the world's fastest Disk Serial Turbo.

ORDERS ONLY 1-800-782-9110

NOTE: – Technical or any other type of Enquiry cannot be answered by the staff on this number.

TECHNICAL SUPPORT CUSTOMER SERVICE (702) 454-7700

MONDAY THRU SATURDAY 9am to 5pm PACIFIC TIME
EXTRA SHIPPING REQD. OUTSIDE CONTINENTAL U.S.

Volume 9, Issue 3

Publisher
Antony Jacobson

Vice-President Operations
Jeannie Lawrence

Vice-President Advertising
Maximillian Jacobson

Assistant Advertising Manager
Julie Cale

Editors
Malcolm O'Brien
Nick Sullivan
Chris Zamara

Contributing Writers
Marte Brengle
Paul Bosacki
Bill Brier
Anthony Bryant
Joseph Buckley
Jim Butterfield
William Coleman
Richard Curcio
Miklos Garamszeghy
Larry Gaynier
Todd Heimarck
Adam Herst
Robert Huehn
George Hug
Zoltan Hunt
Garry Kiziak
Francis Kostella
Mike Mohilo
D.J. Morriss
Noel Nyman
Adrian Pepper
Steve Punter
Joel Rubin
David Sanner
Stephen Shervais
Audrys Vilkas
W. Mat Waites

Cover Artist
Wayne Schmidt

Transactor

The Magazine for Commodore Programmers

Departments and Columns

Reviews

About the cover: *Dreamtime*, by Wayne Schmidt

Wayne Schmidt is our regular cover artist, creating the artwork on the C64 with a variety of software. Wayne explains the work as follows:

"Inspired by the wonderful totemic imagery of the Aboriginal Australians, the 'Dreamtime' refers to a core mythic state in which there is a communion with the Eternal Spirit and an experience of visions that transcend all boundaries of normal experience and human limitation."

Thanks to David Foster at ReadySoft, who supplied the RGB colour values to match the 64's colour set.

Transactors's
phone number is:
(416) 764-5273
Line open Mondays, Wednesdays
and Fridays ONLY

FAX: (416) 764-9262

TOLL-FREE ORDER LINE
1-800-248-2719 Extension 911
(for orders only; have your VISA or Mastercard number ready; available in the U.S. only)

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. 7690, 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, 85-10 West Wilmot Street, 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

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 in the printed listing 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
```

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 :
```

Start Address

Not Fair!

Boy, am I steamed! Recently I was in a computer store in downtown Toronto, waiting in line at the cash register. The customer in front of me was purchasing *GeoPublish* for the Apple. I was amazed to discover that the software is supplied on both 5.25" and 3.5" disks. What's more, a sticker on the front of the package makes the proud boast that the software is - hold your breath - *not copy-protected!*

Yes, you read that right. Apple GeoPublish is not copy-protected. I guess Apple users will never know the terrors of having only one boot disk. I wonder if they have serial numbers...

Why the special treatment for Apple users? Surely the vast majority of Berkeley Softworks' customers are Commodore users. Don't they deserve the consideration that is being shown to Apple users? Isn't it partly as a result of the resounding success of Commodore GEOS that BSW produced GEOS for the Apple? Why is Commodore GEOS copy-protected? Is it because GEOS is an "official" operating system? Was the copy-protection Commodore's idea?

I expect that every regular GEOS user has been inconvenienced, irritated or infuriated by the obstacles put in the way of the legitimate user. I know I have. A friend of mine (who is not a GEOS user) once told me that a copy-protected operating system was his definition of a useless thing. I can see his point and I'm sure that a lot of CP/M, MS-DOS and AmigaDOS users would concur. It just doesn't make sense.

If you want people to use your operating system and if you want programmers to develop applications to run in that environment, make it easy to use. And I don't mean 'point and click' easy.

I don't mean to malign BSW or Commodore. I just had to say something. Maybe one day copy-protection will disappear. Sigh....

Malcolm D. O'Brien

b i t s

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.

Bits puzzle solved

In Volume 8 Issue 5, we posed what we thought was a difficult challenge - the following simple program was presented:

```
1 print "*";: poke NUM,0
```

...and the challenge was to find what value of NUM would cause the program to fill the entire screen with asterisks. We didn't offer any prize for the solution, but half in jest, we offered a free bits book to anyone who could come up with a *second* solution.

Well, it wasn't too long before we received the first solution from Randy Thompson of Greensboro, North Carolina. Randy's answer was the one we expected: "Simply POKE a zero into the low byte of BASIC's TXTPTR ($7A-$7B) to reset CHRGET." We knew of no other solution, but still promised a free bits book to anyone who could come up with one.

Well, surprise! Jim Bond of Spokane, Washington recently submitted an article, and along with it the following solution if we would allow it:

```
1 print "*";: poke 2069,138:
```

Using '138' (and the extra colon) instead of 0 is a slight cheat, but the solution is ingenious enough, and considering we didn't think there even *was* one, we're giving Jim recognition (and the bits book) for the second solution. It works by adding a RUN token to the end of the program, modifying itself to keep running over and over again, printing an asterisk each time. Just goes to show that where there's a will, there's a poke!

Dynaborder
Jean-Yves Lemieux, Rimouski, Quebec

"Dynaborder" stands for "dynamic border". It is an interrupt-driven program that uses the raster line registers to enhance the screen border with a dynamic rainbow of colours. It can offer a bit of animation to your BASIC or machine language program, especially during an INPUT. Its shortness (215 bytes) lies in the fact that it contains self-modifying routines. To make it compatible with both the 64 and the 128 (40 column screen), this version is loaded at $3000 (12288). Enable with 'sys 12288' and disable with 'sys 12493'. The source code for Dynaborder follows:

```
JL  1000 sys700
HD  1010 ; * dynaborder             *
HA  1020 ; * pal source code        *
EO  1030 ; * by jean-yves lemieux *
LL  1040 ; * rimouski (qc) dec 88 *
AO  1050 ; ************************
KJ  1060 ;
GJ  1070 .opt oo
OK  1080 ;
GF  1090 tem     =$254    ;temporary storage
GJ  1100 irqold  =$257
CB  1110 irqvec  =$314    ;irq vector
ED  1120 rashi   =$d011   ;raster line
NG  1130 raslo   =$d012   ;registers
AL  1140 irr     =$d019   ;int. request reg
OK  1150 imr     =$d01a   ;int. mask reg
EH  1160 bcol    =$d020   ;border color
NG  1170 icr     =$dc0d   ;int. cntrl reg
CB  1180 ;
HI  1190 *=$3000
GC  1200 ;
DF  1210          sei
CL  1220          lda irqvec      ;prepare new
DF  1230          ldy irqvec+1    ;interrupt
EC  1240          sta irqold      ;procedure
KJ  1250          sty irqold+1
LN  1260          lda #<newirq
BE  1270          ldy #>newirq
AI  1280 di       sta irqvec
FN  1290          sty irqvec+1
CK  1300          cli
CG  1310          lda #1          ;enable raster
NN  1320          sta imr         ;line interrupt
EL  1330          sta irr         ;reset irr
EF  1340          lda #$1b        ;clear raster
MJ  1350          sta rashi       ;compare bit (8)
```

```
CP  1360          lda #$7f       ;clear irq
OD  1370          sta icr        ;flag bit
HI  1380          lda #$00
MH  1390          sta tem        ;prepare
DO  1400          sta tem+2      ;self-modifying
NL  1410          lda #$05       ;routine
AN  1420          sta tem+1
BB  1430          lda #$d7
CM  1440          sta n6+1
GJ  1450          rts
KC  1460 ;
ON  1470 newirq =*
OD  1480 ;
HK  1490          lda #$32       ;first interrupt
IM  1500          sta raslo      ;at line 50
CD  1510          ldx #1         ;reset
IK  1520          stx irr        ;register
FE  1530          ldy #0
HM  1540 n1       adc #2         ;if a raster line
AE  1550 n2       cmp raslo      ;has been reached
MF  1560          bne n2         ;we display
IK  1570          stx bcol       ;a color stripe
DI  1580 n3       inx
II  1590          adc tem+1      ;separated by
DE  1600 n4       cmp raslo
CI  1610          bne n4
OH  1620          sty bcol       ;a black line
ID  1630 n5       cpx #$05
KJ  1640          bne n1
GO  1650          bit tem
DH  1660          bvs rest
MP  1670 ;
HC  1680          ldx #0         ;modify prior
KB  1690          stx n5+1       ;routine
NJ  1700          ldx #$ca       ;'dex' opcode
EG  1710          stx n3
NL  1720 n6       lda #$00       ;display bottom
CG  1730          sta raslo      ;rainbow
FB  1740          ldx #1
LL  1750          stx irr
IA  1760          dec tem
JD  1770          ldx #4
GC  1780          bne n1
EH  1790 ;
DL  1800 rest =*                 ;restore newirq
PA  1810 ;                        routine
OM  1820          lda #$e8       ;'inx' opcode
AI  1830          sta n3
FC  1840          lda #5
KF  1850          sta n5+1
BJ  1860          inc tem
FF  1870          lda tem+2
DK  1880          beq r1
OP  1890          sec            ;modify raster
IB  1900          lda n6+1       ;line value
JH  1910          sbc #4
CK  1920          sta n6+1
AJ  1930          ldx tem+1      ;and stripe width
AH  1940          inx
KE  1950          cpx #$16
JD  1960          bne cirq
JN  1970          inc tem+2
CG  1980          beq cirq
FG  1990 r1       ldx tem+1
GE  2000          clc
OL  2010          lda n6+1
JA  2020          adc #$04
AB  2030          sta n6+1
PK  2040          dex
DJ  2050          bne cirq
OA  2060          dec tem+2
MI  2070 ;
JO  2080 cirq =*                 ;continue irq
AK  2090 ;
EN  2100          stx tem+1
BJ  2110          lda #$30       ;next raster line
MG  2120          sta raslo      ;interrupt
PD  2130          lda #1
FO  2140          sta irr
MG  2150          jmp (irqold)
GO  2160 disable =*
DB  2170          sei
MN  2180          lda irqold
IA  2190          ldy irqold+1
JC  2200          jmp di
```

The following program is a generator for 'dynaborder.obj'. Once this file is created by the program below, load it and execute it like this:

```
load"dynaborder.obj",8,1
sys 12288
```

```
KG  100 rem  generator for "DynaBorder.obj"
EL  110 n$="DynaBorder.obj": rem  name of program
AD  120 nd=215: sa=12288: ch=22654
```

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

```
AO  1000 data 120, 173,  20,   3, 172,  21,   3, 141
AJ  1010 data  87,   2, 140,  88,   2, 169,  61, 160
KD  1020 data  48, 141,  20,   3, 140,  21,   3,  88
GP  1030 data 169,   1, 141,  26, 208, 141,  25, 208
NE  1040 data 169,  27, 141,  17, 208, 169, 127, 141
FE  1050 data  13, 220, 169,   0, 141,  84,   2, 141
JI  1060 data  86,   2, 169,   5, 141,  85,   2, 169
CM  1070 data 215, 141, 115,  48,  96, 169,  50, 141
BH  1080 data  18, 208, 162,   1, 142,  25, 208, 160
LD  1090 data   0, 105,   2, 205,  18, 208, 208, 251
DJ  1100 data 142,  32, 208, 232, 109,  85,   2, 205
AK  1110 data  18, 208, 208, 251, 140,  32, 208, 224
CM  1120 data   5, 208, 230,  44,  84,   2, 112,  27
HH  1130 data 162,   0, 142,  96,  48, 162, 202, 142
BL  1140 data  83,  48, 169,   0, 141,  18, 208, 162
PH  1150 data   1, 142,  25, 208, 206,  84,   2, 162
LJ  1160 data   4, 208, 198, 169, 232, 141,  83,  48
```

```
CE  1170 data 169,   5, 141,  96,  48, 238,  84,   2
NM  1180 data 173,  86,   2, 240,  22,  56, 173, 115
NM  1190 data  48, 233,   4, 141, 115,  48, 174,  85
HI  1200 data   2, 232, 224,  22, 208,  23, 238,  86
OB  1210 data   2, 240,  18, 174,  85,   2,  24, 173
NN  1220 data 115,  48, 105,   4, 141, 115,  48, 202
GO  1230 data 208,   3, 206,  86,   2, 142,  85,   2
JC  1240 data 169,  48, 141,  18, 208, 169,   1, 141
MC  1250 data  25, 208, 108,  87,   2, 120, 173,  87
IC  1260 data   2, 172,  88,   2,  76,  17,  48
```

Data Mouth
Andrew Millen, Asbestos, Quebec

I recently discovered an amazingly useful method for checking data statements (especially long ones). Remember S.A.M.? For anyone with the Software Automatic Mouth, data-checking becomes a breeze! Simply load up S.A.M., then load in the data you want to check, and add these lines:

```
1 poke 53265,peek(53265) and 239
2 restore
3 read x: x$=str$(x): say x$
4 get a$: if a$="" then 3
5 get b$: if b$="" then 5
6 goto 3
```

S.A.M. will recite your numerical data (including decimals) so that you can easily follow along with your printed listing and compare. To pause (that S.A.M. is relentless!), press any key, and press any key to start up again. Note that line 1 turns off the screen to eliminate the irritating visual flash. When done, hit RUN/STOP-RESTORE to return to normal. I'm sure this trick is easily modified for other software mouths.

Video Reset
Jim Bond, Spokane Washington

Ever have your BASIC program bomb out while in hi-res mode? Can't see the error message showing line number that caused the error, can you? With this program, you just have to tap the RESTORE key by itself and voila! - the screen is restored to text mode without being cleared. Sound and sprites are turned off, too. It doesn't stop a running program, but don't use it during disk operations.

The program works by intercepting the NMI vector - an NMI is generated when the RESTORE key is pressed. It also redirects the 'error message link' vector at $0300 to re-install itself in case a RUN/STOP-RESTORE or another operation restores the NMI vector back to normal. To disable it, use the two ROM routines 'sys 58451: sys 64789'.

```
KC  100 rem video reset - relocatable
DB  110 ml=50000: rem start address
MO  120 :
OB  130 x=ml: x1=x+21: x2=x+95
OE  140 h1=int(x1/256): l1=x1-256*h1
IG  150 h2=int(x2/256): l2=x2-256*h2
BL  160 h3=int(x/256) : l3=x-256*h3
GF  170 gosub 230
OD  180 print"tap restore to reset video"
EP  190 poke ml+1,l1 : poke ml+3,h1
IO  200 poke ml+11,l2: poke ml+13,h2
IF  210 poke ml+99,l3: poke ml+100,h3
JG  220 sys ml: end
HJ  230 read a: if a=-1 then return
IM  240 poke x,a: x=x+1: goto 230
BK  250 data 169, 128, 162, 192, 141,  24,   3, 142
PL  260 data  25,   3, 169, 202, 162, 192, 141,   0
LP  270 data   3, 142,   1,   3,  96,  72, 152,  72
CB  280 data 169,   6, 141,  32, 208, 141,  33, 208
HF  290 data 169,  14, 141, 134,   2, 169,  23, 141
CI  300 data  24, 208, 169, 200, 141,  22, 208, 169
LG  310 data  27, 141,  17, 208, 169, 199, 141,   0
HD  320 data 221, 160,   0, 173, 134,   2, 153,   0
DE  330 data 216, 153,   0, 217, 153,   0, 218, 153
LH  340 data   0, 219, 200, 208, 241, 140,  21, 208
EK  350 data 152, 153,   0, 212, 200, 192,  25, 208
KP  360 data 248, 104, 168, 104,  76,  71, 254,  72
CJ  370 data 138,  72,  32,   0, 192, 104, 170, 104
FC  380 data  76, 139, 227,  -1
```

Alien Video
Brian Spencer, Barrie, Ontario

Alien Video is a machine language program that is installed through BASIC, and is totally relocatable. Just change the number in line 10 to whatever address you'd like the ML to reside at. After running the program, you'll be informed of the SYS to use to start Alien Video. When running, press any key to stop it. Besides a rather wild video display, the program produces some truly unusual sound effects.

```
MA  10 rem alien video
JI  20 sa=828
CM  30 for i=sa to sa+31
AM  40 read d: poke i,d: next i
FB  50 print "* sys";sa;"to start *"
BM  60 data 169,  11, 141,  17, 208, 169,  15, 141
EF  70 data  24, 212, 162,  23, 165, 162,  13,  18
DC  80 data 208, 157,   0, 212, 202, 208, 245, 141
DG  90 data  32, 208,  32, 228, 255, 240, 235,   0
```

How was the video made? At full speed, the machine language reads memory location $a2 (162), performs a logical OR with memory location $d012 (53266), and stores the final result in the SID (sound) chip registers. It is reading from two constantly changing memory locations: $a2 is the least significant byte in the 64's jiffy clock, and $d012 is the lower eight bits of the current screen line of the raster beam. The effect is a strange, alien-like sound. The visual part of the video is created by storing the same resulting byte to the screen border location ($d020, 53280); since the main display was turned off by a write to $d011, this affects the whole screen. That's all there is to it! Simple? Absolutely.

The Edge Connection

CP/M C, more assemblers, CPU bugs and drive tips

by Joel Rubin

CP/M programmer Leor Zolman put a classified ad in the November '88 *FOGHORN* offering his **BDS** C compiler package for $90 (US) for the first copy and $50 for each additional copy. Presumably, the idea is to order through a users' group. You get the source code for a full-screen editor, debugger, xmodem-compatible telecommunications program (will it work on the C128?) and standard I/O library. A few years ago I did some programming on a multi-Z80 MP/M system and had access to both BDS C and **Aztec C** from Manx. I found Aztec to be closer to the Unix/K&R standard (especially when it came to using files); but, once I got used to BDS, I found it easier to work with. BD Software is at P.O. Box 2368, Cambridge, MA 02238, (617) 576-3828. (I think the zip code should be 02138 not 02238.) Mr. Zolman takes check, VISA or Master Card. Be sure to specify disk format or you might get the old CP/M default format - 8" single density!

Speaking of the *FOGHORN*, FOG, the one-time First Osborne Group, which supports CP/M, MS-DOS, and (soon) the Mac, is raising its dues on New Year's Day. You can order up to five years of membership for $25 (or $44 if you want both CP/M and MS-DOS publications) through 1988. (These are going up to $30 and $52.50, respectively.) There is a surcharge of $12 per year per publication if you live in Canada or Mexico or if you live in the U.S. and want first class delivery. FOG is at 210 Lakeshire, P.O. Box 3474, Daly City, CA 94015-0474, or, if you want to join by VISA or Master Card, you can phone (415) 755-2000, Monday through Friday, 1000 to 1730 Pacific Time. They also have a starter disk for $4, modem disk (specify set-up) for $4, and a three-disk catalog set for $10. The catalog set includes CP/M and MS-DOS programs and data files and is only available in Osborne DD or 360K MS-DOS formats. If paying by card you will get charged $1 shipping per $25 merchandise.

One more CP/M note: There is an error on page 684 of the *Commodore 128 Programmer's Reference Guide*. TYPE should be XDPH-1 and UNIT XDPH-2; not reversed as they are. This is correct in the DRI *Systems Guide* but it's somewhat confusing - UNIT and TYPE are shown as the low byte and high byte, respectively, of a word at XDPH-2, and, except in Motorola-land, the low byte of a word is at the lower address.

Since I wrote a comparison of **Merlin128**, **Buddy** and **LADS** in *Transactor* 9:2, I have seen two more 6502 assembler packages - Commodore's own **DevPak** for the C128 and **Geoprogrammer**.

Some Commodore developers prefer to do their development on other machines and then download. Berkeley Softworks credits its use of sophisticated cross development tools for much of its success. Others, such as Eric Rosenzweig, who wrote the **PTD-6510** debugger for Pterodactyl, say that programming on the object machine helps you to get used to the machine and program around its weak points. To quote Mr. Rosenzweig, writing in the September 1984 edition of the newsletter put out by the Programmer's Shop, (800-421-8006 - I don't know if they sell anything for 8-bit Commodore computers in 1988) "Programming on a big machine and downloading to a smaller or slower one results in a program being written for a big machine that runs slow and large when put on the target machine."

Now, we have some programmers using the cross-development method who are so enamoured of their mainframe-based programming tools that they have attempted to port their tools to the object machine. **DevPak 128** and **Geoprogrammer** each have many fine features, but they run slow and large when put on the target machine.

DevPak 128 ($50 U.S. from Commodore Business Machines, 1200 Wilson Drive, West Chester, PA 19380) is extremely disk intensive. First, you edit the source file, using either the *EDT* editor which comes with the package, or, if you don't want to learn new editing commands, any word processor which can save PETSCII text files to disk. Then, you load the assembler which creates files similar to (but not the same as) Intel Hex Files. Finally, you boot the loader, which reads the hex files into memory as binary code, and save the code to disk, using the C128's monitor. The loader can load the hex files into another part of memory if necessary - for example, if the binary image and the loader itself conflict. (Cinemaware's **Warp Speed** cartridge helps in this case as its monitor contains a "save using another load address" directive.)

The *EDT* editor, ported from a Digital Equipment mainframe, includes most of what you might want in a programmer's editor, except, perhaps, for split-screen two file editing, and editor macro commands. It can handle files in PETSCII or ASCII, with line lengths up to 255 characters, and can convert between the two. If you want to type in long lists of numbers with the numeric keypad, you will find, to your chagrin, that *EDT* uses the numeric keypad for commands.

The assembler is a full macro assembler. I think it is possible to write a macro package to allow this assembler to use 8080 or Z80 op-codes, in some form, similar to *x6502.lib* on the CP/M extras disk, in case you wish to write mixed 6502/Z80 programs for the C128, but no such macro package is included. One feature which I missed was an 'offset'-type pseudo-op. Let's say that you are going to write code at one address which will be moved to another address (or downloaded to disk RAM) before it is run. You would like to assemble so that your address references (e.g. in a JSR) refer to the running address rather than to the original loading address. Some assemblers allow you to do this, but **DevPak** won't - you either have to assemble the offset code separately, or add the offset to all the address references.

With **DevPak**, you also get the source code for file compressors, C64 fast loaders, and the DOS for the RAM expanders. There are also some utilities, such as a C64-mode sprite editor. The manual includes a discussion of some ROM differences in 8-bit Commodore equipment, including the SX-64. The discussion of the new 1571 ROM, and the 1541C and 1541-II ROMs sounds as if Commodore thinks they have finally exterminated the save-with-replace bug.

The manual is more a spiral-bound collection of unrelated papers than a manual. Some of the papers, such as the assembler instructions, are well-written and clearly printed. However, some of the program listings seem to have been printed on a 1525, or similar low-quality dot matrix printer. Since these program listings are on disk, you don't need their listing. In case you don't know which way the wind is blowing, the manual cover has the word 'Amiga' twice as large as the word 'Commodore'.

The main problem with **DevPak** is its disk intensiveness for even the most minor programming task. (All assemblers running on a C128 are going to become disk intensive if you try to write a 60K program.) If you use it with a single 1571 or 1541, you are going to find yourself quickly running into the limits on the number of open files caused by the limited disk RAM, and, indeed, the assembler will warn you of that fact. Thus, if you want to include a file of often-used macros and often-used equates, and get both a listing and object file, you may have to repeat the assembly twice.

Geoprogrammer, ported from Berkeley's Unix-based cross development system, is going to come out in Version 2.0 "real soon now". Version 1 only runs under the C64 version of GEOS; version 2 will run under either GEOS or GEOS128. Like RMAC, under CP/M, **Geoprogrammer** is an 'edit, assemble, link, debug' system. The editor, for better or worse, is any version of *geoWrite*. On the one hand, *geoWrite* is slow and clunky for entering text. On the other hand, *geoWrite* allows you to paste in pictures, and *geoAssembler* allows you to define icons or other bit patterns using this. Of course, there's always *Text Grabber*, which converts a file from another word processor to *geoWrite* format.

GeoAssembler is a macro assembler. I don't think its macro language has quite the power of **DevPak**'s, but, on the other hand, *geoAssembler* can compute very complex 16-bit arithmetical expressions using a C-like syntax. *GeoLinker* combines the *.rel* files and turns them into a regular Commodore program, a GEOS sequential program, or a GEOS VLIR program with a resident module and, possibly, overlay modules. One nice feature of *geoLinker* is that if files A and B create global labels with the same name, you will not get an error unless file C tries to access that name as a Random external label (or maybe it allows you to be sloppier than you should be). While *geoAssembler* and *geoLinker* do create, if necessary, error files, and *geoLinker* creates a symbol file, neither one creates listing files, which can be a pain.

The best feature of the **Geoprogrammer** package is *geoDebugger*, but to use the debugger in its full glory requires that you have a RAM Expansion Unit. You can single step, or single step at the top level and execute subroutines at full speed, set break points, and perform all the usual monitor functions. If you have an REU, you can define debugger macros or refer to locations in symbolic terms. The **Geoprogrammer** manual is a huge beast, and is somewhat disorganized, but contains very useful information on programming under GEOS. **Geoprogrammer**'s advantages far outweigh its disadvantages *if* you are writing programs that are to run under GEOS. However, while it *can* assemble non-GEOS programs, I think that other assemblers will do the job with far less hassle and probably more speed.

Geoprogrammer can be purchased directly from Berkeley Softworks (Great Western Building, 2150 Shattuck Ave., Penthouse, Berkeley, CA 94704) for $69.95 plus $4.50 shipping plus $4.90 sales tax in California, or through the usual retail outlets.

Recently, in looking over some machine language reference books, I noticed that several of them do not mention ye olde JMP-indirect bug - including the *C128 Programmer's Reference Guide*. (Even though the bug *does* exist on the 8502!) In case you're learning 6502 programming and haven't run into it, here's the problem:

Ordinarily, you expect JMP (vector) to load the PC with peek(vector) + 256 * peek(vector+1). *However*, if vector is on a page boundary, for example $18FF, you will get peek(vector) + 256 * peek(int(vector/256) * 256)). Thus, if $18FF contains $2D and $1800 contains $4F and $1900 contains $5C, then JMP ($18FF) jumps to $4F2D, not $5C2D - the microprocessor looks up the high byte at $1800, not $1900.

Just in case you're tempted to use this to confuse some pirate, you should know that the bug *has* been fixed on the 65C02 and 65816 - so the resultant code won't work with speed-up boards. Also, by a clear corollary to Murphy's Law, if you try playing with this, you will probably add or delete something and forget to make sure that the vector is or is not on a page boundary - leading to a next-to-impossible debugging job. If you must use JMP indirect, you should use assembler pseudo-ops to add filler bytes if necessary. I think that self-modifying code may be safer in some cases.

Another potential problem on the 6502 involves the TXS op-code. Whenever you decrease the stack pointer (extend the stack) using TXS you should make sure that no interrupt, be it maskable or non-maskable, can possibly take place. For example, consider the following code, intended to let a routine find out where it is in memory:

```
     lda #$60 ; rts
     sta $100 ; you almost never use this part of the stack
adr1 jsr $100
     tsx
adr2 dex
     dex
     txs
     pla
     sta $fc
     pla
     sta $fd
```

You now expect ($FC) to contain *adr1+2* because of the way JSR uses the stack. However, suppose an interrupt strikes on the first DEX. The interrupt overwrites the positions on the stack you are trying to read, and ($FC) now contains *adr2*. It won't happen very frequently, but, again by a Murphy's Law corollary, it will happen at the worst possible time.

If you have a C128 (or, I believe, C16/Plus 4) you have an alternative:

```
jsr primm
.byte 0
```

will leave the address of the null byte in ($CE). As long as you are not actually printing anything, you don't have to be in Bank 15 - any memory configuration in which the high ROMs are visible will work.

(Speaking of Bank 15, if you have a C128, you should always make sure that the I/O chips are visible before you try to do any input or output. This goes double if you try to interface with BASIC, as BASIC tends to leave you in configurations like Bank 14, or Bank 14 with RAM 1. This is why the version of the C128 Verifizer that appeared in *Transactor*s before 9:1 wouldn't work in 80 columns. If your machine language uses C128 BASIC routines that deal with variables or strings, you may find you have to use JSRFAR even though you are going from Bank 15 to Bank 15, because the BASIC routines end up in Bank 14 with RAM 1 and your program will try to return to the right address in the wrong bank - instant crash!)

Finally, a few notes about 1541 and 1571 disk drives:

a) Do you want to distinguish between the two? Try this:

```
open 1,8,15, "m-r" + chr$(dec("67")) +
   chr$(dec("fe")): get#1,a$
```

This will read the first byte of the IRQ routine on either drive. On the 1541, the IRQ routine begins with PHA ($48). On the other hand, since the 1571 has two modes, the IRQ routine begins with a jump indirect instruction ($6C). The vector is at $2A9 and points to $9D88 in 1541 mode and to $9DDE in 1571 mode. Some commercial programs (*Copy II* and *Fast Hack'Em*) got into trouble trying to read the signature byte at $C000 which changed when the 1571 ROM changed.

b) Do you want to change a single-sided 1541 disk into a 1571 disk without losing data on the 1541 disk? (*Follow at your own risk!!! Destroys flippies!!!*)

```
open 1,8,15, "i0": print#1, "m-e"
   chr$(69) chr$(164)
```

will format the second side of the disk. (You use "i0" to set up the disk ID value correctly.)

Of course, this still doesn't finish the job - the double-side flag on track 18, sector 0 is still single, and the BAM for the second side isn't written. So, you will have to change byte 3 (counting from 0), the double-side flag of 18/0, to $80, and copy the bytes 221 to 255 of 18/0 (giving the summary of the side two BAM) from a freshly formatted double-side disk. Then, dclear, which will tell the 1571 that you have a double-sided disk. Finally, copy 53/0 from a freshly formatted double-sided disk.

c) Last, a faster way to dump a 1541 or 1571 ROM to disk - instead of reading each byte into the computer and then writing it back to the disk drive, you get the disk drive to write the bytes directly to disk. Of course, entering the program takes more time than you will ever save, but it's a neat hack. Maybe you can figure out some use for it.

First, **open 2,8,2,"#0"**. Note the '#0' - the '0' tells it you want the buffer at $0300. Now, send the machine code below to the buffer: (**open 1,8,15,"b-p:2,0"**, then print the bytes to file 2)

(*fstad* is the first address of the ROM which you want to dump - usually $8000 for the 1571, $C000 or $C100 for the 1541.)

Now, **open 3,8,1,":dosfile"**. This opens a program file called *dosfile* to write. Now, type:

```
print#1, "m-w" chr$(0) chr$(0)
   chr$(1) chr$(224)
```

This will tell the disk drive to execute the machine code in buffer 0. The dumping of the disk drive ROM will take place, and all files will be closed, independently of the computer.

"romdump" - *Follow directions given in article*

```
     org $300

     lda #1
     sta 0     ; tell the system you finished running this code OK so the
               ; disk drive can do other work, like writing bytes
     cli       ; all sectors are written in the interrupt cycle
     sta $83   ; current secondary address
     lda $022c ; disk drive internal channel for secondary address 1
     and #7
     sta $82   ; current disk drive internal channel
     lda lup+1 ; put the load bytes to file 1
     jsr put
     lda lup+2
     jsr put
lup  lda fstad ; yes, this is self-modifying code
     jsr put   ; to change the address from fstad to the current address
     inc lup+1 ; unfortunately, I couldn't find a zero page address which
     bne ninc  ; didn't get corrupted, so I had to do it this way
     inc lup+2
ninc lda lup+2
     cmp top+1
     bne lup
     lda lup+1
     cmp top
     bne lup
     jsr close
     lda #2
     sta $83
     jmp close
top .word finaladdress+1 mod 65536
put   =  $d19d ; put a byte in the current disk file
close =  $dac0
```

TransBlooperz

Oops! A bug in our program that creates BASIC "generator" programs managed to sneak two bad generators by us before we caught it. The result is that the programs will not work as listed, but fortunately the problem is very easily solved. These are the affected programs:

Volume 9 Issue 2, "Cycle Counting", page 31:
Don't panic - all the DATA statements are correct! Just change line 110 as follows:

```
110 n$="cc.c000"
```

...and replace lines 130-250 with lines 130-260 from the "standard generator" program on page 5.

Volume 9 Issue 1, "Multitasking on the C128", page 21:
This one was even more messed up - after line 1150, the line numbers start again at 1000! Ignore the first set of lines 1000-1150, and use the standard generator on page 5 in their place. Then replace lines 110 and 120 as follows:

```
110 n$="multi.ml"
120 nd=529: sa=4864: ch=57790
```

The ML Column

Creating order from chaos

by Todd Heimarck

The idea for this month's column comes from someone else. Last year, *Byte* magazine reviewed a book that was a compilation of columns from *Scientific American* magazine. One of the programs described in the review sounded interesting.

The scenario is simple enough: You start with a pool of voters who have been assigned random political leanings, pick one of the voters at random, pick a neighbor at random, and change the neighbor's political preference to match the original voter. Then you repeat the process in an endless loop (or until somebody stops the program by pressing a key).

The Commodore 64's hi-res screen has 320 x 200 pixels. That's 64,000 voters. You assign one of two colours to each pixel (I picked blue and white because they contrast with each other). Counting diagonals, each pixel has eight neighbors. In the main loop, you pick one of the voters, one of its eight neighbors, and change the neighbor's colour to match the voter.

You create a tiny universe where everything happens randomly. The voters are given colours at random. A voter is picked at random. A neighbor to be converted is picked at random.

It sounds absurd, but in this chaotic and utterly random universe, patterns of order arise. The first screen looks like television static. After several hundreds of thousands of arguments between neighbors, you see definite blobs growing on the screen.

Although every rule relies on randomness, the voters gather together into blocs of solidarity. Here's a blotch of blue; there's a blotch of white. If you let the program run for a long enough time, you would probably see either blue or white take over the whole screen.

The boon and the bane of assembly language

I wrote the original VOTERS program in the C language on a PC compatible. Then I translated it to run under BASIC 7.0 for the 128.

Both programs were relatively slow. It made sense to switch to assembly language, to squeeze the last drop of speed out of the computer. That's the main reason for programming in assembly language: It's the fastest game in town.

I ran into a problem, however, which became the second topic for this column. Machine language is fast, but it's not always very good at handling randomness.

Modular programming

Let's jump into the program. It starts like this:

```
c000 20 0d c0    jsr gmode      ; turn on graphics mode
c003 20 1e c0    jsr initvote   ; initialize voters
c006 20 68 c0    jsr campaign   ; randomly change votes
c009 20 0d c0    jsr gmode      ; back to text mode
c00c 60          rts
```

Although the program depends on chaos, that doesn't mean we have to be chaotic about writing it. If you believe in modular programming (also called "top-down programming"), you break down the task into modules. The C, BASIC, and assembly programs all looked pretty much the same because they had the same structure.

The GMODE subroutine toggles the 64 between text mode and graphics mode:

```
c00d          gmode  =  *
c00d ad 11 d0    lda $d011          ; scroly
c010 49 20       eor #%00100000; flip bit 5
c012 8d 11 d0    sta $d011          ; toggle graphics mode
c015 ad 18 d0    lda $d018          ; vmcsb
c018 49 0c       eor #%00001100; toggle bits
c01a 8d 18 d0    sta $d018          ; toggle base addresses
c01d 60          rts
```

It's a short routine that makes the screen flip from text mode to graphics mode (and vice versa), but only if you start with a 64 that's set for the default values. The EOR (exclusive-or) command changes the appropriate bits in SCROLY and VMCSB (Commodore's names for locations $D011 and $D018).

The next routine is called INITVOTERS:

```
c01e          initvote =   *
c01e  20 28 c0    jsr  rndinit  ; crank up the noisy SID
                                ;voice
c021  20 36 c0    jsr  fill     ; fill the colour bytes
c024  20 4a c0    jsr  choose   ; the voters randomly choose
                                ;a colour
c027  60          rts
```

Most people think of the SID chip as a musician, but if you tell it to use a noise waveform, you can get random numbers from it. The RNDINIT routine makes the SID chip start acting randomly:

```
c028          rndinit  =   *
c028  a9 ff       lda  #$ff
c02a  8d 0f d4    sta  $d40f   ; max hi frequency
c02d  a9 80       lda  #$80
c02f  8d 12 d4    sta  $d412   ; noise waveform
c032  8d 18 d4    sta  $d418   ; volume off and no output
                               ;for voice 3
c035  60          rts
```

The registers at $D40E and $D40F control the frequency of voice three and $D412 controls the waveform (we're seeking noise). Storing an $80 into $D418 prevents the noise from being heard.

Two ways to fill memory

The hi-res screen will get its colour information from the text screen (although we could change that if we wanted to). The next routine fills locations 1024-2023 with the blue/white byte. The .Y register can only count to 255 and we need to fill 1000 bytes. One way to do it is to count up to 250 four times:

```
c036          fill    =   *
c036  a9 61       lda  #$61       ; foreground 6 (blue) and
                                  ; background 1 (white)
c038  a0 fa       ldy  #250
c03a          col0  =   1024
c03a          col1  =   col0 + 250
c03a          col2  =   col1 + 250
c03a          col3  =   col2 + 250
c03a  88      lpfill dey          ; note that this sets the
                                  ; zero flag
c03b  99 00 04    sta  col0,y
c03e  99 fa 04    sta  col1,y
c041  99 f4 05    sta  col2,y
c044  99 ee 06    sta  col3,y
c047  d0 f1       bne  lpfill
c049  60          rts
```

The four STAs don't affect the zero (equal-to-zero) flag. So when the program does a Branch if Not Equal (BNE) at $C047, it's working from the DEY instruction at $C03A. DEY affects the Z flag and STA doesn't.

Although this subroutine might look a little odd, the oddness is necessary. We want the .Y register to count backward from 249 to 0 (forward from 0 to 249 would be OK, too). I chose 249 to 0 because I could leave out the CPY instruction. The 6502 processor knows when it hits a zero (equal-to-zero) condition. It doesn't recognize 250 unless the program makes an explicit test for 250. You save a little time and a byte or two if you wait for a zero.

Also, we don't really want to loop 250 to 1, we want 249 to 0. But we want to STA when .Y contains a zero, so we DEY before the STAs.

Some people might put the location 1024 into a zero-page pointer and store indirectly with .Y. That would work, but it would probably take more bytes and more clock cycles (try it if you don't believe me).

The next routine fills 8192 bytes of bitmap memory with random numbers:

```
c04a          choose   =   *
c04a          bitmap   =   $2000
c04a  a2 20       ldx  #32          ; 32 pages of 256 bytes =
                                    ; 8192
c04c  a0 00       ldy  #0
c04e  a9 00       lda  #<bitmap
c050  8d 5c c0    sta  selfmod+1
c053  a9 20       lda  #>bitmap
c055  8d 5d c0    sta  selfmod+2 ; set up the address
c058  ad 1b d4  lpchoose lda  random
c05b  99 ff ff  selfmod  sta  $ffff,y   ; not the real address
c05e  c8          iny                   ; count forward
c05f  d0 f7       bne  lpchoose         ; until .y wraps
c061  ee 5d c0    inc  selfmod+2
c064  ca          dex
c065  d0 f1       bne  lpchoose         ; and repeat a total
                                        ; of 32 times
c067  60          rts                   ; and that's all
```

Look at $C05B STA $FFFF,Y. It looks like the value in .A is being stored at $FFFF indexed by .Y, but that's not really true. A few bytes back, SELFMOD+1 and SELFMOD+2 are changed. This is called "self-modifying code."

At $C061, the high byte (SELFMOD+2) increments. The program is programming itself by changing bytes within a loop. If you use this technique, remember four things:

1) You can't rely on values being stable when you enter the subroutine. You should initialize the memory value (see $C04E-$C057) at the beginning of the routine.

2) The 6502 puts the low byte before the high byte. The instruction STA takes a byte, so the low byte is *xxx+1* and the high byte is *xxx+2*.

3) If you know what you're doing, you can do amazing things with self-modifying code. If you don't, you'll get headaches when you try to debug your program.

4) Structured programmers will think you're crazy (or stupid) if you write self-modifying code. If you're majoring in computer science in college, you might be expelled for doing things like this.

The program ends with the final subroutine:

```
c068  20 e4 ff  campaign   jsr  getin
c06b  f0 fb                beq  campaign
c06d  60                   rts
```

This is just a placeholder. The meat of the program would go here. But there's a major problem that I can't solve.

Computers aren't very random

Assembly programs are so fast that the SID chip isn't random enough. It spits out noisy numbers, but they follow a pattern. Painting the screen with output from voice three produces very definite shapes and diagonal lines. Try POKEing various numbers into 54286 (or STAing into $D40E).

I wrote an entire CAMPAIGN routine, but it was flawed because the 64's SID chip couldn't produce random enough values. You can make a computer act chaotic up to a point, but then it insists on being orderly. If anybody has a solution, I'd like to hear about it.

Listing 1: *Source code in PAL format for the voters program*

```
LL  10 rem save"v.src",8
FO  20 sys 700
OF  30        *=49152
AJ  40 .opt oo
IB  50 getin  = $ffe4
CL  60 random = $d41b
ML  70 ;
PM  80 ; ----------
LF  90        jsr gmode        ; turn on graphics mode
GI  100       jsr initvote     ; initialize voters
MB  110       jsr campaign     ; randomly change votes
AJ  120       jsr gmode        ; back to text mode
OG  130       rts
CA  140 ;
FB  150 ; ----------
FO  160 gmode = *
GP  170       lda $d011        ; scroly
MM  180       eor #%00100000   ; flip bit 5
MN  190       sta $d011        ; toggle graphics mode on/off
DM  200       lda $d018        ; vmcsb
AA  210       eor #%00001100   ; toggle bits
CJ  220       sta $d018        ; toggle base addresses
CN  230       rts
GG  240 ;
JH  250 ; ----------
OI  260 initvote = *
MC  270       jsr rndinit      ; crank up the noisy sid voice
HG  280       jsr fill         ; fill the color bytes
HE  290       jsr choose       ; the voters randomly choose a color
IB  300       rts
MK  310 ;
AI  320 rndinit = *
HA  330       lda #$ff
DE  340       sta $d40f        ; max hi frequency
JJ  350       lda #$80
PA  360       sta $d412        ; noise waveform
OH  370       sta $d418        ; volume off and no output for voice 3
IG  380       rts
MP  390 ;
LN  400 fill  = *
EH  410       lda #$61         ; foreground 6 (blue) and background 1 (white)
CF  420       ldy #250
CN  430 col0  = 1024
MP  440 col1  = col0 + 250
JA  450 col2  = col1 + 250
GB  460 col3  = col2 + 250
GC  470 lpfill dey             ; note that this sets the zero flag
JO  480       sta col0,y
HP  490       sta col1,y
FA  500       sta col2,y
DB  510       sta col3,y
AF  520       bne lpfill
OP  530       rts
CJ  540 ;
IP  550 choose = *
OG  560 bitmap = $2000
PH  570       ldx #32          ; 32 pages of 256 bytes = 8192
PI  580       ldy #0
KB  590       lda #<bitmap
HG  600       sta selfmod+1
KC  610       lda #>bitmap
ID  620       sta selfmod+2    ; set up the address
HK  630 lpchoose lda random
PA  640       selfmod sta $ffff,y; this isn't the real address
CP  650       iny              ; count forward
EC  660       bne lpchoose     ; until .y wraps
AJ  670       inc selfmod+2
PF  680       dex
IH  690       bne lpchoose     ; and repeat a total of 32 times
EM  700       rts              ; and that's all
MD  710 ;
PE  720 ; ----------
AG  730 campaign jsr getin
HE  740       beq campaign
KN  750       rts
```

Listing 2: *BASIC generator for the voters program*

```
KM  100 rem  generator for "v.obj"
FP  110 n$="v.obj": rem  name of program
FA  120 nd=110: sa=49152: ch=14246

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

OM  1000 data  32,  13, 192,  32,  30, 192,  32, 104
JA  1010 data 192,  32,  13, 192,  96, 173,  17, 208
NP  1020 data  73,  32, 141,  17, 208, 173,  24, 208
OC  1030 data  73,  12, 141,  24, 208,  96,  32,  40
KE  1040 data 192,  32,  54, 192,  32,  74, 192,  96
HF  1050 data 169, 255, 141,  15, 212, 169, 128, 141
AG  1060 data  18, 212, 141,  24, 212,  96, 169,  97
AF  1070 data 160, 250, 136, 153,   0,   4, 153, 250
BF  1080 data   4, 153, 244,   5, 153, 238,   6, 208
LH  1090 data 241,  96, 162,  32, 160,   0, 169,   0
OP  1100 data 141,  92, 192, 169,  32, 141,  93, 192
CJ  1110 data 173,  27, 212, 153, 255, 255, 200, 208
NN  1120 data 247, 238,  93, 192, 202, 208, 241,  96
JP  1130 data  32, 228, 255, 240, 251,  96
```

Keep-80

Non-destructive windowing on the C128

by Richard Curcio

The C128's 80-column Video Display Controller has features that can enhance our 80-column text screens. Two of these features, 4K of unused RAM and a hardware 'block-copy', can be used to overcome a limitation of the WINDOW command: once a C128 window has been opened, whatever was under it is lost. By copying the 80-column screen to the unused area before opening a window, recalling the saved screen 'closes' the window and restores the text and attributes over-written by it. This can give our C128 programs the look of more advanced (and more expensive) computers.

The VDC and Keep-80

The Video Display Controller (VDC) has its own 16K of RAM. This RAM does not appear in the C128 memory map, and can only be accessed through the VDC. Since 80 columns by 25 rows require 2000 bytes, 2K bytes of VDC RAM are assigned to screen memory. A corresponding 2000 bytes are required by attribute memory, which is similar to the 40-column display's colour memory. The character definitions are also stored in VDC memory. Though only eight bytes are needed per character, each is padded out to 16 bytes for a total of 8K for both upper-case/graphics and lower/upper-case characters. This accounts for 12K of the 16K of VDC RAM, leaving 4K unused in normal circumstances. It is this unused memory that Keep-80 uses to hold a copy of the text and attributes. However, there's more to a video display than the characters and colours. A number of locations in zero page and page three keep track of the screen or window dimensions, cursor position, the current colour, tab positions, 'linked' lines, etc. This information can collectively be referred to as the Editor Values; Keep-80 stores these in the unused area as well.

How Keep-80 works

When the 'store screen' command is issued, after testing for 80-column mode, the routine moves the 40 bytes of editor values from RAM 0 to the unused 48 bytes at the top of 80-column screen memory. The VDC's copy feature is then used to move everything from $0000-$0FCF (beginning of screen to end of attributes), to the unused area, $1000-$1FCF. Instead of calculating the number of pages and bytes to move, and invoking the copy mode the necessary number of times, Keep-80 uses a ROM routine that takes care of everything. The routine at $C53C in Screen Editor ROM is used for 80-column scrolling and line clearing. To use it, the VDC memory destination end address (plus one) is stored in RAM 0 locations $0A3C and $0A3D in low-high format. The destination start is stored in VDC registers $12-$13 in high-low format. The source start address is stored in VDC registers $20-$21, again in high-low format. Setting bit 7 of register $18 tells the VDC that the next block operation will be a copy. **JSR $C53C** does the rest. It is a misnomer to call this operation a block-copy, however, because the ROM routine invokes the VDC copy mode one byte at a time! Still, using this routine simplifies the programming somewhat, and any loss of speed is negligible, especially in FAST mode, which should always be used in 80 columns anyway.

There are two 48-byte areas still available, one at the top of the attributes area and one at the top of the unused area. Since Keep-80 already has code to move editor values to and from VDC RAM, I have given it the ability to preserve two additional sets of editor values. In this way, your program can jump from window to window, perhaps using one to receive input and the other to display results. This feature can be made to function in 40 columns. Obviously, considerable confusion will result if 80-column editor values are recalled to a 40-column screen.

Usage

Keep-80 can be called from BASIC or machine language. The C128 must be in the BANK 15 configuration. The accumulator holds the type of operation and X holds the direction, which is zero to save and non-zero to recall. If **KEEP** is the location of the routine,

```
SYS KEEP, 0, 0
```

saves the current 80-column screen. This should be done *before* opening a window. If the current text mode is not 80 columns, the processor carry bit is set and the routine returns. From BASIC, **RREG,,,SR** will read the status register into the variable SR. **IF SR AND 1=1 THEN** you know you made a mistake. In assembler:

```
lda operation
ldx direction
jsr keep
```

You can then branch on the carry flag appropriately. To recall the saved screen:

```
SYS KEEP, 0, 1
```

To save and recall only the editor values, the accumulator should hold a 1 for the first set or 2 for the second set. Values greater than 2 will also set the carry to signal an error. Direction is as described above.

The BASIC loader (Listing 2 at the end of this article) pokes the Keep-80 machine language into the applications area at location 4864. Keep-80 can be located elsewhere by changing the variable **KE** in line 110. Other possible locations include the RS-232 buffers at 3072-3583, and the sprite definition area at 3584-4096. After running the loader, it will print the range of memory the ML occupies.

Modifications and demo

If one of the 80-column character sets is unused, Keep-80 can be made to store another screen at that location. KEEP+6 and KEEP+7 hold the starting and ending pages of the storage area. The normal contents of these locations are $10 and $1F. The values $20 and $2F in these locations will move storage to the upper-case/graphics character set. To use the lower/upper-case character set area for storage, poke KEEP+6 and KEEP+7 with 48 and 63 ($30/$3F). These pokes should be performed only when one of the character sets is not used. (To regain the complete 80-column character set, use **BANK 15: SYS 49191.**) This modification makes Keep-80 compatible with D.J. Morriss' Twin-80 program (*Transactor*, Volume 8, Issue 3), since that program uses the normally-unused area for a second screen. Keep-80 only copies the default text and attributes locations ($0000-$0FCF), and these pokes do not affect editor-only storage/retrieval.

To save and recall 40-column editor values, Keep-80 can be entered beyond the test for 80 columns with **SYS KEEP+8, A, X.** Be certain that **A** is not zero in that case. With a little more work, the routine can store many more sets of editor values, but only if a complete screen will not be saved, or one of the character sets is unused. First, store the direction value in location 195 ($C3), then **SYS** or **JSR KEEP+103** ('editsr' in the source listing) with **A** holding the high byte and **Y** the low byte of the VDC RAM location to be accessed. Each set of editor values requires 40 bytes.

When a second storage area is created in an unused character set, another 48 bytes at the top of that area are available for yet another set of editor values. Use the method described above to access $2FD0 for the upper-case/graphics area or $3FD0 for lower/upper-case.

The demo program (Listing 3) assumes that Keep-80 is located at 4864. It uses colours that should be readable on a green screen. For amber monitors some adjustment of the COLOR statements will be necessary. The program creates a window on the left half of the screen and lists itself. Two cursor-ups compensate for the line feeds when the listing is completed. The editor values are saved and a window is opened on the right half. Because of **CHR$(27)"R"** (ESC-R), clearing the window with a different **COLOR 5** creates two different 'backgrounds'. This delineates the two windows. The program again lists itself, saves the right half editor values, then returns to the left half and continues the first listing where it left off. **SLEEP** slows things down for observation. The whole screen is saved and a window with a message is displayed. The Keep-80 program is then poked to create a second storage area at $2000-$2FCF, corresponding with the upper-case/graphics character set which is not used by the demo program due to PRINT CHR$(14). Having created another storage area, the demo again saves the whole screen and displays another message window. When a key is pressed, the process is reversed, recalling the saved screens and thus restoring the characters covered up by the two windows.

More free memory?

Is there still more usable 80-column memory? What about the eight 'pad' bytes of each character definition? This amounts to 2K per character set. Can this highly non-contiguous memory be put to use? Is it worth the trouble?

We have seen the C-128 80-column capability used for hi-res graphics, its memory used as a RAM drive, the unused RAM as a second screen and the application described here. What else can we do with the VDC and its memory?

Listing 1: Keep-80.src

```
AB  1000 sys4000
IG  1010 ;
MP  1020 ;power assembler (buddy128)
MH  1030 ;
LH  1040 ;--------------- keep-80 ---------------
AJ  1050 ;
JA  1060 *= $1300
EK  1070 ;
MN  1080 .mem
IL  1090 ;
NF  1100 ;rom routines
MM  1110 ;
EG  1120 wrvdc = $cdca
JB  1130 rdvdc = $cdd8
PB  1140 vcopy = $c53c
EP  1150 ;
BD  1160 ;ram locations
IA  1170 ;
NC  1180 svars = $00e0;start of screen variables
PA  1190 smaps = $0354;start of tab and link maps
BC  1200 pnt80 = $0a3c;end pointer for vcopy
FK  1210 ztemp = $c3;safe temporary location
KD  1220 ;
EE  1230 ;
JL  1240 keep bit $d7;test 80 columns
OL  1250 bmi ok80
KF  1260 err sec
CO  1270 rts
GH  1280 ;
AH  1290 spage .byte $10;start page of unused area
OH  1300 epage .byte $1f;end page
EJ  1310 ;
LC  1320 ok80 cmp #$03
KK  1330 bcs err
```

```
JF  1340 stx ztemp;direction
FB  1350 tay
PG  1360 bne edda
OA  1370 txa
ND  1380 bne rscrn
EO  1390 ;
IK  1400 ;save whole screen
IP  1410 ;
BP  1420 jsr rend;write editor values to $07d0
MA  1430 ;
GF  1440 lda #$d0;destination end+1
HH  1450 ldx epage
CF  1460 sta pnt80
PJ  1470 stx pnt80+1
KI  1480 lda spage;dest. start
FF  1490 ldy #$00
EP  1500 jsr addwr
EB  1510 lda #$00;source start=$0000
PL  1520 tay
DB  1530 setsrce ldx #$20
GA  1540 jsr addwr+2
OJ  1550 setcopy ldx #$18
GA  1560 jsr rdvdc+2
NL  1570 ora #$80;bit 7=1=copy
LD  1580 jsr wrvdc+2
MN  1590 jsr vcopy;call rom routine
AO  1600 clc;no errors
GD  1610 rts
KM  1620 ;
AH  1630 ;recall whole screen
ON  1640 ;
IP  1650 rscrn lda #$d0;copy everything
EB  1660 ldx #$0f;back to $0000-$0fcf
EC  1670 sta pnt80
BH  1680 stx pnt80+1
NL  1690 lda #$00
DH  1700 tay
GM  1710 jsr addwr
AH  1720 lda spage;source is unused area
FE  1730 ldy #$00
NG  1740 jsr setsrce
ME  1750 ;
AB  1760 rend lda #$07;hi-byte of editor storage
GD  1770 bne edsa
KG  1780 ;
BD  1790 edda lda #$0f;store/recall editor values
GF  1800 clc;at $0fd0 or $1fd0
NM  1810 dey
NH  1820 beq edsa
OD  1830 adc #$10
CE  1840 edsa ldy #$d0;lo-byte
AL  1850 ;
AJ  1860 ;store/recall screen editor values
EM  1870 ;
MJ  1880 editsr jsr addwr
MC  1890 ldy #$1a
MJ  1900 lda ztemp;0=store
HF  1910 beq loop3
GP  1920 ;
LA  1930 loop1 jsr rdvdc
KP  1940 sta svars,y
JF  1950 dey
EJ  1960 bpl loop1
FI  1970 ldy #$0d
PD  1980 loop2 jsr rdvdc
CB  1990 sta smaps,y
LI  2000 dey
HM  2010 bpl loop2
AN  2020 rts
EG  2030 ;
AD  2040 loop3 lda svars,y
HC  2050 jsr wrvdc
HM  2060 dey
EA  2070 bpl loop3
DP  2080 ldy #$0d
HE  2090 loop4 lda smaps,y
JF  2100 jsr wrvdc
JP  2110 dey
HD  2120 bpl loop4
OD  2130 rts
CN  2140 ;
DA  2150 ;routine to write to vdc address registers ($12/$13)
AB  2160 ;or any other pair of registers
EF  2170 ;a=first byte,y=next byte,x=first register
KP  2180 ;
LN  2190 addwr ldx #$12
NG  2200 jsr wrvdc+2;here for other pairs
JF  2210 tya
II  2220 inx
LK  2230 jmp wrvdc+2
MJ  2240 .end
```

Listing 2: Keep-80 loader

```
DF  100 rem *** keep-80 loader ***
CL  110 ke=4864:rem relocating ***
HN  120 ck=0
IJ  130 readd:ck=ck+d:ifd=999then150
NC  140 goto130
LH  150 ifck<>16817thenprint"*** error in data ***":end
ME  160 restore:sa=ke
DH  170 readd:ifd=999then220
PP  180 ifd=>0thenpokesa,d:goto210
AI  190 ad=ke+abs(d):h=ad/256:l=ad-int(ad/256)*256
CO  200 pokesa,l:sa=sa+1:pokesa,h
FI  210 sa=sa+1:goto170
BP  220 print"keep-80 installed"ke"to"sa
EJ  230 print"sys"ke"{left}, a, x"
CE  240 print"a=0 for screen","x=0 to save"
EM  250 print"a=1 for editor 1","x>0 to recall"
NO  260 print"a=2 for editor 2"
OA  270 end
MI  280 :
AE  290 data  36, 215,  48,   4,  56,  96,  16,  31, 201,   3, 176, 248
OL  300 data 134, 195, 168, 208,  76, 138, 208,  45,  32, -89, 169, 208
NO  310 data 174,  -7, 141,  60,  10, 142,  61,  10, 173,  -6, 160,   0
MK  320 data  32,-154, 169,   0, 168, 162,  32,  32,-156, 162,  24,  32
IJ  330 data 218, 205,   9, 128,  32, 204, 205,  32,  60, 197,  24,  96
HK  340 data 169, 208, 162,  15, 141,  60,  10, 142,  61,  10, 169,   0
HF  350 data 168,  32,-154, 173,  -6, 160,   0,  32, -45, 169,   7, 208
LB  360 data   8, 169,  15,  24, 136, 240,   2, 105,  16, 160, 208,  32
BB  370 data -154, 160,  26, 165, 195, 240,  21,  32, 216, 205, 153, 224
LE  380 data   0, 136,  16, 247, 160,  13,  32, 216, 205, 153,  84,   3
CK  390 data 136,  16, 247,  96, 185, 224,   0,  32, 202, 205, 136,  16
FL  400 data 247, 160,  13, 185,  84,   3,  32, 202, 205, 136,  16, 247
BK  410 data  96, 162,  18,  32, 204, 205, 152, 232,  76, 204, 205, 999
```

Listing 3: Keep-80 demo

```
EE  100 bank15:keep=4864:rem start of ml
IO  110 pokekeep+6,16:pokekeep+7,31:rem storage @ $1000-$1fcf
NK  120 graphic5:color6,1:color5,12
KJ  130 print"{home}{home}{clr}"chr$(14)chr$(27)"r";
DB  135 rem full-size reverse screen, lower/uppercase
PE  140 window0,0,39,24,1:a=1:x=0:gosub280
PI  150 color5,15:window40,0,79,24,1:a=2:x=0:gosub280
NF  160 a=1:x=1:gosub290:list210-
AM  170 a=0:x=0:gosub290
DE  180 color5,1:window12,7,37,11,1
PN  190 color5,8
KN  200 printchr$(15)chr$(18)"{2 down}{3 right} your message here ";:sleep1
CK  210 pokekeep+6,dec("20"):pokekeep+7,dec("2f")
JB  215 rem move storage to upper/graphics
PI  220 gosub290:window19,8,43,12,1:printchr$(143);
NG  230 print"{2 down}{5 right} press any key ";:getkey a$
MJ  240 x=1:gosub290:sleep2
MA  250 pokekeep+6,16:pokekeep+7,31:gosub290:sleep1
BA  260 a=2:gosub290:list210-
JH  270 sys49191:end:regain char set
DH  280 list-200:print"{up}{up}";
KM  290 sys keep,a,x:sleep1:return
```

KERNAL++

Add a DOS wedge to your C64 Kernal

by William Coleman

Kernal++ is a Kernal enhancement for your C64. It adds a built-in DOS wedge, auto-loading of BASIC or ML programs at power-up, additional screen editor commands, and several other patches that make using the 64 easier.

DOS commands

The DOS Wedge intercepts the crunch vector ($0304-$0305), so program execution speed won't be affected. All wedge commands must start at the first position of a *logical* line. The following commands are supported:

% Load an ML program (same as ,8,1). The end of program pointers are *not* modified, so you can load ML without affecting BASIC. However, for this reason, don't try to load a BASIC program with this command.

/ Load a BASIC program.

↑ Load and run a BASIC program.

← Save a BASIC program.

= Verify the program in memory with a file on the disk.

\# Display a sequential file on the screen. The RUN/STOP key will abort the display. No character checking is done; cursor commands and colour changes will print, so be careful what you try to display. Only SEQ files will work, though you can of course modify the code to display other types.

All of the above commands have the same syntax: **%filename**. You don't need quotes. However, if you list a directory and place one of these characters in the first position of a line with a filename on it, the command will execute properly.

The following commands all begin with '@'. You can also use '>' instead if you prefer.

@ Read error channel.

@#<number> Change the drive number the wedge accesses. The number can be from 4 (yes it's possible to have a drive #4) to 9. To use device 10, enter **@:**, and for drive 11, **@;** (this works for most DOS wedges by the way).

@$ Displays the disk directory. The RUN/STOP key will abort.

@<disk command> Send a command to the drive, e.g. **@s0:filename**.

@£ Toggle the write protect status of the disk. If you use this command and then try to write to the disk you'll get a DOS MISMATCH error. Executing it a second time will return the disk to normal. If you list the directory of a protected disk, the Version String (just after the disk name) will read '2e' instead of '2a'. The routine used is based on one by William Fossett. For more information see *Transactor*, Volume 7, Issue 4.

@Q Quit wedge. To re-enable, use **SYS 65526**.

The '!' commands

The commands in this group of BASIC enhancements are preceded by '!':

!d Restore default screen colours. This command will set the screen colours to the power-up configuration, currently a black background with light green text in lower case. You can modify the *color* subroutine in the source code to your own favourites. This subroutine also pokes the value 128 into location 650, which will make all keys repeat.

!<number> Set background and border colours. Use the same number you would use if you were poking locations $D020 and $D021 directly.

!* Un-new BASIC. If you accidentally enter NEW (or hit a reset button), this restores your BASIC program. It's also handy if you inadvertently load a BASIC program with the '%' command. Just use this command to set the pointers properly.

Screen editing

Several new Screen Editor commands have been added. All are activated by pressing the CTRL key at the same time as the key listed. They can also be used from within a program by using the CHR$() code given.

INST/DEL - CHR$(23): Toggles quote mode on and off. Cancelling quote mode will also cancel insert mode if that is active.

CLR/HOME CHR$(22): Homes the cursor to the bottom of the screen.

RETURN CHR$(21): Clears the line that the cursor is on from the cursor to the end of the line.

VERT. CURSOR CHR$(25): Clears the screen from the cursor to the bottom of the screen.

HORIZ. CURSOR CHR$(26): Clears the screen from the line the cursor is on to the top of the screen.

Other goodies

Several other patches are included to enhance the Kernal's operation or change the standard defaults:

The default LOAD device is now 8. **LOAD "0:filename"** will load from the disk instead of the cassette.

The default OPEN device is now 4 with a secondary address of 7. **OPEN 4** now behaves like **OPEN 4,4,7**. These two defaults can of course be changed to suit your needs.

Pressing SHIFT and RUN/STOP together will generate **<RETURN> RUN <RETURN>**. The logo key and RUN/STOP will generate **LOAD "0:*",8,1** without a RETURN. CTRL and RUN/STOP will generate **LOAD "0:*"** without a RETURN.

The screen will not scroll while the SHIFT (or SHIFT LOCK) key is depressed. This is handy when listing BASIC programs.

Holding down the CTRL key while turning on the computer (or hitting the reset button, if you have one) will load the first program on your disk (same as **%0:?***). This is a handy option for booting games and other programs that have an autoloader.

Holding down the SHIFT key while turning on the computer will load the first program on the disk and RUN it (same as **↑0:?***).

Where's the beef?

The wedge is installed where the cassette routines used to be. To prevent crashes, device #1 is patched out - if you try to access it you will receive an ILLEGAL DEVICE error. Because of where the routines are placed, these improvements should be 100 per cent compatible with commercial programs, although you may have to disable the wedge with **@Q** before loading them in.

The commands added to the screen editor are patched into the print-to-screen routine. Commercial programs that may use the new CHR$() values as commands (CTRL-U for example) won't try to print them, so there shouldn't be any interference.

Learn how to burn!

You will need access to an EPROM burner to install these additions, either a commercial model like the Promenade or a home-built model like the one I use, which was featured in *Transactor*, Volume 7, Issue 4. The source code at the end of this article was written for the Abacus assembler, but should work with PAL with only minor changes.

To make the file that will be burned onto the EPROM, do the following:

1) Load your assembler and monitor. Don't run them (my monitor interferes with my assembler, that's why I do it this way).

2) Load in the source code and run it. The first thing it will do is copy BASIC and the Kernal into RAM. This is done from BASIC, so be patient! If you're not using a PAL-compatible assembler (LADS, for instance), you'll have to do this by hand. *Do not* flip out the ROMs yet.

3) When the assembly is finished, enter your monitor and transfer $E000-$FFFF to $3000 (exactly where isn't critical, $2000 would do just as well). Change the contents of memory location 01 to 53 ($35). If you forget, and the ROM isn't switched out, you won't see many improvements when you install the new Kernal! The 'standard' transfer command is:

```
T E000 FFFF 3000
```

4) Now use the monitor to save memory from $3000 to $5000:

```
S 3000 5000 "filename" 08
```

or possibly:

```
S "filename" 08 3000 5000
```

Read the documentation that came with your monitor for the proper syntax.

While you have the new Kernal in RAM, you may as well test it. Hit RUN-STOP/RESTORE, and enter **POKE 1,53** then **SYS 65526**. All of the options should work (except the autoboot of course).

Now burn the file you just saved in accordance with the instructions that came with your EPROM burner. As far as chips go, you have two choices: 2764s or the MCM68764. The former is the cheapest ($6.95, Radio Shack #276-1251), but it's a 28-pin chip so you'll need to build an adapter. The MCM68764 is more expensive (about $16), but it's pin-for-pin compatible with the Kernal chip (2364). By the way, BASIC and 1541 ROMs are the same type as the Kernal.

If you don't have access to an EPROM burner, you can still use the program, either by using the BASIC loader (Listing 2), or by making a file as explained above, using a disk doctor to change the load address to $E000, and booting with the following program (a faster solution would be to write it in ML).

```
1 x=x+1:if x=1 then load"kernal++",8,1
2 fori=40960 to 49151:pokei,peek(i):next
3 poke1,53:sys65526:end
```

Making your own improvements

The wedge occupies memory from $F72C to $FA80. The cassette routines run through $FCE7, so there's plenty of room left for further improvments. There are also a few shorter segments in the original ROM code (mostly tape routines) that can be re-used. You might even be able to squeeze in a mini-monitor (*very* mini)! The possible improvements are limited only by your imagination!

Listing 1: Kernal++.src

```
MO 1000 goto1055
OO 1005 open15,8,15,"s0:kernal++.src":close15:save"0:kernal++.src",8:end
IG 1010 ;
NN 1015 ; --------------------------------
DK 1020 ; "KERNAL++ V1.0    (C) 14 JUNE 87
CG 1025 ; "William Coleman 1431 Pacetti Rd
CA 1030 ; "      aka        Green Cove Spgs
PM 1035 ; "Master Blaster  Florida   32043
GP 1040 ; --------------------------------
LI 1045 ;
KL 1050 ; these 2 lines copy the roms into ram
FP 1055 : for i=57344 to 65535:pokei,peek(i):next
PN 1060 : for i=40960 to 49151:pokei,peek(i):next
BB 1065 sys32768
GJ 1070 .opt oo
KM 1075 .page 65
CK 1080 ; above is for abacus assembler. for pal, use sys 700, delete .page line
DL 1085 ;
GO 1090 ; *** kernal equates ***
NL 1095 ;
FN 1100 second = $ff93
JK 1105 tksa = $ff96
IO 1110 acptr = $ffa5
AB 1115 ciout = $ffa8
OI 1120 untalk = $ffab
HG 1125 unlsn = $ffae
PC 1130 listen = $ffb1
DK 1135 talk = $ffb4
PF 1140 readst = $ffb7
JK 1145 open = $f3d5
KH 1150 close = $f642
KK 1155 chrout = $ffd2
BM 1160 load = $f49e
IL 1165 stop = $ffe1
HA 1170 clall = $ffe7
NA 1175 ;
OG 1180 ; *** other equates ***
HB 1185 ;
PF 1190 basinit = $e3bf;    initialize basic
EE 1195 basmsg = $e422;     power-up message
FA 1200 vecp3 = $e453;      restore pg 3 vectors
GC 1205 setpnts = $e56c;    set charout pntrs
BL 1210 chardone = $e6a8;   exit 4 screen charout
OC 1215 chkcodes = $e72a;   charout (after patch)
FI 1220 clrline = $e9ff;    clear screenline
EL 1225 upordown = $ec44;   chk for case change
GM 1230 save = $e159
AA 1235 border = $d020
AJ 1240 backrnd = $d021
GC 1245 ciapra = $dc00
PC 1250 ciaprb = $dc01
OD 1255 outnum = $bdcd;     print integer
KA 1260 strout = $able;     outputs a string
MA 1265 newstt = $a7ae;     set up statement
HA 1270 runc = $a68e;       set up for run
KE 1275 clear = $a659;      clear basic
FJ 1280 crunch = $a57c;     tokenize line
PC 1285 link = $a533;       relink basic
HM 1290 crvec = $0304;      crunch vector
GD 1295 spckey = $028d;     ctrl,shift,or c=
FH 1300 repeat = $028a;     keybrd repeat flag
AF 1305 inbuf = $0200;      input buffer
EJ 1310 ;
EL 1315 ; *** zero page equates ***
OJ 1320 ;
CO 1325 cpnt = $f3;         pntr to color mem
EA 1330 llynx = $d9;        line link table
CH 1335 insert = $d8;       >0 = insert mode
MK 1340 row = $d6;          cursor row (0-24)
DD 1345 lmax = $d5;         max chars in line
DE 1350 quote = $d4;        >0 = quote mode
LB 1355 column = $d3;       cursor column
DE 1360 rpnt = $d1;         pntr to video matrix
KC 1365 keycnt = $c6;       keybrd buffer count
BO 1370 wejdev = $be;       wedge device #
PP 1375 fname = $bb
FA 1380 device = $ba;       current device
GJ 1385 snd = $b9;          secondary addr
AD 1390 length = $b7;       length of filename
EJ 1395 eal = $ae;          end of load
GH 1400 kflag = $9d;        kernal message flag
NA 1405 st = $90
NI 1410 txtptr = $7a
EP 1415 sov = $2d;          start of variables
DM 1420 sob = $2b;          start of basic
PF 1425 misc = $22
HH 1430 flag = $02;         flag for autoboot
BB 1435 ;
GO 1440 ctrlret = 21;       ctrl-return
GL 1445 ctrlhm = 22;        ctrl-home
LN 1450 ctrlins = 23;       ctrl-ins/del
GH 1455 ctrlvcr = 25;       ctrl-vert cursor
OG 1460 ctrlhcr = 26;       ctrl-hori cursor
FK 1465 ; ---------------------------------------
ED 1470 ;
DI 1475 ; -- patches default device # --
OD 1480 ;
CE 1485 *= $e1da
MI 1490 .byte 8;   load"file" = load"file",8
EP 1495 *= $e228
AH 1500 .byte 4;   open4 = open4,4,7
KD 1505 ldy #7
MF 1510 ;
CH 1515 ; -- patches vector table --
GG 1520 ;
EE 1525 *= $e44b
NL 1530 .word wedge
FH 1535 ;
FI 1540 ; -- modify power up message --
PH 1545 ;
BE 1550 *= $e488
GH 1555 .asc "Kernal++ V1.0 "
OI 1560 ;
KM 1565 ; -- text for load --
IJ 1570 ;
EH 1575 *= $e4b7
HB 1580 loadtxt .asc "load"
OP 1585 .byte 34
GB 1590 .asc "0:*"
```

```
IA  1595 .byte 34
MH  1600 .asc ",8,1"
LL  1605 ;
HM  1610 ; -- patch to for stop keys --
FM  1615 ;
EN  1620 *= $e5ea
DC  1625 jmp onekeys
HC  1630 nop
EL  1635 ldx #5
ON  1640 ;
LK  1645 ; -- patch to print routine --
IO  1650 ;
CJ  1655 *= $e725
CD  1660 jmp chkquote
KE  1665 nop
PE  1670 nop
MM  1675 *= $e7d1
LF  1680 jmp newcodes
LA  1685 ;
JL  1690 *= $e962
MO  1695 jmp wait
KB  1700 ;
JD  1705 ; -- patch to ctrl table --
EC  1710 ;
AO  1715 *= $ec42
MJ  1720 .byte $84
LA  1725 *= $ec78
JN  1730 .byte ctrlins,ctrlret,ctrlhcr
NE  1735 *= $ec7f
DJ  1740 .byte ctrlvcr
FG  1745 *= $ecab
JE  1750 .byte ctrlhm
GE  1755 *= $ecb7
FM  1760 .byte $85
LF  1765 ;
ML  1770 ; -- patch shift-run/stop --
FG  1775 ;
IG  1780 *= $ece7
ML  1785 .byte 13
CK  1790 .asc "run"
GM  1795 .byte 13
OH  1800 ;
HE  1805 ; -- patch out cassette --
II  1810 ;
MJ  1815 *= $f2ce
KC  1820 jmp $f271
LH  1825 *= $f38b
NC  1830 jmp $f713
GF  1835 *= $f539
HD  1840 jmp $f713
II  1845 *= $f65a
DA  1850 nop
IA  1855 nop
KL  1860 ;
BH  1865 ; -- do stop keys --
EM  1870 ;
KL  1875 *= $f65f
EK  1880 onekeys cmp #$83;   shifted
HF  1885 bne ok1
DM  1890 jmp $e5ee
EB  1895 ok1 cmp #$84;       c= key
JG  1900 bne ok2
DF  1905 ldx #13
AI  1910 bne stickit;        always
LC  1915 ok2 cmp #$85;       ctrl key
AI  1920 bne ok3
ON  1925 ldx #9
MA  1930 stickit sei
DH  1935 stx keycnt
AM  1940 okloop lda loadtxt-1,x
LM  1945 sta $0276,x
FF  1950 dex
JB  1955 bne okloop
AA  1960 jmp $e5cd
BP  1965 ok3 jmp $e5fe
IC  1970 ;
HA  1975 ; -- activates wedge --
CD  1980 ;
FB  1985 *= $f72c
II  1990 wedgeon jsr vecp3
OA  1995 lda #$08
GD  2000 sta wejdev
BM  2005 rts
AF  2010 ;
FF  2015 ;
OA  2020 ; -- wedge proper --
PF  2025 ;
NP  2030 wedge ldx txtptr;   if not input buffer
BF  2035 bne doreg;          then crunch
EF  2040 cmp #"@"
JI  2045 beq doat
IF  2050 cmp #">"
DJ  2055 beq doat
FM  2060 cmp #"_"
GE  2065 beq dosave
OL  2070 wdge cmp #"%";        entry from autoboot
LK  2075 beq doml
GN  2080 cmp #"^"
HF  2085 beq doload
DF  2090 cmp #"/"
BG  2095 beq doload
HI  2100 cmp #"="
LG  2105 beq doload
ND  2110 cmp #"!"
II  2115 beq jdobas
NE  2120 cmp #"#"
EL  2125 beq seq
CF  2130 doreg jmp crunch;   normal crunching
NM  2135 ;
HB  2140 jdobas jmp dobas;   springboard
HN  2145 ;
MJ  2150 ; -- save routine _ --
BO  2155 ;
PE  2160 dosave jsr setup;   set up file params
ID  2165 jsr save;           save program
EC  2170 frmseq jsr prntret; print return
OI  2175 jmp disperr;        display error chan.
KP  2180 ;
LD  2185 ; -- set up for load --
EA  2190 ;
FH  2195 doml lda #1
DH  2200 .byte $2c
BL  2205 doload lda #0
OH  2210 jmp loadit
NB  2215 ;
AO  2220 ; -- read seq file --
HC  2225 ;
PB  2230 seq lda inbuf+1
HJ  2235 beq done;         exit if just #
OL  2240 jsr setup;        set up file parameters
LL  2245 ldy length;       length of filename
KK  2250 iny
KK  2255 lda #","
CC  2260 sta inbuf,y
BE  2265 iny;              add two
OC  2270 lda #"s"
CA  2275 sta inbuf,y;      append ',s'
HH  2280 sty length;       save new length
JA  2285 jsr yoohoo;       tell drive to talk
EE  2290 lda #25;          ctrl-return
PA  2295 jsr chrout;       clear to bottom
EG  2300 seql lda st
JK  2305 bne sqout;        exit if st set
LG  2310 jsr stop
LD  2315 beq sqout;        also check stop key
PJ  2320 jsr acptr;        get a byte
AI  2325 jsr chrout;       and print it
PL  2330 jmp seql;         loop back
FJ  2335 ;
```

```
DL  2340 sqout jsr close;  close file
DJ  2345 jmp frmseq;       exit
EK  2350 ;
FH  2355 ; -- parse @ commands --
OK  2360 ;
IA  2365 doat jsr setup;  set up file parameters
GN  2370 lda inbuf+1
NN  2375 beq jdisperr;    just @
BF  2380 cmp #"#"
BJ  2385 beq chgdev
FO  2390 cmp #"q"
IB  2395 beq quit
IG  2400 cmp #"$"
IM  2405 beq dir
KB  2410 cmp #"\"
NC  2415 beq jwprot
KO  2420 ;
KI  2425 ; -- send string to error channel --
EP  2430 ;
IO  2435 jsr hello;          make drive listen
DN  2440 ldy #0
KF  2445 daloop lda inbuf+1,y; send string
ND  2450 jsr ciout;          to drive
HH  2455 iny
DO  2460 cpy length
IP  2465 bne daloop
KO  2470 jsr unlsn
JL  2475 done jmp bye
AD  2480 jdisperr jmp disperr; read error chan.
LC  2485 ;
HC  2490 jwprot beq wprot;     springboard
FD  2495 ;
GH  2500 ; -- disable wedge --
PD  2505 ;
IB  2510 quit lda #<crunch;  restore default
KI  2515 sta crvec        ;  crunch vector
JI  2520 lda #>crunch
KK  2525 sta crvec+1
IF  2530 ;
HO  2535 ; -- change wedge device --
CG  2540 ;
MN  2545 chgdev lda inbuf+2
JA  2550 and #$0f
BG  2555 sta wejdev
GH  2560 ;
BC  2565 ; -- common exit point --
AI  2570 ;
FB  2575 bye jsr $a67a;      part of clear
KP  2580 jmp $a47b;          main basic loop
PI  2585 ;
AB  2590 ; -- list directory to screen --
JJ  2595 ;
FI  2600 dir jsr yoohoo;  make drive talk
KG  2605 lda #3;          load addr,link,blocks
DF  2610 linein sta $9c
JH  2615 suk jsr acptr;   get byte from drive
IN  2620 sta $9e;         store
LB  2625 jsr acptr;       get another
OP  2630 sta $9f;         store it too
KA  2635 ldx st
ED  2640 bne ddone;       check st
PN  2645 dec $9c;         loop to read in
KF  2650 bne suk;         $9c pairs
GE  2655 ldx $9e;         print decimal
KD  2660 ldy $9f;         number, i.e.
BL  2665 jsr outnum;      number of blks
DK  2670 lda #" "
GG  2675 jsr chrout;      print space
IK  2680 dloop jsr acptr; get a byte
NA  2685 beq endline;     loop till zero (eol)
JF  2690 jsr chrout
HJ  2695 jmp dloop
NM  2700 endline jsr prntret
BC  2705 jsr stop;        check stop key
AM  2710 beq ddone
KI  2715 lda #2
OP  2720 bne linein;       link,blocks
HP  2725 ddone jsr close
IC  2730 jmp bye
FC  2735 ;
DF  2740 prntret lda #13
JB  2745 jmp chrout;       print return
ED  2750 ;
GM  2755 ; -- write (un)protect disk --
OD  2760 ;
JH  2765 ; this routine sends to commands to the
GN  2770 ; drive. the first writes some code and
NN  2775 ; the second one executes that code.
CF  2780 ;
DL  2785 wprot jsr hello
BD  2790 ldy #0
LM  2795 wloop lda protstr,y
PC  2800 jsr ciout
FN  2805 iny
AP  2810 cpy #31
JO  2815 bne wloop
IE  2820 jsr unlsn
DB  2825 jsr hello
JF  2830 ldy #0
IJ  2835 wloop2 lda exestr,y
HF  2840 jsr ciout
NP  2845 iny
JI  2850 cpy #5
FH  2855 bne wloop2
AH  2860 jsr unlsn
KN  2865 jmp disperr
MK  2870 ;
BD  2875 ; these two commands are sent to the
LO  2880 ; drive. the first is a memory write
FM  2885 ; and the second is a memory execute
AM  2890 ;
HH  2895 protstr .asc "m-w"; m-w 00 06 25
JP  2900 .word $0600
HC  2905 .byte 25
DI  2910 jsr $d042;          load bam
BH  2915 lda $0702;          get dos version
KM  2920 eor #4;             a to e/e to a
NO  2925 sta $0702;          store it back
IG  2930 sta $07a6;          directory (2a/e)
HG  2935 lda #$41;           make sure drive
KL  2940 sta $0101;          will write
LH  2945 jsr $ef07;          bam to disk
KL  2950 jmp $d042;          reread bam and exit
BA  2955 ;
DB  2960 exestr .asc "m-e";  m-e 00 06
KD  2965 .word $0600
AB  2970 ;
AH  2975 ; -- load routine % / ^ = --
KB  2980 ;
KN  2985 loadit sta snd
MK  2990 jsr setup;       set up file parameters
OC  2995 ldx sob
EC  3000 ldy sob+1;       get start of basic
MH  3005 lda inbuf;       if verify then
JG  3010 cmp #"=";        accum > 0
ED  3015 beq ver
HL  3020 lda #0
DI  3025 ver jsr load;    load program
KF  3030 bcs lbad;        branch on error
OD  3035 lda st
IM  3040 and #$10
JL  3045 bne lbad2;       branch on st
HJ  3050 lda inbuf
KP  3055 cmp #"%"
EM  3060 beq ldone;       if ml load then done
MA  3065 lda eal
HJ  3070 sta sov
PO  3075 lda eal+1
LL  3080 sta sov+1;       set end of load pntrs
```

```
NE  3085 jsr clear;      reset remaining pntrs
BO  3090 jsr link;       re-link program
BI  3095 jsr runc;       partial clear
JM  3100 lda inbuf
HN  3105 cmp #"^"
KD  3110 bne ldone;      if not ^ then done
GB  3115 lda #0
CL  3120 sta kflag;      suppress kernal mess.
AC  3125 sta inbuf
OF  3130 jmp newstt;     execute next statement
FL  3135 ;
FL  3140 lbad tax
PP  3145 bne lfini
EC  3150 ldx #$1e
OC  3155 .byte $2c
HG  3160 lbad2 ldx #$1c
ID  3165 .byte $2c
EM  3170 ldone ldx #$80;  no error
EC  3175 lda #$ff
MC  3180 sta $3a;        set direct mode
EI  3185 lfini jmp ($0300)
MO  3190 ;
NH  3195 ; -- parse command string --
GP  3200 ;
OB  3205 ; this routine set length and addr
KA  3210 ; parameters of filename in buffer.
FN  3215 ; %    "filename"  will become
JP  3220 ; %filenamelname
DA  3225 ;     this ^^^^^ will be ignored
EB  3230 ;
MB  3235 parse ldy #$02
MN  3240 sty fname+1
IG  3245 dey
ML  3250 sty fname;         filename at $0201
IK  3255 dey; now zero
BA  3260 ploop1 lda inbuf+1,y
HP  3265 beq pdone
MD  3270 cmp #$22
KJ  3275 beq quot
AL  3280 iny
MD  3285 bpl ploop1
CF  3290 quot ldx #0
NP  3295 pmove lda inbuf+2,y; shift string to
JP  3300 sta inbuf+1,x;       start of buffer.
HC  3305 beq x2y;             no trailing quote
EG  3310 cmp #$22
BF  3315 beq x2y
IN  3320 iny
JN  3325 inx
NK  3330 cpx #$25
NP  3335 bne pmove
NH  3340 x2y txa
AO  3345 tay
OH  3350 pdone sty length
HA  3355 rts
GJ  3360 ;
MM  3365 ; -- display error channel --
AK  3370 ;
HE  3375 disperr jsr clrst
NF  3380 lda device
EP  3385 jsr talk
NL  3390 lda #%01101111;     $60+0f
PP  3395 jsr tksa
ID  3400 errloop jsr acptr
EC  3405 jsr chrout
NB  3410 cmp #13
IP  3415 beq errdone
PL  3420 lda st
EO  3425 beq errloop
PJ  3430 errdone jsr untalk
MO  3435 jbye jmp bye
GO  3440 ;
NL  3445 ; -- make disk listen --
AP  3450 ;
KH  3455 hello lda device
IG  3460 jsr listen
IA  3465 lda #%01101111;     $60+0f
HH  3470 jmp second
JA  3475 ;
MB  3480 ; -- make drive talk --
DB  3485 ;
HP  3490 yoohoo lda #%01100000;   $60+0
BF  3495 sta snd;           2ndary addr
IE  3500 jsr open;          open channel
KN  3505 lda device
ON  3510 jsr talk;          make drive talk
ON  3515 lda snd
BL  3520 jmp tksa;          2ndary addr
LD  3525 ;
NM  3530 ; -- setup for drive routines --
FE  3535 ;
BH  3540 setup jsr parse;  parse filename
BA  3545 lda wejdev
DJ  3550 sta device;        set drive #
HL  3555 clrst lda #0
AD  3560 sta st;            clear status
JN  3565 rts
IG  3570 ;
PI  3575 ; -- parse ! routines --
CH  3580 ;
BM  3585 dobas lda inbuf+1
KD  3590 beq jbye;            just !
DH  3595 cmp #"d"
CE  3600 beq default
PC  3605 cmp #"*"
II  3610 beq unnew
LE  3615 cmp #"0"
KG  3620 bcc jbye
JH  3625 cmp #"<"
EL  3630 bcs jbye
NK  3635 sta border
PC  3640 ldy inbuf+2
KH  3645 beq scolor
IL  3650 clc
OP  3655 tya
LL  3660 adc #10
LM  3665 sta border
LE  3670 scolor sta backrnd
JN  3675 jmp bye
GN  3680 ;
LG  3685 default jsr color
IO  3690 jmp bye
FO  3695 ;
FE  3700 ; -- unnew basic --
PO  3705 ;
OO  3710 unnew lda #1
CF  3715 tay
EI  3720 sta (sob),y;    set first link
MF  3725 jsr link;       re-link program
ON  3730 lda misc
IO  3735 sta sov;        link provides the
ID  3740 lda misc+1;     end of program.
CB  3745 sta sov+1;      just move it.
LH  3750 jsr clear
JC  3755 jmp bye
GC  3760 ;
NO  3765 ; -- set default screen colors --
AA  3770 ; -- modify to suit your taste --
FD  3775 ;
CF  3780 color lda #$80
GJ  3785 sta repeat;        make all keys repeat
FF  3790 lda #0;            backround
NE  3795 sta border
DI  3800 nop;               you can insert a
AF  3805 nop;               a lda #xx here.
MP  3810 sta backrnd
BI  3815 lda #153;          char color
DM  3820 jsr chrout
FJ  3825 lda #14;           lowercase
DL  3830 jmp chrout
BH  3835 ;
```

```
BD  3840 ; -- check for autoboot --
LH  3845 ;
BM  3850 autoboot jsr wedgeon
KN  3855 jsr basinit;        initialize basic
OG  3860 jsr color
CN  3865 jsr basmsg;         power up message
EJ  3870 ;
BN  3875 ldx #251
DL  3880 txs; clear stack
KB  3885 lda #1
FP  3890 sta flag;           init load type flag
OH  3895 lda spckey
OG  3900 cmp #1
OF  3905 beq auto1;          if shift key
OH  3910 cmp #4
DC  3915 beq auto2;          if ctrl key
GF  3920 bne fini;           always if no match
OB  3925 auto1 lsr flag;     flag now zero
MI  3930 auto2 jsr clall
MH  3935 ldy #$ff
NO  3940 boot1 iny;          tranfer "0:?*"
EJ  3945 lda star,y;         to input buffer.
JE  3950 sta inbuf+1,y
OF  3955 bne boot1
IP  3960 jsr parse;          parse buffer
CK  3965 lda flag
CK  3970 bne mlload
IP  3975 lda #"^"
HG  3980 .byte $2c
AF  3985 mlload lda #"%"
JC  3990 sta inbuf;          use the wedge
MA  3995 jmp wdge;           to load program.
IN  4000 fini jmp $e386;     to basic
OP  4005 star .asc "0:?*"
OJ  4010 .byte 0
FC  4015 ;
AH  4020 ; -- power up default colors --
PC  4025 ;
JH  4030 setclr jsr color
AI  4035 jmp ($a002)
OD  4040 ;
DA  4045 ; -- stop scroll if shift --
IE  4050 ;
MJ  4055 wait sta $ac
FH  4060 sei
PN  4065 w1 lda #$fd
ID  4070 sta ciapra
BA  4075 lda ciaprb
LN  4080 cmp #%01111111
DA  4085 beq w1;             loop if shift
II  4090 cli
LO  4095 rts
KH  4100 ;
FG  4105 ; -- quote toggle --
EI  4110 ;
BF  4115 chkquote bpl chkq;  part of reg kernal
DC  4120 jmp $e7d4;          key > 128
JL  4125 chkq cmp #ctrlins;  "ctrl-ins pressed?
GO  4130 beq qtog;           yep
NN  4135 jmp chkcodes;       nope
JK  4140 qtog lda insert;    "insert mode?
IF  4145 beq tryq;           nope
OA  4150 qoff lda #0
HF  4155 sta insert;         clear insert
GN  4160 sta quote;          clear quote
FG  4165 beq qdone;          always
PK  4170 tryq lda quote;     "quote mode?
AG  4175 bne qoff;           yep, clear it
HL  4180 inc quote;          nope, set it
OB  4185 qdone jmp chardone
EN  4190 ;
JE  4195 ; -- parse new ctrl codes --
ON  4200 ;
MP  4205 newcodes cmp #ctrlret
AG  4210 beq clr2eol
EN  4215 cmp #ctrlhm
HM  4220 beq bothome
EC  4225 cmp #ctrlvcr
DH  4230 beq clr2bot
CB  4235 cmp #ctrlhcr
DF  4240 beq clr2top
MI  4245 jmp upordown;       check for case change
AB  4250 ;
DF  4255 ; -- clear to end of line --
KB  4260 ;
AL  4265 clr2eol lda #$20;   put a space
NM  4270 sta (rpnt),y;       in video matrix
HK  4275 lda backrnd;        put backround color
DA  4280 sta (cpnt),y;       in color memory
NJ  4285 iny
JK  4290 cpy lmax;           check for eol
HH  4295 bcc clr2eol
KL  4300 beq clr2eol
PI  4305 bcs jchrdone
ME  4310 ;
BA  4315 ; -- cursor to bottom --
GF  4320 ;
CC  4325 bothome ldy #0
BN  4330 ldx #24
EH  4335 jsr $e50c;        jump into clear screen
NN  4340 jchrdone jmp chardone
PG  4345 ;
HD  4350 ; -- clear to bottom of screen --
JH  4355 ;
AN  4360 clr2bot ldx #$19
GE  4365 c2b1 dex;         from the bottom up
NN  4370 cpx row
DO  4375 beq c2b2
LN  4380 lda llynx,x;      clear line links
EP  4385 ora #$80
LK  4390 sta llynx,x
MJ  4395 jsr clrline;      clear line
JN  4400 bmi c2b1;         always
NL  4405 c2b2 jsr $e9f0;   reset pointers
MH  4410 jsr $ea24
GD  4415 ldy column;       clear line the
KC  4420 jmp clr2eol;      cursors on.
PL  4425 ;
JK  4430 ; -- clear to top of screen --
JM  4435 ;
PB  4440 clr2top ldx #$ff
PO  4445 c2t1 inx;         from the top down
BC  4450 lda llynx,x;      clear line links
KD  4455 ora #$80
BP  4460 sta llynx,x
CO  4465 jsr clrline;      clear line
BE  4470 cpx row
EG  4475 bne c2t1
JA  4480 beq jchrdone;     always
LP  4485 ;
LD  4490 ; -- various patches --
FA  4495 ;
IE  4500 *= $fcff
MB  4505 jmp autoboot
GC  4510 *= $fe6f
HK  4515 jmp setclr
AO  4520 *= $ff80
FB  4525 .byte $10;   version byte (1.0)
IC  4530 ;
NK  4535 ; -- sys65526 to reactivate --
CD  4540 ;
LN  4545 *= $fff6;    last jump table entry
NG  4550 jmp wedgeon; is normally unused.
BE  4555 ;
GE  4560 ;
BL  4565 .end
```

Listing 2: Run this to set up the Kernal ROM from BASIC

```
EN  100 gosub 190        : rem set up rom copy routine
CO  110 sys 384          : rem copy roms to ram
BC  120 gosub 190        : rem copy a chunk for replacement rom code
HI  130 if q=0 goto 120  : rem loop till all rom chunks copied
OC  140 poke 1,53        : rem switch out roms
AE  150 sys 65526        : rem activate wedge
AK  160 end
OB  170 :
JI  180 rem read a,n; poke n bytes starting at a
PI  190 read a
JK  200 if a=-1 then q=1: return
KM  210 read n
MK  220 for i=a to a+n-1
HF  230 read b: poke i,b
PM  240 next i
GB  250 return
IH  260 :
JM  1000 data 384,28 : rem poke 28 byte rom copy routine to 384
OJ  1010 data 169, 160,  32, 135,   1, 169, 224, 160,   0, 132, 251, 133
PA  1020 data 252, 162,  32, 177, 251, 145, 251, 200, 208, 249, 230, 252
HE  1030 data 202, 208, 244,  96
EI  1040 :
MI  1050 data 57818,1 :rem 1 byte at $e1da
CL  1060 data 008
PE  1070 data 57896,3 :rem 3 bytes at $e228
EJ  1080 data 004, 160, 007
NH  1090 data 58443,2 :rem 2 bytes at $e44b
NN  1100 data 052, 247
AM  1110 data 58504,14 :rem 14 bytes at $e488
OE  1120 data 203, 069, 082, 078, 065, 076, 043, 043, 032, 214, 049, 046
OO  1130 data 048, 032
DO  1140 data 58551,13 :rem 13 bytes at $e4b7
IH  1150 data 076, 079, 065, 068, 034, 048, 058, 042, 034, 044, 056, 044
HC  1160 data 049
HC  1170 data 58858,6 :rem 6 bytes at $e5ea
JP  1180 data 076, 095, 246, 234, 162, 005
DM  1190 data 59173,5 :rem 5 bytes at $e725
LA  1200 data 076, 025, 250, 234, 234
HP  1210 data 59346,2 :rem 2 bytes at $e7d2
IF  1220 data 058, 250
OO  1230 data 59746,3 :rem 3 bytes at $e962
CF  1240 data 076, 008, 250
IP  1250 data 60482,1 :rem 1 byte at $ec42
DI  1260 data 132
AC  1270 data 60536,3 :rem 3 bytes at $ec78
IF  1280 data 023, 021, 026
OF  1290 data 60543,1 :rem 1 byte at $ec7f
HK  1300 data 025
OI  1310 data 60587,1 :rem 1 byte at $ecab
IL  1320 data 022
LI  1330 data 60599,1 :rem 1 byte at $ecb7
EN  1340 data 133
BK  1350 data 60647,4 :rem 4 bytes at $ece7
AN  1360 data 013, 082, 085, 078
JM  1370 data 62158,3 :rem 3 bytes at $f2ce
ON  1380 data 076, 113, 242
KK  1390 data 62347,2 :rem 2 bytes at $f38b
FA  1400 data 076, 019
JJ  1410 data 62777,2 :rem 2 bytes at $f539
JB  1420 data 076, 019
NM  1430 data 63066,2 :rem 2 bytes at $f65a
KC  1440 data 234, 234
PD  1450 data 63071,39 :rem 39 bytes at $f65f
EE  1460 data 201, 131, 208, 003, 076, 238, 229, 201, 132, 208, 004, 162
ME  1470 data 013, 208, 006, 201, 133, 208, 017, 162, 009, 120, 134, 198
DK  1480 data 189, 182, 228, 157, 118, 002, 202, 208, 247, 076, 205, 229
PF  1490 data 076, 254, 229
JK  1500 data 63276,876 :rem 876 bytes at $f72c
IN  1510 data 032, 083, 228, 169, 008, 133, 190, 096, 166, 122, 208, 036
CJ  1520 data 201, 064, 240, 114, 201, 062, 240, 110, 201, 095, 240, 030
HK  1530 data 201, 037, 240, 038, 201, 094, 240, 037, 201, 047, 240, 033
II  1540 data 201, 061, 240, 029, 201, 033, 240, 007, 201, 035, 240, 026
BD  1550 data 076, 124, 165, 076, 089, 249, 032, 077, 249, 032, 089, 225
JP  1560 data 032, 058, 248, 076, 017, 249, 169, 001, 044, 169, 000, 076
JB  1570 data 140, 248, 173, 001, 002, 240, 094, 032, 077, 249, 164, 183
NL  1580 data 200, 169, 044, 153, 000, 002, 200, 169, 083, 153, 000, 002
PB  1590 data 132, 183, 032, 060, 249, 169, 025, 032, 210, 255, 165, 144
IO  1600 data 208, 014, 032, 225, 255, 240, 009, 032, 165, 255, 032, 210
ME  1610 data 255, 076, 150, 247, 032, 066, 246, 076, 104, 247, 032, 077
PP  1620 data 249, 173, 001, 002, 240, 038, 201, 035, 240, 049, 201, 081
MA  1630 data 240, 035, 201, 036, 240, 054, 201, 092, 240, 025, 032, 050
EE  1640 data 249, 160, 000, 185, 001, 002, 032, 168, 255, 200, 196, 183
CH  1650 data 208, 245, 032, 174, 255, 076, 242, 247, 076, 017, 249, 240
CE  1660 data 094, 169, 124, 141, 004, 003, 169, 165, 141, 005, 003, 173
GD  1670 data 002, 002, 041, 015, 133, 190, 032, 122, 166, 076, 123, 164
HH  1680 data 032, 060, 249, 169, 003, 133, 156, 032, 165, 255, 133, 158
MJ  1690 data 032, 165, 255, 133, 159, 166, 144, 208, 039, 198, 156, 208
KJ  1700 data 238, 166, 158, 164, 159, 032, 205, 189, 169, 032, 032, 210
MI  1710 data 255, 032, 165, 255, 240, 006, 032, 210, 255, 076, 029, 248
DG  1720 data 032, 058, 248, 032, 225, 255, 240, 004, 169, 002, 208, 201
DL  1730 data 032, 066, 246, 076, 242, 247, 169, 013, 076, 210, 255, 032
EL  1740 data 050, 249, 160, 000, 185, 104, 248, 032, 168, 255, 200, 192
OK  1750 data 031, 208, 245, 032, 174, 255, 032, 050, 249, 160, 000, 185
OL  1760 data 135, 248, 032, 168, 255, 200, 192, 005, 208, 245, 032, 174
BM  1770 data 255, 076, 017, 249, 077, 045, 087, 000, 006, 025, 032, 066
HH  1780 data 208, 173, 002, 007, 073, 004, 141, 002, 007, 141, 166, 007
EM  1790 data 169, 065, 141, 001, 001, 032, 007, 239, 076, 066, 208, 077
PO  1800 data 045, 069, 000, 006, 133, 185, 032, 077, 249, 166, 043, 164
BK  1810 data 044, 173, 000, 002, 201, 061, 240, 002, 169, 000, 032, 158
LN  1820 data 244, 176, 047, 165, 144, 041, 016, 208, 047, 173, 000, 002
NA  1830 data 201, 037, 240, 043, 165, 174, 133, 045, 165, 175, 133, 046
BA  1840 data 032, 089, 166, 032, 051, 165, 032, 142, 166, 173, 000, 002
ON  1850 data 201, 094, 208, 019, 169, 000, 133, 157, 141, 000, 002, 076
FC  1860 data 174, 167, 170, 208, 012, 162, 030, 044, 162, 028, 044, 162
HA  1870 data 128, 169, 255, 133, 058, 108, 000, 003, 160, 002, 132, 188
KA  1880 data 136, 132, 187, 136, 185, 001, 002, 240, 029, 201, 034, 240
HM  1890 data 003, 200, 016, 244, 162, 000, 185, 002, 002, 157, 001, 002
OO  1900 data 240, 010, 201, 034, 240, 006, 200, 232, 224, 037, 208, 238
KK  1910 data 138, 168, 132, 183, 096, 032, 084, 249, 165, 186, 032, 180
BG  1920 data 255, 169, 111, 032, 150, 255, 032, 165, 255, 032, 210, 255
EF  1930 data 201, 013, 240, 004, 165, 144, 240, 242, 032, 171, 255, 076
KL  1940 data 242, 247, 165, 186, 032, 177, 255, 169, 111, 076, 147, 255
BM  1950 data 169, 096, 133, 185, 032, 213, 243, 165, 186, 032, 180, 255
KN  1960 data 165, 185, 076, 150, 255, 032, 228, 248, 165, 190, 133, 186
IG  1970 data 169, 000, 133, 144, 096, 173, 001, 002, 240, 209, 201, 068
CH  1980 data 240, 033, 201, 042, 240, 035, 201, 048, 144, 197, 201, 060
KH  1990 data 176, 193, 141, 032, 208, 172, 002, 002, 240, 007, 024, 152
AH  2000 data 105, 010, 141, 032, 208, 141, 033, 208, 076, 242, 247, 032
EN  2010 data 159, 249, 076, 242, 247, 169, 001, 168, 145, 043, 032, 051
KN  2020 data 165, 165, 034, 133, 045, 165, 035, 133, 046, 032, 089, 166
NL  2030 data 076, 242, 247, 169, 128, 141, 138, 002, 169, 000, 141, 032
KM  2040 data 208, 234, 234, 141, 033, 208, 169, 153, 032, 210, 255, 169
LN  2050 data 014, 076, 210, 255, 032, 044, 247, 032, 191, 227, 032, 159
LN  2060 data 249, 032, 034, 228, 162, 251, 154, 169, 001, 133, 002, 173
BG  2070 data 141, 002, 201, 001, 240, 006, 201, 004, 240, 004, 208, 034
KA  2080 data 070, 002, 032, 231, 255, 160, 255, 200, 185, 253, 249, 153
PL  2090 data 001, 002, 208, 247, 032, 228, 248, 165, 002, 208, 003, 169
PB  2100 data 094, 044, 169, 037, 141, 000, 002, 076, 068, 247, 076, 134
HP  2110 data 227, 048, 058, 063, 042, 000, 032, 159, 249, 108, 002, 160
PO  2120 data 133, 172, 120, 169, 253, 141, 000, 220, 173, 001, 220, 201
EB  2130 data 127, 240, 244, 088, 096, 016, 003, 076, 212, 231, 201, 023
OA  2140 data 240, 003, 076, 042, 231, 165, 216, 240, 008, 169, 000, 133
CB  2150 data 216, 133, 212, 240, 006, 165, 212, 208, 244, 230, 212, 076
OO  2160 data 168, 230, 201, 021, 240, 015, 201, 022, 240, 029, 201, 025
GF  2170 data 240, 035, 201, 026, 240, 060, 076, 068, 236, 169, 032, 145
GF  2180 data 209, 173, 033, 208, 145, 243, 200, 196, 213, 144, 242, 240
CF  2190 data 240, 176, 007, 160, 000, 162, 024, 032, 012, 229, 076, 168
KD  2200 data 230, 162, 025, 202, 228, 214, 240, 011, 181, 217, 009, 128
GG  2210 data 149, 217, 032, 255, 233, 048, 240, 032, 240, 233, 032, 036
HK  2220 data 234, 164, 211, 076, 077, 250, 162, 255, 232, 181, 217, 009
CH  2230 data 128, 149, 217, 032, 255, 233, 228, 214, 208, 242, 240, 206
LG  2240 data 64767,3 :rem 3 bytes at $fcff
IG  2250 data 076, 184, 249
FE  2260 data 65135,3 :rem 3 bytes at $fe6f
CF  2270 data 076, 002, 250
NB  2280 data 65408,1 :rem 1 byte at $ff80
CI  2290 data 016
EG  2300 data 65526,3 :rem 3 bytes at $fff6
PI  2310 data 076, 044, 247
BG  2320 data -1
```

Far-Sys for the C64

Reach out and touch some ROM

by Richard Curcio

The Commodore 64 contains 20K of RAM normally unusable from BASIC. Using machine language, however, the BASIC Interpreter and Operating System (Kernal) ROMs can be switched out to allow access to 16K of RAM 'under' them. Another 4K lies under the I/O and character ROM block. Many programs have appeared that use this extra RAM as a storage area or bit-map screen. The utility presented here, *Far-Sys*, provides BASIC with a mechanism for calling machine language located in these 'hidden' areas. Additionally, the utility provides a means for hidden ML to access ROM routines.

Using *Far-Sys*

The syntax for using *Far-Sys* is

```
SYS FAR, TARGET {,a}{,x}{,y}{,s}
```

FAR is the address where *Far-Sys* is located and TARGET is the address of the ML under ROM. The arguments *a*, *x*, *y*, and *s* are optional and, if present, will be loaded into the accumulator, x, y, and status registers respectively. Any argument may be omitted by placing a comma in the corresponding position. Omitted arguments retain the values SYS picks up from locations 780 to 783 ($030C - $030F). For example:

```
SYS FAR, 45056,,8
```

executes a routine at $B000 passing 8 to the x register. Regardless of the value assigned to sr, *Far-Sys* disables IRQs before ROMs are switched out. Upon return to BASIC, addresses 780 to 783 may be PEEKed for results, just like a normal SYS statement.

One POKE is necessary before using *Far-Sys*: POKE FAR +6, BANK. The effect of this poke is similar to, though much simpler than, the BANK command in BASIC 7.0 on the C128. *Far-Sys* provides six 'banks' numbered 0 to 5. These banks should be thought of as temporary configurations in effect *only* during execution of *Far-Sys* code. If too large a value is POKEd into FAR +6, *Far-Sys* will stop with UNDEF'ND STATEMENT ERROR.

Bank 0: This is equivalent to the configuration in effect before executing SYS FAR. This may not be the same as the 64's default configuration since a modified BASIC in RAM could be in effect. The only reason to use this bank would be to more conveniently disable IRQs and pass register values than the normal SYS statement provides.

Bank 1:	$A000 - $BFFF	RAM (BASIC switched out.)
	$D000 - $DFFF	I/O
	$E000 - $FFFF	Kernal ROM
Bank 2:	Same as Bank 1 except;	
	$D000 - $DFFF	Character ROM
Bank 3:	$A000 - $BFFF	RAM
	$D000 - $DFFF	I/O
	$E000 - $FFFF	RAM
	(BASIC and Kernal switched out.)	
Bank 4:	Same as Bank 3 except;	
	$D000 - $DFFF	Character ROM
Bank 5:	Same as Bank 4 except;	
	$D000 - $DFFF	RAM
	(all ROM and I/O switched out.)	

Note that if the machine is already configured with BASIC or BASIC and Kernal in RAM, Banks 1 or 3 also cause no change in configuration.

Why so many?

The different configurations provide a great deal of flexibility. Using Bank 1, a routine under the BASIC ROM could change colour memory or control the SID chip since I/O is visible to the CPU. Using Bank 4, a routine under the Kernal could copy the character ROM into RAM. However, with increased versatility comes increased chance of error. Storing data to the RAM at the $Dxxx block while I/O is present could crash the system. Attempting to call a routine under the Kernal when in Bank 1 or 2 could have the same effect. *Far-Sys* does not compare banks and target addresses. Use caution.

Using FARJSR

Within *Far-Sys* is some code to allow ML in the hidden areas to call ROM routines when the ROMs are switched out. Your hidden routine should follow these steps:

1. Store the address you wish to call in low/high-byte format in zero-page locations $14/$15.

2. Pre-condition any necessary flags by storing the proper value in $030F. One way to do this is PHP, PLA then ORA or AND to set/clear the selected bit(s). *Do not CLI while the Kernal or I/O is not present!*

3. Store any required a, x, and y values in $030C - $030E.

4. JSR FAR +3, where FAR is the beginning of *Far-Sys*.

The C64 will be restored to the configuration in effect before BASIC executed SYS FAR (Bank 0).

As with JSRFAR and JMPFAR in the C128 Kernal, the user should ensure that IRQs and NMIs are handled properly. Step 2, above, is critical in this respect. *Far-Sys* always performs SEI before switching out ROMs and CLI after switching them back, but FARJSR doesn't CLI after switching ROMs in. (NMIs are not affected by SEI and CLI instructions. More about NMIs at the end of this article.) If interrupts are necessary, this *must* be handled by the value in $030F. When the called routine returns to FARJSR, SEI occurs before ROMs are again switched out. The calling routine can then examine $030C-$030F for results, though x and y can be examined directly. These locations will usually be over-written when *Far-Sys* returns to BASIC.

Unlike the C128's JSRFAR, *Far-Sys* and FARJSR always restore the calling configuration - in either direction.

The programs

Program 1 is the BASIC loader for *Far-Sys*. It is designed to relocate the ML if the start address (FAR in line 110) is changed to a location other than 51200. *Far-Sys* can be placed anywhere in normal or 'open' memory. If placing *Far-Sys* at the top of BASIC program space, the top of memory pointer in locations 55-56 should be lowered by at least 157 bytes.

Program 2 is a Demo-Test program to confirm that *Far-Sys* is functioning properly. *Far-Sys* must be located at 51200 for this demo. A short program is POKEd into RAM beginning at 61440 under the Kernal. No bank switching is necessary to POKE to this area, or to locations under the BASIC ROM. Another short program is POKEd to D-block RAM beginning at 53248. Surprisingly, this area can be POKEd from BASIC! The steps to do so, as shown in lines 200-230 of the demo, are similar to those when BASIC is used to copy the character ROM to RAM. First, IRQs are disabled by masking the timer interrupt bit in CIA 1. (Any other sources of interrupts should also be disabled.) Then, I/O is switched out by POKEing the 6510 port at location 1. When character ROM is switched in, D-block behaves like the other ROM regions: POKEs 'fall through' to the underlying RAM. Like the other ROM locations, ML is still necessary to read this RAM. D-block can be read only when *all* ROM is switched out.

The BASIC demo then clears the screen, sets up *Far-Sys* for Bank 3 and executes the ML at 61440 ($F000). This code increments the border color, and uses FARJSR to call the Kernal PLOT routine to position the cursor mid-screen and the BASIC ROM routine, LINPRT which PRINTs a two-byte integer contained in x/a. (See the source listing for *Underkern*.) Control then returns to BASIC via *Far-Sys* and pauses a while to allow the effects to be observed. After the delay, 256 'A's are PRINTed and, after another delay, Bank 5 is set up and the code at 53248 ($D000) is called. This increments the first 256 screen locations, changing the 'A's to 'B's.

Writing hidden ML

There are several ways to write programs under ROM. The code should first be assembled and tested in normal RAM, if possible. If the ML is relocatable, with no absolute JSRs, JMPs, LoaDs or STores within itself, after DATAfication a BASIC Loader can change the start address and POKE the code to RAM under ROM. If not relocatable, a machine language monitor can be used to manually change the absolute addresses of ML assembled in normal RAM. This is tedious at best. The most convenient method is to use an assembler that writes object code to disk. **Load "prog", 8, 1** will bring the ML into hidden RAM, excluding D-block, which can be POKEd as described above. For debugging purposes, there are a few machine language monitors available that can perform their operations on hidden RAM.

Details and possibilities

Far-Sys is arranged so that parts of it may be accessed by other programs. See the subroutines labeled "twobyt", "combyt" and "getargs" in the source listing.

It is not neccessary to use FARJSR if a routine under BASIC needs to call a Kernal routine and the machine is in bank 1 or (possibly) 2. However, a routine under the Kernal or D-block *must* use FARJSR to call any ROM routines.

By changing the contents of the locations labeled "cnfg" and "mask", it is possible to return to a different configuration - though it's hard to see a reason to do so.

When a hidden routine is called, four stack positions are used: two to return to *Far-Sys* and two to return to BASIC. Similarly, using FARJSR uses four more stack positions.

It should be possible to re-write *Far-Sys* as a wedge or TransBASIC module, with an accompanying BANK command. I'm sure that *Transactor* readers can devise many uses and variations of this small but useful program.

IRQs, NMIs and CIAs

As stated earlier, *Far-Sys* always performs SEI before ROMs are switched out. Since I/O and the Kernal are present in bank 1, if a routine under BASIC requires IRQs, CLI will of course take care of them. Also, the 6526 CIA (Complex Interface Adapter) and the VIC-II each contain an ICR, Interrupt Control Register, which can be written to enable or disable IRQ sources.

NMIs are more difficult to deal with. As the name suggests, Non-Maskable Interrupts cannot be disabled by instructions, though sometimes the hardware responsible can be. CIA-2 at $DDxx generates NMIs relating to serial I/O and RS-232 activity. But the ICR is not like a normal memory location. Writing to it enables-disables interrupt sources, but reading it reveals which source generated the interrupt, not the enable-disable status. It's like a read-only register and a write-only register containing different information at the same location. There is no way to determine which NMI sources *had* been enabled so that they may be *re*-enabled after disabling them. *Far-Sys* makes no attempt to deal with NMIs on a 'universal' basis. It is left to the user to handle NMIs properly in a given situation. Not easy.

The real fly in the ointment is the RESTORE key. Unlike the VIC-20, where the RESTORE key connects to a VIA chip, where the resulting NMI can be masked out, the C64's RESTORE key connects to a one-shot which in turn connects directly to the NMI line (through an inverter). If this or any other NMI (or for that matter, IRQ) should occur while *Far-Sys* or any other routine has switched out the Kernal, the computer will crash. Changing the Kernal RAM vectors at $0314-$0319 won't help, because the microprocessor first looks to the 'hardware vectors' in locations $FFFA-$FFFF to find out where it should go when an interrupt or reset occurs. If the Kernal ROM isn't there, the 6510 will use whatever is in the corresponding RAM locations to find its way and will more than likely become hopelessly lost.

There is a partial solution, though. New vectors could be written to the RAM under ROM at $FFFA-$FFFF directing the 6510 to a routine to save the registers and switch the Kernal back in and handle the interrupt, or ignore it. (If the RAM under the Kernal is used for a bit map, 8000 bytes are required, so 192 are still available for 'hidden vectors' and interrupt handling. Make certain any hi-res clear command clears only the first 8000 bytes, not the full 8192.)

Program 1: *BASIC loader for* Far-Sys

```
PI  100 rem *** far-sys ***
DA  110 far=51200:rem relocating ***
HN  120 ck=0
IJ  130 readd:ck=ck+d:ifd=999then150
NC  140 goto130
JF  150 ifck<>11342thenprint"*** error in data ***":end
PH  160 restore:sa=far
DH  170 readd:ifd=999then220
PP  180 ifd=>0thenpokesa,d:goto210
DC  190 ad=far+abs(d):h=ad/256:l=ad-int(ad/256)*256
CO  200 pokesa,l:sa=sa+1:pokesa,h
FI  210 sa=sa+1:goto170
JJ  220 print"far-sys installed"far"to"sa
MB  230 data  76,-18, 76,-78,  0,  0,  0,108
HE  240 data  20,  0,255,246,242,245,241,244
ND  250 data  32,-96,174, -6,224,  6,144,  3
GD  260 data  76,227,168,165,  1,141, -7, 61
BB  270 data -12,141, -8, 32,-124,32,-89,173
EO  280 data  15,  3,  9,  4, 72,173, 12,  3
KC  290 data 174, 13,  3,172, 14,  3, 40, 32
BO  300 data  -9,  8, 72,173, -7,133,  1,104
BK  310 data  40, 88, 96,173, -7,133,  1, 32
HD  320 data  54,225, 32, 71,225,120,173, -8
EF  330 data 133,  1, 96, 32,253,174, 32,138
JK  340 data 173, 76,247,183, 32,121,  0,240
AB  350 data  12, 32,253,174,201, 44,240,  5
OO  360 data  32,158,183, 56, 96, 24, 96, 32
BF  370 data-105,144,  3,142, 12,  3, 32,-105
FE  380 data 144,  3,142, 13,  3, 32,-105,144
EP  390 data   3,142, 14,  3, 32,-105,144,  3
KD  400 data 142, 15,  3, 96,999
```

Program 2: Far-Sys *demo/test* (Far-Sys *must be at 51200)*

```
EE  100 rem *** far-sys demo/test ***
MA  110 far=51200
HN  120 ck=0
PC  130 readd:ifd=-1then150
NH  140 ck=ck+d:goto130
OC  150 ifck<>6830thenprint"data statement error!":end
IL  160 restore
LJ  170 rem *** poke routine to $f000 ***
GM  180 fori=0to55:readd:poke61440+i,d:next
DG  190 rem *** poke routine d-block ***
BN  200 poke56334,peek(56334)and254:rem turn off timer irqs
OG  210 poke1,peek(1)and251:rem switch in chr rom
IP  220 fori=0to8:readd:poke53248+i,d:next
JD  230 poke1,peek(1)or4:rem put back i/o
OM  240 poke56334,peek(56334)or1:rem enable irq
OD  250 printchr$(147);
CA  260 poke far+6,3:sys far,61440:rem execute routine under kernal
JL  270 gosub320
GK  280 printchr$(19);:fori=0to255:print"a";:next
NM  290 gosub320
FE  300 poke far+6,5:sys far,53248:rem execute routine in d-block
GD  310 end
LC  320 for t=0to1500:next
PL  330 return:rem waste some time
CI  340 rem *** underkern ***
JJ  350 data 238, 32,208,169,255,160,240,132
BJ  360 data  20,133, 21, 24,  8,104,141, 15
FF  370 data   3,162, 10,160, 17, 32, 44,240
NK  380 data 169,189,160,205,132, 20,133, 21
CJ  390 data   8,104,141, 15,  3,169,255,170
OH  400 data  32, 44,240, 96,141, 12,  3,142
NF  410 data  13,  3,140, 14,  3, 76,  3,200
LN  420 rem *** move under d-block ***
NL  430 data 162,  0,254,  0, 4,232,208
ME  440 data 250, 96, -1
```

Program 3: *Source code for* Far-Sys

```
MO  1000 sys999
IG  1010 ;
CJ  1020 ;power assembler (buddy)
MH  1030 ;
BB  1040 *= $c800
AJ  1050 ;
IM  1060 .mem
EK  1070 ;
KB  1080 ;------------ far-sys ------------
IL  1090 ;
IP  1100 ;system routines
MM  1110 ;
OG  1120 chrget = $0073
CE  1130 chrgot = $0079
PF  1140 chkcom = $aefd
OG  1150 frmnum = $ad8a
PC  1160 getadr = $b7f7
BK  1170 onebyt = $b79e
CB  1180 ;
NI  1190 ;---------------------------------
GC  1200 ;
HC  1210 farsys  jmp setup
JD  1220 farjsr  jmp relay
EE  1230 ;
```

```
KC 1240 bank   .byte 0         ;poke 0-5 here
JL 1250 cnfg   .byte 0         ;current config
KI 1260 mask   .byte 0         ;new config
MG 1270 ;
AN 1280 jumper  jmp ($0014)
AI 1290 ;
CM 1300 ;table of values to 'and with 6510 port
EJ 1310 ;
HB 1320 msktbl =*
AD 1330 .byte 255              ;bank 0 - no change
MJ 1340 .byte 246              ;bank 1 - bas. out, kern & i/o in
LD 1350 .byte 242              ;bank 2 - bas. out, kern & chr. in
BC 1360 .byte 245              ;bank 3 - bas. & kern out, i/o in
AM 1370 .byte 241              ;bank 4 - bas. & kern out, chr. in
BL 1380 .byte 244              ;bank 5 - all ram
EO 1390 ;
OO 1400 ;
NN 1410 setup   jsr twobyt     ;read address from basic text
GB 1420 :       ldx bank
DB 1430 :       cpx #$06
IK 1440 :       bcc ok
DL 1450 bad     jmp $a8e3      ;display 'undef statement' if bank>5
GI 1460 ok      lda $01
LI 1470 :       sta cnfg
PL 1480 :       and msktbl,x ;mask bits appropo.
EM 1490 :       sta mask
CF 1500 ;
AB 1510 :       jsr getargs
GG 1520 ;
KK 1530 long    jsr romsout
KI 1540 :       lda $030f      ;get srreg
ND 1550 :       ora #$04       ;ensure no irq when plp
NN 1560 :       pha
FC 1570 :       lda $030c
IE 1580 :       ldx $030d
FF 1590 :       ldy $030e
DH 1600 :       plp            ;as per above
KJ 1610 :       jsr jumper     ;goto target
KM 1620 ;
PF 1630 romsin  php            ;back here
BB 1640 :       pha            ;save flags & acc.
KO 1650 :       lda cnfg
FO 1660 :       sta $01        ;roms in
LF 1670 :       pla
EH 1680 :       plp
AF 1690 :       cli
BL 1700 :       rts
EC 1710 ;
DG 1720 ;routine to allow 'hidden' code to call rom routines.
JA 1730 ;assumes address in $14/15, a, x, y and sr in $030c - $030f.
EM 1740 ;also assumes 'cnfg' restores roms and 'mask' is valid
ME 1750 ;
KH 1760 relay   lda cnfg
DE 1770 :       sta $01        ;restore rom(s)
NP 1780 :       jsr $e136      ;part of "sys". loads regs, jmp ($0014)
GP 1790 :       jsr $e147      ;stores regs.
OH 1800 ;
PM 1810 romsout sei
JL 1820 :       lda mask
AM 1830 :       sta $01
ND 1840 :       rts
AL 1850 ;
HE 1860 ;look for comma, get expression 0 - 65535 from basic text
EM 1870 ;
LC 1880 twobyt  jsr chkcom
KL 1890 :       jsr frmnum     ;eval expression
EB 1900 :       jmp getadr     ;two bytes in $14/15
MO 1910 ;
KI 1920 ;this routine returns with carry clear if end of statement or comma
GL 1930 ;followed by comma, carry set and one byte in x if num. expression.
KA 1940 ;
EB 1950 ;
BK 1960 combyt  jsr chrgot     ;current chr.
AK 1970 :       beq comexit    ;end of statement
KH 1980 :       jsr chkcom     ;look for comma and next chr.
FG 1990 :       cmp #$2c       ;another comma"?
OD 2000 :       beq comexit    ;yeah
OC 2010 :       jsr onebyt     ;no. get value
IK 2020 :       sec
LP 2030 :       rts
DG 2040 comexit clc
PA 2050 :       rts
CI 2060 ;
DI 2070 ;routine to read a, x, y, and sr
IJ 2080 ;values from basic text.
AK 2090 ;
EC 2100 getargs jsr combyt     ;first param (.a)
KM 2110 :       bcc xget       ;just a comma. get next
EI 2120 :       stx $030c      ;sareg
MB 2130 xget    jsr combyt     ;next param (.x)
KH 2140 :       bcc yget
LL 2150 :       stx $030d      ;sxreg
AM 2160 yget    jsr combyt     ;get .y
PB 2170 :       bcc sget       ;another comma"?
MN 2180 :       stx $030e      ;syreg
MA 2190 sget    jsr combyt     ;get .sr
OM 2200 :       bcc exreg
FP 2210 :       stx $030f      ;srreg
CP 2220 exreg   rts
MC 2230 ;
```

Program 4: *Source code for "underkern" in demo/test*

```
IG 100 sys999
EO 110 ;
MG 120 *= $f000
DO 130 .obj "underkern"
CA 140 ;
MA 150 ;
NM 160 border = $d020
ED 170 farjsr = $c803
KC 180 ;
FP 190 begin  inc border
BI 200 :      lda #$ff
NN 210 :      ldy #$f0       ;want to call plot
NJ 220 :      sty $14
DJ 230 :      sta $15
EA 240 :      clc            ;will set cursor
BI 250 :      php            ;irqs not needed
JN 260 :      pla
PJ 270 :      sta $030f      ;status reg.
MH 280 :      ldx #$0a       ;row 10
IF 290 :      ldy #$11       ;col 17
CK 300 ;
IG 310 :      jsr regfar     ;acc. not needed
GL 320 ;
BP 330 :      lda #$bd
AD 340 :      ldy #$cd       ;basic linprt @ $bdcd
PB 350 :      sty $14
FB 360 :      sta $15
KE 370 :      php            ;no change in status reg.
BF 380 :      pla
EO 390 :      sta $030f
AN 400 :      lda #$ff       ;two-byte interger in x/a
IJ 410 :      tax            ;will print 65535
KB 420 ;
KN 430 :      jsr regfar     ;.y not needed
OC 440 ;
KE 450 :      rts            ;back to far-sys
CE 460 ;
CN 470 regfar sta $030c      ;prepare registers
GJ 480 :      stx $030d      ;since we jsr'd here
KF 490 :      sty $030e      ;we can safely...
HE 500 :      jmp farjsr
KN 510 .end
```

C128 Parallel Printer Interface

Emulating a parallel interface via the user port

by Bill Brier
Copyright © 1988 Bill Brier

The Commodore C128 has proven to be a popular machine for small business use, primarily because of its low cost, powerful BASIC 7.0 programming language and its business-oriented hardware features.

Unfortunately, a hardware feature that the C128 doesn't have is a parallel printer output (a Centronics interface). Since the Centronics-style printer interface is pretty much the standard in the business world, lack of such an output would seem to limit the usefulness of the C128 as a business system.

One solution to this limitation is to buy a commercial printer interface, for which one can expect to spend anywhere from 50 dollars to over 120 dollars. However, many of the available interfaces offer frill features that aren't necessary for most business printing applications.

A less expensive solution is to make the C128 user (RS-232) port act as a parallel printer output. This solution is practical if the computer's RS-232 functions are not needed.

Employing the user port as a parallel printer output requires that one buy or fabricate a simple cable to connect the port to the printer and wedge a driver program into the computer's operating system.

That is what this article is all about.

For the benefit of those who may not be familiar with the Centronics printer interface system, I'll describe how it operates. Then, I'll cover the hardware interface and the implementation of the driver software, the assembler source for which is given at the end of the article..

(For ease in typesetting, this article uses the * convention to indication low true signals; for example, *RESET means that RESET is a low true signal.)

The Centronics Parallel Interface

The connection scheme of the Centronics parallel interface is:

C128 User Port			Printer	
Pin	Designation	Data Dir.	Pin	Designation
M	*PA2	→	1	*STROBE
C	PB0	→	2	D1
D	PB1	→	3	D2
E	PB2	→	4	D3
F	PB3	→	5	D4
H	PB4	→	6	D5
J	PB5	→	7	D6
K	PB6	→	8	D7
L	PB7	→	9	D8
B	*FLAG2	←	10	*ACKnowledge
A	GND (signal)		16	GND (signal)
			17	GND (shield)
			31	*RESET

The above connection chart was worked out for the popular Star Micronics printers. You might want to check your printer manual for possible differences in the shield GND and RESET connections. On some printers pin 33 is the shield GND instead of pin 17. The balance of the connections are standard for all parallel printers.

The Centronics interface can be described as an eight-bit, asynchronous parallel bus system with hardware handshaking.

The term *asynchronous* means that data bytes are transmitted at random intervals (no clock is used to synchronize transmission). The term *hardware handshaking* describes the technique used to coordinate the computer and printer so that data is passed in an orderly manner.

Referring to the connection chart, the connections D1 through D8 on the printer (PB0 through PB7 on the computer) pass the data byte (character) to be printed (D1 is equivalent to bit 0). When the printer is not being used the logic levels on these lines will be of no concern.

The *STROBE line, which is controlled by the computer, is one of the two handshaking lines that synchronize the computer and printer. Normally, *STROBE will be held at logic one (high or 5 volts). This is why STROBE and PA2 are shown as 'low true' (and hence are asterisked). The *ACKnowledge line, which is controlled by the printer, is the other handshaking line, and it too will normally be held at logic one.

When the computer has a character to print, it will place the corresponding ASCII data byte on the data lines D1 through D8. A clear bit will be represented by logic zero (low or 0 volts) and a set bit will be represented by logic one.

The computer will then inform the printer that a character is waiting by momentarily bringing the *STROBE line low and then high again (the *STROBE line is said to have been toggled). The printer will respond to the toggling of *STROBE by reading the data byte from the data lines.

When the printer has successfully read the data byte it will signal the computer by toggling the *ACKnowledge line in a manner similar to the way the computer toggled *STROBE. Typically, the computer will wait indefinitely for this to happen. Once the *ACKnowledge has been received, the next character can be transmitted.

No error checking is implemented in this system. If a byte is corrupted for any reason, the printer will not know the difference. Corruption can be avoided by limiting the speed at which data bytes are sent, minimizing the distance between the printer and the computer, and by using shielded cable to connect the computer to the printer.

The *RESET line is not actually part of the data transmission system, as its only function is to cause a hardware reset in the printer when it is pulled low (it is normally high). The actual effect of such a reset will vary from one brand of printer to another. In most cases, a reset will clear the printer's buffer, return the head to the left margin and establish a new top-of-form setting.

Now that I've acquainted you with the Centronics interface, I'll describe the hardware connection of the printer and computer.

The hardware interface

You may purchase or fabricate a cable to connect the C128 user port to the input connector on the printer. If you elect to purchase a cable, verify that it conforms to the connection chart in above (Berkeley Softworks makes a nice but somewhat expensive cable called the geoPrint cable). This connection scheme will work with many word processors that offer a user port printer output (it has been tested with SuperScript 128).

If you decide to build your own interface cable, consult this parts list for the necessary items:

Quantity	Item
1	24-pin male PC board edge user port connector
2	36-pin male plug to fit printer receptacle
1	36-pin female receptacle to fit 36 pin plug
A/R	12 conductor shielded or 36 conductor ribbon cable
1	Plastic box, approx. 3-1/4" long x 2-1/4" wide x 1-1/4" high
4	4-40 or 6-32 x 1/2" SEMS head machine screws
1	SPST momentary contact printer reset pushbutton

A source for the 24-pin PC board connector is Jameco Electronics. The other items can be readily procured from local sources such as Radio Shack. The 36-pin plugs must match the type of cable that you intend to use.

I suggest that you mount the 24-pin edge connector and the 36-pin female receptacle to opposite sides of a small plastic box (see photographs). This will make for a more durable and professional-appearing assembly, as well as giving you a place to mount the printer reset button.

Position the 24-pin edge connector so that its centreline will be 7/16" above the bottom surface of the plastic box. This will cause the box to rest on the surface that supports the computer, thus avoiding the application of stress to the connector and the computer's PC board. When connected to the C128, the box will be adjacent to the RGB receptacle. Sufficient room must therefore be provided for the RGB connector from the video monitor.

You may mount the 36-pin female receptacle in any convenient position on the opposite side of the box. Position the reset button so that it is pointed towards the left when the interface is plugged into the computer.

To secure the connector and receptacle to the box, first lay out rectangular slots on the long sides of the box, and cut the slots with a sharp modelling knife. Next, drill either #43 (4-40) or #36 (6-32) pilot holes for the mounting screws, using the connectors to lay out the holes.

Then simply screw the machine screws into the pilot holes to attach the connector and the receptacle. The screws will cold-flow the plastic and make their own threads. Once you have tested your new interface and have verified that it works, you should use a small amount of quick-setting epoxy to permanently bond the connectors to the box for greater durability.

The use of the 36-pin receptacle makes it possible to detach the cable should repairs to the assembly become necessary. If you elect to hard-wire the cable into the box you may omit the receptacle and one of the two 36-pin plugs. Be sure to provide adequate strain relief for the cable.

You will need to fabricate a cable to connect the receptacle on your new interface box to the printer itself. For residential use, I highly recommend the use of shielded cable. Flat ribbon cable, while more economical to purchase and easier to work with, emits too much radiation and may cause radio and television interference problems. The length of a ribbon cable should be limited to six feet.

When using shielded cable, connect the shield to the shield GND pin at the printer end only. Do not terminate the shield at the computer end. Simply insulate it and let it float. There should be no connection between the shield and the signal GND at any point. This is to prevent the shield from acting as an antenna for high-frequency noise. The length of a shielded cable should be limited to ten feet.

When wiring up your cable follow the connection chart above and, if you are using shielded cable and soldered plugs, wire pin number for pin number. If you are using ribbon cable, note that the two plugs must both face the same direction when the cable is folded up (see the photograph of the cable assembly). If in doubt, check your work with some type of continuity checker to avoid an error.

The reset button, while not a required part of the interface, is a useful feature to have in case you wish to reset the printer without shutting it off. It should be wired so as to pull the *RESET line to signal GND when the button is pressed.

In fabricating your interface, you may be as crude or as refined as your time and talents permit. Just be careful to avoid accidentally making incorrect connections or short circuits. The user port is directly connected to the MOS 6526 CIA #2 chip inside the computer. A wiring error may damage the chip and render the computer inoperative. Also, never connect or disconnect the interface while the computer and printer are turned on. An accidental slip of the wrist may bridge connections together, with catastrophic results.

Once the hardware has been connected, always power the printer first. After it has gone through its power-up sequence you may turn on the computer. When enabled, the driver software will configure the user port for output and will set up the STROBE and ACKnowledge lines to the proper logic levels.

The driver software

The C128 user port is an eight bit I/O port with hardware handshaking provisions. It is connected to the CIA #2 chip and is seen in the $DD00 range of the system map. Normally, this port is addressed via the Kernal RS-232 routines, and is typically used to communicate with a modem. If the port is to be used for some other purpose, suitable driver software must be written to implement the desired functions.

The driver software presented here configures the user port so that it emulates a Centronics printer output. This is accomplished by two machine language modules designated *PPD6656* and *PPD5632*. *PPD6656* contains the port driver code and operating system wedges, while *PPD5632* contains the code used to set up or deactivate the driver module. Upon activation of the driver, the *PPD5632* module is no longer required in memory, and may be overwritten without any effect on the system.

The driver is completely transparent to BASIC and to any machine language program that calls the OPEN, CLOSE, CHKOUT, CLRCHN and CHROUT (BSOUT) subroutines in the Kernal via the jump table. Once it has been wedged into the C128 operating system, the driver will intercept calls to the above subroutines and direct output to the port printer when required. Programming considerations will be discussed below.

The driver software's transparency makes it possible to address the user port as a printer using the standard Commodore file handling syntax. You may activate the driver as follows:

- Load the *PPD6656* and *PPD5632* modules into RAM 0 with **BLOAD**.

- Type **SYS 5632,DN,LF** where *DN* is the desired device number (4 through 7) of the port printer and *LF* is the linefeed enable flag. Set *LF* to 1 if you want a linefeed (ASCII 10) sent to the printer after each carriage return (ASCII 13) is sent. Otherwise, set *LF* to 0 to suppress linefeeds.

If a device number of 0 is selected, the driver will be disengaged from the operating system and will no longer function. Selecting a device number outside of the allowable range will result in an ILLEGAL DEVICE NUMBER error. Never attempt to activate or deactivate the driver unless the *PPD6656* module is in memory. Such an error may result in system fatality.

Once a device number has been assigned to the user port printer, any output to that device number will be intercepted and directed to the port. If you assign device 4 to the port and you also have a printer on the serial bus that is device 4, the serial unit will not respond. You may still output to that printer via the low-level Kernal serial bus routines (which are not intercepted).

When opening a file to the user port printer, you may use one of three secondary address (SA) values as part of the **OPEN** file syntax. The effects of the secondary address are as follows:

```
SA  EFFECT

0   Only upper case characters are printed with
     PETSCII/ASCII translation.
5   Transparent mode with no translation... the
     linefeed setting is ignored.
7   Upper and lower case characters are printed with
     translation.
```

Any secondary address other than 5 or 7 will be treated as an SA of 0. The transparent mode (SA 5) results in characters being passed through without alteration by the driver. The linefeed flag setting (LF) will be ignored and a linefeed will not be sent after a carriage return. The transparent mode should be used for printing dot graphics. It may also be used to pass escape sequences.

When an SA of 0 or 7 is used to open the file, the printer will act pretty much like a Commodore printer. Case switching will occur if a cursor up (145) or cursor down (17) character is sent. Any alphabetic character will be translated from PETASCII to ASCII unless it is part of an escape sequence, in which case the character will pass through unchanged.

For example, sending CHR$(65), the PETASCII for 'a', would result in translation to CHR$(97), the corresponding ASCII value. Without translation, the printer would have printed an 'A'.

As mentioned above, if an alphabetic character immediately follows an ESCape character (ASCII 27), no translation of the alphabetic character will occur. This will result in most escape sequences passing through the driver intact. If you prefer, you may open an additional file with an SA of 5 and use it to pass escape sequences.

When a file has been opened with an SA of 0 or 7, the control code CHR$(15) (expanded print off) will be automatically converted to CHR$(20), as most printers will recognize CHR$(20) as expanded off and recognize CHR$(15) as condensed print on. If you need to turn on condensed print, open a file with an SA of 5 and use it to pass the command sequence.

Regardless of the SA used to open the file, no attempt will be made to translate any of the Commodore PET graphics characters. The PETASCII values for those characters will be passed through unchanged, and will produce differing results depending on the printer that you're working with.

Programming considerations

Upon activation of the driver, several Kernal vectors are modified so that the driver intercepts I/O calls. As a result, the driver may be considered part of the C128 operating system. Changing anything in the memory range from $1A00 to $1BF3 may result in system fatality if any file handling routines are called. If you need that memory range for something else, you must load the *PPD5632* module and deactivate the driver. Never deactivate the driver while a file is opened to the port printer.

Attempting to open a file to device 2 (the RS-232 output) will result in an ILLEGAL DEVICE NUMBER error. This is to prevent interference with the driver and the user port setup. No other precautions need be observed to use the driver.

The driver software will loop indefinitely waiting for the printer to ACKnowledge the reception of a data byte. As a result, the system will appear to lock up if the printer is disconnected or is taken off-line. You can break out of this loop with the STOP/RESTORE keypress combination.

Conclusion

I hope that you will find the port driver software a welcome addition to your library of C128 utilities. I also would like to think that you might learn something new by studying the code. The driver should demonstrate that there is nothing to be afraid of when it comes to messing around with the fundamental operation of the computer.

Such hacking can often yield worthwhile improvements to the system. It can also lead the way to a better understanding of how the computer works, which will ultimately give you greater control over the machine and what it can do.

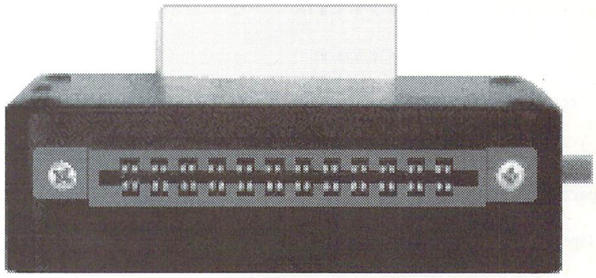


Photo 1. Commodore user port connector.

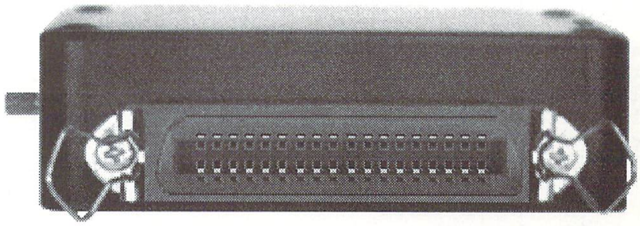


Photo 2. Centronics parallel port connector.

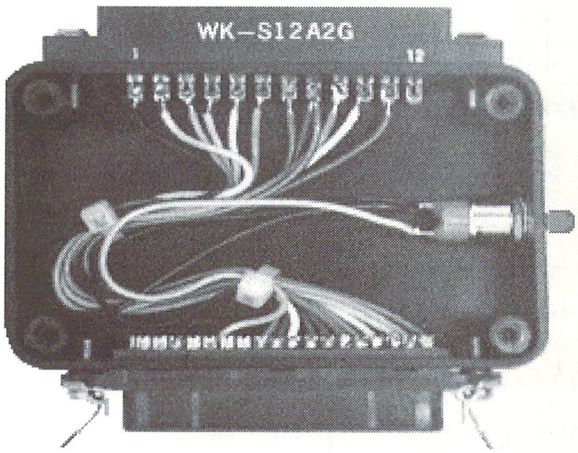


Photo 3. Ribbon cable assembly.

Listing 1: Printer driver source

```
.opt nos
;put"@0:printdriver.src
;* * * * * * * * * * * * * * * * *
;*                              *
;* c-128 centronics printer...  *
;* driver for the user port     *
;*                              *
;* written  1-08-87  w.j. brier *
;*                              *
;* revised                      *
;*                              *
;* copyright (c) 1987           *
;*                              *
;* this program is not to be... *
;* sold.  it is permissible...  *
;* to copy it but credit must...*
;* be given in the documentation*
;*                              *
;* see the documentation for... *
;* instructions on using this...*
;* utility with your software.  *
;*                              *
;* * * * * * * * * * * * * * * * *
                 ;
;* * * * * * * * * * * * * * * * *
;*                              *
;*  <<< program assignments >>> *
;*                              *
;* * * * * * * * * * * * * * * * *
                 ;
;operating system functions...
                 ;
clkspd =$d030 ;system clock speed
                 ;
mmu    =$ff00 ;configuration
                 ;
lkupla =$ff59 ;search for file
indfet =$ff74 ;indirect fetch
chrout =$ffd2 ;output byte
                 ;
;zero page assignments...
                 ;
status =$90   ;i/o status word
ldtnd  =$98   ;number of open files
dfltn  =$99   ;current input device
dflto  =$9a   ;current output device
msgflg =$9d   ;kernal message flag
fnlen  =$b7   ;filename length
la     =$b8   ;file number
sa     =$b9   ;secondary address
fa     =$ba   ;device number
fnadr  =$bb   ;filename address
fnbank =$c7   ;bank holding filename
datax  =$ef   ;character buffer
                 ;
;kernal i/o tables...
                 ;
latbl  =$0362 ;file numbers
fatbl  =$036c ;device numbers
satbl  =$0376 ;secondary addesses
                 ;
;cia #2 registers...
                 ;
d2pra  =$dd00 ;data port a
d2prb  =$dd01 ;data port b
d2ddra =$dd02 ;data direction a
d2ddrb =$dd03 ;data direction b
                 ;
d2icr  =$dd0d ;interrupt control
                 ;
      *=$1a00
                 ;
                 .pag
;# # # # # # # # # # # # # # # # #
;#                              #
;# centronics printer driver 128 #
;#                              #
;# # # # # # # # # # # # # # # # #
       ;
;driver jump table
       ;
       jmp open  ;open file
       jmp close ;close file
       jmp ckout ;open output
       jmp clrch ;close output
       jmp bsout ;output character
       jmp setprt ;set up port
       ;
;---------------------------------
       ;
;alternate indirect vectors
       ;
opena  .byt 0,0
closea .byt 0,0
ckouta .byt 0,0
clrcha .byt 0,0
bsouta .byt 0,0
       ;
       .byt 0,0 ;reserved
       ;
;---------------------------------
       ;
;control flags
       ;
pdev   .byt 0 ;device number
lfflg  .byt 0 ;linefeed flag
       ;
;=================================
       ;
;patch to kernal open routine
       ;
open   lda fa ;current device
       cmp #2 ;rs-232
       beq ilgdev ;illegal device
       ;
       cmp pdev ;port device number
       beq open01
       ;
       jmp (opena) ;not port printer
       ;
open01 lda la ;current file
       jsr lkupla ;search for file
       bcc filopn ;file already open
       ;
       ldx ldtnd ;number of open files
       cpx #10
       beq toomny ;too many files
       ;
       inc ldtnd ;one more file
       sta latbl,x ;add file to table
       ;
       lda fa ;device number
       sta fatbl,x ;add to table
       ;
       lda sa ;secondary address
       cmp #7 ;upper/lower case output
       beq open02
       ;
       cmp #5 ;transparent output
       beq open02
       ;
       lda #0 ;must be 0, 5 or 7
       ;
open02 sta satbl,x ;add to table
       ;
       ldy #0
       sty status ;clear
       sty pmode ;initialize
       cmp #7
       bne open03 ;uppercase only
       ;
       dec pmode ;indicate u.c./l.c.
       ;
open03 jsr setprt ;set up user port
       ;
;output command string...
       ;
       ldy #0
       ;
open04 cpy fnlen ;command string length
       beq open05 ;done
       ;
       lda #fnadr ;filename pointer
       ldx fnbank ;ram bank
       jsr indfet ;fetch character
       ;
       jsr pout ;output character
       iny
       bne open04 ;loop
       ;
open05 lda #0
       clc ;no error
       ;
       rts
       ;
;---------------------------------
       ;
;handle errors
       ;
toomny lda #1 ;too many files
       .byt $2c ;bit op-code
       ;
filopn lda #2 ;file already open
       .byt $2c
       ;
flnopn lda #3 ;file not open
       .byt $2c
       ;
ilgdev lda #9 ;illegal device number
       pha ;save error code
       jsr clrch ;default i/o
       bit msgflg ;kernal message flag
       bvc error3 ;messages disabled
       ;
       ldy #0
       ;
error1 lda errmsg,y ;'i/o error...'
       beq error2 ;end of string
       ;
       jsr bsout ;output message
       iny
       bne error1 ;loop
       ;
error2 pla ;fetch error code
       pha ;write it back
       ora #48 ;change it to ascii
       jsr bsout ;output error number
       ;
error3 pla ;retrieve error code
       sec ;indicate error
       ;
       rts
       ;
;---------------------------------
       ;
;patch to kernal close routine
       ;
close  php ;save status register
       pha ;save file number
       ldx ldtnd ;number of files
       ;
close1 dex ;file table offset
       bpl close3
       ;
close2 pla ;recover file number
       plp ;recover status register
       jmp (closea) ;not port printer
       ;
close3 cmp latbl,x ;file number table
       bne close1 ;not found
       ;
       lda fatbl,x ;fetch device
       cmp pdev
       bne close2 ;not port printer
       ;
       pla ;clear stack
       pla
       dec ldtnd ;one less file
       cpx ldtnd ;check file position
       beq close5 ;no table shift
```

```
        ;
        ldy ldtnd ;new file count
        ;
close4  lda latbl,y ;shift table
        sta latbl,x
        lda fatbl,y
        sta fatbl,x
        lda satbl,y
        sta satbl,x
        ;
close5  lda #0
        clc ;no error
        ;
        rts
        ;
;--------------------------------
        ;
;patch to kernal chkout routine
        ;
ckout   txa ;swap file number
        jsr lkupla ;search for file
        bcs flnopn ;file not open
        ;
        cpx pdev
        beq ckout1 ;port printer
        ;
        tax ;restore file number
        jmp (ckouta) ;not port printer
        ;
ckout1  sta la ;set file number
        stx fa ;set device number
        stx dflto ;set output device
        sty sa ;set secondary address
        jsr setprt ;set up port
        ;
        clc ;no error
        lda #0
        ;
        rts
        ;
;--------------------------------
        ;
;patch to kernal clrchn routine
        ;
clrch   lda dflto ;output device
        cmp pdev ;port printer
        beq clrch1
        ;
        jmp (clrcha) ;normal clrchn
        ;
clrch1  lda #0
        ldy #3
        sta d2prb ;clear output
        sta dfltn ;standard input
        sty dflto ;standard output
        ;
        clc ;all ok
        ;
        rts
        ;
;--------------------------------
        ;
;patch to kernal chrout routine
        ;
bsout   sta datax ;save character
        ;
        lda dflto ;output device
        cmp pdev ;port printer
        beq bout01
        ;
        lda datax ;restore character
        jmp (bsouta) ;normal bsout
        ;
bout01  txa ;preserve registers
        pha
        tya
        pha
        ;
        ldx #0 ;mode flag
        stx status
        lda datax ;fetch character
        ldy sa ;secondary address
        cpy #5
        beq bout08 ;transparent
        ;
        cmp #17 ;cursor down
        bne bout02
        ;
        dex ;set lower case
        bmi bout03
        ;
bout02  cmp #145 ;cursor up
        bne bout04
        ;
bout03  stx pmode ;set mode &...
        jmp bout09 ;exit
        ;
bout04  cmp #27 ;escape
        bne bout05
        ;
        dex
        bmi bout08 ;set escape flag
        ;
bout05  bit escflg
        bmi bout08 ;no conversion
        ;
        cmp #15 ;expanded off
        bne bout06
        ;
        lda #20 ;ascii expanded off
        ;
bout06  bit pmode
        bpl bout07 ;u.c. only
        ;
        cmp #65 ;petscii l.c.
        bcc bout08
        ;
        cmp #91
        bcs bout07
        ;
        ora #32 ;change to ascii l.c.
        ;
bout07  cmp #193 ;petscii u.c.
        bcc bout08
        ;
        cmp #219
        bcs bout08
        ;
        and #127 ;change to ascii u.c.
        ;
bout08  stx escflg ;set/clear
        jsr pout ;write to port
        ;
        lda sa
        cmp #5
        beq bout09 ;transparent output
        ;
        bit lfflg
        bpl bout09 ;linefeeds not enabled
        ;
        cpx #13
        bne bout09 ;not a return
        ;
        lda #10 ;linefeed
        jsr pout
        ;
bout09  pla ;restore registers
        tay
        pla
        tax
        lda datax
        ;
        clc ;all ok
        ;
        rts
        ;
;--------------------------------
        ;
;output to port printer
        ;
pout    tax ;hold character
        tya
        pha ;save .y register
        lda clkspd
        pha ;save clock rate
        ldy #0
        sty clkspd ;slow speed
        sty status ;clear
        ;
        ldy #128
        ;
pout01  dey
        bpl pout01 ;output throttle
        ;
        stx d2prb ;write to port
        nop ;wait 6 microseconds
        nop
        nop
        jsr toggl ;toggle strobe
        ;
        lda #%00010000 ;icr mask
        ;
pout02  bit d2icr ;wait for ack
        beq pout02 ;not received
        ;
        pla
        sta clkspd ;restore clock
        pla
        tay ;restore
        ;
        rts
        ;
;--------------------------------
        ;
;set up user port for output
        ;
setprt  lda #%01111111 ;mask interrupts
        sta d2icr
        ;
        lda d2pra ;port a output
        jsr toggl1 ;set strobe high
        ;
        lda d2ddra ;data direction a
        ora #%00000100 ;set strobe...
        sta d2ddra ;as output
        ;
        ldx #0 ;bring all printer...
        stx d2prb ;output lines low
        dex ;set up port b...
        stx d2ddrb ;as output
        ;
        rts
        ;
;--------------------------------
        ;
;toggle strobe line
        ;
toggl   lda d2pra
        and #%11111011 ;bring...
        sta d2pra ;strobe low &...
        ;
toggl1  ora #%00000100 ;then...
        sta d2pra ;high again
        ;
        rts
        ;
;--------------------------------
        ;
;error message text
        ;
errmsg .byt 13,'i/o error #',0
        ;
;--------------------------------
        ;
;storage
        ;
escflg *=*+1 ;escape mode flag
pmode  *=*+1 ;output mode
        ;
;================================
.end
```

Listing 2: Printer driver set-up source

```
.opt nos
;put"@0:driversetup.src
;* * * * * * * * * * * * * * * * *
;*                               *
;* c-128 centronics printer...   *
;* driver setup module           *
;*                               *
;* written  1-08-87  w.j. brier  *
;*                               *
;* revised                       *
;*                               *
;* copyright (c) 1987            *
;*                               *
;* sys 5632,dn,lf    to enable   *
;* sys 5632,0        to disable  *
;*                               *
;* see the documentation for...  *
;* instructions on using this... *
;* utility with your software.   *
;*                               *
;* * * * * * * * * * * * * * * * *
               ;
;* * * * * * * * * * * * * * * * *
;*                               *
;*  <<< program assignments >>>  *
;*                               *
;* * * * * * * * * * * * * * * * *
               ;
;system vectors & pointers...
               ;
ierror =$0300 ;basic error vector
               ;
iopen  =$031a ;kernal open vector
iclose =$031c ;kernal close vector
ickout =$0320 ;kernal ckout vector
iclrch =$0322 ;kernal clrchn vector
ibsout =$0326 ;kernal chrout vector
               ;
mmu    =$ff00 ;configuration
               ;
;printer driver jump table...
               ;
open   =$1a00 ;kernal open patch
close  =$1a03 ;kernal close patch
ckout  =$1a06 ;kernal chkout patch
clrch  =$1a09 ;kernal clrchn patch
bsout  =$1a0c ;kernal chrout patch
setprt =$1a0f ;port setup
               ;
;printer driver control flags...
               ;
pdev   =$1a1e ;port device number
lfflg  =$1a1f ;linefeed flag
               ;
;alternate indirect vector storage...
               ;
opena  =$1a12 ;open exit
closea =$1a14 ;close exit
ckouta =$1a16 ;chkout exit
clrcha =$1a18 ;clrchn exit
bsouta =$1a1a ;chrout exit
               ;
resrvd =$1a1c ;reserved
               ;
       *=$1600 ;5632
               ;
;================================
        ;
;driver enable/disable
        ;
       stx lfflg ;save linefeed flag
       ;
       ldx #0
       ldy mmu ;get configuration
       stx mmu ;enable roms
       ;
       tax
       bne endr ;enable driver
       ;
       jmp dadr ;disable driver
       ;
;--------------------------------
       ;
;enable driver
       ;
endr   cpx #4 ;check device number
       bcs endr02
       ;
endr01 ldx #9 ;illegal device
       jmp (ierror) ;abort
       ;
endr02 cpx #8
       bcs endr01 ;out of range
       ;
       tya
       pha ;save configuration
       stx pdev ;set device number
       ;
       clc
       lda lfflg
       and #1 ;mask garbage
       ror a ;rotate twice
       ror a
       sta lfflg ;set up flag
       ;
;set up new vectors...
       ;
       ldx iopen ;open vector
       ldy iopen+1
       cpx #<open
       bne endr03
       ;
       cpy #>open
       bne endr03
       ;
       jmp endr04 ;skip setup
       ;
endr03 stx opena
       sty opena+1
       ;
       ldx #<open ;new vector
       ldy #>open
       stx iopen
       sty iopen+1
       ;
       ldx iclose ;close vector
       ldy iclose+1
       stx closea
       sty closea+1
       ;
       ldx #<close ;new vector
       ldy #>close
       stx iclose
       sty iclose+1
       ;
       ldx ickout ;ckout vector
       ldy ickout+1
       stx ckouta
       sty ckouta+1
       ;
       ldx #<ckout ;new vector
       ldy #>ckout
       stx ickout
       sty ickout+1
       ;
       ldx iclrch ;clrchn vector
       ldy iclrch+1
       stx clrcha
       sty clrcha+1
       ;
       ldx #<clrch ;new vector
       ldy #>clrch
       stx iclrch
       sty iclrch+1
       ;
       ldx ibsout ;chrout vector
       ldy ibsout+1
       stx bsouta
       sty bsouta+1
       ;
       ldx #<bsout ;new vector
       ldy #>bsout
       stx ibsout
       sty ibsout+1
       ;
       jsr setprt ;set up user port
       ;
endr04 pla ;old configuration
       sta mmu
       ;
       rts
       ;
;--------------------------------
       ;
;disable driver
       ;
dadr   tya
       pha ;save old configuration
       ;
;check for enabled driver...
       ;
       ldx iopen ;open vector
       ldy iopen+1
       cpx #<open
       bne endr04 ;not enabled
       ;
       cpy #>open
       bne endr04
       ;
;restore vectors...
       ;
       ldx opena ;old open
       ldy opena+1
       stx iopen
       sty iopen+1
       ;
       ldx closea ;old close
       ldy closea+1
       stx iclose
       sty iclose+1
       ;
       ldx clrcha ;old clrchn
       ldy clrcha+1
       stx iclrch
       sty iclrch+1
       ;
       ldx ckouta ;old ckout
       ldy ckouta+1
       stx ickout
       sty ickout+1
       ;
       ldx bsouta ;old bsout
       ldy bsouta+1
       stx ibsout
       sty ibsout+1
       jmp endr04
       ;
;================================
.end
```

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

GEOS LABEL NAMES A handy cross-reference table

Compiled by Francis G. Kostella

Alphabetical Listing

BSW label	Boyce label	hex adr.	Description of routine	AB page	BSW pg.
AppendRecord	APPEND	C289	Add a VLIR chain	1-9	?!
BitmapClip	DRAW	C2AA	Draw a coded image	1-22	94
BitmapUp	CBOX	C142	Draw a click box	1-12	92
BitOtherClip	DRAW2	C2C5	Draw a coded image with user patches	1-22	97
BldGDirEntry	DIRMEM	C1F3	Create a directory entry in memory	1-21	300
BlkAlloc	FALLOC	C1FC	Allocate sectors for a file	1-27	291
BlockProcess	FORBID	C10C	Prevent a timed event from running	1-28	182
BootGEOS	REBOOT	C000	Reboot GEOS	1-48	
BBMult	UMUL88	C160	Unsigned 8 bit by 8 bit multiply	1-56	190
BMult	UM168	C163	Unsigned 16 bit by 8 bit multiply	1-56	191
CalcBlksFree	NUMBLK	C1DB	Compute number of free blocks on disk	1-44	270
CallRoutine	INDJMP	C1D8	Perform an indirect jump	1-32	210
ChangeDiskDevice	CHGDRV	C2BC	Change disk drive device number	1-14	215
ChkDkGEOS	GEOSCK	C1DE	Check if a disk is GEOS format	1-29	256
ClearMouseMode	RESETM	C19C	Reset the mouse	1-49	?!
ClearRam	ZFILL	C178	Fill a memory region with zeros	1-62	206
CloseRecordFile	VCLOSE	C277	Close a VLIR file	1-57	319
CmpFString	BLKCMP	C26E	Memory block comparison	1-10	203
CmpString	STRCMP	C26B	String compare	1-53	202
CopyFString	BLKMOV	C268	Memory block move	1-11	201
CopyString	STRCPY	C265	String copy	1-53	200
CRC	DECODE	C20E	Compute checksum of a memory region	1-20	214
Dabs	ABS16	C16F	16 bit absolute value	1-9	195
Ddec	DEC16	C175	Decrement a 16 bit integer	1-19	197
DeleteFile	DELETE	C238	Delete a file	1-20	266
DeleteRecord	REMOVE	C283	Remove a VLIR chain	1-49	322
DisableSprite	SPROFF	C1D5	Turn off a sprite	1-52	175
Dnegate	NEG16	C172	Negate a 16 bit integer	1-43	196
DoneWithIO	CLSSER	C25F	Close serial communication	1-15	307
DoDlgBox	WINDOW	C256	Window processor	1-60	231
DoIcons	CBOXES	C15A	Draw a table of click boxes (ICONS)	1-13	28
DoInlineReturn	TBLJMP	C2A4	Perform a jump through a table	1-54	?!
DoMenu	MENU	C151	Menu processor	1-42	36
DoPreviousMenu	CLSMNU	C190	Close current menu	1-15	38
DoRAMOp	?	C2D4			?!
DrawLine	LINE	C130	Draw/Erase/Copy an arbitrary line	1-37	78
DrawPoint	PLOT	C133	Draw/Erase/Copy a point on the screen	1-46	72
DrawSprite	COPYSP	C1C6	Copy a sprite data block	1-18	172
DDiv	UD1616	C169	Unsigned 16 bit division	1-55	193
DMult	UM1616	C166	Unsigned 16 bit by 16 bit multiply	1-56	192
DShiftLeft	MASL	C15D	Multiple 16 bit arithmetic shift left	1-41	188
DShiftRight	MLSR	C262	Multiple 16 bit logical shift right	1-43	189
DSDiv	SD1616	C16C	Signed 16 bit division	1-51	194
EnableProcess	EXERTN	C109	Force a recurring timed event to run	1-27	186
EnableSprite	SPRON	C1D2	Turn on a sprite	1-52	174
EnterDeskTop	RESTRT	C22C	Load and run DESKTOP	1-49	269
EnterTurbo	DSETUP	C214	Set up a drive with turboDOS	1-24	309
ExitTurbo	CLRRDY	C232	Stop turboDOS in a drive	1-14	?!
FastDelFile	DELET2	C244	Delete a temporary file	1-20	302
FetchRAM	?	C2CB			?!
FillRam	BLKFIL	C17B	Memory block fill	1-10	207
FindBAMBit	INUSE	C2AD	Check if a disk sector is in use	1-35	296
FindFile	LOOKUP	C20B	Lookup a file in the directory	1-40	263
FindFTypes	TABLE	C23B	Create a table of file names	1-54	257
FirstInit	INIT01	C271	Initialize GEOS variables	1-32	213
FollowChain	TRACE	C205	Create a list of sectors used by file	1-55	301
FrameRectangle	PBOX	C127	Draw an outline in a pattern	1-45	84
FreezeProcess	STOP	C112	Stop a recurring timed event's timer	1-53	183
FreeBlock	?	C2B9	Free a block in the BAM		297
FreeFile	FREE	C226	Free a file's sectors	1-29	304
GetBlock	READ	C1E4	Read a sector	1-48	272
GetCharWidth	CWIDTH	C1C9	Get a character's width	1-19	126
GetDirHead	RD180	C247	Read track 18 sector 0	1-47	281
GetFile	LOAD	C208	Load a file, given a file name	1-37	259
GetFreeDirBlk	HOLE	C1F6	Find a hole in the directory	1-32	289
GetFHdrInfo	LOADAD	C229	Get a file's load address	1-39	276
GetNextChar	GETIN	C2A7	Read a character from the keyboard	1-30	119
GetPtrCurDkNm	DRVNAM	C298	Compute address of disk's name	1-23	254
GetRandom	RANDOM	C187	Change the random number	1-47	198
GetRealSize	CHARST	C1B1	Get a character's stats	1-13	125
GetScanLine	ROWADR	C13C	Compute memory address of screen row	1-50	102
GetSerialNumber	WHATIS	C196	Get user serial number	1-59	211
GetString	INPUT	C1BA	Read a line of text from the user	1-33	111
GoToFirstMenu	CMENUS	C1BD	Close all menu levels	1-16	39
GraphicsString	GRPHIC	C136	Process a graphic command table	1-30	100
HorizontalLine	HLINE	C118	Draw a horizontal line in a pattern	1-31	74
ImprintRectangle	COPYB3	C250	Copy a box from screen 2 to screen 1	1-17	88
InitForIO	OPNSER	C25C	Open serial communication	1-45	306
InitProcesses	CMDTBL	C103	Initialize table of timed events	1-16	180

Alphabetical Listing

BSW label	Boyce label	hex adr.	Description of routine	AB page	BSW pg.
InitRam	BLKSET	C181	Multiple memory location init.	1-11	208
InitTextPrompt	MAKCUR	C1C0	Create the text cursor sprite	1-41	120
InsertRecord	INSERT	C286	Insert a VLIR chain	1-34	?!
InterruptMain	IRQRTN	C100	IRQ routine	1-36	
InvertLine	INVLIN	C11B	Reverse video a horizontal line	1-35	76
InvertRectangle	INVBOX	C12A	Reverse video a box	1-35	86
IsMseInRegion	CKMOUS	C2B3	Check if mouse is inside a window	1-14	153
I_BitmapUp	CBOX2	C1AB	Draw a click box with inline data	1-12	92
I_FillRam	BLKFL2	C1B4	Memory block fill with inline data	1-11	207
I_FrameRectangle	PBOX2	C1A2	Inline Draw a solid outline	1-45	84
I_GraphicsString	GRPHC2	C1A8	Inline Process a graphic cmnd table	1-30	100
I_ImprintRectangle	COPYB4	C253	Inline Copy a box from screen 2 to 1	1-17	88
I_MoveData	INTBM2	C1B7	Inline Intelligent memory block move	1-34	205
I_PutString	DSPTX2	C1AE	Inline Display a text string	1-26	108
I_RecoverRectangle	COPYB2	C1A5	Inline Copy a box from screen 1 to 2	1-17	87
I_Rectangle	PFILL2	C19F	Inline Fill a box with a pattern	1-46	83
LdApplic	LOAD3	C21D	Load and run a file, given dir entry	1-38	284
LdDeskAcc	LOADSW	C217	Load a file with memory swapping	1-39	
LdFile	LOAD2	C211	Load a file, given a directory entry	1-38	287
LoadCharSet	FONT	C1CC	Activate a memory resident font	1-28	132
MainLoop	MAIN	C1C3	GEOS's main loop	1-40	
MouseOff	MOUSOF	C18D	Turn off the mouse	1-43	150
MouseUp	MOUSON	C18A	Turn on the mouse	1-43	151
MoveData	INTBM	C17E	Intelligent memory block move	1-34	205
NewDisk	INITDV	C1E1	Initialize a drive	1-32	283
NextRecord	NEXT	C27A	Move to next VLIR chain	1-44	321
NxtBlkAlloc	FALOC2	C24D	Allocate sectors for a file	1-28	293
OpenDisk	OPNDSK	C2A1	Open a disk	1-44	253
OpenRecordFile	VOPEN	C274	Open a VLIR file	1-58	318
Panic	SYSERR	C2C2	Report system error	1-54	204
PointRecord	GOTO	C280	Goto a specific VLIR chain	1-30	321
PosSprite	POSSPR	C1CF	Position a sprite	1-47	173
PreviousRecord	PREV	C27D	Move to previous VLIR chain	1-47	321
PromptOff	CURSOF	C29E	Turn off the text cursor	1-18	122
PromptOn	CURSON	C29B	Turn on the text cursor	1-18	121
PurgeTurbo	CLRSTS	C235	Stop and remove turbodos in a drive	1-15	308
PutBlock	WRITE	C1E7	Write a sector	1-62	274
PutChar	DSPCHR	C145	Display a character	1-24	123
PutDecimal	DSPNUM	C184	Display a 16 bit integer	1-25	109
PutDirHead	WR180	C24A	Write to track 18 sector 0	1-62	282
PutString	DSPTXT	C148	Display a text string	1-26	108
ReadBlock	READ2	C21A	Read a sector with drive preset	1-48	310
ReadByte	GETBYT	C2B6	Get a byte from a file	1-29	280
ReadFile	LCHAIN	C1FF	Load a chain into memory, given T&S	1-36	277
ReadRecord	VLOAD	C28C	Load a VLIR chain	1-58	324
RecoverAllMenus	ERAMNS	C157	Erase all menus	1-27	?!
RecoverLine	COPYL	C11E	Copy a line from screen 2 to screen 1	1-18	77
RecoverMenu	ERAMNU	C154	Erase the current menu	1-27	?14
RecoverRectangle	COPYB	C12D	Copy a box from screen 1 to screen 2	1-17	87
Rectangle	PFILL	C124	Fill a box with a pattern	1-46	83
RenameFile	RENAME	C259	Rename a file	1-49	268
RestartProcess	ENABLE	C106	Enable a recurring timed event, START	1-26	181
ReDoMenu	DRWMNU	C193	Draw the current menu	1-23	37
RstrAppl	LDSWAP	C23E	Load the SWAPFILE	1-36	?!
RstrFrmDialog	CLSWIN	C2BF	Close a window	1-15	232
SaveFile	SAVE	C1ED	Save memory to a file	1-51	264
SetDevice	DRVSET	C2B0	Select a drive	1-23	252
SetGDirEntry	DIRDSK	C1F0	Create a directory entry on disk	1-21	298
SetGEOSDisk	CONVRT	C1EA	Convert a disk to GEOS format	1-16	255
SetNextFree	ALLOC	C292	Find and allocate a disk block	1-9	295
SetPattern	SETPAT	C139	Select a fill pattern	1-52	82
Sleep	DELAY	C199	Set up a time delay	1-20	184
SmallPutChar	DRAWCH	C202	Draw a character on the screen	1-23	?!
StartAppl	RUN	C22F	Run a program that is in memory	1-50	?!
StartMouseMode	INITMS	C14E	Initialize the mouse	1-33	149
StashRAM	?	C2C8			?!
SwapRAM	?	C2CE			?!
TestPoint	TEST	C13F	Test the value of a pixel	1-55	73
ToBasic	BASIC	C241	Restart BASIC	1-10	212
UnblockProcess	PERMIT	C10F	Allow a recurring timed event to run	1-45	182
UnfreezeProcess	START	C115	Start a recurring timed events timer	1-53	183
UpdateRecordFile	UPDATE	C295	Update a VLIR file	1-57	320
UseSystemFont	SELBSW	C14B	Select the BSW font	1-52	133
VerifyRAM	?	C2D1			?!
VerticalLine	VLINE	C121	Draw a vertical line in a pattern	1-57	75
VerWriteBlock	CWRITE	C223	Verify before writing sector	1-19	?!
WriteBlock	WRITE2	C220	Write a sector with drive preset	1-62	312
WriteFile	SAVE2	C1F9	Save memory to preallocated sectors	1-51	276
WriteRecord	VSAVE	C28F	Save memory to a VLIR chain	1-59	323

GEOS LABEL NAMES

A handy cross-reference table

Compiled by Francis G. Kostella

Sequential Listing

BSW label	Alex B label	hex adr.	Description of routine	AB page	BSW pg.
BootGEOS	REBOOT	C000	Reboot GEOS	1-48	
InterruptMain	IRQRTN	C100	IRQ routine	1-36	
InitProcesses	CMDTBL	C103	Initialize table of timed events	1-16	180
RestartProcess	ENABLE	C106	Enable a recurring timed event,START	1-26	181
EnableProcess	EXERTN	C109	Force a recurring timed event to run	1-27	186
BlockProcess	FORBID	C10C	Prevent a timed event from running	1-28	182
UnblockProcess	PERMIT	C10F	Allow a recurring timed event to run	1-45	182
FreezeProcess	STOP	C112	Stop a recurring timed event's timer	1-53	183
UnfreezeProcess	START	C115	Start a recurring timed events timer	1-53	183
HorizontalLine	HLINE	C118	Draw a horizontal line in a pattern	1-31	74
InvertLine	INVLIN	C11B	Reverse video a horizontal line	1-35	76
RecoverLine	COPYL	C11E	Copy a line from screen 2 to screen 1	1-18	77
VerticalLine	VLINE	C121	Draw a vertical line in a pattern	1-57	75
Rectangle	PFILL	C124	Fill a box with a pattern	1-46	83
FrameRectangle	PBOX	C127	Draw an outline in a pattern	1-45	84
InvertRectangle	INVBOX	C12A	Reverse video a box	1-35	86
RecoverRectangle	COPYB	C12D	Copy a box from screen 1 to screen 2	1-17	87
DrawLine	LINE	C130	Draw/Erase/Copy an arbitrary line	1-37	78
DrawPoint	PLOT	C133	Draw/Erase/Copy a point on the screen	1-46	72
GraphicsString	GRPHIC	C136	Process a graphic command table	1-30	100
SetPattern	SETPAT	C139	Select a fill pattern	1-52	82
GetScanLine	ROWADR	C13C	Compute memory address of screen row	1-50	102
TestPoint	TEST	C13F	Test the value of a pixel	1-55	73
BitmapUp	CBOX	C142	Draw a click box	1-12	92
PutChar	DSPCHR	C145	Display a character	1-24	123
PutString	DSPTXT	C148	Display a text string	1-26	108
UseSystemFont	SELBSW	C14B	Select the BSW font	1-52	133
StartMouseMode	INITMS	C14E	Initialize the mouse	1-33	149
DoMenu	MENU	C151	Menu processor	1-42	36
RecoverMenu	ERAMNU	C154	Erase the current menu	1-27	?14
RecoverAllMenus	ERAMNS	C157	Erase all menus	1-27	?!
DoIcons	CBOXES	C15A	Draw a table of click boxes (ICONS)	1-13	28
DShiftLeft	MASL	C15D	Multiple 16 bit arithmetic shift left	1-41	188
BBMult	UMUL88	C160	Unsigned 8 bit by 8 bit multiply	1-56	190
BMult	UM168	C163	Unsigned 16 bit by 8 bit multiply	1-56	191
DMult	UM1616	C166	Unsigned 16 bit by 16 bit multiply	1-56	192
DDiv	UD1616	C169	Unsigned 16 bit division	1-55	193
DSDiv	SD1616	C16C	Signed 16 bit division	1-51	194
Dabs	ABS16	C16F	16 bit absolute value	1-9	195
Dnegate	NEG16	C172	Negate a 16 bit integer	1-43	196
Ddec	DEC16	C175	Decrement a 16 bit integer	1-19	197
ClearRam	ZFILL	C178	Fill a memory region with zeroes	1-62	206
FillRam	BLKFIL	C17B	Memory block fill	1-10	207
MoveData	INTBM	C17E	Intelligent memory block move	1-34	205
InitRam	BLKSET	C181	Multiple memory location init.	1-11	208
PutDecimal	DSPNUM	C184	Display a 16 bit integer	1-25	109
GetRandom	RANDOM	C187	Change the random number	1-47	198
MouseUp	MOUSON	C18A	Turn on the mouse	1-43	151
MouseOff	MOUSOF	C18D	Turn off the mouse	1-43	150
DoPreviousMenu	CLSMNU	C190	Close current menu	1-15	38
ReDoMenu	DRWMNU	C193	Draw the current menu	1-23	37
GetSerialNumber	WHATIS	C196	Get user serial number	1-59	211
Sleep	DELAY	C199	Set up a time delay	1-20	184
ClearMouseMode	RESETM	C19C	Reset the mouse	1-49	?!
I-Rectangle	PFILL2	C19F	Inline Fill a box with a pattern	1-46	83
I-FrameRectangle	PBOX2	C1A2	Inline Draw a solid outline	1-45	84
I-RecoverRectangle	COPYB2	C1A5	Inline Copy a box from screen 1 to 2	1-17	87
I-GraphicsString	GRPHC2	C1A8	Inline Process a graphic cmnd table	1-30	100
I-BitmapUp	CBOX2	C1AB	Draw a click box with inline data	1-12	92
I-PutString	DSPTX2	C1AE	Inline Display a text string	1-26	108
GetRealSize	CHARST	C1B1	Get a character's stats	1-13	125
I-FillRam	BLKFL2	C1B4	Memory block fill with inline data	1-11	207
I-MoveData	INTBM2	C1B7	Inline Intelligent memory block move	1-34	205
GetString	INPUT	C1BA	Read a line of text from the user	1-33	111
GoToFirstMenu	CMENUS	C1BD	Close all menu levels	1-16	39
InitTextPrompt	MAKCUR	C1C0	Create the text cursor sprite	1-41	120
MainLoop	MAIN	C1C3	GEOS's main loop	1-40	
DrawSprite	COPYSP	C1C6	Copy a sprite data block	1-18	172
GetCharWidth	CWIDTH	C1C9	Get a character's width	1-19	126
LoadCharSet	FONT	C1CC	Activate a memory resident font	1-28	132
PosSprite	POSSPR	C1CF	Position a sprite	1-47	173
EnableSprite	SPRON	C1D2	Turn on a sprite	1-52	174
DisableSprite	SPROFF	C1D5	Turn off a sprite	1-52	175
CallRoutine	INDJMP	C1D8	Perform an indirect jump	1-32	210
CalcBlksFree	NUMBLK	C1DB	Compute number of free blocks on disk	1-44	270
ChkDkGEOS	GEOSCK	C1DE	Check if a disk is GEOS format	1-29	256
NewDisk	INITDV	C1E1	Initialize a drive	1-32	283
GetBlock	READ	C1E4	Read a sector	1-48	272
PutBlock	WRITE	C1E7	Write a sector	1-62	274
SetGEOSDisk	CONVRT	C1EA	Convert a disk to GEOS format	1-16	255
SaveFile	SAVE	C1ED	Save memory to a file	1-51	264
SetGDirEntry	DIRDSK	C1F0	Create a directory entry on disk	1-21	298
BldGDirEntry	DIRMEM	C1F3	Create a directory entry in memory	1-21	300
GetFreeDirBlk	HOLE	C1F6	Find a hole in the directory	1-32	289
WriteFile	SAVE2	C1F9	Save memory to preallocated sectors	1-51	276
BlkAlloc	FALLOC	C1FC	Allocate sectors for a file	1-27	291
ReadFile	LCHAIN	C1FF	Load a chain into memory, given T&S	1-36	277
SmallPutChar	DRAWCH	C202	Draw a character on the screen	1-23	?!
FollowChain	TRACE	C205	Create a list of sectors used by file	1-55	301
GetFile	LOAD	C208	Load a file, given a file name	1-37	259
FindFile	LOOKUP	C20B	Lookup a file in the directory	1-40	263
CRC	DECODE	C20E	Compute checksum of a memory region	1-20	214
LdFile	LOAD2	C211	Load a file, given a directory entry	1-38	287
EnterTurbo	DSETUP	C214	Setup a drive with turbodos	1-24	309
LdDeskAcc	LOADSW	C217	Load a file with memory swapping	1-39	
ReadBlock	READ2	C21A	Read a sector with drive preset	1-48	310
LdApplic	LOAD3	C21D	Load and run a file, given dir entry	1-38	284
WriteBlock	WRITE2	C220	Write a sector with drive preset	1-62	312
VerWriteBlock	CWRITE	C223	Verify before writing sector	1-19	?!
FreeFile	FREE	C226	Free a file's sectors	1-29	304
GetFHdrInfo	LOADAD	C229	Get a file's load address	1-39	276
EnterDeskTop	RESTRT	C22C	Load and run DESKTOP	1-49	269
StartAppl	RUN	C22F	Run a program that is in memory	1-50	?!
ExitTurbo	CLRRDY	C232	Stop turbodos in a drive	1-14	?!
PurgeTurbo	CLRSTS	C235	Stop and remove turbodos in a drive	1-15	308
DeleteFile	DELETE	C238	Delete a file	1-20	266
FindFTypes	TABLE	C23B	Create a table of file names	1-54	257
RstrAppl	LDSWAP	C23E	Load the SWAPFILE	1-36	?!
ToBasic	BASIC	C241	Restart BASIC	1-10	212
FastDelFile	DELET2	C244	Delete a temporary file	1-20	302
GetDirHead	RD180	C247	Read track 18 sector 0	1-47	281
PutDirHead	WR180	C24A	Write to track 18 sector 0	1-62	282
NxtBlkAlloc	FALOC2	C24D	Allocate sectors for a file	1-28	293
ImprintRectangle	COPYB3	C250	Copy a box from screen 2 to screen 1	1-17	88
I-ImprintRectangle	COPYB4	C253	Inline Copy a box from screen 2 to 1	1-17	88
DoDlgBox	WINDOW	C256	Window processor	1-60	231
RenameFile	RENAME	C259	Rename a file	1-49	268
InitForIO	OPNSER	C25C	Open serial communication	1-45	306
DoneWithIO	CLSSER	C25F	Close serial communication	1-15	307
DShiftRight	MLSR	C262	Multiple 16 bit logical shift right	1-43	189
CopyString	STRCPY	C265	String copy	1-53	200
CopyFString	BLKMOV	C268	Memory block move	1-11	201
CmpString	STRCMP	C26B	String compare	1-53	202
CmpFString	BLKCMP	C26E	Memory block comparison	1-10	203
FirstInit	INIT01	C271	Initialize GEOS variables	1-32	213
OpenRecordFile	VOPEN	C274	Open a VLIR file	1-58	318
CloseRecordFile	VCLOSE	C277	Close a VLIR file	1-57	319
NextRecord	NEXT	C27A	Move to next VLIR chain	1-44	321
PreviousRecord	PREV	C27D	Move to previous VLIR chain	1-47	321
PointRecord	GOTO	C280	Goto a specific VLIR chain	1-30	321
DeleteRecord	REMOVE	C283	Remove a VLIR chain	1-49	322
InsertRecord	INSERT	C286	Insert a VLIR chain	1-34	?!
AppendRecord	APPEND	C289	Add a VLIR chain	1-9	?!
ReadRecord	VLOAD	C28C	Load a VLIR chain	1-58	324
WriteRecord	VSAVE	C28F	Save memory to a VLIR chain	1-59	323
SetNextFree	ALLOC	C292	Find and allocate a disk block	1-9	295
UpdateRecordFile	UPDATE	C295	Update a VLIR file	1-57	320
GetPtrCurDkNm	DRVNAM	C298	Compute address of disk's name	1-23	254
PromptOn	CURSON	C29B	Turn on the text cursor	1-18	121
PromptOff	CURSOF	C29E	Turn off the text cursor	1-18	122
OpenDisk	OPNDSK	C2A1	Open a disk	1-44	253
DoInlineReturn	TBLJMP	C2A4	Perform a jump through a table	1-54	?!
GetNextChar	GETIN	C2A7	Read a character from the keyboard	1-30	119
BitmapClip	DRAW	C2AA	Draw a coded image	1-22	94
FindBAMBit	INUSE	C2AD	Check if a disk sector is in use	1-35	296
SetDevice	DRVSET	C2B0	Select a drive	1-23	252
IsMseInRegion	CKMOUS	C2B3	Check if mouse is inside a window	1-14	153
ReadByte	GETBYT	C2B6	Get a byte from a file	1-29	280
FreeBlock	?	C2B9	Free a block in the BAM		297
ChangeDiskDevice	CHGDRV	C2BC	Change disk drive device number	1-14	215
RstrFrmDialog	CLSWIN	C2BF	Close a window	1-15	232
Panic	SYSERR	C2C2	Report system error	1-54	204
BitOtherClip	DRAW2	C2C5	Draw a coded image with user patches	1-22	97
StashRAM	?	C2C8			?!
FetchRAM	?	C2CB			?!
SwapRAM	?	C2CE			?!
VerifyRAM	?	C2D1			?!
DoRAMOp	?	C2D4			?!

Gamemaker's ML Grab-Bag

Splits, sprites and special effects

by Zoltan Hunt

Let's face it, there are more interesting ways of displaying the player's lives than printing a number like '3', '4', or '6'. The same goes for energy, power or strength of the player. Are you tired of writing games with sprites that stop about three-quarters of the way across the screen? How about those keyboard-character displays - wouldn't a hi-res screen with a text window at the bottom look better? While we're at it, how about a more interesting screen clear and a box drawing routine? By using a few simple text-oriented routines to display player status information at the bottom of the screen, along with the split screen and sprite movement routines to handle the hi-res action above, you have a simple toolkit with which to begin building your machine language video game.

Boxes, Strength, Lives and Screens

The BOX routine will print a box of any size at the current cursor position, drawn with the character of your choice.

Example:

```
lda #10     ;box width
sta lx      ;width variable
lda #10     ;box height
sta hi      ;height variable
lda #5      ;distance from left side of screen
sta xd      ;x distance variable
lda #5      ;distance from top of screen
sta yd      ;y distance
lda #"x"    ;character to be printed
sta boxchr  ;character variable
jsr box     ; go!
```

The colour of the box can be specified by something like this:

lda #2: jsr $ffd2: jsr box

This will print the box in red. Another effect possible is by drawing the box in the center and then decreasing XD and YD while increasing LX and HI and calling BOX each time. This will make the box grow, making an interesting finish to a game (by printing blank spaces, this could also be used for clearing screens).

The two routines ENEPNT and ENVPNT will display a value (0-255) in the form of a bar horizontally or vertically on the screen (something like the player's energy levels in ARCHON or similar games). The colour is selected the same way as with BOX, and the bar can be positioned anywhere on the screen using the Kernal routine PLOT ($FFF0) to position the cursor:

```
ldx #row number: ldy #column
clc: jsr plot
```

To use ENEPNT and ENVPNT:

```
lda #45    ;player's energy
sta energy ;energy byte
jsr enepnt ;or envpnt
```

Another use for this routine could be in a graphing program with numbers larger than 255 scaled down (e.g. divided by 2 or 4 or whatever before being stored in ENERGY).

Now we come to the routines PRHMEN and PRVMEN that print a player's lives, ships, shots or whatever. Using these is easy:

```
lda #number ;number you want printed
sta pmem    ;register
jsr prhmen  ;or prvmen
```

The word "men: " can be changed to anything, but be sure to add the right number of cursor-lefts after it. The program, as it is, prints the solid ball character - this can be changed to any other character. Try using different characters and colours to indicate the various values of interest.

Finally, we come to the last routine in this section, CLRSCR. This gives your program that "disintegrating" effect found in some programs. It is called simply with a JSR:

```
jsr clrscr
```

Once again you can change the character it prints - a blank space - to anything you want.

Now we move on to the last two and perhaps most interesting routines.

SPLIT and SEAM

SPLIT splits the screen into two parts: a multi-colour hi-res screen on top, and a regular text screen on the bottom five lines.

Using it is easy. Set the bottom text background colour with the variable IRQTWCOL. Select text or text/hi-res with a 0 or a 1 in IRQSELC.

Here then is a short example:

```
lda #0
sta irqselc  ;set screen to text/hi-res
lda #1
sta irqtwcol ;set text window colour to white
jsr split
```

This routine is one of the most important in many applications, notably games, and is good if you want to easily give the player information, while leaving your richly detailed hi-res masterpiece intact. It can also be used in direct mode, letting you edit or run a program while seeing a high-res screen partially displayed. To change the number of text lines that are displayed, change the byte stored in 'splin'. It is currently set to 20 lines, leaving five lines at the bottom; making it smaller will move the split higher up on the screen. This value can also be changed dynamically, creating a "curtain" effect as the border between graphics and text moves up or down.

Now we come to the last routine: it lets you position a sprite anywhere on the screen without having to work with the sprite registers and numbers greater than 255 (great for machine language programmers)

The best way to show it is through example, so here we go:

```
ldx #40   ;this is half the x position of your sprite
stx xpos ;store it in the x variable
ldy #50   ;this is the y position
sty ylo   ;store in the y variable
lda #0    ;this is sprite you want moved (0-7)
sta xpsnum
jsr seam ;go to it
lda #1    ;sprite to turn on
sta 53269;turn it on
```

This will move your sprite anywhere on the screen. One thing to note though: the X position is doubled, so that storing 40 in 'xpos' will place the sprite at position 80 on the screen. If you need to position a sprite precisely, put the low byte in the accumulator, the high byte (0 or 1) in 'xhi', and jsr 'seam2' instead of 'seam'.

I hope these routines will find their way into some of your programs (I already have one in mind that will make heavy use of SEAM). You should be able to modify them to suit your own needs if required.

```
DP  100 sys 700 ;pal 64
GN  110 .opt oo
BM  120 ; "box"
CN  130 ; draws a box given left edge,
KN  140 ; top edge, width and height
AI  150 ; in "xd", "yd", "lx", "hi".
LL  160 ; character in "boxchr".
AC  170 ;
PI  180 box =*
OM  190 lda #"{clr}"
AF  200 jsr $ffd2 ;optional clear
CD  210 box11 =*
BI  220 ldx yd
OI  230 ldy xd
GG  240 clc
FL  250 jsr $fff0 ;position cursor
LE  260 ldx #0
AH  270 box31 =*
AG  280 lda boxchr
IM  290 jsr $ffd2 ;print char
IA  300 inx
IA  310 cpx lx
CM  320 bne box31
DJ  330 ldx #1
HL  340 box41 =*
MI  350 lda #"{left}"
BA  360 jsr $ffd2
MP  370 lda #"{down}"
FB  380 jsr $ffd2
OM  390 lda boxchr
JC  400 jsr $ffd2
GH  410 inx
EF  420 cpx hi
ED  430 bne box41
BA  440 ldx #1
GC  450 box51 =*
KP  460 lda #"{left}"
PG  470 jsr $ffd2
ML  480 inx
ML  490 cpx lx
OH  500 bne box51
HE  510 ldx #1
NG  520 box61 =*
MB  530 lda #"{up}"
FL  540 jsr $ffd2
CA  550 inx
AO  560 cpx hi
IM  570 bne box61
NI  580 ldx #1
EL  590 box71 =*
GI  600 lda #"{left}"
LP  610 jsr $ffd2
GP  620 lda #"{down}"
PA  630 jsr $ffd2
IM  640 lda boxchr
DC  650 jsr $ffd2
AH  660 inx
OE  670 cpx hi
KD  680 bne box71
LP  690 ldx #1
DC  700 box81 =*
OA  710 lda boxchr
JG  720 jsr $ffd2
GL  730 inx
GL  740 cpx lx
EI  750 bne box81
EO  760 rts
DP  770 lx .byte 15 ;width
PL  790 hi .byte 10 ;height
HN  800 xd .byte 0  ;distance x from side
GE  810 yd .byte 0  ;distance y from top
AI  820 boxchr .asc "*";char used
```

```
DP  100 sys 700 ;pal 64
GN  110 .opt oo
AA  120 ; "enepnt"
LN  130 ; this routine can be used to
```

```
EJ  140 ; show a player's energy level
MA  150 ;
EI  160 enepnt =*
DA  170 lda energy
MF  180 sta ecount
JL  190 eploop =*
CD  200 lda ecount
DF  210 sec
HO  220 sbc #8
ME  230 bcc pntpar
IJ  240 sta ecount
LO  250 lda #<sbox
BF  260 ldy #>sbox
LH  270 jsr $ab1e; print a solid square
HM  280 jmp eploop
ID  290 pntpar =*
GJ  300 lda ecount
BI  310 beq enpnt ;done printing
IB  320 asl: tax  ;index into table
PM  330 lda pntab+1,x
DC  340 tay
EF  350 lda pntab,x
OK  360 jsr $ab1e ;print bar char
IO  370 ;
MO  380 enpnt =*
GI  410 rts
KB  420 ;
EB  430 energy .byte 21 ;player energy
AC  440 ecount .byte 0
OC  450 sbox .asc "{rvs} {rvs off}":.byte 0
LF  460 pnte1 .asc "{logo-g}":.byte 0
HI  470 pnte2 .asc "{logo-j}":.byte 0
LG  480 pnte3 .asc "{logo-k}":.byte 0
HH  490 pnte4 .asc "{logo-k}":.byte 0
HI  500 pnte5 .asc "{rvs}{logo-l}{rvs off}":.byte 0
PG  510 pnte6 .asc "{rvs}{logo-n}{rvs off}":.byte 0
CH  520 pnte7 .asc "{rvs}{logo-m}{rvs off}":.byte 0
II  530 ;
JB  540 pntab .word 0, pnte1, pnte2, pnte3
LI  550 .word pnte4, pnte5, pnte6, pnte7
```

```
DP  100 sys 700 ;pal 64
GN  110 .opt oo
CC  120 ; "envpnt"
PI  130 ; displays a vertical
BD  140 ; bar graph of the value
FE  150 ; in "ecount"
GB  160 ;
CN  170 envpnt =*
HI  180 ;prints 'energy level' vertically
HB  190 lda energy
AH  200 sta ecount
NM  210 eploop =*
GE  220 lda ecount
HG  230 sec
LP  240 sbc #8
AG  250 bcc pntpar
MK  260 sta ecount
PP  270 lda #<sbox
FG  280 ldy #>sbox
PI  290 jsr $ab1e; print a solid square
LN  300 jmp eploop
ME  310 pntpar =*
JM  320 ; print appropriate character
EL  330 lda ecount
BF  340 beq enpte
GD  350 asl: tax ;index into table
BH  360 lda pnvtab+1,x
BE  370 tay
LM  380 lda pnvtab,x
AN  390 jsr $ab1e
GA  400 ;
CP  410 enpte =*
IO  420 lda #19 ;home cursor
HE  430 jsr $ffd2
EK  440 rts
ID  450 ;
JC  460 energy .byte 100 ;player energy
OD  470 ecount .byte 0
PB  480 sbox  .asc "{rvs} {rvs off}{left}{up}":.byte 0
KE  490 pnte1 .asc "{logo-@}{left}{up}":.byte 0
MG  500 pnte2 .asc "{logo-p}{left}{up}":.byte 0
MI  510 pnte3 .asc "{logo-o}{left}{up}":.byte 0
KG  520 pnte4 .asc "{logo-i}{left}{up}":.byte 0
CL  530 pnte5 .asc "{rvs}{logo-u}{rvs off}{left}{up}":.byte 0
LL  540 pnte6 .asc "{rvs}{logo-y}{rvs off}{left}{up}":.byte 0
LI  550 pnte7 .asc "{rvs}{logo-t}{rvs off}{left}{up}":.byte 0
GK  560 ;
CG  570 pnvtab .word 0, pnte1, pnte2, pnte3
JK  580 .word  pnte4, pnte5, pnte6, pnte7
```

```
DP  100 sys 700 ;pal 64
GN  110 .opt oo
DA  120 ; "prhmen"
OM  130 ;this routine prints the player's
MK  140 ;men but could be used to represent
PA  150 ;energy levels, strength, etc
GB  160 ;
FJ  170 prhmen =*
MM  180 lda #<prnmen
CD  190 ldy #>prnmen
LM  200 jsr $ab1e ;print it
IE  210 ;
IF  220 ldx pmen  ;get number of men
GD  230 lda #"Q"  ;this char represents men
PN  240 menlop =*
ME  250 jsr $ffd2 ;print it
OM  260 dex       ;have we printed them all
AH  270 bne menlop
EA  280 rts
OA  290 pmen   .byte 5
AK  300 prnmen .asc "men: "
KC  310 .byte 0
```

```
FD  100 sys 700
JM  110 :.opt oo
PB  120 ; "prvmen"
PN  130 ;this is almost the same as
HB  140 ;prhmen but prints the men
DN  150 ;down insted of across
GB  160 ;
NM  170 prvmen =*
OI  180 lda #<pnmen
EP  190 ldy #>pnmen
LM  200 jsr $ab1e ;print it
OE  210 ldx pmen  ;get number of men
PB  220 menlop lda #"{left}"
PH  230 jsr $ffd2
KH  240 lda #"{down}"
DJ  250 jsr $ffd2
EF  260 lda #"Q"  ;this char represents men
AG  270 jsr $ffd2 ;print it
CO  280 dex       ;have we printed them all
EI  290 bne menlop
IB  300 rts
EK  310 pmen  .byte 5 ;number of men
EO  320 pnmen .asc "men: {left}{left}{left}{left}"
OD  330 .byte 0
```

```
DP  100 sys 700 ;pal 64
GN  110 .opt oo
LO  120 ; "clrscr"
AA  130 ; clears screen using
NI  140 ; "dissolve" effect
MA  150 ;
HJ  160 clrscr =*
BP  170 ldx #0
NC  180 loop =*
AC  190 lda #255
IL  200 sta 54287
KC  210 lda #128   ;set up
JM  220 sta 54290
```

```
CI  230 sta 54296  ;sid chip
AD  240 ldy 54299  ;get random number
BI  250 lda #32
PH  260 sta 1024,y
OA  270 sta 1024+256,y
CA  280 sta 1024+512,y
HD  290 sta 1024+768,y
AB  300 jsr delay
CB  310 inx
MA  320 cpx #254
DO  330 bne loop
EG  340 lda #"{clr}"
HP  350 jsr $ffd2
PH  360 delay txa
OO  370 pha
NM  380 ldx #5
OE  390 clrsbd ldy #0
MC  400 clrsyl dey
JJ  410 bne clrsyl
LF  420 dex
IE  430 bne clrsbd
AE  440 pla
NI  450 tax
IL  460 rts
```

```
DP  100 sys 700 ;pal 64
GN  110 .opt oo
FN  120 ; "split"
GF  130 ; irq driven multi-colour
AL  140 ; hi-res/text screen
PI  150 ; by zoltan hunt, 1988
GB  160 ;
HC  170 split =*
NE  180 sei
PH  190 lda #<main
DI  200 sta $0314
PI  210 lda #>main
IJ  220 sta $0315
FC  230 lda #$81
HN  240 sta $d01a
IG  250 lda #$1b
LN  260 sta $d011
OJ  270 lda #$7f
HD  280 sta $dc0d
AL  290 cli
IB  300 rts
MK  310 ;
HH  320 main =*
HL  330 pha: tya
OL  340 pha: txa
LG  350 pha  ;save a,x,y
FF  360 lda #1
BF  370 sta $d019
GL  380 lda irqselc
IL  390 cmp #1
HA  400 beq irqend
ED  410 lda $d012
AH  420 cmp #60
PL  430 bcc topirq
FN  440 lda 53272  ;set up for text mode
DP  450 and #247
BL  460 sta 53272
MH  470 lda 53265
HA  480 and #223
OM  490 sta 53265
DO  500 lda #2
GN  510 sta $d012
NK  520 lda 53270
GE  530 and #239
PP  540 sta 53270
IL  550 lda irqtwcol
IB  560 sta 53281
FN  570 jmp irqend
KL  580 ;
PM  590 topirq =* ;set up for hires mode
PP  600 lda 53272
JE  610 ora #8
BF  620 sta 53272
MB  630 lda 53265
GM  640 ora #32
OG  650 sta 53265
JD  660 lda 53270
KO  670 ora #16
LI  680 sta 53270
BB  690 lda splin ;split text line
IH  700 asl: asl: asl ;convert to raster
ND  710 adc #50
IK  720 sta $d012
AF  730 ;
IJ  740 irqend =*
PM  750 lda $dc0d
CB  760 lsr a
DI  770 bcc irq2end
JJ  780 pla: tax
HK  790 pla: tay
IK  800 pla
DF  810 jmp $ea31
KK  820 ;
CA  830 irq2end =*
FN  840 pla: tax
DO  850 pla: tay
EO  860 pla
LO  870 jmp $febc
GO  880 ;
GJ  890 irqtwcol .byte 3
MC  900 irqselc  .byte 0  ;hi/text (1)=text
BE  910 splin    .byte 20 ;split text line
```

```
DP  100 sys 700 ;pal 64
GN  110 .opt oo
GN  120 ;  "seam"
ML  130 ; puts a sprite anywhere
PL  140 ; on the screen
AC  150 ; put x/2 in xpos,
BJ  160 ; y in ylo,
IN  170 ; and sprite # in xpsnum.
KC  180 ;
PO  190 seam =* ;uses xpos, ylo, xpsnum
DL  200 lda #0
LD  210 sta xhi
OA  220 lda xpos
BJ  230 asl
LA  240 rol xhi ;holds high bit
MM  250 seam2 =* ;uses xlo, xhi, ylo, xpsnum
HI  260 sta xlo
OI  270 lda xpsnum
IM  280 asl: tax
IG  290 lda ylo
MG  300 sta 53249,x
LH  310 lda xlo
PH  320 sta 53248,x
FH  330 lda xhi
MH  340 bne xpn1
CF  350 ;clear high bit
EE  360 ldx xpsnum
EN  370 lda #255
NP  380 sec
BO  390 sbc xpnum,x
DJ  400 and 53264
NH  410 sta 53264
AJ  420 rts
MK  430 xpn1 =* ;set high bit
EJ  440 ldx xpsnum
HG  450 lda 53264
KE  460 ora xpnum,x
JL  470 sta 53264
MM  480 rts
AG  490 ;
FE  500 xpos   .byte 80  ;sprite x pos / 2
PK  510 ylo    .byte 120 ;y position
PK  520 xlo    .byte 100 ;sprite x pos low
CJ  530 xhi    .byte 0   ;sprite x high bit
OK  540 xpsnum .byte 0   ;sprite # (0-7)
LO  550 xpnum  .byte 1,2,4,8,16,32,64,128
```

The BASIC 7.0 BANK Command

A voyage of discovery in the C128 ROMs

by D.J. Morriss

As is well known, Commodore was experiencing financial difficulties during the development and early marketing of the C128. This may account for the frequent use and misuse of the term 'bank' in connection with the internal architecture of both the C128 and the 1750 RAM Expansion Module.

As has been well explained in earlier issues of the *Transactor*, the term 'bank' in the C128 is most often used to refer to different preselected memory configurations. The Memory Management Unit (MMU) switches different parts of the available 180 kilobytes of RAM and ROM into the C128's 64K of addressable memory, as needed. This switching is going on hundreds of times a second, under the control of the operating system.

For example, a very long BASIC program may occupy RAM in Bank 0 as far as $D600. The program at that point may contain a statement to **PRINT A$**, where the string *A$* could, by coincidence, be stored in Bank 1 starting at that same address, $D600. If the 80-column screen is the active screen, the PRINT statement must pass the string to the 80-column Video Display Controller (VDC) through its two registers at (you guessed it) $D600 and $D601 in Bank 15.

Meanwhile, the 40-column screen may need to know how to draw a character whose shape is defined starting at $D600 in Bank 14. Clearly, interfering in these rapidly changing configurations would be a very tricky and dangerous process. Yet, BASIC 7.0 on the C128 seems to supply a command that does exactly that. Naturally, it is called the BANK command.

Various references seem unclear about just what the BASIC 7.0 BANK command does. One states that the command "switches the system from one bank to another". Another says that BANK "selects one of the 16 memory banks". Most authorities, including Commodore's *C128 System Guide* and *C128 Programmer's Reference Guide* make it clear that the BANK command determines the memory configuration accessed by certain other BASIC 7.0 commands, but there is no general agreement as to which commands are involved. Some digging in the C128 ROMs gave me the answer to most of these questions, and revealed some facts about BANK that are both important and not widely known.

What it does

The BANK routine is located in ROM at $6BC9 (listed at the end of this article). As the disassembly shows, the routine is short and simple. It evaluates the argument of the BANK command, checks if that argument is in the range 0-15, then stores it in $03D5 (decimal 981), and exits. *And that is all it does!* The command, **BANK 15**, is exactly the same as **POKE 981,15**. The BANK command certainly seems innocent enough...

The next obvious question is "Who cares?" or, "What routines reference this memory location, $03D5?" I used the Monitor HUNT command to check the ROMs for all instances of this address. (I thought it unlikely that this location would be referenced by indexed or indirect addressing.)

Naturally, the address would be found, in low-byte hi-byte form, as the sequence $D5, $03. There are exactly nine occurrences of this byte sequence, and they all turned out to be part of valid load or store commands. This is the break-down of the locations and the routines located there:

$40B5	the initialization routine of the cold-start sequence. The routine stores a value of $0F, decimal 15, in location $03D5
$5891	this part of the ROM handles the SYS statement.
$6BD1	handles the BANK statement (see listing).
$6C41	handles the WAIT statement.
$7347	handles the BOOT statement.
$80D2	handles the PEEK statement.
$80F1	handles the POKE statement.

$A3E0 evaluates parameters for disk commands.

$AA60 common code for RAM Expansion Module STASH, FETCH and SWAP commands.

These are the only commands that change or refer to $03D5; these are the *only* BASIC commands that are affected by BANK. There are several significant points that should be made about this list.

The first thing to be noted is the relative permanence of the BANK command. Once BANK stores a value in $03D5, only another BANK command, a poke to $03D5, or a complete system reset will change it. The stored value survives the running of a program, a RUN/STOP-RESTORE, and even a reset with RUN/STOP depressed.

As a consequence, you should never assume you know the value stored in $03D5. Any of the 'banked' commands listed above should *always* include some type of BANK command to set the desired configuration explicitly.

Some C128 references state that, in the absence of any BANK command, Bank 15 is the default value. In one sense, this is correct. If no BANK command has ever been used since the computer was reset, the value of $0F stored in $03D5 by the initialization routine will establish Bank 15 as the one to be accessed.

However, if any BANK command has ever changed the value in $03D5 since the last reset, then that BANK command is the one that determines the bank accessed, even if it was isssued hours earlier.

PEEK, POKE, SYS and WAIT

The importance of BANK to these four commands is obvious. If you are going to look at, or change bytes at memory locations in different banks, the PEEK and POKE routines must check $03D5 to know which configuration you want them to access.

If you are going to SYS to some machine language, SYS needs to know which configuration contains the program. If you are going to WAIT until the bits in a particular memory location match some pattern, again the routine must know which configuration contains the particular location.

STASH, FETCH and SWAP

The inclusion of the RAM Expansion commands STASH, FETCH and SWAP may cause some surprise. The RAM Expansion Module User's Guide certainly seems pretty definite that only Bank 0 can be accessed. On page 14, it states that the BASIC commands "can only be used to transfer or retrieve data in Bank 0 of the C128 computer's internal RAM", and the statement is repeated word-for-word on page 24. The situation is somewhat confused, since the manual goes on immediately to describe how to access Bank 1! In fact, you can FETCH, STASH and SWAP to/from *any* C128 internal bank, simply by using the BANK command first.

The Version 0 C128 ROMs have some problems in doing this. The original DMA (Direct Memory Access) routine insists on creating a new memory configuration, in which the I/O block is visible. Thus, it would be impossible to STASH the Character ROMs, in Bank 14, using the Version 0 ROMs. In addition, the Version 0 routine will occasionally carry out the STASH/FETCH/SWAP with the wrong memory configuration enabled. The new Version 1 ROMs correct both these bugs, allowing you to access any part of any bank without difficulty.

For example, if you have either a 1700 or 1750 RAM Expansion Module and C128 Version 1 ROMs, try this short program in 40-column mode:

```
100 GRAPHIC 1,1
200 BANK 14: STASH 4096, 53248, 0, 0
300 BANK 15: FETCH 4096,  8192, 0, 0
400 BANK 0:  FETCH 4096, 12288, 0, 0
```

The entire character set has been copied from the Bank 14 Character ROMs into the RAM expansion, and from there twice into the Bank 15 hi-res screen memory. The different BANK commands in lines 300 and 400 simply demonstrate that, from $0000 to $3FFF, Bank 0 and Bank 15 are the same.

Disk I/O

The inclusion of the disk parameter evaluation routine is also curious. The commands involved are BLOAD and BSAVE. The description of these two commands makes it clear that you can specify the bank to be accessed by including the *B* parameter in the command string; for example,

```
BSAVE "CHARGEN", B14, P53248 TO P57343
```

will save the character pattern ROM, in Bank 14, to disk; while

```
BLOAD "SPRITES", B0, P3584
```

loads a binary file into the Bank 0 sprite pattern storage area.

Not so clearly stated is the fact that, in the absence of any *B* parameter in the BSAVE or BLOAD command string, the last BANK command is used to set the bank saved or loaded. Enter and run this short BASIC program, in 40-column mode, with a disk in the drive:

```
100 GRAPHIC 1,1
200 BANK 14 : BSAVE "CHAR/SET", P53248 to P57343
300 BANK 15 : BLOAD "CHAR/SET", P8192
```

The complete character set will appear on the screen, as it is first BSAVEd from Bank 14 to disk, and then BLOADed back into Bank 15 into the high-res screen memory.

Either a BANK statement or a *B* parameter is necessary to ensure that BSAVE, and particularly BLOAD, operate reliably. The problem is that the C128 uses the same format for saving files as do all other Commodore 6502-based systems, from the first PET onwards. Thus, while the start address of a file is saved, the Bank is not saved, since the format predates the Bank concept.

This makes the C128 compatible with other Commodore computers, but leads to problems when files are saved and loaded from different Banks. The Bank must be specified separately, by either the *B* parameter in the command string, or the BANK command preceding the disk command. Since the *B* parameter overrides the BANK command, it should be included in the command string whenever the BLOAD or BSAVE commands are used.

BOOT

The fact that BOOT is affected by the BANK command is a total surprise. There are, of course, two versions of the BOOT command. The simple command, **BOOT**, causes the system to carry out instructions according to the contents of Track 1, Sector 0. This version of BOOT is not influenced by BANK. However, the other version of the command, **BOOT "filename"**, *is* affected by BANK, although none of references I have seem to be aware of this.

The command, **BOOT "filename"**, is the equivalent of **BLOAD "filename"**, followed by a SYS to the load address of the file BLOADed. However, as explained above, the load Bank is not saved.

As a result, the BOOT command uses the BANK command flag in $03D5 to determine the Bank where the program will be BLOADed and run. Thus, you should always set this flag with a BANK command before you execute the **BOOT "filename"** command.

An interesting discovery was made about the syntax of the BOOT command. Most references fail to mention that the BOOT command string can contain an alternate load address, specified by a *P*, followed by the new load address.

This is exactly the same as the *P* syntax used in BLOAD and BSAVE. In addition, none of the references mention that the *B* parameter can be included in the BOOT command string to force the BLOAD and SYS into some other Bank. For example:

```
BOOT "GOODIES", B0, P12345
```

will load the file "GOODIES" into Bank 0, starting at 12345 (decimal), and then SYS to this location. The Bank value in $03D5, and the original load Bank and address of "GOODIES", will have no effect. Since this use of the *B* parameter is completely undocumented by Commodore, it would be unwise to make much use of it. There is no requirement on Commodore's part to preserve such undocumented 'features'.

USR

Notably absent from the list of BASIC 7.0 'memory' commands affected by BANK is the USR function. Briefly, USR operates as follows. When a BASIC program encounters a USR statement, such as:

```
400 Y = USR( X )
```

the expression in parentheses is evaluated and stored in Floating Point Accumulator #1. In the example above, the expression is just the variable *X*, but any complex expression that yields a numerical value is permitted. Then the program executes a JMP to a user-supplied machine language routine. This is only possible if you have earlier stored the address of the routine, in low-byte, high-byte order, in 4633-4634 (decimal), $1219-$121A.

The machine language routine may or may not change the value in Floating Point Accumulator #1; also, the routine must end in an RTS (ReTurn from Subroutine). Finally, the value found in Floating Point Accumulator #1 at the end of the machine language routine is used as the value of the USR function; in the above example, this value is assigned to *Y*. Here is a more complex example:

```
500 Y = 3 * ( SQR( 2.5 * USR( LOG( 5 * Y ) ) ) )
```

Here the variable *Y* is multiplied by 5, and the logarithm of the result is calculated and left in Floating Point Accumulator #1. The machine language routine is executed, and the value in the Floating Point Accumulator at the end of the ML is multiplied by 2.5; the square root of the result is multiplied by three and assigned to the variable *Y*.

If you are careless enough to use the USR function without setting the pointer in $1219-$121A, you will receive an ILLEGAL QUANTITY ERROR message. There is no illegal quantity, and USR has functioned as described. It's just that the initialization routine sets $1219-$121A to point to the routine that prints that particular error message, as a precaution against exactly this piece of carelessness!

As far as this article is concerned, the important point is that the JMP to the user-supplied machine language routine takes place in Bank 15. The BANK flag in $03D5 is not consulted, and USR is not affected in any way by the BANK command. Thus, the machine language routine must be in Bank 15 RAM below $4000, or consist of a ROM routine. Of course, there is no reason why the ML cannot jump to a routine in another Bank, as long as it returns to Bank 15 before ending.

Bank 16, 17, 18...?

As stated above, the BANK command is careful to place a number in the range 0-15 (decimal) in $03D5. You may be wondering what would happen if you poked some other value into $03D5, and then used any of the commands above. The results would be quite unusual, and not very useful. Here's why. The actual switch from one Bank to another is accomplished by storing a number in $FF00.

This is an alternate address for $D500, the MMU Configuration Register. Each of the eight bits in the number stored determines some part of the memory configuration, leading to a possible 256 configurations.

Commodore picked 16 individual configurations (or Banks) that it thought would be particularly useful, and stored the Configuration Register value that establishes each of these configurations in a table starting at $F7F0.

For example, to establish the memory configuration that Commodore chose to call Bank 0 requires that 63 (decimal) be stored in $FF00, so the first entry in the table is 63. The BANK flag in $03D5 is used as an offset into this table, to obtain a value that will then be stored in $FF00.

Since there are only 16 entries in the table, setting a Bank higher than 15 would cause the system to read the code that follows the table as more Configuration Register values. The memory configuration that would be established during PEEK, POKE or whatever, by these 'new' table values would be very strange indeed!

Summary

1) Always precede PEEK, POKE, SYS, WAIT, STASH, FETCH, SWAP and BOOT "filename" with a BANK statement to set the desired Bank explicitly.

2) Always include the *B* parameter in the command string for BLOAD and BSAVE to set the desired Bank explicitly.

3) Always locate the machine language for the USR function in Bank 15.

4) Never POKE strange values into $03D5. Better yet, never POKE strange values anywhere!

Listing

BANK Command ROM Listing

```
6bc9  jsr $87f4  ; routine to evaluate
                 ; BANK argument

6bcc  cpx #$10   ; check for valid argument
6bce  bcs $6bd4  ; branch if invalid

6bd0  stx $03d5  ; store BANK argument
                 ; in $03D5

6bd3  rts        ; all done

6bd4  jmp $7d28  ; prints error message
```

REDATE

Notes from the CP/M Plus workbench

by Adam Herst
Copyright © 1988 Adam Herst

CP/M Plus provides sophisticated date and time services to both the user and the programmer. Users can set the system clock from the command line using the DATE or CONF commands. With a little preparation, you can stamp files with the system time to reflect creation, access, and update dates and times. The transient version of the DIR command can display the stamped information. Programmers can manipulate system dates and times and file stamps using a number of BDOS services.

With all of these services provided by CP/M Plus, it is unfortunate that the C128 does not come equipped with a battery-powered clock. If the system clock and file stamps are to be correct, the system date and time must be set or reset on every cold boot, reset or warm boot.

While this is annoying, there are benefits in making sure that the system's date and time are set correctly, or at least in correct chronological sequence. If you're like me - without a watch more often than not - it is convenient to have the time available through the command line. More importantly, the date and time services provide a way to track the many versions of text and executable files that are generated by practically every large writing or programming project.

REDATE, a short assembler program, removes the drudgery of manually resetting the system clock. It uses CP/M's file stamping to set the system date and time to that of the most recently accessed file. Without a battery-powered clock, no program can automatically set the date and time exactly. However, REDATE can ensure that file stamps are chronologically correct and, if there has been recent disk access, that the system date and time are reasonably close to the real date and time.

Setting the date and time

Two utilities with which to set the system date and time, DATE and CONF, are supplied in the CP/M Plus toolkit. DATE is a standard CP/M Plus transient utility provided by DRI (Digital Research Institute, the supplier of the CP/M Plus operating system). It is a relatively large program, and is specialized for setting and displaying the system date and time. CONF is an implementation-specific utility provided by Commodore, with CP/M versions dated Dec 6, 1985 and later. CONF is small and fast, and is designed to manipulate a host of system characteristics, most of them specific to CP/M on the C128. The differences in design and function between DATE and CONF are reflected in the operation of these two utilities.

When used to set the system date and time, DATE can be used in a command line mode or in an interactive mode. Interactive mode is useful in PROFILE.SUB files, since it pauses and prompts for input.

In interactive mode, the form of the DATE command is:

```
DATE SET
```

CP/M will respond with the exchange:

```
Enter today's date (MM/DD/YY):
Enter the time (HH:MM:SS):
Press any key to set time
```

An argument error at any stage will abort the DATE command. An argument can be passed over by pressing the RETURN key.

In command line mode, the form of the DATE command is:

```
DATE SET dd/mm/yy hh:mm:ss
```

where *dd* is the day number, *mm* is the month number, *yy* is the year number, *hh* is the number of hours in 24-hour format, *mm* is the number of minutes, and *ss* is the number of seconds. Both arguments must be supplied in full. An incomplete date or time specification is flagged as an error.

Issuing this command without argument errors results in a prompt to press any key to set the date and time. Pressing a key sets the system clock and returns the CCP prompt. If there are errors in the arguments, either syntax errors or invalid dates or times, the error is flagged and the operation is aborted.

DATE can be used to display the system date and time in command line mode only. The form of the command for display is:

```
DATE
```

CP/M will respond with a display similar to:

```
Mon 08/01/88 11:49:18
```

Note the display of the day name. The code to extract this information from the information actually maintained by CP/M is one of the reasons for DATE's large size and slow operation relative to CONF. Nonetheless, it is a nice feature if you need it.

CONF offers limited functionality compared to DATE. It operates in command line mode only, performs less error handling, and provides a stripped-down display. However, given its relatively small size, and its many other uses, it is much more likely to be found on a currently logged-in disk than DATE, its DRI counterpart. To use CONF to set the date and time, issue the command:

```
CONF DATE = dd/mm/yy hh:mm:ss
```

where *dd* is the day number, *mm* is the month number, *yy* is the year number, *hh* is the hour number in 24-hour format, *mm* is the number of minutes, and *ss* is the number of seconds (though this is ignored and may be omitted). Either argument can be omitted. If both arguments are omitted the date is displayed. An error - either a syntax error or an invalid date - causes the command to be aborted.

Stamping files

CP/M's file-stamping services are the heart of REDATE's operation. Without them, no record of the date and time would exist for REDATE to use. However, CP/M Plus does not stamp files with dates and times by default. (This is probably due to the directory entry overhead imposed by file stamps. As described later in this article, the use of file stamps reduces the number of available directory entries by 25 per cent.) So, before files can be stamped, the INITDIR command must be used to initialize the directory of the given disk to receive file stamps. Also, the SET command must be used to indicate which of the file stamp types is to be active.

To initialize a disk directory for file stamps, issue the command:

```
INITDIR d:
```

where *d* is the letter of the drive containing the disk to be initialized.

INITDIR responds with the exchange:

```
INITDIR WILL ACTIVATE TIME STAMPS FOR THE SPECIFIED DRIVE.
Do you want to re-format the directory on drive: M (Y/N)?
```

If the disk has already been initialized to accept file stamps, INITDIR responds with:

```
Directory already re-formatted.
Do you want to recover time/date directory space  (Y/N)?
```

If the directory space is not to be recovered, INITDIR responds with:

```
Do you want the existing time stamps cleared  (Y/N)?
```

This last exchange is the only way to directly manipulate file stamps through the standard CP/M toolkit. Unfortunately, file stamps can only be explicitly set to a blank entry.

Note that the disk does not have to be newly formatted. Existing data will not be destroyed by INITDIR. There is a chance, however, that a disk with data may not have sufficient directory space remaining to support file stamps. If this is the case, you will have to remove some of the files on the disk. Files that existed before the initialization will have blank entries for the activated stamps.

Once the directory has been initialized, use the SET command to indicate which of the file stamp types should be active for that disk. CP/M Plus supports three types of file stamps: *create*, *update*, and *access*. *Create* stamps indicate the date and time at which the file was created. *Update* stamps indicate the date and time at which the file was last updated. *Access* stamps indicate the date and time at which the file was last accessed.

While three file stamp types are supported, a maximum of two file stamp types may be active at any one time. CP/M dictates that create and access file stamps are mutually exclusive - only one of the two can be active at any one time. Fortunately, the way CP/M interprets update stamps allows them to function as create stamps in most cases.

Update stamps indicate the date and time at which the file was updated 'in place'. A file that is updated in place has altered information written to the same disk record as the original file, and writes new information to the last record of the original file. One program that updates files in place is dBASE II.

Most programs do not update files in place. They create a new file to hold the altered or new version and delete or rename the original file. Consequently, for a newly created file, the update stamp reflects the creation date and time. Activate access stamps instead of create stamps, and interpret them as create stamps. To display the file stamps, use the transient version of DIR:

```
DIR d: [ATT]
```

where *d* is the drive whose disk directory should be shown. A directory display similar to the following will be shown:

```
Scanning Directory...
Directory For Drive A:  User  9

    Name     Bytes  Recs  Attributes  Prot      Update         Access
------------ ------ ------ ----------- ------ -------------- --------------
REDATE   COM    2k      3 Dir RW      None   08/01/88 12:11 08/01/88 12:11

    Total Bytes     =      2k  Total Records =        3  Files Found =    1
    Total 1k Blocks =       1  Used/Max Dir Entries For Drive A:    69/ 128
```

The last two columns of the listing contain the information for the active file stamps, in this example update and access. Practically all of the other forms of the DIR command will display the file stamp information as well.

System level services

The next few paragraphs discuss time and date services and file stamping at the system level; they assume familiarity with the CP/M 3.0 BDOS and the CP/M 3.0 file system. This background information can be found in the *CP/M Plus Programmer's Guide* available through the Commodore CP/M Special Offer.

CP/M uses a four-byte data structure to store date and time information. The first two bytes are used to store the date; the last two bytes are used to store the time. The date is stored as the number of days elapsed since January 1, 1978, in low byte/high byte format. (Your guess as to what will happen when we pass June 4, 2001, the largest date representable under this format, is as good as mine.) The time is stored as the number of hours and number of minutes, in BCD (binary coded decimal) format. The CP/M date structure representing the date and time **7/18/88 22:55** looks like this:

```
0C              0F              22      55
low byte        high byte       hours   minutes
date in days    date in days
```

The system date and time are maintained in the system control block, at byte offset 58h-5ch. (The fifth byte is used to store the seconds in BCD, and is unused for file stamps.) It can be queried and set by directly manipulating the SCB. However, BDOS calls 68h and 69h are provided to facilitate operations that set or query the date and time respectively.

CP/M Plus stores file stamps in the disk directory. Since only two types of file stamps can be active at one time, and four bytes are required for each date structure, a maximum of 8 bytes are required to store the file stamps for a given file. File stamps are not stored in the same directory entry as the file to which they are related - there is no room. They are stored in a directory entry used solely for date and time stamps, and password mode information. There is enough room in a directory entry (32 bytes) to store date and time stamps and password information for three files.

This explains what is happening when a disk directory is processed by INITDIR. When CP/M Plus prepares a directory for file stamping, the directory is rearranged so that every fourth entry is used to record stamp and password mode information for the previous three files. (This results in the 25 per cent reduction of available directory space mentioned earlier.) A file stamp directory entry is identified by a 21h in location 0 of the directory entry, instead of the user number to which the file belongs.

When a directory entry for a file is read using the BDOS 'search for first file' or 'search for next file' system calls, the DMA buffer contains the directory entry for four files. When file stamps are active, the last directory entry in the DMA buffer contains the file stamp and password mode information for the preceding three files.

File stamp information can be obtained directly from the DMA buffer. However, BDOS call 66h gets the file stamp information for the file in the FCB. Bytes 24-27 of the FCB will contain the create or access file stamp (recall that only one of the two may be active). Bytes 28-31 of the FCB will contain the update stamp. File stamp date and times cannot be set directly.

Redating

REDATE sets the system date and time to that of the most recently accessed file as indicated by the access stamp. It frees you from finding a calendar and clock to determine the date and time, and frees you from having to enter that date and time through the keyboard. While REDATE can't accurately set the time, it ensures that stamps are chronologically correct. It is most effective when used immediately after a warm boot or reset. While it can be used after a cold boot, large discrepancies between the system date and time and the real date and time are likely.

REDATE requires that access file stamps be activated on the specified disk and that some disk activity has occurred before the REDATE command is issued.

To execute REDATE, issue the command:

```
REDATE d:
```

where *d* is the drive in which the disk to be searched is located.

If the disk letter is omitted, the default disk will be searched. If no file access stamp is found, the program return code will be set to an error condition.

The REDATE program

REDATE is written in 8080 assembler. It can be assembled 'as is' with MAC and loaded with HEXCOM, both supplied by DRI in the Commodore CP/M Special Offer package.

The code is fully commented, so only an overview will be supplied here. REDATE starts by matching all files in user area 0, and stores them in a simple stack using the *PushFileName* routine. Once all matches have been found, the file names are retrieved one at a time using the *PopFileName* routine. The access stamp information for each file is compared to the saved date (initialized to **1/1/78**) using the *CompareDate* routine. If the access stamp is more recent, it is copied to the saved date where it becomes the standard for further comparisons. When the last file in the current user area is processed, the cycle is repeated for the next user area. When all user areas have been processed, the system date is set to the saved date if it is more recent than the initial date.

Conclusion

REDATE illustrates one use of CP/M 3.0's sophisticated date and time services. Enhancements to REDATE could include an option for a 'fudge factor' to set the date more accurately, or an option to search all the disks in the drive path for the most recently accessed file. Other date and time related utilities could include a MAKE utility to evaluate file dependencies. The features of CP/M 3.0 make ideas like these surprisingly easy to implement.

Listing: Redate.asm

```
Redate

; 1  TITLE

; REDATE (c) 1988 Adam Herst, Toronto, Ontario
;
;   Set the system date to that of the most recently accessed file
;   on the specified disk.
;   Requires that access file stamping has been activated.

; 2  HISTORY

;   v2.1        Adds drive option
;               Sets program return code to error if no stamp found
;   v2.0        First working version
;   v1.0        Non-working prototype

; 3  EQUATES

GetSetRetCode   equ     6ch
SetDMAAddr      equ     1ah
SelDisk         equ     0eh
GetSetUser      equ     20h
ParseFileName   equ     98h
SearchFirst     equ     11h
SearchNext      equ     12h
GetDatePasswd   equ     66h
SetDate         equ     68h

BDOS:           equ     5h
CPMFCB:         equ     5ch
FCBFILENAME:    equ     CPMFCB+1d
FCBACCESS:      equ     CPMFCB+24d
MYDMA:          equ     0400h
FILENAMESTACK:  equ     0500h
DMARECOFFSET    equ     20h
RETCODECCPSUC:  equ     0000h           ; CCP-initialized success code
RETCODEUSRERR:  equ     0FF00h          ; User set error code

; 4  PROLOG

; 4.1  Program start

                org     100h

; 4.2  Set program return code to error

                mvi     c,GetSetRetCode
                lxi     d,RETCODEUSRERR
                call    BDOS

; 4.3  Set dma buffer

                mvi     c,SetDMAAddr
                lxi     d,MYDMA
                call    BDOS

; 5  Find most recent access stamp and save it

; 5.1  Has a drive been specified?

                lxi     h,CPMFCB        ; point to drive letter
                mov     a,m             ; get drive letter
                cpi     0h              ; is it already the default?

; 5.2  If no then start checking files

                jz      CHECKFILES      ; filename is already the default

; 5.3  Set default drive to specified drive

                dcr     a
                mvi     c,SelDisk
                mov     e,a
                call    BDOS

; 5.4  For USERNUM: = 0 to 15

CHECKFILES:
                lxi     h,USERNUM       ; point to counter
                mvi     m,0h            ; set it to 0
FORUSERNUM:
                mov     a,m             ; get counter
                cpi     0fh             ; is it equal to 15
                jnc     DODATE          ; yes so jump to set system date

; 5.4.1  Set user number to USERNUM:

                mvi     c,GetSetUser
                mov     e,m
                call    BDOS

; 5.4.2  Set fcb to match all wildcard

                lxi     h,ALLFILES      ; point to wildcard filespec string
                shld    PFCBFSPECPTR    ; put pointer in PFCBSTRUCT
                lxi     h,CPMFCB        ; point to file control block
                shld    PFCBFCBPTR      ; put pointer in PFCBSTRUCT
                mvi     c,ParseFileName ; parse the string and initialize FCB
                lxi     d,PFCBSTRUCT
                call    BDOS

; 5.4.3  Setup the filename stack to store filespec matches for processing

                lxi     h,FILENAMESTACK ; point to bottom of filename stack
                shld    FILENAMEPTR     ; set top stack pointer to bottom

; 5.4.4  Get directory entry for first file match

                mvi     c,SearchFirst
                lxi     d,CPMFCB
                call    BDOS

; 5.4.5  While there is a file match

WHILEAMATCH:
                cpi     0ffh            ; is it the no match code?
                jz      WHILENOTEMPTY   ; yes so jump to process matches

; 5.4.5.1  Push filename onto filename stack

                call PushFileName       ; save filename from DMA buffer
```

```
; 5.4.5.2  Get directory entry for next file match

                mvi         c,SearchNext
                lxi         d,CPMFCB
                call        BDOS

; 5.4.5.3  Check if there was a match to save

                jmp         WHILEAMATCH

; 5.4.6  While filename stack is not empty

WHILENOTEMPTY
                lxi         h,FILENAMESTACK ; point to bottom of stack
                push        h               ; put pointer in DE
                pop         d
                lhld        FILENAMEPTR     ; point to top of stack
                call        CompareDEToHL   ; do they point to the same location
                jz          NEXTUSERNUM     ; yes so no files to process

; 5.4.6.1  Pop filename from FILENAMESTACK

                call        PopFileName     ; put filename in FCB

; 5.4.6.2  Get access stamp information for file in CPMFCB

                mvi         c,GetDatePasswd
                lxi         d,CPMFCB
                call        BDOS

; 5.4.6.3  Is the file access date newer than the current saved date?

                lxi         d,NEWSYSDATE   ; point to saved date
                lxi         h,FCBACCESS    ; point to access date
                call        CompareDate    ; compare saved date to access date

; 5.4.6.4  If no then process next filename

                jnc         WHILENOTEMPTY  ; saved is larger so do next file

; 5.4.6.5  Save access date of filename

                lxi         h,NEWSYSDATE
                lxi         d,FCBACCESS
                mvi         b,04h
                call        CopyBytesUp

; 5.4.6.6  Check if there is another filename to process

                jmp         WHILENOTEMPTY

; 5.4.7  Do next user number

NEXTUSERNUM:    lxi         h,USERNUM      ; point to user number counter
                inr         m              ; increment it
                jmp         FORUSERNUM     ; check if its valid

; 6  Set the system date if an access stamp has been found

DODATE:

; 6.1  Is the saved date equal to its initial value?

                lxi         h,NEWSYSDATE
                mov         a,m
                inx         h
                ora         m
                inx         h
                ora         m
                inx         h
                ora         m

; 6.2  If yes then do an unsuccessful exit

                jz          EXITERROR

; 6.3  Set the system date

                mvi         c,SetDate
                lxi         d,NEWSYSDATE
                call        BDOS

; 7  EXIT

; 7.1  Success

EXITSUCCESS:
                mvi         c,GetSetRetCode
                lxi         d,RETCODECCPSUC
                call        BDOS

; 7.2  Error

EXITERROR:
                jmp         00h

; 8  SUBROUTINES

; 8.1  CurrentDmaRec - Point to the current DMA record

; Description:  Point to the start of
;               the current record in the DMA buffer.
; Arguments:    A - number of current record in DMA buffer (0-3)
; Returns:      HL - points to start of current record

CurrentDmaRec
                lxi         h,MYDMA        ; point to first DMA record
                lxi         d,DMARECOFFSET ; get the record offset
                inr         a              ; initialize record counter
NEXTDMAREC:
                dcr         a              ; is it the right record?
                rz                         ; yes so return
                dad         d              ; point to next DMA record
                jmp         NEXTDMAREC     ; check if it is the right record

; 8.2  CompareDate - Compare HL date to DE date

; Description   Compare the CP/M date structures.
;               The standard date is smaller/equal/larger
;               than the argument date
; Arguments:    DE - standard date
;               HL - argument date
; Returns:      Z  - set if equal
;               C  - set if std is smaller

CompareDate
                inx         h              ; high byte of years in days of argument
                inx         d              ; high byte of years in days of standard
                ldax        d              ; get standard
                cmp         m              ; is it equal to argument
                rc                         ; no it is smaller
                rnz                        ; no it is larger
                dcx         h              ; low byte of years in days of argument
                dcx         d              ; low byte of years in days of standard
                ldax        d              ; get standard
                cmp         m              ; is it equal to argument
                rc                         ; no it is smaller
                rnz                        ; no is is larger
                inx         h              ; hours byte of argument
                inx         h
                inx         d              ; hours byte of standard
                inx         d
                ldax        d              ; get standard
                cmp         m              ; is it equal to argument
                rc                         ; no it is smaller
                rnz                        ; no it is larger
                inx         h              ; minutes byte of argument
                inx         d              ; minutes byte of standard
                ldax        d              ; get standard
                cmp         m              ; is it equal to argument
                ret
```

```
; 8.3  PushFileName - Push current filename in DMA buffer onto FILENAMESTACK

; Description:  Copy current filename in DMA buffer
;               to the top of the filename stack.
;               Requires that FILENAMESTACK has been set up
;               and FILENAMEPTR has been defined.
; Arguments:    A - current record in DMA buffer

PushFileName
                call    CurrentDmaRec  ; point to current record in DMA
                inx     h              ; point to start of filename
                push    h              ; put source pointer in DE
                pop     d
                lhld    FILENAMEPTR    ; put destination pointer in HL
                mvi     b,11d          ; put number of bytes to copy
                call    CopyBytesUp    ; copy them incrementing pointer
                shld    FILENAMEPTR    ; save the new pointer
                ret

; 8.4  PopFileName - Push filename in FCB onto FILENAMESTACK

; Description:  Copy the filename from the top of the filename stack
;               to the FCB.
;               Requires that FILENAMESTACK: has at least one entry.

PopFileName
                lhld    FILENAMEPTR        ; get source pointer
                push    h                  ; put it in DE
                pop     d
                lxi     h,FCBFILENAME+11d  ; get destination pointer
                mvi     b,11d              ; number of bytes to copy
                call    CopyBytesDown      ; copy them decrementing pointer
                push    d                  ; save new pointer
                pop     h
                shld    FILENAMEPTR
                ret

; 8.5  CompareDEToHL - Compare the word in DE to the word in HL

; Description:  Compare HL to DE.
;               HL is smaller/equal/larger than DE.
;               Set appropriate flags on return.
; Arguments:    HL - word in low byte, high byte format
;               DE - word in low byte, high byte format
; Returns:      Z - set if equal
;               C - set if HL is smaller

CompareDEToHL
                mov     a,l            ; get high byte
                cmp     e              ; is hl high byte equal to de high byte?
                rc                     ; no, it is smaller, so HL is smaller
                rnz                    ; no it is larger, so HL is larger
                mov     a,h            ; get low byte
                cmp     d              ; is hl low byte equal to de low byte?
                ret

; 8.6  CopyBytesUp - Copy B number of bytes moving up from DE to HL

; Description:  Copy the bytes pointed to by DE to the bytes pointed to
;               by HL incrementing the pointers.
; Arguments:    DE - start of source bytes
;               HL - start of destination bytes
;               B  - number of bytes to copy
; Returns:      DE - byte after last byte of source string
;               HL - byte after last byte of destination string

CopyBytesUp
                mvi     c,0h           ; initialize byte counter
NEXTBYTE2:
                mov     a,c            ; get counter for comparison
                cmp     b              ; is it equal to the number of bytes?
                rz                     ; yes so finished
                inr     c              ; increment counter
                ldax    d              ; get source bytes
                mov     m,a            ; put it in destination
                inx     d              ; point to next source bytes
                inx     h              ; point to next destination
                jmp     NEXTBYTE2      ; check if more bytes to copy

; 8.7  CopyBytesDown - Copy B number of bytes moving down from DE to HL

; Description:  Copy the bytes pointed to by DE to
;               the bytes pointed to by HL
;               decrementing the pointers.
; Arguments:    DE - start of source bytes+1
;               HL - start of destination bytes+1
;               B  - number of bytes to copy
; Returns:      DE - last byte of source bytes
;               HL - last byte of destination bytes

CopyBytesDown
                mvi     c,0h           ; initialize byte counter
NEXTBYTE3:
                mov     a,c            ; get counter for comparison
                cmp     b              ; is it equal to the number of bytes
                rz                     ; yes so no more bytes to copy
                inr     c              ; increment the counter
                dcx     d              ; point to source byte to copy
                dcx     h              ; point to destination
                ldax    d              ; get source byte
                mov     m,a            ; put it in destination
                jmp     NEXTBYTE3      ; check if more bytes to copy

; 9  STRUCTURES

; 9.1  VERSION: - Version and copyright string
; Description:  Version and release number and copyright string

VERSION:        db      'REDATE v2.1 (c) Adam Herst 1988'

; 9.2  NEWSYSDATE: - Date to set system time to
; Description:    date and time in CP/M format

NEWSYSDATE:
                db      0h             ; low byte of years in days
                db      0h             ; high byte of years in days
                db      0h             ; hours in bcd
                db      0h             ; minutes in bcd

; 9.3  PFCBSTRUCT: - Parse FCB structure
; Description:  Parse file control block pointer structure

PFCBSTRUCT:
PFCBFSPECPTR:   dw      0000h          ; pointer to cp/m style filespec string
PFCBFCBPTR:     dw      0000h          ; pointer to file control block

; 9.4  USERNUM: - user number counter
; Description:  User number counter

USERNUM:        db      0h

; 9.5  ALLFILES: - filespec string m:*.*
; Description:  CP/M style string for wildcard filespec

ALLFILES:       db      '*.*$'

; 9.6  FILENAMEPTR: - pointer to top of FILENAMESTACK
; Description:  Pointer to top of FILENAMESTACK

FILENAMEPTR:    dw      0500h

; 10  END

                end
```

Serial I/O in Power C

An RS232 function package for C programmers

by W. Mat Waites

One of the most exciting areas of use for the C64 is cheap telecommunications. The ability to communicate with other machines via modem or a hardwired connection adds greatly to the power and value of any computer. The C64 has benefited more than other computers from telecommunications because of Commodore's supplying modems that are very affordable.

Developing telecommunications programs is interesting and fun, but the choices of language for that development have been very few in the past.

Interpreted BASIC is too slow even for 300 baud communication. Compiled BASIC is much faster, but generally utilizes the BASIC interpreter's garbage collection routines for string storage maintenance. The result of this is that the system locks up every few minutes while the string space is being recovered. This will drive you insane after a while.

Assembly language has been the most viable choice for writing programs that would be limited in speed only by the Kernal and the hardware. These assembly programs are very long and difficult to modify, however. The lack of a standardized parameter passing convention and a linker makes it difficult to write functions in assembler that are reusable and sharable with other software developers.

C has come to the rescue with a language that is higher level than assembly, but without the run-time overhead of BASIC and other interpreted languages.

Commodore 64 serial I/O

The C64 actually has a very sophisticated serial I/O system for a microcomputer. It is interrupt driven, which means that incoming characters are taken in by the Kernal even if your program is not quite ready for them. Even such popular systems as MS-DOS do not have this feature. MS-DOS terminal programs must supply their own interrupt-driven code to create this kind of functionality. The Kernal takes care of all of the low level details of accepting characters and sending them out.

Most computers have specialized chips to perform the act of sending out and receiving individual characters. The generic name for this kind of device is Universal Asynchronous Receiver/Transmitter (UART). The C64 emulates the activities of a UART in software. The positive side of this is that software is more flexible than hardware. The C64's serial I/O is at least as configurable as a UART and is more configurable than some. The negative side is that software, especially 1 MHz 6502 software, is slow. The C64 Kernal routines are barely able to keep up at 1200 baud; 2400 baud is not reliable at all.

The two big kludges

There is a problem with the Kernal-supplied serial routines. The timing values supplied for 1200 baud are not exactly correct! The 'width' of the bits coming in to the port don't match up with what the Kernal expects. By supplying new and improved timing values, we can tune the routines to expect the correct bit widths.

This 'bit width fix' introduces another problem, though. With the best possible values in place for receiving characters, there is a problem with transmitting characters. The 'stop bit' (the final bit in a serial character transmission) is too 'narrow'. That is, it doesn't last long enough for the machine at the other end to recognize it reliably. This is not really a problem with simple terminal emulation because no one types fast enough to cause multiple characters to be output one immediately after another.

The short stop bit does cause problems with file transfers. When a block of data is sent, the characters follow one another in rapid succession. The receiver is sometimes still waiting for the end of the stop bit when the next character arrives. This kind of synchronization problem is called a framing error.

To break down this final barrier to reliable 1200-baud communication, a second kludge is introduced. A delay loop is used to wait for each character to be clocked out before another is added to the output buffer. This allows the receiving computer to recover from one character before the next arrives.

Problems with C

In applying C to telecommunications, the first hurdle to overcome is interfacing to the Kernal for several functions. Most obviously, the serial I/O must be accessed from C. Methods for doing this are not documented in either the Abacus Super-C or Pro-Line/Spinnaker Power C manuals (at least none that I have seen). Other functions that must be implemented include: getting a keystroke without 'hanging', producing a cursor on the screen, doing cursor movement, and providing timing functions for communications protocols.

This article introduces a terminal program written in C and provides the details of the implementation of serial I/O in Power C. *(Note: due to space limitations, only the serial and Xmodem routines themselves are included in the C source listings accompanying this article. The full source for Mat's terminal program, and the program itself, will be included on the **Transactor** disk for this issue. -Ed.)*

'Packages' of functions

Power C provides an excellent linking facility that allows the programmer to divide his application into as many compilation units as desired. Software development can be made more efficient by writing subsystems that are independent of each other and placing them in separate files. In this way, several different applications may call the same 'package' of functions.

The reusablility of software is very important if you ever intend to develop a software system of any size. You simply cannot start from scratch at the beginning of every project and expect to do large projects.

The package discussed here contains all of the functions and data structures necessary for serial I/O in Power C. It allows you to open the serial device, set the port parameters, write to the serial device, read from the device, and close the device.

The data structures include the input and output queues and the current state of the port.

Opening and Closing the Device

Openserial() opens the C64 'file' for serial communications. Notice that the BASIC-style open() call is used so that a secondary address may be supplied if desired. RS is the symbol for the stream number used for the serial port, 6. With the BASIC-style open, the stream numbers 5 through 9 should be used to avoid conflict with the automatic stream-number allocation done by the higher-level I/O functions.

The closeserial() function simply does a BASIC-style close on the RS stream.

Moving the buffers

The other activity carried out by the openserial() function is moving the buffer pointers to point to the buffers that have been declared for the serial port queues. These are named *inbuf* and *outbuf*. The Kernal allocates the buffers initially at the top of BASIC memory space, but this falls in the middle of Power C space. We simply move the pointers to point to space that we have allocated for this purpose; the rest of memory can then be used without fear of overwriting the input and output queues. This gives the added benefit of allowing the easy examination of the queues without reading characters from them.

Setting the port parameters

After the port is opened, the baud rate and other parameters must be set. We could have specified the baud rate at open time, but we want to be able to change the baud rate at any time so we must work at a lower level.

The setserial() function allows the caller to set the baud rate, the number of bits per character, the number of stop bits, and the parity. This function may be called at any time to change the parameters. The three baud rates implemented are 300, 450, and 1200. Many 300 baud modems will function reliably at 450 baud, and many BBSs support this speed.

Kludge #1 is included in the timing values supplied in this function. The 1200 baud values seem to work well, but they may be tuned for the best performance with your set-up. The 61 may be varied up or down by about 4 or 5.

The stopbits may be set to either 1 or 2. The bits per character may be set to anything from 5 to 8. The parity is set with a bitmap value corresponding to the 3 bits described in the *Commodore 64 Programmers Manual* for selecting parity.

Table of parity values

0 - disabled
1 - odd parity
3 - even parity
5 - mark parity
7 - space parity

Reading and writing

The getserial() function is called to get a character from the serial port. If no characters are available, a -1 is returned. Notice that if the Power C getc() function were called here, the function would not return until a character had come in the serial port. If you are writing a terminal program or BBS, you do not want to 'hang' waiting on characters. You simply want to get it if it is there, or return if it is not.

The putserial() function is called to output a character to the serial port. This function implements kludge #2. There is a delay loop that was shortened until framing errors began to occur. After the loop it simply calls the Power C putc() function to output the character.

Other functions in the package

Functions are also provided in this package for some other DOS-related activities. Functions are provided for accessing the keyboard without hanging, for checking to see if the 'logo' key is pressed (I use this for an attention key), to wait for a given number of seconds, and to read the disk error channel.

Notice that Kernal calls must be made to achieve some of this functionality, but with these functions making the calls for you, you don't need to directly call the Kernal in applications.

Using the package

To use this kind of a package you simply compile it as you would any other function in Power C.

This will produce a file - "dos.o". When you link your application, simply link in "dos.o" and you have serial I/O. Note that you will only compile it once and then link it in whenever you need it. This is a great advantage over BASIC compilers that force you to recompile your entire program every time you make a change.

The terminal program

Included on the disk is a simple terminal program that calls this serial I/O package. It implements a sprite cursor and Xmodem file transfers.

The Xmodem routines are very portable. The Commodore specific I/O functions are separated out and should make it very easy to move the Xmodem part to another operating system.

Ideas for future development

With the serial I/O stabilized, it shouldn't be too difficult to add other protocols: Xmodem CRC, Xmodem batch, Kermit, Punter, and so on. The most difficult thing about implementing some of these protocols is finding definitive documentation. Implementing a BBS is also a possibility.

The Commodore 64 still has a lot of life left in the area of software development. Hopefully, this article will help spur interest in C programming on the 64. Drop me a note if you have any questions, or if you write any interesting applications with the serial package.

I can be contacted on Usenet (!gatech!emcard!mat) or by mail at this address:

W. Mat Waites
1264 Brandl Drive
Marietta, Georgia 30060

```
/* dos.c - operating system stuff:
  disk support
  serial i/o
  kb i/o
  timers */
/* W Mat Waites - Sept 1988 */

#include <stdio.h>

/* 5-9 may be used with "basic" open */
#define KB 5
#define RS 6

/* kludge for reliable 1200 baud */
#define KLUDGE 40

/* kernel routines */
#define CHKIN  0xffc6
#define GETIN  0xffe4
#define TKSA   0xff96
#define ACPTR  0xffa5
#define TALK   0xffb4
#define UNTLK  0xffab

/* input and output serial buffers */
static char inbuf[256], otbuf[256];

/* serial interface functions */

/* openserial() - open serial port */
openserial()
{
  short *ribuf = 0x00f7;
  short *robuf = 0x00f9;

  /* open serial port */
  open(RS, 2, 0, "");

  /* move pointers to buffers */
  *ribuf = inbuf;
  *robuf = otbuf;
}

/* closeserial() - close serial port */
closeserial()
{
  close(RS);
}

/* 300, 450, 1200 implemented */

static short hibyte[3] = { 6,  4,  1};
static short lobyte[3] = {68, 12, 61};

/* setserial() - set serial port */
setserial(bd, bpc, sb, par)
int bd, bpc, sb, par;
{
  short *m51ajb = 0x0295;
  short *baudof = 0x0299;
  char  *m51ctr = 0x0293;
  char  *m51cdr = 0x0294;
  char  *bitnum = 0x0298;
  unsigned indx;

  switch(bd)
    {
    case 300:
      indx = 0;
      break;
    case 450:
      indx = 1;
      break;
    case 1200:
      indx = 2;
      break;
    default:  /* default to 300 baud */
      indx = 0;
      break;
    }

  /* set baud rate */
  *m51ajb = 256 * hibyte[indx] +
    lobyte[indx];
  *baudof = (*m51ajb)*2 + 200;

  /* stopbits */
  if(sb < 1 _ sb > 2)
    {
    sb = 1;
    }
  sb--;

  /* bits per char */
  if(bpc < 5 _ bpc > 8)
    {
    bpc = 8;
    }
  *bitnum = (char)(bpc + 1);
  bpc = 8 - bpc;

  /* parity */
  if(par < 0 _ par > 7)
    {
    par = 0;
    }

  /* put bpc, sb, and par in regs */
  *m51ctr = (char)((bpc << 5)
      _(sb << 7));
  *m51cdr = (char)(par << 5);
}

/* getserial() - char from serial port */
getserial()
{
  int ch;
  char *rsstat = 0x0297;

  ch = getonechar(RS);
```

```
  /* check for empty buffer */
  if((*rsstat & 0x08) == 0x08)
    {
    return -1;
    }
  else
    {
    return ch;
    }
}

/* putserial() - char to serial port */
putserial(ch)
char ch;
{
  int i;

  putc(ch, RS);

  /* delay loop for 1200 baud kludge */
  for(i=0; i<KLUDGE; i++)
    {
    }
}


/* keyboard interface functions */

/* openkb() - open keyboard */
openkb()
{
  char *rptflg = 0x028a;

  open(KB, 0, 0, "");
  /* let the keyboard repeat */
  *rptflg = 0x80;
}

/* closekb() - close keyboard */
closekb()
{
  close(KB);
}

/* getkb() - get char from keyboard */
getkb()
{
  return getonechar(KB);
}

/* charsinq() - # available kb chars */
charsinq()
{
  char *ndx = 0x00c6;

  return (int)*ndx;
}

/* chkstop() - check for <C=> key */
chkstop()
{
  char *shflag = 0x028d;

  return(*shflag == 0x02);
}

/* getonechar() - get char from chan */
static getonechar(channel)
int channel;
{
  char ac, xc, yc;

  xc = (char)channel;
  sys(CHKIN, &ac, &xc, &yc);
  sys(GETIN, &ac, &xc, &yc);
  return(int)ac;
}


/* disk i/o functions */

#define SADDR  0x6f

/* diskerr() - read error channel */
char *diskerr(disk)
int disk;
{
  int cc;
  char ac, xc, yc;
  static char msgbuf[41];
  char *mp;
  char *second = 0x00b9;
  char *status = 0x0090;

  /* tell drive to talk */
  ac = (char)disk;
  sys(TALK, &ac, &xc, &yc);

  /* tell it what to talk about */
  ac = (char)SADDR;
  *second = SADDR;
  sys(TKSA, &ac, &xc, &yc);

  /* read in the response */
  mp = msgbuf;
  for(;;)
    {
    /* get byte from bus in acc */
    sys(ACPTR, &ac, &xc, &yc);

    if(ac == '\r')
      {
      break;
      }
    *mp = ac;
    mp++;
    }
  *mp = '\0';

  /* tell drive to shut up */
  sys(UNTLK, &ac, &xc, &yc);

  return(msgbuf);
}


/* timer functions */

unsigned getclock();

/* sleep() - sleep for seconds */
sleep(usecs)
unsigned usecs;
{
  setclock((unsigned)0);

  while(getclock() < usecs)
    {
    }
}

struct clock  /* struct matches CIA */
  {
  char tenths;
  char seconds;
  char minutes;
  char hours;
  };

/* setclock() - set timer clock */
setclock(usecs)
unsigned usecs;
{
  unsigned bsecs;
  struct clock *clock1 = 0xdc08;
  char *clmode = 0xdc0f;

  *clmode &= 0x7f;  /* mode is clock */

  if(usecs > 59) usecs = 59;

  /* convert secs to bcd */
  bsecs = usecs%10 _ ((usecs/10)<<4);

  clock1->hours = 0;
  clock1->minutes = 0;
  clock1->seconds = (char)bsecs;
  clock1->tenths = 0;  /* free clock */
}

/* getclock() - get current clock secs */
unsigned getclock()
{
  unsigned usecs;
  char junk;
  struct clock *clock1 = 0xdc08;

  junk = clock1->seconds;
  usecs = (junk & 0x0f) +
          10 * (junk >> 4);

  junk = clock1->tenths; /* free clock */

  return usecs;
}

/* end of file */


/* xmodem.c - xmodem protocol */
/* W Mat Waites - Sept 1988 */

#include <stdio.h>

/* number of retries, timeouts */
#define RETRY 5
#define TOUT 2
#define BTOUT 10

/* protocol characters */
#define SOH 0x01
#define EOT 0x04
#define ACK 0x06
#define NAK 0x15
#define CAN 0x18

#define RECSIZE 128

char *diskerr();

int rec;
int tries;
int timeout;

/* buffer for data in/out */
char buffer[132];

/* sendfile() - send file via xmodem */
sendfile(fname, disk)
char *fname;
int disk;
{
  int st;
  int ch;
  char errbuf[41];
  char locname[21];
  char *status = 0x0090;
  FILE dfile;
```

```
    rec = 1;

    strcpy(locname, fname);
    strcat(locname, ",r");

    /* attempt to open file for read */
    device(disk);
    dfile = fopen(locname);

    /* check for disk error */
    strcpy(errbuf, diskerr(disk));
    st = atoi(errbuf);
    if(st >= 20)
      {
      close(dfile);
      showerr(fname, errbuf);
      return(0);
      }

    printf("%s opened\n", fname);

    /* clear input buffer */
    while(getserial() >= 0)
      {
      }

    tries = RETRY;

    for(;;)
      {
      printf("Synching...\n");
      if(chkstop())
        {
        close(dfile);
        return(0);
        }
      ch = getchtmo(BTOUT);
      if(timeout)
        {
        printf("Timeout\n");
        tries--;
        if(tries > 0)
          {
          continue;
          }
        close(dfile);
        return(0);
        }
      if(ch == NAK)
        {
        break;
        }
      printf("Strange char [%02x]\n", ch);
      }

    printf("Synched\n");

    /* send the file */
    while(fillbuf(dfile, buffer))
      {
      if(chkstop())
        {
        close(dfile);
        return(0);
        }
      if(!txrec(buffer))
        {
        close(dfile);
        return(0);
        }
      }

    /* tell 'em we're done */
    putserial(EOT);
    for(;;)
      {
      ch = getchtmo(TOUT);
      if(timeout)
        {
        putserial(EOT);
        }
      else
        {
        if(ch == ACK)
          {
          printf("sent EOT\n\n");
          break;
          }
        }
      }

    close(dfile);
    printf("%s transferred\n\n", fname);
    return(1);
}

/* recvfile() - recv file via xmodem */
recvfile(fname, disk)
char *fname;
int disk;
{
    int st;
    int ch;
    int i;
    char r1, r2, dt;
    int response;
    char rchk;
    char locname[21];
    char errbuf[41];
    unsigned chksum;
    FILE dfile;

    rec = 1;

    strcpy(locname, fname);
    strcat(locname, ",w");

    /* attempt to open file for write */
    device(disk);
    dfile = fopen(locname);

    /* check for disk error */
    strcpy(errbuf, diskerr(disk));
    st = atoi(errbuf);
    if(st >= 20)
      {
      close(dfile);
      showerr(fname, errbuf);
      return(0);
      }

    printf("%s opened\n", fname);

    /* clear input queue */
    while(getserial() >= 0)
      {
      }

    /* transfer file */
    response = NAK;
    for(;;)
      {
      /* get a record */
      printf("Record %3d ", rec);
      tries = RETRY;
      for(;;)
        {
        if(chkstop())
          {
          close(dfile);
          return(0);
          }
        /* shake hands */
        putserial(response);
        /* get 1st char */
        ch = getchtmo(TOUT);
        if(timeout)
          {
          tries--;
          if(tries > 0)
            {
            continue;  /* try again */
            }
          printf("Can't sync w/sender\n");

          close(dfile);
          return(0);
          }
        if(ch == SOH)  /* beg of data */
          {
          break;
          }
        else if(ch == EOT) /* done */
          {
          printf("got EOT\n\n");
          close(dfile);
          putserial(ACK);
          printf("%s transferred\n\n",
            fname);
          return(1);
          }
        else if(ch == CAN)  /* cancelled */
          {
          close(dfile);
          printf("Transfer cancelled!\n");
          return(0);
          }
        else
          {
          printf("Strange char [%02x]\n", ch);
          gobble();  /* clear any weirdness */
          response = NAK;  /* and try again */
          }
        }

      response = NAK;
      r1 = getchtmo(TOUT);  /* record number */
      if(timeout)
        {
        printf("TMO on recnum\n");
        continue;
        }
      /* get 1's comp record number */
      r2 = getchtmo(TOUT);
      if(timeout)
        {
        printf("TMO on comp recnum\n");
        continue;
        }

      /* get data */
      chksum = 0;
      for(i=0; i<RECSIZE; i++)
        {
        dt = getchtmo(TOUT);
        if(timeout)
          {
          break;
          }
        buffer[i] = dt;
        chksum += dt;
        chksum &= 0xff;
        }
      /* check for data timeout */
      if(timeout)
        {
        printf("TMO on data\n");
        continue;
```

```
    }

   /* get checksum */
   rchk = getchtmo(TOUT);
   if(timeout)
    {
    printf("TMO on checksum\n");
    continue;
    }

   /* compare rec num and 1's comp */
   if((/r1 & 0xff) != (r2 & 0xff))
    {
    printf("Bad recnum's\n");
    continue;
    }

   /* compare checksum and local one */
   if(rchk != chksum)
    {
    printf("Bad checksum\n");
    response = NAK;
    continue;
    }

   if((r1 ==(rec-1) & 0xff)) /* dupe */
    {
    printf("Duplicate record\n");
    response = ACK;
    continue;
    }

   if(r1 != (rec & 0xff))
    {
    printf("Record numbering error\n");
    close(dfile);
    return(0);
    }

   rec++;

   /* write data to file */
   for(i=0; i<RECSIZE; i++)
    {
    putc(buffer[i], dfile);
    }

   printf("OK\n");
   response = ACK;
   }
}

/* showerr() - display disk error */
showerr(fname, errmsg)
char *fname;
char *errmsg;
{
  erase();
  move(11, 5);
  printf("Error accessing %s", fname);
  move(13, 5);
  printf("[%s]", errmsg);
  move(20, 5);
}

/* getchtmo() - get char w/timeout */
getchtmo(timlen)
int timlen;
{
  int serchar;

  timeout = 0;
  setclock((unsigned)0); /* start timer */

  for(;;)
   {
   serchar = getserial();
   if(serchar >= 0)
    {
    return(serchar);
    }

   if(getclock() >= timlen)
    {
    timeout = 1;
    return 0;
    }
   }
}

/* fillbuf() - get buffer of data */
fillbuf(filnum, buf)
int filnum;
char buf[];
{
  int i;
  int echk;
  char *status = 0x0090;

  for(i=0; i<RECSIZE; i++)
   {
   /* get a char from file */
   if((echk=fgetc(filnum)) == EOF)
    {
    break;
    }
   buf[i] = echk;
   }

  if(i == 0) return 0;

  /* set rest of buffer to CTRL-Z */
  for(; i<RECSIZE; i++)
   {
   buf[i] = (char)26;
   }
  return(1);
}

/* txrec() - send rec, get response */
txrec(buf)
char buf[];
{
  int i;
  int ch;
  unsigned chksum;

  tries = RETRY;

  for(;;)
   {
   /* send record */

   printf("Record %3d ", rec);
   putserial(SOH);
   putserial(rec);
   putserial(/rec);
   chksum = 0;
   for(i=0; i<RECSIZE; i++)
    {
    putserial(buf[i]);
    chksum += buf[i];
    chksum &= 0xff;
    }
   putserial(chksum);

   /* get response */
   ch = getchtmo(BTOUT);
   if(timeout)
    {
    tries--;
    if(tries > 0)
     {
     printf("Retrying...\n");
     continue;
     }
    printf("Timeout\n");
    return(0);
    }

   /* analyze response */
   if(ch == CAN)
    {
    printf("Cancelled\n");
    return(0);
    }
   else if(ch == ACK)
    {
    printf("ACKed\n", rec);
    break;
    }
   else
    {
    if(ch == NAK)
     {
     printf("NAKed\n", rec);
     }
    else
     {
     printf("Strange response\n");
     }
    tries--;
    if(tries > 0)
     {
     continue;
     }
    printf("No more retries!\n");
    return(0);
    }
   }
  rec++;
  return(1);
}

/* gobble() - gobble up stray chars */
gobble()
{
  unsigned gotone;

  printf("\ngobbling\n");

  sleep(2);

  for(;;)
   {
   gotone = 0;
   /* clear input queue */
   while(getserial() >= 0)
    {
    gotone = 1;
    }
   if(gotone)
    {
    sleep(1);
    }
   else
    {
    return;
    }
   }
}

/* end of file */
```

Toward 2400

RS-232 revisited

by George Hug

The performance of 2400-baud modems with C64s and C128s will benefit from a new look at the RS-232 servicing routines. That performance is poor at 1 MHz, and errors occur even at 2 MHz when data flows continuously or in both directions at once. The 64 and 128-mode RS-232 drivers (which are almost identical) are inefficient and contain several outright bugs. There is even a hardware glitch in many 6526 CIA chips. New routines overcoming these faults will permit error-free communication at 2400 baud at either CPU speed.

Commodore RS-232

The RS-232 drivers send and receive data one bit at a time. At 2400 baud the transmit driver runs Timer A of CIA#2 in continuous mode with a latch value of 426 (the 1-MHz I/O clock divided by 2400). On each timeout, the NMI service routine places the next bit of outgoing data on pin M (PA2) of the User Port. Ten NMIs must be serviced to transmit one byte of 8/N/1 data.

Timer B is used for received data, which enters on User Port pins B (FLAG) and C (PB0). The high-to-low transition at the beginning of the start bit generates a FLAG NMI. In response, the service routine disables the FLAG NMI, enables the Timer B NMI, and sets Timer B to time out at the mid-point of the start bit. (The service code itself uses 100 of the 213 cycles in a half-bit, leaving a timer load value of 113.) When the NMI occurs, Timer B is set to time out every 426 cycles so that pin C can be sampled at the mid-point of each bit period. At mid-stop-bit the NMIs are switched back to start-bit detection mode - FLAG enabled, Timer B disabled. Eleven NMIs must be serviced to receive one byte of data.

RS-232 inefficiencies

The following characteristics do not produce errors as such, but needlessly limit the baud rate attainable at a given CPU speed. All relate to the receive function.

1. During each byte, 450 clock cycles are consumed in manually re-starting Timer B once per bit. The re-start routine - located at $fedd ($e87f in 128 mode) - determines how long ago the timeout occurred, adds an allowance for its own execution time, subtracts that sum from the bit time, and loads the timer with the difference. After the timer is started, its reload latch is reset to $ffff. This appears to be a software emulation of the VIC 20's 6522 VIA chip. The VIA's Timer B has no continuous mode, but its one-shot mode underflows to $ffff and continues counting down. Since the CIA's Timer B does operate in continuous mode, the VIA emulation seems to be pointless.

2. The RS-232 driver is biased toward a late sampling of pin C. The sampling point is 70 cycles late in the first place because of code execution time between the mid-bit NMI and the actual sampling. (In 128 mode, sampling begins 91 cycles late, but works back to near mid-point at a rate of 12 cycles per NMI.) In addition, since the VIA emulation manually restarts Timer B, any video DMA during that process may cause a permanent, cumulative, 40-cycle delay in all subsequent samplings of the current byte. Finally, the actual pin-C data rate of a 1200- or 2400-baud modem may range from the nominal baud rate to as much as 1.6% fast. The combination of late sampling points and short bit periods may result in a sampling past the end of a bit period.

3. Continuous data flow hits a bottleneck at the junction of the stop and start bits, where three NMIs occur within one bit period. For example, the mid-stop-bit NMI requires 287 clock cycles to service (325 cycles in 128 mode - the extra time results from saving and restoring the current bank), but at 2400 baud the next byte may start after only 213 of those cycles. A related limitation is the 2224 cycles (2639 cycles in 128 mode) of total NMI service time needed to receive one byte of data. At 1 MHz, continuous 2400-baud inflow requires 55% (66%) of available clock cycles just to service NMIs.

RS-232 bugs

The defects described below cause errors without regard to baud rate, mode or CPU speed.

1. The routine at $f0a4 ($e7ec in 128 mode) disables all RS-232 activity so that NMIs will not corrupt disk, tape or REU access. It is called by the KERNAL routines LOAD, SAVE, OPEN, CHKIN, CHKOUT, LISTEN and TALK for serial bus devices and the datasette. It should be called, but is not, by DMACALL - the 128's REU routine at $ff50.

```
f0a4  pha
f0a5  lda $02a1    ;copy of enabled NMIs
f0a8  beq $f0bb    ;none enabled - done.
f0aa  lda $02a1    ;any current activity?
f0ad  and #$03     ;TA(b0) or TB(b1) enabled?
f0af  bne $f0aa    ;yes, test until idle
f0b1  lda #$10     ;no, awaiting start bit
f0b3  sta $dd0d    ;disable FLAG NMI (b4)
f0b6  lda #$00     ;all off now (?)
f0b8  sta $02a1    ;update copy
f0bb  pla
f0bc  rts
```

The $f0aa-f0af sequence pauses until the transmit buffer has been emptied and any incoming byte has been received. Then at $f0b3 it turns off the start-bit detector. However, if an incoming start-bit edge should arrive after $f0aa but before $f0b3, the resulting NMI servicing will disable FLAG (making $f0b3 redundant) and enable the Timer B NMI. Since $f0b8 clears only the mask copy, the Timer B NMI will indeed take place, with unpredictable results.

2. In the BSOUT routine the buffer pointer is incremented (at $f020/$e768) before the byte to be transmitted is placed in the buffer (at $f026/$e76e). If the NMI service routine comes looking for that new byte in the interim, it will transmit the wrong character.

3. The routine at $ef3b ($e67f) is used by the NMI service routines, and by CHKIN and BSOUT, to enable or disable an NMI source, as specified in the accumulator.

```
ef3b  sta $dd0d    ;enable or disable the NMI
ef3e  eor $02a1    ;change copy to match
ef41  ora #$80     ;enable bit on
ef43  sta $02a1    ;update copy
ef46  sta $dd0d    ;enable masked NMIs
ef49  rts
```

The routine is executed while NMIs are enabled. Should an NMI of the opposite "direction" occur after $ef3e and before $ef46, the resulting servicing may change the NMI enabled for that direction. Upon the return, however, $ef43 or $ef46, or both, will restore the old (wrong) NMI. This error occurs only when transmission takes place in both directions simultaneously.

4. It appears that many 6526 CIA chips have a hardware defect involving the interrupt flag for Timer B. If Timer B times out at about the same time as a read of the interrupt register, the Timer B flag may not be set at all. Under the VIA emulation, Timer B will then underflow and count down $ffff cycles before generating another NMI. A whole series of incoming bytes may be lost as a result. The defect was present in five of six C128s and two of three C64s sampled. When "good" and "bad" chips were switched, the problem followed the "bad" chip. There appear to be no such defects with respect to the flags for Timer A or FLAG. This glitch can cause errors during simultaneous I/O - when Timer A generates the NMI and Timer B times out just as the service routine reads $dd0d.

A software solution

The most demanding performance standard for full-duplex RS-232 is the error-free processing of continuous, bi-directional, asynchronous transmission ("CBAT"), meaning that data streams generated by unrelated clocks flow, without pause, in both directions at the same time. Fortunately, such performance at 2400 baud is attainable through software, even at 1 MHz. The approach presented here retains bit-by-bit servicing, but adopts a few key simplifications, beginning with elimination of two receive NMIs. The mid-start-bit NMI exists only to check for a false start bit, which for technical reasons would never be detected on a PSK/QAM modem. The mid-stop-bit NMI tests for a framing error, or missing stop bit, which is ignored by most software.

Another change is the removal from the NMI service routine of all matters related to parity, x-line handshake, half-duplex transmission, multiple stop bits, and the RSSTAT framing, parity, overrun and break errors. All such items take up time, are seldom used, and can be implemented separately if really needed. Finally, the VIA emulation is discarded.

New Modem Routines

Program 1 ("newmodem.src") is generic assembly language source code for a collection of new RS-232 routines. The code is not a patch to any specific BBS or terminal software, but rather one example of what might be installed by the author of such a program, or by one having access to its source code. The assembled code uses less than two pages of memory. In 128 mode it must be visible in bank 15.

The new NMI routine begins at line 3000 by pushing the registers onto the stack (already done in 128 mode, which enters at 3050). Lines 3060-3170 determine which enabled NMI sources have triggered. The 6526 glitch is finessed by comparing the high byte of Timer B before (3060) and after (3110) the read of the interrupt register (3090). If the value is higher after the read than before, then Timer B must have timed out during that period. Line 3140 makes sure B's flag bit is set in the accumulator, and 3150 makes sure it is cleared in $dd0d.

Beginning at 3180 the routine is structured to accommodate CBAT. The NMI routine does only a few critical operations while the NMIs are disabled, saving its "housekeeping" chores for later. That prevents a new NMI (one occurring after 3090) from going unserviced for too long. The critical operation for the Timer A NMI is placement of the next outgoing bit on pin M (3200-3230). The FLAG NMI must load Timer B with the start-bit timer value and start it counting down (3270-3320). The Timer B NMI must sample pin C (3120). (Pin C is sampled on every NMI; the sampling is ignored if Timer B is not an NMI source.) Once these operations have been completed, the NMIs are re-enabled (3360 or 3470).

Housekeeping chores for the Timer A NMI (3720-3920) include isolating the next output bit, or fetching the next byte from the transmit buffer, or stopping Timer A and disabling its NMI if the buffer is empty. (Timer A is loaded and started, and its NMI enabled, only by BSOUT.) In FLAG housekeeping (3330-3420), the FLAG NMI is disabled and the Timer B NMI enabled, the Timer B reload latch is loaded with the full-bit timer value, and the bit counter is initialized. Timer B housekeeping (3510-3680) processes the sampled pin-C value. If the last data bit has been received, the new byte is stored in the receive buffer, Timer B is stopped, and the NMIs are prepared for a new start bit - FLAG enabled, Timer B disabled.

The procedure at 3630 is used to change the enabled NMIs. It disables all NMIs, calculates the new configuration, and then enables that configuration. The duplicate disabling instructions at 3640/50 are necessary because an NMI occurring during the first one will be serviced immediately thereafter, resulting in re-enabled NMIs which must be disabled again by the second (there is nothing left to interrupt the second).

Following the new NMI routine are replacements for the defective routines described earlier. A new DISABL at line 4000 is a substitute for the old one at $f0a4/e7ec. Since the old one cannot be re-vectored, a call to DISABL should be made before any disk, tape or REU operation if there is any chance that the modem might generate an NMI. The NBSOUT routine at 5000 is a new front end for BSOUT which corrects the buffer pointer problem and avoids a call to $ef3b/e67f. A direct call to RSOUT (5050) will send a character to the modem regardless of the current output device.

NCHKIN at 6000 is a new front end for CHKIN which avoids $ef3b/e67f. A direct call to INABLE (6070) will re-enable the RS-232 input function without also selecting device #2 for input. Either NCHKIN (to #2) or INABLE must be called after disk, tape or REU operations to re-enable start bit detection. The BAUD section (6090-6190) sets the receive baud rate by poking the correct timer values into the NMI service code. It assumes that the current baud rate is already reflected in the BAUDOF variable at $299 ($a16), and selects one of three baud rates (2400, 1200, or 300) based on the high byte value of BAUDOF. If NCHKIN (to #2) or INABLE will be called frequently, BAUD should be moved to a separate routine which is called only after OPEN or when the baud rate needs to be changed. Provision could also be made for additional baud rates if needed.

RSGET at 7000 will fetch a character from the RS-232 input buffer regardless of the current input device. It differs from GETIN in that it does nothing to RSSTAT but instead returns with the carry flag set if the buffer was empty.

The SETUP routine at 2000 points the relevant page 3 vectors to the new NMI, NCHKIN and NBSOUT code. SETUP is the first entry in the jump table (1530). Also included in the jump table are the non-vectored routines INABLE, DISABL, RSGET and RSOUT. Finally, the receive start-bit and full-bit timer values for the three baud rates are located in a table beginning at 1590.

Calibration and performance

The new NMI routine was tested under CBAT conditions to establish the receive timer values which work for various combinations of computer, CPU speed, video DMA activity and modem speed. The tests made use of the fact that a 50%-duty-cycle square wave also constitutes continuous transmission of the letter 'u' (%01010101) in RS-232 8/N/1 format. The square wave was generated using the serial port of CIA#1, the clock output of which (CNT1) is available at pin 4 of the User Port. A spare card-edge connector (Cinch #50-24A-30) was installed in the User Port with pins 4, B and C wired together.

Program 2 ('calibrate') was used to run the tests. It keeps the CNT1 "modem" clocking continuously by feeding new output to the CIA#1 serial port during the IRQ routine. It parks in a GETIN loop which prints an '*' to the screen if a received character is not a 'u'. Program 2 also provides for continuous transmission by filling the output buffer with u's and changing line 3820 to read, in effect, 'beq getbuf'.

Timer values for the receive start bit (sb) and full bit (fb), the CNT1 "modem" (cn), and the transmit function (tx) are set in line 210 of Program 2 for each trial, which consists of running the program and looking for asterisks. If none appear then CBAT processing is error-free at those settings. One minute is enough to run through the possible overlaps of transmit and receive NMIs, and video DMAs if enabled. Asynchronous timing is approximated if the fb, cn, and tx values are different.

Table 1 shows the 2400-baud test results with the tx rate fixed at 2400 and the fb rate fixed midway between 2400 and 1.6% fast. For each hardware combination, the tests determined the highest and lowest start-bit times (sb) providing error-free CBAT. While the acceptable sb range varies with each set-up, there is a 70-cycle range, with a mid-point of 459, which works in all set-ups. Any change to the new NMI routine would require re-calibration, and the results might be different.

Table 2 compares the NMI service times required under the old and new routines. Reductions are particularly dramatic in the receive function.

Program 3 ('ciatest64') tests for the glitch in Timers A and B of CIA#2. Load and run in 64 mode only, without the card edge connector. Only a Timer B glitch has been found so far.

For transmission in only one direction at a time, the 'newmodem' routines should be replaced with shorter, faster ones. The "simultaneous" bugs will no longer occur, separate routines for each NMI type can be vectored in at $318/319 in sequence, and NMIs need not be disabled during servicing. Much higher baud rates can be attained under those conditions.

Random thoughts

1. The usual caveats apply about cartridges, special ROMs, IEEE drivers, and connecting anything homemade to the User Port.

2. CIA chips produce a count equal to the timer load value plus one. So a 425 timer value is really 1022727/426 = 2400.8 baud.

3. The SLOW command turns on the video DMAs even in 80-column mode (the 40-column screen shows a border). Turn off the DMAs by clearing the blanking bit - bit 4 of $d011. Program 2 does that through variable dm.

4. New drivers will not cure aborted Xmodem or Punter transfers caused by running 1 MHz transfer routines at 2 MHz, but they will permit the routines to be run at 1 MHz without modem errors.

5. Program 1 starts and stops the timers and also enables and disables their NMIs. If nothing else uses the timers, the NMIs could be left enabled. Time also might be saved by having the transmit NMIs occur only when the level on pin M needs to change, or at the stop bit, whichever occurs first.

Table 1: Calibration Results for 2400 Baud.

Computer mode	64	128	128	128
CPU speed (MHz)	1	1	1	2
Display mode	40	40/80	80	80
Video DMAs	on	on	off	off
TX (Tx bit time)	425	425	425	425
FB (Rx full bit)	421	421	421	421
Nominal modem:				
CN (CNT1 "modem")	426	426	426	426
Low SB (Rx start)	394	392	330	424*
High SB	568	538	618	724
Fast modem:				
CN	418	418	418	418
Low SB	350	348	290	354
High SB	524	494*	580	688

* Most restrictive. Mid-point = 459.

Table 2: NMI Service Times (cycles per byte).

	64 Mode		128 Mode	
	OLD	NEW	OLD	NEW
Transmit:				
Data bits 1-8	1320	1192	1624	1360
Stop bit	196	148	234	169
Start bit	179	173	217	194
Total	1695	1513	2075	1723
Receive:				
FLAG	157	153	195	174
Start bit	188	-	223	-
Data bits 1-7	1393	959	1659	1106
Data bit 8	199	185	237	206
Stop bit	287	-	325	-
Total	2224	1297	2639	1486

Program 1: Source code for the new serial modem routines.

```
1100 ;---------------------------------------------
1110 ;  "newmodem.src" - 64 mode.
1120 ;  @128 = changes for 128 mode.
1130 ;---------------------------------------------
1140 ribuf  =$f7          ;@128 $c8
1150 robuf  =$f9          ;@128 $ca
1160 baudof =$0299        ;@128 $0a16
1170 ridbe  =$029b        ;@128 $0a18
1180 ridbs  =$029c        ;@128 $0a19
1190 rodbs  =$029d        ;@128 $0a1a
1200 rodbe  =$029e        ;@128 $0a1b
1210 enabl  =$02a1        ;@128 $0a0f
1220 rstkey =$fe56        ;@128 $fa4b
1230 norest =$fe72        ;@128 $fa5f
1240 return =$febc        ;@128 $ff33
1250 oldout =$f1ca        ;@128 $ef79
1260 oldchk =$f21b        ;@128 $f10e
1270 findfn =$f30f        ;@128 $f202
1280 devnum =$f31f        ;@128 $f212
1290 nofile =$f701        ;@128 $f682
1500 ;---------------------------------------------
1510 *      =$ce00        ;@128 $1a00
1520 ;---------------------------------------------
1530 xx00   jmp setup
1540 xx03   jmp inable
1550 xx06   jmp disabl
1560 xx09   jmp rsget
1570 xx0c   jmp rsout
1580        nop
1590 strt24 .word $01cb   ; 459 start-bit times
1600 strt12 .word $0442   ;1090
1610 strt03 .word $1333   ;4915
1620 full24 .word $01a5   ; 421 full-bit times
1630 full12 .word $034d   ; 845
1640 full03 .word $0d52   ;3410
1650 ;---------------------------------------------
2000 setup  lda #<nmi64   ;@128 #<nmi128
2010        ldy #>nmi64   ;@128 #>nmi128
2020        sta $0318
2030        sty $0319
2040        lda #<nchkin
2050        ldy #>nchkin
2060        sta $031e
2070        sty $031f
2080        lda #<nbsout
2090        ldy #>nbsout
2100        sta $0326
2110        sty $0327
2120        rts
2130 ;---------------------------------------------
3000 nmi64  pha           ;new nmi handler
3010        txa
3020        pha
3030        tya
3040        pha
3050 nmi128 cld
3060        ldx $dd07     ;sample timer b hi byte
3070        lda #$7f      ;disable cia nmi's
3080        sta $dd0d
3090        lda $dd0d     ;read/clear flags
3100        bpl notcia    ;(restore key)
3110        cpx $dd07     ;tb timeout since 3060?
```

```
3120         ldy $dd01     ;(sample pin c)
3130         bcs mask      ;no
3140         ora #$02      ;yes, set flag in acc.
3150         ora $dd0d     ;read/clear flags again
3160 mask    and enabl     ;mask out non-enabled
3170         tax           ;these must be serviced
3180         lsr           ;timer a? (bit 0)
3190         bcc ckflag    ;no
3200         lda $dd00     ;yes, put bit on pin m
3210         and #$fb
3220         ora $b5
3230         sta $dd00
3240 ckflag  txa
3250         and #$10      ;*flag nmi? (bit 4)
3260         beq nmion     ;no
3270 strtlo  lda #$42      ;yes, start-bit to tb
3280         sta $dd06
3290 strthi  lda #$04
3300         sta $dd07
3310         lda #$11      ;start tb counting
3320         sta $dd0f
3330         lda #$12      ;*flag nmi off, tb on
3340         eor enabl     ;update mask
3350         sta enabl
3360         sta $dd0d     ;enable new config.
3370 fulllo  lda #$4d      ;change reload latch
3380         sta $dd06     ;  to full-bit time
3390 fullhi  lda #$03
3400         sta $dd07
3410         lda #$08      ;# of bits to receive
3420         sta $a8
3430         bne chktxd    ;branch always
3440 notcia  ldy #$00
3450         jmp rstkey    ;or jmp norest
3460 nmion   lda enabl     ;re-enable nmi's
3470         sta $dd0d
3480         txa
3490         and #$02      ;timer b? (bit 1)
3500         beq chktxd    ;no
3510         tya           ;yes, sample from 3120
3520         lsr
3530         ror $aa       ;rs232 is lsb first
3540         dec $a8       ;byte finished?
3550         bne txd       ;no
3560         ldy ridbe     ;yes, byte to buffer
3570         lda $aa
3580         sta (ribuf),y ;(no overrun test)
3590         inc ridbe
3600         lda #$00      ;stop timer b
3610         sta $dd0f
3620         lda #$12      ;tb nmi off, *flag on
3630 switch  ldy #$7f      ;disable nmi's
3640         sty $dd0d     ;twice
3650         sty $dd0d
3660         eor enabl     ;update mask
3670         sta enabl
3680         sta $dd0d     ;enable new config.
3690 txd     txa
3700         lsr           ;timer a?
3710 chktxd  bcc exit      ;no
3720         dec $b4       ;yes, byte finished?
3730         bmi char      ;yes
3740         lda #$04      ;no, prep next bit
3750         ror $b6       ;(fill with stop bits)
3760         bcs store
3770 low     lda #$00
3780 store   sta $b5
3790 exit    jmp return    ;restore regs, rti
3800 char    ldy rodbs
3810         cpy rodbe     ;buffer empty?
3820         beq txoff     ;yes
3830 getbuf  lda (robuf),y ;no, prep next byte
3840         inc rodbs
3850         sta $b6
3860         lda #$09      ;# bits to send
3870         sta $b4
3880         bne low       ;always - do start bit
3890 txoff   ldx #$00      ;stop timer a
3900         stx $dd0e
3910         lda #$01      ;disable ta nmi
3920         bne switch    ;always
3930 ;---------------------------------------------
4000 disabl  pha           ;turns off modem port
4010 test    lda enabl
4020         and #$03      ;any current activity?
4030         bne test      ;yes, test again
4040         lda #$10      ;no, disable *flag nmi
4050         sta $dd0d
4060         lda #$02
4070         and enabl     ;currently receiving?
4080         bne test      ;yes, start over
4090         sta enabl     ;all off, update mask
4100         pla
4110         rts
4120 ;---------------------------------------------
5000 nbsout  pha           ;new bsout
5010         lda $9a
5020         cmp #$02
5030         bne notmod
5040         pla
5050 rsout   sta $9e       ;output to modem
5060         sty $97
5070 point   ldy rodbe
5080         sta (robuf),y ;not official till 5120
5090         iny
5100         cpy rodbs     ;buffer full?
5110         beq fulbuf    ;yes
5120         sty rodbe     ;no, bump pointer
5130 strtup  lda enabl
5140         and #$01      ;transmitting now?
5150         bne ret3      ;yes
5160         sta $b5       ;no, prep start bit,
5170         lda #$09
5180         sta $b4       ;  # bits to send,
5190         ldy rodbs
5200         lda (robuf),y
5210         sta $b6       ;  and next byte
5220         inc rodbs
5230         lda baudof    ;full tx bit time to ta
5240         sta $dd04
5250         lda baudof+1
5260         sta $dd05
5270         lda #$11      ;start timer a
5280         sta $dd0e
```

```
5290        lda #$81      ;enable ta nmi
5300 change sta $dd0d     ;nmi clears flag if set
5310        php           ;save irq status
5320        sei           ;disable irq's
5330        ldy #$7f      ;disable nmi's
5340        sty $dd0d     ;twice
5350        sty $dd0d
5360        ora enabl     ;update mask
5370        sta enabl
5380        sta $dd0d     ;enable new config.
5390        plp           ;restore irq status
5400 ret3   clc
5410        ldy $97
5420        lda $9e
5430        rts
5440 fulbuf jsr strtup
5450        jmp point
5460 notmod pla           ;back to old bsout
5470        jmp oldout
5480 ;---------------------------------------------
6000 nchkin jsr findfn    ;new chkin
6010        bne nosuch
6020        jsr devnum
6030        lda $ba
6040        cmp #$02
6050        bne back
6060        sta $99
6070 inable sta $9e       ;enable rs232 input
6080        sty $97
6090 baud   lda baudof+1  ;set receive to same
6100        and #$06      ;  baud rate as xmit
6110        tay
6120        lda strt24,y
6130        sta strtlo+1  ;overwrite value @ 3270
6140        lda strt24+1,y
6150        sta strthi+1
6160        lda full24,y
6170        sta fulllo+1
6180        lda full24+1,y
6190        sta fullhi+1
6200        lda enabl
6210        and #$12      ;*flag or tb on?
6220        bne ret1      ;yes
6230        sta $dd0f     ;no, stop tb
6240        lda #$90      ;turn on flag nmi
6250        jmp change
6260 nosuch jmp nofile
6270 back   lda $ba
6280        jmp oldchk
6290 ;---------------------------------------------
7000 rsget  sta $9e       ;input from modem
7010        sty $97
7020        ldy ridbs
7030        cpy ridbe     ;buffer empty?
7040        beq ret2      ;yes
7050        lda (ribuf),y ;no, fetch character
7060        sta $9e
7070        inc ridbs
7080 ret1   clc           ;cc = char in acc.
7090 ret2   ldy $97
7100        lda $9e
7110 last   rts           ;cs = buffer was empty
```

Program 2: Calibration program for the 64 or 128.

```
PD 100 rem "calibrate"  for 64 or 128.
MA 110 rem  connect user port pins 4, b & c.
CE 120 rem  load "newmodem" object code at pl.
DK 130 rem  for 128 mode, un-rem 230-250.
LJ 140 rem  adjust values in 210. run. * = error.
LI 150 rem  run/stop restore to end trial.
MG 160 rem  s = (1,2) mhz; dm = dma off(0), on(1).
IM 170 rem
CL 180 close 2: open 2,2,0,chr$(6)+chr$(0): ml=12288
LP 190 for i=ml to ml+116: read a: poke i,a: z=z+a: next
DO 200 if z<>15157 then print"data error": close2: end
PC 210 sb=459: fb=421: cn=418: tx=425: s=1: dm=1
EJ 220 ri=65212: bf=peek(250)*256: bo=665: pl=52736
NH 230 rem ri=65331: bf=3328: bo=2582: pl=6656
HI 240 rem slow: if s=2 then fast: goto 260
NO 250 rem if dm=0 and peek(215)then poke ml+107,234
FG 260 for i=bf to bf+255: poke i,85: next: sys pl
KL 270 a=pl+16+(tx/256 and 6): b=sb: gosub 310
IO 280 a=a+6: b=fb: gosub 310: a=bo: b=tx: gosub310
DI 290 a=251: b=cn: gosub 310: a=598: b=ri: gosub310
NP 300 poke pl+241,0: print#2,"u";: sys ml
GG 310 q=int(b/256): poke a+1,q: poke a,b-q*256: return
HI 320 data 162,   2,  32, 198, 255,  32,  39,  48
PO 330 data  32, 228, 255, 201,  85, 240, 249,  32
DI 340 data 183, 255, 208, 244, 169,  42,  32, 210
CJ 350 data 255,  76,   8,  48, 169, 255, 141,  12
EJ 360 data 220, 173,  13, 220, 108,  86,   2, 120
FA 370 data 166, 251, 164, 252, 169,   0, 141,  26
KM 380 data 208, 141,  15, 220, 169, 127, 141,  13
IA 390 data 220, 141,  25, 208, 142,   4, 220, 140
AI 400 data   5, 220, 169,  81, 141,  14, 220, 160
BL 410 data 255, 140,  12, 220, 162,   5, 173,  13
IA 420 data 220,  41,   1, 240, 249, 202, 208, 246
KJ 430 data 140,  12, 220, 169,  28, 141,  20,   3
PP 440 data 169,  48, 141,  21,   3, 169, 136, 141
IK 450 data  13, 220,  88,  96, 173,  17, 208,  41
AG 460 data 239, 141,  17, 208,  96
```

Program 3: CIA chip test for the 64.

```
LO 500 rem "ciatest64"  for 64 mode only.
MA 510 rem  * = interrupt flag error.
HG 520 rem  reset after test.
AD 530 rem
BL 540 n=12800: for i=n to n+103: read a: poke i,a: z=z+a
OA 550 next: if z<>11949 then print"data error":end
EG 560 sys 65412: x=not x: poke 251,x and 255
OO 570 print chr$(147);"any key switches timer."
ID 580 print"testing timer ";chr$(65-x): sys n
JB 590 wait 198,7: poke 198,0: goto 560
EI 610 data 170, 169,  98, 160,   3, 141,   4, 221
JG 620 data 140,   5, 221, 142,   6, 221, 140,   7
GO 630 data 221, 169,  17, 141,  14, 221, 141,  15
EL 640 data 221, 162,   2, 160,   7,  36, 251,  48
GG 650 data   3, 202, 160,   5, 134, 252, 140,  77
OL 660 data  50, 140,  85,  50, 138,  73, 131, 162
JL 670 data  72, 160,  50, 142,  24,   3, 140,  25
EH 680 data   3, 174,  13, 221, 141,  13, 221,  96
FL 690 data  72, 138,  72, 152, 172,   7, 221,  72
FF 700 data 173,  13, 221, 216, 204,   7, 221, 176
NM 710 data  12,  13,  13, 221,  37, 252, 208,   5
JE 720 data 169,  42,  32, 210, 255,  76, 188, 254
```

Program 4: Generator for the C64 new modem routines.

```
BC 100 rem  generator for "newmod64.obj"
FL 110 n$="newmod64.obj": rem  name of program
GF 120 nd=494: sa=52736: ch=58580

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

DE 1000 data  76,  28, 206,  76, 156, 207,  76,   8
IJ 1010 data 207,  76, 213, 207,  76,  41, 207, 234
IM 1020 data 203,   1,  66,   4,  51,  19, 165,   1
CI 1030 data  77,   3,  82,  13, 169,  59, 160, 206
JB 1040 data 141,  24,   3, 140,  25,   3, 169, 140
NB 1050 data 160, 207, 141,  30,   3, 140,  31,   3
NG 1060 data 169,  33, 160, 207, 141,  38,   3, 140
JA 1070 data  39,   3,  96,  72, 138,  72, 152,  72
EK 1080 data 216, 174,   7, 221, 169, 127, 141,  13
CE 1090 data 221, 173,  13, 221,  16,  77, 236,   7
JF 1100 data 221, 172,   1, 221, 176,   5,   9,   2
OF 1110 data  13,  13, 221,  45, 161,   2, 170,  74
CI 1120 data 144,  10, 173,   0, 221,  41, 251,   5
CG 1130 data 181, 141,   0, 221, 138,  41,  16, 240
JN 1140 data  47, 169,  66, 141,   6, 221, 169,   4
HH 1150 data 141,   7, 221, 169,  17, 141,  15, 221
MM 1160 data 169,  18,  77, 161,   2, 141, 161,   2
JM 1170 data 141,  13, 221, 169,  77, 141,   6, 221
NL 1180 data 169,   3, 141,   7, 221, 169,   8, 133
IN 1190 data 168, 208,  60, 160,   0,  76,  86, 254
NK 1200 data 173, 161,   2, 141,  13, 221, 138,  41
MM 1210 data   2, 240,  44, 152,  74, 102, 170, 198
HG 1220 data 168, 208,  34, 172, 155,   2, 165, 170
JP 1230 data 145, 247, 238, 155,   2, 169,   0, 141
AG 1240 data  15, 221, 169,  18, 160, 127, 140,  13
AP 1250 data 221, 140,  13, 221,  77, 161,   2, 141
NO 1260 data 161,   2, 141,  13, 221, 138,  74, 144
LB 1270 data  14, 198, 180,  48,  13, 169,   4, 102
JC 1280 data 182, 176,   2, 169,   0, 133, 181,  76
BE 1290 data 188, 254, 172, 157,   2, 204, 158,   2
JA 1300 data 240,  13, 177, 249, 238, 157,   2, 133
EE 1310 data 182, 169,   9, 133, 180, 208, 228, 162
EB 1320 data   0, 142,  14, 221, 169,   1, 208, 188
NH 1330 data  72, 173, 161,   2,  41,   3, 208, 249
JD 1340 data 169,  16, 141,  13, 221, 169,   2,  45
KC 1350 data 161,   2, 208, 237, 141, 161,   2, 104
EF 1360 data  96,  72, 165, 154, 201,   2, 208,  96
PH 1370 data 104, 133, 158, 132, 151, 172, 158,   2
GM 1380 data 145, 249, 200, 204, 157,   2, 240,  74
MN 1390 data 140, 158,   2, 173, 161,   2,  41,   1
DE 1400 data 208,  58, 133, 181, 169,   9, 133, 180
NK 1410 data 172, 157,   2, 177, 249, 133, 182, 238
FI 1420 data 157,   2, 173, 153,   2, 141,   4, 221
KI 1430 data 173, 154,   2, 141,   5, 221, 169,  17
JO 1440 data 141,  14, 221, 169, 129, 141,  13, 221
AN 1450 data   8, 120, 160, 127, 140,  13, 221, 140
KL 1460 data  13, 221,  13, 161,   2, 141, 161,   2
EB 1470 data 141,  13, 221,  40,  24, 164, 151, 165
LA 1480 data 158,  96,  32,  59, 207,  76,  45, 207
BF 1490 data 104,  76, 202, 241,  32,  15, 243, 208
FJ 1500 data  60,  32,  31, 243, 165, 186, 201,   2
NA 1510 data 208,  54, 133, 153, 133, 158, 132, 151
FG 1520 data 173, 154,   2,  41,   6, 168, 185,  16
MO 1530 data 206, 141, 114, 206, 185,  17, 206, 141
MP 1540 data 119, 206, 185,  22, 206, 141, 140, 206
FE 1550 data 185,  23, 206, 141, 145, 206, 173, 161
JB 1560 data   2,  41,  18, 208,  35, 141,  15, 221
AI 1570 data 169, 144,  76, 101, 207,  76,   1, 247
NG 1580 data 165, 186,  76,  27, 242, 133, 158, 132
IE 1590 data 151, 172, 156,   2, 204, 155,   2, 240
DG 1600 data   8, 177, 247, 133, 158, 238, 156,   2
IP 1610 data  24, 164, 151, 165, 158,  96
```

Program 5: Generator for the C128 new modem routines.

```
MN 100 rem  generator for "newmod128.obj"
NC 110 n$="newmod128.obj": rem  name of program
DG 120 nd=494: sa=6656: ch=51020

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

KG 1000 data  76,  28,  26,  76, 156,  27,  76,   8
KD 1010 data  27,  76, 213,  27,  76,  41,  27, 234
IM 1020 data 203,   1,  66,   4,  51,  19, 165,   1
PN 1030 data  77,   3,  82,  13, 169,  64, 160,  26
JB 1040 data 141,  24,   3, 140,  25,   3, 169, 140
KK 1050 data 160,  27, 141,  30,   3, 140,  31,   3
JB 1060 data 169,  33, 160,  27, 141,  38,   3, 140
JA 1070 data  39,   3,  96,  72, 138,  72, 152,  72
EK 1080 data 216, 174,   7, 221, 169, 127, 141,  13
CE 1090 data 221, 173,  13, 221,  16,  77, 236,   7
JF 1100 data 221, 172,   1, 221, 176,   5,   9,   2
KF 1110 data  13,  13, 221,  45,  15,  10, 170,  74
CI 1120 data 144,  10, 173,   0, 221,  41, 251,   5
CG 1130 data 181, 141,   0, 221, 138,  41,  16, 240
JN 1140 data  47, 169,  66, 141,   6, 221, 169,   4
HH 1150 data 141,   7, 221, 169,  17, 141,  15, 221
IM 1160 data 169,  18,  77,  15,  10, 141,  15,  10
JM 1170 data 141,  13, 221, 169,  77, 141,   6, 221
NL 1180 data 169,   3, 141,   7, 221, 169,   8, 133
BM 1190 data 168, 208,  60, 160,   0,  76,  75, 250
IK 1200 data 173,  15,  10, 141,  13, 221, 138,  41
MM 1210 data   2, 240,  44, 152,  74, 102, 170, 198
DE 1220 data 168, 208,  34, 172,  24,  10, 165, 170
MN 1230 data 145, 200, 238,  24,  10, 169,   0, 141
AG 1240 data  15, 221, 169,  18, 160, 127, 140,  13
BP 1250 data 221, 140,  13, 221,  77,  15,  10, 141
IO 1260 data  15,  10, 141,  13, 221, 138,  74, 144
LB 1270 data  14, 198, 180,  48,  13, 169,   4, 102
JC 1280 data 182, 176,   2, 169,   0, 133, 181,  76
GP 1290 data  51, 255, 172,  26,  10, 204,  27,  10
LL 1300 data 240,  13, 177, 202, 238,  26,  10, 133
EE 1310 data 182, 169,   9, 133, 180, 208, 228, 162
EB 1320 data   0, 142,  14, 221, 169,   1, 208, 188
HG 1330 data  72, 173,  15,  10,  41,   3, 208, 249
JD 1340 data 169,  16, 141,  13, 221, 169,   2,  45
GC 1350 data  15,  10, 208, 237, 141,  15,  10, 104
EF 1360 data  96,  72, 165, 154, 201,   2, 208,  96
CI 1370 data 104, 133, 158, 132, 151, 172,  27,  10
PK 1380 data 145, 202, 200, 204,  26,  10, 240,  74
AO 1390 data 140,  27,  10, 173,  15,  10,  41,   1
DE 1400 data 208,  58, 133, 181, 169,   9, 133, 180
AJ 1410 data 172,  26,  10, 177, 202, 133, 182, 238
NH 1420 data  26,  10, 173,  22,  10, 141,   4, 221
BI 1430 data 173,  23,  10, 141,   5, 221, 169,  17
JO 1440 data 141,  14, 221, 169, 129, 141,  13, 221
AN 1450 data   8, 120, 160, 127, 140,  13, 221, 140
GL 1460 data  13, 221,  13,  15,  10, 141,  15,  10
EB 1470 data 141,  13, 221,  40,  24, 164, 151, 165
CK 1480 data 158,  96,  32,  59,  27,  76,  45,  27
JA 1490 data 104,  76, 121, 239,  32,   2, 242, 208
PJ 1500 data  60,  32,  18, 242, 165, 186, 201,   2
NA 1510 data 208,  54, 133, 153, 133, 158, 132, 151
MF 1520 data 173,  23,  10,  41,   6, 168, 185,  16
MC 1530 data  26, 141, 114,  26, 185,  17,  26, 141
KB 1540 data 119,  26, 185,  22,  26, 141, 140,  26
MB 1550 data 185,  23,  26, 141, 145,  26, 173,  15
PO 1560 data  10,  41,  18, 208,  35, 141,  15, 221
BM 1570 data 169, 144,  76, 101,  27,  76, 130, 246
PF 1580 data 165, 186,  76,  14, 241, 133, 158, 132
LD 1590 data 151, 172,  25,  10, 204,  24,  10, 240
LD 1600 data   8, 177, 200, 133, 158, 238,  25,  10
IP 1610 data  24, 164, 151, 165, 158,  96
```

Z3PLUS

An extensive and versatile operating system enhancement for C128 CP/M mode

Review by M. Garamszeghy

Z3PLUS
$69.95 from:
Z Systems Associates
1435 Centre Street
Newton Centre, MA 02159-2469
(617) 965-3552

One of the most frequent complaints I hear about CP/M on the C128 is its lack of 'user friendliness', especially towards Commodore junkies who have never bothered to acquaint themselves with other computer systems. Ask what would constitute a user-friendly system, and you are likely to get as many different responses as people you ask. This seems to indicate that the ideal operating system should be customizable so that it can appeal to diverse tastes. Z3PLUS is such a system.

Z3PLUS, or the "Z System" as it is otherwise known, has evolved considerably over the years since it made its debut as ZCPR, almost at the dawn of CP/M computing. Versions exist for almost every Z80 CP/M system around, the latest release running under CP/M 3.0 or CP/M Plus, which just happens to be the CP/M used by the C128 as well as a few other less important (to me, anyway) computers.

What is Z3PLUS?

Z3PLUS is essentially an enhanced replacement command processor for the standard CP/M CCP.COM operating environment. It is a user interface that provides features such as named directories (which can be named across drives and user areas), extensive command line editing, keyboard macros and enhanced batch file processing.

The system comes complete with a number of operating system shells of varying sophistication that allow you to perform routine housekeeping functions such as running programs and copying files from a point-and-shoot type menu. You can still run virtually all standard CP/M programs when using the Z System, as well as many Z System-specific utilities.

Z3PLUS comprises the main operating module (Z3PLUS.COM) and a number of transient command and utility programs. The commands are broken down into three segments:

- the FCP (Flow Command Package), which is used to decide branching and conditional execution in batch file type processing (such as IF and ELSE);
- the RCP (Resident Command Package), containing general commands (such as ECHO and CLS);
- the CPR (Command PRocessor), which contains system commands (like GET, GO and JUMP).

The Z System is customizable in a number of ways. The first level of customization involves which commands you decide to include with your system.

The 'stock' Z3PLUS system includes a wide variety of options and commands in each of the three command types outlined above, such as CLS (clear screen); ECHO (print message to screen); POKE (for changing system memory); IF, AND, OR and ELSE (for conditional batch file execution); GET (load a file); GO, JUMP (execute a previously loaded file); etc.

Any or all of these commands can be included in your personal command library. Obviously, the more commands you make resident, the more memory will be required by system overheads.

By using GET and GO separately, you can load and run programs in areas other than the default start of TPA, providing, of course, that the files were assembled with the non-standard start address in mind. This allows you to have more than one program in memory at once by having each located in a different area of RAM. (In fact, most of the Z System shells and utilities work in this fashion.)

An interesting point is that GET is not restricted to loading program (COM) files, and can even be used to 'load' text files. Of course, you will not be able to execute the text file, but you can bring it into memory if you wish.

The second level of customization involves the use of 'aliases' and script files (an 'alias' is defined in the manual as a "single word or command that stands for a longer or compound command"). The alias allows you to set up custom names for your favourite command sequences.

Script files are more extensive and interactive than aliases, and can be combined into libraries containing some very sophisticated custom menu routines. You write them yourself and can, therefore, include whatever you wish in them.

Of named directories

One of the many interesting features of Z3PLUS is its use of the CP/M user areas as named directories. This can help people to organize large disks into smaller areas associated with easy to remember labels.

For example, with the EDITNDR you can define user area 15 on drive M as the 'SYSTEM' directory. Now when you log onto user 15 of drive M:, the prompt will display the name of the directory 'SYSTEM' in addition to the usual CP/M 'M15' prompt.

When in the Z System, and from within most of its utilities, you can change to the named directory area by simply specifying the directory label without having to remember the exact drive code and user area. The named directory list can also be saved (using SAVENDR) for future use.

Z3PLUS also provides for password protection of files and directories.

The tools and utilities

Most of the utilities provided on the distribution version of Z3PLUS are public domain. (This does not mean, however, that you get the same old tired programs that you probably already have several copies of in your library. They have been put into the public domain by their various authors to the benefit of all Z System users.)

The major ones, such as the operating system shells EASE and ZFILER, have been specifically written to run in the Z3PLUS environment, so would not do too well without it. (They are public domain in the sense that you are free to copy and use them as you see fit. The Z3PLUS.COM main system modules are *not* public domain, however.)

EASE stands for 'Error And Shell Editor'. A 'shell' can be loosely defined as a user interface that provides some degree of simplification for accessing operating system features. In addition to providing a powerful command line editor (the command codes are basically compatible with WordStar), EASE also provides a 'history' file of previously executed commands in sequence that can be easily retrieved, edited and re-executed.

ZFILER is the second operating shell provided with Z3PLUS. It is basically a point-and-shoot menu-driven file management program that does things like batch copying, running other programs, etc. Like the other Z3PLUS utilities, it is clean and very easy to use.

(One interesting feature about the Z System is that it allows you to use multiple levels of shells. If you first activated the EASE shell, then went into ZFILER, you would go back to EASE when you exited ZFILER. You then exit EASE to get back to the Z3PLUS system.)

ZPATCH is a hexadecimal file editor. It is easier to use than the patching modes of a debugger such as SID because it provides a full-screen editor that works in both HEX and ASCII modes.

SALIAS is a mini text editor used for editing and creating alias script files that uses WordStar-type control code commands for editing and cursor movement.

In addition, SALIAS can be used for other general editing of short text notes as well.

ARUNZ is an alias library manager of sorts. It allows you to combine many single alias script files into one large one, thus saving on disk overhead space (one large file can take up significantly less disk space than many small ones due to the CP/M disk allocation unit size of 1 or 2 kilobytes on the C128). When you use ARUNZ, you specify the name of the alias 'module' you wish to run, and ARUNZ will extract it from the alias library file (ALIAS.CMD), then execute it.

The documentation

If I could say but one thing to the first time Z3PLUS user, it would be: read the manual, front to back, in that order, and do not skip anything. The manual, like the Z3PLUS system, was written primarily by a physicist at MIT. (This person is so logical he would make Mr. Spock green(er) with envy, if he were capable of such emotion.) The manual was written to be read in consecutive order.

(As a physicist, he should be familiar with the concept of Brownian motion, which is how I think most people, myself included, tend to read software manuals - randomly taking bits here and there. I made the mistake of skipping a chapter in the middle and was confused for quite some time until I realized that the chapter I had missed contained some vital information that I needed.)

Once you convince yourself that reading the manual is required, initial set-up of the Z3PLUS operating system is quite simple and straightforward. You define your terminal capabilities (Saints be praised, the terminal type selection menu even includes an entry for the C128!) and rename a couple of files (this is the less obvious part that killed me before I read the manual in detail). Type in the magic word Z3PLUS and away you go.

The documentation itself is clear enough, although somewhat lacking when it comes to details. For example, in the section dealing with perhaps the most important utility, ZFILER (the general file handling, copying etc. utility), the part describing the command options merely tells you to look at the menu listing on the screen. I think that at least a command summary could have been expected.

(To their credit, however, a more detailed technical reference manual can be had, at extra cost. A bibliography of suggested further reading is also supplied for those who may be interested.)

To get around the problem of having to read the manual front to back, I would suggest better cross-referencing among the sections, especially between sections that contain vital information required to get a given utility to work.

The Z System is also supported by a network of BBSs (referred to as 'Z-Nodes'), which supply up to date techincal info and help as well as providing a convenient method to distribute new programs written for the Z System. A list of Z-Nodes is included on the Z3PLUS disk.

Final impressions

CP/M is a disk-intensive operating system. Z3PLUS is perhaps even more so because of its reliance on transient commands and script batch files. Because of this, a fast drive is imperative (*don't* try it with a 1541, you will probably die of old age) and a RAM disk is even better.

(An interesting combination is a 64k Quick Brown Box battery-backed RAM cartridge with the QDisk CP/M driver software (reviewed in a recent issue of *Twin Cities-128*). With this you can load most of the Z3PLUS main files and utilities into a non-volatile RAM disk and have them available as soon as you start up CP/M each time without having to copy them into the 1750 RAM disk.)

When I first started up my copy of Z3PLUS, I thought, "another semi-useful product". However, as I used it more, and discovered more of its features, I found myself liking it more and more and consequently using it more and more.

It sort of grows on you. Although $69.95 may seem like a fair bit to spend on an operating system enhancement, it is well worth it if you are seriously into C128 CP/M. What you get is an easily expandable and customizable operating environment that can be as powerful as you want to make it.

JiffyDOS for the C64/C128

"Look, Ma - no cables!"

Hardware review by Noel Nyman

JiffyDOS is available for C64, C64-C, SX64, C128, C128-D and 1541/1541-C/1541-II, 1571, 1581, FSD, MSD, Excelerator +, Excel 2001, Enhancer 2000

C64 series and one drive - $49.95
C128 series and one drive - $59.95
additional drive ROMs - $24.95
all prices plus shipping, US dollars

Creative Micro Designs, P.O. Box 789,
Wilbraham, MA 01095, (413) 525-0023

Specify computer and disk drive models when ordering

My first encounter with hardware to speed up my C64/1541 combination was 1541 FLASH. It was incredibly fast compared to stock machines. Block reads with "Disk Doctor" were on the screen almost before you could release the RETURN key. It also sported an extra cable between the drive and the Datassette port. You could put that plug in upside down. I found that out the hard way. You could break the wires off the plug (found that out the hard way, too).

FLASH permanently replaced the computer and disk drive ROMs (Read Only Memories), and worked only with the 1541. It was supposed to be compatible with everything. But the 'newest' copy protection systems used 1541 ROM codes, and wouldn't work with FLASH.

That was several years ago, and I'm sure FLASH has improved. It, and many similar products, still require an extra cable between the computer and disk drives. A corollary of Murphy's Law says that the cable supplied will always be just inches short of what's needed to locate your equipment where you want it.

A product that does not require extra wiring is JiffyDOS from Creative Micro Designs. The system uses the standard serial bus cable for all data transfers.

JiffyDOS replaces ROMs in the computer and disk drives. I tested it on a C64 (ROM-3) with two 1541 disk drives. Both drives were equipped with JiffyDOS ROMs, although that's not necessary. The system will work at normal speed with any additional drives that are not upgraded.

Unlike some cartridge-based products, ROM replacements speed up SAVE and "block access", as well as LOAD. JiffyDOS LOADs files about nine times faster than a standard system. SAVEs are about three times faster.

JiffyDOS works at this faster speed with all types of files, and with "block accesses" as well. Programs such as SuperBase may LOAD rapidly with many other products. But, they operate at normal 'slow' speed because they rely heavily on sequential or relative files. JiffyDOS improves the drive performance on any SEQ, REL, or USR file. Direct block access was also about three times faster in my tests.

JiffyDOS uses the standard Commodore DOS format to save files. It changes the 'interleave' (the number of disk sectors skipped between consecutive sectors of a file) to six. Commodore uses an interleave of ten. This makes for faster loads of files SAVEd with JiffyDOS, when JiffyDOS is used. Standard DOS can still read these files too, but a bit more slowly then normal.

One disadvantage of ROM replacement is that you must disassemble your computer and disk drive to make the installation. Creative Micro tries to make this as painless as possible. They provide six pages of step-by-step instructions for the computer, and seven pages for the disk drive. There are clear drawings of the various circuit board versions, with the location of the ROM to be removed, and similar drawings showing the JiffyDOS ROM orientation. The instructions are easy to follow, and have enough cautions and comments to keep even a novice from running into difficulties.

I had a minor 'problem' reading a special note for 64C owners. It refers to the ROM for the "older C64 boards" as having 24 pins, while the correct ROM for newer 64Cs has 28 pins.

I have a C64, one of the older boards. But, the ROM I received has 28 pins. The ROM is mounted on a small circuit board. The board has 24 pins on it, which fit into the Kernal ROM socket on the C64 board. The note apparently refers to the number of

JiffyDOS improves the drive performance on any SEQ, REL, or USR file. Direct block access was also about three times faster in my tests...

pins on the circuit board, not on the ROM chip itself. (Creative Micro says that a new version of the instructions makes this clear.)

Which brings up the other disadvantage of ROM replacements. If you have an older C64, your Kernal ROM may not be in a socket. To install JiffyDOS, you'll have to unsolder the ROM from the circuit board. This is not a job to be taken on lightly. If you don't have experience with unsoldering integrated circuits, you should enlist the aid of a professional. Any competent computer tech should be able to remove your Kernal ROM and install a low profile socket in its place for a few dollars. Many C64s, and all 64Cs and C128s have the Kernal ROM socketed rather than soldered in place.

Most 1541 ROMs are socketed. A few rare exceptions have ROMs mounted on 'piggyback' boards. Although these can be unsoldered, the JiffyDOS ROM and socket mounted on the piggyback board will sit too high to clear the top cover. If you encounter this problem, Creative Micro gives you the option of a free special replacement board.

A wedge, and more

JiffyDOS adds several features besides faster disk access. The usual 'wedge' commands are available, with the usual syntax.

/filename loads a BASIC program. **%filename** will do the same for a machine language file. **@$** displays a disk directory, **@S0:filename** will scratch a file, etc. The > symbol can be used in place of the **@**.

JiffyDOS also defines the eight function keys with commonly used wedge commands, and RUN and LIST. **@F** toggles these definitions on and off.

'filename verifies a file against memory. **@U** will 'un-new' a BASIC program. **@D:filename** lists a BASIC program to the screen without disturbing memory. The listing can be paused by pressing any key. The listing can be redirected to a printer with OPEN4,4: CMD4.

@T:filename will display or 'type' sequential files on the screen, again without disturbing memory. Pressing any key will pause the display. CMD will redirect the output to a printer or disk drive. You can use **@T** to copy a sequential file to another disk drive, although "READY." will be appended to the end of the copy.

CONTROL-P will print the current low resolution text screen on your printer... sometimes. The printer must be device #4, and either a Commodore printer or a good emulation. The command worked fine in direct mode.

I hoped to get hard copy of screens from databases and spreadsheets. But, CONTROL-P didn't work from inside most programs. Occasionally one of the public domain "Disk Doctors" printed, but only in upper case/graphics mode, although the screen was upper/lower case.

@N0:disk,id formats a disk in about 20 seconds - not as fast as some systems. But the documentation claims that all normal error checking is maintained. **@N2:disk,id** formats both sides of a disk for 1571 drives in 1541 mode. This facilitates using both heads when working with a C64/1571.

@B toggles 'head bumping' on the 1541. With bumping off, disk read errors will not cause the obnoxious misaligning rattle. Some software may send its own code to the drive which turns the bump back on. In that case, two **@B** commands are needed to turn bumping off again. More on this in a moment.

@Q disables the wedge and function key commands. Fast disk access routines are still in place. A SYS to an address in ROM will re-enable the functions that **@Q** kills.

Wedge commands can be used in BASIC programs. They can be chained, several commands on one line.

```
@"#9": @"S0:test*": @"#8"
```

This can be done in program or direct mode. Note that the quotes are required, and an **@$** in the chain will cause the remaining commands to be skipped.

The wedge commands will accept string variables, but only in program mode. Numeric variables can be used for some parameters, such as disk drive numbers, in either mode.

Compatibility and copyrights

Creative Micro claims that JiffyDOS is completely compatible with all commercial hardware and software. They guarantee it for 30 days from purchase. If you find something that won't work, you can return JiffyDOS for a full refund.

Obviously, a replacement with all these features changes the Kernal ROM code substantially. As usual, the extensive Datassette routines are replaced with the new code. That alone would make the ROM incompatible with one piece of "commercial hardware" - the Datassette.

Some products avoid this problem by providing a board with two sockets - one for the new ROM and one for your old Kernal

ROM. A switch selects one or the other; hence, full compatibility. If something won't work, just throw the switch.

JiffyDOS does this one better. The small circuit board with its 24 pins holds only one ROM. It does have a toggle switch soldered to it, on about a foot of wire. You mount this switch in a hole you drill in the plastic case. The installation instructions suggest places where the switch won't be in the way of internal workings. Switches are connected to the ROMs for disk drives as well.

The switch selects one of two 8K banks of memory in the ROM. One is JiffyDOS. The other is supposed to be fully compatible with your old computer ROM. When I threw the switch and reset the computer, I was greeted with the familiar sign-on message - the exact same message.

Curious, I checked the 'stock' ROM code against the original Kernal ROM. Not only are they "compatible", they're byte-for-byte identical! This makes for a curious situation regarding Commodore's copyright on the ROM code. It does ensure that the user has full compatibility. It also gives you a ROM upgrade in case you have an older (ROM-1 or ROM-2) C64.

The "compatibility" ROM is identical to the Kernal ROM-3. This insures full compatibility and upgrades a ROM-1 or ROM-2 C64...

If you need to disable JiffyDOS on the computer and several disk drives, you'll have to throw a switch on each. This could be a bother if you have several programs requiring the change. You can make it easier by mounting the drive switches on the front panels, or under the front bezel on 1571's.

Compatibility and RAM

The manual says that JiffyDOS "does not use any extra RAM (Random Access Memory) in your computer". Well... almost. It's hard to toggle features without using some memory to remember which state the toggle is in. If the add-on hardware has no RAM, it must borrow some from the computer.

JiffyDOS has only ROM. So, some memory locations are used. The designers minimized this impact by using locations that are uncommon to most software routines.

Locations 674 ($02A2) and 675 ($02A3) are used by the stock Kernal to save CIA (Complex Interface Adapter) control registers during Datassette I/O. Since JiffyDOS doesn't use the Datassette, it uses these addresses as toggles.

Address 674 holds the function key toggle. A non-zero value turns off the pre-defined function keys.

JiffyDOS toggles the value at address 675 between 5 and 133 whenever **@B** is pressed. The value is then sent to disk drive address 106 ($6A). This address controls the number of read attempts when a disk error is encountered. A 5 causes the normal activity, complete with head bump. A 133 bypasses the bumping part (the high bit is set...133 = 128 + 5). This is a 'traditional' method of eliminating head bump. But, a drive reset defeats it. So, some software may still cause head bumps.

JiffyDOS changes several of the vectors in the 768-779 ($0300 - $030B) range. **@Q** resets them to stock values. BASIC add-on utilities and other programs also change these vectors, to point at themselves. A well-written program will save the vector it replaces, and jump to it when done. But, not all programs are well written. Many programmers will assume the stock values and jump directly to them. This will bypass the JiffyDOS commands.

I was pleasantly surprised to find that JiffyDOS does not use location 186 ($BA) to determine which drive to access for wedge commands. Location 186 holds the current device number, actually the last device accessed. If you just printed something on the printer, location 186 will have a value of 4.

Many add-ons, such as Fast Load, and the Datel Mark-IV cartridge, use location 186 to decide which disk drive to access. If you tell the Mark-IV to display a disk directory after printing on the printer, it vainly tries to show you a directory from device #4.

JiffyDOS is smarter. It keeps its own active drive number, the one you set with **@#**. It stores it at location 787 ($0313). This location, marked "unused" on memory maps, sits between the USR (user routine) and the IRQ (hardware interrupt) vectors. It's only one byte, and not in zero page. So, most programmers don't use it. But JiffyDOS does, and I do.

JiffyDOS also knows the legal disk device numbers. I could change between disk drives by POKEing an 8 or 9 to address 787. But any other value was changed to 8 by the next disk access. Since the system works with more than two drives, I assume that values of 10 and up would be accepted if those devices were installed in the system.

If you use address 787 in your own programs, be aware that JiffyDOS may change the value for you. That can be a feature. To tell from program mode if JiffyDOS is active, store 255 at location 787, issue a disk command, and see if location 787 contains an 8.

Summary

JiffyDOS is a good compromise between maximum fast loading and compatibility.

You can use any software or hardware. Your cartridge, Datassette, and user ports are free. You can add a disk drive or use part of your system with other non-JiffyDOS equipment without difficulty. There are no extra wires to bother with, and nothing to forget to plug in.

JiffyDOS supports many non-Commodore drives. It may be your only choice for a speed up system if you use another manufacturer's drive, or mix 1541s and 1571s with the same computer.

JiffyDOS worked with all the software and add-ons I tested, including some surprises. The Datel Mark-IV cartridge worked normally with JiffyDOS active. I loaded a "Warp*25" version of Disk Maintenance in seven seconds with the Mark-IV. Loading the standard program with JiffyDOS took 45 seconds. Disk Maintenance has its own software fast loader, which probably deactivated the JiffyDOS routines. Once running Disk Maintenance, however, JiffyDOS read the blocks from the disk three times faster than with the Mark-IV alone. For ease of use, with some helpful features added, JiffyDOS is a good value.

Here at the Transactor offices we have received JiffyDOS for the C128 and 1571. This product works in 64 mode as well as 128 mode. The instructions were very clear and well-illustrated. Installation was simple and the system works well. In our case, the drive instructions amounted to six pages (the 1541 has been through several revisions and therefore requires seven pages).

JiffyDOS allows 'power on' ROM switching. (Crashing or hanging up is possible; response varies with the program.) Do not switch during a disk access!

On 1571 and 1581 drives, the drives sense whether the computer is in stock or JiffyDOS mode and select the correct routines automatically. JiffyDOS speeds up 1571 and 1581 drives (though not as dramatically as it does the 1541).

Speed Comparison: The chart below is from the JiffyDOS manual and is based on results obtained using ML routines. They do not take into account spin-up delay (.5 sec.) or directory searching time. Other factors may also influence the results that you obtain on your system. ■

Speed Comparison Chart
C64, SX-64, 64 mode

	1541		1571		1581	
Load 202-block PRG file	124	12	124	9	102	8
Save 100-block PRG file	75	24	75	20	40	15
Read 125-block SEQ or USR file	84	15	84	13	63	9
Write 100-block SEQ or USR file	81	27	81	24	44	17
Read 64 154-byte REL records	40	14	40	14	37	10
Write one 154-byte REL record	.350	.125	.350	.120	.325	.110
Read/write 16K on command channel	47	9	47	9	47	9

C128 in 128 mode

	1541		1571		1581	
Load 202-block PRG file	124	12	14	9	12	8
Save 100-block PRG file	75	24	48	25	26	14
Read 125-block SEQ or USR file	84	15	31	12	20	10
Write 100-block SEQ or USR file	81	27	48	33	20	11
Read 64 154-byte REL records	40	14	21	14	17	10
Autoboot 202-block program	125	13	54	10	13	9
Read/write 16K on command channel	47	10	10	6	10	6

SWL

Short-wave decoding for the C64 (and VIC-20)

Hardware review by Noel Nyman

SWL cartridge, available for VIC-20, C64, and C128
$64 US

G&G Electronics
8524 Dakota Dr.
Gaithersburg MD 20877
USA

(301) 258-7373

The SWL cartridge, from G&G Electronics, has been advertised in Commodore-oriented magazines for several years, promising "Worldwide Short-wave Radio Signals on Your Computer."

"Remember the fun of tuning in all those foreign broadcast stations?" You bet I do! I once had a WWII Hallicrafters aircraft receiver, modified for short-wave use. The ad explains that all those "beeps and squeals" you hear in the short-wave bands are digital data. The SWL cartridge will decode them for you. "You'll see the actual text [on your] video screen."

The cartridge plugs into the computer expansion port. It comes with a hook-up cable, a demo cassette, and a manual that explains "how to get the most out of short-wave digital DXing, even if you're brand new at it." DXing is short for Distance Receiving. SWL is an acronym for Short-Wave Listening.

There are several microprocessor based products that decode various sorts of short-wave code. The SWL cartridge, at $64 US, is by far the least expensive. That's because you supply the microprocessor, a C64 (or a C128 in C64 mode). A different model of the SWL is available for the VIC-20.

All the decoders operate on the audio output of a short-wave receiver. They use a circuit called a PLL (Phase Locked Loop) to 'lock in' on a narrow band of audio frequencies. The audio signal is then converted into digital output. A ROM (Read Only Memory) in the cartridge supplies the program that tells the computer how to use the digital output from the PLL.

The cable supplied with the cartridge connects a miniature phone jack on the cartridge to your receiver's headphone jack. If your receiver uses a full size phone jack, you'll need an adapter. A second miniature jack on the cartridge can be used for headphones or a speaker to monitor the signal.

A third jack is provided for connecting a key (the telegraph type), so you can practise your code sending skills with the cartridge. A slide switch is also used to select wide or narrow bandwidth for certain types of signals.

The demo tape contains a long message in Morse code. You play the tape on any cassette player. The headphone output from the player is fed into the cartridge. By monitoring the sound, you can get a feel for the volume and pitch that work best.

The cartridge performed flawlessly with the demo tape.

I connected it to my inexpensive multi-band radio, ran a long piece of wire around the room, and proceeded to look for signals to decode. None that I found was loud enough to get even a glimmer of recognition from the cartridge.

I decided that my receiver simply wasn't up to the task. So I contacted my friend John, who is interested in DXing. He loaned me a Kenwood receiver with digital tuning, sideband switches, adjustable filters, and many other bells and whistles. This was a far cry from that ancient Hallicrafters!

I easily heard hundreds of signals. I also heard incredible amounts of QRM (radio interference). I patiently adjusted, filtered, and tweaked on signals, trying to get the cartridge to respond to them.

SWL provides an on-screen tuning indicator which flashes when the signal fed to the cartridge is recognized by the PLL. An audio tone is also produced in the monitor speaker. Without these tuning aids, getting the audio just right would be impossible. Even with them, it's quite a challenge.

Morse code is sent as CW (Continuous Wave). A circuit called a BFO (Beat Frequency Oscillator) in your receiver creates the audio 'dots' and 'dashes' from a CW signal. The BFO allows you to vary the pitch of the audio. The pitch also varies if the signal's radio frequency drifts. The drift can occur in either the transmitter, your receiver, or both.

The PLL circuit in the cartridge requires the audio input to be very near a specific frequency. You must adjust your receiver to produce audio at that frequency. On the Kenwood, several knobs affected the audio pitch. I found the volume was also important.

Morse code can be sent at a variety of speeds, measured in WPM (Words Per Minute). The cartridge can adjust automatically to changing speed, but only over a limited range. You set the initial speed, and all other cartridge functions, using CTRL key combinations on the keyboard. So you must guess at the speed of a signal and set the cartridge, then adjust for the right pitch and volume. Variations in the signal strength and any frequency drift will cause pitch and volume changes. The controls on your receiver require frequent adjusting to compensate. Too much interference will swamp the cartridge. It won't be able to find the received signal amongst the garbage.

I worked with the SWL and the Kenwood for three evenings. My net result was a partial message which read "I am a retired airline pilot." I determined that most of the QRM was coming from the C64 itself. Some shielding was in order. A better antenna system was needed too.

I explained the problems to John, who put me in touch with Bill. Bill's hobby is DXing. He has three Commodore computers. But he's not really a 'computer person.' Instead, they only serve as aids to his many receivers, scanners, and other specialized listening gear.

Bill was interested in the SWL cartridge, and offered to help me test it. But he didn't expect much from such an inexpensive product. He uses a decoder made by Info-Tech. It's a large black box bristling with switches, and cost him several hundred dollars.

So, I visited Bill in his listening post. He's solved the computer generated QRM problem by using large ferrite traps threaded around the equipment power cords. He also uses shielded cable to feed signals from his sophisticated antennas.

We tested the SWL cartridge by connecting it and Bill's Info-Tech to the audio output of his receiver. Both units got the same audio signal. The Info-Tech has its own microprocessor and connects to a video screen directly. Both devices can decode Morse and RTTY (Radio Teletype) signals. The Info-Tech can deal with several additional types, including packet radio (computer data sent by radio instead of phone lines).

We found that the SWL cartridge and the Info-Tech did equally well with Morse code. Both devices displayed the same text consistently. Bill's receiver has more filtering than the Kenwood, which helped eliminate static and other signal interference. The SWL cartridge also did well with RTTY signals. More set-up is required since there are many more variations in RTTY than Morse transmissions. It was difficult to gauge the SWL's performance against the Info-Tech on RTTY, because they require audio at different frequencies in this mode. So, the two devices could not decode the signals simultaneously.

Bill was quite surprised at the performance of the G&G Electonics cartridge. It did as well as the much higher priced Info-Tech, for the signals it was designed to decode.

But this device is not for the casual user. The cartridge will not work at all with an inexpensive short-wave receiver. You must have a good radio that will let you adjust the audio to the range that the SWL can handle. You must also have a good knowledge of what the signals sound like, and what adjustments to make from the keyboard to decode them. Without Bill's expertise, I would have wasted most of the evening on inappropriate "beeps and squeals".

You'll need good shielding on the computer also. The computer must be within reach of the radio for proper operation, since you'll need to make adjustments on both often. The computer and monitor must not create any radio interference. You'll need clean signals to get proper cartridge operation.

The cartridge is an inexpensive way for the dedicated DX'er with a C64 to add on-line automatic decoding. It is *not* appropriate for a computer owner who's just getting started in the exciting hobby of short-wave listening.

The ZR2 Hardware Interfacing Chip

Control functions via the user port

Hardware review by Noel Nyman

ZR2 - 40 pin DIP hardware interface integrated circuit chip
$29.95 plus $1.55 shipping (US)

ALX Digital
12265 S. Dixie Hwy #922
Miami FL 33156

a disk with BASIC routines to control the ZR2 connected to a Commodore user port is available for $5

Commodore eight-bit computers have an 'open architecture', with all control and data signals brought to the outside world. The VIC-20 and C64 also provide a user port with eight bi-directional lines easily controlled by BASIC software. Their low cost makes them ideal for hardware control systems.

But, the time saved by using these computers is often lost again in building the hardware interfaces you need to make computer signals operate real-world devices.

A product that attempts to make interfacing easier is the ZR2, from ALX Digital. This forty-pin DIP (Dual Inline Package) chip provides several programmable functions. The functions can be loosely grouped as: one-of-X outputs, pulse counter, serial functions, dimmers, and specialized display.

Hardware requirements

The ZR2 has eleven inputs and sixteen outputs. The only additional parts required are pull-up resistors, a capacitor and a crystal (see Figure 1). ALX recommends using buffers on the ZR2 outputs, and specifies 74240's. These are TTL (Transistor-Transistor Logic) tri-state gated buffer packages. Since they invert the ZR2 signals, I used the similar 74244. It is pin-for-pin compatible, but provides un-inverted buffering.

The ZR2 requires about 100ma (milli-amperes) at five volts DC. A C64 with a power supply in top condition might be able to provide 100ma. But I strongly suggest a separate power supply for the ZR2 and the circuits it drives. Be sure to connect computer ground to ZR2 circuit ground.

A crystal frequency between 1MHz and 11MHz will work for most functions. A 4MHz crystal is specified for AC dimming at 60Hz. If you develop a DC dimming or serial transfer application, then change the crystal frequency, you may have to adjust your software to compensate.

The eleven inputs fall into three groups: eight data inputs, two 'logic points', and a reset line. For most applications, one or both logic points are required. The reset signal is necessary if you want to change functions under software control. If you use the Commodore user port, that leaves only five lines for data inputs. This is enough to select all of the ZR2's functions. But, some functions accept parameters using all eight data inputs. With the user port application, you're limited in the range of these functions you can access.

Parallel decoders

The user port itself is a simple eight line decoder. By sending a value between 1 and 255 to the user port, its outputs can be 'turned on' in any combination. The outputs are 'latched' in this state; they don't change until another number is sent to the port.

If want to turn on output #5, your software will have to calculate the appropriate binary value to send to the port (32 in this case). If you want to turn on output #7 without turning off #5, you'll have more calculation to do (128+32=160). If you need more than eight outputs, you have a challenge.

The ZR2 provides some easy alternatives. In what's called the "matrix" mode, you have two eight line decoders. To activate this mode, you first place a value of five on the data bus and ground the reset line. This resets the ZR2 to the function specified by the number on the data bus. Next, place a zero on the data bus. In matrix mode, the zero value is a toggle. But in some modes, a fast zero is required, or the mode selection number may also be interpreted as the first data value.

Now send any value between 1 and 255 to the data bus. The corresponding outputs of "outport #1" will turn on. They will be latched, just as with the user port. Sending a zero to the data bus toggles the data bus to "outport #2". The next data value will turn on the appropriate lines on outport #2. New values sent to the ports will change the output lines in the same manner as the user port.

To zero an outport, you must ground a logic point line while sending data to the outport. This worked fine for me on outport #2. But, I was unable to zero outport #1. Also, with three of the Commodore user port lines connected to the logic points and reset, I was limited to controlling five lines on each outport.

An alternative decoder may be more useful in some circumstances. The one-of- sixteen decoder is selected by resetting the ZR2 with a six on the data bus. If you don't quickly follow this with a zero, the six will appear at the ZR2 output. Even at computer speeds, your real world devices might respond to this brief signal. I'll discuss a 'fix' for this later.

Now a value from one to eight will turn a corresponding line of outport #1 on. Note that this is a true one-of-eight decoder, where the "matrix" was a binary decoder. If you put a number from nine to sixteen on the data bus, a line on outport #2 will turn on. This is the "OR" mode...only one line on each outport will be on at any time. ALX has designed the ZR2 so the outports are somewhat independent in this mode. You can turn on lines in either port with just one number on the data bus. But, following an eight with a nine will leave the last line on outport #1 on and turn the first line of outport #2 on also.

If you have a need for many lines to be on at one time, you can enter the "AND" mode by grounding logic point #2. Now sending in sequence '1', '2', '3' to the data bus will cause the first three lines on outport #1 to come on. Sending a zero to the data bus clears the outputs in both modes.

Serial decoder

The serial decode function provides an interesting alternative, since fewer data lines are required to use it. In fact, if you set up the function using hardware (switches perhaps), you only need two signals. Data is sent on logic point #2. Up to eight pulses can be sent. A line will turn on at outport #1 representing the total number of pulses sent. The timing is moderately critical here. If the pulse widths and frequency aren't right, the decoding will be erratic, or not work at all. The exact timing will depend on the crystal frequency you use.

Outputs are "ANDed". So, sending '1', '2', and '3' in succession will cause all three of those lines to turn on. You can control outport #2 independently by grounding logic point #1 before sending pulses. A nine sent to either outport will zero the outputs.

An interesting possibility here is that the pulses to logic point #2 don't have to come from the computer. You can use prerecorded pulses from tape, or clocked pulses from a ROM, or from another ZR2. Using tape, you could set the device up entirely in hardware with switches... no computer required.

ALX has also implemented its own proprietary serial transmission system, using two ZR2's. The first ZR2 receives a parallel eight byte word, and generates serial pulses on outport #1 line #1. A parallel copy of the word appears on outport #2 for verification. These pulses are fed to logic point #2 of the second ZR2, set for serial input.

In this mode, the receiving ZR2 displays the received data on outport #1 in binary form. The data is latched until a new word is received. New words replace old ones; no "AND" mode. A zero is a "start of transmission" signal to the first ZR2, and is not sent as data. Not being able to transmit a zero value makes this serial system useless for sending program or other data. But it can be used to send one-of-eight input control signals over long distances. There is a decoding delay, which increases with the value of the number sent.

Pulse counter

Pulse counting mode is similar to serial decoding. However, in this mode, new pulses are merely added to the existing count. Both outports are used together. So, numbers up to 65,535 can be "displayed". The lines change with each pulse received. This makes for an interesting display. But, any devices connected to these lines will get momentary pulses as you send new numbers.

DC dimmer

Up to eight separate lines can be selected for dimming via outport #1. The lines start with no output. Grounding logic point #2 causes pulses to be sent to the selected lines. The pulses increase in width, based on the crystal frequency. The observed effect on LEDs connected to the lines is that they gradually come up to full brightness. After briefly grounding logic point #1, logic point #2 is used to dim the LEDs again. Bringing logic point #2 high during the process holds the LEDs at their brightness level. So a slower speed can be implemented by pulsing logic point #2.

Of course, other devices can be used in place of LEDs. I tried a small DC motor with fair results (be sure to use a back-biased diode to prevent reverse voltage spikes entering the integrated circuits). But any robotics usefulness of this mode is eliminated because you *must* come up to full "brightness" before "dimming" again. You can go from any brightness level to zero, a feature not mentioned in the ZR2 documentation.

AC dimmer

To utilize AC dimming, you need a 4MHz crystal driving the ZR2, and some additional parts. AC dimming works differently from DC dimming. Not only can you 'dim' before reaching full brightness, you can't avoid it. Bringing the logic point high to halt the process also toggles the direction. To brighten lights to a particular level, stop, then brighten again, you must send two ground pulses to the logic point. The first toggles dimming, the second switches back to brightening. Since we're dealing with AC devices, this won't be a problem for the outputs. But it means a more device intensive software routine.

Only one output can be controlled in AC dimming. However, the dim rate is selectable over a reasonably wide range by placing a number on the data bus.

Specialized displays

My first electronics construction projects were 'do nothing' boxes. We made them from neon lights, capacitors, and resistors (relaxation oscillators to you knowledgeable folks). The lights flashed in patterns, usually a circle.

Integrated circuits made 'do nothings' much more sophisticated. I wrote two articles for *Radio-Electronics* magazine on LED 'do nothing' boxes that used ROM's to produce a variety of displays.

Perhaps because of my past interest in doing nothing, I found the "chaser" routines most interesting. There are four separate displays that produce 'chasing' patterns on LEDs or other lights connected to the outports. The displays are speed controllable, via the data bus. They can also be operated in "pulse" mode through the logic points.

You can make a simple 'do nothing' that switches among the four displays. Or you could connect the outputs to Christmas tree lights or other displays for some interesting effects. Clever as they are, the chasers are not very useful for hardware control.

Should you buy one?

If you have the electronics expertise, you could build a hardware device to perform any one of these tasks for less than the cost of a ZR2. If you have only one particular project in mind, the ZR2 may be overkill.

If you like to experiment, or if you find your hardware needs changing periodically, the ZR2 may be a reasonable investment. You can certainly connect it in several different circuits more easily than you could construct equivalent hardware. It may even be cheaper to test systems using the ZR2 that you eventually build from discrete parts, if your development time is valuable.

There are some problems, many with the documentation. Some electronics knowledge on the user's part is assumed. In the first example (the chasers), the user is told to "bring the same pin [logic point #2] up to +5 volts." A few sentences later comes the caution "NEVER CONNECT ANY PIN OF THE ZR2 DIRECTLY TO +5!"

This apparent discrepency assumes that the user understands "+5 volts" as a slang terminology which actually means "logic one" in the TTL world. A TTL "logic one" is usually in the range of +3.4 volts. Some input lines can cause internal chip problems if connected to the higher +5 volts. The user must have a good idea when "+5 volts" in the manual really means "logic one."

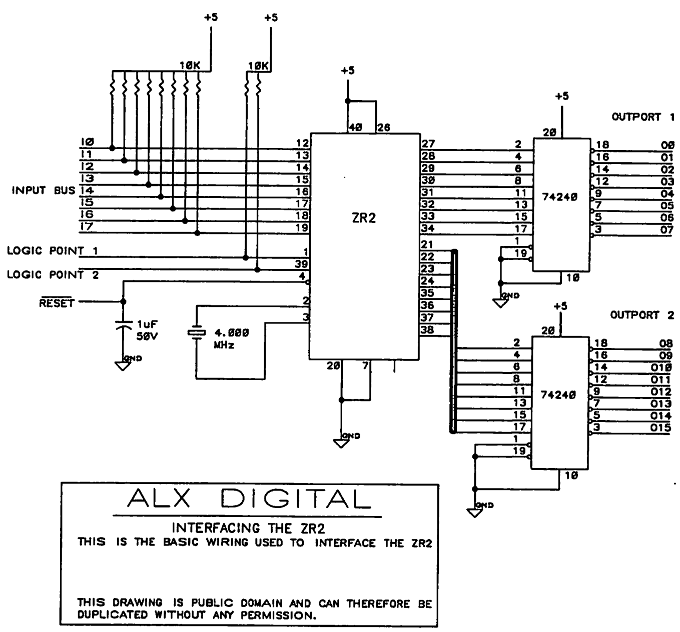
The first example also implies that the ZR2 must be started from a power off condition to change modes. In fact, a grounded reset line will switch modes on the ZR2. It will also force all outputs high briefly. This may be annoying for light displays. It could mean disaster for real world devices connected to those outputs. This and other design 'features' probably stem from ALX's background designing lighting control systems. Lights aren't as fussy about brief spurious signals as are integrated circuit controllers.

My solution was to add a one-shot circuit to the standard schematic provided by ALX. I connected the output of the one-shot to the gate pins on the buffers. The one-shot is triggered by the ZR2 reset signal to disable the buffer outputs. It's timed to hold the gates low until the ZR2 settles its outputs down to their desired state.

It's not always clear from the documentation what the state of the logic points should be. There are several unused pins on the chip. These should not have anything connected to them or internal damage may result. But no caution appears in the manual.

ALX has informed me that they are working on a revision of the documentation to correct some of these problems.

For experienced electronics experimenters, the ZR2 provides a cost effective way to quickly and easily experiment with new interface circuits.



NORTHWEST MUSIC CENTER INC.

539 N. Wolf Rd. — Wheeling, IL 60090
Hours— (Voice) Phone (312) 520-2540
Mon.-Thurs. 12:30-5:00, Sat. 12:00-4:00
(24 Hour Order Recorder)

We want to be your Commodore shop!

Do you feel like no one cares about you and your orphaned computer! NWM cares!! We still stock and sell most major B series and 8000 series programs.

Parts available for some 8000, 9000 and B series models!

Hacker's Corner

One of a Kind • Surplus • Monthly Special • Closeouts

Limited quantities to stock on hand

64 k ieee to parallel buffer	$199.00
4023p 100cps rehab	$99.00
64k ram exp 8032	$110.00
Smith Corona DM-200	$179.95
6400, 8300p, Diablo 630 ribbons	$4.95
8023p and MPP 1361 ribbons	$5.50
9090 7.5 meg rehab	$495.00
4023p ribbons	$6.00
Everex 2400 external modem	$245.00

Commodore's Superpet 9000

only $199.00

while supplies last

With Five Interpretive Languages:
Cobol
Pascal
Basic
Fortran
Apl

Runs 8032 software.
Great for schools and students

SUPER SPECIAL

64 K memory expansion for 8032

— with superbase 8096 both for only **$125.00**
— with superscript 8096 both for only **$125.00**

Superpet 9000 rehab **$150.00**
8050 disk drive new **$400.00**
8050 disk drive rehab **$200.00**

64K Memory Expansion for 8032 only $110 upgrades your 8032 to an 8096.

COMMODORE 8000-9000 SOFTWARE & MISC.

9000 Superpet	$199.00
64K exp for 8032	$110
OZZ Database	$25
BPI General Ledger	$25
BPI Accts Payable	$25
BPI Job Cost	$25
BPI Accts Receivable	$25
BPI Inventory	$25
Superscript 8032	$79
Superbase 8096	$79
Legal Time Acc	$25
Dow Jones Program	$25
Info Designs 8032 Accounting System	$50
Superoffice 8096	$149
Calc Result 8032	$89

B-128
$99 U.S.
rehab

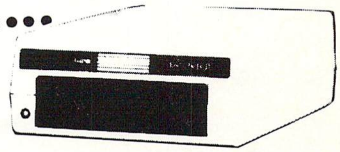
NEW 128K USER INSTALLABLE MEMORY EXPANSION! INTRODUCTORY PRICE OF ONLY $125.

SOFTWARE FOR THE B-128!!!

Superbase	$19.95
Superscript	$19.95
Superoffice Integrated	
Superbase & Superscript	$49.95
Calc Result	$89.95
Word Result	$89.95
Super Disk Doc	24.95
The Power of: Calc Result (Book)	$14.95

C.A.B.S. Accounting

General Ledger	$ 9.95
Accounts Receivable	$ 9.95
Accounts Payable	$ 9.95
Order Entry	$ 9.95
Payroll	$ 9.95
Buy all 5 for only $24.95	
Superbase: The Book	$14.95
Applied Calc Result (Book)	$14.95



SFD 1001 1 Megabyte
PRICED AT **$149.95** (US)
$125 with purchase of Superpet

SFD-1001 is the drive that you should consider when you need large amounts of data storage. It holds over 1 megabyte of data on its single floppy drive. Fast IEEE access for your C-64 or C-128. (C-64 and C-128 need an IEEE interface.) Why settle for slower drives with less storage capacity. This drive stores substantially more programs and data. Think how much money you can save on disk purchases. In fact, it stores almost 7 times more information than your standard drive. Bulletin board owners love them. And what an introductory price! At $169.95 these drives will sell fast, so don't wait. This drive has the identical format of a CBM 8250 drive, one of Commodores most durable floppy drives.

MODEL	SFD-1001
DRIVES	1
HEADS/DRIVE	2
STORAGE CAPACITY (Per Unit)	
Formatted	1.06 Mb
MAXIMUM (Each Drive)	
Sequential File	1.05 Mb
Relative File	1.04 Mb
Disk System	
Buffer RAM (Bytes)	4K
DISK FORMATS (Each Drive)	
Cylinders (Tracks)	(77)
Sector/Cylinder	—
Sector/Track	23-29
Bytes/Sector	256
Free Blocks	4133
TRANSFER RATES (Bytes/Sec)	
Internal	40 Kb
IEEE-488 Bus	1.2 Kb
ACCESS TIME (Milli-seconds)	
Track-to-track	*
Average track	**
Average Latency	100
Speed (RPM)	300

ORDER NOW WHILE STOCK LASTS!

Send or call your orders to: **Northwest Music Center, Inc. 539 N. Wolf Rd., Wheeling IL 60090. 312-520-2540** For prepaid orders add $25.50 for Superpet, $10.45 SFD 1001, $11.45 B-128, $10.45 4023p, $16.45 9090 and $5.45 64K memory expansion. For software add $3.50 for first and $2.00 for each additional book or program. Canadian shipping charges are double U.S. For C.O.D. orders add $2.20 per box shipped. All orders must be paid in U.S. funds. Include phone numbers with area codes. Do not use P.O. Box, only UPS shippable addresses. A 2 week hold will be imposed on all orders placed with a personal or business check. C.O.D. orders shipped in U.S. only and cash on delivery, no checks. 30 day warranty on all products from NWM, Inc. No manufacturer warranty. NWM reserves the right to limit quantities to stock on hand and adjust prices without notice!

All prices quoted in US dollars.