TC-128

TWIN CITIES 128 - ISSUE #32 - JULY 92

$3.95 MAGAZINE ONLY $9.95 MAGAZINE & DISK

Take a look at CMD's Sizzling Summer Specials

RAMLink Packages

- RAMLink, RAMCard II and RTC
- RAMLink Battery
- 1 or 4 MB SIMM Module

1 MB RAMLink package $275.00
4 MB RAMLink package $375.00

HD - RAMLink Packages

- RAMLink, RAMCard II 0 MB and RTC
- HD Series Hard Drive
- Parallel Cable • Shipping $25.00

HD-20 Pkg	$600	HD-40 Pkg	$775
HD-100 Pkg	$975	HD-200 Pkg	$1175

RAMDrive Specials

• RAMDrive 1 MB	$225.00
• RAMDrive 2 MB	$275.00

GEOS Combo Package $60.00

- Perfect Print Complete System
- geoMakeBoot • Shipping Included

SwiftLink Package

- SwiftLink -232 Cartridge
- DB 9 to DB 25 Modem Cable
- Terminal and File Xfer Software
- Shipping Charges Included

SwiftLink Package $45.00

Summer Specials will be offered through August 31, 1992. Quantities may be limited, contact CMD for availability. CMD reserves the right to change pricing if necessary.

perfect print LQ for GEOS™

Eliminates jagged output • Resolution up to 360 x 360 DPI (24 Pin), 240 x 216 DPI (9 Pin)
GEOCABLE compatible • Allows multiple copy printing • HQ drivers enhance graphic output

Perfect Print LQ is a complete print enhancement package for GEOS that delivers the highest quality dot matrix output possible. Includes a unique print utility and font set for enhancing GEOWRITE documents, utilities for creating fonts, and high quality drivers for other GEOS applications. Improves text and graphic output on virtually all 9 & 24 pin dot matrix printers and supports font attributes such as italics, outline, underline, bold, etc.

Main system (All drivers, utilities, and 7 fonts) **$34.95** • Font Package (42 LQ fonts) **$29.95**
Complete System (Main System & 49 fonts) **$49.95** • Shipping/Handling **$5.00** Canada add **$4.50**

RAMDrive™

- Compact - All the features you've ever wanted from RAM expansion in a compact unit. Five capacities: 512K, 1, 2, 4, and 8 megabytes. Dimensions: 6"l x 3"w x 1"h.
- Expandable - Now RDX Models allow for users to expand RAMDrive up to 2, 4, and 8 megabytes.
- Compatible - Use RAMDrive with GEOS, CP/M, Q-Link, BBS programs, productivity software and more.
- Non-Volatile - External power supply eliminates drain on computer power supply and retains data indefinitely. Internal rechargeable batteries retain data up to 7 days. RDX units use an optional external battery and also retain data for up to 7 days.
- FAST - Up to 400x faster than a 1541; 20x faster than RAMDOS; Built-in JiffyDOS speeds access to CMD Hard Drives and JiffyDOS-equipped floppy drives.
- Easy to Use - Plugs into the cartridge port. Operates like a standard disk drive. File and disk copiers included along with partitioning, support utilities, and new GEOS configure.
- RD-DOS - Organize RAM into as many as 30 manageable partitions that emulate 1541, 1571, & 1581 drives or expand to the full RAM capacity with MS-DOS style subdirectories. Autoboot 64 and 128 mode programs.

RD-512	$199.95	RD-1	~~$249.95~~	RD-2	~~$299.95~~
RDX-1	$264.95	RDX-2	$319.95	RDX-4	$399.95
RDX-8	$549.95	RDX External Battery unit			$24.95

HD Series HARD DRIVES

Special Limited-Time Price Savings

The Ultimate in Mass Storage for the 64/128

- Capacity - 20 Mb to 200 Mb capacities enable you to store the equivalent of up to 1250 1541 (170K) disks.
- Speed - The fastest Commodore compatible hard drives. Speeds up to 50x faster than a 1541.
- Compatibility - Ideal for use with GEOS, CP/M, Q'Link, BBS programs, productivity software and much more.
- Compact Size - 3 1/2" SCSI technology allows for a compact case about the same size as a 1581.
- Expandability - Chain up to six SCSI devices or connect to Macintosh, IBM-Compatible & Amiga computers.
- Built-in Real Time Clock - Automatically time and date stamps files and sets the GEOS clock.
- HD-DOS - Organize storage into as many as 254 partitions that emulate 1541, 1571, & 1581 drives or expand to 16Mb with MS-DOS style subdirectories.
- Easy to use - Connects like a standard drive and easy to read manual explains all facets of drive operation. Comes complete with copiers and maintainence utilities.

HD-20	CALL FOR PRICE	
HD-40	$599.95	
HD-100	~~$849.95~~	$799.95
HD-200	~~$1099.95~~	$999.95

RAMLink™

Power Backed REU Interface and Expandable RAM Disk

- **Non-Volatile Storage** - Operates on its own external power supply. Optional rechargeable battery back-up retains data even during power outages.
- **Compatible** - Use GEOS, CP/M, Q-Link, BBS programs, productivity software and more. RAM port for connection of REU or GEORAM of any capacity. Pass-thru port supports most cartridges.
- **User Expandable** - Internal RAMCard allows expansion up to 16 Mb by using standard SIMM's.
- **FAST** - Up to 400x faster than a 1541; 20x faster than RAMDOS; Built-in JiffyDOS plus parallel interface to speed access to CMD Hard Drives.
- **Easy to Use** - Plugs into the Cartridge Port. Operates like a standard disk drive. File and disk copiers included along with partitioning and GEOS support utilities.
- **RL-DOS** - Organize RAM into manageable partitions that emulate 1541, 1571, & 1581 drives or expand to the full RAM capacity with MS-DOS style subdirectories.

RAMLink (no RAMCard)	$179.95	Battery w/cable	$24.95
RAMLink (w/ RAMCardII)	$219.95	HD Parallel Cable	$14.95
RAMLink (RAMCardII/RTC)	$239.95	RTC add-on Kit	$29.95
RAMCard II (with RTC)	$79.95	1Mb SIMM	$42.00
RAMCard II (without RTC)	$59.95	4Mb SIMM	$145.00

JiffyDOS™

Hi-performance ROM upgrade • Performs all disk accesses up to 15 times faster
Guaranteed 100% compatible • Built-in DOS Wedge & file copier • Easy to install
(Please specify computer & drive model with serial#.)
JiffyDOS 64 or SX-64 $59.95
JiffyDOS 128 or 128D $69.95 • Add'l Drive ROMs $29.95

gateWay™

A Powerful New Desktop for GEOS 2.0
Task Switching • Three drive support
Access full capacity of HD, RAMLink, RAMDrive, expanded REU's & GEORAM
gateWay 64 $29.95 • gateWay 128 $29.95
gateWay 64/128 Combo $44.95 • Shipping: $5.00

SwiftLink™

Provides an industry-standard IBM style serial port that communicates at speeds from 300 to 38,400 baud and provides reliable 1200 and 2400 bps using RS-232 Hayes-compatible modems. Includes terminal programs and software for transferring files to other computers.
*Swif*Link (Cart.) $39.95 • *SWIFT*Link Cable $9.95
Shipping: US: $6.00 (Cart), $5.00 (Cable), $7.50 (Both)

SID Symphony

SID Symphony adds a second complete SID chip to your 64 or 128 and can be attached to your home stereo or amplified speaker systems to povide a whole new dimension in sound. Enjoy hundreds of public domain stereo songs, import MIDI files, or use Compute's Music System book and SID editor to create original stereo music.
SID Cartridge $39.95 • Compute's Music Book $22.95
Shipping: US: $6.00 (Cart.), $5.00 (Book), $7.50 (Both).

geoMakeBoot

geoMakeBoot makes booting GEOS from nearly all devices and making back-up copies of the GEOS boot disk easy and convenient. Besides being compatible with virtually all CBM compatible devices, geoMakeBoot also eliminates the need for installing some desk accessories and supports CMD Hard Drives, RAMLink and RAMDrive. geoMakeBoot is simple, inexpensive and easy to use. **geoMakeBoot $12.95 + $2.00 shipping**

Ordering Information and Shipping Charges

CMD Hard Drives:	Continental US: $25.00 per drive (UPS ground), $35.00 (2nd-Day), $45.00 (Next-Day). Canada: $50.00 (Airmail). COD to U.S. only $5.00 add'l charge. Foreign prices: Add $100.00 to U.S. Retail Price
JiffyDOS:	Add $5.50 per order (UPS ground), $10.00 (2nd-Day Air), plus $5.00 for APO, FPO, AK, HI, and Canada, or $15.00 for overseas orders. No add'l shipping if ordered with any hard drive. COD's add'l $5.00
RAMLink:	US: $12.00 (UPS), $20.00 (2nd day), COD add $5.00. Canada: $23.00. Foreign: CALL
RAMDrive:	US: $8.50 (UPS), $16.00 (2nd day), COD add $5.00, Canada $18.00, RDX Battery add $3, Foreign:call
Payment:	MA residents add 5% sales tax. We accept VISA, MasterCard, Money Orders, C.O.D., and personal checks (allow 3 weeks for personal checks to clear). Credit card orders provide the following: Card holders name, billing address, home/work phone, card number, expiration date and issuing bank.

Canadian Customers can now contact HOLZ COMPUTER SUPPLY, Box 47008, Dover P.O., 3525 - 26th Ave., SE, Calgary, Alberta, Canada T2B 3B7 Phone: 403-272-1888 Fax: 403-272-2010

• WE VERIFY ALL CREDIT CARD INFORMATION AND PROSECUTE INDIVIDUALS ATTEMPTING TO PERPETRATE FRAUD •

CMD

Creative Micro Designs, Inc.
15 Benton Drive, P.O. Box 646
East Longmeadow, MA 01028

ORDERS ONLY: 1-800-638-3263
Questions/Support: 1-413-525-0023 • **FAX:** 1-413-525-0147 • **BBS:** 1-413-525-0148
• Office Hours: 10 am - 5 pm Mon. thru Fri •

TWIN CITIES 128 - ISSUE #32 - JULY 92
THE COMMODORE 128 JOURNAL

Table Of Contents Page # Author

Publisher:Parsec, Inc. Editor:John Brown Other Authors: Tom Adams

Twin Cities 128 is Published and Copyrighted 1992 in whole by Parsec, Inc POB 111 Salem, MA 01970-0111 USA All rights reserved. PARSEC's Telephone number 1-508-745-9125 (9-5 EST Mon - Fri Answering Machine/Voice) (10-2 EST Tues/Thur - Voice support when available)

We can be contacted online:GEnie = C128.JBEE (everyday) CIS = 70661,443 (once a month). Twin Cities 128 is done completely, except some ads, on a C-128 using Paperclip iii (text), Geos (graphics), and custom DTP software. Our database is run on a C-128 using our own custom written database software. Pages are output from a C-128 to a laser printer. This magazine is produced from start to finish on C-128 computers and we are proud of it!

This magazine is dedicated to my mother to whom I gratefully owe all my successes and failures.

RECIPE 128

A software review - staff

HOME COOKING WITH
HOME SPUN SOFTWARE.

Recipe is a program that comes to you on five disks and it is for the C-128 in 80 column mode. Each disk covers one area. There is a disk for "Breads", "Desserts", "Entrees", "Vegetables", and "Odds 'n' Ends". There are 50 recipes on each disk, enough to give the novice cook a good start. You have room to add your own favorite recipes. Each disk comes with its own loader program.

At the beginning you are faced with the opening screen and a menu bar across the bottom.

F1 = List - Will list the 50 recipes on the disk. You cannot cursor to the recipe and have it displayed but must use F2.

F2 = View - Is the next option and you enter the code of the recipe such as v-5 for the fifth recipe on the vegetable disk.

F3 = Print - I couldn't get this option to work.

F4 = Write - Lets you write your own recipe and add it to the disk.

F5 = Edit - Gives you the option to edit any of the recipes on the disk. Either the ones that came with the program or ones of your own.

F7 = Quit - You are cautioned to always use this key to exit the program to avoid possible file errors.

HELP = Help - This is another key that didn't work possibly because of my configuration.

The quality of the recipes are on the whole pretty good. Most are rather simple to make like, Double Crispy Chicken, Dill Spiced Carrots, Lemon Muffins, Escalloped Potatoes, and Prize-Winning Apple Pie. Pardon me while I go eat.

In the Odds'n' Ends category there are recipes for Potato-Celery soup, Cantaloupe Mousse, and a Shake and Bake mixture. The recipes are quite good and simple to make with this data base.
(continued on page # 9)

Your Source for Top Quality Disks and Accessories - Free Shipping* too!

Diskettes	
3-1/2" DSDD (30 ct)	29.95
3-1/2" DSHD (15 ct)	27.95
5-1/4" DSDD (30 ct)	16.95
5-1/4" DSHD (15 ct)	17.95
Disk Holders	
3-1/2", Holds 40	5.95
Buy TWO for only	10.95
5-1/4", Holds 50	6.95
Buy TWO for only	11.95

Paper	
20# One-Part Printer Paper, 2500 ct	29.95
To Zip Codes 001-799 add an add'l	10.00
18# One-Part Printer Paper, 3000 ct	32.95
To Zip Codes 001-799 add an add'l	10.00
Carbonless Two-Part Paper, 1500 sets	42.95
To Zip Codes 001-799 add an add'l	10.00
Miscellaneous	
Surge Surpressor	38.95

JD's Computer Supply
P.O. Box 873, Pearl City, HI 96782-0873

Ask about our Special Fundraising offer for User Groups!

* Free shipping available on all prepaid orders of $30 or more. COD and orders on account will be billed actual shipping costs. Please allow 14-21 days for our free shipping service, 5-7 days for COD and account orders. UPS 2nd Day Air and overnight delivery (to some locations) are available at additional cost. Hawaii residents add 4% tax. Shipping to APO/FPO, AK, GU, PR will be billed at actual cost. All products are 100% guaranteed. Not responsible for typographical errors.

OS PLUS

By Michael Gilsdorf

Change the Default Drive,
Display a Drive Prompt,
and More!

OVERVIEW

OS Plus is a simple time-saving routine whose purpose is to make the C-128 easier to use when in direct mode. Written entirely in machine language, it is designed to work with both JiffyDOS and non-JiffyDOS systems, and provides some of the same features commonly found on other operating systems such as CP/M and MS-DOS.

Some features of OS Plus are based upon the concept of a default drive. A default drive provides you the convenience and freedom of not having to type in device numbers every time a command is entered. The C128 uses this feature, but unfortunately, it won't allow you to change the default settings. It always defaults to device 8 for BASIC 7.0 commands, and device 1 (the cassette) for BASIC 2.0. JiffyDos, on the other hand, does allow you to specify the default drive, but it only affects the JiffyDos commands - not the standard Basic commands.

A partial solution to the problem is to replace the device number in commands with PEEK(186), or if you own JiffyDOS, with PEEK(190). For example, the command:

```
key 3,"directory u(peek(186))"+chr$(13)
```

will program the F3 key to display the directory of the last device used. This technique will not work though, if the last device was not a drive, and it can be a little awkward at times if you need to add file names or edit the command string. Furthermore, if the command you want to use isn't assigned to one of the function keys, using PEEK(186) provides no real benefit in terms of reducing the number of key strokes.

OS Plus overcomes these limitations by allowing you to specify and change the default drive for all the drive related BASIC commands:

```
append   backup    bload   boot    bsave
catalog  collect   concat  copy    dclear
dclose   directory dload   dopen   dsave
dverify  header    load    open    rename
run      save      scratch verify
```

Now, when a device number does not appear in the command string, Basic uses whatever default drive you specified. This feature is only active when the computer is in direct mode. That way, a program that is running will not have its operation accidently altered. Additionally, any attempt to use the cassette will be redirected to the default drive.

OS Plus makes it easy to specify which drive Basic should use as the default. Simply type the drive letter followed by a carriage return, and OS Plus will change the current default drive setting. Each letter corresponds to a different device number (e.g., A=8, B=9, etc.). Since legal device numbers for drives range from 8 to 30, valid drive letters are A to W.

OS Plus also provides a new informative command prompt. When the computer is ready to accept a command, instead of displaying a "READY.", OS Plus now displays the default drive letter followed by the error message associated with that drive.

```
A> 00, OK,00,00
```

The prompt serves both as a reminder of the current drive so you do not accidently send a command to the wrong device, and it eliminates the need to Print DS$ when an error occurs on the default drive. If the drive's error light should flash, OS Plus will automatically read and display the error message. It always ensures the DS$ error message is read from the default drive - even after using a BASIC 2.0 command. Moreover, you can redisplay the prompt anytime by merely pressing the return key on a blank line. To better illustrate how the default drive feature operates, suppose the following command is entered:

```
B> 00, OK,00,00

directory u10: directory
```

After the command is issued, the directory for device 10 is displayed followed by the directory for device 9 (the default). When the prompt reappears it displays the error message of the default drive:

```
B> 00, OK,00,00
```

If an error had occurred on device 10, it can also be displayed by simply changing to drive C, and reading the error message.

Besides providing a new command prompt and programmable default drive, OS Plus also reprograms the HELP key so it redisplays the last command line. This feature too is found on MS-DOS systems (i.e., the F3 key). Now if the previous command is no longer visible on the screen, you can display it again with a single key stroke, edit it if you like, and re-execute it.

Lastly, OS Plus makes it easier to indent a line of Basic text. As most of you know, text can be indented by typing a line number, a shifted character, and then spacing or tabbing over to where you wish the text to begin. After you press the return key, the text remains indented when it's listed. However, this procedure requires an extra key stroke, and if you edit the line, you have to remember to retype the shifted character again, otherwise you will loose the indentation. OS Plus eliminates the need for a shifted character or colon.

Sound like a lot of programming? Well surprisingly, it's all accomplished in less than 256 bytes! As you will see, we will be relying heavily on the ROM routines to do most of the work for us.

PROGRAMMING OS PLUS

Except for a small patch to the CHRGOT routine at $0386, OS Plus is designed to be wedged into the operating system through the IMAIN vector at $0302. Normally this vector points to the MAIN routine in ROM ($4DC6) which is executed right after READY is displayed. MAIN's task is to accept a command line, and determine if it is to be executed immediately, or stored in memory as a line of BASIC text. The BASIC loader (see program listing) places OS Plus in the RS232 output buffer, initializes a few free bytes at the top of page zero, and then activates OS Plus by changing IMAIN to point to its starting address of $0D00. Once OS Plus is finished executing, it returns control back to the MAIN routine in ROM. The purpose of the CHRGOT patch is to intercept the drive related BASIC commands when in direct mode, and have them use the default drive when no device number was specified (or when the cassette was chosen).

Now, let's look at how a drive prompt can be generated. OS Plus begins by reading $BE and seeing if it contains a valid drive number. If not, the previous drive is used. It also makes sure the output device is not on the serial bus; otherwise, a conflict may occur when we attempt to read and display the drive's error message.

```
         lda $90          ;check status of current
                          ;device
         bmi previous     ;if not present then use
                          ;previous drive
         lda $be          ;read default drive
devchk   cmp #$08         ;is it 8 or higher?
         bcc previous     ;no
         cmp #$1f         ;is it less than 31?
         bcc outchk       ;yes, it's a legal drive #
previous lda $fe          ;get previous drive
outchk   ldx $9a          ;get current output device
         cpx #$04         ;is it on serial bus?
         bcc readds       ;no, read error channel
         jmp skipmsg      ;don't display prompt
```

Once we have a valid drive number, the error channel of the drive can now be read and stored as DS$. If the device is not present, then the ROM routine aborts and we trap the condition the next time around when the status is read.

```
readds   sta $011c        ;set current drive
         jsr $9243        ;setup to read new ds$
         lda #$bc         ;set lo byte address
         sta $04
         lda #$79         ;set hi byte address
         jsr $f980        ;finish setup then jsrfar
                          ;$79bc - read/save ds$
```

We have read the error channel and know the drive is present, so now we can display the drive letter and error message.

```
update   lda $011c        ;fetch current drive
         sta $be          ;update default drive
         sta $fe          ;update previous drive
display  clc
         adc #$39         ;change device number to
                          ;a letter
         sta drive        ;store it
         jsr $9281        ;display prompt
        .byt $91          ;cursor-up (overwrite
                          ;"ready.")
drive   .byt $41,$3e,$20  ;"a> "
        .byt $00          ;end-of-string terminator
         jsr $55e5        ;display ds$ - error
                          ;message
         jsr $5598        ;carriage return
skipmsg
```

Before we begin writing the code to have the HELP key repeat the last command line, let us briefly look at how to reprogram the HELP and RUN keys using BASIC. BASIC 7.0 provides us with the KEY command to reprogram any of the eight function keys F1 through F8. But it does not allow us to reprogram the HELP or RUN keys - or does it? To find out, let us examine the first few lines of the KEY routine:

```
60e1      jsr $87f4      ;read the key number
60e4      dex            ;decrement the key number
                         ;by 1
60e5      cpx #$08       ;is the key number between
                         ;0 and 7?
60e7      bcc $60ec      ;yes
60e9      jmp $7d28      ;display "illegal quantity"
                         ;error
60ec-6107                ;program the key
```

The purpose of these lines is to ensure the key is one of the eight function keys. If it is, the key is programmed; otherwise, an error message is generated. Now, if these lines are by-passed, then the key number is not checked and we can program the HELP and RUN keys too! Like the KEY command, we will have to confine our strings to a maximum of 128 characters, and the key number must be limited to a range from 0 to 9. The RUN key is assigned a key number of 8, and the HELP key is assigned 9. For example, to reprogram the HELP key to display the drive error message, we simply type:

sys 24812,,9,,,"print ds$" + chr$(13)

Now, that we know the Help key can be reprogrammed using Basic, let us see how to do it using machine language. The kernel routine which programs a function key, resides at $CCA2 via vector $FF65 (PFKEY). So that the previous command line will be displayed each time the HELP key is pressed, we will have to program the key each time a command line is entered. All that is required to use PFKEY is to fill the CPU registers with the information about the key we want to program (i.e., the key number, string length and location). The command string is always stored in the input buffer at $0200 pointed to by $3D/$3E (TXTPTR). The following code programs the HELP key.

```
                            ;initialize .y register to
                            ;$ff before entering
                            ;begin by finding length of
                            ;command string
findlen   iny               ;next character in command
                            ;string
          lda ($3d),y       ;fetch character from input
                            ;buffer
          bne findlen       ;if not end-of-line, keep
                            ;searching
          sty $fd           ;save length of string
          bpl keynum        ;if length less than 128
                            ;characters, program key
null      ldy #$00          ;string too long, program
                            ;key with null string
keynum    ldx #$0a          ;help key number+1
          lda #$3d          ;address of string pointer
          jsr $ff65         ;program the key (pfkey)
          bcs null          ;if out of memory, re-
                            ;program key with null string
```

Note: After returning from PFKEY the carry status is set if the string was too large to fit in memory. If this occurs, BASIC normally displays an error message. We do not want to display an error message here though, since it may be a source of confusion. Instead, if the text is too long to fit in memory, we will just reprogram the HELP key with a null string.

The next feature we will program is the one which will maintain the spaces between the line number and the beginning of the text (i.e., indentation). Before we can write the code, we first need to find out why these spaces are not preserved. The problem can be found in the LINGET routine at $50A0. This routine uses the CHRGET subroutine to read in a line number a digit at a time, and convert it into two hexadecimal numbers. Since CHRGET skips over spaces, any trailing spaces which happen to follow the line number are ignored. LINGET ends when it finds the first non-numeric character which is not a space. The location of this character is held by the text pointer (TXTPTR) in $3D/$3E. Next, the CRNCH routine at $430A is called to tokenize the text. Its first duty is to save the value of TXTPTR so it knows were the BASIC text begins. But this pointer does not point to the first space after the line number any more, instead it points to the first non-space character following the line number!

An easy way to solve the problem is move TXTPTR back by using location $0A which holds the number of digits in the line number. The following code accomplishes the task.

```
          jsr $0380         ;skip any leading spaces,
                            ;get first digit

          ldx $3d           ;get pointer to first digit
                            ;of line number
          stx $ff           ;save text pointer
          jsr $50a0         ;linget - change line number
                            ;to hex and store
          clc
          lda $ff           ;retrieve pointer
          adc $0a           ;add number of digits in
                            ;line number
          cmp $3d           ;any spaces after line
                            ;number?
          sta $3d           ;save text pointer
          beq continue      ;no spaces, then continue
          inc $3d           ;skip a space
continue
```

Now, let's look at how to change the default drive. The code below is fairly straight forward:

```
          ldy $fd           ;fetch length of command
                            ;string
          dey               ;one character in string?
          bne immed         ;no
          lda ($3d),y       ;get first character
          sec
          sbc #$39          ;change letter to number
          jmp devchk        ;check if valid, and change
                            ;default drive

immed     jmp $4dd9         ;return to the main routine
                            ;in rom
```

The last remaining item is to write the CHRGOT patch that will intercept the BASIC drive commands. The following code is placed at the beginning of OS Plus to change CHRGOT so it executes our patch whenever the computer is in direct mode.

```
          ldy #$02          ;set to copy 3 bytes
                            ;(jmp patch)
copy      lda jump,y        ;fetch a byte
          sta $038d,y       ;copy it
          dey               ;next byte
          bpl copy          ;if more bytes, continue
                            ;copying
```

Now when CHRGOT is entered, it gets a byte then passes control to the patch below:

```
jump      jmp patch
patch     sta $fb           ;save contents of .a
                            ;register
          lda $7f           ;check for direct mode
          bpl direct        ;if direct mode, then
                            ;execute patch
```

These next few lines restore CHRGOT so the patch is not executed when a program is running.

```
          sta $ff03         ;bank 14
          ldy #$02          ;set to copy 3 bytes
                            ;(sta $ff03)
copy2     lda $4286,y       ;fetch a byte
          sta $038d,y       ;copy it
          dey               ;next byte
          bpl copy2         ;if more bytes, continue
                            ;copying
          jmp $0386         ;execute a normal chrgot
```

At this point we know the computer is operating in direct mode, so we need to check if a Basic command has been issued which uses a device number. This is done by checking the return address on the stack, and seeing where the calling routine originated.

```
direct    pla               ;get lo byte address from
                            ;stack
          tay               ;save it
          pla               ;get hi byte address from
                            ;stack
          pha               ;put it back
```

These next lines look to see if a $A3E7 return address was on the stack. If so, we know that a BASIC 7.0 command is being executed.

```
          cpy #$e7          ;is lo byte address $e7?
          bne basic2        ;no
          cmp #$a3          ;is hi byte address $a3?
          bne basic2        ;no
          lda $be           ;fetch default drive
          sta $011c         ;set as current device
          bne exit          ;done - always exit
```

It is not a BASIC 7.0 command so let's see if it is a BASIC 2.0 command by checking for a $91E5 return address on the stack.

```
basic2    cpy #$e5          ;is lo byte address $e5?
          bne exit          ;no
```

```
        cmp #$91        ;is hi byte address $91?
        bne exit        ;no
        cpx #$01        ;is device the cassette?
        bne exit        ;no
        ldx $be         ;fetch default drive
        stx $ba         ;set as current device
```

These last lines restore the stack and register contents, and then resume executing CHRGOT

```
exit    tya             ;retrieve lo byte address
        pha             ;put it back on stack
        ldy #$00        ;restore contents of
                        ;.y register
        lda $fb         ;restore contents of
                        ;.a register
        sta $ff03       ;bank 14
        jmp $0390       ;continue with chrgot
```

Well, that is it! The complete routine with all the features programmed is listed below. It is pretty much as we described it, except for some additional code to duplicate a portion of the MAIN routine and a little more to disable the prompt after a Basic program line is entered.

Type in OS Plus using "TC-128 Checksum" which can be found elsewhere in this or a previous issue. The checksum program occupies the same position in memory as OS Plus, so be sure to save OS Plus to disk before running it. Once you have a working copy, you might want to include it as part of your normal boot-up sequence. After you begin using OS Plus you may find it to be one utility you will not want to be without.

Questions or comments? You can reach me on Q-LINK under the name "MIKEALL" or on GEnie using "M.GILSDORF1". Until then...Easy DOS it!

See the next pages for the "os plus.v1.0.src" and "os plus.v1.0.bas"

RECIPE 128

Software review continued from page 4

Like I said I couldn't get the print function to work, it may have something to do with my configuration. The Servant, on a ROM chip, is present on my computer. I tried to copy all the disks to a single 1581 disk but you can not do that. You must use the 1541 disks supplied or a copy.

In one way this is good, if you only want the recipe for Soybean Sandwich Filling, you don't have to wait for all the recipes to load. It would be nice to have the program on a 1581 disk though. It would also be great to be able to cursor up to the recipe you want and have it displayed on the screen with a <RETURN>.

Another problem is that the program is device dependant which means that it must be in drive 8. Why do programmers make their programs run only from device 8 (1541s) or not recognize the fact that the people that buy the most hardware are also the people that buy the most software. Sorry, if it does not work properly with our extra hardware or firmware, it will not work properly with the readers' equipment either.

The seventeen page, unnumbered, dot matrix printed manual says if you have filled up one of your recipe disks the authors will supply an additional disk for that category for $3.00. Another minus.

I give this program a "C-". The recipes and the concept are good but the form and packaging needs work.

Buddy and Jo Anne Cowden
NCL Software 306 Highway 60 East
Dayton, TN 37321
$18.00 plus $1.50 S&H

EASYLIST

A Software review? - Staff

Easylist by Daniel Lee re-defines your function keys with this program.

F1, gets the directory of disk in drive 8
F2, loads the disk directory into memory
F3, prints the directory
F4, clears the screen
F5, lists a Basic program in memory
F6, sends a RUN command with a <RETURN>
F7, is "DLOAD"
F8 is "DSAVE".

This a nice little program, but not worth any money IMHO. It would make a nice type in as the program is short, only 12 lines. For the person just starting to use their computer this would make a good addition to the rest of the utilities. We believe the price is $3 for the disk and one sheet of instructions. Daniel Lee, 1031-B Scott Street, San Francisco, CA 94115

PROGRAM NAME: OS PLUS.V1.0.SRC

```
kc 1000 ;           *********************************************
hd 1010 ;           ************  os plus v1.0  ***************
lg 1020 ;           *********************************************
eo 1030 ;           *******    by michael gilsdorf   **********
le 1040 ;           *******  copyright (c) feb 1992  **********
jf 1050 ;           *******  parsec inc   po box 111 **********
ge 1060 ;           *******  salem ma 01970-0111 usa **********
oj 1070 ;           *********************************************
hh 1080 ;
gm 1090 ; os plus is designed to be patched into the main ($0302)
gd 1100 ; and chrgot ($0386) vectors.   c128 mode only
jf 1110 ;
df 1120 ; features:
mj 1130 ; (1) allows default drive to be changed for all basic commands
jk 1140 ; (2) displayd drive prompt and error message
fl 1150 ; (3) help key displays last command
bh 1160 ; (4) supports indentation of text in basic lines
nb 1170 ;
cc 1180 *           =   $0d00       ;assemble in rs232 output buffer, bank 0
of 1190 ;
dm 1200             ldy #$02        ;set to copy 3 bytes (jmp patch)
ld 1210 copy        lda jump,y      ;fetch a byte
cm 1220             sta $038d,y     ;copy it
cj 1230             dey             ;next byte
hf 1240             bpl copy        ;if more bytes, continue copying
cb 1250 ;
gp 1260             sty $3c         ;(curlin+1)
cm 1270             lda $fc         ;was a basic program line just entered"?
hf 1280             bpl skipmsg     ;yes, don't display drive prompt
ek 1290 ;
fa 1300             lda $90         ;check status of current device
ho 1310             bmi previous    ;if device not present, use previous drive
df 1320             lda $be         ;default device no.
op 1330 devchk      cmp #$08        ;is it 8 or higher"?
gm 1340             bcc chkmore     ;no
nk 1350             cmp #$1f        ;is it less than 31"?
eg 1360             bcc outchk      ;yes, it's a valid drive no.
jk 1370 ;
lm 1380 chkmore     tya
dm 1390             bpl immed       ;illegal drive was entered
ci 1400 previous    lda $fe         ;previous device no.
mc 1410 ;
jk 1420 outchk      ldx $9a         ;current output device no.
if 1430             cpx #$04        ;is it on serial bus"?
gm 1440             bcc readds      ;no
ok 1450 ;
ic 1460             tya
cl 1470             bmi skipmsg     ;don't display prompt
bf 1480 immed       jmp $4dd9       ;back to main - execute cmd
bc 1490 ;
ek 1500 newcmd      ldy $fd         ;length of command line
oi 1510             dey             ;one character in string"?
jm 1520             bne immed       ;no
```

PROGRAM NAME: OS PLUS.V1.0.SRC
(continued from previous page)

```
hi 1530              lda ($3d),y      ;get first character
la 1540              sec
ga 1550              sbc #$39         ;change letter to number
cj 1560              jmp devchk       ;check if valid - change default drive
gd 1570 ;
bp 1580 cmdchk       bne newcmd       ;check for drive change command
an 1590 noready      jmp $4dba        ;back to main - no "ready." prompt
ib 1600 ;
bm 1610 readds       sta $011c        ;set current drive
pm 1620              sty $fd          ;save .y flag
jp 1630              jsr $9243        ;setup to read new ds$
mf 1640              lda #$bc         ;set lo byte address
oo 1650              sta $04
nd 1660              lda #$79         ;set hi byte address
ih 1670              jsr $f980        ;finish setup, jsrfar $79bc - read/save ds$
nb 1680 ;
ko 1690              ldy $fd          ;was a drive change command issued"?
an 1700              bmi update       ;no
de 1710              jsr $4d2a        ;display: cr "ready." cr
pj 1720 ;
ji 1730 update       lda $011c        ;fetch current drive
fa 1740              sta $be          ;update default drive
dl 1750              sta $fe          ;update previous drive
cb 1760 ;
nk 1770 dislay       clc
kg 1780              adc #$39         ;change device no. to letter
ee 1790              sta drive        ;store it
db 1800              jsr $9281        ;(bprimm) display prompt
fn 1810             .byt $91          ;"" cursor-up to overwrite "ready."
io 1820 drive       .byt $41,$3e,$20  ;"a> "
he 1830             .byt $00          ;end-of-string terminator
lc 1840              jsr $55e5        ;display ds$ - error message
ie 1850              jsr $5598        ;cr
ig 1860 ;
om 1870 skipmsg      jsr $4f93        ;(inlin) input line of text
dc 1880              jsr $7923        ;save.x=txtptr($3d)=$ff save.y=txtptr+1=$01
me 1890              jsr $0380        ;(chrget) skip leading spaces, get 1st char
pl 1900              pha:php          ;save .a and .p
ge 1910              bcs setmsg       ;1st char not a number - not a program line
mc 1920 ;
ln 1930 indent       ldx $3d          ;get pointer to 1st digit of line number
mn 1940              stx $ff          ;save text pointer
cf 1950              jsr $50a0        ;(linget)change line # to hex,rtn.x=0.y=0
fd 1960              clc
ia 1970              lda $ff          ;retrieve pointer
bn 1980              adc $0a          ;add number of digits in line number
nl 1990              cmp $3d          ;any spaces after line number"?
fj 2000              sta $3d          ;save text pointer
ll 2010              beq setmsg       ;no spaces, then continue
bj 2020              inc $3d          ;skip a space
da 2030 ;
la 2040 setmsg       stx $fc          ;set message display flag
ef 2050 ;
```

PROGRAM NAME: OS PLUS.V1.0.SRC
(continued from previous page)

```
ie 2060             dey              ;set to $ff
fp 2070 findlen     iny
hh 2080             lda ($3d),y      ;search for end of command line
cm 2090             bne findlen
be 2100             sty $fd          ;save length of command string
ll 2110             bpl keynum       ;if length < 128 chars, program key
ll 2120 null        ldy #$00         ;string too long so null it
dl 2130 keynum      ldx #$0a         ;help key number+1
ck 2140             lda #$3d         ;address of string pointer
cn 2150             jsr $ff65        ;program the key (pfkey)
jl 2160             bcs null         ;if no memory,reprogram key with null
                                      string
ln 2170 ;
of 2180             plp:pla          ;retrieve .a and .p
bb 2190             bcs cmdchk       ;not a basic program line
nj 2200             jmp $4de5        ;back to main - enter/delete basic line
of 2210 ;
jb 2220 jump        jmp patch
lh 2230 patch       sta $fb          ;save .a
pp 2240             lda $7f          ;direct mode"?
ie 2250             bpl direct       ;yes
bh 2260 ;
cd 2270             sta $ff03        ;bank 14
bk 2280             ldy #$02         ;set to copy 3 bytes (sta $ff03)
kj 2290 copy2       lda $4286,y      ;fetch a byte
gi 2300             sta $038d,y      ;copy it
gg 2310             dey              ;next byte
ae 2320             bpl copy2        ;if more bytes, continue copying
gj 2330             jmp $0386        ;chrgot
gi 2340 ;
gp 2350 direct      pla:tay          ;get lo byte address from stack
bg 2360             pla:pha          ;get hi byte address from stack
be 2370             cpy #$e7         ;is lo byte address $e7"?
hg 2380             bne basic2       ;no
jd 2390             cmp #$a3         ;is hi byte address $a3"?
ik 2400             bne basic2       ;no
gp 2410             lda $be          ;fetch default drive
jf 2420             sta $011c        ;set default drive for basic 7.0
ea 2430             bne exit         ;jump always
pm 2440 basic2      cpy #$e5         ;is lo byte address $e5"?
ca 2450             bne exit         ;no
jp 2460             cmp #$91         ;is hi byte address $91"?
de 2470             bne exit         ;no
ih 2480             cpx #$01         ;is device the cassette"?
ei 2490             bne exit         ;no
ao 2500             ldx $be          ;fetch default drive
co 2510             stx $ba          ;set default drive for basic 2.0
op 2520 exit        tya:pha          ;restore stack
nn 2530             ldy #$00         ;restore .y
cl 2540             lda $fb          ;restore .a
dm 2550             sta $ff03        ;bank 14
dg 2560             jmp $0390        ;continue with chrgot
ep 2570 ;
em 2580             .end
```

PROGRAM NAME: OS PLUS.V1.0.BAS

```
mo 100 rem  os plus v1.0
en 110 rem  by michael gilsdorf
kh 120 rem  copyright (c) feb 92
kj 130 rem  parsec inc pob 111 salem ma
       01970-0111
mg 140 :
mb 150 d=peek(186): if d<8 then d=8
cc 160 for a=3328 to 3580: read b: poke a,b:
       c=c+b: next
ka 170 if c<>29413 then print "error in
       data statements": end
di 180 poke 190,d: poke 254,d: poke 252,255
       : rem  initialize
co 190 poke 770,0: poke 771,13: rem
       activate os plus
bj 200 print "os plus v1.0 activated"
fc 210 end
bg 220 :
gj 230 data 160,  2,185,180, 13,153,141,  3
fl 240 data 136, 16,247,132, 60,165,252, 16
md 250 data 100,165,144, 48, 13,165,190,201
gi 260 data   8,144,  4,201, 31,144,  5,152
ol 270 data  16, 11,165,254,166,154,224,  4
ne 280 data 144, 24,152, 48, 72, 76,217, 77
ld 290 data 164,253,136,208,248,177, 61, 56
oj 300 data 233, 57, 76, 23, 13,208,241, 76
ap 310 data 186, 77,141, 28,  1,132,253, 32
ef 320 data  67,146,169,188,133,  4,169,121
oc 330 data  32,128,249,164,253, 48,  3, 32
gk 340 data  42, 77,173, 28,  1,133,190,133
fa 350 data 254, 24,105, 57,141,107, 13, 32
ah 360 data 129,146,145, 68, 62, 32,  0, 32
bm 370 data 229, 85, 32,152, 85, 32,147, 79
jg 380 data  32, 35,121, 32,128,  3, 72,  8
jp 390 data 176, 20,166, 61,134,255, 32,160
dh 400 data  80, 24,165,255,101, 10,197, 61
ik 410 data 133, 61,240,  2,230, 61,134,252
jm 420 data 136,200,177, 61,208,251,132,253
ha 430 data  16,  2,160,  0,162, 10,169, 61
bi 440 data  32,101,255,176,245, 40,104,176
ca 450 data 140, 76,229, 77. 76,183, 13,133
gp 460 data 251,165,127, 16, 17,141,  3,255
ne 470 data 160,  2,185,134, 66,153,141,  3
bc 480 data 136, 16,247, 76,134,  3,104,168
bl 490 data 104, 72,192,231,208, 11,201,163
cc 500 data 208,  7,165,190,141, 28,  1,208
np 510 data  16,192,229,208, 12,201,145,208
df 520 data   8,224,  1,208,  4,166,190,134
mk 530 data 186,152, 72,160,  0,165,251,141
li 540 data   3,255, 76,144,  3
```

Dr. Octal's Sharp Operating Tips

Tip #0001
Gateway & Switcher
From:JBEE

One of the really neat things about gateWay (from CMD) is the task switcher that you can use under Geos. This is especially handy when using geoPaint to paste a image from one geoPaint to another OR even the same picture!

Boot up Geos, run geoPaint, load your picture, and hit the Escape key (saving geoPaint and the document). Call this copy #01. Hit the Escape key again saving the second copy. Call this copy #02. Hit the Escape again. Now clip an area from geoPaint - Copy #01. The area is now saved to disk as a "scrap". Hit escape and enter Copy#02 and post the scrap in a new position or over something else. This makes lining up graphics for things like disk labels a snap! Keep pasting from Copy #01 to Copy #02. This trick works because Geos saves only one paint scrap to a disk, replacing previously saved scraps with newer ones whenever you clip from a geoPaint. A note of caution: try not to scroll too much and close both copies starting with Copy #02.

Tip #0002
Handyscanner
From:J.Robbins

Use a flashlight over the green plastic window of the Handyscanner 64 on those dark background pictures with the first position set on dither, between a 200 and 250% capture, bright arrow lined up with the solid arrow, and contrast lined up with the large end of the arrow. Then the dark background pictures transfer almost exactly the same as the Macpaints and Gifs do to geoPaint (except the color of course). He reports a blue filter gives the best results. After seeing some of his awesome geoPaints (digital captures) we agree!

Tip #0003
C-1571
From:Mike Minnig

If you have a C-128D and other disk drives you must have been frustrated already by programs that will not boot from any other device number except #8 or by Geos that will not recognize more than three disk drives or performs a swap with one of

(continued on page 25)

INTERNAL FUNCTION RAM

By Richard Curcio

I.F.R. REVISITED V2.1

My Internal Function Ram project as it appeared in TC128 #29 has problems. This project allowed the C128 to have a Static Ram in the empty socket reserved for a function ROM or Eprom. The circuit included battery-backup so the Static Ram could retain its contents while the computer was turned off.

I offer no excuses for not detecting the problems sooner. Instead, I offer a fixed and (serendipitously) more versatile design, which will, if you already have a ROM in the empty socket, allow selection of it or the Static Ram, via a switch or software. The program for the original project has been revised as well.

I apologize for any frustrations the earlier project may have caused.

OBSTACLES

A detailed recounting of how the original design was supposed to work and why I thought that all was well -- in other words, what I had overlooked -- would take up too much space. The older circuit won't harm the C128 and will perform as described -- with limitations. If I took the time to explain those limitations, I'm sure most readers would agree that they're unacceptable. So let's just get right to the new circuit.

The Bank 4-7 configurations of the C128 select internal function ROM. In these standard banks a ROM or Eprom in socket U36 occupies $8000 to $ffff with a 4K gap at $d000 for I/O. You can't just plug a Static Ram (Sram) into the empty socket because 1) R/W is not present on the function ROM socket and 2) when the Programmed Logic Array (PLA) detects a WRite to what is supposed to be a ROM location, the function ROM enable (called /FROM in the P.R.G. schematic,) "goes away" and system Ram is instead enabled and written. Since the PLA won't permit writing to a ROM or anything else in the U36 socket, the new design bypasses the PLA. This is not as difficult as one might expect.

Two signals from the MMU called MS0 and MS1 tell the PLA to enable system ROM, external or internal ROM, or system Ram. When MS0 =0 and MS1 =1 internal ROM is selected. The PLA considers a number of other signals in determining which enable to generate. When R/W is low, the PLA disregards the state of MS0/1. The new circuitry removes R/W from the decision and substitutes the "1" period of the two-speed system clock, which is when the processor has control of the address and data busses. The result is a nice, clean enable of the proper duration.

THE CIRCUIT

Figure 1 shows the new design in a mix of mechanical and schematic representation. The static Ram is inserted in the 128's empty ROM socket with its pins 1, 20, 27 and 28 bent out. A 74HC138 decoder receives MS0, MS1 and the 2MHz clock on its inputs and its Y3 output enables the Sram. This decoder MUST be an 'HC part if battery-backup is included. The Sram and the 'HC138 are powered by the computer's +5 volts OR the battery. If you choose to omit the battery and diodes, insert the Sram's pin 28 into the socket and connect pin 16 of the 'HC138 to +5 volts.

The Sram's Write Enable input /WE connects to FR/W, a Read/Write internal to the 128. (It's called R/WA in the SAMs schematics.) It does not appear on the expansion connector, so an REU or other external device cannot pull it low and alter the Sram.(An REU-initiated WRite to what is supposed to be a ROM bank will simply "fall-through" to underlying system Ram, leaving the Sram untouched.) In the low-profile 128, FR/W is available at pin 1 of U57, inside the video box, which is quite close to the empty ROM socket. In the 128D, this signal is at pin 3 of U61, near the left side of the main circuit board and also close to the empty socket.

The nearest point to get the 2MHz clock on the low-profile 128 is at pin 4 of U22, the 80 column controller, which is also inside the video box. On the 'D, different versions of this chip have 2MHz on different pins. For certainty, obtain this signal at pin 1 of the 8502 microprocessor, U6. MS0 and MS1 are at pins 18 and 17 respectively of the PLA, U11, which is located at center front of both the low-profile and 'D. (Although they are rare, there are 'Ds in existence that use a low-profile mother-board. For a positive determination, count the number of dynamic Rams at the front left of the board. Sixteen 16-pin ICs indicates a low-profile board, while four 18-pin chips means you have a true 'D.)

Because the pin-out of the Sram is slightly different from that of a ROM or Eprom, pin 1 must be bent out and connected to address bit A14. On my flat 128 I found A14 at the feed-through immediately to the right of the "2" of the identifier "R32" near U32 and U33. On the 'D, A14 is at pin 27 of ROMs U32 and U34 or pin 18 of U42 (74LS244).

The 32K x8 static Ram could be a 62256, 43256 or 58256. For maximum battery life it should be a low-power device (-L or -LP suffix,) having a "standby" current consumption of 100 microAmps. Regular power Srams have a standby current of 2 milliAmps, which is still pretty miniscule.

I installed the 'HC138 in my low-profile 128 by sticking it to the main board upside-down using double-stick foam tape. Connections were made by wire wrapping directly to the pins. Lithium "coin" batteries are available with solder-tabs, and these can have wires soldered to them. A holder for tab-less coin batteries can be constructed with two rectangular pieces of un-etched printed circuit board; solder wires to the bare copper, put the battery in between the two pieces and hold together with rubber bands and wrap the sandwich with paper and more rubber bands. (Of course the polarity of the battery is important, but the diodes will prevent reversed polarity from damaging the Sram, 'HC138 or the computer itself.) This can then be tucked behind the video box or someplace where it won't flop around. In the 'D, there's enough room for a holder for two AA cells. These won't last as long as lithium, but they should last their shelf life -- at least a few years.

Note that the Sram in the U36 socket still receives the PLA-generated /FROM on its Output Enable (/OE) pin. Normally the PLA enables each ROM via /OE. A write will therefore disable the Sram output and that's fine; for the Sram, /OE is "don't care" when /WE =0. (Certain other memory ICs require that /OE =1 when /WE =0.)

OPTIONS

When the optional Write Protect switch within the dotted lines in figure 1 is open, FR/W can't reach the Sram. Use this to prevent accidental writes to the Sram or to fully emulate an Eprom.

The 10k resistor on the 'HC138 'B input insures that the proper combination of inputs causes output Y3 to go low, selecting the Sram. If 'B is made low by grounding it with the optional Select switch, then another Sram or an Eprom could be selected by connecting its /CS to Y1 (pin 12). One would think that, since the 128 has only one empty socket, this idea is mere theory. There are ways to obtain the room for a second 28-pin IC, but different methods are needed for the low-profile and 128D. These will covered later on.

TESTING/SOFTWARE

After installing the circuit, use the Machine Language Monitor "m" command to display the first page of the IFR: m 48000 <return>. You should see random values. Cursor up to the m command and hit return. If any of the previously displayed values change, bank 4 is empty -- the Sram isn't getting enabled. Turn off your computer and find your mistake. If the random bytes remain constant, fill that first page with some value, say $55 or $aa:

```
f 48000 480ff 55
```

Now use the m command to confirm that the fill was successful. If it was, turn off the computer, wait a few seconds, then power up and confirm that the battery-backed Sram retained the fill value.

Before your Internal Function Ram can be used from Basic, it must be initialized. Use the mlm Transfer command to copy the routines in the last page of system ROM to the same locations in Bank 4:

```
t fff05 fff44 4ff05
```

Cursor up and change the "t" to a "c" (Compare) and hit return. The mlm should print nothing, indicating that all the locations match. Now transfer the system vectors to the last six bytes of Bank 4:

```
t ffffa fffff 4fffa
```

Ignore the "?" that appears upon completion. You can now safely access your IFR with Peek, Poke, BLOAD, BSAVE and SYS. (The mlm safely accessed bank 4 before the initialization because it disables interrupts during "m", "f", "t" and "c".) Bear in mind, however, Poke and BLOAD will also alter underlying system Ram (Ram 0 when Bank 4). Note also that the standard IFR Banks 4-7 and 12 include I/O in the $d000-dfff range.

Program 1 is the loader for a Mover which will transfer data to the IFR without altering the system Ram under it. It is designed to reside and execute in the IFR. The ml can be relocated to a different start address by changing the variable SA in line 200. SA must be between 32768 and 48924. Access the routine with

bank 12: sys sa, host bank, direction,,, host start, host end, ifr start

You MUST use Bank 12, which includes the Kernal and I/O. The host bank is system memory in the standard Bank configurations; 0-3, 14 and 15. Banks 4-13 are not allowed. This restriction can be bypassed. Direction is zero to move data TO internal function Ram and greater than zero to recall data FROM that Ram. The three commas must be present. Host start and end are self explanatory, while IFR start must be at least 32768. No address can be greater than 65279, as that would affect the MMU registers at $ff00-$ff04 and the important routines and vectors in page $ff. As the routine moves data, it checks its pointers and will halt the move and set the Carry bit if either the source or destination reaches page $ff. Carry is also set if either pointer "wraps" to zero page -- which theoretically can't happen, but one never knows. Use the RREG function to test the Carry from Basic. The routine does not detect if an IFR destination will write over the Mover itself.

The Mover calls the system INDSTA and INDFET routines at their Ram 0 locations, instead of through the Kernal jump table. Calling these routines via the jump table is time consuming, because the bank number is converted to a configuration value for each byte. To speed things up, the Mover performs the host bank to configuration conversion just once. Also, to access $d000-dfff of the Sram a modified Bank 4 must be used because the standard Banks include I/O in that range.

Because writing to the Sram also writes system Ram, the Mover performs two loads and stores for each byte moved to the IFR. First the byte in Ram 0 "under" the IFR destination is read and stored on the stack. Then the host source byte is written to the IFR, changing Ram 0 in the process. The byte on the stack is then restored to Ram 0. In this way, we do not lose the use of the underlying system Ram. This is why Banks 4-13 are not allowed as "host." If the "from" portion of the routine were to move data to the IFR as host, the byte under the IFR destination would be lost. The "to" operation would work properly, but this is a "dumb" mover; if the source and destination are in the same bank, and they overlap, the move becomes a fill. If you really need to move data around inside the IFR, and don't care about the under-bytes, you may use Banks 4-13 as host by calling the routine at sa+27.

The "to" portion of the Mover assumes that the MMU Pre-Configuration Register at $d501 (PCRA) contains its default value, selecting the Bank 0 configuration when any value is stored in the corresponding Load Configuration Register at $ff01 (LCRA).

I must emphasize that writing to Bank 4 or the other internal function ROM configurations enables the Static Ram and system Ram simultaneously, a very different situation than what normally occurs when attempting to write to ROM. Any routines you place in your IFR should not use the short-cut method of writing to an internal function ROM region to accomplish a write of system Ram that is not at the moment visible. Use INDSTA. However, if the optional Write Protect switch is installed and opened, then the Sram is as unwriteable as a ROM or Eprom.

128D PLUG-IN BOARD

In a 128D, to have a choice of Sram or Eprom in Bank 4, a board can be constructed to plug into the empty ROM socket. As shown in figure 2, socket 1 is intended to hold an Eprom. This should be a wire-wrap socket with "2-level" length pins. These are long enough to allow the plug-in board to clear the main board components that will be under it; 3-level length pins are acceptable, but longer than necessary and you may have trouble keeping them properly aligned. The perforated board should have "pad-per-hole" copper plating so that the socket can be firmly soldered to it. Pin 20 of socket 1 is cut close to the board so it does not make contact with the corresponding receptacle of the main board U36 socket. The two other sockets could be solder "tail" or wire-wrap with the pins cut as short as possible

Pins 28 and 1 of socket 1 bring +5 volts to the plug-in board, while pin 14 supplies ground. Each pin of socket 1 is wired to the same pin of socket 2 EXCEPT for pins 1, 20, 27 and 28. Pin 27 of socket 1 connects to pin 1 of socket 2 (address bit A14). Pin 27 of socket 2 is the Sram Write

Enable /WE and it gets wired to FR/W, either directly or through the optional switch. Both 28-pin sockets continue to receive /FROM at their /OE pins 22. Pin 20 of each socket gets wired to the specified 'HC138 outputs. All other signals from the main board can connect directly to the plug-in or, better yet, through connectors and pins so that the whole assembly can be removed without unsoldering. There's a large hole in the front of the 128D chassis which will permit wires from the plug-in to reach any switches you mount on the front panel. Other, smaller holes can be used to secure a double AA battery holder using twist ties.

To have a second Sram instead of an Eprom, the simplest method would be to plug it into socket 1 with its pins 1, 22, 27 and 28 bent out and wired as shown in figure 1. As illustrated in figure 2, the plug-in is somewhat roomier than absolutely necessary, but about as roomy as it ought to get.

LOW-PROFILE STRATEGY

To gain another 28-pin socket in the low-profile 128 you need an Eprom programmer so that the two 16K ROMs holding Basic and the Machine Language Monitor can be combined into one 27256 32K Eprom. One ROM, U33, holds the Basic interpreter from $4000 to $7fff (call it baslo). The other ROM, U34, holds the rest of Basic and the MLM (bashi, $8000 to $bfff). The original ROMs can be copied without removing them from the system board by saving their contents to disk:

```
bsave "baslo", b15, p16384 to p32768 (end +1)
bsave "bashi", b15, p32768 to p49152
```

Burn baslo into the lower 16K (0-$3fff) of a 200 nanoSecond 27256 and bashi into the upper 16K ($4000-$7fff). It's a good idea to label each original ROM with the socket number it came from, so they can be re-installed correctly, if needed. (If your Eprommer software works only in C64 mode, use the 128 mlm to transfer bashi to $4000-7fff in Ram 0, then BSAVE "BASHI", B0, P16384 TO P32768. In C64 mode load and burn each file separately.)

Figure 3 shows how I installed this Basic Eprom in my flat 128. Diode logic enables the Eprom's /OE when baslo (labeled /ROM2 in the P.R.G. schematic) OR bashi (/ROM3) go low. These signals are obtained at feed-through holes near the sockets. (Note that the bashi feed-through is partially covered by the U34 socket.)

Additionally, baslo pulls the Eprom's A14 low, selecting the lower 16K. The Sram can then be installed in the U34 socket, and a function ROM or ready programmed Eprom (if I ever get one,) plugged into U36 with its pin 20 lifted and wired to pin 14 of the 'HC138. I could instead install a second Sram.

The Eprom could also be burned with baslo in the UPPER 16K, and bashi in the lower, duplicating the arrangement in 128D Roms. In this case, the feed-through connections would be reversed, so that bashi pulls pin 27 low. Incidentally, all 128 system ROMs have large areas containing $ff; that is, unused. Once a ROM has been copied to Eprom, the ambitious might consider burning their own routines into these unused areas. Note that the 128 Kernal ROM must be removed from the computer to copy it because 4K of Z80 start-up code is "hidden" while the 128 is in 8502 native mode. You'll need to use another 128 or 64.

SOFT SELECT

Since a logic 0 or 1 on input B ofthe 'HC138 allows a choice of pins 12 or 14 as the device enable, this can be accomplished via software, instead of a switch. One possibility is to use a Cassette control line. CASS SENSE, which detects when play/record on the Datasette is pressed, is normally an input. It defaults to logic 1 on reset. Connect this to pin 2 of the 'HC138 and, when low, it will select whatever is connected to pin 14 (Y1). The first time you want to change the state of CASS SENSE you'll have to change bit 4 of the data direction register at location 0: Poke 0, Peek(0) OR 16 (or the ml equivalent,) does this without changing the other ddr bits. Thereafter, a 0 on bit 4 of location 1 will select pin 14 of the 'HC138 as the active output, while a 1 will select pin 12. Use Poke 1, Peek(1) AND 239 for a zero bit 4, and Poke 1, Peek(1) OR 16 for a 1. (Note that some software, after manipulating location 1, may not return it to the state in which it was found. Be wary of programs that use custom characters in 40 columns or tinker with the VIC's Color Memory blocks. All the system routines that alter location 1 do so in a "considerate" manner.)

CASS SENSE is available at pin 26 of the 8502 (U6) or finger 6 of the cassette connector. If CASS SENSE =1 selects Sram as the default Internal Function device, you could write an auto-starting program that looks for a certain keypress on start up to keep the Sram or select function ROM, if any. All of this assumes you will not be using cassette storage.

Note that ground on input C of the 'HC138 selects pins 12 or 14 as active outputs. Connecting C instead to another control line (CASS WRT?) would allow pins 10 and 7 to select two more Internal Function devices for a total of four! This, I think, might be taking things a bit too far, especially in the cramped quarters of the flat 128.

PROGRAM NAME: IFR.MOVER.BAS

```
hn 100 rem written by richard curcio
di 110 rem copyright (c) 1992
cd 120 rem parsec inc  po box 111
co 130 rem salem ma 01970-0111  usa
mg 140 :
om 150 rem program name 'ifr.mover.bas"
nk 160 :
ek 170 rem *** initialize bank 4 ***
na 180 bank15:poke53274,0:rem irqs off
pc 190 fori=65285to65348 :rem ff05-ff44
ph 200 bank15:x=peek(i):bank4:pokei,x:next
lc 210 fori=65530to65535 :rem fffa-ffff
al 220 bank15:x=peek(i):bank4:pokei,x:next
mb 230 bank15:poke53274,241:rem irqs on
ck 240 :
jd 250 rem *** install ifr mover ***
hg 260 rem !!! destroys ram 0 bytes !!!
fd 270 sa=34000:rem relocating
kg 280 ifsa<48923andsa>32768then300
fl 290 print"bad address!":end
ek 300 ck=0:bank12
kf 310 fori=0to226:readd:pokesa+i,d:ck=ck+d
md 320 next
kj 330 ifck=29213then350
mc 340 print"error in data!":end
dp 350 x=sa+192:gosub420
mm 360 pokesa+38,l:pokesa+39,h
cc 370 x=sa+203:gosub420
hi 380 pokesa+166,l:pokesa+167,h
eo 390 print"ifr mover installed in"
de 400 print"bank 12,"sa"to"sa+226
bl 410 end
ng 420 h=int(x/256):l=x-h*256:return
ca 430 data 201, 16,144, 15,169, 15,162,125
jl 440 data 160, 40,133,  2,134,  3,132,  4
po 450 data  76,227,  2,201, 14,176  4,201
lm 460 data   4,176,233,134,207,170, 32,107
of 470 data 255,133,206,162, 11,189,192, 19
jo 480 data 157, 16,  1,202, 16,247, 32, 16
dd 490 data   1,132,172,133,173, 32, 16,  1
ah 500 data 201,255,240,200,132,174,133,175
bp 510 data  32,183,238,176,191, 32, 16,  1
co 520 data 201,255,240,184,201,128,144,180
el 530 data 132,195,133,196,162,195,160,172
hd 540 data 165,207,240, 65,142,170,  2,140
bo 550 data 185,  2,160,  0,169,255,197,173
nb 560 data 240, 41,197,196,240, 37,162, 23
an 570 data  32,162,  2,166,206, 32,175,  2
ph 580 data  56,165,172,229,174,165,173,229
pb 590 data 175,240, 18,230,172,208,  4,230
bl 600 data 173,240,  8,230,195,208,  8,230
pg 610 data 196,208,  4, 56, 96, 24, 96,165
ji 620 data 207,208,201,240, 19,140,170,  2
jm 630 data 142,185,  2,162, 23,189,203, 19
lg 640 data 157, 16,  1,202, 16,247,160,  0
kc 650 data 169,255,197,173,240,221,197,196
kg 660 data 240,217, 32, 16,  1, 56,176,185
ek 670 data  32,221,  2, 32, 15,136,162,  6
km 680 data  76,201,  2,141,  1,255,177,195
fm 690 data  72,166,206, 32,162,  2,162, 23
fh 700 data  32,175,  2,104,145,195,162,  6
jo 710 data  76,201,  2
```

PROGRAM NAME: IFR.SRC

```
db 1000 sys4000
da 1010 ;
md 1020 ;written by richard curcio
ed 1030 ;copyright (c) 1992
no 1040 ;parsec inc  po box 111
fk 1050 ;salem ma 01970-0111  usa
gd 1060 ;
bp 1070 ;program name 'ifr.src'
hh 1080 ;
ib 1090 ;
ag 1100 ;power assembler (buddy128)
jf 1110 ;
km 1120 *= $84d0
kj 1130 ;
be 1140 ;address = 34000 decimal
ln 1150 ;
hk 1160 .bank 12
nb 1170 ;
fg 1180 ;some assemblers might not allow a
dh 1190 ;rom bank. if so, assemble to ram 0
ad 1200 ;and use mlm to transfer to bank 4
pj 1210 ;
jd 1220 .mem
an 1230 ;
gm 1240 ;move to/from internal function ram
gp 1250 ;in bank4 (normally eprom or rom)
cl 1260 ;
ff 1270 ;.a=bank (0-3,14,15), .x=0=to
ea 1280 ;
pc 1290 setup  cmp #$10    ;host bank <16
gc 1300 :      bcc xcp     ;yes
fo 1310 ;
oe 1320 ;call basic illegal quantity
```

```
af 1330 ;using jmpfar
hm 1340 ;
hk 1350 illqty lda #$0f    ;bank15
jf 1360 :      ldx #$7d    ;addr hi
lc 1370 :      ldy #$28    ;addr lo
hl 1380 :      sta $02
oi 1390 :      stx $03
pn 1400 :      sty $04
im 1410 :      jmp $02e3  ;jmpfar
mm 1420 ;
en 1430 xcp    cmp #$0e
cj 1440 :      bcs ok      ;banks 4-13
cg 1450 :      cmp #$04    ;not allowed
ml 1460 :      bcs illqty
ob 1470 ok     stx $cf     ;save direction
nc 1480 :      tax
hn 1490 :      jsr $ff6b  ;get config.
ce 1500 :      sta $ce     ;store it
cg 1510 :
bj 1520 ;copy code to low end of stack
dk 1530 ;
op 1540 :      ldx #$0b
ph 1550 cs1    lda stack1,x
ob 1560 :      sta $0110,x
bj 1570 :      dex
ki 1580 :      bpl cs1
he 1590 :      jsr $0110  ;call it
ib 1600 ;
bn 1610 :      sty $ac     ;host start lo
be 1620 :      sta $ad     ;host start hi
oi 1630 :      jsr $0110  ;get host end
pg 1640 :      cmp #$ff
jj 1650 :      beq illqty ;page ff no good
np 1660 :      sty $ae
ja 1670 :      sta $af
an 1680 :      jsr $eeb7  ;start < end
lc 1690 :      bcs illqty
od 1700 :      jsr $0110  ;get ifr start
dm 1710 :      cmp #$ff
mo 1720 :      beq illqty
do 1730 :      cmp #$80
kn 1740 :      bcc illqty ;<$8000 n.g.
hb 1750 mlalt  sty $c3
hi 1760 :      sta $c4
oc 1770 :      ldx #$c3
gc 1780 :      ldy #$ac
gb 1790 :      lda $cf     ;direction flag
lf 1800 :      beq movto
fe 1810 :
jb 1820 ;move data from int. func. ram
gi 1830 ;
ae 1840 :      stx $02aa  ;indfet pointer
dj 1850 :      sty $02b9  :indsta pointer
ke 1860 :      ldy #$00
in 1870 movfr  lda #$ff
op 1880 :      cmp $ad     ;page $ff n.g.
jb 1890 :      beq fferr
am 1900 :      cmp $c4
kf 1910 :      beq fferr
fm 1920 :      ldx #$17    ;bank4, no i/o
bl 1930 :      jsr $02a2  ;do indfet
fn 1940 :      ldx $ce     ;get dest cnfg
fd 1950 :      jsr $02af  ;do indsta
gk 1960 :      sec
de 1970 cbump  lda $ac     ;compare $ac/ad
ao 1980 :      sbc $ae     ;to $ae/af
if 1990 :      lda $ad
kn 2000 :      sbc $af
mk 2010 :      beq exit    ;reached end
kd 2020 :      inc $ac     ;increment pntr
nf 2030 :      bne zzzz
nh 2040 :      inc $ad
ob 2050 :      beq fferr   ;$ad rolled over
ch 2060 zzzz   inc $c3
jf 2070 :      bne dtest
jm 2080 :      inc $c4
kj 2090 :      bne dtest
bm 2100 fferr  sec         ;rolled over
gm 2110 :      rts
hc 2120 exit   clc         ;complete move
ia 2130 :      rts
jp 2140 ;
bh 2150 dtest  lda $cf     ;direction
pl 2160 :      bne movfr
ji 2170 :      beq mov2
mh 2180 :
hl 2190 ;move data to int. func. ram,
aj 2200 ;preserving underlying data,
ii 2210 ;byte by byte
op 2220 ;
fb 2230 movto  sty $02aa  ;indfet pointer
fk 2240 :      stx $02b9  ;indsta
ga 2250 :      ldx #$17    ;copy to stack
mo 2260 cs2    lda stack2,x
kj 2270 :      sta $0110,x
ob 2280 :      dex
hh 2290 :      bpl cs2
dp 2300 :
gi 2310 :      ldy #$00
md 2320 mov2   lda #$ff
ld 2330 :      cmp $ad     ;page $ff n.g.
ff 2340 :      beq fferr
na 2350 :      cmp $c4
gj 2360 :      beq fferr
ig 2370 ;
ei 2380 :      jsr $0110  ;stack code
jk 2390 :
ce 2400 :      sec
ho 2410 :      bcs cbump
```

```
li 2420 ;
ao 2430 ;these routines are moved to the
ci 2440 ;low end of the stack as needed.
ng 2450 ;
kd 2460 ;the first sets bank 15, calls
en 2470 ;basic expression evaluator, sets
og 2480 ;bank 12 and returns
po 2490 ;
ph 2500 stack1 jsr $02dd  ;part of jsrfar
oc 2510 :      jsr $880f  ;.y=lo, .a=hi
fb 2520 :      ldx #$06   ;bank 12
cl 2530 :      jmp $02c9
da 2540 ;
mm 2550 ;this code moves data from host
fn 2560 ;to int. funct. ram, preserving
ca 2570 ;ram 0 bytes
fj 2580 ;
hf 2590 stack2 sta $ff01  ;lcra=ram0
hg 2600 :      lda ($c3),y;get byte
da 2610 :      pha        ;save it
fg 2620 :      ldx $ce    ;host config.
ah 2630 :      jsr $02a2  ;indfet
cp 2640 :      ldx #$17   ;bank4, no i/o
dg 2650 :      jsr $02af  ;indsta
nl 2660 :      pla        ;restore byte
me 2670 :      sta ($c3),y;to ram 0
pc 2680 :      ldx #$06   ;bank 12
gc 2690 :      jmp $02c9  ;& rts
me 2700 .end
```

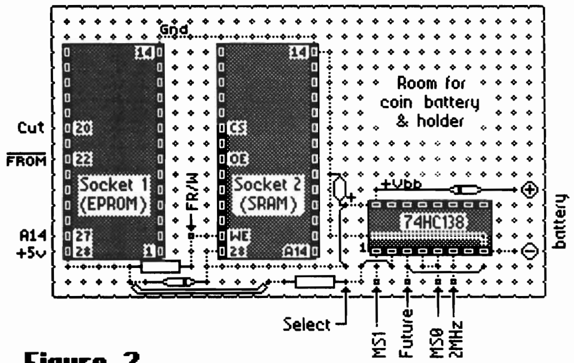

Figure 2
Plug-in board for 128D.
R.C.'92

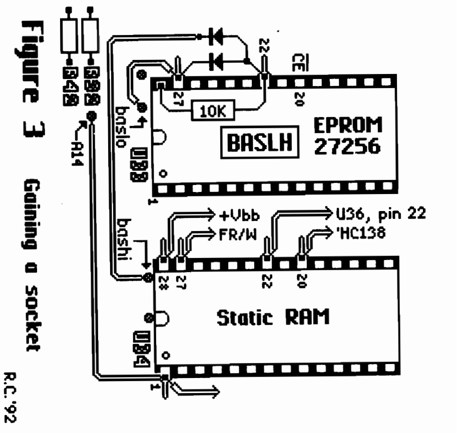

Figure 3 Gaining a socket
R.C.'92

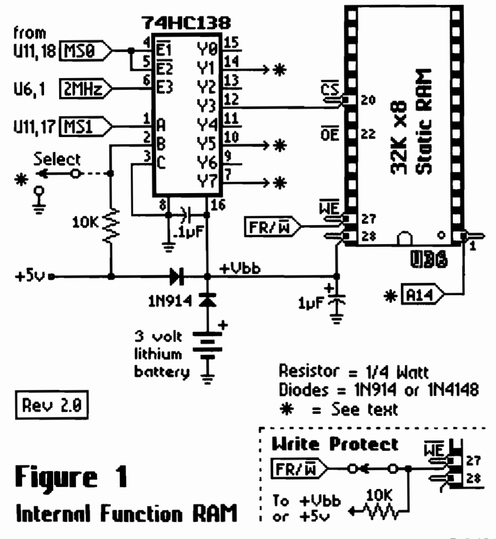

Figure 1
Internal Function RAM
R.C.'92

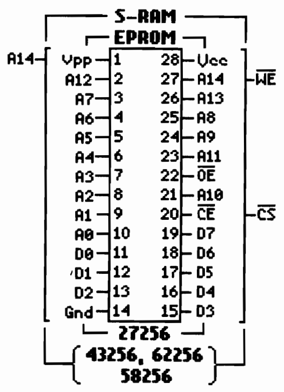

Figure 4
pin assignments

SERVICING THE C-128 KEYBOARD
by Dave Farquhar

DISCLAIMER
This modification will render any warranties on your equipment null and void. The Author and Publisher do not assume any liability for Purchaser's implementation of these instructions. All information is believed to be accurate.

SERVICING THE C-128 KEYBOARD
Traditionally, Commodore microcomputers have been extremely reliable. The C-128 is no exception. However, nearly every computer, Commodore or otherwise, eventually develops problems with its keyboard, because it is exposed to the elements much more than any other component.

Such failures are usually caused by dust accumulation on the printed board, a film developing on the conductive rubber pads of one or more keys, or a combination of the two.

When you fall victim to this problem, you have several options. You could take the machine to the local service center, if there is one, for repair. But, this can be time-consuming and expensive. You could replace the keyboard, but C-128 keyboards can cost you $70 or more, if you can find one. This price is outrageous when you consider that Radio Shack sold surplus C-16 keyboards for years at $4.95 a pop. The last option is to service the keyboard yourself. This is not as monumental a task as it first seems. Usually, it can be done in 20 minutes or less, at a very low cost using household items.

You will need the following materials:

Flathead Screwdriver
3/32" or 2.4 mm Phillips screwdriver
3/16" or 3 mm hex socket
3.8 mm Phillips screwdriver
Soldering iron
3 containers
Cotton swab
Isopropanol or Rubbing Alcohol
Pencil Eraser

If you cannot match the screwdriver or socket sizes exactly, don't worry about it, the sizes are approximate. Also, if you do not have a Phillips screwdriver that fits, a flathead will do, but be very careful not to strip the screw's head.

It is best to read these instructions at least once before attempting this project, and it is probably best not to undertake it until failure arises. This is not a difficult project, but I feel that the old adage "If it ain't broke, don't fix it" applies here.

This project should be undertaken on a relatively dark surface, such as a dark table cloth or bedsheet, so as to make it harder to lose the screws. This is because sheet metal screws are tough to find on a light surface because the screws themselves are light in color.

The first and most difficult step is actually opening the keyboard of the flat C-128. First, unplug the power supply from the wall outlet. Next, unplug the power supply from the computer. Remove the screws on the bottom of the machine and put them in one of the containers for retrieval. Then, insert the flathead screwdriver near the place where the seam angles. Gently pry out on the lower half of the case, while simultaneously prying in on the top half. The case should then separate fairly easily.

The keyboard is bolted to the top half of the case, and attached to the motherboard via a grounding strap and a "D" connector. Unscrew the grounding strap, and gently unplug the keyboard with a rocking, upward motion, being extremely careful not to bend or break the pins. Set the lower half of the case aside.

Before proceeding further, it is a good idea to now plug in the soldering iron, so it will be ready when you need it.

Next, using the hex socket, unbolt the keyboard from the top half of the case, being sure to keep track of the bolts and their plastic washers. Set aside the top half of the case. You may wish to use a socket wrench for this, but I find it just as easy to grasp the socket between two fingers and turn it that way.

There are several methods to the actual cleaning, presented below.

The slowest but most economical and most thorough method involves a complete disassembly. Gently remove the many tiny screws on the lower surface of the keyboard, being extremely careful not to lose them. Take your time, as the screws are very easy to strip.

You will notice that 3 keys on the board have soldered connections: shift lock, caps lock, and 40/80 display. You will need to de-solder these before you can disassemble the keyboard any further. If you are uneasy about using a soldering iron, you could clip these connections with wire cutters, but you will lose use of those 3 keys. The desoldering process is simple: hold the flathead screwdriver beneath the wire, touch the soldering iron to the connection, and pry up with the screwdriver as soon as the solder melts. After all 6 connections have been detached, the keyboard easily lifts away from the printed board.

Examine the printed board, especially the areas beneath whatever keys have been malfunctioning. Clean any offending areas with a cotton swab soaked with alcohol. Next, examine the rubber pads of the keyboard. A like-new pad will have a slightly dull finish. If the computer has been used in the vicinity of smokers, humidifiers, or fireplaces, they may have developed a nonconducting film on them. Clean any offending pads with a pencil eraser (do not do this to all of the pads, as it would be time-consuming and would expose them to unnecessary wear). Personally, I like the Pentel "Clic" erasers, available in many college book stores, because they are very thorough yet less abrasive than most erasers, but any eraser should do. Simply rub the eraser on each pad until its surface is dull.

A second, less ambitious method simply involves running the entire keyboard under hot water for a few minutes and blow-drying it. This will work, but may not eliminate the film on the rubber pads. Also, if your community has particularly hard water, you could be subjecting your keyboard to excess mineral build-up, although probably not enough to cause serious problems. Be sure the keyboard is completely dry before reassembly.

Another method simply requires soaking the entire keyboard in an alcohol bath. This procedure is thorough and faster drying than water, but the alcohol could wash off the key designations, or possibly damage the plastic. It also may not be enough to remove film from the contacts.

Re-assembly is relatively simple: just reverse the disassembly process.

The final result of this project: a like-new keyboard, money saved, and the satisfaction of having done the job yourself. The cost was negligible as well: just the price of this magazine, the cost of the materials used, and your time. Better deals are few and far between.

A few additional procedures will be mentioned for people with well used and worn keyboards.

Before you close up your C-128 you can use a good keycap plunger for a "dead" or "flat" one.

Since most people hardly use the shift/lock, Q, X, or tab keys on a regular basis these are prime keys to use to revive "flat" or "dead" ones. You could also use the keys from the numerical keypad or from the top row of number keys.

If your springs are weak you can always use the springs from the C-16 keyboard mentioned earlier in the article.

Though C-16 keyboards are not even close to a match you can still use some of the parts. If you cut the springs for the C-16 keycaps down to the proper height (not an easy job) and carefully fit them under the caps of the C-128 keyboard, they do provide good enough bounce. The C-16 springs are especially good for the F-Keys and other heavily used keys.

One caution though, use care when removing the keycaps because it is easy enough to damage the keycap assembly. Since these sell for $8 a key (a unit) a bit of caution and a light hand is advised.

Dave Farquhar: This project is dedicated to the memory of the late Norbert McGuire, who made this entire project possible about 2 years ago by loaning the author equipment and instructing him in its use, changing him from an unenlightened goof to an enlightened one. Thanks, I owe you one.

TURBO CHARGING CP/M WITH "SG TOOLS PROGRAMMER'S TOOL BOX"

by Steve Goldsmith

Part 1 of 3 - Updated: 03/31/92

INTRODUCTION

How would you like to have 80 column color windows that pop up in a flash, drop down menus, page flipping, full access to all the C128's I/O chips, and more in CP/M mode? Just imagine all the applications you could create if you had a programmer's tool box customized for the C128 in CP/M mode. Now you don't have to imagine because we are going to build our own CP/M tool box with Turbo Pascal! I have used SG Tools to create a 80 column, color, windowing application, a VDC 640X200 PCX file viewer, and more in CP/M!

OVERVIEW

This series of articles is not intended to be a tutorial of Pascal. If you have not used Pascal, but are proficient in Basic or structured Basic I recommend "Turbo Pascal for Basic programmers" from Que books. Pascal is easy to learn because it was designed as a teaching language like Basic and there are many good books on Pascal programming. You can also port the tool box modules to Mac, Rmac, Basic, C, or other Pascal compilers, so even if you don't program in Turbo Pascal you can still learn how to build a C128 CP/M tool box.

The SG Tools tool box was created, so the programmer can access all the C128's features under CP/M and create applications ready for the 90's! We will start by covering all the low level code needed to access the I/O chips, VDC, and memory. The second installment will use these low level modules to create page flipping, fast write, and window modules to drive the 80 column screen. The final installment will cover calling CP/M's BDOS functions, building applications and customizing your new tool box.

WHY USE TURBO PASCAL?

* Pascal is designed to be a compiled language unlike interpreted BASIC and the resulting executable code is many times faster and usually smaller than equivalent BASIC code.

* The Turbo Pascal System has a built in text editor that uses Word Star commands. If you get a compiler or run-time error the compiler puts your cursor at the offending statement in the editor!

* Turbo Pascal compiles, assembles and links programs to memory or a CP/M stand alone COM file all in one step. COM files can be sold or distributed. Many CP/M versions of BASIC, Pascal, C, and Small C require a separate proprietary run-time system file and/or have a separate slow assemble and link process.

* Source code can be ported to many different types of computers that have Turbo Pascal compilers. I have done this with a text adventure game I wrote back in 1987. It was originally written for MS DOS, but it compiled and ran without modification under CP/M. If you want to learn Object-Oriented programming you can step up to Turbo Pascal 6.0 for Ms-Dos with Object-Oriented extensions and Borland's own Object-Oriented tool box called Turbo Vision. All the stuff you learn with Turbo Pascal and CP/M you can take with you to Ms-Dos, Microsoft Windows, and beyond.

GETTING STARTED

Here is a list of items you will need to get started:

The CP/M boot disk that came with your C128. I used the May 87 release to develop all my programs.

I suggest you read the article "Supercharging Your CP/M BIOS, CCP, & Bootdisk Utilities" by Randy Winchester in TC128 Issue #28. This will give you helpful information on optimizing your CP/M system as well as covering some good public domain CP/M utilities.

TC-128 on disk. Due to the volume of source code and programs I strongly suggest ordering the TC-128 companion disk. Each disk will include all source code, pre-compiled programs, and bonus programs that can be run with or without Turbo Pascal. This issue will include a 80 column color window demonstration with sound!

Turbo Pascal 2.0 or higher for CP/M. There are a couple of ways you can go to obtain a copy. The cheapest way is to find someone at a CP/M user's group with an original copy of Turbo Pascal 2.0 or higher and buy it for $25.00 or less. I bought a bunch of CP/M stuff from a guy with an Apple II CP/M system that included Turbo Pascal 2.0.

You can also buy the latest and greatest Turbo Pascal 3.1 for CP/M from Elliam Associates (listed at the end of the article). Turbo Pascal 3.1 is $64.95 plus $3.50 shipping in the U.S.. They also carry a large selection of CP/M software. Send $1.00 to the listed address for their catalog.

Some useful utilities include: SID, other 8080 or Z80 disassemblers, MAC, RMAC, Native Z80 Assemblers or a Z80 macro library like the one Commodore gives you with the 1581 version of CP/M.

Some type of file utility program other than PIP like SWEEP (New Sweep) for file maintenance.

Books that most Commodore programmers have like: the "Commodore 128 Programmer's Reference Guide", "CP/M Plus Programmer's Guide", Compute!'s "128 Programmer's Guide", books on programming the 8080, Z80 and 6502, the December 1988: Volume 9, Issue 2 of the "Transactor" has a good C-128 CP/M Plus memory map with comments, "Programming the Z80" in the August 1986 Issue 38, Volume 4, COMPUTE!'s Gazette, or any other related material.

STRUCTURE OF TOOL BOX

Each module of SG tools is a separate include (.INC) file. To add modules to your main program use Turbo Pascal's include file compiler directive: [$I MODULE.INC] where "module" is the name of the tool box module. It is best to keep the modules as simple as possible, so you do not have a lot of uncalled procedures which generate dead code at compile time. You also want to be able to reuse modules for many different applications. Borland did not add units and smart linking until Turbo Pascal 4.0 for Ms-Dos. So make sure you call all or most of the procedures in a module. Some modules may depend on others being previously defined. This will become clearer in the next issue when we cover fast string writes and windows. Now that we have all the basic information out of the way let's start programming some I/O ports!

ACCESSING I/O PORTS WITH THE Z80

As a C-64 or C-128 programmer you know how to access the Sid, Vic and Cia chips with 65XX "lda" and "sta" instructions or Basic "peek" and "poke" commands. But the Z80 works a little differently with the C128's I/O block. You cannot access the I/O chips with Z80 LD type instructions, but the Z80 can communicate with the C128's I/O chips via IN reg,(C) and OUT (C),reg instructions. We will be using Turbo Pascal's Inline method to insert 8080 and Z80 machine code. You can easily create in-line machine code by writing your Assembler modules with MAC and using the output listing. Just load the listing file in a text editor or Turbo Pascal and use the machine code portion for your in-line code.

The first SG Tools module is called PORT.INC. This allows your applications to access the C-128's Input and Output chips.
(see the listing at the end of the article)

Problems with Turbo Pascal's PORT ARRAY

If you are an experienced Turbo Pascal programmer you might say that Turbo Pascal already has an I/O array called Port. To access a VIC register you might use BorderColor := Port[$d020]. This would be great if it worked on the C128, but it doesn't. I have found using Turbo Pascal's Port array to read the C128's I/O block returns false values. Like any other hacker I wrote a simple program using Turbo Pascal's Port array and fired up my disassembler to find out why this occurs. What I found is that Turbo Pascal uses IN E,(C) and OUT (C),E just like my PortIn and PortOut, so why doesn't it work? If you can find the answer send it to JBEE or myself via GEnie. To be safe we will be using my port routines because I know they work all the time on the C128! I would like to stress that using Turbo Pascal's Port array to write works fine though. For an example of this anomaly compile and run PORT128.PAS or run PORT128.COM on the TC128 disk. PORT128 will read and display various I/O locations on the C128 with My PortIn and Turbo Pascal's Port array.

ACCESSING THE VDC

I was programming late one night on a IBM PC in Dos and thought how great it would be if CP/M had the nice user interfaces like Dos, Windows 3.X, Geos, native 64, and 128 mode applications. You are probably saying to yourself that CP/M is too slow to handle the windowing and full screen updates required in a graphic operating enviroment. Well, your right, but who needs CP/M to read and write the screen? I'm sure you have displayed characters directly to screen memory in native 64 or 128 mode instead of using the Kernal's CHROUT routine because it is much faster. Most of the professional tool boxes for IBM PC DOS use direct screen I/O instead of DOS or BIOS calls for the same reason. Direct screen I/O can also be applied to CP/M on the C-128.

Our next SG Tools module is called VDC.INC. It allows you to read and write VDC registers. Once again we will be using Inline code for speed. (see the listing at the end of the article)

ACCESSING MEMORY
Your application can easily access memory with Turbo Pascal's Mem array.

To read memory use: Buffer := Mem[$80];
To write memory use: Mem[$80] := Buffer;

Buffer is a byte type variable. Now that we have a way to access the C128's I/O ports, VDC and memory let us see how fast they are.

TIMING EVENTS IN
TURBO PASCAL
How fast is fast? Terms like "Turbo Charging" and "Supercharging" do not really tell you how fast a certain procedure is, so I created a timing module called "timer.inc". It uses CIA #2's "time of day" clock and does not affect CP/M's system time. The program IO128.PAS compares various I/O operations by timing how long it takes to do 10,000 operations of each. Total time in seconds, tenth of seconds and operations per second are displayed. My VDC routines are 30% faster than using Turbo Pascal's Port array! My PortIn and PortOut are much slower that Turbo Pascal's Port array, but they always work on the C-128. Turbo Pascal's Port and Mem are the fastest I/O methods of all. This is not surprising considering the overhead of calling a procedure for port I/O compared to using the port array. The TIMER.INC file depends on PORT.INC being previously defined. (see the listing at the end of the article)

FINAL THOUGHTS
We have covered quite a bit of ground with this first installment! I encourage you to modify the example programs and experiment with the C-128's I/O chips in CP/M mode. If you're a hacker you may be able to tweak my modules for greater speed. If you do, add them to IO128.PAS, so we can compare methods. The modules provided here are fast enough to drive windows, page flipping, 80 column video games, or whatever other applications you can dream up!

If you have any questions or ideas, send me a message on GEnie at address "s.goldsmith2". Until next time...

Mail address:
Elliam Associates
PO Box 2664
Atascadero, CA 93423

UPS address:
Elliam Associates
4067 Arizona Ave.,
Atascadero, CA 93422
(805) 466-8440

(DR.OCTAL TIPS - CONTINUED PG #25)
your hardwired #11 devices. Here is a hardware method I learned from a friend:
*Find u113
*Cut pin 1 at the point where it enters the board using an exacto knife. Put a spst switch between pin 1 of the ic and where pin 1 used to enter the motherboard. You can mount the switch anywhere, preferably under the drive led. There is a lot of room on the C-128d's front panel.
*With the "ATN" line out of the serial loop (switch open), the device is effectively off the bus and will not be recognized, no matter what device # you use for it.

NEWS RELEASES:
Note that news releases are NOT an endorsement of the product. They are news tidbits available for your use in case you want to pursue something further that sounds interesting.

Bookdisks: "Non-Abberrational Capitalism:Template for a New World Order" is a book on disk(s) that comes on two double-sided 1541 diskettes. The diskettes also include a functional text reader which allows the reader to print out the text files. Cost is $5 and available from: Paperless Press, 109-47 117th St, Ozone Park, NY 11420

Video Board: The Auxiliary Video Board (AVB) is basically a C-128 80 Column VDC chip with 64K Dram to be plugged into your expansion port. AVB blank PCB, assembly and board options, parts list, and examples are available for $34 plus $5 S&H from: Gregory Clark, PO Box 660366, Sacramento CA 95866

PROGRAM NAME: PORT.INC
To read a port you use: BorderColor := PortIn ($d020);
To write a port use: PortOut ($d020,BorderColor);

BorderColor is a byte type variable.

```
function PortIn (MemLoc : integer) : byte;

var

  Load : byte;
  RegPair : integer;

begin
  RegPair := MemLoc;
  Inline (
    $ED/$4B/RegPair/        [ld     bc,(RegPair)]
    $ED/$58/                [in     e,(c)]
    $7B/                    [mov    a,e]
    $32/Load                [sta    Load]
  );
  PortIn := Load
end;

procedure PortOut (MemLoc : integer;
                   Value  : byte);

var

  Store : byte;
  RegPair : integer;

begin
  RegPair := MemLoc;
  Store := Value;           [Store = byte value to store]
  Inline (
    $ED/$4B/RegPair/        [ld     bc,(RegPair)]
    $3A/Store/              [lda    Store]
    $5F/                    [mov    e,a]
    $ED/$59                 [out    (c),e]
  )
end;
```

PROGRAM NAME: VDC.INC
To read a VDC register use: FgBgColor := ReadVDC (26)
To write a VDC register use: WriteVDC (26,FgBgColor)
FgBgColor is a byte type variable.
I have also included TPVDC.INC which uses TP's Port array instead of Inline code.

```
function ReadVDC (Reg : byte) : byte;

var
```

```
PROGRAM NAME: VDC.INC (continued from previous page)
  VDCReg : byte;

begin
  VDCReg := Reg;
  Inline (
    $01/$00/$D6/          [lxi      b,$d600   point BC to $d600]
    $3A/VDCReg/           [lda      VDCReg    VDC reg]
    $ED/$79/              [outp     a         VDC reg to read]
    $ED/$78/              [inp      a         get VDC status]
    $CB/$7F/              [bit      7,a       test status bit]
    $28/$FA/              [jrz      rep       until bit high]
    $0C/                  [inr      c         point BC to $d601]
    $ED/$78/              [inp      a         read VDC reg]
    $32/VDCReg            [sta      VDCReg    stash result]
  );
  ReadVDC := VDCReg
end;

procedure WriteVDC (Reg   : byte;
                    Value : byte);

var

  VDCReg,
  VDCValue : byte;

begin
  VDCReg := Reg;
  VDCValue := Value;
  Inline (
    $01/$00/$D6/          [lxi      b,$d600   point BC to VDC]
    $3A/VDCReg/           [lda      a         VDC reg]
    $ED/$79/              [outp     a         put reg in VDC]
    $ED/$78/              [inp      a         get VDC reg]
    $CB/$7F/              [bit      7,a       check status]
    $28/$FA/              [jrz      Rep       until bit high]
    $0C/                  [inr      c         point BC data reg]
    $3A/VDCValue/         [lda      b         value to store]
    $ED/$79               [outp     a         put value in VDC reg]
  )
end;


PROGRAM NAME: TIMER.INC
To read time use: GetTOD;
Values are stored in Global tod variables.

To set time use: SetTOD (Hours,Mins,Secs,Tens);

const
  cia2TODTen = $dd08;
  cia2TODSec = $dd09;
  cia2TODMin = $dd0a;
  cia2TODHrs = $dd0b;
  cia2ConRegB = $dd0f;
```

```
var

  todTen, todSec, todMin, todHrs : byte;

procedure GetTOD;

begin
  todHrs := PortIn (cia2TODHrs);
  todMin := PortIn (cia2TODMin);
  todSec := PortIn (cia2TODSec);
  todTen := PortIn (cia2TODTen)
end;

procedure SetTOD (hh,mm,ss,tt : byte);

begin
  PortOut (cia2ConRegB,$00);
  PortOut (cia2TODHrs,hh);
  PortOut (cia2TODMin,mm);
  PortOut (cia2TODSec,ss);
  PortOut (cia2TODTen,tt)
end;
```

MUSIC PROGRAM USING PORTOUT
PROGRAM NAME: MUSIC.COM
This simple program uses PortOut to play music with the SID chip.

```
[
SG Tools (C) 1992 Parsec, Inc.

Music is a short music demo using PortOut to access the SID chip.
]

program Music;

[$B-,R-]

[$I PORT.INC]

const

  Sid = $d400;
  Music : array[0..50] of integer =
  (
  25,177,250,
  28,214,250,
  25,177,250,
  25,177,250,
  25,177,125,
  28,214,125,
  32,94,750,
  25,177,250,
  28,214,250,
  19,63,250,
  19,63,250,
  19,63,250,
```

PROGRAM NAME: MUSIC.COM (continued from previous page)

```
  21,154,63,
  24,63,63,
  25,177,250,
  24,63,125,
  19,63,250
  );

procedure ClearSID;

var

  I : byte;

begin
  for I := Sid to Sid+24 do
    PortOut (Sid,0)
end;

procedure Run;

var

  I : byte;

begin
  PortOut (Sid+5,9);    [attack/decay]
  PortOut (Sid+6,0);    [sustain/release]
  PortOut (Sid+24,15);  [maximum volume]
  for I := 0 to 16 do
  begin
    Write ('.');
    PortOut (Sid+1,Music[I*3]); [high freq]
    PortOut (Sid,Music[I*3+1]); [lo freq]
    PortOut (Sid+4,33);         [gate sawtooth]
    Delay (Music[I*3+2]);       [note duration]
    PortOut (Sid+4,32);         [release sawtooth]
    Delay (10)
  end;
  ClearSID
end;

procedure Init;

begin
  ClrScr;
  ClearSID;
  Writeln ('Music (C) 1992 Parsec, Inc. - All Rights Reserved');
  Writeln;
  Writeln ('Music will play a short song with the SID.');
  Writeln
end;
begin
  Init;
  Run
end.
```

****
*   CLASS(Y) ADS   *
****

These ads are free for TC128 subscribers and advertisers. Commodore BBS and UG listings are free to all.

All people with POBs have to submit a street address to Parsec with a matching night time telephone number (we will not release the street address to anyone UNLESS there is an unresolved problem with your ad). For sale and wanted ads must include either an address (street or POB) OR a telephone with the time to call. An example is, (1-508-745-9125 EST 9-5 answering machine) so people can easily contact you.

All ads will run until you ask for them to be removed or until they are "bumped" off the listing by a newer ad. The date the ad was 1st run will be expressed as 920105 (year 92, 1st month, fifth day).

ALL ADS *MUST* be submitted on either 1541 or 1581 disks as either PetAscii or straight Ascii sequential text disk files, no exceptions! If you can send matching hardcopy it would be appreciated. We will take ads from subscribers through e-mail. We are not responsible for anything including typos. BUYER BEWARE!

The guidelines to buying through the mail are unless you know the person well:
1) Buyers and sellers should insist on COD, ship by UPS (if possible), cash or money order!
2) Get a telephone number!
3) Try to have some fun horse trading! :)

FOR SALE HARDWARE

*920201 - Clay MacDonald 303-927-4498
C128D, JD V6.0, fan, all docs, like new $300.00, 1581, JD, like new $100 - both units have device switches. QBB 64 - $50, 1764(512k) - $100, Many other items

*920201 Alex Dundek 612-645-6636
1571 - new in box - best offer over $150

*920221 Bill Golden, PSC 76 Box 2629 Army, APO AP 96319-2629
1581 $135, 1571 $125, 1541 device #8 $65

FOR SALE SOFTWARE

WANTED HARDWARE

*920220 John Stewart 602-378-6316
BI Buscard II IEEE Interface
CSI 425 or Interpod serial to IEEE interface
Users and/or Maintainence Manual MSD drive

WANTED SOFTWARE

*920220 John Stewart 602-378-6316
Catalog program by Intergrated Software Systems
Masterdisk, Masterdual, Super-Masterdisk(preferred)

SI/FI - MISC

*920709 Wanted - a complete set of the Transactor magazines. Willing to pay $2.50 per issue - contact Parsec Inc. by phone, mail, or e-mail.

BBS LISTINGS

USER GROUP LISTINGS

*920201 - Basic Bits Commodore Group, PO Box 447
PO Box 447, North Ridgeville OH 44039

*920225 - CBM Users Group of Lewis County
c/o Al Kistenmacher 2476 PeEll-McDonald Rd.
Chehalis, WA 98532

*920310 - Boise Area Commodore Users Group
3213 Kelly Way, Boise, ID 83704-4620

*920626 - "Meeting 64/128 Users Through the Mail"
1576B County Rd 2350 E. St. Joseph, IL 61873
is a correspondence User Group 6 years old with 240 members. Dues $12. Write for more information.

Twin Cities 128 Subscription Information

Magazine only:
6 issues a year - $20 for the US
6 issues a year - $26 all others

Magazine with companion disks:
6 issues a year - $36.50 for the US
6 issues a year - $46.50 for all others

User Group Rates (minimum of two subscriptions)
Magazine only:
6 issues a year - $17 for the US
Magazine with companion disks:
6 issues a year - $26.00 for the US

For the US:
The cost to add the companion disk to TC128 from a current subscription starting with issue #32 is $4.00 for each remaining issue.

For all others:
The cost to add the companion disk to TC128 from a current subscription starting with issue #32 is $6.00 for each remaining issue.

Effective 920201 we will acknowledge all (re)subscriptions, change of addresses, and subscription inquires by either a letter or postcard.

How to read your address label:
1st line: ## # ABC.XXXXXX.XXX

The first number is the issue on which your subscription ends and the second number following it will have a "1" if you are owed the companion disk. The third set of letters & numbers is your customer ID #. This may not be on all labels

2nd line:Attention line - usually not used

3rd line:Your Name
4th line:Your Address
5th line:City, State, Postal Code
6th line:Country (not written on US labels)

Make checks payable in U.S. funds to:

PARSEC INC
PO BOX 111
SALEM MA 01970-0111
USA

GEOS MACHINE LANGUAGE PROGRAMMING ON THE 128: "THE TOOLS OF THE TRADE"

by Robert A. Knop Jr.

I. INTRODUCTION

Why would the prospective C-128 machine language programmer want to program under Geos? For one, the Geos128 operating system provides an excellent user interface, which you might want to take advantage of in your own programs. More importantly, under Geos it is a lot easier to write powerful user-friendly programs than it is on a cold C-128. The Geos Kernal supplies routines for icons, menus, dialogue boxes, graphics, text, and more, all available to an application, which reduces the burden on the application's programmer. No longer do you have to worry about "how" to implement a pull-down menu; tell Geos what menus to put on the screen and what to do when one is selected, and Geos takes care of the rest.

II. YOUR PRIMARY PROGRAMMER'S TOOL: GEOPROGRAMMER

In 1987, Berkeley Softworks published geoProgrammer, a complete software development system for Geos64. GeoProgrammer has three parts.

First is geoAssembler, which does the dirty work of converting your assembly language source (written with geoWrite) into relocatable machine language code. GeoAssembler is an extremely powerful label-based assembler, complete with local labels, macros, conditional assembly, an impressive expression evaluation facility, and more.

The second application, geoLinker, allows you to build a program out of several separate modules, each independently assembled with geoAssembler. GeoLinker also allows you to create VLIR applications. A VLIR application consists of one memory resident module, as well as several swap modules which are loaded from disk as needed. This means you can write programs longer than the memory space available. For instance, both geoWrite and geoPaint are VLIR applications.

The third and perhaps most impressive part of geoProgrammer is the geoDebugger. With this, you can set break points in your code, step through your code, examine memory, and track down lingering bugs in your program. The original geoDebugger came in two forms: the powerful label-based SuperDebugger, available only if you have a RAM expansion unit, and the scaled down mini-Debugger.

Plus, aside from the three major applications, a number of goodies come with the geoProgrammer package. This includes a complete file of the Geos symbols which you can include in your source code, as well as a file of useful macros for common operations like loading a memory location with an immediate value. Full source code to three sample do-nothing applications is included. Which is useful in demonstrating the structure of the various Geos application types.

Shortly after the release of the first version of geoProgrammer, advertisements started to appear for the soon-to-be-released geoProgrammer2.0, which promised support for the Commodore 128. Unfortunately, geoProgrammer2.0 was never published, but was left unfinished as Berkeley Softworks (now GeoWorks) began to develop PC-Geos for MS-DOS machines. Thus, it would seem that potential Geos128 programmers were out of luck. If they wanted to develop applications for Geos128, they would have to do so under Geos64. Fortunately, some Geos programmers were not content to let things sit the way they were.

PATCHING GEOPROGRAMMER

As it is, the three geoProgrammer applications, geoAssembler, geoLinker, and geoDebugger, refuse to run under Geos128, due to a flag in their headers which says that they can only be run from Geos64. If that flag is changed, geoAssembler and geoLinker "almost" run under Geos128. More precisely, they do run, but things are slightly out of kilter in 80 columns; and if one is going to geoprogram on the 128, it would be nice to take advantage of the 128's two megahertz fast mode available only in 80 columns.

All is not lost; at least two people have developed patches for the geoProgrammer applications which allow them to be run neatly from Geos128. The patch I shall discuss, "geoProgrammer Patch 2.1," was written by Robert J.G. Norton; patches for the programs have also been written by Jean F. Major.

Once the patch has been applied, geoAssembler and geoLinker both run quite nicely in 40 or 80 columns under Geos128. The patched geoDebugger can only debug 40 column applications, and the patched SuperDebugger trashes the 128's RAM reboot code. However, one need not concern oneself with this, due to the existence of geoDebugger2.0, which I shall discuss shortly.

How does one perform this patch? The patch is designed to operate on an already installed geoProgrammer. Moreover, it is wise to perform the patch on a copy of your geoProgrammer applications, rather than the original disk. This is not a problem if you have both Geos64 and Geos128, and if both have the same Kernal ID. If, when installing one Kernal, you told it to match itself to an application installed with the other Kernal, then both Kernels will have the same ID. If you are unsure whether both have the same Kernal ID, try running a Geos64 program, say geoPaint, from Geos128. If it doesn't complain, then you should be OK. In this case, you can simply install the programs with Geos64, copy them to another disk, and perform the patch on the second disk with Geos128.

If you don't have Geos64, don't despair. In order to install the programs with Geos128, you must modify one byte in the header of each application. To do this, you will need a disk sector editor (such as Disk Doctor 128, or DiskMon). Sector editor in hand, look at track 18, sector 1 ($12, $1) of your geoProgrammer disk. You should see the first block of the disk's directory. Locate the directory entry for (say) geoAssembler. That directory entry will look like (sample DiskMon output):

```
>00b00 12 09 83 0a 04 47 45 4f
       41 53 53 45 4d 42 4c 45
       :.....geoassemble

>00b10 52 a0 a0 a0 a0 0a 11 01
       06 57 0b 02 0e 2f 56 00
       :r ....w.../v.
```

Immediately following the 16 character file name (padded with $a0's) is a track sector pair. This is the track and sector of the header record for geoAssembler- in this example, track 10, sector 17 ($0a, $11). Read that sector. The 96th ($60) byte of that sector should contain the value 128 ($80).

This is the flag which indicates that geoAssembler cannot be run from Geos128. Change this byte to 0, which indicates that the application can be run from Geos128 in 40 columns. Be very careful when doing this; a misstep could scramble your geoProgrammer disk!

Repeat this procedure for geoLinker. Once finished, you should be able to install both programs with Geos128. Note that if you are using a 1571 disk drive, you must temporarily configure it (using, of course, Configure) as a 1541 for the installation to work.

Now that you have installed the geoProgrammer applications, copy them to a second disk; do not patch your original copy of geoProgrammer. Then copy the geoProgrammer Patch 2.1 program to that disk. Simply double click on the Patch program, follow the instructions the program gives you, and you will have copy of geoAssembler and geoLinker that can be cleanly run from Geos128! At this point, in order to distinguish the patched applications from the originals, you may want to rename them to "GEOASM 128" and "GEOLINK 128," or something along those lines.

GEODEBUGGER 2.0

Although geoProgrammer 2.0 was never released, it was reasonably close to being finished. In particular, version 2.0 of the Debugger had been nearly completed. This Debugger features full 128 support. One impressive feature is the BackRAM debugger. With this, one does not need an RAM EXPANSION UNIT to run the SuperDebugger! The SuperDebugger loads itself into the "other" 64K of the 128's RAM, giving the full power of the SuperDebugger without taking away any of your application space.

Much to the delight of Geos programmers, GeoWorks (the new name for Berkeley Softworks) released the 2.0 debugger for informal distribution as an upgrade for geoProgrammer owners. What this means is, if you own geoProgrammer, then you are entitled to upgrade to the 2.0 Debugger by downloading it from Q-Link, or wherever else you can find it. (Note, however, that legally you must purchase geoProgrammer before obtaining a copy of geoDebugger 2.0.)

Although geoDebugger 2.0 works quite nicely as-is with the 128, this has not stopped people from writing patches which enhance the functionality of the debugger. For instance, in order to get the BackRAM debugger, you must hold down the space bar while geoDebugger loads. Since forgetting to do this loads the SuperDebugger into your ram expansion unit, trashing the ram reboot code, I wrote a patch (backdebug.patch) which forces geoDebugger 2.0 to load the BackRAM debugger. Other patches which improve upon the 2.0 debugger exist, such as Jean F. Major's "Debugger Update."

COPING WITHOUT GEOPROGRAMMER

If you don't have geoProgrammer, there do exist (at least) two shareware Geos assemblers: geoCope, by Bill Sharp, and the Springboard Assembler by Jim Holloway. Both assemblers produce Geos code, and work with Geos128, although only in 40 column mode. Each of these assemblers is a single unit; neither provides, nor requires, a linker.

GeoCope comes with its own editor and assembler, as well as with some sample files. Since the assembler takes files from the editor, you can't use the full power of geoWrite to edit your source code. The editor, while usable, isn't nearly as powerful as geoWrite, and is at times somewhat hard to use. For instance, you can't use the mouse to position the cursor, and I wasn't able to figure out how to move more than a line at a time within a page. The assembler is label based, allows the use of multiple source files through the .include directive, and does support VLIR files. It is limited to 8K of object code. One thing nice about this assembler is that it gives more information about the program being assembled than does geoProgrammer.

The Springboard Assembler takes as its source geoWrite files. It too is label based, although with certain commands (e.g. lda, sta) you can't use forward referenced labels. This makes it difficult to put data blocks after the end of your code. Like geoCope, the Springboard Assembler lets you have separate source code modules. However, there is no .include directive, so any equates you use will have to be typed directly into the source file that needs them.

If you are serious about programming in Geos, you are going to want to get a hold of geoProgrammer. However, if you just want to dabble in programming Geos, or experiment with some small things, then either of these assemblers could be useful for you. The shareware price of geoCope is $15; of the Springboard Assembler, $5.

III. TEXTWARE

The manual which comes with geoProgrammer is excellent. It clearly and completely describes the operation of all three geoProgrammer applications. However, it assumes prior knowledge of both assembly language and programming under Geos.

"The Official Geos Programmers Reference Guide" (PRG) from Bantam Books is a good book for learning how to program under Geos. It assumes knowledge of 6502 assembly language, and a user's familiarity with Geos, and from there quickly gets you up to speed on the basics of programming in Geos.

It describes the concept of event-driven programming, and discusses how Geos uses this. It goes on to describe how to create icons and menus, and use them within your programs. Following a chapter on how to convert applications to Geos format (which can be ignored by geoProgrammer owners, since geoProgrammer takes care of all of that for you), the book goes on to discuss using graphics and text with Geos, as well as more advanced topics such as processes, dialogue boxes, the file system, input drivers, and printer drivers.

Although occasionally obtuse, and sprinkled with a truly amazing number of typos, the PRG is very useful for the programmer who wants to learn how to program Geos. Unfortunately, the PRG is now out of print, but you may be able to find a copy at a local used bookstore, or convince a friend who no longer needs it to sell it to you.

When GeoWorks decided to make the move into MS-DOS software, they stopped work on a second Geos programming manual. Although they maintain the copyright, GeoWorks has released the latest draft of this manual as the freely distributable "Hitchhiker's Guide to Geos". You can obtain this manual by copying it from a friend, or by sending $25 to GeoWorks.

Sporting an alphabetical list of all the Geos Kernal routines, as well as chapters of updated programming information and more in-depth technical information about Geos64, Geos128, and

Apple Geos, the Hitchhiker's Guide is a better reference than the Programmer's Reference Guide. It is more complete, and its information is more up-to-date and thus more reliable. Moreover, it has some information specific to Geos on the Commodore 128, something lacking from the PRG. Also, in spite of being a draft rather than a final work, it has far fewer typos than the PRG!

Before geoProgrammer or the PRG came out, an individual by the name of Alexander Boyce disassembled and deciphered the entire Geos64 kernal. The result of his efforts is his shareware "Geos Programmer's Reference Manual" (not to be confused with the Official PRG by Berkeley Software). This manual contains succinct and clear documentation of the Geos Kernal routines, as well as some general informational topics. The information is not current (it was written long before Geos128 v2.0), and because Boyce had to invent his own symbol names, his labels for routines are completely different from the Geos standard. (If you can get your hands on a copy of Volume 9, Issue 3 of the "Transactor", you will see that Francis Kostella has compiled a complete cross reference between geoProgrammer symbols and Boyce's symbols.)

Boyce's manual is interesting as a different writer's explanation of the Geos Kernal; it is also impressively accurate, although he does fail to note certain restrictions and conventions that can be found in the official literature. (For instance, he doesn't note certain parts of zero page which are off limits for Desk Accessories.) He does address a few topics not found in either the PRG or the Hitchhiker's Guide (for example, the geoPaint file format). It can be found, among other places, in archived PETASCII format in the GEnie FlagShip library under the names "GeosTECHREFx.ARC", where x is 1, 2, and 3.

Finally, the standard programming references for the 128 can come in handy. This includes your favorite book on 128 assembly language, as well as other classics such as Bantam's "Commodore 128 Programmer's Reference Guide", Compute! Books' "Mapping the 128", and ABACUS's "128 Internals".

IV. HARDWARE

Although in theory you could geoprogram the C-128 with just a C-128, Geos128, and a 1541 disk drive, practically speaking, at the very least you will need a 1571 or a second drive for anything but small projects. Although an application may only take up (say) 10K on a disk, its numerous geoWrite assembly source files can become quite large.

As with anything in Geos, the presence of an ram expansion unit makes everything much faster, smoother, and easier. With a sufficiently large ram disk (and for a large enough project, a ram1571 may not be enough!), you can put geoProgrammer, your source files, intermediate files and final assembled and linked programs all on a ram disk (remembering to frequently copy modified source files to a disk). Or, perhaps you would prefer to use a shadowed drive. If pinched for space, you can keep the applications and output files on a RAM DISK, and store the valuable source files on a floppy. In any event, a ram expansion unit is almost a necessity if you want to do much geoprogramming and stay sane.

V. CONCLUSION

Geoprogramming is not just for the 64. Thanks to the efforts of some dedicated Geos128 programmers, you can take advantage of the full power of geoProgrammer completely from your Commodore 128. If you have programmed in assembly language before, you will be surprised how little effort it takes to produce very impressive results under the Geos operating system. For a true programmer, there is nothing like the satisfaction of seeing a complete, powerful, user-friendly program ... with your name in the Info box!

The freely-distributable software discussed in this article such as geoProgrammer Patch 2.1, geoCope, and the Springboard Assembler, as well as geoDebugger2.0, can be found on Q-Link, GEnie, and the internet ftp site milton.u.washington.edu. Also check other on-line services, and BBSs local to your area.

Editor's note: you can also find these files on Parsec's public domain disks Geou 13, Geou 14, Geou 15, and Geou 16.

MEMO WRITER & CALENDAR

For CMD Hard Drives and RamLinks

by Ronald Robert

INTRODUCTION

When I first got RamLink I wrote a two line program to read the hard drive clock and write the time and date to the screen on power up. Then I thought gee wouldn't it be nice if I could leave myself reminders and have them list on power up too. Well having a 128 I wanted it to work in all modes and be a relatively short program. The following is what I came up with. I used some Jiffy Dos commands in the programs so if you want to use them without Jiffy Dos (why would anyone have a CMD hard drive and not have Jiffy Dos ?) you'll have to make some changes. By the way it also works with the Ramcard II time clock. The two programs are written in basic so they can be easily customized.

MEMO WRITER

This is a text editor that creates a list of up to 12 things to do. The list is stored by date and used by the calendar program. The first few lines set the storage device number and default date, they can be changed as required. The program will run in 64 or 128 mode in either 40 or 80 column mode. It starts by asking for the memo date, this is the date that you want your list to be displayed on. Once you input the date the program checks for an existing memo for that date. If one exists, it is loaded into the editor and is displayed. The current memo date is shown at the top of the screen then the 12 lines of the things to do list. Below that is a list of commands, they all use the Commodore key and a letter, they are as follows:

C -- Change line:

This command will let you type the information on lines 1 thru 12. You will be asked which line #, then you can type in the information. The line can only be 36 characters long so don't type past the arrow. If you need more room use the next line to complete your note. When you hit return the line will be added to the list.

D - Delete a line:

This will delete a line or a range of lines. Ex. (3 to 3) will delete line three. (3 to 5) will delete lines 3, 4 and 5.

I - Insert a line:

Will move the lines so you can put a note between two existing lines.

S - Switch a line:

Can be used to exchange the positions of 2 of the 12 memos.

P - Print a list:

Dumps the list of the 12 things to do to device 4, the printer.

W - Write a file:

Saves the list to the storage device. NOTE: It will automatically overwrite an existing file of the same date.

R - Read file and Reset memo date:

This will change the memo date but when it does it will check to see if there is already a memo for that date. If there is the existing memo will be loaded and the memo on the screen will be overwritten.

Q - Quit:

Although the Commodore Q command is not listed in the command box it is available and will reset the computer.

NOTE: If you want to put leading spaces in a memo to indent a line for instance, you must use shifted spaces.

CALENDAR

Now that you have a list of things to do you need a utility to show it to you on the right day. Well this is it. The first few lines set the time clock device # and the storage device # (should be the same as the storage device # in memo writer) these can also be changed as required.

This utility can be run when you want to check your memos or set up as an autoboot file and will display the date and time when ever you start your computer, along with any messages that there are for that particular day. If there are any messages they will be listed then you will be given the option to delete them. If you say yes (Y) the file will be scratched and the screen will clear. The next time you turn the computer on that day there will be no messages. If you say no (N) the screen will clear but the file will stay intact so that the next time you power up the message will be shown again.

I use the programs in ramlink (in the default partition) and use the ramcard II clock so that is what the programs are set up for. If you want to use the hard drive and hard drive clock, you can, by changing the variables as explained in the first few lines of both programs. I used the hard drive clock before I upgraded to ramcard II and it worked fine a little slower but fine. This program also runs in 64 and 128 40 and 80 column modes.

The lines to change in "Calendar.bas" for your device numbers are on line # 160, the variables "tc" and "sd".

The lines to change in "Memo.writer.bas" for your device number is on line # 160, the variables "sd".

TC=Time Clock
SD=Storage Device
AD$=Default Memo Date

PROGRAM NAME: CALENDAR.BAS

```
eo 100 rem by ronald robert
hh 110 rem copyright (c) 1992 by
id 120 rem parsec inc  pob 111
ao 130 rem salem ma 01970-0111
ja 140 rem program name = calendar.bas
na 150 :
ld 160 tc=08:sd=08
dj 170 rem tc = timeclock device #.
lh 180 rem sd = memo and program storage device #.
em 190 printchr$(147):s=9:ifpeek(215)=128thens=s+20
cf 200 printchr$(14)
pb 210 open15,tc,15
cp 220 print#15,"t-ra"
hp 230 get#15,a$:t$=t$+a$:ifst<>64then230
jb 240 close15
pb 250 dt$=left$(t$,13)
lb 260 mm$=mid$(t$,5,9)
lh 270 dy$=left$(t$,4):gosub460
ed 280 hr$=mid$(t$,15,5)
nh 290 dn$=right$(t$,4)
lm 300 printtab(s)" today is "dy$+mm$
ke 310 printtab(s)" the time is "hr$+dn$
jj 320 @tright$(dt$,8),tc
gm 330 open15,sd,15:input#15,e:close15
dn 340 ife=62thengoto420
ml 350 print"delete today's messages  n"
bh 360 forzq=1to1:printchr$(145);:next
fo 370 forzq=1to23:printchr$(29);:next:input dl$
bc 380 ifdl$="y"thengoto430
lh 390 ifdl$="n"thengoto410
np 400 goto350
cn 410 printchr$(147):printchr$(142)
eh 420 new
md 430 sc$="s:"+right$(dt$,8)
kc 440 @sc$,sd
lh 450 printchr$(147):printchr$(142):new
mh 460 ifdy$="sun."thendy$="Sun.":return
jj 470 ifdy$="mon."thendy$="Mon.":return
gk 480 ifdy$="tues"thendy$="Tues":return
hh 490 ifdy$="wed."thendy$="Wed.":return
na 500 ifdy$="thur"thendy$="Thur":return
nc 510 ifdy$="fri."thendy$="Fri.":return
jg 520 ifdy$="sat."thendy$="Sat.":return
fi 530 goto410
```

PROGRAM NAME: MEMO.WRITER.BAS

```
eo 100 rem by ronald robert
hh 110 rem copyright (c) 1992 by
id 120 rem parsec inc  pob 111
ao 130 rem salem ma 01970-0111
pj 140 rem program name = memo.writer.bas
na 150 :
bh 160 sd=08:ad$="01/11/92"
ah 170 printchr$(14)
ph 180 rem sd is the program and memo storage
       device #.
cn 190 rem  ad$ is default memo date
dm 200 open15,sd,15,"i"
bc 210 jg=1:dim tt$(12)
ih 220 sp$="                                      "
np 230 poke 53280,15:poke 53281,15:poke 646,1
ik 240 d$="[SH/N]":n=1:goto1240
dl 250 rem menu screen / d$ is option choice
ek 260 gosub 400
ed 270 printchr$(147):print"                    "+ad$
ba 280 print"          list of things to do        "
ik 290 fori=1to13:print chr$(17);:next
dn 300 print"       [] commodore key + letter       "
dj 310 print"  [C]=change line   [I]=insert line  "
dd 320 print"  [D]=delete line   [S]=switch line  "
nl 330 print"  [W]=write file    [P]=print list   "
ol 340 print"  [R]=read file and reset memo date  "
fa 350 print:print:printspc(38)"*"
df 360 printspc(17)"dont type past here  *",
mj 370 ifjg<>0then380
al 380 return
```

```
bg 390 rem
if 400 printchr$(19):forzq=1to20:printchr$(17);
       :nextzq
em 410 print"
jj 420 printchr$(19):forzq=1to20:printchr$(17);
       :nextzq
dn 430 return
no 440 rem display
jl 450 printchr$(19):print:print
dl 460 for x=nton+11
nd 470 print right$(str$(x),2);" ";tt$(x)+right$
       (sp$,36-(len(tt$(x))))
id 480 next:printchr$(19)
hj 490 return
kj 500 rem main menu
gk 510 gosub260:jg=1
fo 520 gosub450
nc 530 getd$:ifd$=""then530
jb 540 if d$="[C=/C]"then630
cg 550 if d$="[C=/D]"then700
cf 560 if d$="[C=/I]"then840
ik 570 if d$="[C=/S]"then950
jh 580 if d$="[C=/W]"then1210
jl 590 if d$="[C=/R]"then1210
oa 600 ifd$="[C=/P]"then1280
nm 610 ifd$="[C=/Q]"then1380
lk 620 goto 530
ln 630 gosub 400
kg 640 input" what number";wn
ck 650 if wn<1 or wn>12 then 690
nl 660 gosub 400
pa 670 input" ";tt$(wn)
bg 680 if len(tt$(wn))>36 then tt$(wn)=""
pk 690 goto 510
ad 700 gosub 400
aa 710 input" from what number";ff
ef 720 if ff<1 or ff>12 then 830
cb 730 gosub 400
ke 740 input"   to what number";tn
oo 750 if tn<ff or tn>12 then 830
el 760 for x=ff to tn
nl 770 tt$(x)=""
jb 780 next
bl 790 if tn=12then830
ga 800 for x=tn+1to12
hn 810 tt$(x-(tn+1)+ff)=tt$(x)
lj 820 next
ih 830 goto 510
ja 840 gosub 400
hj 850 input" what number";wn
no 860 if wn<1 or wn>12 then 940
ko 870 gosub 400
kp 880 input" ";tt$
hn 890 if len(tt$)>36 then 940
ok 900 for x=11 to wn step-1
ab 910 tt$(x+1)=tt$(x)
bn 920 next
kl 930 tt$(wn)=tt$
pf 940 goto 510
po 950 gosub 400
mi 960 input" first number";ff
nj 970 if ff<1 or ff>12 then 1020
bm 980 gosub 400
cn 990 input" second number";sn
me 1000 if sn<1 or sn>12 then 1020
kf 1010 ss$=tt$(ff):tt$(ff)=tt$(sn):tt$(sn)=ss$
ef 1020 goto 510
ci 1030 aw$=ad$+",s,w"
lo 1040 print#15,"s0:"+ad$
hm 1050 open8,sd,8,aw$
ol 1060 for x=1to12
dm 1070 if tt$(x)=""then tt$(x)=" "
nn 1080 print#8,tt$(x)
mi 1090 next
ga 1100 close8
ka 1110 goto 510
cl 1120 al$=ad$+",s,r"
fo 1130 open8,sd,8,al$
lo 1140 input#15,a:ifa<>0then1190
ef 1150 forx=1to12
bj 1160 input#8,tt$(x)
on 1170 if st=64then1190
cc 1180 next
lk 1190 close8
pk 1200 goto 510
ad 1210 gosub400
om 1220 ifd$="[C=/R]"thenprint"Read  Memo Date    "+ad$;
ec 1230 ifd$="[C=/W]"thenprint"Write Memo Date    "+ad$;
gp 1240 ifd$="[C=/N]"thend$="[C=/R]":print" set  memo
        date    "+ad$;
co 1250 fori=1to10:printchr$(157);:nexti:input ad$
ed 1260 ifd$="[C=/W]"then1030
eg 1270 ifd$="[C=/R]"then1120
jb 1280 open4,4,7:close4:open4,4,7
go 1290 print#4,chr$(14);"things to do : ";ad$
bj 1300 print#4
og 1310 forx=1to12
dh 1320 iftt$(x)=""thentt$(x)=" "
jd 1330 print#4,right$(str$(x),2);" ";tt$(x)
md 1340 next
kl 1350 print#4,:print#4,:print#4,:print#4,:
        print#4.:print#4,:print#4
fn 1360 close4
kf 1370 goto510
ak 1380 md=peek(215):ifmd=13thensys64738
ci 1390 sys57344
```

A note about letters inside of brackets [] within quotes. [C=/W] = Press the Commodore key with the W key. [SH/N] = Press a shifted N with the quotes.

TWIN CITIES 128 CHECKSUM PROGRAM BY MICHAEL GILSDORF

If you decide to type in programs from Twin Cities 128 magazine, you should first type in and run TC128 Checksum. This program checks your typing by generating a two-letter checksum each time you enter a program line and press the RETURN key. The checksum is displayed in the upper left hand corner (home position) of the 40 or 80 column screen. To check for typing errors, compare the checksum on the screen with the one appearing in the magazine listing. If they're different, then you know you've made a typing error. The magazine listing will show the correct two letter checksum in front of each line number.

TC128 Checksum will detect most typing errors such as transposed characters and misspellings but can on rare occasion be fooled. It uses the line number and value of each character as well as its position on the line to generate the checksum. TC128 Checksum will ignore spaces unless they appear inside quotes or within BASIC keywords. You can use BASIC keyword abbreviations such as ? for PRINT without affecting the result.

TC128 Checksum is also designed to make it easier for you to indent text or enter blank lines. To indent text, simply type the line number, space or tab over to where you wish the text to begin, and then begin typing. This feature will improve the readability of your listings by making portions of your program such as FOR-NEXT loops and DO loops stand out more easily. To enter a blank line, type a line number followed by at least two spaces (or tab) and a shifted character. When the program is listed, only the line number will appear.

```
30 FOR J=1 TO 80
```

TC128 Checksum also has the ability to generate a checksum listing. This listing will show the checksum along side each line number as the program is listed. To begin the listing, type a # in direct mode (without a line number) as the first character on a line. Do not include any additional BASIC commands on the line; otherwise they will be ignored. Once the listing begins, you can use the NO SCROLL key or STOP key to pause or stop the listing as desired. You'll find the checksum listing especially useful if you need to redisplay the checksums and double check the lines you've already entered.

```
OB 30 FOR J=1 TO 80
```

Also, should you decide to submit a program listing to Twin Cities 128 magazine for publication, you can use the # command to save a checksum listing to disk. To create an a SEQ file listing, type:

```
OPEN 2,8,2,"0:FILENAME,S,W": CMD 2
#
PRINT# 2: CLOSE 2
```

The same technique can be used to send the listing to a printer:

```
OPEN 2,4: CMD 2
#
PRINT# 2: CLOSE 2
```

The TC128 Checksum program is listed below. Be sure to save a copy to disk before running it. Once run, it will automatically activate itself.

```
1 print chr$(147);"tc128 checksum  v1.0"
2 print "by mike gilsdorf (c) oct 91": print
3 bank 15: for a=3328 to 3583: read d: poke a,d:
  t=t+d: next
4 if t<>29208 then print "data error": end
5 poke 770,0: poke 771,13
6 print "tc128 checksum activated"
7 print "to list, type:  #": print
8 print "to deactivate, type:"
9 print "poke 770,198: poke 771,77"
10 :
100 data 162, 255, 134,  60,  32, 147,  79, 134
105 data  61, 132,  62,  32, 128,   3, 170, 240
110 data  14, 144,  15, 201,  35, 208,   7, 166
115 data  45, 165,  46,  76, 223,  13,  56,  76
120 data 212,  77,  32, 160,  80, 169,  32, 198
125 data  61, 209,  61, 208,   8, 198,  61, 209
130 data  61, 240, 250, 230,  61, 230,  61,  32
135 data  10,  67, 132,  13, 160,   0,  32,  89
140 data  13,  56,  32, 240, 255,  32, 129, 146
145 data  19,  18,  32,  78,  75,  32, 146,  27
150 data  81,   0,  24,  32, 240, 255,  76, 234
155 data  77, 162,   0, 134, 251, 134, 254,  24
160 data 165,  22, 101,  23, 133, 253, 177,  61
165 data 240,  33, 170, 224,  34, 208,   2, 230
170 data 251, 165, 251,  74, 176,   4, 224,  32
175 data 240,  14, 166, 254, 177,  61,  24, 101
180 data 253, 133, 253, 202,  16, 246, 230, 254
185 data 200, 208, 219, 152, 208,   5, 169,  45
190 data 168, 208,  17, 165, 253,  74,  74,  74
195 data  74,  24, 105,  65, 168, 165, 253,  41
200 data  15,  24, 105,  65, 140,  75,  13, 140
205 data 205,  13, 141,  76,  13, 141, 206,  13
210 data  96, 200,  32, 236,  66, 153,  20,   0
215 data 192,   3, 208, 245, 200, 169,  63, 141
220 data   0, 255,  32,  89,  13, 169,   0, 141
225 data   0, 255,  32, 129, 146,  78,  75,  32
230 data   0, 166,  22, 165,  23,  32,  35,  81
235 data  32, 181,  75, 166,  65, 165,  66, 134
240 data  97, 134,  61, 133,  98, 133,  62,  32
245 data 152,  85, 160,   0,  32, 236,  66, 133
250 data  65, 200,  32, 236,  66, 133,  66, 208
255 data 184, 197,  65, 208, 180,  76,  55,  77
```

PARSEC INC
POB 111
SALEM MA 01970-0111   USA

ADDRESS CORRECTION

BULK RATE
U.S. POSTAGE
PAID
SALEM, MA
PERMIT NO. 188

CUSTOMS INFORMATION:
REGULAR PRINTED MATTER

0 0 0
CINCINNATI COMM. COMPUTER CLUB
ROGER HOYER
5575 PLEASANT HILL RD
MILFORD OH 45150