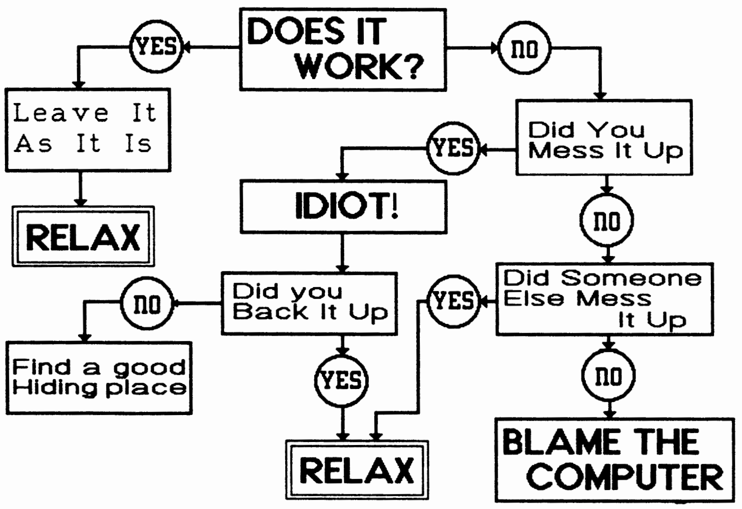
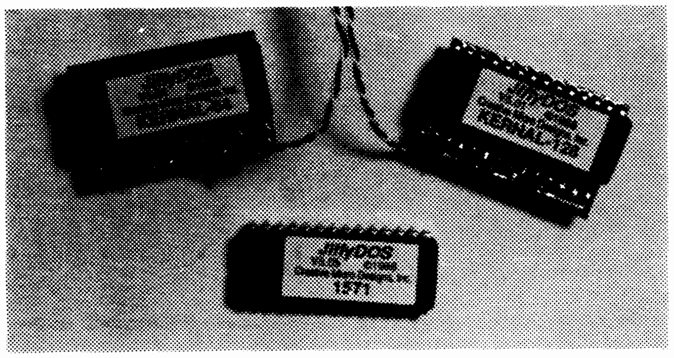
TWIN CITIES 128
THE COMMODORE 128 JOURNAL

ISSUE 23

$2.50 US

NEW C-128 SOFTWARE
Software packages recently received, announced, and rumored:
Pocket Writer 3
Spectrum 128
Newsmaker 128
GEOS v2.0 128
Sketchpad 128
COMAL Powerdriver
Jiffy DOS 128
and more...





Photography & Graphics by Loren Lovhaug, Photo Processing by Bruce Jaeger

Advertisers Index:
Briwall 27 & 31, Brown Boxes 4, Busy Bee Software 22, Free Sprit Software 30, Herne Data Systems 19, Micro Aided Designs 32, Mountain Wizardry Software 6

RUMOR/OPINION/MAYHEM

by Loren Lovhaug

I Have A Dream
Three years ago, in January of 1986, I started Twin Cities 128. My dream was to provide the best C-128 support I could. With the help of a number of gracious people I have been fairly successful. While there is still vast room for improvement, Twin Cities 128 has earned the reputation as the best outlet for C-128 specific information in North America. But now I want to tell you about a new dream, one that involves you and the C-128.

My dream, although certainly not as important as the one held by Dr. Martin Luther King, involves an emancipation of sorts. Just as racial minorities have in the past been prevented from reaching their maximum potential by institutional and structural barriers, we the owners of Commodore 128 personal computers have to a certain extent been prevented from reaching our maximum potential by barriers inherent within the marketplace. Yes, the emancipation of which I speak involves the "freeing" of the Commodore 128 from Commodore.

Sound silly? Let me explain. As we all know, since the C-128's inception Commodore has hesitated to get behind the machine in a big way. This is understandable when you realize that the C-128 has always been thought of in Commodore marketing circles as a "transitional machine"; a product that would tide the company over profit-wise as Commodore followed the rest of the industry into the realm of 16 and 32 bit products. The C-128 was designed and perfected in a mere seven months for a fraction of what it cost to acquire, cost-reduce, and popularize what was then called the "Mazerati" of personal computers. From the beginning the C-128 has been a machine that was designed to be generate a healthy profit with a minimum amount of investment. And given those parameters the C-128 has been a smashing success. The machine has sold an estimated two million units worldwide with virtually no advertising support fueled only by the appetites of first time home computerists and C-64 upgraders. In addition the healthy sales of the C-128 have been accompanied by the very profitable business of marketing add-on peripherals such as the 1571 and 1581 disk drives, the 1700 and 1750 RAM expanders, the 1350 and 1351 mice, the 1902 and 1902a RGBI monitors and the 1670 modem, none of which required huge outlays to design, manufacture, or promote.

However as the C-128 approaches its fourth birthday, Commodore has nearly completed its transformation from a company in dire financial shape that depended on volume sales of 64s and 128s to ward off the creditors into a thriving operation with a rosy outlook based on the higher margins being reaped from the Amiga and PC clone lines. And within this new framework the 128 and its supporting cast of peripherals are increasingly being viewed as excess baggage. Indeed shortages of items and a paring down of eight bit development and technical support activity have over the past year been telltale signs that our machine was/is losing favor in West Chester. As much as we may not like this, there are some strong reasons why such actions are in Commodore's best interest. And like any corporation, Commodore must do what is best for its bottom line.

But before the tears begin forming in your eyes, recall that this essay is about my new dream. A dream which I believe is both positive and pragmatic. I want to engineer a method by which Commodore can "bow out" of the C-128 gracefully. Notice, I did not say: a method by which Commodore can kill the 128 nor did I imply a method by which Commodore can abandon C-128 owners, instead I carefully used the words "bow out" and "gracefully" to suggest otherwise. Here is what I have in mind:

Commodore should help me form a company to provide after market support for the 128.

It has been no secret that Commodore has always relied on third parties such as magazines, user groups, and telecommunications networks to provide usage and technical support for its products. In the case of the C-128 this has been doubly true. So it is quite natural that Commodore "farm out" the long term after market support for the 128. My involvement in this company is also natural since over the past three years, with your help and the help of the many fine contributors to these pages, I have steadily built what is now the defacto vehicle for unadulterated 128 development and support. Over the past three years as North America's only C-128 exclusive publication we demonstrated unequalled success when it comes to re-defining how the C-128 is used and perceived. Our triumphs have included literally the most important ideas and events that have taken place in the Commodore 8 bit industry since the beginning of 1986. These triumphs include: the popularization of desktop publishing with GEOS, the initial and continued disemination of information concerning the C-128 and 1571 upgrade ROMs, the popularization of expanded C-128 video RAM, pre-release information on the 1581 drive and continued coverage including post-release usage and bug fixes, initial and consistent coverage of Basic 8, and most recently the announcement and presentation of detailed information concerning the display and manipulation of "mega-hires" interlaced graphics (including on screen resolutions of 640 x 400 and 640 x 600) on the C-128. And that is not all: Twin Cities 128 has provided online C-128 support on the GEnie telecommunications network, and in October of 1987 we were contracted to manage the C-128 hardware group on Quantum Link. And most importantly we have accomplished this on a ridiculously low budget.

Commodore then must use this company to liquidate its inventory of C-128s and to market C-128 peripherals.

Consider this nightmare: It is the *summer of 1990, and Commodore is liquidating the C-128 and C-128D via traditional liquidation outlets. These include slick cable television operations, second-hand discount and overstock chains, and mail order warehouses. The sales people hawking the machines continuously reread the meaningless information printed on their 4 x 6 inch blurb cards. Their banter includes choice enticements such as: "it can use a disk drive", "it has a 128K memory expandable to 640K", "it can run the C...P...slash...M operating system and is compatible with the model Cee...six...four". While Commodore benefits from the quick cash these outfits provide to clear out their inventory, the public relations effect of this move is disasterous. Long-time C-128 owners become militant. They vow never to buy another Commodore product. They see their purchase falling prey to the pattern established by the Plus/4, C-16, SX-64, refurbed brown 64s, now the 128. They wonder how long it will be before the Amiga 500 and the Amiga 2000 end up being the object of "toot-toots" on late night cable. New owners of the bargain basement 128s are bewildered. Most of them are first time computerists who decided to buy this machine to learn about computers because of the low price. They read on their boxes and in their manuals about non-existent peripherals and software products long since discontinued. Their impression of Commodore is of a company that does not provide long term product support and that cannot be trusted. The next computer they buy will be an IBM or an Apple.*

Consider the alternative: *It is the summer of 1990, eight months earlier Commodore announced that it was committed to supporting its C-128 customers into the 1990s and to help them do just that they have served as a catalyst in forming a company for that purpose. Now Commodore has decided to begin clearing out its inventory of Commodore 128 and 128D*

Continued on page 13

PRICE & PROGRESS REPORT

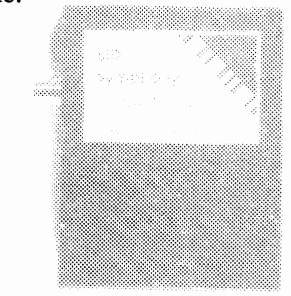
by Loren Lovhaug

128 Games in hiding... You say there are only a few C-128 games? There may be more than you think! Although not widely known, Anton Treuenfels, the programmer responsible for the Commodore 64 and Commodore 128 adaptations of the Wizardry series (and now Twin Cities 128 contributor) informs me that there are really two different versions of Wizardry on each Wizardry disk. The scenario goes somthing like this: If you autoboot Wizardry on your C-128, the loader activates a version of Wizardry which actually operates in and is optimized for the C-128 mode. The C-128 implementation uses all of bank 1 RAM as well as an any REU (if you have one) as a virtual RAM cache which holds most of the game's data and code. The result is much smoother game play and faster disk copying as Wizardry does not have to access the disk drive nearly as often. Another benefit of C-128 operation is its ability to use the 128's enhanced keyboard. Additionally, if you own a 1571 disk drive, Wizardry uses the 1571's burst mode to read and write data much more quickly. And as a good programmer should, Anton is constantly improving his techniques. In fact, he recently enhanced the virtual RAM cache so that it also takes advantage of the 16 or 64K of video RAM that was not being used for this 40 column game. (Anton says he decided to do this after learning how to tell whether a user has 16 or 64K of VDC RAM from Fred Bowen's article in issue #18 of Twin Cities 128...*See the techie stuff does pay off!*) The VDC RAM enhanced cache will make its first appearance with the third scenario of Wizardry when it is released. The 64 and 128 versions of Wizardry were the first to implement this RAM caching technique, which became so popular at Sir-Tech that within days similar caching schemes were ported to the Apple and IBM versions.

Speaking of Hiding... Many of you wrote to tell me you caught my commentary in a recent issue of RUN magazine concerning the lack of C-128 native mode support in Q-link's proprietary software. Most of you also indicated that you agreed with me but felt that the situation would likely not improve. However it appears my shouts are not falling on deaf ears. During the Friday evening session at the World of Commodore Show in Philadelphia an improptu discussion on Q-link's software developed in front of the TC-128 display. Present at this discussion were two Q-link executives, a Q-link programmer, Fred Bowen, and myself. As you might expect Fred and I were bombarded with the same old lame excuses, that is, until Fred suggested a radically different approach. It was right after one of the executives had suggested that the expense of creating and maintaining a second version of the software made 128 support too costly that Fred said, "Well, one way to do this would be to activate C-128 features from C-64 mode. In this way you could provide C-128 luxury and still have only one version of the code". At first, both the executives and the programmer looked at us with doubt, but as Fred and I began to explain how you could utilize the numeric keypad, the extra grey keys and even the 80 column screen and fast mode while still in 64 mode, we could see genuine interest and enthusiasm building. We also discussed emulating a forty column display on the 80 column screen for those Q-link applications that required the 40 column screen, as well as enhanced features such as DOS support while online and a better email/text editor. The Q-link execs indicated they are interested in the concept, since the "C-128 driver" for this new version of Q-link would be small and work very well without requiring a separate version. I will keep you posted as things develop.

Cool stuff from the labs... One software developer who has employed this "128 driver" approach when creating a 64 mode application is Dr. Evil Labs of St. Paul, Indiana. Dr. Evil's shareware KERMIT telecommunications package employs this technique giving 128 owners a much nicer package while keeping development time and effort at a minimum. Fred and other people at Commodore are so impressed with the result that a new version of the 1670 disk will include the latest release of KERMIT (version 2.2). The KERMIT package is specifically designed to facilitate communications with mainframe and minicomputers which utilize the KERMIT file transfer protocol and supports a variety of common terminal emulations including very well done VT-100 emulation. If you just want to check out this "128 driver" concept I strongly suggest you download the current or earlier versions of Kermit, or better yet, send a $5 or $10 donation to Dr. Evil Labs, P.O. Box 190, St. Paul IN 47272. In return for your donation you will get a bound manual. I should also tell you that it is one of the best looking software manuals I have ever seen.

Another "cool thing" from Dr. Evil Labs is their stereo cartridge called the "SID Symphony". This cartridge adds a second SID chip to your 128, allowing you to create and listen to six voice stereo music. Having had the pleasure to enjoy one for several weeks I have to tell you this is a real treat! The cartridge sells for $34.94, add another dollar and they will include a double-sided disk with music playing software and over two-dozen of the best stereo SIDplayer songs. For music creation, the cartridge is compatible with the enhanced SIDplayer music system from Compute! Books and there are hundreds of music files in the public domain on Q-link and GEnie.
At present, the music player software is written in 64 mode, however, Mark Dickenson, the player's author and the stereo SID concept originator is working to create a C-128 version.



On the visual end of the Spectrum... Spectrum 128 is a full featured paint program which uses 80 column display for 640x200 pixel resolution. Spectrum 128 will display all 16 standard colors and 128 colors through color dithering. Spectrum 128 is menu operated and requires a 1351 or compatible mouse. Among its other many features are air brush, erase, mirror, multi-color, block fill or erase, pixel editor, color editor, built-in slide show, uses 128 fast mode, support for 1750 REU, support for 1541, 1571, and 1581 drive, support for most printers, create hexagons, diamonds, octogons, lines, boxes and circles, uses 8x2 color cells, create 3D solids, adjustable font sizes, text mode includes reverse, underline, sideways, mirror, flip, complement, and pattern modes. Spectrum 128 is compatible with Basic 8 files, Print Shop graphics, Sketchpad 128 graphics and News Maker 128. Spectrum 128 must be run on a 128D or C-128 with expanded video RAM. Spectrum 128, $39.95, Free Spirit Software, PO Box 128, 58 Noble St, Kutztown, PA 19530, 215-683-5609, 800-552-6777

Pocket Writer 3, Talk to me? Although the gang at Digital Solutions for some reason is unable to return my calls (as of this writing three). I will pass along what I know about Pocket Writer 3. The upcoming release supposedly will be available in Febrauary 1989 and include all of the features of Pocket Writer 2 with the addition of: automatic support for multiple columns, macro capability, undo/cancel last command, markers for marking up to 10 locations in one document, book paging, odd/even and left/right, line and box drawing modes, word, sentence and paragraph count, find/replace in either direction, cursor movement by sentence and paragraph. Word through the grapevine is that upgrade price for current Pocket Writer 2 owners is $30, but I have not been able to verify this. Maybe you will have better luck: Digital Solutions, P.O. Box 345, Station A, Willowdale, Ontario, Canada, M2N 559, 416-731-8878.

continued on page 4

PRICE & PROGRESS REPORT

continued from page 3

Overheard: Len Lindsay and his COMAL Users Group, are considering the development of a C-128 version of their COMAL powerdriver. The powerdriver is an enhanced version of the disk based COMAL .14 system which includes a run time library so programs can be used outside of the COMAL environment. Len says user interest will determine whether they will go ahead with the project. I implore you to write him at: The COMAL Users Group, 6041 Monona Drive, Madison WI 53716. **Lou Wallace** tells me that legal hassles with the former distributor of BASIC 8 are holding up some of the C-128 products he and **Dave Darus** have been working on. However they are optimistic they will be able to deliver the 100% assembly language paint package I described in issue #22 sometime this spring. They are also working on a BASIC 8 pre-compiler which would allow BASIC 8 applications to be compiled using the Abacus BASIC complier! **Les Lawrance** of Software Support International sent us a copy of SSI's new 1581 Toolkit and it is awesome! Besides containing just about every utility for the 1581 you could probably conceive, the package includes a bonus 150 page authoratative text on the 1581 by disk drive expert **David Martin**. The package is excellent and runs in 64 mode, however Les indicated that if sales go well they may do a C-128 version. It sells for $39.95 plus $3.50 shipping and is available from Software Support International, 2700 N.E. Andresen Road, Vancouver WA 98661, 1-800-356-1179. Various beta-testers of **GEOS 128 v2.0** tell me that the project is very nearly complete. They tell me that the package is essentially an 80 column version of GEOS 2.0 for the 64. While this means it is a substantial upgrade and worth getting if you are into GEOS, I am disappointed to hear that Berkeley Softworks, at least as of this writing has failed to include 64K VDC RAM support with color or interlace video support. Maybe they just have not caught up on their back issues of Twin Cities 128!

And last but not least! Before you type in any of the programs in this issue, as a service to you our readers, I will make an effort to get these programs online on both GEnie and Quantum Link. However if you prefer, we can put them on disk for you. Just send $5.00 to: Twin Cities 128, P.O. Box 4625, Saint Paul MN 55104, Attn: Issue #23 programs. In addition, we are extending our back issue special! While they last, you can obtain three back issues of Twin Cities 128 for $5.00. We have back issues 4 (yes we found a box of old issue fours! A real collectors item!), 19, 20, and 22 in stock. Single issues are $2.50. Subscriptions are still: Six issues for $12.50, Twelve issues for $25. Till next time...

NOTHING LOADS YOUR PROGRAMS FASTER THAN THE QUICK BROWN BOX !!

A NEW CONCEPT IN COMMODORE CARTRIDGES

STORE UP TO 30 OF YOUR FAVORITE PROGRAMS IN A SINGLE BATTERY BACKED CARTRIDGE FOR EASY AND INSTANT ACCESS! CHANGE THE CONTENTS OF THE BOX AS OFTEN AS YOU WISH. THE QUICK BROWN BOX ACCEPTS MOST UNPROTECTED AND "FROZEN" PROGRAMS INCLUDING THE ONLY WORD PROCESSOR THAT SAVES YOUR TEXT AS YOU TYPE: THE WRITE STUFF! THE BOX EVEN CO-EXISTS WITH GEOS, AND THE COMMODORE RAM EXPANSION UNITS. LOADER UTILITIES INCLUDED FOR BOTH THE COMMODORE 64 AND THE COMMODORE 128.

**Brown Boxes Inc.
26 Concord Drive
Bedford MA 01730**

617-275-0090 617-862-3675

32K QUICK BROWN BOX...$99   64K QUICK BROWN BOX...$129

UTILITIES DISK..$6   Q-DISK CP/M RAMDISK...$10

SPECIAL PACKAGES

64K QUICK BROWN BOX & THE WRITE STUFF 64....$139
64K QUICK BROWN BOX & THE WRITE STUFF 128...$144

Add $3.00 shipping and handling
Mass. orders add 5% state sales tax
C.O.D add $3.00, overseas add $5.00

JIFFYDOS 128 REVIEW

by Fritz Neumann

Most serious Commodore users have at one time or another used something to speed up their disk drives. This "something" can take various forms, from a software wedge loaded from disk all the way to massive hardware modifications involving the addition of several chips in both the computer and drive with ribbon cables stretching between the two. Most often, however, Commodore users have resorted to cartridge-based fastloaders to give their systems a little extra zing at the cost of some compatibility. JiffyDOS is a disk speedup system with some important differences that make it a more compatible and permanent solution to the drive speed problem.

The cartridges do work, but they have some major drawbacks. First of all, there are only a few that support the C-128 in its native mode, and those usually require the user to toggle a switch on the cartridge to match the C-128's current operating mode. In addition to that annoyance, many copy-protected disks are incompatible with the cartridge fastloaders, sometimes requiring the user to turn off the computer and remove the cartridge before loading the program. Cartridges also use the same port as the Ram Expansion Units, requiring special adaptor cards in order to use both simultaneously. Also, most cartridges only speed up the initial program load; only the latest, most expensive ones speed up disk access from within a program when loading and saving all file types. Some cartridges modify the normal disk format so heavily that their disks are nearly unreadable on stock systems. Given these problems, some opt for complicated hardware solutions, which are usually very expensive, require special cables to the user port (eliminating the possibility of using a modem), and still have software compatibility problems. No hardware fastloader I have seen will work with multiple dissimilar drives (this means you cannot have both a 1571 and a 1581 connected at once, for example).

So is there a solution that does not interfere with your system? Actually, it is too much to ask for a cartridge-based fastloader not to mess with normal system operation, since the whole concept of a cartridge-based fastloader is based on interrupting normal system activity in order to activate its own fast code. The problem is that the cartridges are simply trying too hard to "patch" the real problem: an inefficient operating system that is hard-wired in your Commodore's Kernal ROM code. The hardware patches may avoid the cartridge problem, but by providing a non-standard system with limitations.

Realizing that cartridges are at best only a temporary solution to the problem of slow disk access, the makers of JiffyDOS decided to re-write the serial bus transfer code in the Kernal Rom itself. The result this effort is JiffyDOS, a replacement Rom chip set for both your computer and disk drive which neither "patches" the bad code like the cartridges nor adds new hardware like the more elaborate hacks, but simply takes full advantage of your existing C-128 hardware and serial bus. This degree of optimization is what makes JiffyDOS a much more elegant and permanent solution than any other fastloader.

Physically, JiffyDOS arrives as a replacement Rom chip set for both your computer and your disk drive. Extra drive chips are available for your other drives, but it is not necessary to have a JiffyDOS Rom installed in every drive: the JiffyDOS computer will automatically slow down to normal speed when accessing a non-JiffyDOS drive. Currently, JiffyDOS is available for the C-128, C-128D, C-64, 64C, and SX-64 computers and the 1541, 1541C, 1541-II, FSD 1&2, MSD 1&2, Excel 2001, Enhancer 2000, Excelerator Plus, 1571, and 1581 drives. Because of the different chips used in each system, it is necessary to specify which computer and drive you are using, although the 128D and SX-64 sets include chips for their built-in drives. Installation in the 128D is really very simple, and the C-128 is only slightly more complicated because its metal shield takes more time to remove. In both machines, and in the 1571 and 1581, the old Rom chips must be removed from their sockets and the JiffyDOS chips put in their place according to the excellent and well-illustrated installation manuals included. The JiffyDOS chips have a small switch wired to them which requires a small 1/8" hole to be drilled in the side or back of your C-128 or 128D, allowing the user to enable or disable JiffyDOS by merely toggling the switch. The 1571 and 1581 chips do not need the switch (they will automatically change modes according to the computer), but the 1541 does. The entire installation procedure should not take more than 15-20 minutes.

Once the chips are in and JiffyDOS is switched on, you will notice an additional line in your power-up message informing you that JiffyDOS is active. All disk accesses to JiffyDOS-equipped drives will then be accelerated, whether you are simply loading a program to run it or accessing disk files from within a program! Here are some observations from a 6-month JiffyDOS veteran:

* YOU WILL NEVER HAVE TO TURN OFF JIFFYDOS: IT WORKS WITH NEARLY EVERY PROGRAM!
* ANY FORMER SPEED DIFFERENCE BETWEEN C-64 AND C-128 MODES IS ELIMINATED: BOTH RUN EQUALLY FAST UNDER JIFFYDOS.
* RE-SAVING FILES TO FRESHLY-FORMATTED DISKS WILL GIVE YOU TOP PERFORMANCE, BUT YOUR EXISTING PROGRAMS WILL ALSO BE LOADED FASTER.
* DISK ACCESS IS SO FAST THAT THE NEED FOR A COMMERCIAL COPIER IS LARGELY ELIMINATED: PUBLIC DOMAIN AND SHAREWARE UTILITIES LIKE UNI-COPY AND DISK WHIZ HANDLE FILE-COPYING TASKS ALMOST AS QUICKLY.
* YOU CANNOT ACCESS THE DATASSETTE WHILE JIFFYDOS IS ACTIVE.
* YOU CAN ACCESS BOTH SIDES OF A DOUBLE-SIDED DISKETTE WITH YOUR 1571, EVEN WHEN IN C-64 MODE, WITHOUT HAVING TO SEND ANY SPECIAL COMMANDS TO THE DRIVE. SINGLE-SIDED FORMATS ARE ALSO FASTER, TAKING ONLY 20 SECONDS.
* ALL YOUR PORTS ARE FREE: YOU CAN USE YOUR MODEM, RAM EXPANDER, CARTRIDGES, WHATEVER YOU WISH. NO EXTRA CABLES ARE NECESSARY.
* DISKS WRITTEN UNDER JIFFYDOS ARE PERFECTLY COMPATIBLE WITH NON-JIFFYDOS SYSTEMS; YOU MAY EVEN MIX JIFFYDOS AND NON-JIFFYDOS DRIVES OF ANY DEVICE NUMBER IN ONE SYSTEM WITH NO PROBLEMS.
* JIFFYDOS DOES NOT INTERFERE WITH PRINTER INTERFACES.
* WITH A FLIP OF JIFFYDOS' SWITCH, YOUR SYSTEM IS FULLY RESTORED TO ITS STOCK CONFIGURATION. THIS CAN EVEN BE DONE WITH THE POWER ON.
* JIFFYDOS INCORPORATES THE LATEST ROM UPGRADES FROM COMMODORE (ALTHOUGH IT DOES NOT UPGRADE THE 128'S BASIC 7.0 CHIPS). THE 1571 JIFFYDOS ROM ALSO INCORPORATES THE LATEST REVISIONS, BUT UNLIKE COMMODORE'S UPGRADE IT WILL WORK WITH THE OLDER COPIERS LIKE FAST HACK'EM AND COPY II.
* ON TOP OF ALL THIS, JIFFYDOS IS MUCH CHEAPER THAN THE HARDWARE HACKS, IT COMES WITH A 30-DAY "SATISFACTION GUARANTEED OR YOUR MONEY BACK" GUARANTEE, AND IS ALSO COVERED BY A 90-DAY WARRANTY FOR DEFECTIVE ROMS.

Just how fast is JiffyDOS? JiffyDOS will speed up program loads from your existing disks by a few seconds with a 1571 or 1581 in C-128 mode, but the big difference occurs after a file has been re-saved under JiffyDOS, which also happens much faster. JiffyDOS will speed up the save of a 202-block program file to a 1571 from a stock 95 seconds to roughly half that: 49 seconds. A 1581 performing the same task would take 52 seconds normally, but only 27 with JiffyDOS. This re-saved 202-block program could then be loaded in only 9 seconds from either drive! A 125-block sequential file that used to take 60 seconds to write and 31 seconds to read would take only 41 to write and 12 to read with JiffyDOS. Reading 64 154-byte records in a relative file takes nearly 40 seconds even with a 1581 and a C-128; JiffyDOS accomplishes the task in just 10 seconds, even from within a database program such as Superbase! Switch to 64 mode and the effect is even more dramatic: that 202-block program file used to take over 2 minutes to load with a 1571, and

Continued on page 6

JIFFYDOS 128 REVIEW

Continued from page 5

now it is loaded in just 9 seconds! Essentially, all drive-related activity is accelerated to some degree, depending on the program. The only exception occurs with programs that avoid the standard Rom routines and use their own code to achieve greater speed. Programs like GEOS and Fast Hack'em are not significantly faster because of this reason, although they will work fine and even load faster. I also noticed little difference in CP/M mode, except on some file saves.

In addition to its drive-speed enhancement, JiffyDOS also provides a DOS wedge, allowing direct drive access from Basic with commands most wedge users will be familiar with like "@N:" for formatting a disk, "/" for loading a Basic program, "%" for loading ML, and so on. The JiffyDOS wedge also has several enhancements, however, like "@U" to "un-new" a Basic program in memory, and "@T:" to list a text file directly from disk without disturbing memory. The wedge commands may either be used directly or in Basic programs, or they may even be disabled via the "@Q" command. The wedge is especially useful in C64 mode, where disk access is usually quite cumbersome.

Does JiffyDOS have any drawbacks? Well, I have had them installed for over six months on both my C-128 and my 128D and used them with both 1571 and 1581 drives (some with JiffyDOS installed, some without) and I can think of only a few possible shortcomings. First, it does require you to open up your computer to change the ROMs; this will violate your warranty if it is still in effect, but is an absolute necessity for such a permanent solution to drive speed problems. Second, JiffyDOS reprograms the default function keys; you may restore the stock definitions by simply typing "@F <RETURN>", but it still annoys me when I power up my 128, press F3, and do not get a directory on my screen. Third, the <CONTROL>-P text screendump does not work properly in 128 mode, although it works fine in 64 mode. Fourth (and last), the special "@N2:" format, which is supposed to format a 1571 disk as double-sided when in 64 mode, does not seem to work; both sides are indeed formatted, but the double-sided flag is not set on the directory track, preventing the 1571 from recognizing the disk as double-sided. These shortcomings, however, are rather trivial compared to the luxury of fast, compatible, smoothly-running disk drives 100% of the time.

As a final note, I should also mention another asset that JiffyDOS holds over most other software or hardware add-ons of any sort: customer service. The people at Creative Micro Designs, the creators, manufacturers, and marketers of JiffyDOS, are extremely helpful and knowledgeable about their product and they support purchases from individuals and users' groups. I have conducted two bulk orders through our users' group and have been very impressed with the courteousness and efficiency of the staff at Creative Micro Designs. I only wish more computer equipment retailers were like them!

JiffyDOS/128, $59.95, Each JiffyDOS order includes ROM chips for one computer and one drive. Additional drive ROMs: $24.95, Users'group quantity discounts are also available.

Creative Micro Designs, PO Box 789, Wilbraham, MA 01095, (413)-525-0023

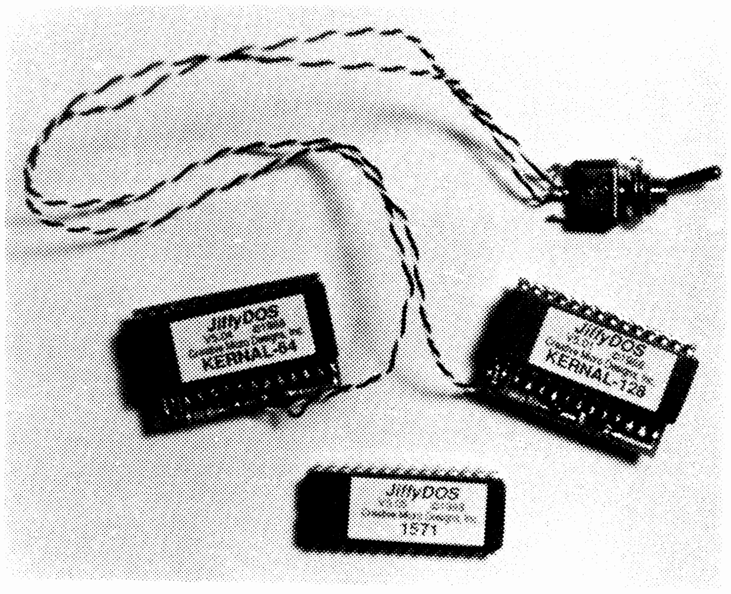
MAS-128

Full-Featured 6502/6510/8502 Symbolic Assembler for the C-128 based upon the Innovative C-64 Midnight Assembly System.

- ☆ 80 Column RGB Full Screen Editor.
- ☆ Executes from RAM 0 or RAM 1.
- ☆ Supports 1700/1750 REU with Built-in Proprietary RAMDOS, complete with commands for transferring entire disks to and from REU.
- ☆ Makes absolutely NO calls to ANY of the the C-128 ROMs (contains its own keyboard, video, and serial I/O drivers). Indispensable to the programmer who wishes to design C-128 ROM replacements.
- ☆ Multiple Statements per line of Source Code.
- ☆ Not Copy Protected! Archival Backup Encouraged.

Direct Inquiries to:

Mountain Wizardry Software
P.O. Box 66134
Portland, OR 97266
Tech. Support (503) 265-2755



NEWS MAKER 128 REVIEW

by Frank Hudson

With the 40/80 key on the top of its keyboard in the down position, the C-128 has many good word processing tools and a still unfolding talent in graphics display as well. Put that together and you would have the basis for a desktop publishing (DTP) program that could turn out newsletters, pamphlets, small books, and so forth.

Having recently become aware of the "proudly produced entirely on the C-128 in native mode" production process of TC128, I will note only that a whole battery of C-128 programs are brought to play in a unstinting journalistic battle to bring you the magazine you hold in your hands. TC128 owns no special, secret C-128 desktop publishing solution, just some clever ideas on how to combine the talents of various word processing and graphics programs. But wouldn't it be nice if there was a program allowing the best of the C-128's text and graphics to be freely manipulated together on a hi-res 80 column screen?

Free Spirit Software now distributes a new 128 DTP program, News Maker 128 written by David "Whiz Kid" Krohne, which is a tentative step toward the kind of software that would allow more of us to lay out our own pages using all the power the 128 is capable of.

News Maker is a 80 column mode program, one of the first to require 64K of VDC RAM--display RAM which is needed to present the C-128's highest resolution screens. All 128D's have 64K VDC RAM installed as they come from the box, and since the initial experiments concocted by TC128 a couple of years ago, many of the older flat C-128's have had their VDC RAM upgraded by users to this level. A 1351 mouse is also necessary to run the program. DTP programs vary considerably in power and complexity. Light duty programs are suitable for short newsletters or flyers, while the higher cost, higher performance DTP packages can do full size books and offer more layout options. News Maker is designed as a light duty program. This in itself is neither good nor bad. Users would be better off with an inexpensive easy to learn DTP program if that is what fits their needs. At a price of $24.00, News Maker is easy on the wallet.

Unfortunately, News Maker 128 does not perform acceptably. The program's overall design concept is pretty good, and if its multitude of problems are someday fixed, News Maker could be a good beginner's program for smaller projects. News Maker's features, could stand as a model of what a simple DTP program should offer in that it: 1. Accepts text in PETASCII form from most C-128 word processors while embedded formatting commands are ignored, 2. Imports Basic 8 or Print Shop graphics, 3. Offers a selection of fonts scalable to 9 different sizes, 4. Uses the mouse to place text and graphics on the page, 5. The 80 column screen display means about 1/2 the height and the full width of the page is visible at all times, 6. Text can be "poured" into boxes placed and stretched with the mouse to whatever size is desired, 7. A graphics toolbox is included to draw lines, boxes, circles, etc. Even if you are importing all your graphics or producing a text only page, such facilities are useful for setting off a section of text or for drawing rules dividing columns or headlines, 8. The 1750 REU is supported, though not as a fully functional Ram disk.

News Maker's chief conceptional flaw is its total lack of an undo function for any of its features. DTP is better than traditional cut and paste techniques largely to the degree it allows layout give and take between "All the News" and what "Fits to Print". Once anything is placed on the a page it's there for good unless one erases and starts over.

A file on the News Maker disk claims that text can be adjusted to fit around graphics, but this is not explained further in the 9 page program documentation, and I was unable to discover any way to do this, unless the author meant only the equivalent of "Don't put any text where there are graphics on your first and only try".

Placement of text and graphics is not much aided by the computer. There is no layout grid (except for the location markers on the margins and the column guidelines), nor is there any "snap to grid/guideline" or similar feature. News Maker doesn't allow fine positioning to the pixel level anyway, but such a feature would aid the user in getting text blocks or pictures in the right place during the only try they are allowed. I would also wish for better looking fonts and a programmer's decision to use either the ESC key or the right mouse button as the escape key, but not the combination of both News Maker currently uses. News Maker's manner of handling files longer than the current page is probably too limiting for even a simplified DTP program.

But where News Maker really falls down is in execution. The Orkin man needs to spend some time spraying down the corners of this program's code! Innocuous acts like deleting one space beyond the screen's left margin in write text mode will crash you back to BASIC. The mouse pointer can at times go on vacation off the visible screen, leaving the poor user to try to figure out which way to drag it back into action. Using the erase function to remove unwanted sections of text or graphics can also remove the program's location landmarks and column guides. And something as routine as a simple file requester is hampered by not allowing the delete and insert keys to correct typing errors, and efforts to use these keys will result in invisible characters being added to the file's name.

With the lack of undo, boxes that can be moved or changed after placement, snap to grid, or for that matter reliable and intuitive program function, News Maker is inferior to traditional cut an paste techniques.

How's the print quality you ask? Beats me! Despite repeated efforts to utilize my home printer and those at the TC128 headquarters, we were unable to get a printout from News Maker 128. The documentation has little to say on these matters, referring one instead to the helpful folks on Q-Link or your local users group!

News Maker recalls the early days of personal computer software when customers were expected to work around bugs or fix them themselves. Anyone wishing to relive that experience will be happy to know that News Maker, just as programs often were then, is in listable, modifiable Basic. Owners of Walrusoft's Basic 8 package could fix some of these problems by writing their own code, but since News Maker 128 is a copyrighted commercial program, distribution of such modified versions without the express written permission of the Whiz Kid, Free Spirit Software, and the Commissioner of Major League Baseball would be prohibited.

For now, News Maker 128 proves that you can write "quick and dirty", insufficiently error-trapped code in even a good, powerful language such as Basic 8.

C-128 owners looking for a simple DTP program of similar capabilities, but greater reliability than News Maker, could consider Softsync's Personal Newsletter running under GEOS 128 in (alas) 40 column mode.

For the user seeking more flexibility and willing to put in a bit longer learning time to gain it, geoPublish, again running in the GEOS 128's 40 column mode is a fine effort. Xetec's Fontmaster 128 or a combination of word processing and graphics programs will allow you to keep your 40/80 key down, but without the effortless integration of elements that a dream C-128 DTP program could offer.

News Maker 128, Free Spirit Software Inc., Box 128, 58 Noble Street, Kutztown PA 19530, 1-800-638-5757, $24

Z3 PLUS REVIEW

by Miklos Garamszeghy

One of the most frequent complaints I hear about CP/M on the C-128 is its lack of 'user friendliness', especially towards Commodore junkies who have never bothered to acquaint themselves with other computer systems. Ask what would constitute a user friendly system, you are likely to get as many different responses as people you ask. This seems to indicate that the ideal operating system should be customizable so that it can appeal to diverse tastes. Z3plus is such a system.

Z3plus, or the Z system as it is otherwise known, has evolved considerably over the years since it made its initial debut as ZCPR, almost at the dawn of CP/M computing. Versions exist for almost every Z80 CP/M system around, the latest release running under CP/M 3.0 or CP/M Plus which just happens to be the CP/M used by the C-128 as well as a few other less important (to me anyway) computers.

What is Z3plus?
Z3plus is essentially an enhanced replacement command processor for the standard CP/M CCP.COM operating environment. It is a user interface that provides features such as named directories (which can be named across drives and user areas), extensive command line editing, keyboard macros and enhanced batch file processing. The system comes complete with a number of operating system shells of varying sophistication which allow you to perform routine house keeping functions such as running programs and copying files from a point-and-shoot type menu.

Z3plus is comprised of the main operating module (Z3plus.COM) and a number of transient command and utility programs. The commands are broken down into three segments: the FCP (flow command package - which is used to decide branching and conditional execution in batch file type processing, such as IF, ELSE, etc.); the RCP (resident command package - general commands ECHO, CLS, etc); and the CPR (command processor - system commands GET, GO, JUMP).

The Z system is customizable in a number of ways. The first level of customization involves which commands you decide to include with your system. The 'stock' Z3plus system includes a wide variety of options and commands in each of the three command types outlined above such as CLS (clear screen); ECHO (print message to screen); POKE (for changing system memory); IF, AND, OR, ELSE (for conditional batch file execution); GET (load a file); GO, JUMP (execute a previously loaded file); etc. Any or all of these commands can be included in your personal command library. Obviously, the more commands you make resident, the more memory will be required by system overheads.

By using GET and GO separately, you can load and run programs in areas other than the default start of TPA, providing of course, that the files were assembled with the non-standard start address in mind. This allows you to have more than one program in memory at once by having each located in a different area of RAM. (In fact most of the Z system shells and utilities work in this fashion.) An interesting point is that GET is not restricted to loading program (COM) files and can even be used to 'load' text files. Of course, you will not be able to execute the text file, but you can bring it into memory if you wish.

The second level of customization involves the use of 'aliases' and script files. An 'alias' is defined in the manual as a "single word or command that stands for a longer or compound command". (The manual talks extensively about creating and using 'aliases'. I think it would be far less confusing to the average reader to adopt more standard computer terminology and refer to them as keyboard macros or batch files. I admit that in the purest sense an 'alias' may not be either a macro or a batch file, but as far as I am concerned, it is close enough for jazz.) The alias allows you to set up custom names for your favorite command sequences. Script files are more extensive and interactive than 'aliases' and can be combined into libraries containing some very sophisticated custom menu routines. You write them yourself and therefore include whatever you wish.

Of Named Directories
One of the many interesting features of Z3plus is its use of the CP/M user areas as named directories. This can help people to organize large disks into smaller areas associated with easy to remember labels. For example, with the EDITNDR you can define user area 15 on drive M as the "SYSTEM" directory. Now when you log onto user 15 of drive M:, the prompt will display the name of the directory "SYSTEM" in addition to just the usual CP/M 'M15' prompt.

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

Z3plus uses also provides for password protection of files and directories.

The Tools and Utilities
Most of the utilities provided on the distribution version of Z3plus are public domain. (This does not mean, however, that you get the same old tired programs that you probably already have several copies of in your library. They have been put into the public domain by their various authors to the benefit of all Z system users.) The major ones, such as the operating system shells EASE and ZFILER, have been specifically written to run in the Z3plus environment, so would not do too well without it. (They are public domain in the sense that you are free to copy and use them as you see fit. The Z3plus.COM main system modules are NOT public domain, however.)

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

ZFILER is the second operating shell provided with Z3plus. It is basically a point-and-shoot type menu driven file management program which does things like batch copying, run other programs, etc. Like the other Z3plus utilities, it is clean and very easy to use.

(One interesting feature about the Z system is that it allows you to use multiple levels of shells. If you first activated the EASE shell, then went into ZFILER, you would go back to EASE when you left ZFILER. You then exit EASE to get back to the Z3plus system.)

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

SALIAS is a mini text editor used for editing and creating alias script files which uses WordStar type control code commands for editing and cursor movement. In addition, SALIAS can be used for other general editing of short text notes as well.

ARUNZ is an alias library manager of sorts. It allows you to combine many single alias script files into one large one, thus saving on disk overhead space (one large file can take up significantly less disk space than many small ones due to the size of the CP/M disk allocation unit size of 1k

continued on page 10

TAROT 128 REVIEW

by Frank Hudson

No one knows the origin of the deck of 78 picture cards known collectively as the Tarot. Though widely supposed to be the prototype of our common playing cards, and from the symbolism used, often estimated to be of medieval European derivation, the very name "Tarot" is subject to various theories. Some say the name's ending refers to the rotating wheel (Latin "rota") pictured on one of its cards, a common medieval motif known in those Pre-Vanna times as the Wheel of Fortune, a symbol of mankind's ever changing state. Another hypothesis credits two words of unspecified derivation "tar" meaning "way" and "rog" meaning royal, together: "the royal way". In any case, the deck was little noticed until the 18th century when both French intellectuals and charlatans (even then in close association?) began to write of it.

As these things go, they squabbled over the order of the cards, their source, and some details of card names and depictions, while generally agreeing on what constituted the Tarot deck and that it was not of recent invention.

The occult theories of the Tarot's origin are often quite colorful, usually holding to Egyptian or Kabalistic Hebrew origin. Some go so far as to pin down the cards authorship to a particular Egyptian demi-god. An unanswered question is why these guys would take a break from inspiring or building pyramids to design a Tarot deck rife with medieval European Christian symbolism.

Of all the popular fortune telling methods of the pre-scientific era, the Tarot and the I Chins remain the most respected by intelligent moderns. Astrology may have the mass blue collar support mixed in the with the belief of a politician here and there, but the hermetic poetry of the I Ching commentaries and the vivid imagery of the Tarot retain a power, even over skeptical audiences.

I say all this first, not because TC128 has become New Age Journal or Etymologists Review, but because the Tarot deck itself is so much more fascinating than the computer program under discussion, Tarot 128 from Moonshine Software.

Put simply, Tarot 128 is a fortune telling program, no different than a fortune cookie and about as filling. In its favor, its price is not much different from the good Chinese meal that would precede the cookie, and it is a too rare example of non-productivity software for the 128's native mode. Tarot 128 is a very simple program to use. It self-boots in 80 column mode displaying a four choice menu. Pressing the F1 key begins a reading by requesting you to chose a certain card based on your age and astrological sign, a card that then will signify the questioner. You are also asked to select from a short menu what the area of your inquiry is to be. These limited choices appear to be totally ignored by the flow of the program, excepting only that the signifying card is "removed" from the deck.

The next screen is highly disappointing. The cards are laid out on the computer screen in a mercilessly simplistic representation of what is in reality a highly visual experience. Each Tarot card is shown in a manner similar though actually inferior to older public domain poker games! Instead of the powerful and troubling scene of, for example, two people falling or flying from a toppling citadel, one sees a colored rectangle with the text "The Tower" written on it. Even the real Tarot's pip cards are illustrated.

For example, the featureless rectangle labeled the eight of swords on Tarot 128's screen (corresponding to the eight of spades in our familiar deck) represents a card which should depict a woman bound and surrounded by eight swords pointed at her. Furthermore, each card is "dealt" with an IBM PC-like "plink" and an annoying screen flash.

A Return keypress gets one out of this ugly screen and takes the user to a simple listing of which cards have been selected by the computer dealer in which order and gives aterse note as to what area of the question is to addressed by each card. In another difference from the common playing card, all the Tarot deck have a "right side up", and the meaning of a card is essentially reversed if it is dealt out upside down. The too schematic card display cannot show this directly, and the card listing here uses a paradoxical up arrow to show a reversed card.

The user must then individually select each card by number to receive the meaning of the reading. The drive is then accessed for a short SEQ text file which is displayed to the screen. Alternatively, a printer dump of the readings of all the cards can be ordered, and this option is to be preferred as it is the only way the whole of the interpretation can be viewed at once. There is but one description/meaning file on the disk for each card however, so no matter if one's question was about the stock market, romance, or a trip to Cleveland, the answer will be the same for the series of cards that were dealt.

Here is the text displayed for the Three of Pentacles (Diamonds): "A sculptor is finishing a carving of three pentacles on a church archway. A monk and a nun stand nearby watching. In numerology, the number three represents completion. Divinatory Meaning-Gain in a commercial transaction. Material increase. This card represents mastercraftsmen and skilled artists. Reversed-Ignorance and lack of skills. Selfish preoccupation with material gain. Uninspired ideals and philosophies." It is in this way that the program attempts to compensate for its appalling graphic display. Taking an Infocom approach to the Tarot's images which ordinarily would be coursing powerfully through the unconscious is a fundamental mistake. A Tarot card is not a "meaning", it is an experience. Yes, commentaries can increase the understanding of this experience, but it was no doubt designed to be and should remain a fundamentally aliterate business.

The interpretations are fine as far as they go. The program could serve as a brief introduction to the Tarot deck at a slightly greater price but with a faster retrieval speed than a book on the cards.

If Moonshine Software is considering an upgraded version of Tarot 128 I'd encourage use of the 80 column graphic abilities of the 128's native mode. BASIC 8 and the interlace 640X400 monochrome mode would allow the cards to be depicted with some fidelity, or a color slideshow display could be DMA'ed over from a REU. A large GEOS-like virtual screen could also do the layout of the cards justice.

Although the program package encourages thinking of the Tarot as more than a fortune telling method, that is plainly what the program is cut out to do. Questions, methods, and results are simplified, though they make the program a breeze to run. Documentation is brief, but the program requires none besides booting instructions anyway. The history and tradition of the Tarot deck are covered more fully in this review than in the software package itself.

In the spirit of the fortune telling nature of the program and TC128's unflagging investigative commitment, I'm going to close the review with an actual test of Tarot 128, one we can all evaluate. I decided to place before our computer oracle the question of how the Minnesota Twins will do in 1989. Here's how the cards fell:

Card one dealing with the general atmosphere surrounding the question was The Hierophant, reversed indicating unconventionality and change...refers to Peter Ueberroth stepping down as Commissioner?...The Hierophant we're told wears a triple crown...since his card is reversed, does this mean more bad luck for Kirby Puckett?

continued on page 10

TAROT & Z3 CONTINUED

continued from page 9

Card two covers card one with comment on the forces of good and evil. It's the seven of wands, showing a man defending against six enemies...and obvious reference to the other six teams in the AL West...Tarot 128's reading is victory and success against opposition.

Card three, the ace of swords, refers to a central past experience...Tarot 128 says it is the sign of Championship...again, any fool can see the cards are recalling the Twins 1987 World Championship.

Card four recalls the influences that are passing away at present. It's The World, reversed. Right side up it's another card of triumph, but reversed it signifies failure from fear of change or vision...but since this card's location covers what's ending, it's saying there will more trades and lineup changes before long.

Card five is The Wheel of Fortune in the slot of something that may happen in the future...it's another lucky sign, boding well for successful change...it's card 10 of the Major Arcana, and 10 is Tom Kelly's uniform number, Kelly being signified by the Sphinx atop the wheel...therefore, Kelly's good luck may well increase.

Card six tells something that will happen in the near future. The four of swords depicts a sleeping knight and indicates a healing rest...the offseason perhaps?...anyway, it's a sign that major changes are not yet close at hand, though perhaps still to come (see card four).

Card seven is for negative fears, the five of wands, is reversed. It depicts a melee between five men, but reversed it means victorious cooperation...harmonious clubhouse?

Card eight, for opinions of friends, is the page of pentacles...since it's reversed it indicates wastefulness and news of money losses...expect the press and fans to decry big money contracts.

Card nine, hopes, is the ace of wands...it's a portent of a good beginning...a fast start this spring?

The last card shows the final outcome. After all these felicitous omens, its a bummer, the nine of swords. The picture on the card is of a dreamer awakened to find nine swords hanging in the air above the bed...Tarot 128 describes the meaning in three words "doubt, desolation, heavy burdens"...The nine swords seem to refer to a batting lineup...what is the meaning of this incongruous card here?...a pennant but a bad loss in the league championship or the World Series?...the heavy burden of trying to stay on top?...my guess is that means success followed by the breakup or rapid decline of the team.

That's Tarot 128's forecast for the '89 Twins. Preseason consensus will probably place them second or third in their division, but the cards say better than that. There is to be some sort of bittersweet ending. If this comes to pass you'll want to ignore my opinions and purchase a copy.

Tarot 128, Briwall, P.O. Box 129, 58 Noble Street, Kutztown PA 19530, 1-800-638-5757

Z3 PLUS REVIEW

continued from page 8

or 2k bytes on the C-128). When you use ARUNZ, you specify the name of the alias 'module' you wish to run, and ARUNZ will extract it from the alias library file (ALIAS.CMD), then execute it.

The Documentation
If I could say but one thing to the first time Z3PLUS user it would be: read the manual, front to back in that order and do not skip anything. The manual, like the Z3PLUS system, was written primarily by a physicist at MIT. (This person is so logical he would make Mr. Spock green(er) with envy, if he were capable of such emotion.) The manual was written to be read in consecutive order. (As a physicist, he should be familiar with the concept of Brownian motion, which is how I think most people, myself included, tend to read software manuals, i.e. randomly taking bits here and there. I made the mistake of skipping a chapter in the middle and was confused for quite some time until I realized that the chapter I had missed contained some vital information that I needed.)

Once you convince yourself that reading the manual is required, initial setup of the Z3PLUS operating system is quite simple and straightforward. You define your terminal capabilities (Saints be praised, the terminal type selection menu even includes an entry for the C-128!!) and rename a couple of files (this is the less obvious part that killed me before I read the manual in detail). Type in the magic word Z3PLUS and away you go.

The documentation itself is clear enough, although somewhat lacking when it comes to details. For example, in the section dealing with perhaps the most important utility, ZFILER (the general file handling, copying etc. utility), the part describing the command options merely tells you to look at the menu listing on the screen. I think that at least a command summary could be presented. (To their credit, however, a more detailed technical reference manual can be had, at extra cost. A bibliography of suggested further reading is also supplied.)

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

Of note is that the Z system is also support by a network of BBS's (referred to as Z-Nodes) which supply up to date technical information and help as well as providing a convenient method to distribute new programs written for the Z system. A list of Z nodes is included on the Z3PLUS disk.

Final Impressions
CP/M is a disk intensive operating system. Z3PLUS is perhaps even more so because of its reliance on transient commands and script batch files. Because of this, a fast drive is imperative (don't try it with a 1541, you will probably die of old age) and a RAM disk is even better. (An interesting combination is a 64k Quick Brown Box battery backed RAM cartridge with the QDisk CP/M driver software (see the review in the previous issue of TC-128). With this you can load most of the Z3PLUS main files and utilities into a non-volatile RAM disk and have them available as soon as you start up CP/M each time without having to copy them into the 1750 RAM disk.)

When I first started up my copy of Z3PLUS, I thought 'another semi useful product'. However, as I used it more, and discovered more of its features, I found myself liking it more and more and consequently using it more and more. It sort of grows on you. Although $69.95 may seem like a fair bit to spend on an operating system enhancement, it is well worth it if you are seriously into C-128 CP/M. What you get is an easily expandable and customizable operating environment that can be as powerful as you want to make it.

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

MAS 128 REVIEW

by Anton Treuenfels

The C128 Midnight Assembly System (MAS128), an editor/assembler from Mountain Wizardry Software, is an ideosyncratic product indeed. Like its C-64 predecessor, MAS128 takes the unusual step of disregarding all of Commodore's built-in ROM software, including the Kernel, in favor of its own operating environment. The somewhat rough-edged result is similar to several other Commodore editor/assembler packages with just enough differences to be disconcerting.

MAS128 is not auto-booting, nor can it be safely loaded from Basic. The pre-release version reviewed here came with cookbook-type instructions for booting the program from the C-128's built-in monitor. The instructions are adequate for users already familiar with the monitor and its operations. MAS128 can be loaded into either RAM bank of the C-128 and will normally generate code into the opposite bank.

After initializing MAS128 allows users to set the current date or accept the default date for use in source file date-stamping. The alpha pre-release did not do particularly extensive error-checking of the user-supplied date, allowing almost any text to be entered and then apparently interpreting the result as best it could. Dates not appearing on any normal calendar proved easy to generate (program author Matthew Montchalin reports this behavior has since been changed. He also indicates that, unlike common convention, "alpha" refers to the best pre-release version, after Huxley's "Brave New World").

The editing environment of MAS128 is similar to Basic's. The 80-column only full-screen scrolling editor treats lines starting with a line number as source text and lines without as commands to be acted upon immediately. List, auto line numbering, renumber, delete line range, and text search and replace are familiar editor features. Source text entry is free format, and several assembly language statements may be entered on the same line. Source files are loaded and saved using "disk wedge" commands.

The editor exhibits several behaviors of the either/or love/hate variety. If the screen becomes blank text scrolling stops, so that the screen remains blank until a line number is typed in. Listing does not put any spaces between a line number and a label at the start of the line's text. The syntax for specifying subrange listings will not permit a leading or trailing default value (eg., "LIST 2000 65535" is required instead of Basic's "LIST 2000-"). MAS128 employs its own keyscan driver and the keyboard does not always respond the way a user might expect. The function of temporarily halting screen output is handled by the ESCAPE key rather than the NOSCROLL key, for example (this maintains compatibility with MAS64). The SHIFT state of the C-128's numeric keypad is programmable in the fashion of the function keys.

MAS128 employs its own internal line format, but has some ability to import source text in other formats, including tokenized Basic, MAE, PetASCII, standard ASCII, and screen codes. Some additional reworking of such files will still be necessary, however, to adapt the code to MAS128 conventions. There is also a "TXT" command to produce an ASCII text file for export from MAS128. This command requires a filename, to which MAS128 automatically appends an extension. A test of this capability showed that Pocket Writer 2 did not get along with the extension and failed to load or otherwise work with the resultant output file.

MAS128 immediate mode commands are often three letters long and appear very much like pseudo-ops, perhaps because of MAS128's tokenizing scheme. Many of the immediate commands are disk-oriented at both the file and sector levels. Commands that operate at the sector level, such as "GET" (load) and "PUT" (write), work the drive job queue directly. The pre-release notes specifically warn that there is no error-checking of the user-supplied parameters. Specifying a non-existent track for one of these commands runs the risk of physically jamming the drive head.

The MAS128 assembler works in two passes. Source code can be scanned directly from the text buffer or from a "linked" disk file. Object code is stored at its execution address. There is no provision for assembly directly to disk. There is no built-in method of examining the object file in memory, nor is access to the C128's built-in monitor supported. Object files are loaded and saved using immediate mode commands.

Expression evaluation is handled as 16-bit integer quantities. Symbols may be 100 characters long and are differentiated by graphics characters and alphanumeric characters. Numeric values may be specified in binary, decimal, or hexadecimal format. The four basic arithmetic operators are supported. No logical operators are supported, while " > " and " < " denote left and right shift.

In general MAS128 does not allow polymorphic operators (ie., operators with more than one meaning depending on context), and so must find a unique method of specifying any allowed operation. Since the "*" character refers only to multiplication, the current value of the program counter is represented by the " = " character. The "-" character indicates binary subtraction and cannot be used for unary negation, ruling out the use of expressions such as "LDY #-1" (MAS128 in fact objects to this usage as soon as it is typed in. This can be worked around by expanding the expression to "LDY #0-1"). Extraction of high and low eight bits of 16-bit operands, commonly indicated by the " > " and " < " characters, is handled by pseudo-ops "HI@" and "LO@".

The assembler does not support user-definable macros, conditional assembly, or relocatable (linkable) object files. Several pseudo-ops offer a limited 16-bit macro capability, eg., "UP@" increments its operand by one. Labels beginning with a "." character are limited in scope to the currently linked source file; otherwise there is no support for local labels.

The distribution disk is not copy protected and in addition to MAS128 contains several example programs and a help file filled with comments from the author. The example programs include a sprite demo, a sound demo, and a program that implements "extra" microprocessor instructions by trapping the system BRK vector (actually this last is capable of and intended for incorporation into the user's own programs).

The full MAS128 documentation was not available for review, but the supplied material (the MAS64 manual in addition to pre-release notes) is primarily reference material covering commands, file formats, memory maps, and so on, much of which is useful for the intermediate to advanced user. Tutorial material is limited largely to recommendations to "try this key and see what happens" without any indication of the expected effect. Mountain Wizardry Software plans to allow users to purchase laser-printed copies of the full MAS128 source code for customization or study purposes.

MAS128 supports the 1541, 1571, and 1581 disk drives, including burst loading on the latter two. A 1700 or 1750 REU is also supported as device 0 via a proprietary RAMDOS. Whole disks may be copied to or from the REU with single commands (again relying on the drive job queue directly).

Author Matthew Montchalin must be commended for the time, skill, and effort involved in his attempt to bring his own preferred programming environment into being. MAS128 is a product with several interesting and useful features and qualities (support for REU's, availability of source code).

continued on page 15

EASY DIRECTORY READS

by Ed Parry

Most programmers will eventually need a directory routine in their program. Users often need to know what files are on a disk as well as how much room is free on the disk.

The included routines and files should be enough to help you understand how to program disk directory routines in basic and/or assembly. Each file is heavily REM'd for your convenience.

Obtaining directories has been made easy by our friends at CBM. We simply open the directory file with a secondary address of zero and basically read and print the directory data as it comes from the drive. The trick here is to be sure to use a secondary address of zero, for example, open 2,8,0,"$0:(pattern)". Using a non zero secondary address requires far more effort to obtain similar results.

Some Directory reading pattern tips:
Directory patterns were not well known (or documented) a few years ago, but we users eventually discovered them. We in turn pass the information along to other users. Hopefully by now, most users are aware of useful directory patterns.

Just in case you were not aware of them, here's a few directory patterns:

```
*     ;show ALL files in the directory.
*=s   ;show only SEQ entries.
*=p   ;show only PRG entries.
*=u   ;show only USR entries.
*=r   ;show only REL entries.
a*    ;show all entries starting with the letter 'a'.
a*,e* ;show all entries starting with 'a' and 'e'.
???   ;show all 3 letter entries.
$     ;show ONLY header and free blocks.
```

These patterns are not limited to the included directory files, but also work with your Catalog and Directory Basic 7.0 commands as well as all other programs using standard directory reading techniques. For instance, CATALOG"* = p" will display only the PRG (program) files on a disk.

Once you study these files a bit you should be able to see how you could use these techniques to obtain variable data from each entry. This is great for directory sorting routines, disk catalogers, file copiers, multi file RS232 xfer routines, custom disk shell's etc. In ML, I generally set aside a table for the entries. I figure two bytes for the filesize (in CBM blocks), 16 for the file name (pad with chr$(160) to the 16th place and use a file length of 16 and you will not have to track actual filename lengths!), and 3 characters for the file type. (IE: SEQ, PRG, USR or REL.)

Program Examples
dir.bas: This is the 1st basic directory program I came up with several years ago on my C-64. I was looking for a better way to obtain directories. At that time the 1541 manual included a SEQuential directory read listing/method that I found difficult to work with. It used a Secondary Address (SA) of 2 (instead of 0). To study how the directory bytes were layed out, I tried two types of Open techniques. I would Open the directory file, get one byte, poke the byte into a buffer area, then read the next byte (etc) until I hit the end of the dir file. This showed me exactly how the directory files were layed out when I examined my buffer area with a ML monitor. I tried both a SA of zero and a SA of 2. The secondary address of zero open technique appeared to be an almost ready and useable directory. With some minor figuring, I saw I merely had to throw a few bytes away here and there, watch for nulls to end the loops and that was it. This routine is the slowest of all the included dir reading techniques, but is far faster than other techniques I have seen in basic.

dir + .bas: This is more like a ML directory routine written in basic. After compilation this routine runs almost at ML speed. *(Editor's Note: This routine can be easily in moddified situations where you want to read directory data into basic variables. Just substitute variable definitions which concatenate the ascii value of each byte being read from disk into the string variable of your choice where Ed is calling the kernal's chrout routine via sys co,a. For instance, for reading filenames you might want to subtitute f$ = f$ + chr$(a) for the sys co,a in line 270. Then eliminate line 290 which prints the carriage return.)*

dir.dsm: This is a commented disassembly of an assembly language equivalent of dir.bas +.

All three dir programs use the same technique. Here's a step by step generalized explanation of what's happening:

```
1) OPEN THE DIRECTORY FILE WITH A PATTERN.
2) SET INPUT TO READ FROM THE DISK FILE WE JUST OPENED. (#2)
3) READ AND THROW AWAY THE 1ST FOUR BYTES.
4) READ THE TWO FILESIZE BYTES. (LSB AND MSB).
5) PRINT THE DECIMAL FILESIZE
6) PRINT A SPACE FOR LINE FORMAT PURPOSES.
7) GET AND PRINT BYTES 'UNTIL WE HIT A NULL. (CHR$(0))
8) PRINT A CARRIAGE RETURN TO START A NEW LINE & ENTRY.
9) READ AND THROW AWAY ONE BYTE.

A) READ ONE BYTE.  IF NOT A NULL THEN LOOP TO NUMBER 4.
B) IF BYTE IS A NULL THEN WE HIT THE END OF THE DIR FILE.
C) RESET IO, CLOSE FILES AND END. (OR RTS.)
```

When OPENing your file you can use any valid disk device number. I hope you find these examples useful.

dir.bas:

```
110 INPUT"DIRECTORY PATTERN";DI$:  REM GET DIR PATTERN
120 IF DI$="" THEN END:  REM END PROGRAM IF NO PATTERN INPUT
130 OPEN2,8,0,"$0:"+DI$: REM OPEN DIR 'READ ONLY' FILE
140 GET#2,A$,A$,A$,A$:   REM THROW AWAY 1ST 4 BYTES
150 GET#2,L$,H$:         REM GET LSB AND MSB OF ENTRY SIZE
160 PRINT CHR$(157)ASC(H$)*256+ASC(L$);
161 REM PRINT ENTRY BLOCK SIZE IN DECIMAL
170 GET#2,A$:            REM GET DIR ENTRY BYTE
180 PRINT A$;:           REM PRINT DIR ENTRY BYTE
190 IF A$>"" THEN 170:   REM LOOP IF BYTE <> NULL
200 PRINT:               REM PRINT A CARRIAGE RETURN
210 GET#2,A$,A$:         REM GET NEXT TWO BYTES
220 IF A$>"" THEN150:    REM LOOP IF 2ND BYTE <> NULL
230 CLOSE2:              REM CLOSE FILE
```

dir.bas +:

```
110 INPUT"DIRECTORY PATTERN";DI$: REM GET DIR PATTERN
120 IF DI$="" THEN END:           REM END PROGRAM IF NONE
130 CI=DEC("FFC6"):               REM CHKIN KERNAL ROUTINE
140 CO=DEC("FFD2"):               REM CHROUT
150 CC=DEC("FFCC"):               REM CLRCHN
160 GI=DEC("FFE4"):               REM GETIN
170 HA=DEC("8E32"):               REM HEX TO ASCII
180 KE=212:                       REM KEYSCAN
200 OPEN2,8,0,"$0:"+DI$:          REM OPEN DIR FILE
210 SYSCI,,2:                     REM SET INPUT TO FILE #2
220 SYSGI:SYSGI:SYSGI:SYSGI:      REM DUMP 4 BYTES
230 SYSGI:RREGL:SYSGI:RREGH:      REM GET FILESIZE LSB/MSB
240 SYSHA,H,L:                    REM PRINT FILESIZE BLOCKS
250 SYSCO,32:                     REM PRINT SPACE
260 SYSGI:RREGA:                  REM GET DIR ENTRY BYTE
270 SYSCO,A:                      REM PRINT DIR ENTRY BYTE
280 IF A>0 THEN 260:              REM LOOP TIL NULL
290 SYSCO,13:                     REM PRINT CARRIAGE RETURN
292 IF PEEK(KE)=60 THEN320        REM ABORT ON SPACE
300 SYSGI:SYSGI:RREGA             REM GET NEXT TWO BYTES
310 IF A>0 THEN 230:              REM LOOP TIL EOF NULL
320 SYSCC:CLOSE2:                 REM CLEAR CHANNEL/CLOSE
```

You will find dir.dsm on page 13

continued from page 2

computers. The support corporation is the only authorized liquidator of these machines. The support company sells these machines alone or in various attractively priced system packages. Bundled with each computer are four first class applications programs, a two month subscription to a C-128 bi-weekly (yes, bi-weekly) newsletter, a Q-link disk, and an extensive catalog of Commodore, in-house, and third party software and hardware products marketed by the support company. In addition the company has signed many local Commodore dealerships to handle its products. The company also has a customer support line staffed by competent and helpful people as well as product support sections on Q-link, GEnie, and Compuserve. Commodore showcase's the 128 support company as evidence that it is committed to long term support of its products. As time marches, new and long time 128 owners have few regrets about their 128 purchase, and as their needs and technology evolve they naturally consider a Commodore product.

Does this sound naive and too far-fetched to become reality? Well Apple did not think so! You see the paradigm for the 128 support company is a successful company by the name of Sun Remarketing in Utah. Established with Apple's assistance, Sun Remarketing is the authorized liquidator of the Lisa and Apple III microcomputers. Sun markets a variety of system bundles and produces software for these "forgotten" Apple products. Interestingly enough statistics show that many of Sun's customers either own or will someday own a Macintosh. Beyond the obvious public relations and advertising benefits Sun Remarketing provides for Apple, Apple actually gets a better price for their Lisa's and Apple IIIs than they would from more conventional liquidation channels. Lisa and Apple III owners benefit from the continued hardware and software development while Apple and Sun make money. Everyone wins!

I hope I have tantalized you with these prospects. I want to make it clear that at this point this whole concept is nothing more than an idea. I am however exploring it sincerely. However, please DO NOT send us contributions to finance such an endeavor. I don't want to start a charity or a religion. However, you can help by popularizing this notion. Permission is hereby granted to reprint this proposal (and only this proposal). I urge you to place it in user group newsletters, on local bulletin board systems, and on national telecommunications networks. Show it to your friends and to local Commodore retailers. And above all, please ask Commodore to adopt this idea, or at least to consider it seriously. The key players you should contact are: Irving Gould, Max Toy, David Klein, Richard McIntyre, Julie Bauer, Brian MacDonald, and Jim Gracely. You can write them at: Commodore Business Machines, 1200 Wilson Drive, West Chester PA 19380. As the old song says: *"You can't always get what you want, but if you try, sometimes you get what you need!"*

EASY DIRECTORY READS

continued from page 12

This file shows how programmers can obtain disk directories from within their own programs. All that needs to be done before calling this routine, is opening the dir file like this: open 2,8,0,"$0:pattern"

Program resides from $b00 to $b87. (C128's cassette buffer.)

```
;START OF CODE.
. 00B00  4C 05 0B JMP $0B05 ;JMP TO START

;INTERNAL VARIABLES.
. 00B03  00                 ;THESE TWO BYTES COUNT THE TOTAL
. 00B04  00                 ;DIRECTORY ENTRIES.

;INITIALIZE & SETUP FOR OUR DIR DISPLAY.
. 00B05  D8        CLD       ;CLEAR DECIMAL MATH MODE
. 00B06  20 CC FF  JSR $FFCC ;SET I/O TO DEFAULTS.
. 00B09  A9 FE     LDA #$FE  ;SETUP LSB OF ENTRIES COUNTER.
. 00B0B  8D 04 0B  STA $0B04 ;
. 00B0E  A9 FF     LDA #$FF  ;SETUP MSB OF ENTRIES COUNTER.
. 00B10  8D 03 0B  STA $0B03 ;
. 00B13  A2 02     LDX #$02  ;SET INPUT TO FILE #2.
. 00B15  20 C6 FF  JSR $FFC6 ;
. 00B18  B0 60     BCS $0B7A ;EXIT IF WE HAVE A PROBLEM
                             ;SETTING INPUT CHANNEL.

;DUMP 1ST 4 (THROW AWAY) BYTES OF DIR FILE.
. 00B1A  20 E4 FF  JSR $FFE4 ;
. 00B1D  20 E4 FF  JSR $FFE4 ;
. 00B20  20 E4 FF  JSR $FFE4 ;
. 00B23  20 E4 FF  JSR $FFE4 ;

;GET AND PRINT EACH ENTRIES DECIMAL FILESIZE IN CBM BLOCKS.
. 00B26  20 E4 FF  JSR $FFE4 ;GET FILESIZE (CBM BLOCKS) LSB.
. 00B29  AA        TAX       ;COPY TO THE .X REG
. 00B2A  20 E4 FF  JSR $FFE4 ;GET FILESIZE MSB.
. 00B2D  20 32 8E  JSR $8E32 ;PRINT THE FILESIZE IN BLOCKS.
. 00B30  A9 20     LDA #$20  ;PRINT A SPACE.
. 00B32  20 D2 FF  JSR $FFD2 ;

;READ ACTUAL TEXT OF DIRECTORY ENTRY.
. 00B35  20 E4 FF  JSR $FFE4 ;GET CHARACTER OF DIR ENTRY.
. 00B38  20 D2 FF  JSR $FFD2 ;PRINT DIR ENTRY CHARACTER.
. 00B3B  D0 F8     BNE $0B35 ;LOOP UNTIL WE HIT A NULL
. 00B3D  A9 0D     LDA #$0D  ;PRINT A CARRIAGE RETURN.
. 00B3F  20 D2 FF  JSR $FFD2 ;

;INCREMENT ENTRIES COUNTER.
. 00B42  EE 04 0B  INC $0B04 ;ADD ONE TO ENTRIES COUNTER LSB.
. 00B45  D0 03     BNE $0B4A ;IF NOT TIME TO UPDATE THE
                             ;COUNTER MSB THEN
. 00B47  EE 03 0B  INC $0B03 ;SKIP, ELSE ADD ONE TO MSB.

;CHECK STOP KEY.
. 00B4A  20 E1 FF  JSR $FFE1 ;CHECK STOP KEY.
. 00B4D  D0 02     BNE $0B51 ;IF NO STOP KEY PROCEED W/DIR.
. 00B4F  F0 29     BEQ $0B7A ;IF STOP KEY THEN ABORT & EXIT.

;CHECK FOR END OF DIR FILE.
. 00B51  20 E4 FF  JSR $FFE4 ;GET ONE THROW AWAY BYTE.
. 00B54  20 E4 FF  JSR $FFE4 ;GET EOF CHECK BYTE.
. 00B57  D0 CD     BNE $0B26 ;NOT NULL THEN GET NEXT ENTRY.

;PRINT TOTAL ENTRIES.
. 00B59  AD 03 0B  LDA $0B03 ;GET ENTRIES COUNTER MSB.
. 00B5C  AE 04 0B  LDX $0B04 ;GET ENTRIES COUNTER LSB.
. 00B5F  20 32 8E  JSR $8E32 ;OUTPUT DECIMAL ENTRIES NUMBER.
. 00B62  20 7D FF  JSR $FF7D ;PRINT TEXT TIL NULL IS REACHED.
. 00B65  20 44 49            ;FOLLOWING HEX VALUES REPRESENT:
. 00B68  52                  ;'(SPACE) DIRECTORY ENTRIES.'
. 00B69  45 43
. 00B6B  54
. 00B6C  4F
. 00B6D  52
. 00B6E  59 20 45
. 00B71  4E 54 52
. 00B74  49 45
. 00B76  53
. 00B77  2E 0D 00 ;HERE'S OUR TEXT TERMINATEING NULL.

;THIS IS WHERE WE EXIT!
. 00B7A  20 CC FF  JSR $FFCC ;CALL CLEAR CHANNEL/RESET IO.
. 00B7D  A9 02     LDA #$02  ;CLOSE FILE # 2. (CLOSE 2)
. 00B7F  20 C3 FF  JSR $FFC3 ;
. 00B82  20 E1 FF  JSR $FFE1 ;LOOP IF STOP KEY IS PRESSED.
. 00B85  F0 FB     BEQ $0B82 ;(AVOIDS BREAK MESSAGE.)
. 00B87  60        RTS       ;RETURN TO BASIC, JSR, ETC.
```

HACKING RAMDOS

by Fred Bowen

Fred Bowen, a Senior Systems Software Engineer at Commodore, co-designed the C-128 and co-authored the RAMDOS software for 128 and 64. He is also a frequent contributor to Twin Cities 128.

Often I have been asked about the possibility of using the Commodore RAMdisk with Basic 8, or even finding a way to partition the RAMdisk so that part of it could be used as a RAMdisk and the remainder used for another application.

So on the day after Thanksgiving, faced with snacking on turkey leftovers or hacking on overdue promises, I chose the latter. As it turned out, the hacks were a fairly simple, painless affair. With a single four byte patch to the RamDos, you can partition it. Another four byte patch to Basic 8 makes it compatible with the RamDos *(my apologies to Lou and Dave, but it's not like they don't misuse any of my code!).* While using both these patches will enhance your Basic 8 programming, I have not forgotten the C64 users. The modified RAMDOS.BAS program provided at the end of this article allows you to partition both C-128 and C-64 versions of the RamDisk.

Basic 8 and the RAMdisk
First, let me describe the contention between the RamDisk and Basic 8, two rather large, memory-intensive utilities which make many demands upon the operating system. They were designed without any knowledge of the other, so by all rights they should not be expected to work together! It turns out they are amazingly tolerant of each other. The only conflict arises from both trying to use the same 8-byte hunk of "shared" Ram ($03E4) for a key application-specific, memory-banking subroutine. The following program patches Basic 8 (either the runtime version or the editor version), relocating the downloaded subroutine to the bottom of the system stack, the only available hunk of "shared" Ram I could find without too much effort. You cannot patch Basic 8 at runtime, by that point Basic 8 has already downloaded its code, overlaying the RamDos code, and the damage has already been done.

```
100 REM PATCH BASIC 8 TO ACCOMMODATE RAMDISK
110 :
120 PRINT"ENTER 1 FOR RUNTIME VERSION"
130 PRINT"ENTER 2 FOR EDITOR  VERSION"
140 PRINT
150 INPUT"ENTER VERSION (1-2)";V: IF V<1 OR V>2 THEN 150
160 : IF V=1 THEN F$="RTLB8"
170 : IF V=2 THEN F$="P.BASIC8"
180 : BLOAD (F$),B1,P(DEC("1300")): IF DS THEN PRINT DS$: END
190 BANK 1: PRINT
200 FORI=1 TO 4: READ ADR$,WAS$,IS$
215 : ADR=DEC(ADR$): WAS=DEC(WAS$): IS=DEC(IS$)
210 : IF PEEK(ADR)<>WAS THEN PRINT"UNEXPECTED DATA FOUND": END
220 : POKE(ADR),IS
230 : NEXT
240 SCRATCH (F$): IF DS THEN PRINT DS$: END
250 BSAVE (F$),B1,P(DEC("1300")): IF DS THEN PRINT DS$: END
260 BANK15: PRINT"DONE."
270 :
275 REM  ADR,FROM,TO
280 DATA 1AEE,E4,38, 1AF3,03,01, 1B29,E4,38, 1B2A,03,01
```

True hackers can simply load the module into memory from the Monitor, make the changes per line 280 above, and save it back. Basic 8 starts at $1300 and ends at $6FFC. Another approach you may prefer would be to add the POKEs in the code above (lines 190-230 and line 280) to the Basic routine which boots Basic 8. Do as I did above, checking that the code being replaced is what was expected. This affords some protection in the event there are different versions of Basic 8 floating around of which I am not aware. To use Basic 8 with the RamDisk, first install the RamDisk, select an interface page of 8, and an appropriate drive number. Then load and run the patched version of Basic 8 as you normally would. Unless you have partitioned the RamDisk, you cannot use any part of the REU for Basic 8 without trashing the contents of the RamDisk and possibly crashing the system. Watch out for @buffer commands which define a buffer in external Ram banks (Buffer #2-9 in Basic 8 lingo). You can have the best of both worlds a RamDisk and external Basic 8 buffers, if you partition the RamDisk.

Partitioning the RamDisk
If you can live with a few limitations, it is possible to fool the RamDisk into believing it has fewer 64K Ram banks than it actually has. When the RamDisk is installed, one of the very first things it does is "sniff" the REU to determine how big it is. A simple patch to this routine allows you to specify the number of banks the "sniff" routine "discovers". Any memory banks above this number are then free to use for non-RamDisk purposes. I have re-written the RAMDOS.BAS (see listing on the next page) program which installs the RamDisk. This version will ask if you want to partition the RamDisk, and it works on either C-128 or C-64 systems.

Simply replace your current version of RAMDOS.BAS with this version. This new version will patch the RamDisk only if you answer Yes to the partition query, otherwise it won't change a thing. For C-128 users, it also defaults the interface page to page 8 (the usual default page also conflicts with Basic 8). Page 8 in the C128 is the bottom of the Basic run-time stack, and could cause a real mess if you have very deeply nested loops and things; something to watch, and one of the two big gotchas with this scheme.

C-64 users do not have built-in Ram expander commands (FETCH, STASH, and SWAP), neither do C-128 users when the RamDisk is installed, the second gotcha. It's not possible to patch the RamDisk to use them (did I say impossible? Make that "not easy"). If you are using an application like Basic 8, this is not a big deal. But if you want to run something which uses FETCH/STASH/SWAP commands you must substitute the following Basic/ML equivalents:

```
10 REU=DEC("DF00"):slow
20 POKE REU+7,num bytes low: POKE REU+8,num bytes high
30 POKE REU+2,128 addrs low: POKE REU+3,128 addrs high
40 POKE REU+4,reu addrs low: POKE REU+4,reu addrs high
50 POKE REU+6,reu bank:REM  must be > highest ramdisk bank!
60 SYS mlcode,128 bank,,mode:REM stash=$84 fetch=$85 swap=$86
```

Note that the "128 bank" above should be a real MMU value, not the translated stuff the Basic BANK command uses. The "ml code" should be located in "shared" Ram (below $400), and is nothing fancy, just a copy of the normal kernel DMA code which normally appears at $3F0 when the RamDisk is not present:

```
xxx008         PHP
xxx178         SEI
xxx2AE 00 FF   LDX $FF00
xxx58C 01 DF   STY $DF01
xxx88D 00 FF   STA $FF00
xxxB8E 00 FF   STX $FF00
xxxE28         PLP
xxxF60         RTS
```

So there you have it. I hope this rather sketchy explanation gives you the inspiration if not the means to add versatility to your expanded system. I might add this one disclaimer: compatibility with possible future versions of Basic 8 or the RamDisk cannot be expected. Well, back to the turkey...

you will find the modified RAMDOS.bas program on page 15

MAS 128 & RAMDOS

continued from page 11

However in its current form MAS128 suffers in comparison to other C128 development tools, such as Merlin 128 or the C128 Developer's Package. Partly this evaluation stems from MAS128's relative lack of features associated with the heavyweight packages, such as macros, conditional assembly, and so on. This tends to make MAS128 relatively less suitable for commercial development work.

To some users, particularly novice users, lack of these sometimes confusing features is not necessarily a drawback. But MAS128 also gives the impression of a program designed largely from the perspective of the programmer and his problems, rather than the user and his problems. One example of this is the lack of error checking in certain contexts, one instance of which is actually allowed to trigger a potentially disastrous hardware error (stepping a drive head beyond its limits). Sooner or later a user will accidentally type in an incorrect parameter despite "knowing" not to. A reliance on the user's knowledge of what not to do for error prevention is not especially well-suited to the needs of the novice user. MAS128 has potential, but it has not realized that potential yet. *(Editor's Note: After several conversations with Matthew Montchalin I can't say that I am certain how he intends to distribute his program. It sounds to me like it is going to be a kind of shareware product. However he is advertising the program as if it is a fully commercial venture.)*

C128 Midnight Assembly System, Mountain Wizardry Software, PO Box 66134, Portland, Oregon 97266, (503) 265-2755

HACKING RAMDOS

continued from page 14

```
10 IF P THEN 100
30 PRINT"<clr>";
50 PRINT:PRINT"  RAM DISK INSTALLATION - V112188":PRINT:PRINT
100 U=9: GOSUB 9000: IF R THEN 440
110 INPUT "INSTALL RAM DISK AS UNIT     9<3 cursor lefts>";U
120 : U=INT(U): IF U<4 OR U>30 THEN 110
130 PRINT "RAM DISK INTERFACE PAGE IS   ";P;LEFT$("<6 cursor lefts>",L);: INPUT A
140 : IF F=0 AND (A<2 OR A>207) THEN 130
150 : IF F=1 AND (A<2 OR A>32)  THEN 130
160 : P=A
170 IF NB=0 THEN 230
180 : A$="Y": INPUT "PARTITION RAM EXPANDER       Y<3 cursor lefts>";A$
190 : IF LEFT$(A$,1)<>"Y" THEN 230
200 :  PB=INT(NB/2):PRINT" NUMBER OF BANKS FOR DISK    ";PB;"<4 lefts>";:INPUTPB
210 :   IF PB>NB OR PB<1 THEN 200
220 :  IF F=0 THEN POKESN+3,PB: POKESN+4,76: POKESN+5,134: POKESN+6,126
225 :  IF F=1 THEN POKESN+3,PB: POKESN+4,76: POKESN+5,184: POKESN+6,62
230 A$="Y": INPUT "INITIALIZE RAM DISK         Y<3 cursor lefts>";A$
240 : M=3: IF LEFT$(A$,1)="Y" THEN M=0
260 IF (F=0 AND P=207) OR (F=1 AND P=14) OR (F=1 AND P=8) THEN 290
270 : A$="N": INPUT "<down>CHECK INTERFACE PAGE: ARE YOU SURE  N<3 lefts>";A$
280 : IF LEFT$(A$,1)<>"Y" THEN 30
300 REM  C128  C64      WHAT IT DO
310 REM $2300 $6300 --> INSTALL    RAM DISK
320 REM     3     3 --> RE-INSTALL RAM DISK
330 REM     6     6 --> INSTALL    RAM DISK W/ ARG: UNIT=.A PAGE=.X
340 REM     9     9 --> RE-INSTALL RAM DISK W/ ARG: UNIT=.A PAGE=.X
350 REM     C     C --> DISPLAY COPYRIGHT NOTICE
390 POKE RE,U: POKE RE+1,P :REM  LDA UNIT: LDX PAGE
400 SYS ML+6+M             :REM  (RE)INSTALL RAMDISK, USING UNIT# & PAGE
420 IF F=1 THEN PRINT: DIRECTORY U(U): GRAPHICCLR
430 PRINT
440 PRINT "<down>PRESS ANY KEY TO RETURN TO MENU"
450 GETA$:IFA$=""THEN450
460 LOAD "STARTUP.*",8      :REM  GOODBYE
9000 REM VERIFY PRESENCE OF RAM CARD
9015 IF P THEN 9180
9020 R=57088
9030 FORI=2TO5:POKER+I,I:NEXT
9040 FORI=2TO5:IFPEEK(R+I)<>ITHEN9060
9050 NEXT: R=0
9060 IF R>0 THEN PRINT "<down><rvs> RAM EXPANDER NOT PRESENT ": RETURN
9080 REM DETERMINE IF C64 OR C128
9100 SYS65418                :REM RESTORE SYSTEM VECTORS
9110 F=ABS(PEEK(65533)=255)  :REM F=0 IF C64, F=1 IF C128
9120 IF F THEN BANK 15       :REM SELECT 128 SYSTEM BANK
9130 IF F=0 THEN P=207: L=6: ML=25344: RE=780: F$="RAMDOS64.BIN*"
9140 IF F=1 THEN P=8:   L=5: ML=8960:  RE=6:   F$="RAMDOS128.BIN*"
9150 IF F=0 THEN LOAD F$,8,1
9160 IF F=1 THEN GRAPHIC1,1: BLOAD (F$): IF PEEK(DEC("D7"))=0 THEN GRAPHIC 0
9180 REM VERIFY SIZE OF RAM CARD
9200 NB=0                                  :REM #BANKS=0 IF CANNOT PARTITION
9210 IF F=0 THEN SN=32317: B=24837: V=25741  :REM SNIFF ROUTINE, SIZE, VERSION
9220 IF F=1 THEN SN=15983: B=8453:  V=9357
9230 : FORI=0TO2:V$=V$+CHR$(PEEK(V+I)):NEXT
9240 :  PRINT"RAMDISK VERSION "V$" LOADED": SYSML+12: PRINT
9250 :  IF LEFT$(V$,2)="4."  THEN SYS SN: NB=PEEK(B)
9270 RETURN
```

The Telcom Hacking Primer

by Ed Parry

For anyone that has ever wanted to do some C-128 telecommunications programming, I need not tell you there is not an overwhelming amount of concentrated information available from any one source on the subject. The following is a programmers (non technical as possible) guide and help with RS232 modem programming. First off, let's review a few things we'll need to know up front, like what bits are and how to deal with them.

A byte is made up of 8 bits. Each bit can be thought of as a two position light switch. The switch (our bit) can ONLY be on or off. When the bit is ON, this is called SET. When the bit is OFF, this is called CLEARED.

Each BIT in a byte has an equivalent decimal value, depending on the position of the bit within the byte. Here's some examples:

```
Decimal 1  = 0000 0001        Decimal 32  = 0010 0000
Decimal 2  = 0000 0010        Decimal 64  = 0100 0000
Decimal 4  = 0000 0100        Decimal 128 = 1000 0000
Decimal 8  = 0000 1000        Decimal 255 = 1111 1111
Decimal 16 = 0001 0000
```

Now we need a way to test, set and clear these bits as needed. In our examples we'll use location 56577, since this is the RS232 memory location that we will use most for our modem operations.

To TEST a bit, in this case bit 4, (Determine whether or not the bit is SET or CLEAR?) use the following scheme:

```
if peek(56577) and 16 then 'bit is set':else 'bit not set'.
```

To CLEAR (or mask) a bit (Clears bit regardless of it's current setting) we use the following (where bit 4 decimal value 16 is our target bit to clear):

```
poke 56577, peek(56577) and 255-16
```

To SET a bit (Sets bit regardless of current setting) use:

```
poke 56577, peek(56577) or 16
```

You will need to understand bit handling because RS232 modem signals are reported and handled via bits at two locations within your 128's memory. These locations are 56577 and 56579. Location 56577 is all we'll need worry about for now.

Programmers will also be interested in some other RS232 related memory locations listed below. A full explanation of these locations is covered completely in Mapping The 128 by Otis Cowper and Compute Books. *(Editor's Note: Mapping the 128 is a must for anyone who want to program or gain better insight into the 128)*

Location 2575: RS232 activity flag - Can be used to determine if we have data bits incoming or outgoing. I prefer to use the RS232 IO buffers to determine if data is still pending to be read in (input) or sent out (output). One use for this location can be to hold up a program until ALL data is read in or sent out. For example:

```
wait 2575,1,1 ;pause until all outgoing data is sent out.
wait 2575,2,2 ;pause until all incoming data is received.
```

Location 2576: RS232 control register - This location contains the data used to determine baud rate, word length and stop bits. Normally for BBS/Network ops word length is set to 8 and stop bits set to 1. OPENing a file on device 2 (user port/modem) updates this location.

Location 2577: RS232 Command register - This location determines the type of handshaking (exactly how the modems communicate with each other), duplex (full or half), and parity.

The C-128 offers two forms of handshaking: 3-line and x-line. The easiest and most common to use is 3-line. This forces your receive and send 'lines' high at all times, insuring compatibility with modems that might not support x-line handshaking. Earlier C-128 Rom versions had a bug in x-line handshaking, but is reported to be fixed in the new update roms.

Duplex is normally full for BBS and network operations. Some folks confuse duplex with local screen echo, but this is not quite accurate. It is more of a defacto BBS/Network standard. Full duplex means data can go both ways at the same time. You can be sending and receiving data at the same time in full duplex. In half duplex, data can only travel one way at a time.

Parity is a crude (and often unused) form of error checking. Parity for BBS and network calling should normally be set to none (no parity at all). Other parity settings available include odd (parity bit set to make sure word = ODD number of set bits), even (same as odd but word = EVEN number of set bits), MARK (parity bit ALWAYS set) and SPACE (parity bit always cleared).

Locations 2578 & 2579: Baud rate factor - This location holds the RS232 modem baud rate timing factor. Advanced programmers can use this location to adjust or customize actual baud rates. Maximum baud rate on the C-128 is 2400 baud unless you use and/or write custom RS232 routines yourself. This has been done, by the way! CSTERM128 (part of CSDOS/ARC128) offers functional baud rates up to 9600 baud!

Location 2580: RS232 Status register - This location is used to handle different types of RS232 data errors. IE: Parity error, framing error, receive buffer overflow error, etc. One useful way to use this location is to check the receive buffer empty bit:

```
if peek(2580) and 8 then 'receive buffer is empty':else
'receive buffer is NOT empty'
```

Locations 2582 & 2583: Baud rate timing constant - This location is related to 2578/2579 and should be left to advanced programmers using customized or directly altered baud rates.

Before getting into the RS232 buffer Indexes, I better mention the actual buffers first. The C-128 designers were good enough to set aside permanent RS232 buffer space. (The C-64 has temporary RS232 buffers). The C-128 input buffer is 255 bytes ranging from $c00-$cff and the output buffer is 255 bytes ranging from $d00-$dff. These buffers are always there and ready for our RS232 input/output buffering. Another nice little side feature of these buffers is that we can use the 512 bytes for ML workspace if we are sure no actual RS232 IO will be occurring during our ML routines execution.

The following four locations are RS232 input/ouput buffer indexes. They contain the OFFSET's to the current position within the actual RS232 input/ouput buffers. I have found POKEing and PEEKing these locations very useful as you will soon see.

Location 2584: Index to first character of input buffer - This location contains the offset to the very first character in the input buffer. Reading one byte from the input buffer will increment this index, which is circular. Once the value here reaches 255, it circles back to 0.

Location 2585: Index to last character of input buffer - This location contains the offset to the last character in the input buffer. This location increments every time a new byte is read into the input buffer, and is also circular.

continued on page 17

The Telecom Hacking Primer

continued from page 16

Following are some useful tips for using these two locations:

Tip # 1:
poke 2585, peek(2584): This effectively clears all pending bytes from the input buffer. I have used this to clear the input buffer before file transfers, in BBS key-get routines (in case the user gets over anxious and hits too many keys), etc.

Tip #2:
If peek(2585) <> peek(2584) then 'we have characters to read':else 'no characters to read'

This is another handy way to check if we have data in our RS232 buffer to read or not. If not, we can go do other things (update clocks, count timeout ticks, etc.) I have used this technique in conjunction with a one second timeout routine to read incoming x/ymodem blocks. Long as data is coming, everything is okay. If we timeout for one second (or more) without receiving all the bytes we expected, then we have a short (bad) block.

Tip #3:
Another neat trick is to 'back read' buffer data. Since the indexes and actual buffers are circular, we can go back and read or search the last 255 bytes. This can be handy if you want to search for text strings without buffering them yourself. Keep in mind that each buffer only holds 255 bytes. This is roughly one fourth of your 80 column screen (and not as much data as it sounds like.)

Tip #4:
In a terminal I am currently writing (GeekTerm 128), I found that at 2400 baud, the RS232 input buffer would overflow sometimes. If I tried to check the RS232 input buffer overflow error bit, it would mean the error had already occurred and data loss would be unrecoverable. I tracked the problem down to my status line buffer readout routine. This routine is responsible for updating the Buffer counter for each byte. It was just too slow to keep the RS232 input buffer cleared and do all the updating and math for every byte. Sometimes when data is coming in at full speed at 2400 baud it would overflow the RS232 input buffer. I decided what I needed was an early warning routine. This would detect when the buffer was getting near full, but would leave time to take action before the buffer actually overflowed. After some searching I came up with the ML solution, which goes like this:

```
sec         ;get ready to do some subtraction.
lda $a18    ;get index to first input buffer byte.
sbc #+15    ;subtract 15 from value.
cmp $a19    ;is index to last input buffer chr within 15
            ;bytes of $a18?
bcs OHOH    ;if so, best to skip our routine for now.
bcc OKAY    ;if not, then okay to proceed per norm.
```

OHOH = * ;Means it's better to bypass our routine this time. The buffer reading will catch up quickly. Our routine will get serviced soon as the last character index is greater than 15 bytes away from the first character index.

OKAY = * ;Means our input buffer is not in danger of overflowing. We can proceed with business as usual.

This routine works for non-critical routines only. It is ideal for a buffer output routine, because the reading of input bytes catches up quickly when I skip outputting the adjusted count of the terminal buffer.

Now for the RS232 OUTPUT buffers:
Location 2586: Index to first character in the rs232 output buffer, it's similiar to location 2584, except that it is the offset to the 1st character in the output buffer.

Location 2587: Index to last character in rs232 output buffer - Again, this location is very similiar to location 2585, except that this index contains the offset to last character in the output buffer.

Tip #1:
poke 2587, peek(2586) ;this clears the output buffer.

Tip #2:
do:if peek(2586) <> peek(2587) then loop:'put ouput buffer clear code here'

This is very useful for determining if you are still sending data. Once the output buffer indexes equal each other it means the output buffer is empty.

OPENing RS232 files:
In Basic, this will open your RS232 channel (device 2) at 8 bit wordsize, 1 stop bit, no parity, full duplex and 3-line handshaking. This is the defacto standard that 99% of the BBS's and networks use. If you need other setups, I suggest you read up on location's 2576 and 2577 in Mapping The 128.

```
OPEN 1,2,3,chr$(X)+chr$(0)
where X = 6 for 300 baud, X = 8 for 1200 baud & X = 10 for
2400 baud.

To OPEN the same rs232 channel in ML:
lda #2          ;length of filename.
ldx #<filename  ;lsb of filename address (bank 0).
ldy #>filename  ;msb of filename address (bank 0).
jsr $ffbd       ;call kernal SETNAM routine.

ldx #0          ;set .x to bank 0 for SETBNK.
jsr $ff68       ;call the kernal SETBNK routine.
                ;SETBNK tells OPEN the bank the filename can
                ;be found.

lda #1          ;set file # 1. (This is the same as OPEN
                ;1,2,3, in BASIC).
ldx #2          ;set device # 2 (modem).
ldy #3          ;set secondary address to 3.
jsr $ffba       ;call kernal SETLFS routine.

jsr $ffc0       ;call kernal OPEN routine.
bcs open'error  ;exit if there's a prob OPENing the file.

filename .byte 8,0 ;this is the same as the chr$(8)+chr$(0)
                   ;part of the BASIC open.
```

The main RS232 registers:
CIA # 2 is set up to handle modem RS232 operations. Specifically locations 56577 and 56579. This is where the BIT handling discussed above comes in handy. These two locations use BIT's to handle each RS232 function, since most RS232 functions have only two states (on and off). The only one condition I can think of off hand that could use 2 bits (4 possible combinations) instead of one, would be a baud detection signal for 300, 1200 and 2400 baud. Here's what the bits at 56577 represent:

Pin	Bit	Dec	Function	Description
C	0	1	RXD	Receive data.
D	1	2	RTS	Request to Send.
E	2	4	DTR*	Data Terminal Ready.
F	3	8	RI*	Ring indicator.
H	4	16	DCD*	Data Carrier detect.
J	5	32	MS/AA*	Modem speed, auto answer prime or unused.
K	6	64	CTS	Clear to send.
L	7	128	DSR	Data set ready.

continued on page 18

The Telecom Hacking Primer

continued from page 17

Note: Those marked with an * are the ones we are generally interested in. The others seem to take care of themselves pretty well with 3-line handshaking. Also, not all modems are created equal. Some modems have positive polarity, while others have negative polarity. I have even encountered one that seems to have a combination of both! Polarity simply means that some modems use positive signals, while others use negative signals. Here's a quick test that will indicate your polarity. I suggest disconnecting the phone line for this modem test. Also make sure your modem is plugged into the C-128, is connected to its power source, and is turned on.

```
10 OPEN 1,2,3,CHR$(6)+CHR$(0):PRINT#1,CHR$(13)
11 REM OPEN RS232 AT 300 BAUD
20 P=PEEK(56577) AND 8
21 REM TEST RING INDICATOR BIT FOR POS OR NEG.
30 IF P THEN PRINT "NEGATIVE";:ELSE PRINT "POSITIVE";
40 PRINT" POLARITY."
```

About DTR: Most modems offer two choices for DTR handling: 1) You can have your modem FORCE DTR on at all times, or 2) You can choose to have the computer control DTR.

I prefer the later option, since most modems will drop carrier (hangup) with a simple POKE to drop DTR. The following test requires that you have a carrier signal present. To test if your modem will drop the carrier with the DTR poke try this:

```
poke 56577, abs((peek(56577) and 4) - 4)
```

This is sort of a 'cheater' EOR in basic. Normal EOR (BASIC calls this XOR) flips the current status of bits. If the carrier drops when you do this poke, then your modem is setup to allow the computer to control the DTR line.

The newest generation of 2400 baud modems almost always use EEPROM's (Electronically Erasable Programmable ROM's) to keep track of user defined defaults. (Their version of electronic DIP switches.) These modems require that you set their internal defaults via extended Hayes commands, which are entered from your terminal program or software.

Following are EEPROM modem DTR setting options:

AT &D0: Force DTR always on (ignores computer's DTR signal). This setting should work with any terminal, no matter what is being done (if anything) with the DTR bit from the computer's side.

AT &D1: If DTR is turned off from the computer, this setting causes the modem to drop to the modems command state. The command state means you can enter Hayes commands etc. This is normally allowed only when there is no carrier present with other DTR settings.

AT &D2: Follows the computers DTR state. This setting allows you to 'on hook' (hangup) the modem when computers DTR is turned off. This setting also disables auto answer until the DTR signal is turned back on from the computer. This setting is the one I recommend for terminals and BBS software.

AT &D3: Not all modems support this option. This DTR setting actually resets the modem (just like sending an ATZ!) when the DTR signal is turned off.

The main usage of the DTR line is to prepare the modem for normal operations and/or hang up the line when a carrier is present. Here's how to do this when a carrier is present and turning the DTR bit off does not hangup:
1) Wait one second. (Data cannot be incoming or outgoing.)
2) Type + + +.
3) Wait one second. (Data cannot be incoming/outgoing.)
4) Computer SHOULD respond with "OK" text.
5) Type "ATH(RETURN)".
6) Carrier should drop and modem should respond "OK".

About RI: Ring indicator has a singular function. It tells the computer that someone is calling on the modems phone line. I wish ALL RS232 functions were this easy. When bit 4 (8 decimal) inverts from its normal 'no caller' state, then we know we have an incoming call.

This short program demonstrates how to setup and use the ring indicator bit. RI can only be 8 (bit set) or 0 (bit clear). Once you determine your normal RI setting, testing for a ring is easy!

```
10 REM MAKE SURE NO ONE IS CALLING WHEN YOU 1ST RUN ME.
20 RI = ABS((PEEK(56577) AND 8) - 8)
21 REM DETERMINE DEFAULT RI VALUE.
30 PRINT "READY FOR CALL"
40 IF PEEK(56577) AND 8 <> RI THEN 40
41 REM LOOP UNTIL WE GET CALL.
50 PRINT "WE HAVE A CALL!":END
```

About DCD: Carrier detect also has a singular function. It tells your computer when a carrier is detected on the modem phone line.

In EEPROM modems make sure that you have the correct DCD setting. As with DTR, DCD can be made forced on at all times (which is undesirable for 99% of all applications). The highly suggested EEPROM modem setting is: AT &C1(RETURN) ;this tells your modem that we want the carrier signal to be determined by the actual line signal, and not have the carrier signal forced on by the modem. Also make sure that your if your modem has DIP switches, that you check and insure that your carrier DIP switch is set to the not forced configuration. Testing for a carrier is as easy as testing for a ring:

```
10 REM MAKE SURE NO ONE IS CONNECTED BEFORE RUNNING ME.
20 CA = ABS((PEEK(56577) AND 16) - 16)
21 REM DETERMINE DCD VALUE.
30 IF PEEK(56577) AND 16 <> CA THEN 30
31 REM LOOP ON NO CARRIER.
40 PRINT "WE HAVE A CARRIER!".
```

ABOUT BIT 5: This bit is used for two things with SOME modems, but most modems do not use this bit at all.

The first usage is for the older 300 baud modems like the 1650. A ring could be detected, but the modem would not connect until you set this bit. This was sort of like a "go ahead" and "Let 'em connect" signal. Here's how to allow the connect:

```
poke 56577, peek(56577) or 32:rem set so we can connect!
```

The second usage is for some 300/1200 baud modems. They support different line signals depending on the callers actual connecting baud rate.

For example, some modems left bit 5 (32 decimal) clear when the modem had a 300 baud connect. This same modem would the set bit 5 if it had a 1200 baud connect.

This is by FAR the best and easiest way to determine a caller's connecting baud rate. We can simply test the bit, set our software baud rate accordingly, and proceed. Unfortunately, few 300/1200 modems support this signal, and with the flood of inexpensive good 2400 baud modems, one bit just is not enough.

There are two other possible ways to figure the caller's connecting baud rate:

continued on page 19

The Telecom Hacking Primer

continued from page 18

1) Manually change the baud rate at 5 second intervals, while prompting the caller to hit a certain key (generally Return/Enter) at each baud rate. Only the correct baud rate will allow the caller to see any legible text, so the other baud rate prompts appear as garbage. (Admittedly a minor nuisance.) I like this method because it will work with any modem, not just Hayes compatibles. It also seems to be more reliable than method two, since some modems garble response codes from time to time.

2) Read the response codes. This method is probably the next best method if reading bit 5 is not supported or you have a 2400 baud modem. I have encountered problems getting correct readings with both numeric response codes and text response codes though. It appears that occasionally the response code gets garbled. This method also requires that the user make sure their modem is configured (via Hayes commands) properly. The following command will set up most Hayes compatibles to work satisfactorily with this method:

```
AT X1 V0(RETURN)
```

This enables the extended response codes (to handle baud rate connect messages for bauds greater than 300) and enables the numeric response code method. (less problematic than trying to read text string connect messages in my opinion.)

GETTING AND SENDING CHARACTERS:
Once the modem channel has been OPENed properly, you will need to know how to handle sending and receiving data. This is fairly straightforward in both Basic and assembly:

```
10 OPEN1,2,3,CHR$(8)+CHR$(0):REM OPEN MODEM CHANNEL
20 GET#1,A$:GET B$:REM GET BYTES FROM KEYBOARD AND MODEM.
25 REM NULL'S ("") ARE RETURNED IF NO CHARACTERS DETECTED.
30 IF A$ + B$ = "" THEN 20
31 REM LOOP UNTIL WE GET A CHARACTER.
40 PRINT A$ + B$;:PRINT#1,A$ + B$;:GOTO 20:
41 REM PRINT CHARACTERS & LOOP.
```

This is a simple no frills full duplex, local/remote echo petascii terminal! Enhancing it would require accounting for delete's and special keys, filtering out certain keys and characters (ie: quotes), etc. In assembly the GET CHARACTER's loop would look like this:

```
open1,2,3,chr$(8)+chr$(0) first, then:

(Note: + and - labels are simply temp forward and backward
assembly labels.  This is a feature of Buddy/Power assembler
128 and 64.)

Start = *   ;start of routine.
  jsr $ffcc ;clear channels.
  ldx #1    ;set input to rs232 channel # 1.
  jsr $ffc6 ;
  ldx #2    ;set output to rs232 channel # 1.
  jsr $ffc9 ;

;this is our key-get loop.
- jsr $ffe4 ;get byte from modem.
  bne +     ;if not a null then go print it.
  jsr $eeeb ;get byte from keyboard.
  beq -     ;if a null then loop.

+ jsr $ffd2 ;print byte to modem.
  jsr $c72d ;print byte to screen.

  jsr $ffe1 ;check stop key (so we can escape from the
            ;routine.)
  bne -     ;if no stop key then loop.

  jsr $ffcc ;clear channels.
  rts       ;bring it on home.
```

The above info should be plenty to get you rolling with your own RS232 projects and/or experiments.

Herne Data Systems Ltd.

**Featuring C-128 CP/M software by:
Miklos Garamszeghy**

Jugg'ler-128 Now shipping version 3.4.

Let your C-128 CP/M system read, write and format over 140 types of CP/M disks with this easy to use utility. Only $19.95*

QDisk Now shipping version 2.1.

Now you can use your Quick Brown Box as a non-volatile RAM disk in CP/M mode with this memory resident driver. Only $11.95*

Scramb'ler-128 Now shipping version 2.0

High security data encryption system for C-128 CP/M. Protect all of your vital and confidential data from un-authorized snooping. Only $19.95*

**Herne Data Systems Ltd.
P.O. Box 714, Station 'C'
Toronto, Ont. M6J 3S1
(416) 535-9335**

*US $, including postage & handling

Software worth hunting for!

RLO: MERLIN RELOCATION

by Anton Treuenfels

Jim Butterfield's public domain monitor Supermon64 remains one of my favorite C-64 utilities. One particularly impressive feature is the way it relocates itself to the top of available BASIC memory when first RUN. In an environment where the norm is for most machine language utilities to fight over the same areas of memory space, Supermon64 can be counted on to coexist peacefully with almost everything.

Supermon64 is almost unique partly because tools which make it easier to write such nicely behaved programs are few and far between in the Commodore 8-bit world. Each one thus represents an exercise in patient craftsmanship by the author when most of us would rather just bang out a quick program and require the user to worry about all the details of when and where it can be used.

RLO, the program presented here, is an attempt to alleviate some of the problems involved in writing neighborly machine language programs for the C-64 and C-128 by providing a tool to ease to process.

The basic idea
A common computing problem occurs when an assembler or compiler is asked to create a machine language program but not given all the information required to produce a completed program. For example there may not be any indication of where in memory the completed program will execute, or there may be a label used which is never defined but instead marked "to be filled in later". In these cases what is produced is a "link" file, which is a partially completed object program plus a list of things that need to be adjusted to finish the job. A program called a linker combines this file with one or more other link files which together provide the missing information required to produce the finished program.

Why do it this way? Sometimes because is generally faster to link than to compile or assemble, so that an often-used routine might be kept around in link form instead of source form, or so that only the portion of the source currently being worked on needs to be assembled or compiled. Other times because the operating system determines only at run time where the program will go so as to avoid space conflicts between several programs in memory at once.

The idea of deferring the link process until run time to avoid memory conflicts is not unknown in the Commodore 8-bit world. It is essentially what Supermon64 does every time it is RUN. There is also a variant of the well-known PAL 64 assembler called RPAL which produces relocatable object files which can then be adjusted at run time by a companion program called MLOADER to execute at the top of BASIC memory (hmm, is Supermon64's loader perhaps a variant of MLOADER?).

There are a few C-128 compilers and assemblers capable of producing link files for a companion linker. All of them seem to do so for general production purposes such as cutting down assembly time and so on, and the linkers tend to produce one big absolute object file for later execution. None of the compilers or assemblers seems to provide a "linking loader" after the fashion of MLOADER. RLO is essentially a linking loader for Merlin format link files.

The rlo header
The format of Merlin 128 link files is described in a companion piece elsewhere in this issue (readers may wish to compare this format with the one used by Spinnaker's POWER C package, described in David Godshall's article "The Link Between C and Assembly", Transactor, Vol. 8, Issue 5, March 1988). It is possible, although not much fun, to simulate this format by hand using other assemblers.

RLO functions as a BASIC header program to which any number of link files may be attached. When executed RLO looks for attached link files, completes the linking process, moves the object code to the execution address, and if desired will call an initialization procedure at that address.

RLO extends the normal Merlin link format to include additional information in the Header section. Merlin itself does not provide any method of extending the link format so what this really amounts to is a user convention, an agreement to include certain header data at the start of a program for use by RLO. The normal Merlin linker would consider this part of the program, but RLO strips this extra information off as part of its own link procedure.

The extended header is modelled after the autoboot header used by the BOOT CALL routine in the C-128 Kernel. The first four bytes are part of the normal Merlin file header. All succeeding bytes are part of the extended header and must be provided for by appropriate code at the start of a source file.

Byte	$00	asmlo	assembly address
	$01	asmhi	
	$02	sizlo	object code size
	$03	sizhi	
	$04	'R'	"RLO" signature
	$05	'L'	
	$06	'O'	
	$07	grdlo	guard address
	$08	grdhi	
	$0	exelo	execution address
	$0A	exehi	
	$0B	bank	execution bank
	$0C	type	control flags
	$0D+	title	optional title
	<$FF	$00	terminator

The header extends to the first $00 byte after the type byte and should be less than 256 bytes long including title. RLO considers that the Code section starts immediately after the terminator byte. The "RLO" signature is used to identify an attached link file.

The guard address specifies either an upper or lower limit beyond which object code cannot be placed. It is an upper limit if the object code starts at the execution address, a lower limit if the code ends there. The guard address may be treated as an indirect pointer rather than an absolute value (ie., it may be used to indicate the memory locations that contain the "real" guard address).

The guard address is a secondary method of avoiding memory conflict. For example, code relocated into the free RAM0 block at $1300 on the C-128 should not extend past $1C00, the start of BASIC memory, if for no other reason than because this would overwrite RLO itself. If a specific guard address is not required, the header value should be set to a "don't care" upper limit of $FFFF or lower limit of $0000.

The execution address indicates where the code in the link file should be relocated to. The specified address may represent either the beginning or the end of the relocated code. Additionally, the specified address may be treated as an indirect pointer rather than absolute value.

Execution address indirection is the primary method by which RLO attempts to avoid memory conflicts between programs. For example, multiple programs which specify "use the address currently in the top-of-BASIC pointer" can all be loaded correctly, whereas multiple programs which specify "use the absolute address which is the normal top of BASIC" will overwrite each other.

continued on page 21

RLO: MERLIN RELOCATION

continued from page 20

On the C-128 the bank byte should be one of the 16 defined bank configurations. On the C64 this byte is used directly to set the value of the low three bits ("memory configuration") of the I/O register at address $01. On the C-128 this value is used both when moving and initializing the object code. On the C64 this value is used only when initializing the object code.

At present only four bits are defined in the "type" byte. Undefined bits should be cleared to maintain compatibility with possible future versions:

Bit 6: if this bit is set, RLO will perform a JSR to the execution address (ie., to the code start) immediately after the object code has been moved there. The A-register will contain the low byte of the execution address, the X-register the high byte, the Y-register will be set to $00, and all status flags will be cleared except for interrupt disable, which will be set. The bank configuration will be the one specified by the header bank byte.

Bit 3: if this bit is set, RLO treats the header guard address as an indirect pointer instead of an absolute address.

Bit 1: if this bit is set, RLO treats the header execution address as an indirect pointer instead of an absolute address.

Bit 0: if this bit is set, RLO treats the execution address (absolute or indirect) as the ending address of the object code rather than the start address.

Link file attatchment
The easiest way to attach link files to RLO is to use the DOS "Copy and Concatenate" command. For example, the DOS command "c0:myprogram = rlo128.o,mylinkfile" attaches a copy of "mylinkfile" to a copy of "rlo128.o" and calls the result "myprogram". This program can then be LOADed and RUN like any normal BASIC program (well, almost: the start of the BASIC program text area must be at its "normal" value. On the C-128, this means that no bitmapped graphics area should be allocated when attempting to RUN an RLO-type program).

Unfortunately it is not always possible for things to be done the easy way. This particular method has several limitations: the files involved must all exist on the same disk, only a limited number of files may be concatenated at one time, and the command may be crippled or absent on some mass storage devices. If such a difficulty becomes intolerable, the easiest alternative is to use the "append" file access mode to copy one or more link files onto the end of an RLO header. The following short BASIC example program is the equivalent of "c0:myprogram = rlo128.o,mylinkfile":

```
10 A$="RLO128.O,P,R"
15 B$="MYPROGRAM,P,W"
20 GOSUB 50
25 A$="MYLINKFILE,U,R"
30 B$="MYPROGRAM,P,A"
35 GOSUB 50
40 END
50 OPEN 2,8,2,A$
55 OPEN 3,8,3,B$
60 GET#2,A$
65 SS=ST
70 IF A$="" THEN A$=CHR$(0)
75 PRINT#3,A$;
80 IF SS=0 THEN 60
85 CLOSE 3
90 CLOSE 2:RETURN
```

A really useful program of this type would be much more elaborate, of course, but even as it stands this example will work with the Commodore RAMDOS programs, which do not support the "Concatenate" portion of the DOS "C0:" command.

If neither of these approaches to attachment is feasible, then one other method that might be tried is to build the relevant files into one contiguous block in computer memory and save the block.

About the program
A copy of RLO with attached link files looks to the user like a one-line BASIC program. The C-128 version looks like this:

4864 BANK0:SYS(7189):NEW

The line number is simply a tricky way of allocating space in the free RAM block which starts at 4864 ($1300) on the C-128. No officially defined memory location holds the address of this popular area, but programs which wish to be placed there can use RLO's line number to help avoid space conflicts with each other. The C64 version of RLO similarly uses the address of the free RAM block on that machine (49152, $C000) for its line number.

The SYS call causes execution of the machine language portion of RLO. The NEW statement serves two purposes: first, it eliminates the possiblity of RLO accidentally executing twice (a Bad Thing, since data in memory is modified in a way that can't be done successfully twice), and second, it causes a thorough update of BASIC pointers and propogates any changes that may have been made (for example, if the C64 top-of-BASIC pointer has been changed, the bottom-of-strings pointer also needs to be changed).

RLO at about 1K is fairly sizeable as Commodore loader programs go. Most of its size is due to the decision to allow it to resolve external as well as internal references in a link file. This means that code in an attached link file can refer to addresses in other attached link files and RLO will make it work properly. This allows independent programs which are carefully written and defined to use each other's code and data, but additionally it permits a single program to be written in two or more pieces. This raises such possibilities as placing the main working portion in the high memory of a RAM bank (on the C-128) or "underneath" a ROM (on the C64) and keeping only a small interface portion in the most heavily used areas of memory (eg., the free RAM blocks).

At least in the present case linking is a fairly straightforward process. The object code is relocated simply by adding the difference between the execution and assembly addresses to the assembly address of each internal and external reference. The problem is to figure out what the two values are for each reference. The Merlin link format represents one solution.

RLO links in two passes. The main purpose of the first pass is to build an "entry table" for use in resolving external references. This table consists of all the entry labels found in the Reference sections of all the attached link files together with their assigned absolute addresses. The trickiest part is to correctly find the execution address of each link file since RLO allows this to be specified in several ways. If the header specifies an indirect execution address RLO automatically updates the address at the RAM location given to reflect the size of the object code that will be moved there. This assures that the address is correct for the next link file that might want to use it. This also means that it is not guaranteed that the execution address value can be correctly recomputed on the second pass, so RLO saves it (RLO saves some values for the second pass and recomputes others, mostly because saving all values would require more code to build and manage additional tables. This is certainly possible and in some ways preferable, but at present the idea seems like more work than the potential gain justifies).

continued on page 25

The Write Stuff 128

- 100% ML (40K). 63K text, 16K buffer. Plus utilities.
- Priced to sell. School/Group discounts. Not copy protected.
- Includes both 40 and 80-column versions on 2-sided disk.
- Menu and command driven (menus optional): Help, Edit, Print, Save, Load menus (more in 80-column version)
- 90K of on-disk documentation: 68 help files on disk; 16 help screens in 40-column version; 2 (full screen) in 80-column version; 30+ Tutorials and sample files.
- All 128 keys, including numeric, are put to good use.
- 80-column version preview shows boldface, underlining, italics, subscripts, superscripts, and reverse video characters as such.
- Print to Memory to view entire page up to 250 columns wide in 80-column version (screen scrolls about page).
- Quick preview to check a section of text in up to 250 columns (uses hortizontal scrolling if necessary).
- 60 user-definable keyboard macros: Commodore key plus one other key saves up to 4 keystrokes when executing keyboard commands or when entering embedded format commands.
- Built-in outline generator. Preceed lines with colons to indicate the level of indentation. When printed or previewed, lines will be appropriately numbered (Roman, capital letters, Arabic, lower case) and indented. Print only main headings, or only main heading and subheading, and so on.
- Auto numbering of lists and records in Roman, Arabic, letters
- Includes BB Menu Maker (create 40/80-column disk menus for all your disks), and BB Manual Maker which prints all 68 help files in 2-sided booklet format.
- Delete, transfer, copy text by w,s,p or by block.
- One pass double column output (all or part of text).
- Define 100's of word/phrase macros–reduce typing 25%
- Multi-text areas—have 1-10 docs in memory simultaneously. Divide the text area 50:30:20, or 60:40, or 70:10:10:10....
- Supports 17XX RAM Expanders; with RAMDOS & file copier.
- Supports Gold/Super Graphix interfaces with printer fonts.
- Interface with Superbase and other databases.
- Hunt/Search & Replace. Batch Search & Replace.
- Mail merge for form letters, etc. More flexible than most.
- Split screen option in 80-column version allows you to view two parts of the same document or two different documents and switch between them in a single key stroke.
- Sort on 1-10 columns. Ascending or descending (A-Z, Z-A).
- Insert mode. Autocaps mode capitalizes sentences for you.
- Dvorak<—>QWERTY keyboard toggle.
- Hyphenator--allows easy entry of soft hyphens.
- Memorize/recall/switch between 1-2 positions in text.
- File encryption/decryption for confidential files.
- 21 Function BASIC calculator—insert answer.
- Tabs (0-250). Decimal tabs align numbers on decimal.
- Word/Paragraph count. Set alarm clock for 10-90 minutes.
- Bytes used & bytes free, plus file size in blocks.
- All colors user-definable. Key click, word-wrap toggles.
- All, some, or no keys repeat. User-definable screen font.
- All DOS commands supported. Selective directory listing.
- Disk directory. 1581 Sub-directory support.
- Load/Verify/Save to disk or tape. Merge & Append files.
- Load/Save directly to/from the 16K buffer.
- Create (load/save) muliple printer drivers.
- Load/merge from directory. If not a normal BB file, the file translator is automatically called up.
- View files without affecting text in memory.
- Limit line length to X. Strip extra RETURN marks.
- Load/merge/append disk directories into text.
- Supports drives 8-11 & dual drives (0/1)
- Option to change drive device # from 8 to 9.
- Auto scratch & SAVE feature avoids retyping file name.
- Linked file capacity if 63K is not enough.
- Built-in menu-driven file converter allows you to Read/Write SEQ/PRG/USR files and convert to/from ASCII<–>Screen Code<–>True ASCII (9 file types).
- Built-in file translation for: Speedscript, EasyScript, PaperClip and 11 other wps (translates format commands).
- Works with MACH 128, and other cartridges.
- Special autoboot option: Autoboots custom format file, or user's current "ROUGH DRAFT" file.
- Multi format files. Hold any key down during bootup and "-file x" will be loaded where "x" is the key being pressed.
- Two user-definable keyboard commands. BBW 128 is extensible. A memory map is included to allow machine language programmers to add even more features!
- Dot matrix, RS-232, LQ, & laser printers supported.
- Justification & Right alignment supported.
- Micro justification/line pitch spacing (some printers).
- Screen dump (print help screens).
- Multi-line headers and footers with separate margins.
- Odd/even-page headers/footers/page offset.
- Odd/even page printing (for back to back printing).
- Auto-page numbering—Arabic or Roman numeral.
- Alternate left/right pagination for book style numbering.
- Auto-centering (even double-width). Form feed toggle.
- Multi-copy printing—print multiple labels.
- Print a given page—Skip to page X. Print from cursor.
- Margin release & auto-indent/outdent of paragraphs.
- Relative right/left margins & offset printing.
- Soft/hard hyphens. Active secondary address.
- Auto-linefeed and true ASCII options for transparent mode.
- Forced page & wait options. Conditional forced page.
- Industrial strength printer macros: 32 macros each of which can send up to 32 control codes. Macros can be defined as decimal or ASCII strings ("1=65,66,67" same as "1=abc").
- Special macros allow for creation of custom characters.
- Single character toggles for: <u>Underlining</u>, *Italics* **Boldface**, Condensed, Reverse field, Double-width Emphasized, LQ/DQ toggle, Subscript$_S$, Superscript^S

Plus 6 user-definable toggles

BB Talker 64 *The talking word processor. (Special version for blind with "audio preview" now available.)*

BB Talker 64 is the talking version of BB Writer 64 that works with S.A.M., the Software Automatic Mouth. S.A.M. is included along with a talking story book featuring over 30 nursery rhymes, fables, songs, and fairy tales. All of S.A.M.'s features can be accessed by menu or by embedded commands. For only $5 more, get a copy for your kids (or the kid in you).

QBB Versions *For use with the Quick Brown Box 64K battery-backed CMOS RAM cartridge.*

Autoboots in seconds; allows loading and saving of text to QBB. Alarm can be set to autosave text. Alternate version boots from disk and uses entire QBB for non-volatile text area. Only $10.00 when purchased with regular version (needed for help/sample/tutorial files and reference manual/keyboard overlay). This may be the ultimate in 8-bit word processing.

The Write Stuff 128 with *BB Talker*: $34.95 *The Write Stuff 128* without *BB Talker*: $29.95 Free shipping. MC/VISA/C.O.D.

Special: 64K *Quick Brown Box*, *TWS 128*, plus *TWS 128 QBB Versions* all for only $129.95 (regular price of QBB alone!)

Includes 12-page booklet, 60-page reference manual, and keyboard overlay. Substantial discount when purchased in quanities of 5 or more. Any user or group can become a distributor. For more information, write or call:

Busy Bee Software::P.O. Box 2959::Lompoc, CA 93438 (805) 736-8184 M-F, 11-5 p.m. PST

AID 128 AN RLO WEDGE

by Anton Treuenfels

One of the strong points of programming in Basic on Commodore 8-bit microcomputers has always been the availability of a relatively sophisticated full screen editor, and the C-128 incarnation is the most powerful version yet. Still, it is not beyond improvement. There are a number of useful features which might be added to enhance the editor's basic function of entering and modifying program text, and over the years several widely available utility programs to accomplish this have appeared. For one reason or another I didn't like any of them and set out to write one that behaved exactly the way I wanted it to. I can't pretend that the program presented here is the last word on the subject, but I do hope that it might be considered a useful contribution to the discussion.

Running the program
This version of AID128 is designed to work with RLO, a relocating loader program for Merlin 128 format link files. A copy of AID128.RLO must be appended to a copy of RLO128 in order to function properly. Once attached, the program may be Loaded and Run as if it were a normal Basic program (caution: any graphics area must be deallocated before attempting to Run an RLO-type program).

For example, the following sequence uses Basic 7.0 commands to create and run a copy of AID128, assuming that copies of RLO128 and AID128.RLO already exist on a disk in drive 8:

```
COPY "RLO128" TO "AID128"
CONCAT "AID128.RLO" TO "AID128"
RUN "AID128"
```

Using the program
The first enhancement is bi-directional program text scrolling in Basic immediate mode. Scrolling works on either the 40- or 80-column screen and confines itself to the current window boundaries. Whenever the cursor is at the top or bottom edge of the window further cursor movements bring the previous (if at the top) or succeeding (if at the bottom) program line into view. Exceptions occur at the start or end of program text, where there is no preceeding or succeeding line. In this case the text just scrolls with no new line brought into view. Another exception occurs if there is no program line already visible in the window, in which case scrolling off the bottom will bring the first program line into view, while scrolling off the top brings in the last program line. Combining these two exceptions shows that if a cursor key is held down long enough, the text eventually scrolls back to the point it started from.

Program text scrolling is very handy but introduces a difficulty in finding empty window space to use for entering new text, since scrolling now brings up additional program lines instead of blank lines. One way to create blank space is to press "Clear-Home". The built-in escape sequences of the C-128's screen editor offer other ways: "escape-Q " clears from the cursor to the end of the line, "escape-I" clears the entire line the cursor is on, and "escape-@" clears the entire window below the cursor.

The "Home" and "Clear-Home" keypresses make it easy to move the cursor to the top of the window and begin scrolling backwards through the program code. This is handy enough that similar functions to make it easier to scroll forwards through the program code have been added: "escape-Home" moves the cursor to the bottom left corner of the window and "escape-Clear-Home" clears the window before doing the same. In this version of AID128 these two escape functions work in Basic program mode and within the built-in monitor well as in Basic immediate mode (and are the only part of AID128 that works outside of immediate mode).

All other editor enhancements involve the use of newly defined keywords. AID keywords function in much the same manner as Basic keywords executed in immediate mode. They may appear anywhere on a line but cannot be preceeded by a line number or any other text; AID128 assumes that any line which does not start with an AID keyword is meant to be a normal Basic program line. Keywords may be abbreviated by typing in only as many characters as necessary to distinguish one keyword from another with the last character shifted (in this version this means all keywords can be abbreviated using only one letter). Only the first keyword on a line is recognized and acted upon.

This version of AID128 recognizes three keywords: FIND, CHANGE, and QUIT.

FIND [<line range>] <delim> <search$> [<delim>]
Searches for and lists Basic program lines in which occurrences of the search string are found. There is no particular limit on the length of the search string other than that it must fit on the input line. Three delimiters are recognized: "/" searches for normal text outside of quotes, "." searches for normal text inside of quotes, and "@" searches for tokenized text outside of quotes. If a second delimiter is not present at the end of the search string the entire range will be searched, otherwise the search will be abandoned after the first match. A line range may be specified in the normal Basic manner to limit the search to a specific subset of the program, otherwise the entire program will be searched.

Examples:
FIND /SS
(search entire program for the variable SS)
FIND .CROSS
(search entire program for text strings containing CROSS)
FIND -1000 @GOSUB1500@
(search up to line 1000 for first occurrence of GOSUB1500)

CHANGE [<line range>] <delim> <search$> <delim> [<replace$>] [<delim>]
Behaves exactly like FIND, except that occurrences of the search string are changed to the replace string. <Replace$> is optional in the sense that if it is not present occurrences of <search$> are simply deleted. <Search$> and <replace$> must have the same "quote parity", ie., both must have an even number or both must have an odd number of quote marks. A "syntax error" results if this requirement is not met.

If <replace$> is longer than <search$> it is possible that a single program line could grow larger than 255 characters or that the program as a whole could exceed all available program text space. If a given replacement would cause either of these problems to occur the process is aborted with an error message. The program is undamaged but it may be wise to change occurrences of <replace$> to <search$> before proceeding further.

QUIT deactivates AID128 by restoring various vectors to the values they held when the utility was first activated. AID128 is not removed from the computer's memory however, and if it was the first or only program attached to RLO128 it will usually be possible to reactive it by the sequence:

BANK 0: SYS DEC("1300") <RETURN>.

About the program
RLO breaks AID128 into two pieces, an interface/control piece in the free RAM0 block starting at $1300 and a text manipulation piece at the top of RAM0 Basic program text space. In general the interface/control piece contains code that requires access to the Basic, Kernel, and screen editor ROMS (bank 15), while the text manipulation piece is concerned mostly with Basic program text in RAM0 (bank 0).

continued on page 24

AID 128 AN RLO WEDGE

continued from page 23

Splitting AID128 in this manner minimizes use of the popular free RAM0 block and also permits the program to expand freely in the event new functions are added.

Basic program code scrolling is implemented by diverting the screen editor indirect vectors CTLVEC and SHFVEC. These vectors are part of the routine which outputs characters to the screen. At the point they are taken the output routine has already determined that the PETASCII value of the character in the accumulator is less than 32 (CTLVEC) or greater than 128 (SHFVEC). In Basic immediate mode AID128 looks for a cursor up or down character that is at a window edge and outside quotes. If these conditions are not met, the character is simply sent to the original output routine.

When AID128 decides to scroll the program code, the current window is opened as a logical file and searched for a line number at the start of a logical line. The search proceeds in the direction opposite the scroll, eg., when the cursor moves down (scrolling forwards) the window is searched upwards from bottom to top so as to find the highest numbered line on the screen first. As much as possible the cursor is moved around the screen using defined editor escape and plot routines. An exception occurs when moving from one logical line to another. It is possible to accomplish this in both directions entirely using defined character sequences, but in practice moving downwards by this method is unacceptably slow when large areas of the window are blank. Instead, the cursor is moved from one physical line to the next by the PLOT routine and then the line link table is examined to see if a new logical line has been reached.

If the search finds a line number the preceeding or succeeding line is located (if there is one) in the program text and listed by calling an undocumented Basic ROM routine which lists a single line (this is the only use of a ROM routine or RAM variable for which no reference in the "C128 Programmer's Reference Guide" can be found). In the event that no line number is found a default value is used which causes the first or last line of the program to be listed.

The escape sequences to move the cursor to the bottom of the screen are implemented by diverting the screen editor indirect vector ESCVEC and using the PLOT routine to move the cursor.

One problem with using the screen editor indirect vectors is that they are reset to their original values by a RUN/STOP-RESTORE sequence. Fortunately, this sequence is terminated by jumping through an undisturbed indirect vector which normally points to the Basic warm-start routine. Whenever this jump is taken AID128 checks to see that the character output vectors are still pointing toward itself and resets them if necessary before jumping to the original destination.

Keyword functions are implemented by diverting the Basic indirect vector ICRNCH, which normally points to the tokenizing routine. AID128 scans the input buffer looking for one of its keywords. If one is found control is passed to the appropriate routine, otherwise the line is tokenized normally. Because the scan occurs before tokenization any parameters following the keyword remain in their original form. In general this both simplifies and increases the flexibility of parameter handling.

Perhaps the most unusual feature of this implementation of FIND and CHANGE is the distinction between text "inside" and "outside" quotes. The desirability of this feature was first appreciated when a test run blithely changed both the variable "SS" and the letters "SS" in words such as "CROSS" and "PRESS" to something else. Besides preventing this sort of thing from occurring, another useful result of this distinction is that keyword tokens are not confused with quote mode characters (eg., the token for LOAD has the same value as the "Clear-Home" reverse heart character but a search for one will not produce the other as well).

The practical implementation of this distinction in the match subroutine really only cares about the "quote parity" (odd or even) at the point of the first character matched. This is so commands such as FIND @PRINT"HELLO" function as expected. Note that a search string which begins with a quote mark generally requires an odd parity at that point, so the "/" and "@" delimiters will usually not find anything which starts with a quote mark (eg., FIND ."HELLO" will locate "HELLO" but FIND /"HELLO" will not).

Because in Basic the quote mark is a character with special qualities any CHANGE routine gives rise to a troubling possibility. If a given replace string changes the quote parity of a program line, it will no longer list or execute properly. The possibility is outlawed in this implementation by requiring that the search and replace strings have the same quote parity. This seems to be a sufficient condition to minimize problems, as even fairly tricky games with the screen editor to create unusual search and replace strings have produced the desired results. While this is not absolute proof that problems will not occur, they are reasonably unlikely to occur by accident.

ABOUT THE SOURCE

The accompanying source code program was produced with the Merlin 128 assembler. Like most assemblers, Merlin 128 has its own ideosyncratic set of pseudo-ops. The most important of these will be discussed briefly here.

For convenience both segments of AID128 are produced using the same source file, as this simplifies the process of keeping track of everything. Assembly of one or the other segment is controlled by the flags IFACOD (interface) and TOPCOD (Basic top) and the pseudo-ops DO, ELSE and FIN. DO is a numeric-oriented (as opposed to character-oriented) conditional assembly pseudo-op that permits assembly if its argument evaluates to non-zero. ELSE reverses the assembly sense set by the last DO, and FIN terminates a conditionally assembled section.

The REL pseudo-op instructs Merlin to produce a linkable (as opposed to executable) file. Merlin requires that linkable files be assembled directly to disk and this is accomplished by the DSK pseudo-op. Labels to be made available to the other segment are marked by ENT, while labels that must be imported from the other segment are declared EXT.

The source file produces the link files AID0.O and AID1.0. These are concatenated together to produce AID128.RLO, whic in turn must be concatenated to a copy of RLO128. When run, RLO will resolve the external references and relocate AID128.

AID 128 Source code begins on page 28

RLO: MERLIN RELOCATION

continued from page 21

The second pass resolves each entry in the relocate section of each attached link file. For simplicity resolution of each entry is handled on a full word basis, even if only the high or low byte is required. Internal reference resolution is very straightforward. The only tricky concept is how high byte references are handled. These are the only references for which the low byte of the word value cannot be found in the code itself and instead has to come from the "value" byte of the Relocate section entry. The low byte is necessary when resolving a high byte reference so that the carry flag will be correct for the computation of the high byte of the word (in the opposite situation of resolving a low byte reference there is no way to discover what the high byte of the word is, but this doesn't matter since it wouldn't be used anyway).

External references are a little harder. The "value" byte of the Relocate entry is used to locate a label in the Reference section. The label is then used to locate a value in the entry table built during the first pass. Using the Relocate "value" byte in this way means it cannot be used to resolve high byte references, so in the case of high byte external references the low four bits of the "type" byte indicate the offset (this is why such references must be within seven bytes of the external label's value).

A caution regarding external labels: the two pass design allows both forward and backward reference to external labels in other link files, but only backward references should be made in code initialization sections because the code associated with forward references will not yet be in place at the time initialization is executed.

RLO traps several errors beyond guard address violation. Any error causes a cryptic message to be printed and execution returns to the caller. This facility is intended mostly for developmental debugging purposes.

And so ends this introduction to RLO. For example of RLO's application, see AID 128 in this issue and several other nifty programs in future issues of Twin Cities 128.

RLO source code follows below and continues through page 27
AID 128 source code begins on page 28

Editor's note: This source code is provided for your reference. Please see note in the Price & Progress Report on obtaining Twin Cities 128 programs.

```
* MERLIN 128 LINKING LOADER

* WRITTEN BY ANTON TREUENFELS
* 5248 HORIZON DRIVE
* FRIDLEY, MINNESOTA 55421
* 612/572-8229

* LAST REVISION:  12/01/88

* ASSEMBLY FLAGS

        LST OFF         ;MONITOR LISTING

C128    =    1          ;C128 VERSION
C64     =    1!C128     ;C64 VERSION

* RELOCATION TYPE BYTE FORMAT

WRDFLG =    $80         ;WORD
BHIFLG =    $40         ;HI BYTE
RVSFLG =    $20         ;REVERSE
EXTFLG =    $10         ;EXTERNAL
EXTOFF =    $0F         ;OFFSET

* REFERENCE TYPE BYTE FORMAT

XREF    =   $80         ;EXTERNAL
EREF    =   $40         ;ENTRY
AREF    =   $20         ;ABSOLUTE
LLEN    =   $0F         ;LABEL LENGTH

* RLO HEADER TYPE BYTE FORMAT

RLOINT =    $40         ;CALL INIT
RLOIGD =    $08         ;IND GUARD
RLOIEX =    $02         ;IND EXECUTE
RLODWN =    $01         ;END ADDRESS

HDRSIZ =    13          ;HEADER SIZE

NSPERR =    1           ;NOT SUPPORTED
NRFERR =    2           ;NO REFERENCE
NLBERR =    3           ;NO LABEL
DUPERR =    4           ;DUPLICATE LABEL
BNDERR =    5           ;BOUNDARY
MEMERR =    6           ;OUT OF MEMORY

        DO   C128
ASMADR =    $1C01       ;ASSEMBLY ADDRESS
FRERAM =    $1300       ;FREE RAM BLOCK

BNK00   =   %00111111   ;BANK 0
BNK15   =   %00000000   ;BANK 15

        ELSE            ;C64
ASMADR =    $0801
FRERAM =    $C000

RRR     =   %000        ;RAM-RAM-RAM
IKB     =   %111        ;I/O-KERNEL-BASIC
        FIN

* PROGRAM USAGE

        DUM $24         ;UTILITY
ADDR1   DS  2
ADDR2   DS  2
TEMP    DS  2
        DEND

        DUM $47
RLOTYP DS   1           ;RELOCATE "TYPE"
RLOADR DS   2           ;-> DATA TO RELOCATE
RLOVAL DS   1           ;RELOCATE "VALUE"
OBJSTA DS   2           ;CODE START
OBJSIZ DS   2           ;CODE SIZE
        DEND

        DUM $5A         ;HEADER IMAGE
RHORGN DS   2           ;ASSEMBLY ADDRESS
RHSIZE DS   2           ;OBJECT SIZE
RHSIGN DS   3           ;'RLO' SIGNATURE
RHGARD DS   2           ;GUARD ADDRESS
RHDEST DS   2           ;EXECUTE ADDRESS
RHBANK DS   1           ;EXECUTE BANK
RHTYPE DS   1           ;RELOCATE TYPE
        DEND

        DUM $6A         ;DATA POINTERS
CODSCT DS   2           ;-> OBJECT CODE
RLOSCT DS   2           ;-> RELOCATION TABLE
REFSCT DS   2           ;-> REFERENCE TABLE
ENTDAT DS   2           ;-> ENTRY TABLE
        DEND

* BASIC USAGE

        DO   C128
TXTTOP =    $1210       ;END OF PROGRAM TEXT
MAXTXT =    $1212       ;TOP OF PROGRAM SPACE

        ELSE
TXTTOP =    $2D         ;AKA VARTBL
MAXTXT =    $37         ;AKA MEMTOP
        FIN

* KERNEL USAGE

        DO   C128
        DUM $02         ;"FAR CALL" DATA
BANK    DS  1           ;BANK
PCHI    DS  1           ;ADDRESS
PCLO    DS  1
SREG    DS  1           ;STATUS
AREG    DS  1           ;A-REG
XREG    DS  1           ;X-REG
YREG    DS  1           ;Y-REG
        DEND

STAVEC =    $2B9        ;STORE POINTER LOCATION

        ELSE
        DUM $30C
AREG    DS  1
XREG    DS  1
YREG    DS  1
SREG    DS  1
        DEND

        FIN

* KERNEL ROM

        DO   C128
JSRFAR =    $FF6E       ;BANKED SUBROUTINE CALL
INDFET =    $FF74       ;INDIRECT FETCH
INDSTA =    $FF77       ;INDIRECT STORE
        FIN

BSOUT   =   $FFD2       ;OUTPUT BYTE

* HARDWARE REGISTERS

        DO   C128
MMUCR   =   $FF00       ;MEMORY CONFIGURATION

        ELSE
MEMCTL =    $01
        FIN

********************************

        ORG ASMADR

* BASIC START-UP PROGRAM

LINBAS DA   LINE00      ;LINE LINK
        DA  FRERAM      ;LINE#

        DO   C128
        HEX FE,02       ;'BANK0:'
        TXT '0:'
        HEX 9E          ;'SYS(7189):'
        TXT '(7189):'
        HEX A2          ;'NEW'

        ELSE
        HEX 9E          ;'SYS(2065):'
        TXT '(2065):'
        HEX A2          ;'NEW'
        FIN

        HEX 00          ;EOL

LINE00 DA   0

* RELOCATE CODE BLOCK(S)

RELBLK JSR PASS1        ;BUILD ENTRY TABLE
        BCS :1          ;B:ERROR
        JSR PASS2       ;RELOCATE OBJECT(S)
        BCC :2          ;B:OK
:1      JSR SHWERR
:2      RTS

* PASS ONE:  BUILD ENTRY TABLE

PASS1   LDA TXTTOP      ;PUT TABLE AFTER
        STA ENTDAT      ;LINK FILES
        LDA TXTTOP+1
        STA ENTDAT+1
        LDA #$00        ;CLEAR TABLE
        TAY
```

Source code for RLO Page 2 of 3

Note top half is columns 4, 5, and 6

Bottom half is is columns 7, 8, and 9 respectively.

```
        STA (ENTDAT),Y
        JSR FRSHDR      ;FIRST BLOCK
        BNE :1          ;B:NOT FOUND
]BB1    JSR SETTRU      ;SIZE AND START
        JSR SETEXE      ;EXECUTION ADDRESS
        BCS :1          ;B:BOUNDARY
        JSR SETDLT      ;RELOCATION DELTA
        JSR SAVVAL      ;SAVE
        JSR SETSCT      ;SECTION POINTERS
        JSR ADDLBL      ;BUILD TABLE
        BCS :1          ;B:ERROR
        JSR CHKHDR      ;ANOTHER?
        BEQ ]BB1        ;B:YES
:1      RTS

* PASS TWO:  RELOCATE OBJECT(S)

PASS2   JSR FRSHDR      ;FIRST BLOCK
        BNE :1          ;B:NOT FOUND
]BB1    JSR SETTRU      ;SIZE AND START
        JSR SETSCT      ;SECTION POINTERS
        JSR RELOBJ      ;RELOCATE
        BCS :1          ;B:ERROR

        DO  C128
        LDA #BNK15      ;KERNEL IN
        STA MMUCR
        FIN

        JSR SHWTTL      ;SHOW TITLE
        PHP             ;SAVE STATUS
        SEI             ;DISABLE IRQ

        DO  C64
        JSR MAPRRR      ;ROMS OUT
        FIN

        JSR MOVOBJ      ;MOVE
        JSR INTOBJ      ;INIT

        DO  C128
        LDA #BNK00      ;KERNEL OUT
        STA MMUCR
        ELSE
        JSR MAPIKB      ;ROMS IN
        FIN

        PLP             ;RESTORE STATUS
        JSR SKPREF      ;NEXT BLOCK
        JSR CHKHDR      ;ANOTHER?
        BEQ ]BB1        ;B:YES
:1      RTS

* CHECK FIRST POSSIBLE BLOCK

FRSHDR LDA #<RLOEND
        STA REFSCT
        LDA #>RLOEND
        STA REFSCT+1

* CHECK FOR ATTACHED CODE BLOCK

CHKHDR LDY #HDRSIZ-1
]BB1    LDA (REFSCT),Y ;COPY PRESUMED HEADER
```

```
        STA RHORGN,Y
        DEY
        BPL ]BB1
        LDA RHSIGN      ;CHECK 'RLO'
        CMP #'R'        ;SIGNATURE
        BNE :1
        LDA RHSIGN+1
        CMP #'L'
        BNE :1
        LDA RHSIGN+2
        CMP #'O'
:1      CLC
        RTS

* SAVE ADJUSTED VALUES

SAVVAL LDY #HDRSIZ-1
]BB1    LDA RHORGN,Y   ;SAVE ADJUSTED
        STA (REFSCT),Y ;VALUES
        DEY
        BPL ]BB1
        RTS

* SET "TRUE" START AND SIZE

SETTRU LDY #HDRSIZ-1
]BB1    INY             ;FIND END OF TITLE
        LDA (REFSCT),Y
        BNE ]BB1
        SEC
        TYA
        ADC REFSCT
        STA OBJSTA      ;"TRUE" CODE START
        LDA #$00
        ADC REFSCT+1
        STA OBJSTA+1
        SEC
        TYA
        SBC #4-1        ;STRIP OFF MERLIN HEADER
        STA TEMP
        LDA RHSIZE
        SBC TEMP
        STA OBJSIZ      ;"TRUE" SIZE
        LDA RHSIZE+1
        SBC #$00
        STA OBJSIZ+1
        RTS

* SET EXECUTION ADDRESS

SETEXE LDA RHTYPE
        AND #RLOIGD     ;INDIRECT GUARD?
        BEQ :1          ;B:NO
        LDY #0
        LDA (RHGARD),Y
        TAX
        INY
        LDA (RHGARD),Y
        STX RHGARD
        STA RHGARD+1
:1      LDX RHDEST      ;ASSUME ABSOLUTE
        LDY RHDEST+1
        LDA RHTYPE
        LSR
```

```
        AND #RLOIEX/2   ;INDIRECT EXECUTE?
        BEQ :2          ;B:NO
        STX ADDR1
        STY ADDR1+1
        LDY #0
        LDA (ADDR1),Y
        TAX
        INY
        LDA (ADDR1),Y
        TAY
:2      BCC :3          ;B:GIVEN START
        TXA
        SBC OBJSIZ      ;START BELOW
        TAX
        TYA
        SBC OBJSIZ+1
        TAY
        BCC ERRBND      ;B:BANK UNDERFLOW
        CPX RHGARD      ;RHDEST>= RHGARD?
        SBC RHGARD+1
        BCC ERRBND      ;B:NO
:3      STX RHDEST      ;EXECUTION ADDRESS
        STY RHDEST+1
        BCS :4          ;B:GIVEN END
        TXA
        ADC OBJSIZ      ;END ABOVE
        TAX
        TYA
        ADC OBJSIZ+1
        TAY
        BCS ERRBND      ;B:BANK OVERFLOW
        CPX RHGARD      ;RHDEST< RHGARD?
        SBC RHGARD+1
        BCS ERRBND      ;B:NO
:4      LDA RHTYPE
        AND #RLOIEX     ;INDIRECT EXECUTE?
        BEQ :5          ;B:NO
        TYA             ;UPDATE (RHDEST)
        LDY #1
        STA (ADDR1),Y
        TXA
        DEY
        STA (ADDR1),Y
:5      CLC
        RTS

ERRBND LDA #BNDERR      ;BOUNDARY ERROR
        SEC
        RTS

* SET RELOCATION DELTA VALUE

SETDLT CLC
        LDA TEMP        ;ACCOUNT FOR EXTENDED
        ADC RHORGN      ;HEADER BYTES
        STA RHORGN
        BCC :1
        INC RHORGN+1
:1      SEC
        LDA RHDEST      ;EXECUTION ADDRESS
        SBC RHORGN      ;MINUS ASSEMBLY ADDRESS
        STA RHORGN
        LDA RHDEST+1
        SBC RHORGN+1
```

```
        STA RHORGN+1
        RTS

* SET LINK FILE SECTION POINTERS

SETSCT LDA REFSCT
        LDX REFSCT+1
        CLC
        ADC #4
        BCC :1
        INX
        CLC
:1      STA CODSCT      ;-> OBJECT BASE
        STX CODSCT+1
        ADC RHSIZE
        STA RLOSCT      ;-> RELOCATION TABLE START
        STA REFSCT
        TXA
        ADC RHSIZE+1
        STA RLOSCT+1
        STA REFSCT+1
        JSR CHKEOT      ;ANY RELOCATION TABLE?
        BEQ :2          ;B:NO
]BB1    CLC
        LDA #4
        JSR NXTENT      ;LOOK FOR END
        BNE ]BB1
:2      RTS

* MOVE POINTER TO NEXT POSSIBLE BLOCK

SKPREF JSR CHKEOT       ;ANY REFERENCE TABLE?
        BEQ :1          ;B:NO
]BB1    JSR NXTREF      ;LOOK FOR END
        BNE ]BB1
:1      RTS

* RELOCATE OBJECT CODE

]BB1    JSR LODREL      ;GET RELATIVE
        JSR RELADJ      ;ADJUST OBJECT
        BCS RBJ1        ;B:ERROR
        JSR STOABS      ;STORE ABSOLUTE
        LDA RLOSCT
        ADC #4          ;NEXT TABLE
        STA RLOSCT      ;ENTRY
        BCC RELOBJ
        INC RLOSCT+1
RELOBJ JSR RLOENT       ;GET TABLE ENTRY
        BNE ]BB1        ;B:FOUND ONE
        CLC
RBJ1    RTS

* GET RELATIVE ENTRY

RLOENT LDY #0
        LDA (RLOSCT),Y
        BEQ :1          ;B:EOT
        STA RLOTYP
        CLC
        INY
        LDA (RLOSCT),Y
        ADC CODSCT
        STA RLOADR      ;OFFSET TO BYTE(S)
```

```
        INY             ;TO RELOCATE
        LDA (RLOSCT),Y
        ADC CODSCT+1
        STA RLOADR+1
        INY
        LDA (RLOSCT),Y
        STA RLOVAL
        TYA             ;RESET Z-FLAG
:1      RTS

* ADJUST RELATIVE OBJECT

RELADJ LDA #EXTFLG
        BIT RLOTYP
        BNE :1          ;B:EXTERNAL
        CLC
        LDA RLOVAL      ;OFFSET LO BYTE
        ADC RHORGN
        TAX
        TYA
        ADC RHORGN+1
        TAY
        CLC
        RTS

:1      BVC :2          ;B:NOT HI BYTE
        LDA RLOTYP      ;LOW NYBBLE
        AND #$0F        ;IS OFFSET
        TAX
        AND #$08        ;NEGATIVE?
        BEQ :2          ;B:NO
        TXA
        ORA #$F0        ;SIGN EXTEND
        TAX
:2      LDY #$00
        TXA
        BPL :3
        DEY             ;Y= $FF
:3      STA TEMP        ;OFFSET FROM EXTERNAL
        STY TEMP+1
        JSR FNDEXT      ;LOCATE EXTERNAL
        BCS :4          ;B:NOT FOUND
        TXA
        ADC TEMP        ;ADD OFFSET
        TAX
        TYA
        ADC TEMP+1
        TAY
        CLC
:4      RTS

* LOAD RELATIVE OBJECT

LODREL LDA #RVSFLG
        BIT RLOTYP      ;WHAT'S IN OBJECT CODE?
        BVS LODBHI      ;B:HI BYTE
        BNE LODRVS      ;B:REVERSE WORD
        BMI LODWRD      ;B:WORD

LODBLO LDY #0           ;LO BYTE
        LDA (RLOADR),Y
        TAX
        RTS
```

```
LODBHI LDY #0           ;HI BYTE
        LDA (RLOADR),Y
        TAY
        LDX #0
        RTS

LODWRD LDY #0           ;WORD
        LDA (RLOADR),Y
        TAX
        INY
        LDA (RLOADR),Y
        TAY
        RTS

LODRVS LDY #1           ;REVERSE WORD
        LDA (RLOADR),Y
        TAX
        DEY
        LDA (RLOADR),Y
        TAY
        RTS

* STORE ABSOLUTE OBJECT

STOABS LDA #RVSFLG
        BIT RLOTYP
        BVS STOBHI      ;B:HI BYTE
        BNE STORVS      ;B:REVERSE WORD
        BMI STOWRD      ;B:WORD

STOBLO TXA              ;LO BYTE
        LDY #0
        STA (RLOADR),Y
        RTS

STOBHI TYA              ;HI BYTE
        LDY #0
        STA (RLOADR),Y
        RTS

STOWRD TYA              ;WORD
        LDY #1
        STA (RLOADR),Y
        TXA
        DEY
        STA (RLOADR),Y
        RTS

STORVS TYA              ;REVERSE WORD
        LDY #0
        STA (RLOADR),Y
        TXA
        INY
        STA (RLOADR),Y
        RTS

* FIND EXTERNAL REFERENCE

FNDEXT LDA REFSCT       ;REFERENCE
        STA ADDR1       ;TABLE FOR
        LDA REFSCT+1    ;CURRENT BLOCK
        STA ADDR1+1
        JSR LKUEXT      ;LOOK UP REF
        BCS :1          ;B:NOT FOUND
```

Source code for RLO Page 3 of 3

Note top half is columns 10, 11, and 12

Bottom half is is columns 13 and 14 respectively.

```
        JSR LKULBL      ;LOOK UP VALUE
:1      RTS

* FIND REFERENCE IN REFERENCE TABLE

]BB1    CLC             ;NEXT ENTRY
        TYA
        ADC #3-1
        ADC ADDR1
        STA ADDR1
        BCC LKUEXT
        INC ADDR1+1
LKUEXT  LDY #0
        LDA (ADDR1),Y
        BEQ ERRNRF      ;B:EOT
        TAX
        AND #$0F        ;LABEL LENGTH
        TAY
        INY             ;-> "VALUE"
        TXA
        AND #$F0        ;"TYPE"
        CMP #XREF       ;EXTERNAL?
        BNE ]BB1        ;B:NO
        LDA (ADDR1),Y   ;TARGET?
        CMP RLOVAL
        BNE ]BB1        ;B:NO
        CLC
        RTS

ERRNRF  LDA #NRFERR     ;NO REFERENCE
        SEC
        RTS

* ADD LABELS TO ENTRY TABLE

ADDLBL  JSR CHKEOT      ;END OF TABLE?
        BNE :1          ;B:NO
]BB1    CLC
        RTS

]BB2    LDY #0          ;ENTER LABEL
        LDA (REFSCT),Y
        AND #$0F
        STA (ADDR2),Y   ;ENTER LENGTH
        TAX
]BB3    INY
        LDA (REFSCT),Y
        STA (ADDR2),Y   ;ENTER NAME
        DEX
        BNE ]BB3
        INY             ;ENTER VALUE
        LDA (REFSCT),Y
        ADC RHORGN
        STA (ADDR2),Y
        INY
        LDA (REFSCT),Y
        ADC RHORGN+1
        STA (ADDR2),Y
        INY             ;MARK END
        LDA #$00
        STA (ADDR2),Y
        TAY
]BB4    LDA (REFSCT),Y
        JSR NXTREF      ;NEXT REFERENCE
```

```
        BEQ ]BB1        ;B:END OF TABLE
:1      AND #$F0
        CMP #AREF       ;ABSOLUTE?
        BEQ ERRNSP      ;B:YES
        CMP #EREF       ;ENTRY?
        BNE ]BB4        ;B:NO
        LDA REFSCT
        STA ADDR1
        LDA REFSCT
        STA ADDR1+1
        JSR LKULBL      ;ALREADY DEFINED?
        BCC ERRDUP      ;B:YES
        LDA ADDR2+1
        ADC #1-1
        CMP MAXTXT+1    ;OVERFLOW?
        BCC ]BB2        ;B:NO
        LDA #MEMERR     ;OUT OF MEMORY
        DFB $2C
ERRDUP  LDA #DUPERR     ;DUPLICATE LABEL
        DFB $2C
ERRNSP  LDA #NSPERR     ;NOT SUPPORTED
        DFB $2C
ERRNLB  LDA #NLBERR     ;NO LABEL
        SEC
        RTS

* LOOK UP LABEL IN ENTRY TABLE

LKULBL  LDA ENTDAT      ;TABLE START
        STA ADDR2
        LDA ENTDAT+1
        STA ADDR2+1
        LDY #0
        BEQ :1          ;B:FORCED

]BB1    CLC             ;NEXT ENTRY
        LDY #0
        LDA (ADDR2),Y
        ADC #3
        ADC ADDR2
        STA ADDR2
        BCC :1
        INC ADDR2+1
:1      LDA (ADDR2),Y
        BEQ ERRNLB      ;B:EOT
        TAX             ;SIZE
]BB2    INY
        LDA (ADDR2),Y   ;LABELS MATCH?
        CMP (ADDR1),Y
        BNE ]BB1        ;B:NO
        DEX
        BNE ]BB2
        INY             ;GET VALUE
        LDA (ADDR2),Y
        TAX
        INY
        LDA (ADDR2),Y
        TAY
        CLC
        RTS

* CHECK FOR END-OF-TABLE

CHKEOT  LDY #$00
```

```
        BEQ NXT1        ;B:FORCED

* ADVANCE PAST TABLE ENTRY

NXTREF  CLC
        AND #$0F        ;LABEL LENGTH
        ADC #3          ;THREE OTHER BYTES
NXTENT  ADC REFSCT
        STA REFSCT
        BCC NXT1
        INC REFSCT+1
NXT1    LDA (REFSCT),Y
        BNE :2          ;B:NOT EOT
        INC REFSCT      ;SKIP EOT BYTE
        BNE :1
        INC REFSCT+1
:1      TYA             ;RESET Z-FLAG
:2      RTS

* DISPLAY TITLE

SHWTTL  LDY #HDRSIZ-4
        BNE :1          ;B:FORCED

]BB1    JSR BSOUT       ;OUTPUT
        INY

:1      DO  C128        ;FETCH
        LDX #0
        LDA #<CODSCT
        JSR INDFET

        ELSE
        LDA (CODSCT),Y
        FIN

        BNE ]BB1
        RTS

* MOVE OBJECT CODE

MOVOBJ  DO  C128
        LDA #<ADDR1     ;STORE POINTER LOCATION
        STA STAVEC
        FIN

        LDA RHDEST      ;SAVE EXECUTE ADDRESS
        STA ADDR1
        LDA RHDEST+1
        STA ADDR1+1
        INC OBJSIZ+1    ;ADJUST POINTERS
        LDY OBJSIZ
        BEQ :3          ;B:WHOLE PAGES ONLY
        CLC
        TYA
        ADC OBJSTA      ;TRICKY METHOD TO
        STA OBJSTA      ;DECREMENT POINTERS
        BCS :1          ;BELOW BLOCK STARTS
        DEC OBJSTA+1
:1      CLC
        TYA
        ADC ADDR1
        STA ADDR1
        BCS :2
```

```
        DEC ADDR1+1
        SEC
:2      LDA #$00        ;2'S COMPLEMENT
        SBC OBJSIZ      ;POINT TO START
        TAY             ;OF BLOCK

]BB1    DO  C128
        LDA #<OBJSTA
        LDX #0
        JSR INDFET      ;GET BYTE
        LDX RHBANK
        JSR INDSTA      ;STORE BYTE

        ELSE
        LDA (OBJSTA),Y
        STA (ADDR1),Y
        FIN

        INY
        BNE ]BB1
        INC OBJSTA+1    ;NEXT PAGE
        INC ADDR1+1
:3      DEC OBJSIZ+1
        BNE ]BB1
        RTS

* EXECUTE INIT CODE

INTOBJ  BIT RHTYPE      ;INIT?
        BVC :1          ;B:NO
        LDA #$04        ;SEI
        STA SREG
        LDA RHDEST
        STA AREG
        LDX RHDEST+1
        STX XREG
        LDY #$00
        STY YREG

        DO  C128
        STA PCLO
        STX PCHI
        LDA RHBANK      ;COMES BA
```

```
        DFB $2C
MAPIKB  LDA #IKB            ;I/O-KERNEL-BASIC
MEMMAP  EOR MEMCTL
        AND #%111
        EOR MEMCTL
        STA MEMCTL
        RTS

        FIN

* REPORT ERROR

SHWERR  TAY

        DO  C128
        LDA #BNK15
        STA MMUCR
        FIN

        LDX #RLOEND-RLOERR-1
]BB1    LDA RLOERR,X
        JSR BSOUT
        DEX
        BPL ]BB1
        TYA
        ORA #$30
        JSR BSOUT
        TYA
:1      SEC
        RTS

RLOERR  REV 'RLO ERR #'
        DFB $0D

RLOEND  =   *
```

BONUS DOLLARS

EARN BONUS $$$ WHEN YOU ORDER!!! For every $50 of software and accessories that you order, you earn one bonus dollar. That's an additional 2% discount!! Use your bonus dollars on future purchases.

BRiWALL

SOLID PRODUCTS & SOLID SUPPORT
P.O. BOX 129 58 Noble Street
Kutztown, PA 19530
24 HOURS TOLL FREE
1-800-638-5757

Source code for AID 128 Page 1 of 3

Note top half is columns 1, 2, and 3

Bottom half is is columns 4, 5, and 6 respectively.

```
* C128 BASIC PROGRAMMER'S AID

* WRITTEN BY ANTON TREUENFELS
* 5248 HORIZON DRIVE
* FRIDLEY, MINNESOTA 55421
* 612/572-8229

* LAST REVISION:  12/11/88

* ASSEMBLY FLAGS

        LST OFF         ;MONITOR LISTING
        REL             ;RELOCATABLE

IFACOD =    1           ;INTERFACE CODE
TOPCOD =    1!IFACOD    ;TOP-OF-BASIC CODE

* RLO

RLOINT =    $40         ;CALL INIT
RLOIGD =    $08         ;IND GUARD
RLOIEX =    $02         ;IND EXECUTE
RLODWN =    $01         ;END ADDRESS

FREEND =    $1C00       ;END OF FREE BLOCK
RLOFRE =    $1C03       ;RLO "LINE#"

* CHARACTERS

CR     =    $0D         ;CARRIAGE RETURN
CSD    =    $11         ;CURSOR DOWN
HOM    =    $13         ;HOME
SPC    =    $20         ;SPACE
QUO    =    $22         ;QUOTE

CSU    =    $91         ;CURSOR UP
CLR    =    $93         ;CLEAR/HOME

* ERRORS

ERRSYN =    11          ;SYNTAX
ERRMEM =    16          ;MEMORY
ERRLEN =    23          ;TOO LONG

* BANKS

BNK00  =    %00111111   ;RAM0
BNK15  =    %00000000   ;KERNEL-BASIC-RAM0-I/O

* PROGRAM ZERO-PAGE USAGE (MULTIPURPOSE)

ADDR1  =    $28         ;POINTERS
ADDR2  =    $2A

TEMP   =    $6A
DLMTOK =    $6C         ;DELIMITER TOKEN
QUOPAR =    $6C         ;QUOTE PARITY
KEYCNT =    $6C         ;KEYWORD COUNT
LINPNT =    $6D         ;LINE POINTER
BUFPNT =    $6D         ;INPUT BUFFER POINTER

DLMONE =    $FB         ;DELIMITER OFFSETS
DLMTWO =    $FC
TXTDLT =    $FD         ;TEXT SIZE DIFFERENCE
```

```
* BASIC 7.0 ZERO-PAGE

LINNUM =    $16         ;LINE NUMBER
TXTSTA =    $2D         ;PROGRAM TEXT START
TXTPTR =    $3D         ;TEXT POINTER
LNKPTR =    $61         ;LINE LINK POINTER

* KERNEL & SCREEN EDITOR ZERO-PAGE

MSGFLG =    $9D         ;MESSAGE CONTROL FLAG
SCBOT  =    $E4         ;WINDOW BOTTOM
SCTOP  =    $E5         ;WINDOW TOP
TBLX   =    $EB         ;CURSOR LINE
QTSW   =    $F4         ;QUOTE MODE FLAG
INSRT  =    $F5         ;#PENDING INSERTS

* BASIC 7.0 NONZERO-PAGE

BUF    =    $200        ;INPUT BUFFER

CHRGET =    $380        ;FETCH NEXT PROGRAM CHAR
CHRGOT =    $386        ;FETCH CURRENT PROGRAM CHAR

TXTEND =    $1210       ;PROGRAM TEXT END
TXTTOP =    $1212       ;PROGRAM TEXT LIMIT

* KERNEL & SCREEN EDITOR NONZERO-PAGE

BITABL =    $35E    '   ;LINE LINK TABLE

* BASIC 7.0 INDIRECT VECTORS

IERROR =    $300        ;REPORT ERROR
ICRNCH =    $304        ;TOKENIZE LINE

BASVCT =    $A00        ;GIVE BASIC CONTROL

* KERNEL & SCREEN EDITOR INDIRECT VECTORS

CTLVCT =    $334        ;OUTPUT CONTROL CHAR
SHFVCT =    $336        ;OUTPUT SHIFT CHAR
ESCVCT =    $338        ;OUTPUT ESCAPE CHAR

* BASIC 7.0 ROM ROUTINES

LSTLIN =    $5123       ;LIST LINE

TIELIN =    $AF87       ;LINK LINES
FNDLIN =    $AF8D       ;LOCATE LINE#
ASCLIN =    $AF9F       ;CONVERT ASCII TO LINE#

* SCREEN EDITOR ROM ROUTINES

PLOT   =    $C018       ;GET/SET CURSOR
ESCAPE =    $C01E       ;ESCAPE HANDLER

* KERNEL ROM ROUTINES

LKUPLA =    $FF59       ;LOOKUP FILE#

SETLFS =    $FFBA       ;SET FILE DESCRIPTOR
OPEN   =    $FFC0       ;OPEN FILE
CLOSE  =    $FFC3       ;CLOSE FILE
```

```
CHKIN  =    $FFC6       ;SET INPUT FILE
CLRCHN =    $FFCC       ;SET DEFAULT I/O (AKA CLRCH)
CHROUT =    $FFD2       ;OUTPUT CHAR (AKA BSOUT)
STOP   =    $FFE1       ;CHECK STOP KEY
GETIN  =    $FFE4       ;INPUT BYTE

* HARDWARE REGISTERS

MMUCR  =    $FF00       ;MEMORY MANAGEMENT
LCRA   =    $FF01       ;LOAD CONFIGURATION A
LCRC   =    $FF03       ;LOAD CONFIGURATION C

********************************

        DO  IFACOD      ;INTERFACE

        DSK "AIDO"

* EXTERNAL

GFTEXT EXT              ;GET TEXT
MTCFRS EXT              ;MATCH TEXT
MTCNXT EXT
INSERT EXT              ;INSERT TEXT

* RLO HEADER

ROHEAD TXT 'RLO'
       DA   FREEND      ;STAY IN FREE BLOCK
       DA   RLOFRE      ;RLO ARBITRATES
       DFB 0
       DFB RLOINT.RLOIEX
       DFB CR
       TXT 'C128 BASIC AID V121188'
       DFB CR
       TXT 'BY ANTON TREUENFELS'
       DFB CR,00

* INSTALL AID

INSTAL LDA #>CTLTRP
       CMP CTLVCT+1
       BEQ :1           ;B:DON'T INSTALL TWICE
       CLC
       JSR ALLVCT       ;INSTALL VECTORS
:1     RTS

* RE-INSTALL AFTER RUN-STOP/RESTORE

REINST LDA #>CTLTRP     ;NOT NEEDED IF RETURNING FROM
       CMP CTLVCT+1     ;MONITOR VIA 'X' COMMAND
       BEQ :1
       CLC
       JSR OUTVCT       ;RE-INSTALL OUTPUT VECTORS
:1     JMP (OLDWRM)     ;CONTINUE

* EXECUTE QUIT

EXQUIT SEC              ;FALL

* SET INDIRECT VECTORS

ALLVCT LDX #5*2-1       ;ALL VECTORS
       DFB $2C
```

```
OUTVCT LDX #3*2-1       ;OUTPUT VECTORS
]BB1   LDA RPLVCT-1,X ;GET VECTOR LOCATION
       STA ADDR1
       LDA RPLVCT,X
       STA ADDR1+1
       LDY #1
]BB2   LDA OLDCTL,X
       BCS :1           ;B:RESTORE OLD VECTOR
       LDA (ADDR1),Y    ;SAVE OLD VECTOR
       STA OLDCTL,X
       LDA NEWVCT,X
:1     STA (ADDR1),Y    ;SET NEW VECTOR
       DEX
       DEY
       BPL ]BB2
       TXA
       BPL ]BB1
       RTS

* REPLACEMENT VECTORS

RPLVCT DA   CTLVCT      ;CONTROL OUT
       DA   SHFVCT      ;SHIFT OUT
       DA   ESCVCT      ;ESCAPE OUT
       DA   ICRNCH      ;TOKENIZE
       DA   BASVCT      ;TO BASIC

NEWVCT DA   CTLTRP
       DA   SHFTRP
       DA   ESCTRP
       DA   ACRNCH
       DA   REINST

********************************
* SEARCH FOR AID KEYWORDS
********************************

ACRNCH LDY #-1
]BB1   INY
       LDA BUF,Y        ;EXAMINE INPUT BUFFER
       BEQ XCRNCH       ;B:EMPTY
       CMP #SPC         ;SKIP SPACES
       BEQ ]BB1
       CMP #'A'         ;FIRST NONSPACE ALPHABETIC?
       BCC XCRNCH       ;B:NO
       STY BUFPNT
       LDX #0
       STX KEYCNT       ;KEYWORD COUNT
       DEY              ;BACK UP ONE
       DEX
]BB2   INY              ;NEXT BUFFER CHAR
       INX              ;NEXT KEYWORD CHAR
]BB3   LDA BUF,Y
       SEC
       SBC KEYWRD,X     ;MATCH?
       BEQ ]BB2         ;B:YES
       CMP #$80         ;END-OF-MATCH?
       BEQ HAVKEY       ;B:YES
       LDY BUFPNT       ;START OVER
       INC KEYCNT       ;NEXT KEYWORD
]BB4   INX
       LDA KEYWRD-1,X ;SKIP TO NEXT KEYWORD
       BPL ]BB4
       LDA KEYWRD,X     ;END OF KEYWORDS?
```

```
       BNE ]BB3         ;B:NO
XCRNCH JMP (OLDCRN)     ;TOKENIZE

* FOUND KEYWORD

HAVKEY STY TXTPTR       ;POINT TO LAST CHAR OF MATCH
       LDA #>BUF
       STA TXTPTR+1
       LDA KEYCNT       ;KEYWORD#
       ASL
       TAX
       LDA KEYADR+1,X
       PHA
       LDA KEYADR,X
       PHA
       JMP CHRGET       ;EXECUTE

* RETURN TO BASIC

RETURN LDX #$80         ;NO ERROR
RETERR ENT
       STA LCRC         ;BASIC IN
       JMP (IERROR)

* KEYWORDS

KEYWRD DCI 'FIND'
       DCI 'CHANGE'
       DCI 'QUIT'
       DFB 0            ;END-OF-TABLE

* KEYWORD ADDRESSES

KEYADR DA   EXFIND-1
       DA   EXCHAN-1
       DA   EXQUIT-1

********************************
* EXECUTE KEYWORDS
********************************

* EXECUTE FIND AND CHANGE

EXCHAN LDA #$80
       DFB $2C
EXFIND LDA #$00
       STA FNDCHG       ;FIND/CHANGE FLAG
       JSR LINRNG       ;GET LINE RANGE
       JSR CHRGOT
       STA LCRA         ;RAM0 IN
       JSR GFTEXT       ;GET TEXTS
       JSR RNGLIN       ;CHECK RANGE
       BCS :3           ;B:OUT OF RANGE
]BB1   JSR STOP         ;STOP KEY PRESSED?
       BEQ :3           ;B:YES
       STA LCRA
       JSR MTCFRS       ;MATCH IN LINE?
       BNE :2           ;B:NO
       BIT FNDCHG       ;CHANGE?
       BPL :1           ;B:NO
]BB2   JSR INSERT       ;CHANGE TEXT
       JSR MTCNXT       ;ANOTHER MATCH?
       BEQ ]BB2         ;B:YES
       JSR CHKTIE       ;RELINK
```

```
:1     JSR OUTLIN       ;LIST LINE
       LDA #CR
       JSR CHROUT
       LDA ENDTOK       ;ONLY DO FIRST MATCH?
       BNE :3           ;B:YES
:2     JSR NXTLIN
       BCC ]BB1
:3     JMP RETURN

* SET LINE RANGE

LINRNG JSR CHRGOT
       JSR ASCLIN       ;GET LINE# (= 0 IF NOT DIGIT)
       JSR FNDLIN       ;SET LINK POINTER TO LINE
       JSR CHRGOT
       BEQ :1           ;B:END OF LINE
       CMP #'-'         ;LINE RANGE?
       BNE :1           ;B:NO
       JSR CHRGET
       JSR ASCLIN       ;GET ENDING LINE#
:1     LDA LINNUM       ;AT LEAST ONE LINE# GIVEN?
       ORA LINNUM+1
       BNE :2           ;B:YES
       LDA #$FF
       STA LINNUM       ;IMPOSSIBLY HIGH ENDING LINE#
       STA LINNUM+1
:2     RTS

* CHECK FOR TEXT TOKENIZATION

TOKTXT ENT
       LDA DLMTOK
       CMP #'@'         ;TOKENIZE?
       BNE :1           ;B:NO
       INC TXTPTR
       STA LCRC         ;BASIC IN
       JSR XCRNCH
       STA LCRA         ;RAM0 IN
:1     RTS

* CHECK FOR PROGRAM LINE RELINKING

CHKTIE ENT
       LDA TXTDLT
       ORA TXTDLT+1
       BEQ :1           ;B:NOT NECESSARY
       STA LCRC
       JSR TIELIN       ;RELINK
       STA LCRA
:1     RTS

* SETUP FOR NEXT LINE

NXTLIN JSR GETLNK       ;FOLLOW LINK TO NEXT LINE
       STX LNKPTR
       STA LNKPTR+1

* CHECK IF LINE WITHIN RANGE

RNGLIN JSR GETLNK       ;CHECK LINK TO NEXT LINE
       CMP #$00         ;LINK EXISTS?
       BEQ :1           ;B:NO
       JSR GETLIN       ;GET LINE#
       CMP LINNUM+1
```

Source code for AID 128 Page 2 of 3

Note top half is columns 7, 8, and 9

Bottom half is columns 10, 11, and 12 respectively.

```
        BNE :1          ;CLEAR CARRY IF LESS THAN LIMIT,
        CPX LINNUM      ;SET IF GREATER THAN LIMIT
        BNE :1
        CLC             ;CLEAR CARRY IF EQUAL TO LIMIT
:1      RTS

*********************************
* CHAR OUTPUT TRAPS
*********************************

* CONTROL TRAP

CTLTRP CMP #CSD         ;CURSOR DOWN?
       BNE XCNTRL       ;B:NO
       LDX SCBOT        ;WINDOW BOTTOM
       JSR CURTRP       ;TRAP?
       BEQ HUNTUP       ;B:YES
XCNTRL JMP (OLDCTL)     ;TO USUAL HANDLER

* HUNT UPWARDS FOR LINE#

HUNTUP LDX #-1          ;HUNT UPWARDS
       JSR HNTLIN       ;HUNT FOR LINE#
       BCS :1           ;B:FOUND ONE
       LDX TXTSTA       ;START WITH FIRST PROGRAM LINE
       LDA TXTSTA+1
       BCC :2           ;B:FORCED

:1      JSR GETLNK      ;FOLLOW LINK TO NEXT LINE
:2      STX LNKPTR
        STA LNKPTR+1
        LDA #'V'        ;SCROLL UP
        JSR ESCAPE
        JSR BOTLFT      ;CURSOR TO BOTTOM LEFT CORNER
        JSR GETLNK      ;CHECK LINK OF NEW LINE
        TAX
        BEQ :3          ;B:END-OF-PROGRAM, SO DONE
        JSR OUTLIN      ;LIST LINE
:3      RTS             ;TO NORMAL BSOUT EXIT

* SHIFT TRAP

SHFTRP CMP #CSU         ;CURSOR UP?
       BNE XSHIFT       ;B:NO
       LDX SCTOP        ;WINDOW TOP
       JSR CURTRP       ;TRAP?
       BEQ HUNTDN       ;B:YES
XSHIFT JMP (OLDSHF)

* HUNT DOWNWARD FOR LINE

HUNTDN LDX #1           ;HUNT DOWN
       JSR HNTLIN       ;HUNT FOR LINE#
       BCS :1           ;B:FOUND ONE
       LDX #$00
       TXA
       JSR FNDLNK       ;LOCATE LAST LINK OF PROGRAM
:1     LDA #'W'         ;SCROLL DOWN
       JSR ESCAPE
       JSR TOPLFT       ;CURSOR TO TOP LEFT CORNER
       LDX LNKPTR
       LDA LNKPTR+1
       CMP TXTSTA+1     ;FOUND FIRST LINE OF PROGRAM?
       BNE :2           ;B:NO
```

```
        CPX TXTSTA
        BEQ :3          ;B:YES, SO DONE
:2      JSR FNDLNK      ;LOCATE PREVIOUS LINE
        JSR OUTLIN      ;LIST LINE
:3      RTS

* CHECK CURSOR UP/DOWN TRAP

CURTRP CPX TBLX         ;AT WINDOW EDGE?
       BNE :2           ;B:NO
       BIT MSGFLG       ;CHECK ENABLED KERNEL MESSAGES
       BPL :1           ;B:BASIC PROGRAM RUNNING
       BVS :1           ;B:IN MONITOR
       LDX QTSW         ;QUOTE MODE ACTIVE?
       BNE :2           ;B:YES
       LDX INSRT        ;INSERTS PENDING?
       BEQ :2           ;B:NO
:1     LDX #$FF         ;FLAG NO TRAP
:2     RTS

* LIST PROGRAM LINE

OUTLIN JSR GETLIN       ;GET LINE#
       JSR LSTLIN       ;LIST ONE LINE
       LDA #BNK15
       STA MMUCR        ;RESTORE BANK
       LDA #'O'         ;CANCEL QUOTE, ETC.
       JSR ESCAPE
       RTS

* FIND GIVEN LINE LINK

FNDLNK STX TEMP         ;TARGET LINK VALUE
       STA TEMP+1
       LDX TXTSTA       ;START OF PROGRAM TEXT
       LDA TXTSTA+1
]BB1   STX LNKPTR       ;LINE TO INVESTIGATE
       STA LNKPTR+1
       JSR GETLNK       ;LINK VALUE OF THIS LINE
       CMP TEMP+1       ;TARGET?
       BNE ]BB1         ;B:NO
       CPX TEMP
       BNE ]BB1         ;B:NO
       RTS

* GET LINE LINK / LINE#

GETLIN LDY #2           ;LINE#
       DFB $2C
GETLNK LDY #0           ;LINE LINK
       LDA #BNK00
       STA MMUCR
       LDA (LNKPTR),Y
       TAX
       INY
       LDA (LNKPTR),Y
       LDY #BNK15
       STY MMUCR
       RTS

* HUNT FOR LINE# ON SCREEN

HNTLIN STX HNTDIR       ;DIRECTION
       JSR OPNFIL       ;OPEN SCREEN FILE
```

```
]BB1    LDA #'J'        ;CURSOR TO START OF LOGICAL LINE
        JSR ESCAPE
        JSR CHKLIN      ;LOOK FOR LINE# ON SCREEN
        BCS :1          ;B:NOT FOUND
        JSR FNDLIN      ;LOOK FOR LINE# IN PROGRAM TEXT
        LDA #BNK15
        STA MMUCR       ;RESTORE BANK
        BCS :3          ;B:FOUND
:1      SEC
        JSR PLOT        ;GET CURSOR POSITION
        TXA
        ADC HNTDIR      ;NEXT PHYSICAL ROW
        TAX
        CLC
        JSR PLOT
        BCS :2          ;B:OUTSIDE WINDOW
        LDA TBLX        ;PHYSICAL ROW
        AND #%111
        TAX
        LDA TBLX
        LSR
        LSR
        LSR
        TAY
        LDA BITABL,Y    ;LINE LINK TABLE
        AND MSBSET,X    ;LINKED TO ROW ABOVE?
        BNE :1          ;B:YES
        BEQ ]BB1        ;B:NO

:2      CLC
:3      PHP
        JSR CLSFIL      ;CLOSE SCREEN FILE
        PLP
        RTS

* CHECK SCREEN LINE FOR LINE# AT START

CHKLIN LDX #$00         ;LINE# 0
       STX LINNUM
       STX LINNUM+1
       JSR SCNDIG       ;GET FIRST CHAR
       BCS :2           ;B:NOT DIGIT
]BB1   TAX              ;SAVE DIGIT
       LDA LINNUM+1
       CMP #>64000/10   ;TOO BIG?
       BCS :2           ;B:YES
       STA TEMP
       LDA LINNUM
       ASL
       ROL TEMP
       ASL
       ROL TEMP         ;LINE#*4
       ADC LINNUM
       STA LINNUM
       LDA TEMP
       ADC LINNUM+1     ;LINE#*5
       ASL LINNUM
       ROL              ;LINE#*10
       STA LINNUM+1
       TXA              ;GET DIGIT BACK
       ADC LINNUM       ;ADD DIGIT
       STA LINNUM
       BCC :1
       INC LINNUM+1
```

```
:1      JSR SCNDIG      ;NEXT CHAR
        BCC ]BB1        ;B:DIGIT
        CLC
:2      RTS

* GET DIGIT FROM SCREEN

SCNDIG JSR GETIN        ;GET CHAR
       SEC              ;CHECK IF DIGIT
       SBC #'0'
       CMP #9+1
       RTS

* OPEN TEMP FILE

OPNFIL LDA #127
]BB1   SBC #1-1         ;FIRST TIME = MINUS 0 OR 1
       JSR LKUPLA       ;LOOK FOR UNUSED FILE#
       BCC ]BB1
       STA TMPLA        ;FOUND ONE
       LDX #3           ;SCREEN DEVICE
       LDY #$FF         ;(NOT USED)
       JSR SETLFS
       JSR OPEN         ;OPEN SCREEN FILE
       LDX TMPLA
       JSR CHKIN        ;SET FOR INPUT
       RTS

* CLOSE TEMP FILE

CLSFIL JSR CLRCHN       ;DEFAULT I/O
       LDA TMPLA
       JSR CLOSE
       RTS

* ESCAPE TRAP

ESCTRP CMP #CLR
       BEQ ESCCLR
       CMP #HOM
       BEQ BOTLFT
XESCAP JMP (OLDESC)

* ESCAPE CLEAR/HOME

ESCCLR JSR XSHIFT       ;CLEAR WINDOW
BOTLFT SEC              ;CURSOR TO BOTTOM LEFT CORNER
       LDA SCBOT
       SBC SCTOP
       TAX
       DFB $2C
TOPLFT LDX #0           ;CURSOR TO TOP LEFT CORNER
       LDY #0
       CLC
       JSR PLOT         ;SET CURSOR
       RTS

* BIT TABLES

MSBSET HEX 80,40,20,10,08,04,02,01

* VARIABLE STORAGE

PADBYT DS  PADBYT-ROHEAD&$0001
```

```
OLDCTL DS  2            ;CONTROL
OLDSHF DS  2            ;SHIFT
OLDESC DS  2            ;ESCAPE
OLDCRN DS  2            ;CRUNCH
OLDWRM DS  2            ;TO BASIC

FNDCHG ENT
       DS  1            ;FIND/CHANGE FLAG
ENDTOK ENT
       DS  1            ;FIND/CHANGE END DELIMITER

TMPLA  =   FNDCHG       ;TEMP FILE#
HNTDIR =   ENDTOK       ;LINE HUNT DIRECTION

*********************************

       ELSE             ;TOP-OF-BASIC

       DSK "AID1"

* EXTERNAL

FNDCHG EXT              ;VARIABLES
ENDTOK EXT

RETERR EXT              ;ERROR
TOKTXT EXT              ;TOKENIZE
CHKTIE EXT              ;RELINK

* RLO HEADER

R1HEAD TXT 'RLO'
       DA  TXTEND       ;STAY ABOVE RLO
       DA  TXTTOP       ;TOP-OF-BASIC
       DFB 0
       DFB RLOIGD.RLOIEX.RLODWN
       DFB $00

* GET FIND/CHANGE TEXT

GFTEXT ENT
       LDX #$01
       CMP #'.'         ;TEXT INSIDE QUOTES
       BEQ :1
       DEX              ;X= $00
       CMP #'/'         ;TEXT OUTSIDE QUOTES
       BEQ :1
       CMP #'@'         ;TOKENIZED TEXT
       BNE SYNERR
:1     STA DLMTOK       ;DELIMITER
       STX DLMPAR       ;QUOTE PARITY
       LDY TXTPTR
       STY DLMONE       ;FIRST DELIMITER POSITION
       BIT FNDCHG       ;CHANGE?
       BPL :2           ;B:NO
       JSR MTCDLM       ;MATCH DELIMITER
       TXA              ;FOUND SECOND DELIMITER?
       BEQ SYNERR       ;B:NO
       STY DLMTWO       ;SECOND DELIMITER POSITION
       JSR TOKTXT       ;TOKENIZE
       LDY DLMTWO
       STY TXTPTR
:2     JSR MTCDLM
```

```
        STX ENDTOK      ;SAVE TOKEN OR $00
        JSR TOKTXT
        BIT FNDCHG      ;CHANGE?
        BPL :3          ;B:NO
        JSR GTXDLT      ;LENGTH AND PARITY DELTAS
:3      RTS

* REPORT ERROR

SYNERR LDX #ERRSYN
       JMP RETERR

* MATCH DELIMITER

MTCDLM INY
       LDX BUF,Y
       BEQ :1
       CPX DLMTOK
       BNE MTCDLM
       LDA #$00
       STA BUF,Y
:1     RTS

* FIND LENGTH AND QUOTE PARITY DIFFERENCE

GTXDLT LDX #2
]BB1   LDA #$00         ;PARITY STARTS EVEN
       STA TEMP+1,X
       LDY DLMONE-1,X
]BB2   CMP #QUO         ;EVERY QUOTE CHANGES PARITY
       BNE :1
       INC TEMP+1,X
:1     INY
       LDA BUF,Y
       BNE ]BB2
       SEC
       TYA
       SBC DLMONE-1,X ;GIVES LENGTH+1
       STA TEMP-1,X
       DEX
       BNE ]BB1
       LDA TEMP+1       ;'CHANGE' LENGTH
       SBC TEMP         ;'FIND' LENGTH
       BCS :2           ;B:POSITIVE
       DEX              ;X= $FF
:2     STA TXTDLT
       STX TXTDLT+1     ;SIGN EXTENSION
       LDA TEMP+3
       EOR TEMP+2       ;QUOTE PARITIES MATCH?
       LSR
       BCS SYNERR       ;B:NO
       LDA DLMPAR       ;PARITY AT FIRST CHAR OF MATCH
       LDY DLMONE
       LDX BUF+1,Y      ;THE IDEA IS THAT THE PARITY
       CPX #QUO         ;AT THE LAST CHAR BEFORE THE
       BNE :3           ;START OF THE MATCH PLUS THE
       EOR #$01         ;PARITY OF THE REPLACEMENT TEXT
:3     EOR TEMP+3       ;EQUALS THE FINAL PARITY
       STA RPLPAR
       RTS

* SEARCH PROGRAM LINE FOR FIND MATCH

MTCFRS ENT
```

Source code for AID 128 Page 3 of 3

Note top half is columns 13, 14, and 15

```
        LDY #4          ;FROM LINE START
        STY QUOPAR      ;EVEN PARITY
        DEY
        STY LINPNT
MTCNXT  ENT
        LDY LINPNT
        LDX DLMONE      ;TRY TO MATCH FIRST FIND CHAR
]BB1    INY
        LDA (LNKPTR),Y
        BEQ MTC2        ;B:END OF PROGRAM LINE
        CMP #QUO        ;QUOTE PARITY CHANGE?
        BNE MTC1        ;B:NO
        INC QUOPAR
MTC1    CMP BUF+1,X
        BNE ]BB1        ;B:NO MATCH
        LDA QUOPAR      ;CURRENT QUOTE PARITY MATCHES
        EOR DLMPAR      ;DESIRED QUOTE PARITY?
        AND #%1
        BNE ]BB1        ;B:NO
        STY LINPNT      ;SAVE POSITION
]BB2    INX             ;TRY TO MATCH REST OF FIND TEXT
        LDA BUF+1,X
        BEQ MTC3        ;B:MATCH COMPLETE
        INY
        CMP (LNKPTR),Y
        BEQ ]BB2        ;B:CONTINUE
        LDA (LNKPTR),Y ;AT END OF PROGRAM LINE?
        BNE MTCNXT      ;B:NO
MTC2    LDA #$FF        ;FLAG NO MATCH FOUND
MTC3    RTS

* INSERT TEXT

INSERT ENT
        LDA TXTDLT
        ORA TXTDLT+1
        BEQ :1          ;B:NO SIZE CHANGE
        JSR CHGMOV      ;SET UP BLOCK MOVE
        JSR CHKCHG      ;VERIFY SIZES OK
        JSR BLKMOV      ;MOVE BLOCK
:1      LDY LINPNT      ;REPLACE DEST
        LDX DLMTWO      ;REPLACE SOURCE
        LDA BUF+1,X
        BEQ :2          ;B:NO REPLACE
]BB1    STA (LNKPTR),Y ;REPLACE TEXT
        INX
        INY
        LDA BUF+1,X
        BNE ]BB1
:2      DEY
        STY LINPNT      ;POINT TO RESUME MATCHING
        LDA RPLPAR      ;UPDATE QUOTE PARITY
        STA QUOPAR
        RTS

* SET TEXT MOVE PARAMETERS

CHGMOV SEC
        TYA             ;LNKPTR+ENDOFMATCH+1= SOURCE
        ADC LNKPTR
        STA ADDR1
        LDA #$00
        ADC LNKPTR+1
        STA ADDR1+1
        LDA TXTDLT      ;TXTDLT+SOURCE= DEST
        ADC ADDR1
        STA ADDR2
        LDA TXTDLT+1
        ADC ADDR1+1
        STA ADDR2+1
        SEC
        LDA TXTEND      ;TXTEND-SOURCE= COUNT
        SBC ADDR1
        STA TEMP
        LDA TXTEND+1
        SBC ADDR1+1
        STA TEMP+1
        RTS

* CHECK TEXT CHANGE SIZES

CHKCHG LDY TXTDLT+1
        BMI :1          ;B:TEXT CONTRACTS
        LDA (LNKPTR),Y ;Y= $00
        SBC LNKPTR      ;OLD LENGTH
        ADC TXTDLT      ;NEW LENGTH
        BCS LENERR      ;B:TOO LONG
:1      CLC
        LDA TXTDLT
        ADC TXTEND      ;NEW PROGRAM SIZE
        TAX
        TYA
        ADC TXTEND+1
        TAY
        CPX TXTTOP      ;SPACE LEFT?
        SBC TXTTOP+1
        BCS MEMERR      ;B:NO
        STX TXTEND      ;NEW END OF TEXT
        STY TXTEND+1
        RTS

* REPORT ERRORS

LENERR LDA #ERRLEN
        DFB $2C
MEMERR LDA #ERRMEM
        PHA
        JSR CHKTIE      ;RELINK
        PLA
        TAX
        JMP RETERR

* MOVE BLOCK

BLKMOV LDA ADDR2+1
        CMP ADDR1+1
        BNE :1
        LDA ADDR2
        CMP ADDR1
        BEQ MVU1        ;B:SOURCE=DEST, NO MOVE
:1      BCC MOVEDN      ;B:SOURCE>DEST

* MOVE BLOCK UP

MOVEUP LDX TEMP+1       ;#WHOLE PAGES
        BEQ :1          ;B:NONE
        CLC
        TXA
        ADC ADDR1+1     ;POINT TO BLOCK END
        STA ADDR1+1
        TXA
        ADC ADDR2+1
        STA ADDR2+1
:1      INX             ;ADJUST FOR PARTIAL PAGE
        LDY TEMP        ;PARTIAL PAGE COUNT
        BEQ :3          ;B:WHOLE PAGES ONLY
        TYA
        LSR             ;CHECK PARITY
        BCS :2          ;B:ODD
]BB1    DEY
        LDA (ADDR1),Y  ;ODD BYTE
        STA (ADDR2),Y
:2      DEY
        LDA (ADDR1),Y  ;EVEN BYTE
        STA (ADDR2),Y
        TYA
        BNE ]BB1
:3      DEC ADDR1+1     ;NEXT PAGE
        DEC ADDR2+1
        DEX
        BNE ]BB1
MVU1    RTS

* MOVE BLOCK DOWN

MOVEDN LDX TEMP+1       ;#WHOLE PAGES
        INX
        LDY TEMP
        BEQ :4
        CLC
        TYA
        ADC ADDR1       ;POINT BELOW BLOCK
        STA ADDR1
        BCS :1
        DEC ADDR1+1
:1      CLC
        TYA
        ADC ADDR2
        STA ADDR2
        BCS :2
        DEC ADDR2+1
        SEC
:2      LDA #$00
        SBC TEMP
        TAY             ;POINT TO START OF BLOCK
        LSR             ;CHECK PARITY
        BCS :3          ;B:ODD
]BB1    LDA (ADDR1),Y  ;EVEN BYTE
        STA (ADDR2),Y
        INY
:3      LDA (ADDR1),Y  ;ODD BYTE
        STA (ADDR2),Y
        INY
        BNE ]BB1
        INC ADDR1+1     ;NEXT PAGE
        INC ADDR2+1
:4      DEX
        BNE ]BB1
        RTS

* VARIABLE STORAGE

PADBYT DS  PADBYT-R1HEAD&$0001

DLMPAR DS  1            ;DELIMITER QUOTE PARITY
RPLPAR DS  1            ;REPLACE QUOTE PARITY

        FIN
```

News Maker 128

Desk top publishing program for the Commodore 128D computer and the Commodore 128 computer with 64K video RAM upgrade. News Maker 128 supports the Commodore 128D in its native 80 column mode! News Maker 128 can be used to create professional looking newsletters, reports, signs and posters. News Maker 128 can be used as a stand alone program, in combination with a word processor or in combination with graphics software. Full page layout, text pouring, pop down menus, smooth screen scrolling, font selection, cut, paste, mirror, flip are among the features available with News Maker 128. The 1750 REU and 1581 Disk Drive are optional equipment supported.

C128D desk top publishing for only $29.95!

Sketchpad 128

Sketchpad 128 fully supports your C128 and takes advantage of its crisp 80 column graphics capabilities. It is packed with all the features of a professional drawing package such as drawing smooth freehand lines, 3D solids, creating slideshows, cut & paste, clip, flip, enlarge, shaded fill patterns, a variety of fonts, air brush and more! It supports Printshop graphics and is completely compatible with all BASIC 8 files.

Sketchpad 128 unleashes the graphics power of your C128! It supports your 1351 mouse, 64K video chip, 1581 drive and 80 column display. What more could any real C128 user ask for?

ONLY $29.95

See your local dealer or order from:
Free Spirit Software, Inc.
P.O. Box 128/58 Noble Street
Kutztown, PA 19530
(215)683-5609

Home Designer

When the details are important'

Home Designer CAD 128

Given glowing ratings by every major Commodore magazine, this CAD system outclasses every other CAD program, because of its object-based design. With over 50 powerful commands, 5 drawing layers, superb support of library figures and lazer-quality printouts at ANY scale on your dot matrix printer or plotter, you can create drawings so accurate that a blueprint can be made from them!

Tired of working with poor quality/inaccurate printouts, manipulating little dots on a bit-map, giving up on detailed work because you can't zoom in close enough? Join the professionals!

only $45.00

mouse or joystick required
NEW!Engineering Library disk available now. **$10.00**

BASIC 8 IS BACK!!

!NEW PACKAGING!
!NEW SUPPORT!

This popular package adds extensive 80 column graphics capabilities to your C128. A must for C128 programmers! As an added bonus several preprogammed BASIC 8 applications, such as BASIC PAINT, WRITE and CALC, are included on the flip side!

JUST $29

Free Spirit Software Inc

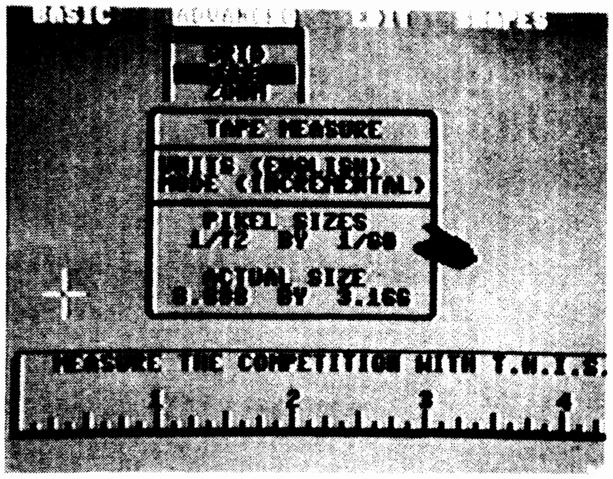
1541/1571 DRIVE ALIGNMENT

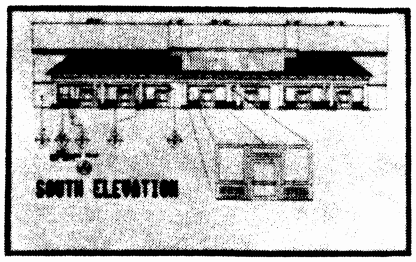
1541/1571 Drive Alignment

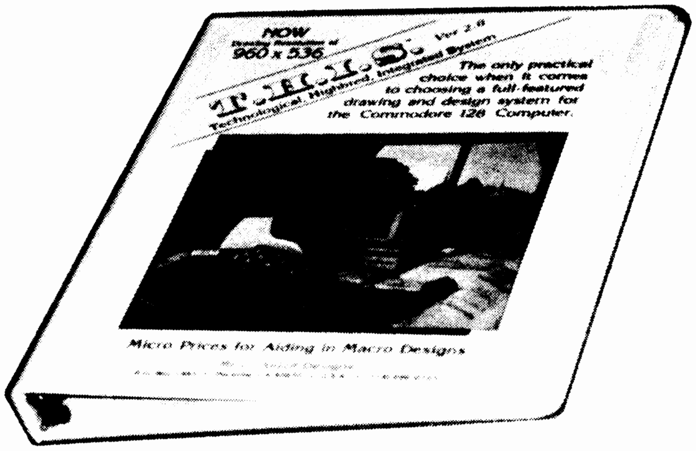
This excellent alignment program is a must have for every Commodore owner. Easy to use, it helps you to align your drive so that it runs just as if it were new! The simple instructional manual and on-screen help prompt you thru the alignment procedure and help you adjust the speed and head stop of your drives. It even includes instructions on how to load the alignment program when nothing else will load. Don't be caught short! We get more RED LABEL orders for this program, then any other program we sell. Save yourself the expense! Order now, keep it in your library and use the 1541/1571 Drive Alignment program regularly!! Works on the C64, C128, and SX64 for both the 1541 and 1571 drives.

STILL ONLY $25.00

Micro Aided Designs
P.O. Box 1982
Placentia, CA 92670
(714) 996-0723






NOW
960 x 536
T.H.I.S. Ver 2.0
Technological Highbred Integrated System
The only practical choice when it comes to choosing a full-featured drawing and design system for the Commodore 128 Computer.
Micro Prices for Aiding in Macro Designs

T.H.I.S.

THIS (Technological Highbred Integrated System) is a program designed exclusively for the Commodore 128. Listed as one of the best programs of 1987 by "Commodore Magazine," THIS *contains many advanced features not found in any other program.* THIS *proves to be the most user-friendly graphics program, complete with a step-by-step easy to understand manual. Using a 1700 or 1750 RAM and a light pen or mouse,* THIS *is a drawing system capable of producing a simple doodle or a sophisticated blue print.*

With THIS *you have a possible resolution of 960 by 536 dots per drawing page and seven (7) totally separate pages to draw upon. Giving the user a larger area to draw on not only allows for larger sized drawings but also increases the amount of details that can be added.*

When it comes to the final step of printing a complete drawing, THIS *offers exact-scaling. With exact-scaling the completed drawing can be printed to scale within 1/64 of an inch.* THIS *allows for the possibilities of templates to be created and used in real life applications. The printout sizes can vary anywhere between 2 by 2 inches to 480 by 321 inches.*

EXCLUSIVE FEATURES

- *Up to 7 pages of graphic designs*
- *Maximum virtual screen resolution of 960 by 536 pixels per page*
- *Exactly scales printouts as large as 480 by 321 inches*
- *Real time object rotation and move*
- *Real time cut, copy, paste, and zoom*
- *Elastic modes include lines, boxes, circles, ellipses, arcs, and other geometric shapes*
- *Easily accessed graphics library including over 100 detailed electronic symbols*
- *Over 700 type sizes*
- *10 font styles*

Now Just $49.95!

MEASURE THE COMPETITION WITH T.H.I.S.

1 2 3 4 5 6 7 8

Our Third Year In Business!

We Know How To Service Our Customers!

BRiWALL

SOLID PRODUCTS & SOLID SUPPORT

P.O. Box 129 / 58 Noble Street
Kutztown, PA 19530

24 HOURS — TOLL FREE
1-800-638-5757
OUTSIDE USA CALL 1-215-683-5433

SATISFACTION GUARANTEED

Earn Bonus $$

No Surcharge On Charge Orders

Friendly Service

ACCESSORIES

40/80 Column Switch Cable	18
Apro Ext.64-Crt. Port Ribbon	17
Apro Ext.64-User Port Ribbon	19
Aprospan 4 Slot Cartrdg Holder	25
Cover 1541	7
Cover 1571	9
Cover C128	9
Cover C64	8
Drive Box	27
Final Cartridge V3	49
Hotshot Plus Interface	69
Joystick-Bat Handle	22
Joystick-Ergo Stick	22
Joystick-Super 3 Way	25
Joystick Y Adapter	7
Leroy's Cheat Sheets C64 each	3
Leroy's Cheat Sheets C128 each	7
Lightpen-Model 170c	69
Lightpen-Model 184c	49
M-3 Mouse 64/128-Proport.	39
Mach 128 Cartridge	35
Mousepad	8
Paper Banner Colors each	10
Power Supply-C128-Repairable	55
Power Supply-C64-Repairable	39
Quick Brown Box 32K	79
Quick Brown Box 64K	99
RGB to RCA Cable	9
RS232 Deluxe Interface	39
Ribbons for Printer	Call
Ribbon Renew Re-Ink w/no eqp.	5
Serial Box 2 for 1	29
Serial to Serial Cable-6 ft.	9
Super Chips 128	45
Super Chips 128-D	45
Super Chips 64	25
Super Chips 64 mode on 128	25
Super Graphix Gold Prntr Intrfc	89
Warpspeed 128	35

BOOKS

1541 Troubleshoot & Repair Guide	17
1571 Internals*	17
Anatomy of C64*	17
Anatomy of the 1541 Book	17
Basic 7.0 Internals Book	19
Basic 7.0 for C128 Book	10
Beginner's Guide to BASIC 8	15
C128 Assembly Language Prog Book	14
C128 Internals*	17
C128 Programmer's Ref. Book	20
C128 Tips & Tricks Book	17
C128 Troubleshooting & Repair Book	17
C64 Basic Programming	25
C64 Programmer's Reference Guide	17
C64 Tips & Tricks*	17
C64 Troubleshoot & Repair Guide	17
C54/128 Asmbly Lang. Prog Book	14
GEOS Get Most Book	9
GEOS Inside & Out*	17
GEOS Programmer's Ref. Guide	17
GEOS Tips & Tricks*	15
Hot to Program in BASIC	24
I Speak Basic to My C64 Book	8
K Jax Book Revealed 2	23
K Jax Book Revealed 1	23
Machine Language 64 Book	13
Superbase The Book 64/128	15
Troubleshoot & Repair C64 Book	18
Twin Cities 128 Compendium 1	15
Abacus Disks each*	12

GENERAL PRODUCTIVITY

Bankstreet Writer	35
Business Form Shop/64	30
CMS Accounting/128	129
CMS Inventory Module 128	53
Chartpak 128	29
Chartpak 64	29
Datamanager 128	39
Datamanager 2 (64)	17
Fleet System 2 + /64	39
Fleet System 4	52
Fontmaster 128 w/Speller	39
Fontmaster II/64	35
KFS Accountant 128	109
Leroy's Label Maker	24
Microlawyer/64	39
Paperclip 3	39
Paperclip Publisher	39
Partner 128	39
Partner 64	30
Personal Portfolio Manager 128	42
Personal Portfolio Manager 64	29
Pocket Dictionary 64/128	10
Pocket Filer 2	29
Pocket Planner 2	29
Pocket Superpak 2	67
Pocket Writer 2	39
Security Analyst 128	35
Superbase 128	39
Superbase 64	29
Sprbase/Sprscrpt/Book 128 PAK	69
Sprbase/Sprscrpt/Book 64 PAK	59
Superscript 128	29
Superscript C64	29
Swiftcalc 128 w/Sideways	39
Swiftcalc 64 w/Sideways	17
Sylvia Porter's Financial Plan 128	39
Technical Analysis System 128	42
Technical Analysis System 64	29
Timeworks Account Payable 64	39
Timeworks Accounts Receiveable 64	39
Timeworks General Ledger 64	39
Timeworks Inventory Managemnt 64	39
Timeworks Payroll 64	39
Timeworks Sales Analysis 64	39
Vizistar 128	60
Viziwrite 128	60
Wordpro 128 w/Speller w/Filepro	30
Wordpro 64 w/Speller w/Turbo Load	30
Wordwriter 128	39
Wordwriter 3 for C64	30
Write Stuff 64	19
Write Stuff 64 w/Talk	24
Write Stuff C128 Version	24

OUR POLICY

Our policy is to stock what we advertise and carry the best products available for your C64 and C128 computers. Over 400 of the best productivity, educational & technical software and a host of accessories in stock now! And CHECK OUT OUR C128 LINE! Your will be amazed at the number of products that we carry for this fantastic machine!

EDUCATIONAL

Alphabet Zoo	18
Alphabuild	6
Calculus by Numeric Methods	27
Counting Parade	6
Designasaurus	
Early Learning Friends	6
Easy Sign	17
Evelyn Wood Dynamic Reader	17
Facemaker	16
First Men on Moon Math	19
Grandma's House	6
Hayden SAT Preparation	32
Jungle Book (Reading 2)	19
Kids on Keys	18
Kidwriter	22
Kindercomp	18
Linkword: German	19
Linkword: French	19
Linkword: French 2	19
Linkword: Italian	19
Linkword: Russian	19
Linkword: Spanish	19
Little Computer People	12
Mathbusters	6
Mavis Beacon Teaches Typing	30
Peter & Wolf Music	19
Peter Rabbit (Reading 1)	19
Protutor Accounting 128	69
R.S.V.P.	21
Sky Travel	35
Stickybear ABC's	22
Stickybear Math 1	22
Stickybear Math 2	22
Stickybear Numbers	22
Stickybear Opposites	22
Stickybear Reading 1	22
Stickybear Reading Comprehension	22
Stickybear Shapes	22
Stickybear Spellgrabber	22
Stickybear Townbuilder	22
Stickybear Typing	22
Success w/Math Series each	23
Toy Shop 64	22
WhereinEurope is Carmen Sandiego	29
Where in USA is Carmen Sandiego	29
Where in World is Carmen Sandiego	29
Widham Classics-Alice/Wonderland	32
Widham Classics-Below the Root	12
Widham Classics-Swiss Famly Robs	12
Widham Classics-Treasure Island	12
Widham Classics-Wizard of OZ	12

GEOS

Becker Basic for GEOS 64	35
Desk Pak Plus-Geos	22
FontPak Plus-GEOS	22
GEOS 64 v 2.0	39
GEOS Programmer/64	45
GEOS Write Workshop/64	39
GEOS Write Workshop/128	45
Geocalc 128	45
Geocalc/64	35
Geofile 128	45
Geofile/64	39
Geopublish 64	45
Geos/128	45
Geospell 64/128	19
Wordpublisher 64/128 (for GEOS)	28

CREATIVITY

Animation Station	65
Award Maker Plus/C64	29
Billboard Maker/64	25
Blazing Paddles	25
Bumpersticker Maker	35
Business Card Maker	29
Button & Badge Maker	39
Cad 3D/64	39
Cadpak 128	42
Cadpak/64	29
Certificate Maker	17
Certificate Maker-Library	12
Colorez 128	12
Computer Eyes	105
Create a Calendar	22
Doodle/64	29
Flexidraw 5.5/64	29
Flexifont/64	24
Graphics Art Disk 7-12 each	9
Graphics Galleria Pak 1	24
Graphics Galleria Pak 2	24
Graphics Intergrator 2/64	24
Home Designer	45
Home Designer/Circuit Symbol Lib	10
Icon Factory/64	25
Label Wizard	21
Newsmaker 128	24
Newsroom	17
Newsroom Clip Art Disk 1-3 each	13
PC Board Maker 64	79
Perspectives II/64(3D)	39
Photo Finish	19
Postcards	19
Printmaster-Fantasy Art Gallery	15
Printmaster Gallery I/64	15
Printmaster Gallery II/64	15
Printmaster Plus/64	25
Printshop/64	35
Printshop Companion	27
Screen F/X	25
Sketchpad 128 NEW	24
Slideshow Creator	13
Video Title Shop w/Gr Comp 1	25
Video Title Shop s/Gr Comp 2	17

PERSONAL

Boston Bartender's Guide	12
Bridge 5.0	22
Cardio Exercise	75
Cardio Exercise & Heartlab Combo	85
Celebrity Cookbook/64	19
Dr. Ruth's Book of Good Sex	22
Dream Machine Analyze/64	19
Family Tree 128	39
Family Tree 64	39
Heartlab	49
MK America Cooks Chinese	12
MK America Cooks Italian	12
MK Great Chefs of PBS Vol 1	12
MK Great Chefs of PBS Vol 2	12
MK Great Chefs of PBS Vol 3	12
Memory Academy 128	15
Micro Kitchen Companion	29
Monopoly C64	25
Muscle Development Package	54
Sexual Edge/64	19
Stress Reduction-Enhanced	299
Stress Reduction-Standard	89
Strider's Classic 1-10 each	5
Tarot 128	15

UTILITIES

1541/1571 Drive Alignment	25
1581 Toolkit	32
Assembler/Monitor/64	29
Basic 8	25
Basic Compiler/128	42
Basic Compiler/64	29
Big Blue Reader 64/128	32
Bobsterm Pro/128	42
Bobsterm Pro/64	35
CP/M Kit	22
Cobol/128	29
Cobol/64	29
Gnome Kit 64/128	29
Gnome Speed Compiler 128	29
Kracker Jax Super Cat	22
Kracker Jax-Hacker's Utilities	17
Merlin 128	45
Merlin 64	35
Oxford Pascal 128	29
Oxford Pascal 64	25
Physical Exam 1541	29
Physical Exam 1571	29
Power Assembler 64/128	30
Power C 64/128	30
Programmers Toolbox/64	15
Protolinc BBS 128	29
Prototerm 128	12
RamDOS 128	29
RenegadeCopier V2	29
Super 64 Librarian	19
Super 81 Utilities for C128	29
Super 81 Utilities for C64	29
Super Aide 64	25
Super C 128	45
Super C 64	45
Super Disk Librarian for C128	19
Super Disk Utilities 128	25
Super Pascal 128	45
Super Pascal 64	45
Super Snapshot 4 w/C128 Disable	65
Super Snapshot V4	59
SysRES Enhanced	30
Z 3 Plus	59

ENTERTAINMENT

AD&D Dungeon Master Assistance	22
AD&D Pool of Radiance	29
Alf	12
Battleship	24
Blackjack Academy	29
California Raisins	19
Caveman Ugh*lympics	22
Chessmaster 2001	30
Crossword Magic	39
Double Dragon	27
Faery Tales	39
Jeopardy 2	12
Jordan vs Bird	24
Lord of the Rising Sun	25
Main Frame	25
Monopoly	24
Newromancer	29
Ocean Ranger	0
Operation Wolf	29
Platoon	25
Powerplay Hockey	24
Rampage	27
Rocket Ranger	27
Roger Rabbit	15
Scrabble	26
Scruples	29
Skate or Die	22
TV Sports Football	25
Three Stooges	27
Ultima V	42
Wheel of Fortune 2	12
Zach Macrakin	25
Zoom	22