ISSN 0737-9161

LDOS QUARTERLY

July 1, 1983 Volume 2, Number 3

LOGICAL SYSTEMS INC.

Table of Contents

ABOUT OUR COVER: LSI has been experimenting with a new photomagnetic emulsion transfer process. The cover is a graphic representation of a 5.1.3 master diskette. This is similar to the process by which small plastic records are printed and inserted into magazines. If anyone is able to successfully read this cover, please let us know so that we can use the information to refine the process.

8970 N. 55th Street P.O. Box 23956
Milwaukee, Wisconsin 53223
Main switchboard: (414) 355-5454
LDOS Hotline: (414) 355-4463

The LDOS Quarterly policy on the submission and payment for articles is as follows:

Articles sent for consideration must be submitted in the following format:

1. A cover letter, summarizing the content and intent of the article.
2. A printed hardcopy (lineprinted or typewritten) of the article.

A diskette with--

3. A 'plain vanilla' ASCII text file containing the article. This file should be formatted as 87 characters per line, and 62 lines per page, with no headers or footers. Do NOT send SuperScripsit or Newscript files. Also, please do not embed print effects.
4. If the article involves assembly language programs, include both the source code, and the object code.
5. Any other necessary files or patches should also be supplied in machine readable form.

Please do not send in printed text without a diskette, as it will NOT be considered for publication. Payment will be made in the form of a LSI product, or $49 per published page in the current Quarterly format. The size of the article will determine the value of the LSI product available as payment.

Please include your name, address, telephone number and LDOS serial number with your submission, firmly attached to your hardcopy printout, and affixed to the diskette you submit.

LSI is extremely interested in seeing submissions from our users, and is open to suggestions on any ideas for the Quarterly.

Submissions should be sent to:

The LDOS Quarterly Editor
c/o Logical Systems, Inc.
8970 N. 55th Street
P.O. Box 23956
Milwaukee, Wisconsin 53223

Trademark Acknowledgements:

LDOS™ and smal-LDOS™ are registered trademarks of Logical Systems, Inc.
TRS-80™ and TRSDOS™ are registered trademarks of Radio Shack/Tandy Corporation.
MAX-80™ is a registered trademark of LOBO Systems, Inc.
CP/M™ is a registered trademark of Digital Research, Inc.

V I E W F R 0 M T H E B 0 T T 0 M F L 0 0 R

by Bill Schroeder

It's Quarterly time again, a little early because we want to have this issue ready to give out at our open house. In any case, here we go...

We have had a great response to the new column by **Rich Hilliard** (Let Us Assemble, Vol 2, No. 2). His writing style and method of imparting tutorial information has been very well received. For this reason, we intend this column to continue as a regular long term feature of the LDOS QUARTERLY. The second part of this ongoing series is to be found in this issue. I'm sure that you will find it very informative and fun to read. I've seen the outlines for many of the upcoming issues and feel that this column will grow to have a large following.

We have also started a new column in this issue that will deal with assembly interfacing to LDOS 6.0 type systems. This column will examine the use of the SVC system present in the 6.0 environment and give examples and in-depth discussions of opsys functions. This technically informative column will be written by our own **Les Mikesell.** I am sure this new feature will also develop an avid clan of followers.

Speaking of large followings, I am often asked "HOW MANY LDOSes ARE OUT THERE??". Of course, the exact number is considered confidential, and is known to only a select few. But I will say that there are well over 40,000 legitimate (registered) LDOS users. There are, of course, untold thousands of pirate users. Trying to get any type of service from LSI if you are not a properly registered owner is a very interesting experience, which usually ends with the pirate's support coming directly from our attorney.

We have a new person entrusted with the day to day operation of our customer service department. This person is **JOE KYLE-DIPIETROPAOLO** (Try that fast, ten times). If you have not dealt with Joe as of yet, you will be pleasantly surprised when the time comes, as he is proving to be a very competent individual.

Let me mention just a couple of other things about customer service. First, the TRSDOS 6.0 product that is delivered with the Radio Shack Model 4 computer is supported directly by Radio Shack, **NOT** LSI.

We receive many calls daily requesting help and information on TRSDOS 6.0, the BASIC provided with the machine, software running on the Model 4, and the hardware itself. In most cases we can not be of much help, due to contractual obligations and company policies. HOWEVER, our customer service people will help with Model 4 related questions and problems under the following conditions. You must first have attempted to get the problem resolved or question answered through the normal channels of the Radio Shack Customer Service program. The Radio Shack Customer Service program is an excellent program, and is a lot better than most people give it credit. Give it a chance to work.

We get a lot of calls from Model 4 users that have not even checked with their Radio Shack store, or with Customer Service in Fort Worth, yet they insist that Radio Shack can't, won't or are unable to help them. This could not be further from the truth. In the vast majority of cases, the normal service channels through Radio Shack will provide complete satisfaction, given the chance.

If you make an effort to solve the problem through Radio Shack, but are unable to get satisfaction, then by all means contact our customer service department, and we will help. We do **insist** on one thing, and that is that you know who (individuals' names and departments) were unable to assist you and their reason for why they were unable to do so.

In some cases, it is possible that the information you need has simply not been provided to that individual, or that LSI has inadvertently not provided the needed information to

Radio Shack at all. If we know exactly where information is lacking, and who is responsible, the problem can be addressed and corrected.

Technical information about the Model 4 hardware and the TRSDOS 6.0 system should be available very shortly. This should clarify many of the technical questions. This publication is very detailed, and jammed with complete facts, examples and electronic information. This publication is called the MODEL 4 TECHNICAL REFERENCE GUIDE, and will be available as Radio Shack part number 26-2110 at a very nominal cost. So hold those technical questions a few weeks, as the answers should be in that publication.

Now another Customer Service point. As LSI grows, we are confronted with ever changing condition and must adjust policies accordingly. To avoid frequent interruptions of our programming staff, a new policy has been set in motion at LSI. If any customer service person is unable to provide an answer to a technical question, he will not run around the building trying to locate our resident expert in that particular area. Instead, he will take down complete details on a special form, and tell you when he expects to have a complete and accurate answer to your query.

To provide those answers, we now have general customer service meetings twice a week (Mondays and Thursdays). At one of these meetings, your query will be presented. Discussions among our staff will result in the best possible answer or solution. We wil1 try to provide the best of service to you in this manner. So far, the results have been excellent. We feel that we are now better serving our users. These meetings have proved of great value in "cross-educating" ourselves here at LSI. The next time the same question arises, all of our customer service personnel will have the correct response without needing to consult with others.

Now on to a different matter entirely. LSI has always been against such concepts as "SOFTWARE LIBRARIES", and User's Groups that exist mainly to share copyrighted software. We have found, however, that the small percentage of pirates and thieves among our customers is costing LSI an unacceptable amount of revenue. As a systems software company, it would be very easy for us to develop many diverse schemes to make it very difficult to copy or backup our products. Up until now, we have resisted this method of protecting our work, and preferred to rely on the integrity of our customers, and to provide excel lent support and value with our products. We had thought that if the user really got a good product, at a fair price, from a company that would stand behind the product, that we would be subject to very little abuse by pirate activities. NOT SO!!!

We have reconsidered our position on behalf of our honest, loyal customers and have found that without some kind of protection, it is THE HONEST CUSTOMER who is being hurt by pirate activities. Our customer service lines are often ALL busy, and customers are told they will have to call back. For this we apologize. However, a sizable portion of our customer service department's time is being taken up by persons that have stolen our products. We have several ways to determine when we are servicing, or have serviced an illegal copy of our product. For obvious reasons, I would rather not go into how we are able to flush out these pirate products.

The result to our legitimate users is that the department that is set up to assist them also must deal with this pirate problem. The honest customer suffers the loss of the availability of this technician. The legal, honest customer is suffering. There is also the lost revenue to LSI which will affect present and future pricing on products. Again the more illegal duplication, the more the honest users suffer.

It is with this in mind that LSI will begin to provide certain new items to run with LDOS/TRSDOS 6.0 on special "Limited Backup Disks". NOTE: This will NOT apply to 5.1 products, or LDOS itself. The new 6.0 products will generally be supplied with three backups allowed from the original master. Since these files cannot be COPYed individually, it will not be possible for a single drive user to move these from the original, non-system master disk to a system disk. Any limited backup disk that fails will be replaced at NO CHARGE upon receipt of the unusable copy by LSI, providing that the allowed number of backups have NOT already been made.

This method may seem very simple, but we feel that it is the easiest way we could give ourselves at least a minimum amount of protection without severely inconveniencing the end user. WARNING: This protection system is NOT as simple as it sounds. It will seem rather easy for a pirate to remove or by-pass this protection, but this is NOT SO. After any tampering with the protection system, the functions of the program WILL BE ALTERED. If anyone offers you a normally protected LSI product for free or almost free, BEWARE - YOU ARE ACQUIRING A TIME BOMB. Unrecoverable damage to your data will usually occur if you use a pirated copy of an LSI protected product.

OK . . . . Now how is the our customer going to know when a product is provided in a protected manner? Very simple. All products that are provided in a protected form will be identified as "protected" in the OFFICIAL LSI CATALOG. They will also have a little difference in their TITLES. All protected LSI products will contain the prefix "LS-" or just "LS". So our "FED-III' package when provided on a protected disk would become "LS-FED II".

DO NOT PURCHASE AN "LS" TYPE PRODUCT IF YOU HAVE ONLY ONE DISK DRIVE!!

We hope this new program will address some of the problems created by the illegal piracy of LSI software. We also hope that our customers can understand the benefit they receive from this new program, and will be willing to cooperate. Lower pricing and easier access to customer service should more than offset any concern about having a limited number of backups.

Someday, strong legislation that will treat software pirates just like "burglars", "muggers", and other criminals who would steal another's property, may stop or slow this cancer in our industry. But until that time, it is up to each to protect his own, like getting a guard dog to protect your property. We sincerely hope that legislators will provide adequate protection for our property in the future, and we will be able to forever get rid of our guard dog.

At this point I would like to restate LSI policy regarding this publication. We do not see ourselves as journalists. The articles in this publication will come from two major types of sources.

First, there is the case where the author is an LSI employee. In this situation, LSI views and policies are expressed in the writing and the material is watched as closely as possible to confirm that it is in keeping with LSI standards and policies.

Now, the other source of material for this publication is "outside authorship". We do not want to become censors or editors that do not allow proper freedoms to these authors. Their material is NOT edited, rewritten, or altered in any way by LSI (other than technical or typographical errors, or the addition of "editor's notes"). It is therefore important for you to note that some articles appearing here may contain material that LSI does not support or agree with. We feel very strongly that we should present the material as it is written, and we are sure you feel the same way. Therefore, we will not assume responsibility for the opinions or beliefs of our outside authors. We will, however, try to keep the technical content of articles correct and accurate regardless of the author.

We are looking to expand our product line here at LSI. To this end, we are constantly seeking program submissions from our users. We are open to almost any type of software in any language. We are looking for quality products from our users to publish and add to our line. The owners of these products will be very fairly compensated for their work. We are open to discussion concerning almost any type of compensation plan that is workable. There can be many thousands of dollars made by authors who publish through LSI. We will even consider accepting NON-EXCLUSIVE rights to a product rather than none at all.

Then, there are those people with great product ideas, but alas, are not programmers Well OK, LSI has an offer for you also. If any outside individual has a great idea for a

product and is willing to work with LSI to have his idea become a reality, LSI will listen. Call me directly. We will discuss your idea (and it will remain YOUR IDEA). If it has promise, I will ask you to write it up in detail. We can then work out a royalty agreement should we implement your concepts and ideas. You don't HAVE to be a programmer to make some money from the software industry.

We are interested in establishing a Southern Wisconsin LSI products USERS GROUP. LSI would assist with getting it started, and probably be willing to provide technical advisors, guest speakers and a meeting facility to this group, as well as our official backing. So, if you are interested, and live within a 100 mile radius (about) of Milwaukee, let me or Chuck know. Let's get one started!!

We have now taken on some of the more popular products from POWERSOFT. This of course includes the ever popular SUPER UTILITY PLUS by Kim Watt. For complete details on these new products in the LSI line, check out our official LSI JULY '83 catalog (FREE ON REQUEST).

Our new catalog also contains several price changes and new product entries, so make sure to get a copy soon and browse.

LSI has been forced to drop from our product line all items provided by SNAPPWARE, Inc. This action is due to the inability of SNAPPWARE to provide LSI with adequate quantities of product to satisfy our customers. The SNAPPWARE products are still considered by LSI to be excellent products and good values. These products are still available directly through SNAPPWARE, but will no longer be available through LSI. Thank you for your understanding in this matter.

We get dozens of calls each day requesting some type of extended support agreement for the LSI 6.0 type system (Model 4 at present). We are working up a method to accomplish this for those who would like it. I will probably announce our finalized plans in the next issue of the Quarterly.

This issue contains a **VERY IMPORTANT PATCH** to the 5.1.3 FORMAT utility. You will find this small but important patch later in this issue. As usual, you may install the patch yourself , or send LSI your disk with the normal upgrade fee, and we will send you back a fresh cut from our master. The new 5.1.3 masters are dated 06/19/83. If it's been awhile since you've had your master in for a "DUSTING and CLEANING", then please send that "critter" in and let us make it just like new.

Remember "smal-LDOS"? Well, it' s now an official member of the LDOS family. The sma1-LDOS product is now available for general purchase through LSI and our dealers, at a suggested retai1 price of JUST $59. That's right, for those who only need the reduced environment that smal-LDOS provides, this is the affordable alternative to the ful1 LDOS system. Smal-LDOS is directly compatible with full LDOS in both Model 1 and Model 3 versions. Smal-LDOS contains very few "bells and whistles", unlike its BIG brother, but is a fully functional operating system for just $59. So, do LSI a favor and let the world know that a version of the most popular TRS-80 DOS is now available for just $59.

Our competition (Newdos, Dosplus, Multidos) offer fine products, but all at close to $100 retail (or more). I feel that the full LDOS at $129 is a super value when you think of what you are getting for your money. However, there are many would-be LDOS users who have little or no need for a majority of the "fancy" features offered by our larger system. The smal-LDOS system is the answer for those users. For $59 they get full LDOS 5.1 compatibility with the reliability and quality of support and design that have made LDOS "THE" DOS alternative. LSI even includes a $30 coupon with every smal-LDOS, which is redeemable at any time toward the purchase of the full LDOS system (this would make the NET cost of smal-LDOS just $29).

In the last issue I announced the expected release of Wordstar on LDOS 5.1 as being in July. As of the day I am writing this article, it appears that two small bugs have been

detected. These have been reported to Micro-Pro, and they promise their prompt attention to the correction of these problems. We are now at their mercy.

LSI wi11 not release a product that contains a known bug. We see no reason that the release of Wordstar will not occur as scheduled, but at this point, that is in the hands of Micro-Pro. So, if there is a slight delay, please be patient. As with any other order you have placed with LSI, any Wordstar order may be canceled, and your money cheerfully refunded at any time prior to shipment of the product. Thank you for your understanding in this matter.

One last thing should be said which is often overlooked by the management of many companies. That is to express my appreciation for the loyal support and backing of our ten s of thousands of customers, without whom LSI is nothing. We like what we are doing, and try to do it in the best way we can. You, our customers, have supported LSI and we will try not to let you down. **THANKS!**

I wish you and your families (and computer) a safe and pleasant summer.

NEW PRODUCT ANNOUNCEMENTS

The following new products are in the July LSI catalog, but were not mentioned in the last LDOS Quarterly.

smal-LDOS 5.1

smal-LDOS 5.1 is a abbreviated version of the LDOS 5.1 operating system, available for the Radio Shack Models 1 and 3. This is a complete operating system, containing all of the most needed functions from the full LDOS 5.1, but at a much lower price. smal-LDOS is ideal for the user who wants the quality of LDOS, but cannot afford the complete system. Please check out the July LSI catalog to see exactly what is and is not included in the new smal-LDOS.

FED II

FED II is an enhanced version of the FED File Editor. This new version includes mapping of machine language files, instruction disassembly, advanced search and reference functions, and much, much, more! Sorry, at this low price there will be no upgrades from the original FED.

The LSI Help System

The LSI Help System is an advanced utility that allows the search and display of the information that **YOU** need in the daily use of your computer system. Help for LDOS and LBASIC is included with the base Help Package. Technical Help is also available, and with the HELPGEN utility you can even produce your own Help files. In many cases, you can access the Help system from within your own programs.

Anthology One and Anthology Disk One

Here it is! Finally, you can get a complete bound set of the LDOS Quarterly, Volume One. Many of these issues have been unavailable for some time, and this is the only place to get them. A veritable wealth of information is presented here, with INDEX! The Anthology Disk contains many of the computer programs from Volume One of the LDOS Quarterly, and that means no typing is required.

MODEM-80

LSI is now carrying the famous Modem-80 communications package. This system allows for automatic file transfer between similarly-equipped machines, and an excellent terminal

uti1ity. Translation tables are used for character re-definition, and best of all, it's LDOS compatible. A special MAX-80 version is also available.

Powersoft Products

LSI is now a distributor for many of your favorite Powersoft products. These include Super Utility Plus version 3.0, and both SU+ reference books; Inside SU+ and the SU+ Tech Manual. We will also be carrying the Toolbox for LDOS, and the Master Mechanic Set for LDOS, both of which are very useful LDOS utility sets.

In terms of new software from other vendors, the Utility Disk from MicroConsultants is a collection of seven utilities and filters designed to enhance your utilization of LDOS. Briefly, here is what each program does:

KSTAT/CMD - When executed at 'LDOS Ready' it gives you a map of the current KSM/FLT or KSMPLUS/FLT key assignments

PSTAT/CMD - When executed at 'LDOS Ready' it displays the current settings of PR/FLT; you may optionally change individual parameters without affecting the others. Use of PSTAT will not affect active filtering of *PR or use additional high memory.

RSTAT/CMD - When executed at 'LDOS Ready' it displays the current settings of the RS232/DVR; you may optionally change individual parameters without affecting the others. Unlike reuse of the RS232/DVR, RSTAT will not affect active filtering of the serial line or use additional high memory.

PRINT/CMD - When executed at 'LDOS Ready' it allows you to send a line of text directly to the printer (for titles, etc.). Use of embedded line feeds and form feeds is also supported.

SETRS232/FLT - This filter, when applied to the serial communications line, allows you to change the basic RS-232 parameters from within an application pro gram (i.e. now you can change baud rate, word 1ength, stop bits or parity while in LCOMM).

EDITCMD/FLT - This filter, when applied to *KI, allows you to edit the last command line entered at 'LDOS Ready'; the edited command line may then be executed. This means that when you type:
FIXTER *PR PR/FLT(FFHARD,L=56,XLATE=X'7FlD',MARGIN=8)
a few keystrokes will change FIXTER to FILTER.

PRLCOMM/FLT - This filter, when applied to *PR, will strip all control codes (except LF, CR, FF) going to the printer while in LCOMM. It will prevent your printer from going crazy when you are dumping text to it straight off the communications line.

All programs will work under Model I, Model III OR MAX-80 LDOS. On the Model I, only the Radio Shack E/I is supported. All programs are provided with documentation and commented assembler source code.

The Utility Disk is available for $25 from MicroConsultants - Central, P.O. Box 55023, Madison, WI 53705

RUNNING 8-INCH DRIVES ON THE MODEL III UNDER LDOS

by Peter S. Simon

How would you like to see 606.OK in response to the FREE command? I'm getting this now on my Model III, and when I upgrade my system to a faster clock rate, I'll be able to access more than 1 Megabyte on each of drives :2 and :3! How do I do it? I'm running 8-inch drives, of course. If you have the right hardware, this article will show you how to tie it into the LDOS operating system for high performance 8-inch operation.

Before discussing the driver routines, let's review the hardware requirements. Naturally, you need to have 8-inch drives. Mine are Tandon TM848-2 thinline drives. These differ from the "classical" 8-inch drive in that there is no head-load solenoid and the disks are not turning constantly. The heads are loaded whenever the drive door is closed. Also, the Tandons have a DC motor which turns on for a few seconds whenever the drive is selected. In effect, the drive acts just like a normal 5-inch drive, except for capacity. I believe that the driver routine should work for the old style drives too, provided that the disk controller motor on logic is used.

The only other requirement is a floppy disk controller capable of 8-inch operation. I have a Micro Mainframe FDC-3C controller installed in my system, which has performed flawlessly since its initial installation. The controller supports both 5-inch and 8-inch drives. The user selects which type by writing to port X'E4' with bit 4 set or reset for 8-inch or 5-inch operation, respectively. I understand that most controllers that support 8-inch drives use the same scheme, so my routines should work for them also.

Program Listing 1 contains Config8, the relocatable 8-inch disk driver routine. By patching the existing system driver routine, calling it, and then restoring it before exiting, I'm able to keep my driver routine quite small. The single density version occupies only 67 bytes of high memory. It consists of 3 parts: the new interrupt handler, the relocator, and the actual relocatable driver. The function of each of these sections of code is described in the remainder of the article.

The default system disk interrupt handler resides at X'46F9'. Its function is to disable further disk interrupts and discard the current return address off the stack. Unfortunately, when it disables disk interrupts by writing X'00' to port X'E4', it also resets the floppy disk controller to the 5-inch mode of operation. Normally this would not be a problem, since an interrupt is not generated until the pending operation is completed. However, the system has several built in retries in case of a disk error, and these retries are guaranteed to fail on an 8-inch disk if the interrupt handler is not modified.

Fortunately, there are a couple of spare bytes between the end of the interrupt handler and the beginning of the Drive Code Tables (DCT's) which allow room for a small patch. I change the interrupt handler to load an immediate value (either X'00' or X'10') in A and then output this to the port. The relocatable driver will modify the code here to supply the correct output for 8-inch or 5-inch drives. Because the relocator code will modify a byte in the interrupt handler, it is imperative that my interrupt handler (not the system's) be resident whenever the 8-inch driver routine is used. You may wish to SYSGEN the 8-inch driver in high memory to make it a permanent part of your configuration upon boot-up. Realize that when you re-boot the system, the original system interrupt handler routine will overwrite my interrupt handler in memory. There are several solutions to this problem. One thing you can do is patch SYS0/SYS (as I have done) so that my routine becomes the system routine. Perhaps a less drastic measure is to create a small program which just writes the new interrupt handler into memory, then returns to LDOS READY. This program can then be AUTO'ed to execute every time the system is booted. I have included such a program, Init8, as Listing 2. Init8 also performs a RESTORE on drives :2 and :3 to assure that the system knows the actual head position (track 0).

The function of the relocator is to place the relocatable code just under the current value of HIGH$, then lower HIGH$ to protect the routine. Of course, it must also modify any references to addresses in the relocatable portion to reflect their new location in high memory. After installing the relocatable code, it performs a RESTORE operation on drives :2 and :3, which has the effect of moving the heads to a known location (track 0) so that subsequent SEEK commands will increment the heads correctly. Finally, we come to the relocatable driver routine itself. Upon entry, it writes X'10' to port X'E4', enabling the 8-inch drives. Next, it modifies code in the existing system disk driver routine which it then calls. Upon return from the system routine, it restores the changes it previously made, then writes X'00' to port X'E4', re-enabling 5-inch operation.

One of the changes is worth expanding on here. If a Record Not Found (RNF) error is generated during disk I/O, a short routine is called by the LDOS driver which toggles the density indicator bit in the DCT (and changes some associated allocation information in the DCT.) Then the operation is tried again. This is how LDOS performs its famous auto-density recognition. Since the new allocation data would be incorrect for 8-inch drives, this function must be changed for 8-inch double density operation. In the routine, I show the necessary changes as a group of commented-out lines which you may insert if you have the necessary clock rate increase needed for double density. Please note that I haven't tested these additions.

I would enjoy hearing from anyone else using 8-inch drives on the Model III, especially double density. Finally, I would like to thank Les Mikesell for patiently answering all my questions about the LDOS drivers over the phone. LDOS truly has fantastic support.

```
00100          TITLE '<Listing 1>'
00110 ;        Config8       4/21/83
00120 ;        LDOS ver. 5.1.3 disk driver for 8" drives.
00130 ;        written by Peter Simon
00140 ;        1950 Hawk Drive
00150 ;        Point Mugu, CA 93042
00160 ;
00170 ;        This is a free, public domain program; however,
00180 ;        I would certainly appreciate an acknowledgement
00190 ;        from persons using this routine. (Please don't
00200 ;        delete my name from the messages.)
00210 ;
00220 ;
00230 ;        Hardware requirements:
00240 ;        TRS-80 Model III microcomputer.
00250 ;        Micro Mainframe FDC-3C floppy disk
00260 ;        controller, or a compatible controller
00270 ;        which uses bit 4 of port E4H to enable
00280 ;        8" operation.
00290 ;        Tandon slim-line 8" disk drives or
00300 ;        compatible drives.
00310 ;        For 8" double density operation, a clock
00320 ;        speed-up modification which at least
00330 ;        doubles the clock speed is required.
00340 ;
00350 ;
00360 ;        Limitations:
00370 ;        The changes required for 8" double
00380 ;        density operation are indicated in the
00390 ;        code below. However, these changes have
00400 ;        not been tested, since I don't have a
00410 ;        clock speed-up kit installed. Please
00420 ;        let me know if you experience any trouble
00430 ;        with double density.
00440 ;
00450 ;
00460 ;        Equates follow:
00470 ;
00480 HIGH$    EQU     4411H           ;protected memory pointer
00490 DSKINT   EQU     46F9H           ;disk interrupt handler
00500 DCT2     EQU     4700H+20        ;DCT for drive :2
00510 DCT3     EQU     4700H+30        ;DCT for drive :3
00520 @DSPLY   EQU     4467H           ;display a message line
00530 @EXIT    EQU     402DH           ;normal return to LDOS
00540 ;
00550 ;
00560 ;        The following section of code replaces the system
```

```
00570 ;       disk interrupt handler routine.  This means that
00580 ;       Config8 should not be SYSGENed unless Init8 is
00590 ;       AUTOed as explained in the text of the article.
00600 ;
00610         ORG     DSKINT          ;address of old routine
00620         XOR     A               ;clear flags
00630 REPL1   EQU     $+l             ;label will be used
00640                                 ; for code modification
00650         LD      A,0             ;immediate value to be
00660                                 ; changed to 10H for 8
00670                                 ; inch operation
00680         OUT        (OE4H),A        ;turn off disk interrupts
00690         POP     HL              ;discard I/O loop return
00700         RET                     ;done
00710 ;
00720 ;
00730 ;
00740 ;       The actual 8 inch driver routine follows:
00750 ;
00760 ;       Entry requirements:
00770 ;               IY ==> points to DCT for desired drive
00780 ;               HL ==> points to the memory buffer
00790 ;               DE ==> contains cyl & sector #
00800 ;               B  ==> contains function # as defined
00810 ;                      on page 6-15 of LDOS manual
00820 ;               C  ==> contains drive physical address
00830 ;               A  ==> anything
00840 ;
00850 ;       Exit conditions:
00860 ;               Z  <== flag set if no error
00870 ;               A  <== contains LDOS error code if Z not
00880 ;                      set
00890 ;                  <== Other registers may be altered
00900 ;
00910 ;
00920         ORG     6500H           ;this code is relocated
00930                                 ; to high memory.
00940 ;
00950                                 ;standard LDOS header:
00960 START   JR      START2          ;jump around header
00970 OLDMEM  DEFW    $-$             ;previous HIGH$ address
00980         DEFB    7               ;# characters in title
00990         DEFM    'Config8'       ;title of program
01000 START2  LD      A,19H           ;enable 8" drives
01010         OUT     (OE4H),A
01020         LD      (REPL1),A       ;change interrupt handler
01030                                 ; to enable 8" drives
01040         LD      A,0D0H          ;change write routine to
0l050         LD      (46A5H),A       ; enable 8" drives
01060         PUSH    HL                ;save HL
0I070         LD      HL,46FFH        ;point to RET instruction
01080 OFFSET1 EQU     $-START-2       ;symbol for assembly
01090         LD      (4682H),HL      ;disable density toggling
0ll00         POP     HL              ;restore HL
01110         LD      A,05H
01120         LD      (46A9H),A       ;shorten write routine's
01130                                 ; pause loop counter
01140         CALL    4585H           ;call system driver
01150         PUSH    AF              ;save status & error code
01160         XOR     A               ;enable 5 inch drives
01170         OUT     (OE4H),A
```

```
01180         LD      (REPLI),A       ;restore int'pt handler
01100         LD      A,0C0H          ;restore write routine
01200         LD      (46A5H),A
01210         LD      A,50H           ;restore write delay
01220         LD      (46A9H),A
01230         POP     AF              ;restore error status
01240         PUSH    HL              ;save HL
01250         LD      HL,456BH        ;re-enable density
01260         LD      (4682H),HL      ; toggling for 5"
01270         POP     HL
01280         RET                     ;done!
01200 ;
01300 ;*****************************************************
01310 ;
01320 ;       The following lines are commented out for 8"
01330 ;       single density operation. Add these lines to
01340 ;       the program if you have the required clock speed-
01350 ;       up installed and you wish to enable automatic
01360 ;       8" density recognition.
01370 ;
01380 ;OFFSET2        EQU     $-START ;symbol for assembly
01300 ;TOGGLE8        EQU     $       ;Toggle 8" density specs
01400 ;       LD      A,(IY+3)        ;get flag from DCT
01410 ;       XOR     40H             ;toggle density bit
01420 ;       LD      (IY+3),A        ;replace it in DCT
01430 ;       LD      BC,270FH        ;single density info.
01440 ;       BIT     6,A             ;check new density
01450 ;       JR      Z,FIXDCT        ;go if single
01460 ;       LD      BC,491DH        ;double density info.
01470 ;FIXDCT LD      (IY+7),C        ;update DCT for new
01480 ;       LD      (IY+8),B        ; dens. allocation info.
01490 ;       JP      45A0H           ;go RESET the drive
01500 ;
01510 ;*****************************************************
01520 ;
01530 END     EQU     $-1             ;symbol for assembly
01540 LENGTH  EQU     $-START         ;symbol for assembly
01550 ;
01560 ;
01570 ;       The following section of   code relocates the 8"
01580 ;       driver to high memory and initializes drives :2
01590 ;       and :3 for 8" operation:
01600 ;
01610         ORG     6000H    ;start of relocator program
01620 ;
01630 ENTRY   LD      HL,(HIGH$)      ;get old HIGH$ value
01640         LD      (OLDMEM),HL     ;put old HIGH$ in header
01650         EX      DE,HL           ;destination for LDDR
01660         LD      HL,END          ;source pointer for LDDR
01670         LD      BC,LENGTH       ;byte counter
01680         LDDR                    ;relocate driver
01690         LD      (HIGH$),DE      ;lower HIGH$ to protect
01700         INC     DE              ;point to routine start
01710         LD      (DCT2+1),DE     ;change disk driver
01720                                 ; pointer for drive :2
01730         LD      (DCT3+1),DE     ;same for drive :3
01740 ;
01750 ;*****************************************************
01760 ;
01770 ;       The following lines are commented out for single
01780 ;       density operation.    As above, these lines should
```

```
;       be added if you have the requisite hardware and
;       desire to enable auto 8" density recognition.
;
;       LD      IY,OFFSET1      ;offset for code change
;       ADD     IY,DE           ;IY = address to change
;       LD      HL,OFFSET2      ;offset for TOGGLE8
;       ADD     HL,DE           ;HL = TOGGLE8 new addr.
;       LD      (IY+0),L        ;move low order byte
;       LD      (IY+1),H        ;move high order byte
;
;*****************************************************
;
        LD      B,04H           ;RESTORE function
        LD      DE,0000H        ;track 0, sector 0
        LD      IY,DCT2         ;drive :2 DCT
        LD      A,(IY+4)        ;get drive address
        AND     07H             ;only drive address
        LD      C,A             ;place in C
        CALL    DCT2            ;RESTORE drive :2
        LD      B,04H           ;RESTORE functio-n
        LD      DE,0000H        ;track 0, sector 0
        LD      IY,DCT3         ;drive :3 DCT
        LD      A,(IY+4)        ;get drive address
        AND     07H             ;only drive address
        LD      C,A             ;place in C
        CALL    DCT3            ;RESTORE drive :3
        LD      HL,MESS         ;point to message
        CALL    @DSPLY          ;display it
        jp      @EXIT           ;back to LDOS
MESS    DEFB    0AH
        DEFM    'Config8 ver 1.0 by Peter Simon'
        DEFB    0AH
        DEFM    'Drives 2 and 3 configured for
        DEFM    'Tandon slim-line 8 inch drives'
        DEFB    0AH
        DEFM    'Driver relocated to high memory'
        DEFB    0AH
        DEFB    0DH
;
;
;       The following fixes drives :2 and :3 Drive Code
;       Table (DCT) for 8" operation.
;
        ORG     DCT2            ;drive :2 OCT
        jp      $-$             ;address supplied by
                                ; relocator.
        DEFB    00100110B       ;indicate by bits:
                        ;7) not software protected
                        ;6) single density
                        ;5) 8 inch drive
                        ;4) side 0 currently selected
                        ;3) floppy drive
                        ;2) 1/2 second delay for SELECT
                        ;1) & 0) 10 ms STEP delay
        DEFB    00100100B       ;indicate by bits:
                        ;7) unused
                        ;6) single density only!
                        ;*** change 6) for double dens.
                        ;5) 2 sided drive
                        ;4) non-alien controller
                        ;3)-0) drive address 4
```

```
02400         DEFB    0               ;current cyl
02410         DEFB    76              ;77 tracks on the drive
02420         DEFB    15              ;16 sectors/track
02430         DEFB    00100111B       ;indicate by bits:
02440                         ;7)-5) 2 grans/track
02450                         ;4)-0) 8 sectors/gran
02460         DEFB    38              ;dir. on cyl #38
02470         ORG     DCT3            ;fix drive :3 same way
02480         JP      $-$
02490         DEFB    00100110B
02590         DEFB    00101000B       ;different drive address
02510                                 ;*** change bit 6) for
02520                                 ;    double density
02530         DEFB    0
02540         DEFB    76
02550         DEFB    15
02560         DEFB    00100111B
02570         DEFB    38
02580
02590         END     ENTRY           ;start running at ENTRY

00100         TITLE   '<Listing 2>'
00110         Init8           4/21/83
00120         A program to restore 8" drives :2 and :3 to
00130         track 0, sector 0. This seems to be necessary
00140         for the system to properly access the 8" drives
00150         after a boot.   Written by Peter Simon
00160                                    1950 Hawk Drive
00170                                    Point Mugu, CA 93042
00180
00190
00200         Equates follow:
00210
00220 @EXIT   EQU     402DH           ;normal return to LDOS
00230 DCT2    EQU     4700H+20        ;drive :2 DCT
00240 DCT3    EQU     4700H+30        ;drive :3 DCT
00250 DSKINT  EQU     46F9H           ;disk interrupt handler
00260 ;
00270 ;
00280 ;
00290 ;
00300         ORG     6000H           ;start of program
00310 ENTRY   LD      B,04H           ;RESTORE function
00320         LD      DE,0000H        ;track 0, sector 0
00330         LD      IY,DCT2         ;drive :2 DCT
00340         LD      A,(IY+4)        ;get drive address
00350         AND     07H             ;only drive address
00360         LD      C,A             ;place in C
00370         CALL    DCT2            ;RESTORE drive :2
00380         LD      B,04H           ;RESTORE function
00390         LD      DE,0000H        ;track 0, sector 0
00400         LD      IY,DCT3         ;drive :3 DCT
00410         LD      A,(IY+4)        ;get drive address
00420         AND     07H             ;only drive address
00430         LD      C,A             ;place in C
00440         CALL    DCT3            ;RESTORE drive :3
00450         JP      @EXIT           ;back to LDOS
00460 ;
00470 ;
00480 ;       The following  section of  code replaces the system
00490 ;       disk I/O interrupt handler routine. It has been
```

```
00500 ;       located in this program with the expectation that
00510 ;       Init8 will be AUTOed to load and execute every
00520 ;       time that the system is booted. It will not
00530 ;       affect normal 5" operation, but will prevent
00540 ;       any weirdness from occurring if Config8 is
00550 ;       SYSGENed into the system. See text of article
00560 ;       for further explanation.
00570 ;
00580         ORG     DSKINT          ;replace system routine
00590         XOR     A               ;leave alone
00600         LD      A,O             ;"0" changed by Config8
00610         OUT     (0E4H),A        ;turn off disk interrupts
00620         POP     HL              ;discard I/O loop return
00630         RET
00640 ;
00650         END     ENTRY           ;begin execution at ENTRY
```

ACTIVE VARIABLE DUMP for LBASIC

By Alan R. Moyer, 993 San Angelo Dr., Hamilton, Ohio 45013

Quite often in developing and/or debugging BASIC programs over 50 lines long, it is useful, if not downright essential, to determine the contents of any or all of the variables used in a program. A well used technique is to stop program execution and enter "PRINT xx" from the keyboard or sprinkle similar lines throughout the program itself. I desired a better tool than that. LBASIC/OV4 is what I developed to meet that need for me. LBASIC/OV4 is an active variable dump that is patched into LBASIC and will display the contents of simple variables, array variables (up to 5 subscripts), and user defined functions. Integers, single and double precision, and string variables are supported. Output can be sent to either video display (*DO) or printer (*PR). As with standard LBASIC, shift-@ will pause the display and BREAK will abort back to LBASIC.

LBASIC/OV4 is called as an LBASIC overlay with the "CMD xx" convention. The syntax is:

CMD "V devspec parms"

The available parameters are:

-S Simple variables only
-A Array variables only
=x Only variables whose names begin with "x"

Devices *DO (video) and *PR (printer) are the only legal output devices, *DO is the default device. These devices can be routed accordingly for specific user needs.

The reason for the limited output devices is the trick used to display the variables. Not wanting to reproduce ROM code that converts variable data into ASCII, I decided to patch into the ROM "PRINT" and "LPRINT" code to shorten my program. Very simply, this program looks through the variables table of a program that is in memory, building a buffer with each variable's ASCII name and "type" along with an equal sign (=), all inside quote characters. The variable name is then duplicated after the last quote and a zero is added to terminate the buffer. At this point the buffer would look something like this:

"AX$ = "AX$

The HL register is pointed to the beginning of the buffer and the ROM "PRINT" or "LPRINT" code is called. User defined functions and arrays are handled in overall similar, but obviously more complex, fashion.

This version was specifically written for the Model I and the LBASOV4/FIX patch file is for Model I LBASIC 5.1.3. Since I don't have access to a Model III, I have no information about compatibility with Model III systems.

If you don't want to modify your version of LBASIC for some reason, you can remove LBASIC/OVl or LBASIC/OV2 from your disk and rename LBASIC/OV4 to whichever you removed. For instance, replacing LBASIC/OVl (normally the renumbering overlay) will allow variable dumping with the CMD "N" command. Just use the new syntax for the "V" command with the "N" command, for example, CMD "N *PR –V".

Comments and suggestions about this LBASIC utility can be routed to me via the LDOS SIG on Micronet.

```
.LBASIC patch to allow use of LBASIC/OV4.BASIC
.Patch code to allow parameter passing
X'5464'=C3 A9 64
X'64A9'=FE 58 CA 68 54 04 04 FE 56 CA 68 54 C3 D7 54
.Patch code for single letter access
X'5803'=C3 B8 64
X'64B8'=04 04 FE 56 CA 68 54 C3 4A lE 00
.Alter end of LBASIC code for buffer info
X'53CE'=C2 64
```

Use this code in conjunction with BINHEX to produce the LBASIC/OV4 file.

```
05 06 4C 42 41 53 49 43 1F 23 43 6F 70 79 72 69 67 68 74 20 28 63 29 20 31 39 38 32 20
62 79 20 41 6C 61 6E 20 52 2E 20 4D 6F 79 65 72 01 02 00 52 CD 52 54 21 E3 54 CD 41 54
3E 01 B7 CA 12 52 CD 3B 52 3E 01 B7 CA 1B 52 CD B3 52 21 30 52 11 80 44 D5 01 0B 00 ED
B0 D1 21 18 43 36 2A C3 33 44 4C 42 41 53 49 43 2F 43 4D 44 03 21 21 55 CD 41 54 DD 2A
F9 40 ED 5B FB 40 CD 27 54 C8 3A E3 55 B7 CA 5C 52 47 DD 7E 02 B8 C2 A0 52 CD B8 53 DD
7E 02 FE 81 FA 8F 52 E5 DD 5E 03 DD 56 04 21 EE 56 E5 06 3A CD EC 53 36 00 E1 C1 CD 85
2B C5 E1 11 64 55 CD EA 53 21 EB 55 CD 41 54 C3 A0 52 11 5F 55 CD EA 53 06 3D 11 EE 55
CD EC 53 CD 0D 54 DD 7E 00 C6 03 4F 06 00 DD 09 CD 2E 54 CA 1B 52 C3 45 52 21 34 55 CD
41 54 DD 2A FB 40 ED 5B FD 40 CD 27 54 C8 3A E3 55 B7 CA D4 52 47 DD 7E 02 B8 C2 A2 53
CD B8 53 DD 7E 05 32 D7 55 FE 06 DD E5 DA F0 52 11 46 55 CD EA 53 CD 0D 54 C3 A0 53 87
47 DD 23 10 FC FD 21 D9 55 06 0A FD 36 00 00 01 02 00 53 FD 23 10 F8 36 28 23 FD 21 D9
55 3E 01 32 D6 55 E5 3A D6 55 B7 CA A0 53 DD 6E 04 DD 66 05 FD 5E 00 FD 56 01 B7 ED 52
CA 84 53 D5 E1 13 FD 73 00 FD 72 01 CD F7 53 06 04 1A FE 30 C2 41 53 13 10 F7 E1 E5 CD
EA 53 3A D6 55 4F 3A D7 55 91 CA 6C 53 36 2C 23 DD 2B DD 2B FD 23 FD 23 3A D6 55 3C 32
D6 55 CD 2E 54 CA 1B 52 C3 10 53 36 29 23 11 5F 55 CD EA 53 06 3D 11 EE 55 CD EC 53 CD
0D 54 E1 C3 10 53 FD 36 00 00 FD 36 01 00 FD 2B FD 2B DD 23 DD 23 3A D6 55 3D 32 D6 55
E1 E1 C3 10 53 DD E1 DD 5E 03 DD 56 04 01 05 00 DD 09 DD 19 CD 2E 54 CA 1B 52 C3 BD 52
21 EE 55 DD 7E 02 FE 80 FA C5 53 D6 80 77 23 DD 7E 01 B7 CA D0 53 77 23 DD 7E 00 FE 08
C2 DC 53 0F C3 E3 53 FE 04 C2 E3 53 3C 3C 47 3E 27 90 77 23 C9 06 00 1A B8 C8 B7 C8 77
13 23 C3 EC 53 11 E8 55 0E 05 3E 0A CD C4 01 02 00 54 44 C6 30 12 1B 0D C2 FE 53 11 E4
55 C9 36 00 FD E5 DD E5 21 EA 55 3A D8 55 B7 F5 C4 9B 20 F1 CC 67 20 DD E1 FD E1 C9 D0
E5 E1 B7 ED 52 C9 DD E5 FD E5 CD 2B 00 FE 60 CC 49 00 FE 01 FD E1 DD E1 C9 3A D8 55 B7
DD E5 F5 C4 67 44 F1 CC 6A 44 DD E1 C9 CD C4 54 FE 0D C8 FE 2A C2 82 54 CD CC 54 FE 44
C2 70 54 CD CC 54 FE 4F C2 DA 54 C3 81 54 FE 50 C2 DA 54 CD CC 54 FE 52 C2 DA 54 AF 32
D8 55 23 CD C4 54 7E FE 0D C8 FE 2D CA 96 54 FE 3D CA B1 54 C3 D1 54 CD CC 54 FE 53 C2
A5 54 AF 32 13 52 C3 81 54 FE 41 C2 D1 54 AF 32 0A 52 C3 81 54 CD CC 54 FE 41 FA D1 54
FE 5A F2 D1 54 32 E3 55 C3 81 54 7E FE 20 C0 23 C3 C4 54 23 7E CB AF C9 21 75 55 CD 67
44 C3 1B 52 21 9B 55 CD 67 44 C3 1B 52 2A 2A 2A 2A 2A 20 41 63 74 69 76 65 20 56 61 72
69 61 62 6C 65 20 44 75 6D 70 20 2A 2A 01 DB 00 55 2A 2A 2A 0A 20 20 20 20 56 65 72 73
69 6F 6E 29 34 2E 30 20 28 4C 44 4F 53 29 20 2D 20 41 72 6D 0D 0A 53 69 6D 70 6C 65 20
56 61 72 69 61 62 6C 65 73 3A 0D 0A 41 72 72 61 79 20 56 61 72 69 61 62 6C 65 73 3A 0D
2A 20 54 6F 6F 20 6D 61 6E 79 20 73 75 62 73 63 72 69 70 74 73 20 2A 29 00 20 3D 20 22
00 20 3C 55 73 65 72 20 44 65 66 20 46 75 6E 63 3E 0D 41 63 74 69 76 65 20 56 61 72 69
61 62 6C 65 20 44 75 6D 70 3A 20 50 61 72 61 6D 65 74 65 72 20 65 72 72 6F 72 0D 41 63
74 69 76 65 20 56 61 72 69 61 62 6C 65 20 44 75 6D 70 3A 20 2A 44 4F 20 6F 72 20 2A 50
52 20 6F 6E 6C 79 20 76 61 6C 69 64 20 6F 75 74 70 75 74 20 64 65 76 69 63 65 73 0D 00
00 01 01 0D E3 55 00 20 20 20 20 20 00 22 20 20 20 02 02 00 52
```

MY BASIC ANSWER

by E.R. Cheatham, 5127 Blueberry Hill, Houston, Texas 77084

Intrigued by ads, I bought The Basic Answer (TBA) software package. However, I have to admit wondering if TBA would work satisfactorily with LDOS and Newscript in a Radio Shack Mod I computer. My fear was soon put to rest. These three programs make a dynamite package!

BTBA (Before The Basic Answer), my BASIC programs were written with the standard BASIC editor. Since I write programs by "semi-flowcharting" them in my mind (due to laziness), there are many times when I need to insert more program lines than there is space available. If this has ever happened to you, I'm sure you know that your choices are simple. Either use a renumber command to give you more space, or overtype existing line numbers- - - which means retyping the original commands using new line numbers. Either way, it is time consuming and a real hassle.

Other times, I would have liked to repeat a line or lines in another area of the program. The old way was to type all of these lines again. In a similar manner, there were times when I wanted to move a block of commands to another area and erase them at the old location. Again, it meant retyping and then deleting the old lines. Time consuming!!

Almost any block move of program lines will involve correcting any line number references used in gosub, goto, on x goto commands to the new line numbers. Usually, one or two references are overlooked which gives you even more program bugs to play with.

There is a new and superior method for writing a basic program on the Radio Shack computer---LDOS, a good word processor, and The Basic Answer.

TBA offers these major advantages:
1. There is no need for line numbers.
2. Variable names can be long (14 characters).
3. The text is self-documentary.

NO LINE NUMBERS REQUIRED

The ROM-based basic editor requires a program with line numbers to be able to save and load the program to/from a diskette, and to run the program. Now, don't worry about line numbers. Use the powerful editing capabilities of a good word processor. Let your ideas flow.

Of course, it is evident that the word processor can find and change a word, a letter, or a phase easily. Or it can globally find and exchange letters and words. But more importantly, consider how easy it is to insert one or more lines of text. Using an editor and incrementing line numbers by ten, there is room to only insert nine lines of new text before something must be done to provide more space. Now, you can insert as many lines as you want to add.

Block moves of text are also important. Newly added lines can be exact copies of original lines--made in a second. You can move data around freely--- place a copy of data where needed, leave original data where it is, or delete it automatically. This text will be compiled by TBA later. Then, and only then, will line numbers be important.

TBA makes all of this possible with the use of "labels". A label allows cross-referencing a line within other lines. For instance, a reference line can be given as "gosub @yes.no.answer". (The "@" symbol denotes a label or a variable name.) Then, at the beginning line of the subroutine called @yes.no.answer, place a label "@yes.no.answer:..... etc.". The label allows referencing to a specific text line

without the use of numbers. The actual subroutine can be placed anywhere in the program and TBA will find it. Each 1abel is unique within that program. The 1abel can be referenced many times within the program. Text lines without numbers can be moved throughout the program without any concern for changing reference numbers.

LONG VARIABLE NAMES

The ROM editor will allow only one or two character significant variable names. TBA permits up to 14 characters for variable names such as @ROLL.DICE, @CREDIT.MEMO, @EMPLOYEE.NAME. These names actually mean something and are easier to remember while programming than names like Zl, Z2, Z3(x). TBA can even tell the difference between two variable names such as @EMPLOYEE-NAME and @Employee.Name.

SELF-DOCUMENTING

The completed text is easy to read in words that are understandable. This may not seem important to you until you have returned to work on a program after leaving it for several months. Without TBA text, the program can look as familiar as Martian, and even less understandable.

HOW TO USE TBA

As discussed earlier, use a good word processor to produce the program text. Then save the program to disk. The processor saves the program in ASCII format (as a sequential output file). Use the file extension name "/TBA" such as "payables/TBA". From DOS Ready mode, call TBA. TBA will ask a few questions concerning output and then begin the compiling. TBA makes five passes through the text to check that all variables and labels are defined once only; and then it assigns line numbers. TBA reports any error and its line number in the original text. TBA does not proofread for incorrect BASIC commands such as goti or gosib--- that is the job of the ROM editor.

The fina1 product of TBA is a disk file such as PAYABLES/TXT which is saved in ASCII format. I prefer to go into BASIC mode, load in PAYABLES/TXT, then resave it as PAYABLES/BAS. The only reason for this is that PAYABLES/BAS is saved in standard, compressed format which loads much faster.

TBA GOODIES

Upon initializing, TBA asks several questions and gives several options. You choose the display, either CRT screen or lineprinter. One option will print one text line followed by its equivalent, translated command line. I personally do not like this option because a long program will produce reams of paper printout.

In my opinion, the best option is no program listing, but a very comprehensive cross-reference listing which shows every label and variable used, and the program lines in which they occur. It even shows the long, text name for a variable and its translated, two character name. This is valuable for debugging the program.

Here is a warning! Never have the LDOS PR/FLT print filter active when using TBA! TBA has its own print filter. The two filters will fight and the printout is a disaster - - - 2-3 lines, skip 1 1/2 pages, print 1 1/2 pages, etc. This is not mentioned in an otherwise well-written TBA manual.

WRITING LARGE PROGRAMS

TBA is disk-based so the program length is limited only by the capacity of a disk to contain all of the text program. The real problem is with Newscript. Newscript capably handles a file by using subfiles of approximately 350 lines each. At the end of each subfile, Newscript places an append command which is interpreted during printout to fetch the next subfile. The point is that Newscript must have the file in memory so that it can be edited. TBA does not recognize the append commands. Therefore, TBA cannot use these subfiles.

My game program turned out to be some 990 lines long. What do you do when your program is too large for Newscript? One answer is to break it into several subfiles. Mine had three subfiles. The subfiles could not be merged since the ROM editor cannot load these ASCII subfiles directly as they have no line numbers. I wrote the following simple program to 1ine input one line at a time. add a 1ine number, and save it back out to disk. This was done for each subfi1e. The beginning number for the next subfile was greater than the last number of the previous subfile. Finally, all of these files are numbered. Then load subfile 1 into memory, merge subfile 2, and then merge subfile 3. The composite program is saved as "GAME/TBA" in ASCII option A command. TBA will operate on a program which does have line numbers! It simply ignores the line numbers as it compiles. The resultant program is a "standard" basic program.

```
10 REM A short program to add line numbers to word processor output
20 REM text so that the text can be used by the ROM Editor
30 REM by ER Cheatham   04/15/83
40 REM Program is used when text is larger than the processor
50 REM can hold in memory at one time.  Write the last line
60 REM number of present subfile.  Beginning line number of
70 REM next subfile is last number plus 30-40.
80 CLEAR 3000: CLS
90 INPUT"What is name of file to be numbered ";FS$
100 INPUT"What drive is it on ";Dl$: PRINT: PRINT
110 INPUT"What is new file name ";FD$
120 INPUT"What drive will it be on ";D2$: PRINT: PRINT
130 INPUT"What is the beginning line number ";LN%: IF LN% < 10 THEN LN%= 10
140 CLS: PRINT@ 448,"Present line # is==>";
150 OPEN "I",1, FS$ + ":" + D1$: OPEN "O",2, FD$ + ":" + D2$
160 LINEINPUT #1, A$: N$= STR$(LN%): L=LEN(N$)
170 B$= RIGHT$(N$, L-1) + " " + A$
180 PRINT@ 468, LN%: REM could put print or lprint B$ here
190 PRINT #2, B$
200 LN%= LN% + 10
210 GOTO 160: REM will run until end of file error and stop
```

Just a note--- as you run the program to debug it, there are apt to be a few bugs around. Obviously, any errors should be corrected in the original text so that it will always be the current version.

TBA is a tool which allows the power of a word processor to be used in writing basic programs. It gives you an extensive variable cross-references listing. TBA is a very good program. I am pleased that I have it!

An ASCII File Listing Utility for The BASIC Answer

By Jeffrey Brenton, P 0 Box 146, Woodstock Illinois 60098

Several years ago, I read a couple of articles about programs to 'beautify' BASIC program listings. This program is the result of my labors to do it my way. As the old saying goes, 'any given program will expand to exceed the limits of available memory'. What started life as a ten-line LBASIC program for ASCII-saved BASIC files eventually grew into the 'monster' you see now. It still runs in less than 32K, including string space, but for how much longer?

When The BASIC Answer (or TBA) arrived, it became obvious that lower-case recognition would have to be added to LISTER (the old name) to allow it to work on all TBA - acceptable files, so why not re-write it in TBA? Within two hours, I had a workable version up and running. Testing was done on its own source file, written mostly in lower case. Unfortunately, it was SLOW! No real problem, just run it thru BASCOM. This is when I found out that all those patches to get BASCOM to run on the model III (October 1981 Quarterly) apply equally to 5.1 on the Model I. After getting that fixed, everything

was great. LISTFILE/CHN (new name) was faster than my MX-100 at formatting lines, but still lacked any error handling. This has been added for this article, along with default extensions to reduce typing.

LISTFILE is written in TBA format. There are several reasons for this. It allowed me to be very generous with the comments without slowing the program down any more than the interpreter would, as only remarks proceeded with 'REM' appear in the final program. It also gives those of you who do not yet own TBA a chance to see what a TBA source file looks like. An explanation of some of the features is in order.

TBA is a text-processing utility that allows you to write structured programs in TRS-80 BASIC (or MBASIC v4). You may use variables with up to 14 significant characters, as we11 as meaningful1 labels for procedures instead of 1ine numbers. 'GOSUB @CHECK.RANGE' is much more meaningful to someone reading your program than 'GOSUB 920', and merging library subroutines into a program is much easier when you don't have to worry about conflicting line numbers.

Variables may be 'global', available throughout the program, or 'local', valid only within their own subroutine. But with this flexibility comes some restrictions. All variables must be declared somewhere in the program or they cannot be properly translated into a form BASIC can handle.

An example of global definitions is found in lines 4, 5 and 6 of LISTFILE. The '=' sign in the first column of each line tells TBA that these are global variables. Each variable starts with a letter, and ends with a type-declaration character. In between, you can use any combination of between 1 and 13 additional letters and/or numbers, plus two special characters, the period and underscore. Spaces are NOT allowed, and reserved words that end in a type-declaration character, such as 'TIME$' or 'STRING$' cannot be used. Reserved words can, however, be WITHIN a variable name, so the name DAY.TIME$ is valid.

Local variables follow the same rules as globals, but are declared differently. When you start a subroutine, you assign it a label beginning with '@'. A local is declared by putting it after the label to which it applies, as in '@CHECK.RANGE = HIGH.LIMIT%,LOW.LIMIT%'. In this example, the variables 'HIGH.LIMIT%' and 'LOW.LIMIT%' are valid only in this subroutine, from the label line up to the 'RETURN' instruction. Either or both may share their name with a global and/or one or more other locals, but each will be translated as a unique BASIC variable. A note of caution, however. If you plan on taking advantage of the 'RUN program, V' option to chain programs together, avoid local variables. TBA translates all local variables into two letter BASIC variables before doing the globals, so even if all your global declarations are matched one-for-one in both source programs, if one has more locals than the other there can be conflicting translations. TBA translates variables by first trying to truncate them to their first two characters. If this combination has already been used, it increments the second character by one and tests it again. Thus, if you use TEST% as both a local and a global, the local version would become TE%, since locals are translated before globals and the global would become TF%. If your second program uses TEST% as a global and has no local variables that begin with 'TE', TEST% would become TE%, and would get the value of the local TEST% passed to it instead of the global value. Take it from someone who spent three days learning this tid-bit, it is a hard-to-find bug.

Another feature of TBA is conditional compilation, an example of which is found in lines 7 through 10. Here we decide whether or not to include the 'clear' statement in the BASIC program. If this program is to be used with the interpreter, we must give it string space to work with. But BASCOM will commit suicide if you include a 'CLEAR' statement. By using *IF directive / *END, the intervening lines are only included if you specify 'directive' when you run TBA. This allows you to write one source program to cover any number of machines, and by including the machine-dependent items within *IF's, non machine-dependent changes need only be made once to cover all machines.

The *TITLE command lets you add a 14 character title that will appear on each page of the TBA listings, while *PRLINES and *PAGE control the format of the listing. By using *LIST ON and *LIST OFF, you can have some or all lines skipped in the TBA listing.

All lines are given line numbers according to their position in the source file, i.e. the 30th line in the source becomes line 30 in the BASIC file. If you specify full compression during compilation, all unnecessary spaces will be removed from the final program. It will not, however, combine statements from different lines to 'optimize' the program.

After processing your source file, TBA gives you a cross-reference listing of all labels and variables, showing what they were translated to and all lines referencing them. This is VERY helpful in tracking down errant variable, and is much like CMD"X" in LBASIC, except for being more readable.

Getting back to LISTFILE itself, the program was only designed to work on pure ASCII files. This means that if you want a neat listing of you favorite BASIC program, you must first SAVE it with the ',A' option. It checks each line one character at a time looking for the 'trigger characters' : and E. When it finds one that is not within double quotes, it takes appropriate action, either immediately starting a new line (':') or checking the next few characters for a match ('E').

The line numbering option is included so that if TBA has problems understanding your source code, you can find the offending line quickly. The same effect can be obtained with LIST filename (P,N) from LDOS Ready, but without the nicety of one statement per line.

Have fun with The BASIC Answer and, hopefully, with LISTFILE. You'll find it much easier to read those tightly-packed lines of BASIC code now. That can save almost as much time as it takes to run the program (maybe more)! One warning though. LISTFILE will break your line up incorrectly if you use 'ELSE' inside a label or variable name. It can't be helped unless I get really ambitious and have it analyze the line in both directions and try to find 'IF' statements to match up, or test against the variable declarations. Maybe if I use all 48K and give it all night to run.....

Any questions can be sent to me at the above address or via the LDOS or QSD bulletin boards on CompuServe to <73l05,532>

Sample output:

If your program contains a line such as:

```
@test.line: if true% then print output$: true% not(true%) else print "False": goto
@try.again
```

it would become:

```
12     @test.line
            :   if true% then print output$
            :   true = not(true%)
       else print "False"
       goto @try.again
```

TBA Source code

```
*TITLE"List utility"
*PRLINES=56
rem program to list ascii files from disk to printer with breaks at ":" and "ELSE" and
optional line numbers.  Version 2.1 by Jeffrey Brenton, Tandern-Flow Systems Ltd
=line.from.disk$,test.character$,counter%,line.counter%
=file.name$,drive.number$,print.numbers$,quoted.string%
=file.to.open$,uppercase$,yes.else%
```

```
'let's separate these declarations from the rest of the program in the listing
*PAGE
*IF INT
'let's get ourselves some working room, but only if running under the INTerpreter
clear 15000
*END
'define a generic upper-case converter function
def fn uppercase$(line.from.disk$, counter%) = chr$(asc(mid$(line.from.disk$, counter%,
1)) and &H5F)
'program really starts here
@begin
line.counter%=l
cls
@file.name
input "Please enter file name to list";file.name$
if len(fi1e.name$) > 12 or len(file.name$) < 1 then print "Invalid file name.": goto
@file.name
@drive.number
input "Please enter drive number";drive.number$
'check range of input drive number - change to suit your system
if len(drive.number$) > 1 or drive.number$ < "O" or drive.number$ > "3" then print
"Invalid drive number": goto @drive.number
on error goto @file.not.found
@open.the.file
file.to.open$=file.name$ + ":" + drive.number$
open"I",1,file.to.open$
on error goto 0
@line.numbers
print.numbers$="Y"
input"Do you wish to have line numbers printed (<ENTER> = yes)";print.numbers$
print.numbers$=fn uppercase$(print.numbers$, 1)
if instr("YN",print.numbers$) 0 then print "Invalid response.  Try again": goto
@line.numbers
lprint file.name$;tab(50)time$
'it is now time to start reading the file
@read.record
lend of file yet?
if eof(l) then @done.with.file
'no, so lets get another line
lineinput#l, line.from.disk$
if print.numbers$ = "Y" then lprint line.counter%; tab(7);
'check to see if any of our 'trigger' characters are in line - if not, print line
immediately and skip remainder of tests for time's sake
if instr(line.from.disk$,":") + instr(line.from.disk$,"E") + instr(line.from.disk$,"e") =
0 then lprint line.from.disk$: line.counter% = line.counter% + 1: goto @read.record
'now we have to check each character individually to find 'triggers'
for counter% = 1 to len(line.from.disk$)
test.character$ = mid$(line.from.disk$, counter%, 1)
'if test character is a double quote, toggle state of quoted.string%
if test.character$ = chr$(34) then quoted.string% = not (quoted.string%): lprint
test.character$;: goto @done.testing
'if quoted.string% is TRUE, then we can ignore this character - just print it and go on
if quoted.string% then lprint test.character$;: goto @done.testing
'for colons, we must terminate the current line and then indent the next, either 5 spaces
without line numbers, or 12 with
if test.character$ = ":" then lprint: if print.numbers$ = "Y" then lprint
tab(12)":";:goto @done.testing else ]print tab(5)":";: goto @done.testing
'if this is an "E", then it might be an "ELSE", so go to the subroutine to test the next
3 characters - if it is an "ELSE", terminate current line and start new one, either at
left margin or indented 7 spaces
'depending on whether or not line numbers are enabled
```

```
if fn uppercase$(test.character$,l) = "E" gosub @maybe.else: if yes.else% then_lprint:
if print.numbers$ = "Y" then lprint tab(7)test.character$;: goto @done.testing else
lprint test.character$;: goto @done.testing
'plain, old, ordinary character, so let us print it on current line and try the next
one
lprint test.character$;
@done.testing
next counter%
'we're at the end of this line, so let's terminate it, reset quoted.strings% increment
line counter and read next line
lprint
quoted.string% = 0
line.counter% = line.counter% + 1
goto @read.record
'done reading this file, so be neat and close it
@done.with.file
close 1
'ring Epson's bell (optional) and do a form-feed
lprint chr$(7);chr$(12);
'and then go get another file name to list
goto @begin
'here's where we check to see if we've found an "ELSE"
@maybe.else
yes.else%=O
'if the line has less than 3 characters left, then it can't be an "ELSE", so to avoid
function call errors, skip it
if len(line.from.disk$)-counter% < 3 then @not.else
if fn uppercase$(line.from.disk$, counter%+l) <> "L" then @not.else
if fn uppercase$(line.from.disk$, counter%+2) <> "S" then @not.else
if fn uppercase$(line.from.disk$, counter%+3) <> "E" then @not.else
'passed all tests to qualify as an "ELSE", so make flag TRUE
yes.else%=-1
@not.else
return
@file.not.found
'make sure error is 'File not found'
if err <> 106 then @abort
'if the file isn't found on the first try, see if any extension was specified - add a
default, such as /tba, and try again
if instr(file.name$,"/") = 0 then file.name$ = file.name$ + "/TBA": resume @open.the.file
'if not found with extension, get another name
print "File ";file.name$;" not found on drive ";drive.number$;". Please try again."
resume @file.name
@abort
'error was something other than 'File not found'
on error goto 0
end
```

EASY VISICALC

by James E. Bruckart, M.D., 387 High Street, Hanover, PA 17331

The positive feedback I received from last Quarterly's "EASY LSCRIPT" article prompts me to discuss the idea of using LDOS's key-stroke multiply (KSM) feature to enhance VisiCalc.

VisiCalc uses a "presentation character" (i.e.-"/") prior to each user command. The LDOS keyboard driver and KSM facility allow the user to input discrete commands (that wi11 not

be confused with data) by using the <CLEAR> and alphabetic key combinations. For example, rather than </> <P> <P> to activate the printer mode, simply type <CLR-P>. In addition, type <CLR-L> to load a disk file or <CLR-S> to save the workspace to disk. LDOS's KSM facility issues the original instruction in response to the abbreviated commands, and hence there is no need to alter the original VisiCalc program (except to apply the appropriate patches designed to allow VisiCalc to run under LDOS). For special matrix operations, VisiCalc commands continue to work as before.

VC/KSM is created by typing BUILD VC/KSM, and typing the responses noted in listing 1. Plenty of blank responses are provided for your personal key combinations. For example, a frequently used file may be designated to load and save with the responses /SLfilename/VC; and /SSfilename/VC;.

I have included a "Help" feature which is selected by pressing <CLR-H>. This command saves the current workspace in TEXT/VC and loads the Visicalc file HELP/VC (listing 2). To return to the original workspace, type <CLR-C> (to clear the screen) and <CLR-Z> (to load TEXT/VC). HELP/VC is a sample of the mnemonic keyboard responses in VisiCalc format. It can be entered with a text editor or by typing BUILD HELP/VC from the LDOS Ready prompt. More inventive users may prefer a machine language "Help" routine that utilizes the /X- command provided by the LDOS patch to Visicalc (see Earl Terwilliger's Z-Command from the Jan 83 Quarterly).

Finally, I include a Job Control Language (JCL) program - EASY/JCL. The previous Quarterly missed the JCL program to initialize the Easy LScript configuration, so I have rewritten it to select LScript or Visicalc. Simply type DO EASY (program), where program is LSCRIPT or VC.

In summary, EASY VISICALC continues the trend to standardized and more memorable mnemonic commands. Frequent and occasional users will prefer the faster command input and more easily remembered commands. This "ease-of-use" can keep a program up-to-date and prevent its fall into disuse.

```
.Easy LScript and VisiCalc JCL Initialization Program
//IF LSCRIPT
FILTER *KI KSM LSCRIPT
LSCRIPT

//KEYIN Type <1> to initiate spelling checker =>
//1

PROOFRDR
//STOP
///
//EXIT
//ELSE
//SET NOTLSCRIPT
//END
//IF VC&NOTLSCRIPT
FILTER *KI KSM VC
VC
//STOP
//ELSE
//SET NOTVC
//END
//IF NOTLSCRIPT&NOTVC
.Syntax error - proper form is => DO EASY (program)
                where program is "LSCRIPT" or "VC"
//EXIT
//END
```

```
                                VC/KSM Filter Data
A=>           G=> /SLTEXT;  M=> /M        R=> /R       X=> /X
B=> /B;       H=> /SSTEXT;/SLHELP;        S=> /SS      Y=>
C=> /C        I=> /I        N=>           T=> >Al;     Z=>
D=> /D        J=>           O=>           U=>
E=> /SQ       K=>           P=> /PP       V=>
F=> /F        L=> /SL       Q=>           W=>


          Text for HELP/VC
>D8:"xt!
>C8:"eturn to tex
>B8:"CLR-G> to r
>A8:"<CLR-C> & <
>D6:"X Command
>C6:"Replicate
>B6:"Load
>A6:"Format
>D5:"W
>C5:"Q
>B5:"K
>A5:"Exit
>D4:"V
>C4:"Print
>B4:"J
>A4:"Delete
>D3:"U
>C3:"O
>B3:"Insert
>A3:"Clear
>E2:"Z
>D2:"Top
>C2:"N
>B2:"Help
>A2:"Blank
>E1:"Y
>D1:"Save
>C1:"Move
>B1:"Get Text
>A1:"A
/W1
/GOC
/GRA
/GC11
/X>Al:>A9:
```

Review of Profile III+ for the LDOS Operating System

by Sam Goldberg

Laurelwood Products, Inc.
12520D Quicksilver Drive
Rancho Cordova, Ca. 95670
(916) 351-0607
CompuServe # 72265,416

Profile III+ Data Management Package was written by The small Computer Company. This package is sold through Radio Shack Stores as the "hard disk version" or by ordering it directly from them at 230 West 41st Street, Suite 1200, New York, NY 10036, (212) 398-9290.

Profile III+ is a Data Base Management program designed for the Radio Shack Model III Micro-Computer. This package allows the user to define the initial structures, formats, and will permit a wide range of manipulations to the data base.

Profile III+ has several powerful features that I will discuss in more detail later on, but here are some highlights...

- Permits quick access to individual records or groups of related records
- 36 User Defined Search Fields
- Five User Defined Screen Formats
- Five User Defined Report Formats
- Five User Defined Label Formats
- Special Index Feature for High-Speed locating of a specific record or series of records.
- Custom User Menus for Internal Profile Use or Overall System Applications
- Math Package allows Addition, Multiplication, Subtraction, and Division of data within fields.
- Associated Fields can be 'Clustered' into search groups
- Password protection available for each screen, report, or label format
- Mass-Mode for Recalculation, Hardcopy, Deletion, and Purge features for specified records
- Separate diskette to allow access for selected employees to creation functions of Profile III+
- Can be used to pass information to Model III SuperScripsit for Form Letters
- Can pass information to Model III Visicalc for extended mathematical functions

The minimum equipment required for this package is a Model III Computer with 48k memory and One disk drive. The manual states that a two drive Model III is required, but I found that the program can be run with just a single disk drive or on the 5Meg Hard Disk Unit. You will also need a printer with printer cable to produce Hardcopy of records.

You receive this program on two 5 1/4" diskettes. The first diskette is the Creation Disk and the other is the Runtime Diskette.

Creation Diskette

This disk contains all the necessary programs to create your database system. This includes the modules to define the fields, screens, reports, labels, mathematical formulas, and selection formats.

In defining your file, you are prompted to name the file to contain your records. This is an up-to-eight character name that identifies your file. You are then able to define your fields. These fields are placed into one of four available segments. Each of these four segments can hold a record length of from 1 to 255 characters, with these characters being defined into field lengths with appropriate headings and field numbers.

In the first segment you can define related fields as being in the same logical group. These fields are treated as a single field for searching, sorting, and selecting records. I find this to be a very powerful feature.

You are also able to review and edit any segment in your fields, and can reserve characters previously unused for future additions (this feature is a life saver when you found out you forgot a heading ...)

Now you can define your screens (Yes, I said screens). Profile III+ allows up to five screens for your files. You can even switch instantly between them while updating or searching later on.

Another nice feature of the screen formatting is the availability to use the Model III's Special Character Set on the screen. However, these special characters are only for screen displays, as the printer translates them into periods during any printing from the screen (oh well, there goes the pointing finger...).

Field indicators define the type of entry you can make to each field. There are the standard indicators for alphabetic and numeric characters as well as several special feature indicators such as Must-Fill Alphabetic, Numeric, Decimal, and two forms of Must-Fill Date fields, Protected fields to show information but not allow updating, add-to and subtract-from fields, and three special date fields. These allow for limited errors in the entry of information into your fields, and can format the information into your records.

The Report section is an extremely flexible in its formation. For each of the five report formats that are available, you can assign titles for the headings as well as automatic page numbering, 'Date Printed', and Sort Field Information.

You have two lines for titles and the above information, two heading lines to head the information to follow below, and two lines to present the information from your records. These reports can be up to 132 characters in width, and will, if requested, space between each line of the report. In printing the reports, the program will automatically prompt you for a 'control break field' to divide your report if wanted. If you are using information you want totaled, Profile III+ will give a grand total at the bottom of-the report.

Creating Labels is just like creating a mini-report. All the features are available, and you have up to eight lines in each of your five label formats. One special feature I like is the "Move-Over" indicator. This will move the information to the left of another field and line it up only one space away, this looks great when first and last names are next to each other and not spaces away. I use a label format for one of my Customer Reports, as eleven six line labels fit on a standard page with room in between. I've found this format a dream when the record has more characters than will fit onto two 132 character lines in a report format.

Another special feature is the Selection Formats to pass information from your records to SuperScripsit and Visicalc. You can choose information to pass to these other programs and put it into any sorted order you want. Form Letters are now a snap as you can send a 'custom' looking letter to any or all of your clients. As a mailing list package, Profile III+ functions very well. For performing complicated mathematical functions, the power of Visicalc can be utilized just by sending the base information to it in the form of a label or number. A simple analysis of your information can sometimes show a hidden trend in costs, sales, or orders.

The math package that is included in Profile III+ can solve up to 16 formulas, each using up to 20 fields! Addition, subtraction, multiplication, and division are supported, and your answers can be automatically placed into the correct field. You can also assign 'real' numbers, such as fixed percentages, to be used in your formulas, as well as formatting the answer to integer or floating decimal place style. I found that you can place a calculation into a dummy field number to allow more complex calculations, and then move it back into the real field later.

Having the Runtime programs separate from the Creation programs will keep anyone from changing passwords or gaining access to your sensitive information. The program will note the password in upper and lower-case letters and WILL NOT substitute one for another! The only way to bypass a password is to use the Creation Diskette to re-define the format with a new password or none.

Killing a Data Base is easily done from either menu by pressing the 'K' key. This is not a listed option on the menu as being a choice (good!) and will prompt you for the file name of the file you wish to delete. Once you press the enter key after you specify the file name, all portions of the file will be deleted! There is, unfortunately, no second

prompt for verification of deletion, so you must be careful that you really want to delete the file.

The manual for Profile III+ is very complete in presenting a lengthy sample use of the package, with several examples and explanations. However, some of the more esoteric functions, such as the extended selection mode are not explained to their fullest, and the user must experiment to really understand the power of this feature. There are several pages of instructions for backup and format of blank disks for use with program, as well as hints and tips for usage.

This program is not a 'protected' package and allows for the user to make as many backups as they feel comfortable with. It comes in a nice binder with the instructions printed on good quality stock for long life.

I thoroughly recommend the package for anyone who has anything to keep in order. In our company, we are using Profile III+ to track all of our business from the time we make the bid, through receiving the order, manufacturing the products, through delivery, and to track final billing. A complete business package designed around a single data base management package - a clean, concise, and accurate solution. In combination with SuperScripsit and Visicalc, these three packages make a powerful set of programs that every business could use to promote computer use, save time, money, and put the time back your day for coffee breaks.

I think I'll have a donut with this cup.......

Color Comes to the TRS-80's

by Scott A. Loomer

reviewed: CHROMAtrs Color Graphics System
$168 kit, $208 assembled and tested

Have you ever found yourself trying to convince a Apple owner that having a sophisticated disk operating system was more important to you than color graphics? How would you like to tell the next one you meet that you have the best microcomputer DOS AND color graphics? If you feel inclined to turn the screw farther, you can also explain that your color graphics are superior to an Apple's. Interested? Read on ....

At least two companies are now advertising color graphics add-on systems for the Model I and III. This article will review the graphics capabilities, hardware and software of the first of these companies to get its product on the market, specifically the CHROMAtrs system from South Shore Computer Concepts, Inc., Hewlett, New York. The second company that has recently started advertising a color board is Micromint, Inc., Cedarhurst, New York. My remarks that deal with the graphics capabilities would apply to both systems since they employ the same color graphics processor (CGP), the Texas Instruments 9918A. If you are interested in more background information, I suggest that you read the article on sprite graphics in the August 1982 Byte magazine.

Graphics Capabilities

Enough of the background; what are we talking about here? The graphics capabilities are provided and managed by the TI CGP chip. Sixteen colors are available (counting transparent, black, white and 13 others) which also provide eight discrete luminance values (gray levels) if viewed on a black and white TV. The graphics capabilities of this system can be thought of as consisting of three layers: the background, the pattern plane and the sprites. The background forms a top and bottom border to the actual graphics area (the screen is clipped top and bottom to make it more orthogonal). The background can be set to any color which will also show through any portions of the pattern or sprite planes that are transparent. The pattern plane is visually in front of the background plane and is where high resolution graphics images can be created.

The resolution of this plane is 256 picture elements (pixels) horizontally by 192 pixels vertically. This resolution is the same as the Apple's; the TI CGP, however, has fewer restrictions on the placement of colors. All colors are available anywhere on the screen with the restriction that in eight contiguous pixels horizontally (a sliver) there can only be two different colors. In practice, this does not greatly hamper your artistry.

The nearest visual plane is where the power of the TI chip really becomes apparent. The system supports up to 32 prioritized sprites. A sprite is a pattern (either 8 x 8 pixels or 16 x 16 pixels) that can be defined and moved as a single entity anywhere on the screen. The sprites will appear in front of the pattern plane and will block out the pattern plane image except in areas of the sprite that have been left transparent. Animation of games becomes extremely easy. Merely define the shape of your hyper-megalon infinite-probability space ship and you can move the entire ship by passing one address to the CGP. When the image of the ship is moved, there is no effort necessary to restore the pattern plane image where it had been covered by the ship as the CGP maintains the image continuously. The sprites are prioritized which means that a sprite of higher priority will superimpose a lower priority sprite. There are some limitations to all this magic. Each sprite can only have a foreground and background (usually transparent) color. Since only one address is necessary to move a sprite, two or more different colored sprites can be combined to form an object and they can be moved apparently simultaneously even from a BASIC program. Sprites can also be moved off of the edges of the screen and will disappear line by line.

The graphics capabilities described above represent the highest resolution of the TI CGP and is termed Graphics Mode II. There are three other modes of operation available, Graphics Mode I (a more restricted version of Graphics Mode II), Multicolor Mode (low resolution, 64 x 48 pixels) and a text Mode (24 lines of 40 characters). The other graphics modes use less of the CGP's RAM and can consequently be loaded into the RAM somewhat faster than Graphics Mode II.

Hardware Description

The CHROMAtrs system from South Shore consists of a peripheral that attaches to a Mod I or III with ribbon cable to the system bus connector. The CHROMAtrs is powered by its own power supply. The video output from the CHROMAtrs is available as a composite video signal (for use with a monitor) or a radio frequency signal (with the optional RF modulator) for use with a color TV.

The system also contains the 16K RAM used by the CGP to build and store the graphics image(s) . The unit is port mapped to ports 78H to 7FH, but can be changed via jumpers if this conflicts with other peripherals. The South Shore unit consists of a circuit board approximately 9 inches square. The board is high quality with components nicely spaced, silk screened and solder masked. The board has both 40 and 50 pin edge connectors so that the same unit can be used with either a Model I or III (and presumably a MAX-80 via the 40 pin connector). South Shore has added two nice embellishments to the graphics capabilities of the TI CGP -- sound and joysticks. The sound capabilities are rudimentary, but a wide range of tones can be created that are brought out on the board to an RCA connector or can be heard on the audio of the attached TV set. The joystick capability supports two Atari style joysticks or four game paddles. The joystick data is read by doing an input from a port.

Construction of the kit form of the CHROMAtrs will not be difficult for anyone who has built projects with integrated circuits. I must confess to have had an atypical experience as I built the CHROMAtrs from the bare circuit board and not the kit. I did this only because I already had purchased the TI 9918 chip in anticipation of building a graphics unit from scratch. I DON'T recommend buying the bare board. South Shore's kit price is very reasonable and I doubt if you could save money by buying the components yourself; I didn't if you exclude the previously purchased TI 9918.

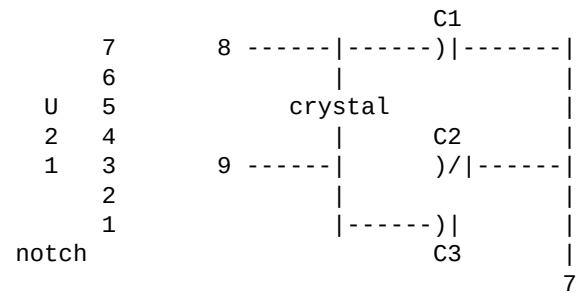
My construction of the unit probably represents an acid test of the instructions since I didn't have their parts assortment. There is an addenda sheet which should be posted to

the instructions prior to starting. There was only one inconsistency that I found; one value of resistor was missing from the parts list and necessitated another trip to buy them when I got to the installation step. I would make one suggestion for kit builders: socket all of the IC's. South Shore provides sockets for the high cost chips (TI 9918, RAM) but not the others. It may not seem to make sense to spend 30 cents to socket a 35 cent IC, but it will the first time you have to remove it (for whatever reason). Socketing will also ensure that you don't damage an IC by overheating. It's well worth the extra buck or two.

The caveat on kit building is, of course, that the darn thing may not work when you turn it on. The troubleshooting instructions are minimal so it helps to:

1) have a lot of test equipment and the knowledge on how to use it or
2) have a good friend (Phil Docken in this case) who satisfies item 1.

I di d have some problems that Phil helped me fix, but all but one are attributable from building the unit from the bare board and should not occur with the kit. The one problem that you might encounter is that the unit as designed may be fussy about the TV set that you use (symptom: black and white image, but no color). Phil talked with TI about the problem and has designed a small circuit with trimming capacitor that replaces one of the IC's and licked this problem for me. This is Phil's replacement circuit:



Parts List
Cl & C3 - 47 pf capacitors, Radio Shack #272-121
C2 - 5-60 pf trimmer capacitor, Radio Shack #272-1340

Instructions
1. Remove IC U21.
2. Build the replacement circuit on a small scrap of perfboard.
3. Attach jumpers from the connections marked 7-9 to the respective holes in the socket for IC U21.
4. With the CHROMAtrs on, adjust trimmer capacitor C2 for the best color.

Software

Software frequently determines the value of any peripheral. Well, as they say, I have some good news and some bad news. First some good news. South Shore provides a machine language driver that provides set, reset, test, line drawing, sprite definition, joystick and sound routines. The best part of this is that they supply the assembler source code for their driver as well.

Now the bad news. The driver is what I've come to think of as T**DOS standard. It is non-relocatable and uses the top 18k of memory! Give South Shore 10 brownie points for protecting the driver by setting HIGH$, but take away 20 points because they set both the Model I and III HIGH$ locations! When I called South Shore about this they were surprised to learn that this is not a good procedure. They said it hadn't caused them any problems under whatever DOS they had been using (I find that unlikely, as stuffing random data into any area of the resident DOS has got to mess something up). I convinced them that they should change it. The LDOS symptom was an inability to save programs from BASIC.

As stated, the other major problem with the driver is the loss of 18k of memory. It turns out that the driver code itself occupies less than 2k, but the top 16k of memory is reserved as a buffer for moving images into the video RAM. South Shore moves images (3 samples provided with the CHROMAtrs) into the video RAM by using the DOS LOAD command to place the 16k image (saved as a load module) into the top 16k of memory. A PUT command (supplied by South Shore) then merely moves the image into the external video RAM. This negates the entire advantage in the CGP having its own dedicated RAM. The balance of the software provided by South Shore consists of a BASIC program that has a line drawing demo (the line function of the driver is fast) and a lunar lander game. The game is primarily interesting as a demonstration of how sprites ease the programming burden so much that you can write a real-time game in BASIC. My last comment about the supplied software is to note that South Shore states that they will evaluate software that supports their board and offer it on a commission basis to their users via a newsletter.

Now let's look at what can and has been done to improve the state of the graphics software. Writing (or rewriting) a peripheral driver definitely makes one appreciate the myriad features of LDOS.

My first effort was to eliminate the image buffer. A new DISPLAY program was written that reads an image in as 256 byte blocks and transfers it a block at a time. This program takes about 4 seconds to access the 16k file on disk and output it to the CHROMAtrs. Parameters were also provided to set screen background color and to control the blanking of an image during the loading. New sprite tables could be loaded in without disturbing the displayed image for instance. The Z-80 OTIR command (block output) works perfectly as it is just slightly slower than the TI CGP data capture rate. Of course, that's at a Z-80 clock rate of 2 MHz. A delay loop would be required on a MAX-80 or any system using a faster clock. A program called FETCH was written to accomplish the opposite function, to grab an image out of the video RAM and save it to disk.
The second step was to make the driver relocatable. Would you believe 140+ relocated addresses? The article by Chuck Jensen on relocating code in the January 1982 LDOS Quarterly was a boon here. It ought to be noted here that there isn't a single relative jump in the entire South Shore driver (it turns out that all but two can be relative, however). The driver advises LBASIC of its entry points by passing a base address via the USTOR$ location provided by LDOS.

The third step was to make the driver initialization SYSGENable as was described by Roy Soltoff in the October 1982 LDOS Quarterly. I've since decided to rewrite the entire driver to include other functions (circles, filling areas, etc.).

There are many other exciting possibilities of this piece of hardware that I've not yet had time to pursue. The video RAM might have other non-graphic potentials. For example, how about a new version of the LDOS SPOOL command that uses the 16k of video RAM as its buffer? This would be nearly equivalent to buying one of the dedicated printed spoolers that typically cost about $159 for a 16k buffer. This way you would get color graphics for "free". A MEMDISK type program could also be developed that would have little impact on your main system RAM but would provide a transfer rate equal to a floppy.

If any of you now succumb to the urge to buy a CHROMAtrs and are interested in cooperating on the development of supporting software, please drop me a line. I think that the LDOS community is likely to take this peripheral far beyond the expectations of its designers.

Final Impressions

I don't want to leave you with the impression that the South Shore software is no good. With the exception of the HIGH$ problem, the software works as advertised. It just doesn't meet the standards that we've come to expect under LDOS. The capabilities that color graphics add to your system are tremendous. The dedicated TI CGP makes image creation and manipulation quite simple and fast; for example, it is possible to do some very credible programming using LBASIC because of this speed. The South Shore unit is

well designed and reasonably priced (though you have to add some components up to get the total costs):

Item	Kit	Built-Up
CHROMAtrs	$99	$169
Case	$18	included
Bus cable	$14/15	$14/15 (Mod I/III)
RF Modulator	$25	$25
Power Supply	$12	included
Total	$168	$208

All in all, the CHROMAtrs is an inexpensive peripheral that will add color to your life.

If you have any questions or comments about this article, please address them to:

Scott A. Loomer
315 Palomino Lane
Madison, WI 53705
608-233-7739 or MNet [70075,0933]

Since the preceding article was written in February, a few items need updating. First, the driver software now provided by South Shore Computer Concepts has the HIGH$ problem fixed. The new software comes with a BASIC program called MLOGO which is a creditable interpreter of a subset of the LOGO language. All of those fancy pinwheel designs that you've seen can now be yours. South Shore is also advertising a color graphics BASIC (appends to the DOS BASIC) for $30. I haven't had a chance to try this out yet, but it is supposed to be compatible with LDOS.

An additional piece of advice if you decide to purchase the ChromaTRS board is to order the Texas Instruments manual for the 9918A ($5) from South Shore at the same time. This recently printed manual gives precise details about what the chip can do and will allow you to utilize the unit to its capacity.

As a note to the MAXers in the crowd, the ChromaTRS board does work with the Lobo MAX-80. Two things must be corrected, however. First, the ground on the MAX-80 expansion BUS is only brought out on one line. You must jumper it to the two other ground lines on the ChromaTRS board. Second, the MAX is TOO FAST. The driver software must have delay loops added to slow the data transfer rate down to the 2 MHz that the color board can handle. I will provide the details on both fixes if you send me a SASE.

Last, I was chastised about my article on Alcor Pascal for not providing the company's address (and by trans-Atlantic call yet!) so here it is: South Shore Computer Concepts, 1590 Broadway, Hewlett, NY 11557 (516-569-4390).

..... er.....

by Earle Robinson
300 Grenola
Pacific Palisades, CA 90272

So many peop1e called in to LSI for the name of the pub1isher of the Plum book on 'C' programming which I mentioned in my last article, that I am putting it here for whomever may require it.

Learning to program in C
Plum Hall,
1 Spruce Avenue
Cardiff, NJ 08232.
Telephone:(609)927-3770

From The Author of Super Utility Plus™

INTRODUCING

THE LDOS™ UTILITIES

Kim Watt has written a complete set of utilities for use with LDOS. Unlike, SUPER UTILITY+, these will work on double-sided, 8" drives or rigid drives (including RADIO SHACK'S hard drive). They will work with Model I, III, LX80, or the MAX80 in single or double density. Each TOOLBOX consists of TWO diskettes, and contains all files mentioned below. Requires LDOS 5.1.3. Contains complete documentation. There is not room here to fully describe the package, so please write for COMPLETE information.

THE TOOLBOX FOR LDOS™

The two diskettes include:

PMOD/CMD
Very Sophisticated Disk/File/Memory Modification Utility.

PCHECK/CMD
Directory Check Utility for LDOS, including Rigid drives.

PFIX/CMD
A comprehensive directory repair utility that allows the user to completely repair most directory problems, and also has the ability to transfer a faulty boot sector from a good disk. All LDOS supported disk devices may be operated on, including hard drives. In conjunction with PCHECK, most directory problems can be easily located and corrected without extensive knowledge of how directories are formatted.

PFIND/CMD
Finds any occurance of strings, bytes, or words on the disk on a sector by sector basis. Optionally will REPLACE every occurance it finds.

PCOMPARE/CMD
Compare any file or any sector with any other file or sector for differences.

PREFORM/CMD
Reformat a disk without erasing the data! Strengthens the format tracks on older diskettes and easily repairs CRC errors and "sector not found" errors.

PVU/CMD
Verifies a disk for bad sectors.

PERASE/CMD
A disk bulk eraser through software for 5 in. floppy only. Removes all traces of data and returns the disk to a blank state.

PCLEAR/CMD
A comprehensive Disk Clean-Up Utility. Erases Unused Disk Sectors and Directory Records OR Clears Sectors in a File. Removes ALL traces of "killed" files for your protection.

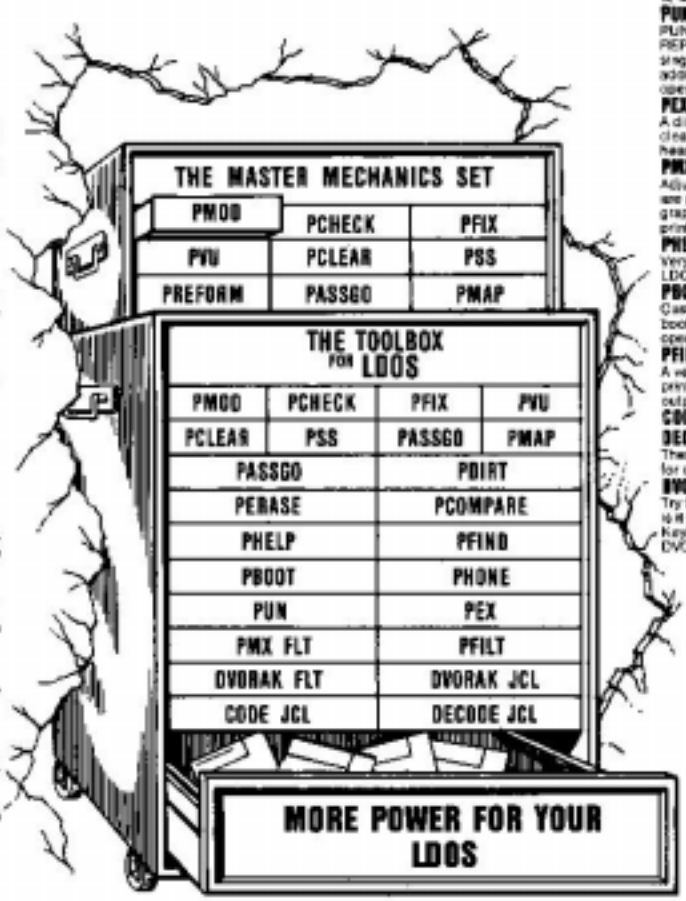
PSS/CMD
Sector Status. PSS allows you to easily identify which file is assigned to any sector or an entire diskette.

PMAP/CMD
Allows you to easily locate file sectors. You can map a file or the entire diskette.

PMOVE/CMD
A SUPER FAST, intelligent, multiple transfer routine. Allows you to type in a whole string of files, and then copies them lightening FAST! A must for hard drives.

PKILL/CMD
Will allow you to kill files in an orderly manner using a mask of your choice. Also a must for hard drives.

PDIRT/CMD
Reads a TRSDOS Model III directory from LDOS.



PASSGD/CMD
Easily remove passwords on a single file or an entire diskette.

PUN/CMD
PUN is the opposite of the LDOS REPAIR utility. It will UN-REPAIR a single-density disk so that the data address marks are readable by other operating systems. Model I ONLY.

PEX/CMD
A disk exerciser for the many head cleaning kits on the market. BOOTING a head cleaning diskette is not enough!

PMX/FLT
Allows graphic characters so that they are printed out as normal TRS-80 graphics when using an Epson MX-80 printer.

PHELP/CMD
Very complete HELP command for LDOS.

PBOOT/FIX
Customize the way your LDOS graphics boot up! Allows personalization of your operating system's appearance.

PFILT/FLT
A very comprehensive user definable printer filter that may be used for input or output devices.

CODE/JCL and DECODE/JCL
These files show how PFILT may be used for coding/decoding purposes.

DVORAK/FLT
Try the famous DVORAK keyboard! This is it in a filter form (a pure ascii data file). Keyboard toggle switchable between DVORAK or QWERTY.

$69.95

MASTER MECHANIC SET FOR LDOS™

Kim's MASTER MECHANICS SET consists of a selection of programs from THE TOOLBOX package as previously shown.

PREFORM/CMD	PCHECK/CMD
PMAP/CMD	PCLEAR/CMD
PMOD/CMD	PUN/CMD
PVU/CMD	PFIX/CMD
PASSGD/CMD	PSSTAT/CMD

$39.95

Enhance Your LDOS Operating System With Either Set. A Powerful Collection From:

POWERSOFT

Products from Breeze/QSD, Inc.

Available through selected dealers everywhere!

11500 Stemmons Fwy. Suite 125 Dallas, Texas 75229
To Order CALL TOLL FREE **1-800-527-7432**
For product information **(214) 484-2976**

LDOS is a Registered Trademark of LSI

Please excuse the plug, but I should like to announce that I can now provide SuperScripsit printer drivers for the Comrex, Diablo 630, Gemini 10, Gemini 15 & others will be added to the regular line, which includes the MX80/100, Prowriter, Nec l023a, C. Itoh Fl0, Qume, Brother. All drivers will run on the Model I, III, and the MAX80.

(Editor's note: We have been getting some of these inquiries. Please contact Earle Robinson directly.)

The Model 4 has finally made its appearance, and in spite of the excellent operating system written by you know whom (LSI), I was very disappointed in it. First of all, in spite of the market's eager acceptance of more ergonometrically designed microcomputers, especially due to the detachable keyboard, as shown in the IBM PC, the Tandy Model 12 among others, the new Radio Shack's offering is one big box. It is painted 'computer room' white, but is the same clumsy looking machine that the Model III was. The Model 4 looks like a 1980 computer, not a 1983 one!

Secondly, Tandy has introduced the machine over a month before I began writing this article, but there is still no technical manual available! Since a large number of early purchasers of a new computer model are software houses, OEM companies, and others wishing to develop programs or systems, how can they effectively do anything without technical specifications, listings and the like? IBM issued their very complete Technical Reference Manual, containing not only references, but a complete listing of the BIOS, at the same time as the PC. Since the Model 4 has no software yet, except the new version of Microsoft Basic, to run in the 6.0 mode (other than Misosys' excellent utilities), how can Tandy expect to maintain its already shrunken market share?

Finally, it seems to me that the price of the Model 4 is on the high side in the present market environment. It is possible to get much more for the money with say the Kaypro II which costs over $300 less and is bundled with an excellent package of word processing, dictionary, spread sheet, and two Basics. And, the Max80, with a couple of drives and a video monitor should cost somewhat less, too. Further, it is already possible to run 80 columns on the Max under 5.1.x. In the Compuserve data base of the LDOS board there is a driver which enables 80 columns. It works very nicely running EDAS, Lmodem80, and even in Basic!

In an earlier article I bad-mouthed the Brother HR-1 because it was incapable of doing anything else than print plain vanilla copy: no underlining, no bold, no superscript nor subscript. Well, the Brother people have quietly replaced the original chip in the printer so that it now does all those wonderful things. This means that if you are attracted by this printer, you won't have to pay an extra $100 to get the Comrex, which until now was the only way to have a Brother which would provide the underlining, etc. For those of you who have an older version, I suggest you talk with your dealer about getting the new chip installed.

Among the new products I have recently been able to try, I particularly liked two of the programs on the LSI Utility disk #2. The first one is based on an old enhancement I did for early versions of LDOS which permits re-defining keystrokes from within a KSM file. For example, let us say that I am working on a program called 'Myfile'. By redefining a key from within Basic or EDAS to output that name each time as required, I save time, and avoid errors of misspelling which can occur. This program is called Ksmplus on the disk. The other program is a very elegant filter for input or output to translate a user defined key into a group of characters, up to 255 in length. This is a sort of expanded KSM type filter which will permit translation of output of whatever you want from the keyboard, a communications port, or any other device. Similarly, it can be used for input translation into whatever you wish, to the video display, a printer, or whatever device you wish. This program is called Maxlate. The concept is beautiful, and its implementation ideally done.

I am enclosing a little program to change your prompt device from the 'LDOS Ready' which we know so well. So, if you prefer a '%' or a '$' 1ike under Unix, or whatever, all you have to do is type the command as you desire it. Use a '=' as the first character if you

wish to suppress the 1ine feed which normally precedes the 'LDOS Ready' prompt. And, if you wish to suppress the carriage return which normally follows the prompt, use the '=' as the last character. So, if you wish to have a '%' without linefeed and without a carriage return you would type 'Prompt =% =' then hit <Enter>. That will give you the prompt without a line feed and without a carriage return. Or, you could type 'Function: ', if you wish that as your command and wish to have both the linefeed and a carriage return. To get back the 'LDOS Ready' merely type 'Prompt'. For those of you who may not wish to type the hex fi1e below and then convert it, you may send me $10 and I'll send PR0MPT to you, together with primitive instructions. I hope that I don't need to remind you that you should NOT use this program with the SYS1 file on your Master diskette!

Until next quarter, when I hope to talk a little about UNIX, and what it may mean to all of us. Perhaps, I shall have had the opportunity to meet some of you at the LSI open house before you read this issue of the Quarterly.

```
05 06 50 52 4F 4D 50 54 01 02 00 52 22 68 53 11 99 53 21 D0 54 06 00 CD 24 44 C2 A2 52 01
03 00 CD 42 44 CD 36 44 CC 4D 53 11 6A 53 21 D0 53 06 00 CD 24 44 20 72 01 04 00 CD 42 44
CD 36 44 20 6D D5 16 0C 3A 67 53 B7 C2 53 53 06 FF 7E FE 4C 23 28 04 10 F8 18 63 7E 05 FE
44 23 20 EF 2B 2B 2B ED 4B 68 53 0A E5 03 FE 3D 28 03 23 0B 15 0A FE 3D 28 0C 77 23 03 FE
0D 28 08 15 28 42 18 EF 3E 03 77 E1 D1 01 04 00 CD 42 44 CD 3C 44 CD 28 44 20 21 11 99 53
22 D0 54 01 03 00 CD 42 44 CD 3C 44 CD 28 44 C3 2D 40 21 BA 52 C3 09 44 21 DA 52 C3 09 44
21 F1 52 C3 09 44 21 08 53 C3 09 44 21 21 53 C3 09 44 54 68 65 72 65 20 69 73 20 6E 6F 20
73 79 73 31 20 6F 6E 20 79 6F 75 72 20 73 79 73 74 65 6D 0D 52 65 61 64 20 65 72 72 6F 72
2E 20 20 41 62 6F 72 74 69 6E 67 21 0D 57 72 69 74 65 20 65 72 72 6F 72 2E 20 41 62 01 02
00 53 6F 72 74 69 6E 67 21 0D 54 68 65 72 65 20 69 73 20 6E 6F 20 6E 65 77 20 6D 65 73 73
61 67 65 21 0D 54 68 65 72 65 20 61 72 65 20 6F 76 65 72 20 31 30 20 63 68 61 72 61 63 74
65 72 73 20 69 6E 20 74 68 65 20 6D 65 73 73 61 67 65 0D 3E FF 32 67 53 C9 2A D0 54 ED 4B
68 53 E5 0A FE 0D C2 5B 52 01 C4 53 C3 63 52 00 00 00 53 59 53 31 2F 53 59 53 2E 45 5A 54
4F 3A 30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 50 52 4F 4D 50 54 2F 43 4D 44 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0A 4C 44 4F 53 20 52 65 61 64 79 0D
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 D3 00 54 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 02 02 00 52
```

***** PARITY = ODD *****

(c) 1983 Tim Daneliuk, T&R Communications Associates
4927 N. Rockwell St., Chicago, IL 60625

This installment of PARITY = ODD marks one full year since I've been writing regularly for the LDOS Quarterly. Since a good deal of my computer related work involves not only program development, but technical writing ABOUT the industry, I decided to dedicate this issue to a review of computer magazines!

A recent editorial in 80-U.S. Journal exhorted the readers to treat their magazines as "peripherals". The writer had a point. A well conceived and produced magazine can serve as a great teacher, and save you a lot of grief. On the other hand, magazine subscriptions have been steadily increasing in cost. If you're like most microcomputer users, you can ill afford to waste 20 or 30 bucks on something which doesn't meet your needs.

I decided to evaluate the four magazines which are most likely to come to your attention as a TRS-80 owner. Two are industry-wide journals, and two are specific to the TRS-80.

In each case, I looked at three major areas in the evaluation: 1) Technical Depth / Readability, 2) Timeliness of the magazine's contents, and 3) Quality of the product reviews. The first point needs a little explanation. Merely because a magazine has only tutorial material or deeply technical material doesn't necessarily make it bad. There is a place in the magazine market for journals which are directed only to a very narrow group of users. The real question is whether that magazine presents material appropriate to their intended audience in a readable form.

I've also included the latest advertising rates for these magazines that I could find. A magazine derives the lion's share of its income from advertising, not subscriptions. In turn, these rates affect the cost of computer products we buy. I have quoted only the "one time" rate for an ad, which is the most expensive rate there is. Keep in mind that the more often a given ad is scheduled to appear, the lower the rate per publication becomes. Finally, for those of you with an inclination to write, I've included what payment for a typical article might be. This figure is based on my own experience in working with these magazines.

BYTE

BYTE has been around longer than virtually every other microcomputer magazine. It started out with a heavy emphasis on hardware "hacking", and some of this Emphasis still remains. With the advent of "appliance" computers such as the TRS-80, Apple, and Pet, BYTE's articles and reviews have changed to encompass these machines. Of all the popular computing magazines, this one has the most technically thorough articles. If you enjoy reading material dealing with abstract areas of computer design such as searching and sorting algorithms, benchmark design, and interface standards, BYTE is for you. The one objection I do have to these articles is that they tend to be very machine specific. I would rather see a generalized article on writing efficient sorting algorithms than a program listing of such a sort which only runs on a DEC 350, or a TRS-80, or whatever. Articles which rely heavily on demonstrating a concept only by programming example usually don't explain the concept itself very well. In general, BYTE articles are not for the rank beginner. They assume some computer experience on the reader's part. Apart from that, the writing is quite readable, and well edited.

In the area of timeliness BYTE falls flat on its face! I personally have had articles submitted which took almost a year to get into print. I've also seen reviews published in BYTE when the product in question was no longer being manufactured (i.e. the BASE 2 printer). The so-called "news" sections of this magazine are usually a minimum of 3-4 months behind what the industry is actually doing. In fact, I have NEVER read a news item in BYTE's pages which had not already appeared in several other industry journals.

BYTE product reviews are probably the most comprehensive published. This does not necessarily mean that they are good, they're just long! Time after time, I've seen software reviews which contain a detailed description of every command in the product, the command syntax, and a complete explanation of the product's installation procedure! This is decidedly NOT what a review is supposed to be. These details are what the instruction manual is for. A review should tell you what a product is supposed to do, how well it does that task, the quality of the documentation, and whether value is being delivered for the money. In my experience, reviews which go into endless detail usually miss the point of a particular product completely.

In all fairness to BYTE, good reviews are hard to come by. For example, someone reviewing a new 16 bit personal computer really should have experience with at least one other similar machine so a basis for comparison exists. This is not usually the case, so the review suffers somewhat. The only other choice the magazine editors have is to get review samples of various machines and do the evaluation themselves. One thing that BYTE could do to alleviate this problem would be to establish a set of uniform review guidelines which all reviewers would have to follow. The one area in which BYTE reviews DO excel is that of comparative reviews. Many magazines will not do a side-by-side comparison of several similar products, perhaps in fear of losing advertisers. BYTE, however, has no qualms about this, and several excellent comparisons of products have been run in its pages in the past few years.

BYTE Summary
============
TECHNICAL DEPTH: INTERMEDIATE TO ADVANCED
TIMELINESS: EXTREMELY POOR
REVIEW QUALITY: GOOD (But could be much better)

Full page ad, 1 time: $5950 (Oct. '82 Rate Card)
Typical Payment To Author: $50 per published typeset page

INFOWORLD

INFOWORLD is a relative newcomer to the world of computing magazines, but it already has a prestigious reputation. This magazine is published every two weeks and makes no attempt to be a "how to" magazine. You won't find articles on programming technique, interface design, or the like here. Instead INFOWORLD concentrates on the news of the industry. When new products are announced, you'll hear about them almost immediately in INFOWORLD. There are also several columns which deal with trends in the industry as a whole, as well as stories about people in the industry. The writing style is light and eminently readable.

INFOWORLD seems to appeal primarily to people who USE their computers a lot, like lawyers, accountants, and so on. It keeps them in touch with the latest products and trends in the microcomputer industry. By its very nature INFOWORLD is an extremely timely magazine. News items are right up to date, and you will often see things in INFOWORLD which don't appear elsewhere for months. Their product reviews are generally well chosen, and accurately reflect the present state of products available to the consumer.

INFOWORLD is also THE leader in product reviews. All reviewers are given a set of guidelines to which they must conform. Software is evaluated in the categories of PERFORMANCE, EASE OF USE, DOCUMENTATION, and ERROR HANDLING. Hardware is categorized by SETUP, EASE OF USE, PERFORMANCE, DOCUMENTATION, and SERVICEABILITY. Each review concludes with a "Report Card" which rates each of these areas as Poor, Fair, Good, or Excellent. No other magazine reviews have the consistency that INFOWORLD reviews have. If you're considering a major hardware or software purchase, this is one resource you shouldn't ignore.

INFOWORLD Summary
=================
TECHNICAL DEPTH - None - News Magazine
TIMELINESS - EXCELLENT
REVIEW QUALITY - EXCELLENT

Full page ad, 1 time: $3780 (Apr. 4, '83 rate card)
Typical payment to author (review): $85

80-MICROCOMPUTING

80-Micro is a TRS-80 specific magazine, though it does occasionally address other brands of computers. This magazine tries to address all segments of the TRS-80 industry. The material ranges from tutorials to such complex areas as microprocessor emulation. This is almost all "how to" material though. 80-Micro makes little attempt to deal with the design of software and hardware, but rather presents the finished results.

A majority of each issue's program listings are available on cassette or disk to save you the drudgery of typing it in yourself. This is a great idea, and one which more magazines ought to follow. The writing style varies, but is generally acceptable for the magazine's intended audience. There is one major problem with this magazine, however. It has an editorial policy which not only enjoys controversy, but manages to create it as well . This is in no small measure due to the publisher and owner, Wayne Green. Mr. Green's editorials are generally full of shots at Radio Shack, the computer industry, and almost everyone except Wayne Green Inc.! Now, there's no question that Radio Shack and others occasionally have deserved a slap on the hand for a bad decision, but I find it

difficult to believe that nothing good EVER comes from Fort Worth. I find these editorials pompous, unsubstantiated, and hypocritical. For example, Mr. Green has on several occasions taken Radio Shack to task for not offering enough value for the cost of their products. Yet, if you look at 80-Micro itself, the value delivered is rapidly decreasing. The subscription rate is being increased, but the number of pages per issue appears to be decreasing! There is little here that you can't get in other magazines, so I cannot recommend you spend you money to read Mr. Green's self-serving editorials.

Just like BYTE, 80-Micro gets a decided thumbs down in the timeliness department. The "news" articles rarely are, and the main articles are often dated by the time they appear in print. About the only up-to-date material you can rely on in 80-Micro are the product reviews.

The reviews in 80-Micro are similar in form to those in BYTE. Each reviewer sets pretty much their own approach to the product being reviewed so there is little correlation between reviews written by different people. 80-Micro does publish comprehensive listings of a given type of product from time to time, and these are quite good. For example, reports on word-processing products and programming aids recently were published. I found both of these articles informative, current, and truly useful.

80-MICROCOMPUTING Summary
==========
TECHNICAL DEPTH - "How To" Primarily/ Little Design Material
TIMELINESS - POOR
REVIEW QUALITY - FAIR to GOOD
Full page ad, 1 time: $2395 (Jan. 15 183)
Typical Payment to Author: $50 per typeset printed page

80-U.S. JOURNAL

80-U.S. is also a TRS-80 only magazine. Its thrust is primarily for the novice programmer and newcomer to the world of personal computing. There is little here for the intermediate or advanced user, but there is a WEALTH of information for the rank beginner. If you are new to all this, 80-U.S. is an absolute MUST. You can learn more in one issue of this magazine than you'll learn in weeks of "hacking" by yourself. The writing style is direct, and delightfully simple to understand. Though the majority of the material in 80-U.S. is "how to", there is also usually an article or two on the "why" behind a particular TRS-80 feature or command. Most examples are done in BASIC, and 80-U.S. hasn't forgotten that cassette only systems are still in use.

Insofar as 80-U.S. presents newsy material, it is fairly current. The first published reports on the TRS-80 Model 100 and Model 4 were in 80-U.S., as were a string of Tandy product announcements.

The reviews are usually also reasonably current. 80-U.S. has two types of reviews. There is a collection of "mini" reviews in each issue as well as one or two in-depth reviews as feature articles. I generally find these to be too superficial to be of any real help in determining the value of a product. Several times I have seen reviews of products which I KNEW were poor, but got passable ratings in 80-U.S. Here again, a set of uniform review guidelines would help enormously.

80-U.S. JOURNAL Summary
==========
TECHNICAL DEPTH - NOVICE to INTERMEDIATE
TIMELINESS - GOOD to EXCELLENT
REVIEW QUALITY - POOR to FAIR
Full page ad, 1 time: $1000 (Feb. 1, '83 Rate Card)
Typical payment to author (article): $l00

SPENDING YOUR MONEY

In my estimation, everyone should get INFOWORLD. It is THE source for information about the computer industry. If you're a beginner, you'll want to subscribe to 80-U.S. for a

year or two, 'til1 you get your feet wet. There really is no good magazine for the advanced TRS-80 programmer, so you're pretty much on your own if you're past the beginner stage. Perhaps someone with a lot of cash would 1ike to start a new magazine?.....

THE "C" LANGUAGE (Part III)

by Earl 'C' Terwilliger Jr.
647 N. Hawkins Ave.
Akron, Ohio 44313

WELCOME back for Part III! As promised, the main topic for this article is operators. The C language is very rich in its capabilities of data manipulation. Operators perform this data manipulation. In the C language, operators are usually grouped into categories. These categories (or types) are shown in a chart below.

Operators specify what is to be done to variables and constants. Operators, when combined with variables and constants, are called expressions. When an expression is followed by a semicolon it becomes a statement. In the next article we will see more on expressions, statements and the logic control and flow in the C language.

Before I go on, I would like to clarify a few things you will encounter in the C programming language in dealing with expressions and operators. Basically, these points involve the concepts of]value, rvalue, "side effects" and type conversions.

An lvalue (left hand value) is an expression which references storage in the computer. In C, an object is a manipulatable area of storage. An lvalue is an expression which refers to that object. An lvalue is the only value which can be on the left side of an assignment statement. An lvalue or an rvalue can be found on the right side of an assignment statement. An expression which is not an Ivalue is sometimes called an rvalue. An rvalue doesn't refer to an object (area of storage). An example? Consider the following C statements:

```
int c;
c = 0;
```

For Z80 based machines, if the HL register pair was used to obtain the value of c from the stack, you might see the following instruction (among others) generated as a result of compiling the above C statements:

```
LD HL,O        ;Load HL pair with 0
```

The 0 is contained in the instruction and doesn't take up any data storage. Side effects come into play when there are no explicit C rules as to how an expression is to be evaluated. In this article, a chart is used to summarize the rules of precedence and associativity for the various operators.

If no rule of evaluation applies, the C compiler is free to evaluate an expression or subexpressions as it sees fit. The results (expected or not) are called side-effects. To overcome side-effects, break up larger expressions or statements and use explicit "temporary" lvalues.

When expressions involve arithmetic with different data types, a conversion (or promotion) to a common data type occurs. Since both data types are of a common type, the result is also of that type. Since "lower" data types are converted to "higher" types, the process is called promotion. Of course there is a set of rules for this conversion or promotion. The "ladder of promotion" (if I can be allowed to invent a phrase) is shown to illustrate this promotion:

```
double
float
unsigned long
long
unsigned int, unsigned short
int, short
unsigned char
char
```

Besides arithmetic expressions, type conversions can take place when a function is called. After all, a function argument is an expression. Conversions can also take place across an assignment statement. When this happens the value on the right side is converted to the type on the left side. The above promotions can occur or the reverse process can also occur. (Demotion!) A word of caution! The C language guarantees any character in the machines standard character set will never be negative when used in arithmetic expressions. However, C does not specify whether variables of type char are signed or unsigned. So, be careful when using various bit patterns (other than the standard character set) because you may be dealing with negative values. Later, we will see how to force a type conversion using the cast operator.

Now that these points have been mentioned, let's get back to more on expressions. As an expression is evaluated , there is a precedence or order of evaluation. It is important to know the order of evaluation when the different types of operators are combined in an expression, especially if you want a correct result! Below is a chart of the operators available in C. It shows the relative order (level) of precedence, and the associativity of operators of equal precedence. The associativity is given to show how expressions are evaluated if operators of an equal level of precedence are found side by side in the same expression. (You saw the following chart in Part II.) It is repeated here with examples of each operator and sample expressions.

Operators	Level	Type	Associativity
() [] ->	15	Primary	left to right
! ~ ++ -- -	14	monodactic	right to left
(type) * & sizeof	14	monodactic	right to left
* / %	13	arithmetic	left to right
+ -	12	arithmetic	left to right
<< >>	11	shift	left to right
< <= > >=	10	relational	left to right
== !=	9	relational	left to right
&	8	bitwise logical	left to right
^	7	bitwise logical	left to right
|	6	bitwise logical	left to right
&&	5	logical	left to right
||	4	logical	left to right
?:	3	conditional	right to left
= += -= *= /= %=	2	assignment	right to left
|= ~= &= >>= <<=	2	assignment	right to left
,	1	comma	left to right

Since variables must be declared before they are used, I will declare a few so they can be used in our sample expressions.

```
int a, b, c, d;
```

Variables can be initialized in their declaration statement. As you might guess, an operator (the equals sign) is needed to accomplish this. For example:

```
int a = 10;
int b = 5;
int c = 2;
```

An easier way to initialize the above variables is:

```
int a = 10, b = 5, c = 2;
```

(Well... anyway it takes less lines in your program.) To accomplish the initialization, the variable name is followed by an equals sign and a constant. The constant's value serves as an initializer. Multiple initializations can be performed in a single statement (as is done above) when separated by a comma. The comma, used in this way as a separator, is not an operator and does not mandate the rules of evaluation as does the comma operator. Some variable types, external and static, are automatically initialized to zero by default of the C compiler. However, as stated by K&R, it is better programming practice to state their initialization anyway. If automatic variables are not initialized, their values will be of an undefined value. (Garbage!) You should note that for external or static variables, the initialization is done only once.

For automatic variables which are initialized in a function, they are re-initialized each time the function is called.

I may need to declare a few more variables to illustrate all of the operators. However, I will do so when each individual operator is introduced. Let's begin!

Operators	Level	Type	Associativity
() [] ->	15	Primary	left to right

printf("C is a very useful language to know!\n");

The () is called the function call operator. The left and right parentheses separate any optional arguments passed to the called function.

char buffer[100]

The [] operator(s) is used for arrays. In this example, an array of characters, whose name is buffer, is declared to have 100 elements. The [] is used to reference (index or subscript) elements of the array. In this example, the elements are buffer[Ol thru buffer[99]. In C the index starts at 0. Be careful about your use of these subscripts, since the C compiler does not check to see if the subscript you use is within the bounds of the array.

struct date { int month, day, year;} birth;

```
struct date *pdate;
pdate = &birth;
birth.month = 10;
pdate->day = 3;
```

The . and the -> (period and arrow) operators are used when dealing with structures. The "." operator is called the structure member operator because it connects the structure name and a member name. Pointers to structures are very frequently used, due to the restrictions imposed by the C language. These limitations imply that structures are not to be dealt with as a single unit. This limitation does not hold for pointers to structures. Therefore a shorthand operator, ->, is used to make pointers to structures and structure members easier to use. In the examples above, birth is a structure of type date, month and day are members of the structure and pdate is a pointer to a structure of type date. More later on structures! (As you can see, we haven't yet discussed the * and & operators!)

Operators	Level	Type	Associativity
! ~ ++ -- -	14	monodactic	right to left
(type) * & sizeof	14	monodactic	right to left

All of these operators are unary or monodactic operators. This means they need only one operand.

!a

The truth value of a is reversed. If a was TRUE (non-zero) it is now FALSE (zero). If a was FALSE it is now TRUE. Since I declared a to be 10 earlier, after the !a, the result returned will be FALSE (zero).

~a

The one's complement of the operand a is returned. Just invert each bit of the operand. If a bit was a one switch it to a zero and if the bit was a zero switch it to a one.

++a --a a++ a--

These are the increment ++ or decrement -- operators. They may prefix or postfix the operand. If the operator prefixes the operand, its value is taken after the increment or decrement. If the operator postfixes the operand, its value is taken and returned by the expression before it is incremented or decremented. These operators do not necessarily mean to add 1 or subtract 1 from the operand. The increment or decrement value is based upon the storage TYPE of the operand!

-a

This is the unary minus operator. It reverses the sign of a. After -a, since a was declared to be 10, the result is now a –l0. (There is no unary + operator.)

myfunc((double) c);

The variable c was defined as an integer above. If the function myfunc() needs to be passed a double precision floating-point variable, c can be coerced into that type. This construct of forcing a variable into another type is called a cast. In this example, c is cast into type double.

char name[30];

```
name = "Earl C. Terwilliger Jr.";
char *ptr;
ptr = &name;
```

The & operator obtains the address of an object. (The & operator applies only to variables and array elements.) The * operator precedes an operand containing an address. This address is then used to fetch the data stored there. A lot more will be said about these operators when pointers and arrays are discussed in greater detail!

sizeof(a)

This operator returns the size of any object. The value returned is determined by the number of units contained in the object. Each "unit" or "byte" has the same relative size as type char.

Operators	Level	Type	Associativity
* / %	13	arithmetic	left to right
+ -	12	arithmetic	left to right

```
d = a * b;              /* a multiplied by   b */
d = a / b;              /* a divided    by   b */
d = b % c;              /* b modulus         c */
d = a + b;              /* a added to        b */
d = a - b;              /* b subtracted from a */
```

The *,/,+,- operators perform the arithmetic functions of multiplication, division, addition and subtraction respectively. Since integer division in C truncates the fractional part, use the % or modulus operator to obtain the remainder. (% can not be applied to float or double operands.) (Note that parentheses are also used around

expressions to specify order of evaluation. Watch out! The C compiler can sometimes rearrange a parenthesized computation. For some associative and commutative operators (remember your arithmetic for * and + ?) the C language does not specify the order of evaluation. Use explicit temporary variables to overcome this, if in fact this re-arrangement makes any difference.)

Operators	Level	Type	Associativity
<< >>	11	shift	left to right

```
a >> 2              /* a shifted right by 2 bits */
b << 4              /* b shifted left  by 4 bits */
```

These shift operators perform left and right shifting of the bits in their operand. The number of bits shifted is the right operand. When an operand is shifted left, vacated bits are filled with a 0. Right shifting an unsigned operand causes the vacated bits to be filled with zeros. Right shifting a signed operand may be an "arithmetic" or "1ogical" depending on the machine and the C compiler.

Operators	Level	Type	Associativity
< <= > >=	10	relational	left to right
== !=	9	relational	left to right

```
a <  c              /* a less than c */
a <= c              /* a less than or equal to c */
a >  c              /* a greater than c */
a >= c              /* a greater than or equal to c */
a == c              /* a equal to c */
a != c              /* a not equal to c */
```

Each of these operators takes two operands and returns the result TRUE or FALSE (1 or 0). The result is based upon the relationship of the two operands. (Note: one of the more frequent errors by new C programmers is the failure to differentiate the == operator from the = operator. The == operator tests for equality while the = always sets equality.)

Operators	Level	Type	Associativity
&	8	bitwise logical	left to right
^	7	bitwise logical	left to right
|	6	bitwise logical	left to right

```
a & Ox0f            /* a bitwise ANDed with Ox0f */
a ^ OxfO            /* a bitwise XORed with OxfO */
a | Ox01            /* a bitwise ORed  with Ox01 */
```

These operators provide C the capabilities of bit manipulations. Bits of the left operand can be ORed, ANDed or exclusively ORed by bits in the right operand.

Operators	Level	Type	Associativity
&&	5	logical	left to right
||	4	logical	left to right

```
&&   and
||   or
```

These operators logically connect their operands. (They should not be confused with the & and | bitwise operators.) These operators can be used to connect expressions. For examples:

```
(a == 2) || (a == 3)
(a != 0) && (c == 2)
```

Operators	Level	Type	Associativity
?:	3	conditional	right to left

```
(a == b) ? (c = 1) : (c = 2)
```

This ?: construct is a ternary (three) operator. It provides an alternative to the IF . . . THEN ... ELSE . . . type logic. If the expression to the left of the ? is TRUE then the expression immediately after the ? is evaluated. If it is false then the expression after the : is evaluated.

Operators	Level	Type	Associativity
= += -= *= /= %=	2	assignment	right to left
|= ~= &= >>= <<=	2	assignment	right to left

```
a += 1              /* Add         1 to     a */
a -= 3              /* Subtract    3 from   a */
a *= 5              /* Multiply    a times  5 */
c /= 2              /* Divide      c by     2 */
c %= 2              /* Store remainder of c/2 into c */
c |= Ox80           /* c is bitwise ORed   with Ox80 */
c ^= OxO2           /* c is bitwise XORed  with Ox02 */
c &= Oxf0           /* c is bitwise ANDed  with OxfO */
c >>= 2             /* c is shifted right 2 bits */
c <<= 4             /* c is shifted left  4 bits */
```

These operators represent the assignment operators. They were seen before without the equals sign. They make expressions such as: a = a + 1 much easier when written as: a += 1 . Usually the resultant assembler (machine) code produced as a result of these operators is more efficient too. They are especially helpful if a long complicated expression is on the left of the = and needs to be used to the right of the =.

Operators	Level	Type	Associativity
,	1	comma	left to right

The comma is used in C as punctuation. It is seen separating expressions in declarations and it is seen separating arguments passed to functions. The comma separator is different from the comma operator. The comma operator is usually found used in a for statement. An example of a 'for' expression? OK!

```
for (i = 1, j = 2, c = 3; i != 50; ++i)
```

In the next article, I will be discussing the logic of a C program (the control and flow of how and in what order computations are done). Until then, read some more of K&R and the rest of the LDOS QUARTERLY!

There is a possibi1ity of misunderstanding the PDSHELP/FIX patch that was in the PDS article in the January 1983 Quarterly. This patch is to be applied to a new PDS after you have created it with the BUILD command. - Scott Loomer -

Since the January 1983 Quarterly article was written, Alcor has introduced version 2.0 of their Pascal system. The upgrade cost to current owners is $21 which includes new disks. Briefly, these are the enhancements in version 2.0. The Pascal functions GET and PUT have been added (the only Jensen and Wirth langauge standards that had been missing). The flag register status is now returned in the machine language CALL. The procedure to internally define file names has been simplified. Random access files are now supported. The additional compiler options INCLUDE, LIST, PAGESIZE, WIDELIST RANGECHK and PTRCHECK have been added. The LOCATION function will now work with procedures as well as variables. The linking loader will now accept lower case. The error in subrange variables has been corrected.

As is typical , just prior to getting the update announcement, I had just finished patches to version 1.2 to allow 1ower case filespecs in all modules. After I've examined 2.0, I wi11 submit any of the patches that are necessary to ensure complete lower case compatibility. - Scott Loomer -

Patches, patches, patches ....

The following is a mandatory patch to FORMAT/CMD. This is for all versions of FORMAT produced with file dates before 06/l0/83. This patch is the same for al1 5.1.3 releases, and is the same for Model 1, Model 3, and MAX-80.

```
. FORMATC/F3X - 06/08/83
.
.  Corrects problem with setting "D" BEFORE calling
.   SELECT so the FDC driver can properly deal with precomp
.
X'60DF'=8B 67
X'6OE5'=08 64
X'678B'=FD 56 05 C3 F4 63
.
. EOP
```

Optional Patches

```
. Patch to Enhanced VisiCalc to remove parallel printer ready check
. this is necessary if the *PR device is routed to a serial printer
.
. original contents was: 3A E8 37
.
. Patch for version VC-150Y0-T83
D55,2C=AF C9
.
. Patch for version VC-160Y0-T83
. D55,4D=AF C9

. Patch to extend the keyboard debounce time for 'problem' keyboards
. NOT for MAX-80
. file to patch is: KI/DVR       original contents was: 03
.
. for Model 1
D01,A9=04
.
. for Model 3
. D01,AC=04
```

```
. Patch to disable the Model 3 external I/O bus on boot, if non-standard
. peripherals are causing a problem.  Do NOT use this patch on a
. hard disk system
.
. file to patch is: SYS0/SYS      original contents was: 38
D0D,20=28


. Patch to use "." rather than "/" and ":" on bootup DATE and TIME prompts
. NOT for MAX-80
. file to patch is: SYS0/SYS      original contents was: "/" and ":"
.
. Model 1
X'4EB8'="."
X'4FA5'="."
.
. Model 3
. X'4EB9'="."
. X'4FA6'="."


. Patch to allow LPRINT TAB past position 63 (up to 127)
. this patch is for MAX-80 ONLY!
.
. file to patch is: SYS0/SYS      original contents was: 3F
.
X'213B'=7F


. Patch to force lower-case driver on boot.  This is necessary on some
. non-RS type lower-case mods
. this patch is for Model 1 ONLY
.
. file to patch is: SYS0/SYS    original contents was: 2V 06

X'4E36'=00 00
```

LET US ASSEMBLE

by Rich Hilliard

Welcome back. Last issue we listed several methods of working on your ego so that along with assembly language expertise, you could become (as is normally the case) an insufferable egomaniac at the same time. Several letters from Country & Western Fans pointed out that C & W people may substitute the song "It's Hard to be Humble" rather than "Nobody Does It Better". I agree that this is a fine substitution. Broadway buffs may substitute "C'est Moi, C'est Moi" (It's me, It's me) from Camelot and classical fans, the theme from Beethoven's "Eroica" (The Hero) symphony.

Lawrence Welk fans may simply pick up their Model III computer sideways (which then looks like an accordion), and do a Myron Florn imitation. (Anyone thinking that more than three minutes of accordion music is entertainment MUST be an egomaniac.)

By the way, if after reading this series of articles you still cannot grasp assembly but you have participated in the ego exercises, you can always write reviews. If you cannot do anything you can always become a computer salesperson, or if you are a complete idiot, you already have sufficient background to be either a computer consultant or a systems analyst. So fear not and forge ahead! The computing industry has a place for everybody.

If the computing industry is a "melting pot", just consider mother nature. The evidence is overwhelming that nature looks upon all her children equally - - as a quick meal; or why else is humanity nothing more than a protein bag stretched over many pounds of obscene glop.

To those who find it hard to believe in mankind as a quick lunch, I point to the so called loyalty of man's erstwhile "best friend", the family dog. If humanity is not a tasty morsel, then why do they always try to lick us?

Well now that the fun is over let's get to work. In the last issue we wrote the following:

```
00100           ORG     5200H           ;define entry
00110           LD      HL,HELLO        ;point to message
00120           CALL    4467H           ;print it
00130           JP      402DH           ;return to system
00140   HELLO   DB      1CH,lFH,'HELLO I AM YOUR TRS-80 COMPUTER',ODH
00150           END     5200H
```

To see it execute after assembly, at LDOS Ready merely type "PRINT" and the screen will clear and print the message defined in line 00140.

Assembler, or for that matter, any compiled language, has a tremendous disadvantage in not being capable of halting any run to examine for problems. For this reason, a class of "monitor" program such as DEBUG is very handy. Let's examine PRINT/CMD under the LDOS system DEBUGger. Switch on DEBUG by typing its name at LDOS Ready. Unlike most programs, DEBUG will not execute immediately. To remove other memory clutter, do a MEMORY (C), before entering DEBUG. After switching on DEBUG, it will activate mostly by executing a user (non-system) program or by pressing the <BREAK> key. To DEBUG PRINT/CMD, type "PRINT" at LDOS Ready. Do this before going farther (don't cheat either; we have ways of making you DEBUG).

Down the left side of the video are the twelve paired/16-bit registers commonly used in the Z-80. The next two columns after the equal sign display the contents of each eight bit register separately. The four lowest registers are normally not addressable by less than 16 bits at a time (but they can be "fooled", which we will get into a dozen or so articles from now). Since our PRINT program has just started, our code has affected only one register. Notice that the PC register contains 520O which points to the next instruction to fetch from memory and execute.

After the register contents is a "=>" followed by 16 bytes of information. This "=>" symbol shows what each pair "points" at in memory. You can now see the difference between the register contents and the pointed-to address. The PC register contains the value 5200 and points to the value 21 in memory. All of these values are in hex, by the way. If you look at the assembly listing from this program in the last issue, you will note that the assembler's first columns read like this: 5200 210952, which you recall means that there is a 21 at 5200, a 09 at 5201, and a 52 at 5202. Looking past the 21 that PC points to, you can see the remaining bytes of code in the same horizontal row.

Now I draw your attention to the last four rows of the video. These display any 64 bytes in memory. To see the rest of our program, type "D 5200<ENTER>". This points the lower "D"isplay to 5200 through 523F. If you cleared memory before entering DEBUG, you can readily see the entire PRINT program, all 43 bytes of it, ending with the OD at 522B. (If you did not clear memory, take out a flexible plastic object and smack yourself repeatedly across the knuckles for not following instructions. This simulates a quality education and tenderizes your hand for Rover.)

To proceed as if the code really were executing, DEBUG provides the "I" and the "C" commands. The difference is that "I" executes one instruction at a time but "C" will

perform a CALL instruction as a single step. This has an advantage. Recall that we CALL @DSPLY (4467H) in this program. We know that the system call involves all the code necessary to print our sentence to the video. We can bypass the tedium of this routine with the "C" instruction since we are only interested in our own code. It should also be noted that DEBUG will not single step through ROM, and that if a call to a ROM routine is made by the system, a "C" must be used.

Press the <C> once. (If pressed twice please revert to flexible plastic discipline unless, of course, you enjoy it in which case cut yourself off for a month.) Notice that the PC now contains 5203 which points to the next instruction, CD. Note that the HL register now contains 5209. Our first instruction has been executed. The assembler converted the label "HELLO" into the address 5209.

Press the <A> key. This turns the right side of the screen to an "A"SCII display. Now look at the HL register. The contents of HL points to the screen message. The two dots preceding the message are non-ASCII characters. Pressing the <H>, for hex, key will convert the display back to hex, although you may run under either mode. Press <C> again and the video will flash, the PC contains 5206 and is pointing to C3. Our system call to @DSPLY has been performed and control returned to DEBUG. The message cleared the screen, printed, and re-printed the DEBUG display so quickly that we could barely notice the effect. Using DEBUG to examine video results is, therefore, done on a theoretical level.

The last command we will examine for now is the "G" or Go command. Leaving DEBUG is accomplished by means of the Go command. We can exit by typing G 402D. This is a jump to the @EXIT system vector which also happens to be line 00130 of our PRINT program. Typing G alone wil1 cause the program to continue normal execution. In this case we conveniently are a1ready returning to LDOS Ready, so simply do a "G<ENTER>" and turn off the debugger by typing "DEBUG (N)<ENTER>" at LDOS Ready.

You can explore the rest of the debugger on your own. This cursory explanation was to show you the most used features in order to assist you for the first time. Other system monitors worthy of note which provide many features beyond DEBUG are: UltraMon and Macro-Mon.

Sorts Illustrated.

Now for some real fun. Did you ever wonder (wonder should be a trademark of Creative Use of Glue in Preposterous Bread Commercials, Inc.) why SORTS have such bizarre names such as Ripple, Bubble, and Shell. It probably is do to the "action" caused by the code in memory causing a ripple or bubble effect to occur. The shell effect is interesting because if you put your ear on a RAM chip while the shell sort is running, you will hear nothing and most likely burn your ear. Since the TRS-80 video is RAM memory mapped, the effects can be observed by sorting ONE character arrays directly in video.

We will examine both the bubble and shell sorts in TBA source code as well as assembler. Stay tuned for an announcement regarding the ripple sort.

The bubb1e sort takes a list of objects and compares them two at a time. If the order is incorrect, the two objects are exchanged. The next pair of objects is then examined and so on until the entire list has been examined. Obviously, it requires several passes, before the list is entirely correct. A flag is reset prior to every pass and set if any exchange occurs. The process is repeated until a pass is completed without the flag being set. This indicates that no exchanges took place and, therefore, the list is now ordered. The effect of this sort is that the "light" stuff drifts to the top and the "heavy" stuff sinks like a rock. The TBA listing is : (for those of you who do not have TBA, I will translate this one time that you may see the folly of not possessing it.)

I think you wi11 find it easier to fo11ow the 1ogic of the TBA code rather than the old fashioned straight BASIC. (and now a message from our sponsor ... TBA is available for $69.00 in the 5.1 version ... and now back to our program).

```
=FLAG%,LOOP%,TEMP%,LOOPl%
'Fill half screen with random data 33 to 127
     CLS                                    3 CLS
     FOR LOOP%=15360 TO 15872               4 FORLO%=15360T015872
     POKE LOOP%, RND(95)+32                 5 POKELO%,RND(95)+32
     NEXT LOOP%                             6 NEXTLO%
'Reset flag and scream through data
 swapping as necessary
@ORDER.DATA
     FLAG%=0                                9 FL%=0
     FOR LOOP%=15360 TO 15871               10 FORLO%=15360TO15871
          LOOP1%= LOOP%+1                   11 LP%=LO%+1
            IF PEEK(LOOP%)<=PEEK(LOOP1%)
             THEN @SKIP.IT                  12 IFPEEK(LO%)<=PEEK(LP%)THEN18
          TEMP%=PEEK(LOOP1%)                13 TE%=PEEK(LP%)
          POKE LOOP1%,PEEK(LOOP%)           14 POKELP%,PEEK(LO%)
          POKE LOOP%,TEMP%                  15 POKELO%,TE%
          FLAG%=1                           16 FL%=1
@SKIP.IT
     NEXT LOOP%                             18 NEXTLO%
'Loop and try until no swaps occur
       IF FLAG% THEN @ORDER.DATA            20 IFFL%THEN9
@LOOP: GOTO @LOOP                           21 GOT021
```

This program sorts 512 items and takes more time than drinking hot coffee even with a loose filling. Our assembler version will sort 1024 items in several seconds. First, we need to know another pseudo-op. This is EQU or equate. We will start the program at 5200H and the first four lines are

```
00100           ORG       5200H
00ll0   VIDEO   EQU       3C00H
00l20   @EXIT   EQU       402DH
00130   @KEY    EQU       0049H
```

The equate sets the symbol in the label zone equal to the value/expression specified. This means that we can now substitute the term for the number as in CALL @KEY, etc. Our first task is to get the "random" data to the screen. There are several complicated algorithms for accomplishing this but they are not necessary here. Someday if we write some stupid game it may be necessary to deal with it. The idea is to scream through memory and locate characters in the range of ASCII 33 through 127 and reject everything else. To do this we need the equivalent of IF ... THEN or a conditional branch.

The Z-80's F register contains flags used to indicate conditions that occurred in the last completed instruction. Some instructions do not affect flags at all. In the F register, bits 7, 6, 4, 2, 1, and 0 are used for flags. (Bits 3 and 5 are not used.) The conditions can be used in conjunction with CALL, RET (RETurn), JP, and JR (Jump Relative) to either GOSUB, RETURN, GOTO, or "kind of goto", on a conditional basis. CALL and JP were covered in the last session. RET is simply a RETURN from a subroutine. JR has no exact analog in BASIC except it is another form of JP. The advantage is that unlike a jump (JP 402DH), JR requires no address but the jump must be in a range from -126 to +129 (effectively) from the instruction. In other words, it must be close. As you will see from the code below, it is usually referenced by a label which the assembler converts to the relative jump location. This is like saying GOTO ten lines ago or GOTO fifteen lines ahead. It executes slower than a jump but takes 1 byte less of code. Its primary advantage is that unlike a JP, it does not need to have the modifying address changed if the code is relocated. However, it does not have all of the branch conditions available to it.

A conditional branch in assembler means "Execute the instruction only if this condition exists" to the Z-80. JP Z,402DH means GOTO address 402D if the Z flag is set. The conditions can either be met or excluded in that every flag has its opposite

conditional branch. JP NZ,402DH means GOTO address 402D if the Z flag is NOT set. We will discuss the flags in detail as we need to.

Briefly, they are as follows:

```
              C - NC  = Carry Flag Set/Reset (bit 0)
              PE - PO = Parity Set/Reset     (bit 2)
              Z - NZ  = Zero Set/Reset       (bit 6)
              M - P   = Sign Flag Set/Reset  (bit 7)
```

Now let's fill the screen (memory) with characters,

```
00l40         LD       HL,4000H          ;start anyplace, actually
00150         LD       DE,VIDEO          ;point DE to video RAM
00l60 GETCAR  LD       A,(HL)            ;get a byte into A from HL address
00l70         CP       '!'               ;char < !
```

The A register is used for most math functions. The CP (ComPare) instruction performs a subtraction which throws away the result but leaves the flags affected as if the subtraction took place. This is how A is tested for its contents. Let us assume a value X for the following chart of comparisons utilizing the instruction CP X:

Relationship:	Branch Flag	When to go
if A < X	C	then go on C flag
if A <= X	C or Z	then go on C (if less) or Z flag (if equal)
if A = X	Z	then go on Z
if A <> X	NZ	then go on NZ
if A >= X	NC	then go on NC
if A > X	NZ & NC	then go on both NZ and NC (this is tricky)

X can be another eight bit register, a number (0-255), (HL) or (IX/Y +d). The last we will discuss in the future. It can be seen that we want characters greater than or equal to an '!' (ASCII 33) and that if A >= !, then we want that character, but only if it is <= ASCII 127, so the remaining selection code is as follows:

```
00l80         JR       NC,TEST           ;if A >= ! then test further
00l90         INC      HL                ;point to next memory location (INC) functions
                                         ;like this : HL = HL + 1
00200         JR       GETCAR            ;Jump back to line 160 (GETCAR)
00210 TEST    CP       7FH               ;is char <= 127
00220         JR       C,OK              ;char < 127, go
00230         JR       Z,OK              ;char = 127, go
00240         INC      HL                ;as above
00250         JR       GETCAR
00260 OK      LD       (DE),A            ;put A into memory pointed to by DE
00270         INC      DE                ;point to next place to write
00280         BIT      6,D               ;test for end of screen
00290         JR       Z,SORT            ;if D=40H we are at end of video plus 1
00300         INC      HL                ;as above
00310         JR       GETCAR
```

It should be noted that when a LD A,(HL) is performed, that neither of the contents of the H nor the L register are placed in A. Rather the one byte contents of the ADDRESS pointed to in the HL pair is loaded into A. Similarly, the LD (DE),A does not affect D or E but does write the contents of A to the memory location pointed to in the DE pair. Note the use of conditional branches to hop around to the desired spots. Another test which can be performed on a register is the BIT test in line 00280 above. The video screen occupies memory from X'3C00' (15360) through X'3FFF' (16383). The upper limit of video ram looks like this 0011 1111 in D, and 1111 1111 in E. Adding 1 to this number yields X'4000' (16384) or 0100 0000 in D, with E containing all zeros. Therefore, when bit 6 of the D register is not zero (NZ) we no longer need to loop because the screen is full.

We can now sort the data as we did in lines 9 through 21 of the BASIC program.

```
00320 SORT    LD      HL,VIDEO        ;point HL to screen start
00330         LD      C,0             ;C register is our flag
00340 SORT1   LD      D,H             ;point DE at HL + 1
00350         LD      E,L
00360         INC     DE
00370         LD      A,(DE)          ;get DE's pointed character
00380         CP      (HL)            ;if (HL) <= A then skip it
00390         JR      NC,SKIPIT
00400         LD      B,A             ;this starts the exchange A has (DE) value
00410         LD      A,(HL)          ;xfer A to B and get (HL)
00420         LD      (DE),A          ;swap characters
00430         LD      (HL),B
00440         LD      C,l             ;a swap a swap, mark it!
00450 SKIPIT  INC     HL              ;advance
00460         BIT     6,H             ;check for finish
00470         JR      Z,SORT1
00480         INC     C               ;passover I mean pass over check for C being a
                                      ;zero which means no swap occurred on last pass
00490         DEC     C               ;this sets Z flag if C was zero but not if C was 1
00500         JR      NZ,SORT         ;try another pass
00510         CALL    @KEY            ;entry point which waits for a keystroke
00520         JP      @EXIT           ;back to LDOS Ready
00530         END     5200H
```

DEC is the opposite of INC, so that in line 00490 C = C - 1. Why this manipulation? The LD instructions do NOT affect the flags, but on single registers, if the result of a DEC is a Zero, then the Z flag is set. At the end of a pass C contains either one or zero. If a swap occurred, C is loaded with one. At the end of the pass, C is INCreased to two and DECreased back to one which does not set Z. Therefore, a JR NZ executes and that branches to the code which reloads C with zero and does another pass. If C was zero at the end of a pass, it would be INCreased to one and DECreased to zero which does set Z, and control falls through the JR NZ to the keyboard call.

BIT tests the argument (in this case the H register) to see whether or not a specific bit is set (1) or reset (0). The flag Z is set if a zero is detected at the specified bit (in this case 6). One CAUTION should be noted. The test for the end of a pass (BIT 6,H) is a little tacky. Since we point DE at HL + 1, when HL is 3FFFH, DE will already BE 4000H or one byte past the end of VIDEO RAM. We get away with this because X'4000' happens to contain a C3H which is above our upper limit so that it will never swap. If the program were modified to use characters through byte FF then a SWAP would occur on a vital system vector (a C3 is a jump) and an FF would end up there (an FF is a RST 38H) which would positively blow your buns away. PART ONE of the ASSEMBLY contest is to rewrite the assembly code so that this can never occur. Also in PART ONE, see if you can rewrite lines 2l0-230 to do the same job with one less compare.

The Shell sort takes a list of elernents and divides them into zones of ever increasing length. A list of l00 elements is divided by an arbitrary number. In advanced languages, this is usually not two but in assembler it makes little difference due to speed and is usually done to take advantage of the easy method of dividing by two in a binary base numbering system. You merely shift the number one place to the right, which is exactly what we do when dividing by ten in a decimal system.

The items in each shell are then placed in order and the list advances to each successive shell until the list is exhausted. The control number is then divided again which effectively increases the size of the shells and the process is repeated until the shell length equals the number of objects in the list. Since this might be a bit vague, let's look at some actual numbers. In a list of l00 we divide by 3 to get 32 shells of 3 items each and one of 4. The first shell is comprised of the first element plus all elements offset by 33 or the elements 1, 34, 67, and l00. The second shell contains elements 2,

35, and 68, shell three is composed of 3, 36, 69 etc. So that each of the shells has very few elements. Each shell is then ordered by comparing a pair of elements within it. 1 is compared to 34 and if necessary they are swapped. Then 2 is compared to 35 etc. until the list reaches 34. Now we are back to the first shell and compare 34 to 67. If no swap is necessary we also know that since 1 < 34 < 67 that shell 1 is in order within itself. If, however, item 67 is less than item 34 we swap them, but we still have no idea if 1 is less than that item (now at 34) so we compare 1 and 34 again, and if necessary swap them also.

We divide the number 33 by two and get 16, now there are 16 shells of length 6 or 7. These are already somewhat re-ordered by pass 1 and we are about 20% "zoned" at the end of this pass. In 2 passes we have accomplished more than on l00 passes in the bubble sort. As the sort progresses, you will note the migration of the elements toward their final resting place. (Kind of like life, isn't it?) The TBA code for the shell screen sort is as follows:

```
=OFFSET%,ELEMENTS%,LOOP%,TEST%,TEMP%
CLS
CLEAR100
FOR LOOP%= 15360 TO 15872
POKE LOOP%,RND(95)+32
NEXT LOOP%
ELEMENTS%=512
OFFSET%=INT(ELEMENTS%/3)
FOR LOOP% = 0 TO ELEMENTS%-OFFSET%
TEST%=LOOP%+15360
IF PEEK(TEST%) <= PEEK(TEST%+OFFSET%) THEN @ANOTHER
OFFSET%= INT(ELEMENTS%/2)
@GO:      FOR LOOP% = 0 TO ELEMENTS% - OFFSET%
TEST%=LOOP%+15360
@AGAIN:        IF PEEK(TEST%) <= PEEK(TEST%+OFFSET%) THEN @ANOTHER
TEMP%=PEEK(TEST%)
POKE TEST%,PEEK(TEST%+OFFSET%)
POKE TEST%+OFFSET%,TEMP%
TEST%=TEST%-OFFSET%
IF TEST%>15360 THEN @AGAIN
@ANOTHER: NEXT LOOP%
OFFSET%=INT(OFFSET%/2)
IF OFFSET%>0 THEN @GO
@DONE: GOTO @DONE
```

The assembler code is as follows

```
00090         TITLE   <'SHELL/ASM shell sort of video contents'>
00100         ORG     05200H
00ll0 VIDEO   EQU     03C00H
00120 @EXIT   EQU     0402DH
00130 @KEY    EQU     00049H
00140 START   LD      HL,4000H       ;random selection
00l50         LD      BC,l024        ;do l024 times
00160         LD      DE,VIDEO       ;point to screen
00180 SELECT  LD      A,(HL)         ;stuff contents in A
00190 TESTLO  CP      '!'            ;A >= !
00200         JR      NC,TESTHI      ;test upper limit
002l0 BUMP    INC     HL             ;A < i go
00220         JR      SELECT         ;try again
00230 TESTHI  CP      80H            ;check for > 127
00240         JR      C,OK           ;if not, proceed
00250         JR      BUMP           ;else try again
00260 OK      LD      (DE),A         ;put on screen
00270         DEC     BC             ;decrease count
```

```
00280         LD      A,B             ;test for end of loop
00290         OR      C               ;does C = B = 0?
00300         JR      Z, SORT         ;BC is 0, goto sort
003l0         INC     HL              ;else point to next char
00320         INC     DE              ;point to next video
00330         JR      SELECT          ;goto select
```

With EDAS, the TITLE pseudo-op places the string between the <> at the top of every page if you send the assembly listing to the printer. Rather than wait for DE to acquire a value of 4000H, this time we set up a definite loop in the BC register which DECs every time a character which meets our discriminating standards is selected. We test for BC = 0 in lines 280 and 290. OR performs a logical or operation of the argument (in this case the C register) against the accumulator. The result of this operation will be zero only when both B & C are zero. This must be done because DEC BC or the DEC of any 16 bit pair does not affect flags. Therefore, it is a way to determine when the 16 bit pair is zero.

```
00340 SORT    LD      BC,512          ;initial offset is 512
00350         LD      HL,VIDEO        ;point to CRT
00360 SORT1   PUSH    HL              ;save pointer in case we need to go back
00370         POP     IX              ;loads 16 bit IX through SP
00380         PUSH    HL              ;save it one more time
00390         ADD     HL,BC           ;make DE the next Shell member
```

Note that we did not convert the BC load number into any other number base. We knew we wanted 512 so the assembler will convert the number to hex for us. A PUSH instruction places the contents of a 16 bit register or register pair into the memory stack. I will not discuss the stack at all here. Suffice it to say that it places the contents in memory and knows where they are. A POP does exactly the opposite, retrieving the contents from the stack. The operation in lines 360-370 is the most common method of transferring a 16 bit pair (HL) to an index register. Effectively, after execution of line 370, IX = HL. HL may be used as a 16 bit "accumulator" for addition and subtraction. In this case BC was set to 512 (and will be decreased later) which makes HL = HL + 512 after execution of 390.

```
00400         LD      E,L
00410         LD      D,H
00420         BIT     6,H             ;test for end of pass
00430         POP     HL              ;restore current shell element to HL
00440         JR      NZ,DIVIDE       ;if end of pass goto divide
00460 COMP    LD      A,(DE)          ;compare HL to DE
00470         SUB     (HL)            ;if A < 0 swap
00480         JP      P,NEXT          ;if A >= 0 goto next
```

SUB takes the argument (in this case (HL)) and subtracts it from the accumulator and stores the result back in A (A = A - (HL)). In this case, because the original information is destroyed, it may be more desirable to use a CP, but we save one byte this way and I wanted to show you different methods of de-dermitizing the feline. Notice the JP P,NEXT. Although a JR would be in range, a JR on condition P is not allowed, so a JP had to be used.

```
00490         LD      A,(DE)          ;restore A for swap
00500         PUSH    AF              ;save element 1
00510         LD      A,(HL)          ;get element 2
00520         LD      (DE),A          ;& put at position 1
00530         POP     AF              ;restore element 1
00540         LD      (HL),A          ;& put at position 2
00550         LD      E,L             ;since a swap occurred, we must look
00560         LD      D,H             ;backward in the shell
00570         XOR     A               ;this always clears the C flag
00580         SBC     HL,BC           ;point HL to -offset
```

XOR does an exclusive OR logical operation on A. We will discuss logical operators next time. It is necessary in this case because the only 16 bit subtraction possible is an SBC which is SuBtract with Carry. This subtracts one from the result if the carry flag is set, which is necessary for multiple precision math. In this case, we do not want the carry bit set, so we make sure it is gone by blasting it with XOR.

```
00590         LD       A,H           ;we must be sure we are >= 3C00
00600         CP       03CH          ;if H < 0 then proceed
00610         JR       NC,COMP       ;use same code to check negative shell
00620 NEXT    PUSH     IX            ;restore HL to current element
00630         POP      HL
00640         INC      HL            ;point to next shell
00650         BIT      6,H           ;test for > 4000H
00660         JR       Z,SORT1       ;if not go again
00670 DIVIDE  SRL      B             ;divide BC offset by 2
00680         RR       C
```

The SRL instruction (Shift Right Logical) takes all eight bits of a register and bumps every one one space to the right. A zero is loaded into bit 7 and if there was a 1 at bit zero the carry flag is set. In this case, the instruction is used to divide the BC off set by two. The RR C instruction rotates all eight bits of register C to the right, and will store any carry from the previous SRL B instruction into bit seven of C. Any carry from C would be a fraction and we don't care about it. The effect is C = INT(C/2).

```
00690 NOCR    LD       A,B           ;if BC = 0 we are done
00700         OR       C
00710         LD       HL,VIDEO
00720         JR       NZ,SORT1      ;go back if BC <> 0
00730         CALL     @KEY          ;wait for keystroke
00740         JP       0402DH        ;return to LDOS
00750         END      START
```

Well that almost wraps it up. But wait, there's more. Announcing the first LET US ASSEMBLE CONTEST. Below is a TBA source for a Ripple Sort. Write the assembler version which works on the screen and answer part one described above. Submit both your answer to part one and the code for the ripple sort by September 1, 1983 to LSI attn: Ripple contest. All CORRECT replies will be placed in a hat and three winners will be drawn at random. Each of the winners will receive A FREE YEAR of extended support.

No purchase is necessary to enter (or is probably possible). All entrants must be alive at the time the code is written. Employees of cesspool cleaning firms are eiigible but must wash your hands before handling the envelope.

Ripple Sort TBA listing :

```
=FLAG%,LOOP%,TEMP%,LOOPl%
CLS
FOR LOOP%=15360 TO 15872
POKE LOOP%, RND(95)+32
NEXT LOOP%
FOR LOOP%=15360 TO 15871
FOR LOOP1%= LOOP%+1 TO 15872
IF PEEK(LOOP%)<=PEEK(LOOP1%) THEN @SKIP.IT
TEMP%=PEEK(LOOP1%)
POKE LOOP1%,PEEK(LOOP%)
POKE LOOP%,TEMP%
@SKIP.IT
NEXT LOOP1%,LOOP%
@LOOP: GOTO @LOOP
```

LDOS: HOW IT WORKS - Non-Radio Shack disk drives

Creating system disks, booting double-sided discussed
or--- How to make a directory sandwich.
by Joseph J. Kyle-DiPietropaolo

This quarter's topic is one of constant demand, mainly due to the proliferation of non - standard hardware out there. Those of you with 80 track, or double-sided (DS) disk drives wil1 find this in formation of tremendous interest. Anyone with "normal" hardware out there? You should come along for the ride anyway. There wi11 be some important information given that isn't available anywhere else.

First of all, anyone who doesn't have at least one forty track disk drive will be wacked over the knuckles with this ruler I borrowed from Rich. Forty track is **THE** standard for distribution, and not having a forty track drive will cause you no end of headaches right up until the day you break down and buy one. For those unfortunate souls that don't own a forty track drive, LSI can provide most of its products on eighty track single-sided media for a modest additional charge. Also, on the LSI Utility Disk #1, there is a program that will allow READING of a forty track disk on a eighty track drive, within certain limits.

The first step in the use of double-sided disk drives is to get them communicating with your computer/interface and LDOS. A good discussion of the hardware requirements is in the LDOS Quarterly, Vol. 1 Number 2 (which has been reprinted in The Anthology, Volume One). As a quick summary, the drive should be configured as a single physical drive, with side select on pin 32. A continuous cable should be used, with each drive programmed to its proper drive select. If you have any questions, contact your drive vendor to get the information for your particular drive.

Now, to the heart of the matter. How do we create a optimized system disk in a 80 track or double-sided configuration?

Step 1) Get a hardcopy directory of your system disk with all files and allocation information (use the a,s,i,p parameters).

Step 2) Format a clean, new diskette with the appropriate density, number of sides, and tracks. Now, consult this chart to determine the number of sectors per cylinder, and granules per cylinder for your diskette.

Size	Density	Sides	Sectors	Granules	Size	Density	Sides	Sectors	Granules
5"	sgl	1	10	2	8"	sgl	1	16	2
5"	dbl	1	18	3	8"	dbl	1	30	3
5"	sgl	2	20	4	8"	sgl	2	32	4
5"	dbl	2	36	6	8"	dbl	2	60	6

Now, we need to calculate how many full cylinders will be needed to hold SYS0 and SYS6. To use the MAX-80 as an example, on that machine SYS0 requires 81 sectors. On a 5" Dden 2-sided disk, that would be 14 granules. SYS6 requires 53 sectors, or 9 granules. Together that would be 25 granules, or almost five cylinders.

Step 3) Use the patch from the last Quarterly (to SYS8, by the way) to determine where files will be allocated on the disk. For convenience, I have reprinted the patch at the end of this article. We want to put SYS0 and SYS6 immediately below the directory, so we patch SYS8 to place them there for us. In our example, we would patch SYS8 to start allocation at cylinder 15, allowing five full cylinders. Now, backup SYS0 from your system disk to your new diskette. A typical command would be:

backup SYS0:0 :1 (s)

By following this procedure, SYS0 will be stored in one extent, starting at the beginning of a cylinder, and starting on the 'front' of the diskette. This will ensure that the

resulting diskette will boot on your system, if your system will handle the density involved. Model I users with SOLE, use the diskette processed by SOLE1 starting with Step 2, create the sysgen file after Step 7, and SOLE2 after Step 8.

Step 4) Now backup SYS6 to the new diskette. It will be placed immediately below the directory. The remainder of the system files other than SYS7 will require 10 granules Calculate how many cylinders will be necessary to hold these files, and patch SYS8 to start allocating that far above the directory. For our example, that would be two cylinders, and we would tell SYS8 to start allocating at cylinder 23 (17 hex), leaving 21 and 22 free for the other system files. You can also check the free space map of the diskette, and see if there are any free granules immediately below the directory left over after SYS6. If so, these can be used for the other system files, and can be subtracted from the ten needed. Now, backup SYS7.

Step 5) We now have SYS7 above the directory, with enough blank space in-between to hold all of the other system files. At this point, we're most of the way home. Re-patch SYS8 with the cylinder number from Step 3, and backup the remainder of the system files. A typical command would be:

backup /sys:0 :1 (new,s)

Step 6) Now backup everything else from your system disk. Use a command like:

backup :0 :1 (new)

Model I users will then want to transfer control to their new system disk (see the 'system (system=' command), and backup the files from their LDOSXTRA disk to their new system disk.

Step 7) Re-patch SYS8 back to normal, or to the desired allocation on both your old and new system disks.

That's it! Now, that wasn't so painful, was it?

For your convenience, here is the SYS8 patch information reprinted from the last Quarterly:

```
.  Optional patch to SYS8/SYS
.  This patch allows for arbitrary allocation, as opposed to random
.  The patch is the same for Model I, Model 3, and MAX-80, for LDOS 5.1.3 ONLY
.
D00,FE=2E 01 00 00 00 00
.  This (the 01) is the cylinder number (hex) that allocation will start at
.
.  To reverse the patch, the original code in SYS8 was:
.  D00,FE=D5 CD 4E 44 D1 6C
.
.  end of patch
```

As an example, here's the format of a patch to start allocation at cylinder 15:
PATCH SYS8/SYS.SYSTEM (D00,FE=2E OF 00 00 00 00)

LES INFORMATION

by Les Mikesell

The differences between device and file access under the 5.1 version of LDOS have been discussed in earlier issues. The status in the Z flag is different for requests through @GET. A character will be input from a file with the Z flag set if there are no errors, while @GET returns with an NZ condition if the request goes to a device driver. If the

ROM keyboard driver is used instead of the LDOS KI/DVR, the status flag is unspecified, and it is necessary to test for a non-null character.

Under LDOS 6.0 (alias TRSDOS 6.0) the status conditions returned by @GET and @PUT requests are handled identically for files and devices. This simplifies the task of dealing with I/O devices which may be ROUTED to files by the operating system, and allows programs to use the same routines with either devices or files if a choice is given at run-time. The routines using these functions should be written to handle the error conditions that might be returned by either a device or file access.

For @PUT, the Z flag will be set after a successful completion. If the function returns with NZ status, the error number will be in the A register. Possible errors from a @PUT request might include "device not available" from a timeout on the printer ready test, or any of the file access errors. If it is possible to recover from an output error, a program should take the appropriate steps when the error is detected.

The status for @GET is similar: returning with the Z flag set indicates a successful input, and the input character will be in the A register. If the Z flag is reset, the value in the A register indicates the type of error. However, a very likely possibility is that the error number will be 0, indicating "no error". This simply means that the device did not have a character available to return at the time of the input request. Thus, the indication is that since the Z flag is reset, the input request was not successful, but there was actually no error involved. A program would normally loop repeating the @GET request until a character is received, if the input is necessary before further processing.

Examples of @GET and @PUT for 6.0

```
First a file or device must be OPENed using the defined FCB
FCB     DS      32
"
.. then to output a character...
"
        LD      DE,FCB   ;point to OPEN FCB
"
        LD      C,CHAR   ;put character in C
        LD      A,4      ;@PUT SVC number
        RST     40       ;do it
        JP      NZ,ERROR ;process error (or quit)
..continue..

.. to input a character...
        LD      DE,FCB
LOOP:   LD      A,3      ;@GET SVC number
        RST     40       ;do it
        JR      Z,GOTCHR ;continue if char rcvld
        OR      A        ;otherwise test for error
        JR      Z,LOOP   ;wait for some input
        CP      1CH      ;End of file error?
        JP      Z,EOF    ;to completion handler..
.. could test for other specific errors here..
.. depending on the program..
        JP       ERROR   ;to fatal error exit..
GOTCHR  .... normal continuation of program
```

One built-in function that is available on many other systems but not the TRS80's is a simple method of temporarily linking the video output to the printer. The lack of this function may stern from the incompatibility between the TRS80 video and printer control codes. The following filters will provide this ability for both 5.1 and 6.0 versions.

The program actually filters both the *KI and *DO devices, using the presence of a control-P (shift-down arrow P on the Model 1 & 3) detected during a keyboard input to toggle the link between the video and printer on and off. When the toggle is on, each character sent to the video is also sent to the printer, except for those values that are likely to cause strange effects. The SYSTEM (GRAPHIC)-bit is tested to determine if the characters above X'7F' should be sent or converted to a period, as in the screen-print function. A typical use of the function would be to obtain a listing of the output from a program that normally displays only on the video. Just before pressing <ENTER> on the command that invokes the listing, enter the control-P character. Enter the control-P again to terminate the listing any time the system is requesting keyboard input.

After assembling the program with the name CTLP/FLT, the 5.1 version of the filter would be installed with the command: FILTER *KI CTLP

The 6.0 version requires two steps to install, since filters take their own device name and DCB:

```
SET *CP CTLP
FILTER *KI CTLP
```

The 6.0 version filters the video by modifying the driver address in the DCB, like the 5.1 version instead of creating an additional DCB for the video portion of the filter also. This method requires that no other filtering, routing, or linking be done to the *DO device before loading CTLP/FLT.

The assembly language listings follow, starting on Page 59.

THE JCL CORNER - By Chuck

There will be no JCL Corner this issue, as the JCL Corner is currently undergoing a change in format. Because all the basics for JCL have been covered, in the future, it will be going to a question and answer format. Question and answer columns covering advanced subjects will alternate with reprints of earlier introductory information.

Because of the late mailing of the last Quarterly, the winners of the JCL Corner question competition will not be announced until the next Quarterly.

So, until next time.....

//EXIT

```
00100 ;Filter for LDOS 5.1.x to allow toggling a link
00110 ; *DO *PR by typing CTL-P any time the
00120 ; keyboard driver is called
00130 ;
00140 TGLCHR EQU 'P'-40H        ;ctl-P
00150 ;
00160 ; Hardware dependant EQUates..   Mod 1 addresses used
00170 HIGH3   EQU     4411H
00180 PARAM3  EQU     4454H
00190 HIGH$   EQU     4049H
00200 @PARAM  EQU     4476H
00210 @LOGOT  EQU     447BH
00220 LOGOT3  EQU     428AH
00230 KIJCL$L EQU     43BEH
00240 KIJCL$3 EQU     42BEH
00250 SFLAG$L EQU     430FH
00260 SFLAG$3 EQU     442BH
00270 DFLAG$L EQU     441FH
00280 DFLAG$3 EQU     4289H
00290 ;
00300 ; General EQUates...
00310 KDCB$   EQU     4015H
00320 DODCB$  EQU     401DH
00330 @EXIT   EQU     402DH
00340 @ABORT  EQU     4030H
00350 @DSPLY  EQU     4467H
00360 @PRT    EQU     3BH
00370 LF      EQU     l0      ;linefeed character
00380 CR      EQU     13      ;carriage return
00390 ;
00400 ; LDOS 'FILTER'  command  handler
00410 ;
00420         ORG     5200H
00430 LNTRY   PUSH    DE              ;put DCB pointer
00440         POP     IX              ;into IX register
00450         PUSH    HL              ;save buff pointer
00460         CALL    MODEL
00470         LD      A,(DE)          ;Get DCB type byte
00480         PUSH    AF
00490         LD      HL,SIGNON       ;=>Signon message
00500         CALL    @DSPLY          ;print it
00510         POP     AF              ;restore type byte
00520         BIT     3,A             ;Device routed NIL?
00530         JP      NZ,ISNIL        ;Go if so
00540         BIT     4,A             ;Routed?
00550         JP      NZ,ROUTED       ;Go if so (error)
00560         AND     1               ;dvr handle input?
00570         CP      1
00580         JP      NZ,DEVERR       ;Go if not
00590         POP     HL              ;rest buff pointer
00600         LD      DE,PRMTBL       ;Scan parameters
00610         CALL    @PARAM
00620 MMOD4   EQU     $-2
00630         JP      NZ,PRMERR       ;quit if error
00640 ;test parameter value and initialize filter
00650         LD      BC,$-$          ;value set by @PARAM
00660 CHARP   EQU     $-2             ;<=here
00670         LD      A,C
00680         OR      A               ;set flag
00690         JR      Z,GETDVR        ;go if not specified
00700         LD      (TGL),A         ;stuff byte if wanted
00710 ;allow installation to *KI in JCL
00720 GETDVR  PUSH    IX              ;If the DCB is for *KI,
00730         POP     DE              ; then use KIJCL saved
00740         LD      HL,KDCB$        ; vector for hooks
00750         XOR     A               ;clear carry
00760         SBC     HL,DE           ;Zero if *KI
00770         LD      L,(IX+l)        ;P/u DCB vector address
00780         LD      H,(IX+2)
00790         JR      NZ,SETADD       ;use if not *KI
00800         LD      A,(SFLAG$l)     ;Is DO in effect?
00810 SFLAG   EQU     $-2
00820         AND     20H
00830         JR      Z,SETADD        ;Not KIJCL if no DO
00840         LD      HL,(KIJCL$l)    ;P/u JCL saved vector
00850 KJCL    EQU     $-2
00860         LD      (WASKI+l),A     ;Note for later
00870 ; put previous driver addresses into filter
00880 SETADD  LD      (DVRADD),HL     ;put in filter
00890         LD      (DVRl),HL
00900         LD      HL,(DODCB$+l)   ;get *DO dvr addr
00910         LD      (DODVR),HL      ;put in filter
00920 ; find available high memory
00930         LD      HL,(HIGH$)      ;find top of memory
00940 MMOD1   EQU     $-2
00950         LD      (OLDMEM),HL     ;save in flt header
```

```
00960         PUSH    HL              ;save
00970 ; relocate JP and CALL addresses
00980         LD      DE,LAST         ;end of code now..
00990         OR      A               ;clr carry
01000         SBC     HL,DE           ;offset of move
01010         EX      DE,HL           ;into DE
01020         LD      HL,(REL1)       ;fix absolute memory
01030         ADD     HL,DE           ;references
01040         LD      (REL1),HL       ;in filter
01050         LD      HL,(REL2)
01060         ADD     HL,DE
01070         LD      (REL2),HL
01080         LD      HL,(REL3)
01090         ADD     HL,DE
01100         LD      (REL3),HL
01110 ; move into high memory
01120         LD      HL,LAST       ;end of relocated code
01130         POP     DE              ;old HIGH$
01140         LD      BC,LAST-FENTRY+l  ;len of rel code
01150         LDDR                    ;move it
01160         EX      DE,HL
01170         LD      (HIGH$),HL      ;set new HIGH$
01180 MMOD2   EQU     $-2
01190         INC     HL              ;pnt to flt entry
01200 ; install new addresses in DCBs
01210         DI                    ;Off while DCB update
01220 WASKI   LD      A,0           ;If DCB was for KI &
01230         OR      A             ; DO was in effect,
01240         JR      Z,WASKIL      ; then update KIJCL
01250         LD      (KIJCL$l),HL
01260 KJCL2   EQU     $-2
01270         JR      WASKI2
01280 WASKI1  LD      (IX+l),L        ;set new addr in DCB
01290         LD      (IX+2),H
01300         LD      DE,DOEPT-FENTRY ;offset to DO filter
01310         ADD     HL,DE           ;calc new address
01320         LD      (DODCB$+l),HL   ;for video DCB
01330 WASKI2  EI
01340 ;*=*=*
01350 EXIT    JP      @EXIT           ;Done
01360 ;*=*=*
01370 ;       Error handling
01380 ;*--*=*
01390 ISNIL   LD      HL,ISNIL$
01400         JR      ERROUT
01410 DEVERR  LD      HL,DEVER$
01420         JR      ERROUT
01430 ROUTED  LD      HL,ROUTD$
01440         JR      ERROUT
01450 PRMERR  LD      HL,PRMER$       ;'Parameter error'
01460 ERROUT  CALL    @LOGOT          ;Display and log
01470 MMOD3   EQU     $-2
01480         JP      @ABORT          ;Quit
01490 ;*=*=*
01500 MODEL   LD      A,(125H)
01510         CP      'I'
01520         RET     NZ
01530         LD      HL,HIGH3
01540         LD      (MMOD1),HL
01550         LD      (MMOD2),HL
01560         LD      HL,LOGOT3
01570         LD      (MM003),HL
01580         LD      HL,PARAM3
01590         LD      (MMOD4),HL
01600         LD      HL,KIJCL$3
01610         LD      (KJCL),HL
01620         LD      (KJCL2),HL
01630         LD      HL,SFLAG$3
01640         LD      (SFLAG),HL
01650         LD      HL,(DFLAG$3)
01660         LD      (DFLAG),HL
01670         RET
01680 ;*=*=*
01690 ;       Data area
01700 ;*=*=*
01710 SIGNON  DB 'CTLP/FLT - LDOS printer control filter'
01720         DB LF,CR
01730 PRMER$  DB 'Parameter error!',CR
01740 ISNIL$  DB 'Device not active!',CR
01750 DEVER$  DB 'Incorrect device type!',CR
01760 ROUTD$  DB 'Device is routed!',CR
01770 ;
01780 PRMTBL  DB 'CHAR  '
01790         DW CHARP
01800         DB 'C     '
01810         DW CHARP
```

```
01820         DW      0       ;end of list
01830 ;
01840 ;*=*=*
01850 ;       Actual filter moved to high memory
01860 ;       LDOS style header...
01870 ;*=*=*
01880 FENTRY  JR      START           ;Branch over linkage
01890         DW      $-$             ;Last byte used
01900 OLDMEM  EQU     $-2             ;prev HIGH$ value
01910 ;
01920         DB      4,'CTLP'
01930 TOGGLE  DB      0               ;on/off switch
01940 ;
01950 ; actual filter routine
01960 START   JP      NC,$-$  ;go if not input request
01970 DVRL    EQU     $-2     ;old driver address
01980         CALL    $-$     ;otherwise ca]] old driver
01990 DVRADD  EQU     $-2     ;stuff dvr addr here also
02000 ;
02010         PUSH    AF      ;save char/status
02020         CP      TGLCHR  ;is it a toggle?
02030 TGL     EQU     $-1
02040         JR      NZ,DONE ;go if not
020S0         LD      HL,TOGGLE       ;get current state
02060 REL1    EQU     $-2
02070         LD      A,(HL)
02080         CPL                     ;reverse it
02090         LD      (HL),A          ;and save
02100         POP     AF              ;dump toggle from
02110         XOR     A               ;input stream
02120         RET
02130 ;
02140 DONE    POP     AF              ;restore char/status
02150         RET                     ;and return it
02160 ;
02170 ;This code is actually a filter for *DO
02180 DOEPT   JR      Z,ISOUT         ;only affect output
02190 DOPUT   JP      $-$             ;previous *DO driver
02200 DODVR   EQU     $-2
02210 DOOUT   XOR     A               ;Set Z for output
02220         JR      DOPUT           ;go to driver
02230 ;test if link wanted..
02240 ISOUT   LD      A,(TOGGLE)      ;*PR active now?
02250 REL2    EQU     $-2             ;relocate address
02260         OR      A
02270         JR      Z,DOPUT         ;go if toggle OFF
02280 ;send to both *DO and *PR
02290 ;but test first for video control codes...
02300 ;that may do strange things to a printer...
02310         LD      A,C             ;check char..
02320         CP      20H             ;for ctl range
02330         JR      NC,DUAL         ;go if printable
02340         CP      0DH+1           ;allow CR/LF/FF/BKSP
02350         JR      NC,DOOUT        ;avoid 0EH -19H
02360 DUAL    PUSH    BC              ;save char in C
02370         CALL    DOOUT           ;CALL video output
02380 REL3    EQU     $-2
02390         POP     BC              ;restore character
02400         PUSH    BC              ;save again
02410         LD      A,LF            ;is character..
02420         CP      C               ;a line-feed?
02430         LD      A,CR            ;make carriage ret
02440         JR      Z,PRCHAR        ;if so.
02450         BIT     7,C             ;GRAPHIC char?
02460         JR      Z,NOTGRP        ;go if not
02470         LD      A,(DFLAG$l)     ;graphics ok?
02480 DFLAG   EQU     $-2
02490         RLCA                    ;quick bit 7 test
02500         LD      A,'.'           ;change to a period
02510         JR      NC,PRCHAR       ;if GRAPHIC off
02520 NOTGRP  LD      A,C             ;get character in A
02530 PRCHAR  CALL    @PRT            ;send to current *PR
02540         POP     BC              ;then rest original
02550         LD      A,C             ;char in case calling
02560         CP      A               ;program needs it
02570         RET
02580 ;
02590 LAST    EQU     $-1    ;used for length calculation
02600 ;
02610         END     ENTRY
```

```
00100 ;Filter for LDOS 6.x to allow toggling a link
00110 ; *DO *PR by typing CTL-P any time the
00120 ; keyboard driver is called
```

```
00130 ;
00140 TGLCHR  EQU 'P'-40H     ;ctl-P
00150 ;
00160 ; SVC number equates
00170 @HIGH$  EQU     100
00180 @PARAM  EQU     17
00190 @LOGOT  EQU     12
00200 @DSPLY  EQU     10
00210 @PRT    EQU     6
00220 @CHNIO  EQU     20
00230 @FLAGS  EQU     101
00240 @GTMOD  EQU     83
00250 @GTDCB  EQU     82
00260 ;flag table offset values
00270 DFLAG$  EQU     3
00280 ;
00290 ; General EQUates...
00300 LF      EQU     10      ;linefeed character
00310 CR      EQU     13      ;carriage return
00320 ;
00330 ; LDOS 'FILTER' command handler for 6.0
00340 ; The filter is first installed with the 'SET'
00350 ; command and takes its own DCB
00360 ; Then it is connected to the desired device with
00370 ; the 'FILTER' command
00380 ;
00390         ORG     3000H
00400 ENTRY   PUSH    DE              ;put DCB pointer
00410         POP     IX              ;into IX register
00420         PUSH    HL              ;save buff pointer
00430         LD      HL,SIGNON       ;=>Signon message
00440         LD      A,@DSPLY
00450         RST     40              ;print it
00460         LD      A,@FLAGS        ;pnt IY to flg table
00470         RST     40
00480         PUSH    IY
00490         POP     HL              ;base address to HL
00500         LD      DE,OFLAG$       ;ofst to DFLAG$ addr
00510         ADD     HL,DE           ;point there
00520         LD      (DFLAG),HL      ;put addr in filter
00530         POP     HL              ;=>command line
00540         BIT     3,(IY+2)        ;System request?
00550         JP      Z,NOTSET        ;must inst] w/SET
00560         LD      DE,PRMTBL       ;Scan parameters
00570         LD      A,@PARAM
00580         RST     40
00590         JP      NZ,PRMERR       ;quit if error
00600 ;test parameter value and initialize filter
00610         LD      BC,$-$          ;val set by @PARAM
00620 CHARP   EQU     $-2             ;<=here
00630         LD      A,C
00640         OR      A               ;set flag
00650         JR      Z,CHKDVR        ;go if not specified
00660         LD      (TGL),A         ;stuff byte if wanted
00670 ; see if filter is already installed
00680 CHKDVR  LD      DE,FLTNAM       ;=>CTLP
00690         LD      A,@GTMOD        ;lk up module header
00700         RST     40
00710         JP      Z,ISRES         ;already loaded
00720 ; check if memory available
00730         BIT     0,(IY+2)        ;HIGH$ frozen?
00740         JP      NZ,NOROOM       ;quit if so
00750 ; hook into video driver directly but first check
00760 ; that it is not linked or routed or filtered
00770 GETDVR  LD      E,'D'
00780         LD      D,'O'           ;DO name to DE
00790         LD      A,@GTDCB        ;find the DCB
00800         RST     40
00810         JP      NZ,DOERR        ;error if not found
00820         LD      A,(HL)          ;check TYPE byte
00830         AND     01110000B       ;filter/route/link??
00840         JP      NZ,CANT         ;all illegal
00850         INC     HL              ;=>driver LSB
00860         LD      (DODCB),HL      ;save addr of DCB+L
00870         LD      E,(HL)          ;get *DO driver
00880         INC     HL              ;address
00890         LD      D,(HL)          ;in DE
00900         LD      (DODVR),DE      ;put into filter
01000 ;
00920 ; find available high memory
00930         LD      A,@HIGH$        ;get HIGH$ into HL
00940         LD      HL,0
00950         LD      B,L             ;B=0
00960         RST     40              ;get top of memory
00970         LD      (OLDMEM),HL     ;save in flt header
00980         PUSH    HL              ;save HIGH$
```

```
00990         LD      (DCBADD),IX     ;put DCB addr in hdr
01000 ; relocate JP, CALL and LD addresses in filter
01010         LD      DE,LAST         ;end of code now..
01020         OR      A               ;clear carry flag
01030         SBC     HL,DE           ;offset of move
01040         EX      DE,HL           ;into DE
01050         LD      HL,(REL1)       ;fix absolute memory
01060         ADD     HL,DE           ;references
01070         LD      (REL1),HL       ;in filter
01080         LD      HL,(REL2)
01090         ADD     HL,DE
01100         LD      (REL2),HL
01110         LD      HL,(REL3)
01120         ADD     HL,DE
01130         LD      (REL3),HL
01140         LD      HL,(REL4)
01150         ADD     HL,DE
01160         LD      (REL4),HL
01170 ; move into high memory
01180         LD      HL,LAST         ;end of rel code
01190         POP     DE           ;old HIGH$=destination
01200         LD      BC,LAST-FENTRY+l  ;len of rel code
01210         LDDR                ;move it, leaving DE..
01220         EX      DE,HL           ;=>new HIGH$
01230         PUSH    HL
01240         LD      A,@HIGH$        ;set new HIGH$
01250         RST     40              ;note: B=0
01260         POP     HL
01270         INC     HL              ;pnt to flt entry
01280 ; set up new in DCB
01290         LD     (IX+0),01000111B ;filter/getputct]
01300         LD     (IX+l),L         ;set new addr in   DCB
01310         LD     (IX+2),H         ;for new Device/flt
01320 ; hook into video driver
01330         LD      DE,DOEPT-FENTRY ;offset to DO filter
01340         ADD     HL,DE           ;calc new address
01350         EX      DE,HL           ;move to DE
01360         DI
01370         LD      HL,(DODCB)      ;=>*DO DCB+L
01380         LD      (HL),E
01390         INC     HL
01400         LD      (HL),D          ;repl driver address
01410         EI
01420 ;*=*=*
01430         LD        HL,0          ;indicate no error
01440         RET
01450 ;*=*=*
01460 ;       Error handling
01470 ;*=*=*
01480 NOROOM  LD        HL,NOROOM$
01490         JR        ERROUT
01500 ISRES   LD        HL,ISRES$
01510         JR        ERROUT
01520 NOTSET  LD        HL,NTSET$
01530         JR        ERROUT
01540 DOERR   LD        HL,DOERR$
01550         JR        ERROUT
01560 CANT    LD        HL,CANT$
01570         JR        ERROUT
01580 PRMERR  LD        HL,PRMER$     ;'Parameter error'
01590 ERROUT  LD        A,@LOGOT      ;Display and log
01600         RST       40
01610         LD        HL,0FFFFH     ;indicate error
01620         RET
01630 ;*=*=*
01640 ;*=*=*
01650 ;       Data area
01660 ;*=*=*
01670 DODCB   DW   0                  ;store DCB ptr
01680 SIGNON  DB 'CTLP/FLT - LDOS printer control filter'
01690         DB LF,CR
01700 PRMER$  DB 'Parameter error!',CR
01710 DOERR$  DB 'Cannot find *DO DCB',CR
01720 NTSET$  DB 'Must install with SET command',CR
01730 CANT$   DB 'Display device is routed, linked,'
01740         DB ' or filtered.',CR
01750 ISRES$  DB 'Filter already resident',CR
01760 NOROOM$ DB 'No memory available',CR
01770 ;
01780 PRMTBL  DB 'CHAR  '
01790         DW  CHARP
01800         DB 'C     '
01810         DW  CHARP
01820         DW      0       ;end of list
01830 ;
01840 FLTNAM  DB      'CTLP',3 ;name terminated for @GTMOD
```

```
01850 ;*=*=*
01860 ;       Actual filter moved to high memory
01870 ;       LDOS style header...
01880 ;*=*=*
01890 FENTRY  JR      START        ;Branch around linkage
01990         DW      $-$             ;Last byte used
01910 OLDMEM  EQU     $-2             ;prev HIGH$ value
01920 ;
01930         DB      4,'CTLP'
01940 DCBADD  DW      $-$             ;DCB using filter
019S0 SPARE   DW      0
01960 TOGGLE  DB      0               ;on/off switch
01970 ;
01980 ; actual filter routine (looks for toggle)
01990 START   LD      IX,(DCBADD)     ;get DCB addr to IX
02000 REL4    EQU     $-2             ;relocate address
02010         LD      A,@CHNIO        ;cont down dev chain
02020         JP      NC,40           ;dn't come back here
02030         RST     40              ;unless input req
02040         RET     NZ              ;back if no char
02050 ;check  input character for toggle value
02060         PUSH    AF              ;save char/status
02070         CP      TGLCHR          ;is it a toggle?
02080 TGL     EQU     $-1
02090         JR      NZ,DONE ;go if not
02100         LD      HL,TOGGLE       ;=> switch byte
02110 REL1    EQU     $-2
02120         LD      A,(HL)          ;pick it up
02130         CPL                     ;reverse it
02140         LD      (HL),A          ;and save
02150         POP     AF              ;discard toggle from
02160         OR      0FFH            ;input stream
02170         LD      A,0             ;by ret 0 w/NZ stat
02180         RET
02190
02200 DONE    POP     AF              ;restore char/status
02210         RET                     ;and return it
02220 ;
02230 ;This code is actually a filter for *DO
02240 DOEPT   JR      Z,ISOUT         ;only affect output
02250 DOPUT   JP      $-$             ;previous *DO driver
02260 DODVR   EQU     $-2
02270 DOOUT   LD      A,C             ;Set Z for output
02280         CP      A
02290         JR      DOPUT           ;go to driver
02300 ;test if link wanted..
02310 ISOUT   LD      A,(TOGGLE)      ;*PR active now?
02320 REL2    EQU     $-2             ;relocate address
02330         OR      A
02340         JR      Z,DOOUT         ;go if toggle OFF
02350 ;send to both *DO and *PR
02360 ;but test first for video control codes...
02370 ;that may do strange things to a printer...
02380         LD      A,C             ;check char..
02390         CP      20H             ;for ctl range
02400         JR      NC,DUAL         ;go if printable
02410         CP      0DH+1           ;allow CR/LF/FF/BKSP
02420         JR      NC,DOOUT        ;avoid 0EH -19H
02430 DUAL    PUSH    BC              ;save char in C
02440         CALL    DOOUT           ;CALL video output
02450 REL3    EQU     $-2             ;relocated address
02460         POP     BC              ;restore character
02470         PUSH    BC              ;save again
02480         LD      A,LF            ;is character..
02490         CP      C               ;a line-feed?
02500         LD      A,CR            ;make carriage ret
02510         JR      Z,PRCHAR        ;if so.
02520         BIT     7,C             ;GRAPHIC char?
02530         JR      Z,NOTGRP        ;go if not
02540         LD      A,($-$)         ;graphics ok?
02550 DFLAG   EQU     $-2           ;fnd from flg tbl indx
02560         RLCA                    ;quick bit 7 test
02570         LD      A,'.'           ;change to a period
02580         JR      NC,PRCHAR       ;go if GRAPHIC off
02590 NOTGRP  LD      A,C             ;get character in A
02600 PRCHAR  LD      C,A             ;send to current *PR
02610         LD      A,@PRT
02620         RST     40              ;out to PR
02630         POP     BC              ;then rest original
02640         LD      A,C             ;char if calling
02650         RET
02660 ;
02670 LAST    EQU     $-1   ;used for length calculation
02680 ;
02690         END     ENTRY
```