LDOS QUARTERLY

April 1, 1983 Volume 2, Number 2

In this issue:

New Headquarters:

LOGICAL SYSTEMS INC.

8970 N. 55th Street
P.O. Box 23956
Milwaukee, WI 53223
(414) 355-5454

Table Of Contents

8970 N. 55th St, Milwaukee, WI 53223
(414) 355-5454

All Mail, UPS and phone calls to Logical Systems should be addressed as follows:

LOGICAL SYSTEMS, INC.
8970 N 55th St.
P.O. Box 23956
Milwaukee, WI 53223

414/355-5454

NORMAL HOURS OF BUSINESS

Monday thru Friday
9:00am thru 5:00pm
Central Time Zone

***** HAVE YOU MOVED *****

If you have moved, be sure to notify us at least 30 days prior to the effective date of your address change to insure that you do not miss any of the valuable issues of the quarterly, or any other important notices which may be sent to you. Include you Serial # with this information to expedite the changes to our files.

LOGICAL SYSTEMS DEPARTMENTS

* Contract Sales Department
* Customer Service Department
* LDOS Quarterly Department
* Order Processing Department
* Product Sales (Information) Department
* Publication & Product Review Department
* Subscription & Registration Department

Phone calls and/or correspondence to Logical Systems will get YOU the best results if you address your questions, comments, or orders to the proper department:

Have your SERIAL NUMBER handy if you call. Also try to include it on your correspondence to us. We may require this for any number of reasons to be able to properly assist you.

V I E W F R O M T H E B O T T O M F L O O R by Bill Schroeder

Our move is over, and I hope we don't have to do it again for a long, long time. We have moved the entire LSI operation to the building so proudly displayed on the front of this issue of the Quarterly. The building is about 10,000 square feet and is located on about 4 acres of land in Brown Deer, Wisconsin (a suburb just north of Milwaukee).

We have completely redone the interior of our building to meet our exact needs, and I believe we have the best designed software facility anywhere, bar none. We have a MITEL SX-100 Phone system (GENERIC/217) which is a state of the art computer controlled phone system. For phones themselves, we use Panasonic KX-2202 feature phones. The building is completely wired for multiple phone paths to all locations, plus multi-path coaxial network system with complete cross connect and inter connect capability. If that is not enough, the entire building is wired for RS-232 networking with all cabling, for all systems, "HOME RUN" to a central control room.

This system will allow for all forseeable types of inter-office communications, much of which is already in daily use. Our intent is to maintain the best office communications system available. LSI hopes to be involved very deeply in the concepts and real world implementations of office communications and data management. The office of the future may be at LSI today.

We are very pleased with the new location and believe it should help us be more efficient when serving our customers, thus keeping costs (and prices) under control.

LSI is so pleased with our new facility (and proud of it) that we want to show it off. To this end we have decided to have an OPEN HOUSE on Saturday, June 25, 1983 from 12 noon until 5pm. Logical Systems will open its doors to the public. If you would like to visit a state of the art software facility and one of the best software staffs in the micro industry, please feel free to stop in. You will be very welcome.

At this OPEN HOUSE you will also have the pleasure of meeting some of the well known persons you have heard so much about in the TRS-80 industry. We, of course, can not promise which of our special guests will be able to attend, but here is a sampling of the industry figures that have been sent personal invitations:

Roy Soltoff	MISOSYS
Harv Pennington	IJG, Publishing
John Vanderlaughn	AEROCOMP
Bill Barden	TRS-80 Author
Irv Schmidt & Cameron Brown	80-US Magazine
Roger Billings & Kirk Hobart	LOBO Systems Inc.
Don White	TANDY Corp.
Tim Mann	LDOS "wizard"
Ray Daly	The Program Store
Kim Watt & Dennis Brent	Powersoft
John Lancione & John Long	Montezuma Micro
Earl Robinson	SoftERware
Bob Grayson & "MOJO" Jones	Micro Pro
Jim Crocker	Microsoft
Charlie Butler	The Alternate Source
Don Stanfield	TANDY Corp.
Tim Daneliuk	TRS-80 Author
P.T. Wolf	SAMS (software)
Wayne Green & Jake Commander	80-Micro Magazine
John Harding	MOLIMERX (England)
Earl Terwilliger	"C" programming Guru
Howard Gossman	H & E Computronics
Paul Grupp	Adventure International
Bill Driscol	S.B.S.G.
Bob Snapp	Snappware Inc.
Howard Wallowitz & Bill Prady	Small Computer Co.

These are just a few of the well know names that will be invited to attend. We wish they could all join us, but of course that will not be possible. Certainly some of these individuals will be present for you to meet and talk to. It should be a good time for all. Hope you can make it.

This newsletter arrived very late for two reasons. One, of course, was our move which took place as this newsletter was being prepared. The other reason is the Radio Shack Model-4 computer. As the official release date of the Model-4 was April 26th, we had to wait until that date if we wanted to mention the Mod-4 in this issue.

I'm sure that by the time you receive this you will already have heard much about the Mod-4, and what it is and is not. Here are the facts. The Mod-4 looks externally like a Mod 3 that was painted white, and had a few extra keys added to the keyboard. Internally the Mod-4 contains a Mod-3. That's right - all Mod-3 software should function UNCHANGED. The real Mod-4 part of the machine is a Z-80 clocked at 4 mhz. with 64k ram (optionally 128k), 80 X 24 video display, three function keys, a "control" key and "caps" key. It also has sound and reverse video capabilities.

All in all, the Mod-4 is a very nice Z-80 machine at a fair price. This same claim, however, can be made by several other machines. The one thing that separates the Mod-4 from the crowd is the TRSDOS 6.0 operating system. This is the first appearance of an LDDS 6.0 type operating system. It is without a doubt the most feature-laden OS ever provided as the standard operating system with a micro-computer. Of course, I am more than just a bit biased. Even discounting the fact that I would be unlikely to down play an LSI developed product, I feel the 6.0 generation of operating systems from LSI will set a new standard of excellence in the industry. We have every reason to believe that the 6.0 product will become a very popular OS on several micros by the end of 1983.

Tandy's Technical Reference Manual, catalog # 26-2110, will contain information on the technical aspects of TRSDOS 6.0 (which is LDOS 6.0). Therefore, for programmers wanting information on the LDOS 6.0 system, this manual should do the trick. The exact price and date of availability are not known for sure at this time, but it should be available shortly.

The LDOS 6.0 product for the Mod-2/12 type hardware is already underway and should be available yet this year. There will probably even be a couple of other machines flying the LDOS 6.0 banner in that time frame.

LSI will of course be offering most of our products in versions to run on the 6.0 type product. With regard to upgrades of 5.1 version products to 6.0 versions, there will be none. No trade-ins, updates or upgrades of any kind will be offered. The reason for this should be quite obvious, as all 5.1 versions of these products will function on the Mod-4 (when it is run in the Mod-3 mode) under LDOS 5.1.3. As our users upgrade their Mod-3 computers to Mod-4s they will still have the same fully functioning product on that machine. Therefore LSI has decided that there is no reason to offer upgrades, trade-ins or discounts of any sort.

So there are no misunderstandings it should be understood that LDOS 5.1 and the LSI 6.0 type systems represent TWO distinctly seperate product lines. Versions of products written for 5.1 will not run on 6.0 and visa-versa. Please be careful when ordering and be sure to specify the type of computer and the type of operating system for which the product is intended.

You will be able to tell the difference between 5.1 and 6.0 versions of LSI products by their titles. All 5.1 versions will have only a product name as their title or a product name followed by "5.1". With this in mind, a 5.1 product may have a title like "FED" or "FED/5.1" while the 6.0 versions of all products will have the 6.0 identifier as in "FED/6.0". It should also be noted that at present, the market place for 6.0 type products is much much smaller than the 5.1 type arena. It is for this reason that in most cases, the 6.0 version of an LSI product will be priced slightly higher than that of its 5.1 counterpart.

Let me now move on to some new products from LSI. The first of which is "FM" (File Management system). This a super file and directory management package (not a catalog program) that is fantastic in its abilities, but very simple to use and easy to understand. "FM" even has its own built in help system and is very forgiving. I find it to be like having an operating system within an operating system. "FM" is currently available in a 5.1 version at just $39.00, as "FM/5.1". Very shortly the 6.0 version of FM will be available for $49.00 as "FM/6.0". An article on the FM product appears in this issue. Check it out.

We are now offering a new package called "LSI/HELP". In the past we had marketed a product from MISOSYS called "HELP/QRC". This has now been replaced in our product line by the official LSI "HELP SYSTEM". This is actually a product line in itself, and as a product line offers much more than just HELP for LDOS. This product line is a complete help file generation and access system. Pricing and functions are varied depending on the portions or modules of the system that are ordered. To get all the details, read the article on LSI/HELP in this issue.

DUPE is a high speed LDOS type diskette duplicator for use by persons who need to make dozens or hundreds of perfect copies of LDOS type disks. This product came about mainly out of our own needs here at LSI. We have purchased many duplication programs from various authors at prices up to $500 and have not been satisfied with any of these. They all lacked in reliablity, error handling, and Support. The LSI duplication system works directly over the DOS and will produce totally verified exact duplicates at a very rapid rate, and catch almost any fault. This product is intended for software publishers and producers, and will not be available to individual users. The price is $200 and is available in a Model-3 (5.1.3) or MAX-80 version only. It will duplicate all LDOS type media 5.1 or 6.0. Much more info on this product can be found later in this issue.

FED II for 5.1 and 6.0 is really something. The very popular FED product has been enhanced with a built in Disassembler, a track/sector mode and much, much more. The best part is the price has not changed. Instead we have dropped the price of the original FED product to make this powerful maintenance tool available to many more users. The Price for FED-II/5.1 will be $39.00 and FED-II/6.0 will be $49.00. The price for the FED/5.1 is dropped to $19 effective 5/1/83. There is an article in this issue detailing the features and enhancements to be found in the FED-II product.

TBA, our very well received preprocessing translator for the BASIC programmer, is also available in a 6.0 version as TBA/6.0 for $79.00. This version of TBA has the same specifications as the TBA/5.1 product.

WordStar is probably the most popular and most widely used word processing software in the world. Of course, WordStar has, to this point, never been available for the TRS-80 Model-1 or 3 user except under CP/M. Well, the wait is over. MICROPRO, (the authors of WordStar), and LSI have now put together the ultimate in word processing packages for the TRS-80 Mod-1 and 3. WordStar is now available on smal-LDOS from both LSI and from MICROPRO. The package is provided on the LSI smal-LDOS operating system and will run on all LDOS 5.1.3 implementations, hard disk or floppy.

The WordStar/smal-LDDS package will carry an LSI suggested retail price of $395 as of September 1, 1983. Until that time LSI will offer a special introductory price of just $249!!! To take advantage of this special price, LSI must receive your order no later than August 30th 1983. This special amounts to a discount of over 36% and should be considered by anyone who is serious about word processing. Deliveries will begin in July and will be processed for shipment in the order received. From word of mouth we already have many prepaid orders. As a courtesy to those customers who have paid in advance, all prepaid orders will be sent out prior to all credit card or COD orders. So if you would like WordStar as soon as it is available, send LSI a check or money order for $249.00. LSI will pay all shipping and handling charges on prepaid orders of WordStar. We will also accept credit card and COD orders. Shipping and handling will be added to this type of order. Also available will be the MailMerge option for WordStar, as well as the WordMaster editor. Contact LSI for prices and availability.

LSI is now providing official support for the PERCOM DATA hard drive systems for the TRS-80 running under LDOS. The system provided for use with Percom hard drives has complete compatibility with software written for Radio Shack hard drive systems. All software that will run on the Radio Shack hard drive LDOS system will run without change on the PERCOM system. We at LSI are very pleased with the quality, reliability and the small size (and small price). There are several types of hard drive packages available from Percom with capacities starting at 5 meg. If you are in need of hard drive storage for your Model-1,3,4 this may be the system for you. For more information contact: PERCOM DATA CORP. 11220 Pagemill Rd., Dallas, Tx. 75243 (214) 340-7081

Our dealers and distributors have had some problems with customers expecting to purchase "LSI Specials" through them. Please understand that LSI is also a retail sale company and that we have our own promotions and specials. All PERMANENT price changes will be reflected by our dealers, but probably not until their old stock has moved. Our dealers are under no obligation to honor any "LSI Special". Specials and promotions by LSI MUST be ordered DIRECTLY from LSI unless otherwise stated.

Now for the specials of the quarter. Our present Jan '83 catalog is still available in limited supply, FREE for the asking. Our July '83 catalog will be available about mid-June and will also be FREE for the asking.

As noted previously, I have permanently dropped the price of the very popular LSI product FED (the LDOS "ZAP" type utility) to just $19.00. We would like a majority of our users to have this product for use in maintaining their software in a simple, easy to handle manner.

From now until JUNE 30, 1983 there will be special pricing on back issues of the LDOS Quarterly. They are normally $5.00 each. Until JUNE 30th (or until we run out), back issues will be just $2.50 each, plus shipping and handling. The July and October 1981 issues are already sold out, and some other issues are in very short supply. If you want to fill out your set with the real thing, order soon to avoid disappointmerit. The
price will go back to $5.00 each for back issue effective July 1st 1983.

Our popular MAIL/FILE and INVENTORY packages have now been reduced to just $99.00 across the board. These packages are available in MOD-1, 2, and 3 versions (no Mod 2 inventory). Hundreds of these were sold at up to $259. Now you can obtain these for a real bargain price. Write or call LSI for more details.

We should have plenty of our new catalog on hand for our OPEN HOUSE on June 25th, and they will be given out there. LSI will also be giving away a FREE QUICK REFERENCE CARD (a $5.95 value) to all LDOS owners that attend our open house, and bring with them their LDOS MASTER disk. We will be updating all LDOS MASTER disks brought to the open house at no charge. So come to the LSI open house on JUNE 25, 1983 from 12 noon to 5 PM and receive a FREE UPDATE, a FREE LDOS 5.1 QUICK REFERENCE CARD, a FREE LSI JULY CATALOG, and if all goes well, a FREE AUTOGRAPHED copy of the JULY '83 LDOS QUARTERLY.

READER SURVEY RESULTS from the Editors

In the October, 1982 Quarterly, we included a reader response card that listed most of our major products, and had space at the bottom for suggestions concerning future Quarterly content. Believe it or not, your suggestions actually played a major part in determining the content of the last newsletter, and those (including this one) that will follow! In order of popularity were the following requests:

Reviews and use of application programs - 24%
Basic assembly language programming - 21%
Languages other than BASIC or assembler - 17%
Explanations of LDOS utilities - 10.5%
JCL uses - 9.5%
Communications and R5232 - 9%
Interfacing to strange hardware - 9%

Several of these catagories are normally covered by the regular columns in each newsletter. Starting with the last issue, the LDOS: HOW IT WORKS column gives an in-depth explanation of a particular LDOS command or utility. This issue has three articles written by the LSI staff dealing with basic assembly language programming.

The one thing that we on the Quarterly staff do not have time to do is compile complete evaluations and reviews of application programs that will run with LDOS. We get many requests and calls asking "what data base manager/accounting package/etc. will run with LDOS." If you are currently running programs of this type with LDOS, why not write up a review for the Quarterly? Many other readers would appreciate it, and because we pay for published articles, you could even recover part of the purchase price.

We plan on including another reader survey card in this year's October issue, so if you want to see something in the Quarterly, write down your ideas and wait for the next round.

NEW PRODUCT ANNOUNCEMENTS

FM - File Manager Utility

FM stands for File Manager. It is a utility program designed to facilitate specific mass manipulation of files. Four modes are supported: Display, Kill, Move, and Remove. The first three modes correspond respectively with the LDOS library commands DIR, PURGE, and BACKUP by Class. The Remove mode is a combination of BACKUP and PURGE. This combination is, in effect, a transfer to a destination disk because files moved to it are purged from the source.

FM's utility comes from its ability to involve more than the usual number of drives associated with the normal operations. This means that comparison opportunities that can simplify maintenance of sets of diskettes are possible. For example, it is now possible to get a directory of the files on drive based on whether or not they exist on a second drive! The other modes can then be used to transfer those files between drives.

FM lets you specify a string of ASCII characters, inserts any matching filename into the string, and then writes the string to a file. This file can then be used as a JCL file to do things such as multiple renames, feed an editor assembler, etc.

The partspec (partial file specification) abilities of FM include three types of wildcards as well as the capability to specify a separate filename and extension for comparison purposes.

FM parameters include standard features such as modification of visibility status, file dating, file allocation information, sorted list switch, printer switch, and a query switch.

New parameters concern recent dates (today, yesterday etc.), unmodified files, a mod flag clearing utility and more.

Several special parameters deal with large volume drives to facilitate moving files from these larger drives to smaller volume diskettes.

FM increases the speed of file transfer by approximately 50% over the normal BACKUP, yet still includes a full read verification of transferred data. Also, error retry handling is built in. No more aborting a backup on a Parity error!

The parameters of FM can be considered to be grouped into classes by function. Certain parameters deal with the attributes of files, some with dates, and others with size. To allow FM to be controlled by a JCL file, the JCL and ABORT parameters are included. One group of parameters deals with the type of display you will get from FM; sorted or unsorted, on the video or the printer, and prompt or go non-stop.

All of this gives the user much more specific, simplfied control of file listing, killing, and moving than was previously possible. In one command line, it is possible to copy files from a source to a destination which only exist on yet a third drive; or purge them or simply view or print them. This kind of machination would have taken at least two printed directories and at least one backup assuming unprecedented luck. FM can do it in ONE LINE!

FM allows hundreds of combinations and, in general, can save the adroit user thousands of keystrokes and many hundreds of minutes.

FM for the LDOS 5.1 operating system is available from LSI for $39.$0. A 6.0 version will be available later this year.

LSI HELP PACKAGE

There is now a new release of HELP available. The HELP system consists of three different packages. All packages are available as a 5.1.3 version, with a 6.0 version ready shortly.

Package number one is called LDOS HELP. This includes help information on LDOS library and utility commands, and help on all LBASIC statements and functions. To access the information, it is only necessary to type something like : "HELP LDOS SYSTEM". This will display to the video or printer all information concerning the LDOS library command SYSTEM, which includes the syntax line and a brief description of the parameters available.

The video is restored to the calling screen to allow help to be invoked from within programs. In addition, the help module can reside in high memory to further enhance its usefulness in programs which can not ordinarily or readily access a system command. The resident module can address up to fifteen HELP data files simultaneously.

All HELP data files end in an extension of "/HLP". If the user is unaware of what help is on line, typing the word "HELP" at LDOS Ready causes all available files to be displayed, and an option is given to utilize one of them.

The high memory module has a disable parameter which in most cases can release the memory used when the HELP module is no longer required.

LDOS HELP includes two data files LDOS/HLP and LBASIC/HLP as well as the two HELP modules HELP/CMD (from LDOS) and HELPRES/CMD (in memory). This package also comes with a 5.1.3 Quick Reference Card.

The second package is called TECHNICAL HELP. This includes information in four data files which cover all Z-80 mnemonics as well as MOST of the technical section of the LDOS manual! Imagine programming within EDAS and never having to look up anything pertaining to system entry points, machine specific calls, SVC numbers, or the registers employed on a call to the system. The Z-80 section includes mnemonic description, object code, flag settings, timing and format.

TECHNICAL HELP includes both HELP modules as well as the data files Z80A/HLP, Z80M/HLP, TECH1/HLP and TECH2/HLP. The TECH help files are not turorials, but merely reference the information in the LDOS manual technical section.

The last module is called HELP GENERATOR. This allows the user to create HELP data files by processing a user generated ASCII text file into a HELP usable file. Help may even be generated for user applications programs and then, through use of the HELPRES/CMD, employed from within almost any program. Simply type the desired text into a word processor or a text editor, save it in ASCII, and run the HELPGEN/CMD. The rules regarding proper text format are simple to follow.

The HELP GENERATOR includes HELPGEN/CMD as well as the two display modules, HELPRES/CMD and HELP/CMD.

All three HELP packages come in either a 5.1.3 or a 6.0 version. (Note that LDOS HELP 6.0 does NOT contain a quick reference card.) HELP 6.0 will be available in early June. The HELP 5.1 prices are $19 for LDOS Help, $29 for TECH Help, and $49 for the HELPGEN package. HELP 6.0 prices will be $29 for LDOS Help, $39 for TECH Help, and $59 for the HELPGEN package.

DUPE/CMD

This program is designed for software distributors, not for the general public, and as a result is priced as such. It is the same program as used by LSI to duplicate all of our products.

DUPE is a disk duplicating program for use with the LDOS 5.1 operating system. It is a single-pass duplicator, formatting and writing each track on all destination drives before stepping in to the next track. Since it is a one pass duplicator, it is significantly faster than the normal Format and Backup duplication method. There are several error catching features that are built in, including checksumming both the source and destination disks to detect hardware or memory related errors, as well as the normal CRC checks. A bi-directional verify is also available, and tests each track while stepping the head out as well as when stepping in.

Once DUPE is loaded, it does not require a system disk in drive 0, and can copy the source disk to more than one destination drive on each pass. Since this is a byte for byte duplication, this means that Model I disks can be made on a Model III, and vice versa (assuming double density capability on the Mod I).

Disks which contain errors are identified at the end of each pass, and a running total of the good disks created is displayed. The program can be used to duplicate any LDOS type floppy disk in any LDOS system with two or more drives.

DUPE is available only from LSI, priced at $200.00.

FEDII - The LDOS File/Disk Editor

FEDII is an all purpose File and Disk editor, and is an enhancement of the original FED file editor. The display consists of a 256 byte sector with a hex display and ASCII display area representing each byte of the sector. Separate cursors in both display areas provide for easy identification of the current modification byte, and full cursor positioning makes it possible to quickly position to any byte in the record.

Additional display information in both modes includes the filename/drive number, record number (track and record number in the drive mode), relative byte number in the sector over which the cursor is positioned, and also the decimal and binary representations of that byte.

If you are in the file mode and the file is a Command File (machine language program), additional information will also be displayed regarding the byte over which the cursor is positioned. If the cursor is positioned over a load block header, information regarding the load block will be displayed. If the cursor is positioned within program code, the load address of that byte will be displayed, along with a disassembly of the instruction.

If you are in the drive mode, the filename associated with the current sector will be displayed (if applicable).

Any byte within the current sector can be modified by entering either hex digits or by directly typing in ASCII characters. Changes to the current sector will not be made until a Save Sector command is issued. Additionally, the current sector or entire file (or disk) can be sent to the printer, and the contents of the current sector from the present cursor position to the end of the sector can be nulled out (all bytes being set to X'00').

Several other means of positioning to specific records are supported. Commands may be issued to position to the beginning record, ending record, or a specific record in either the file or disk mode. Additionally, the next record and previous record may be accessed.

In the file mode, positioning commands are also available to jump to the address specified in an instruction, position to the next instruction, position to the previous load block and position to the next load block. These commands are all relative to the current cursor position.

FEDII also provides search capabilities in either the file or disk mode. Hex and ASCII strings may be searched for, and the cursor will be positioned to the search string if it is found. ASCII text strings may be searched for, and the case of the string (upper or lower) is ignored in the search. In the file mode, if the file is a command file, a load address may be specified. If found, the cursor will be positioned to the byte which loads at the specified address.

FEDII is available for just $39.00 for Version 5.1, and $49.00 for Version 6.0

TBA-60 (The BASIC Answer)

TBA-60 is a BASIC text processing utility. It is designed to allow the BASIC programmer to construct code in a structured manner. "Source code" is created with a word processor or text editor which allows the user to exploit the powerful editing and movement features characteristic to those types of software. Source code can also be created by means of a BASIC interpreter. TBA-60 is then used to process this source code into ordinary interpretive BASIC code.

TBA-60 utilizes labels in lieu of line numbers. Branching in a program is accomplished by means of a descriptive label as opposed to an arbitrary line number. This means that blocks of code (subroutines) can be referenced by names which reflect their function, such as @SORT.NAMES. Labels may be up to fourteen characters in length. The use of labels allows for relocatable BASIC subroutines without the problems associated with renumbering.

TBA-60 allows the use of longer variable names. Variable names may be up to fourteen significant characters. This allows the use of descriptive names to represent variables, which augments program readability in the case where the program has not been examined for some period of time.

TBA-60 introduces the concept of "Conditional Translation". The feature allows the co-existence of "machine dependent" code within the same source file. Irrelevant sections of code may be ignored during processing.

TBA-60 also allows the use of Local and Global variables. Local variables are those variables which retain their value only in a given subroutine. This means that variable tracking and confilct problems are minimized.

TBA-60 is available for $79.00, and requires the 6.0 operating system.

WordStar 3.0 - A word processor

WordStar is probably the most widely used word processing program in the world. Until now, using it meant running under CP/M or some other operating system. Being one of the first word processors means that WordStar is relatively bug free, which is a real boon when large documents are at stake! Even though the final release version was not available when it came time to start this Quarterly, the initial beta test copy proved so reliable that what you are reading was done with WordStar. The file is 200K+, so you have an idea of one of the nicest features of WordStar.

This issue was formatted with a MAX-80 and a hard disk. The printing was done on our Daisy Wheel II from Radio Shack. At the time, the incremental print driver was not completed, so the normal space justification was used, along with our old boldface and underline filters.

Some of WordStar's text handling features are horizontal scrolling, block moves and insertions of columnar data (!), text insertion anywhere in the text from files on disk (!), the ability to save a block of text to a file of your choice, and automatic file backup when opening a document. Screen oriented features are constant display of page number, line number, and column position, adjustable levels of online help (a real plus when learning the system), and the ability to justify the text ON THE SCREEN!. Some special print features are the ability to define user print codes, the ability to redefine headers and footers whenever desired, and the ability to set conditional page breaks (i.e., if there are less than so many lines left on the page, start a new page here).

From an LDOS standpoint, WordStar lets you use the LDOS KI/DVR and keyboard filters, as well as the standard printer driver and any filters. The spooler will also function.

There are just too many features to mention all of them here. As WordStar is so popular, you may be able to get a look at a manual in a local non-RS, computer store. I
have used (or attempted to use) many word processors in putting together the Quarterly. My personal choice is WordStar, both for the features and the reliability it offers.

The MailMerge option from Micropro is also available for the TRS80 and will be available from LSI.

CASE/CMD

By Rick Toblas

Here's a little monitor routine for those of us without the lower case modification on the Model I. I had this thing about hitting the shift zero '0' when keying the '*' or the ')'. To combat the problem of not knowing what case I was in , I decided to write this program.

The program runs on a Model I under control of the task processor. The routine monitors the KFLAG$ bit 5 (caps lock bit) and prints on the video screen (the up arrow if in upper case or the down arrow if in lower case).

The program allows for 2 parameters:

DIS -- disable routine
POS -- screen position for arrows (abbrevations P for POS)

No parameters will enable the routine and use position 62.If only P05 (or P) is used the routine will enable and use the position given.

The routine relocates itself in high memory, but the program does not reuse the same area if re-activated. The program does check to see if the routine is already active and the program will not re-activate but returns to DOS.

```
00100         TITLE   <CASE/CMD.UP/LOW  MONITOR>
00110 ;***********************************************
00120 ;**                                           **
00130 ;**     RICK TOBIAS     07/82    VER 01.01     **
00140 ;**            THE COMPUTER SERVICES           **
00150 ;**                 CASE/CMD                  **
00160 ;**           ----------------------          **
00170 ;**   THIS PROGRAM IS IN TWO PHASES            **
00180 ;**   PHASE I  WILL RELOCATE PHASE II TO HIGH  **
00190 ;**   MEMORY. PHASE II WILL ATTACH TO THE      **
00200 ;**   INTERRUPT LEVEL SLOT 2 AND MONITOR       **
00210 ;**   THE VALUE OF KFLAG$ BIT 5. IF ONE        **
00220 ;**   AN UP ARROW WILL BE DISPLAYED OTHERWISE   **
00230 ;**   THE DOWN ARROW WILL BE DISPLAYED.         **
00240 ;**   PARAMETERS CAN BE PASSED TO THE PROGRAM   **
00250 ;**   ALLOWABLE PARAMETERS ARE  (DIS) (P=NNNN)  **
00260 ;**      (POS=NNNN)                            **
00270 ;**   NO PARAMETERS WILL ENABLE ROUTINE AND USE **
00280 ;**   POSITION 62 OF THE VIDEO TO PRINT ARROWS  **
00290 ;**   DIS - WILL DISABLE ROUTINE                **
00300 ;**   P OR POS WILL USE THAT SCREEN POSITION    **
00310 ;**   TO DISPLAY THE ARROWS                     **
00320 ;**   NO TEST IS MADE FOR POSITIONS             **
00330 ;**   POSITIONS MUST BE 0 THRU 1023             **
00340 ;**           --------------------              **
00350 ;***********************************************
00360 ;
00370 ;
00380 CR      EQU     0DH             ;RETURN CHAR
00390 @VDCLS  EQU     01C9H           ;CLEAR SCREEN
00400 @DOS    EQU     402DH           ;RETURN TO DOS
00410 HIGH$   EQU     4049H           ;HIGHEST USEABLE ADDRESS
00420 @VDLINE EQU     4467H           ;DISPLAY STRING
00430 @PARAM  EQU     4476H           ;SCAN OPTIONAL PARMS
00440 KFLAG$  EQU     4423H           ;KEYBOARD FLAG
00450 USTOR$  EQU     4DFEH           ;USER STORAGE AREA
00460 @ADTSK  EQU     4410H
00470 @RMTSK  EQU     4413H
00480 @RPTSK  EQU     4416H
00490 TCBOFF  EQU     45B5H           ;ADDR IF TCB OFF
00500 @TCB2   EQU     4504H           ;ADDR OF TCB SLOT 2
00510 ;
00520         ORG     5200H
00530 CASE    EQU     $
00540         PUSH    HL              ;SAVE $INBUFF
00550         CALL    @VDCLS          ;CLEAR SCREEN
00560         LD      HL,MSG1         ;FIRST MESSAGE
00570         CALL    @VDLINE         ;DISPLAY IT
00580         POP     HL
00590         LD      DE,PRMTBL$      ;GET ADDR OF PRAMETERS
00600         CALL    @PARAM          ;DETERMINE PARAMETERS
00610         JP      NZ,PARMERR      ;BAD PARAMETERS
00620 ;
00630 PPARM   LD      BC,3EH          ;INIT TO P05 62
00640         LD      HL,3C0H         ;1ST SCREEN P05
00650         ADD     HL,BC
```

```
00660          LD      (A1+1),HL       ;LOAD SCREEN POS
00670          LD      (A2+1),HL
00680 DPARM    LD      BC,0            ;INIT AT OFF
00690          LD      A,B
00700          OR      C
00710          JR      Z,EPARM
00720 RMVTSK   LD      A,2
00730          CALL    @RMTSK
00740          LD      HL,MSG4         ;DISABLE MESSAGE
00750          CALL    @VDLINE         ;DISPLAY IT
00760          JP      @DOS
00770 ;
00780 ;
00790 ;
00800 EPARM    EQU     $
00810          LD      HL,(@TCB2)      ;TCB SLOT 2
00820          LD      BC,TCBOFF       ;TCB SLOT INACTIVE
00830          SBC     HL,BC           ;IF ZERO THEN EQUAL
00840          JR      NZ,ACTIVE
00850          LD      HL,(HIGH$)      ;REDUCE HIGH$ BY THE
00860          LD      BC,LAST-START   ;LENGTH OF ROUTINE
00870          XOR     A               ;CLEAR THE CARRY FLAG
00880          SBC     HL,BC           ;CALC NEW HIGH$
00890          LD      (HIGH$),HL      ;ROUTINE NEW PROTECTED
00900          INC     HL              ;POINT HL AT NEW START
00910          LD      (USTOR$),HL     ;SAVE ENTRY FOR TCB
00920          EX      DE,HL           ;XFER NEW START TO DE
00930          LD      HL,START        ;LOAD ADDRESS OF ROUTINE
00940          LDIR                    ;MOVE ROUTINE TO TOP
00950          JP      DOIT
00960 ACTIVE   LD      HL,MSG5
00970          CALL    @VDLINE
00980          JP      @DOS            ;RETURN TO DOS
00990 DOIT     LD      DE,USTOR$       ;LOAD TCB ADDR
01000          LD      A,2             ;POINT TO SLOT 2
01010          CALL    @ADTSK          ;ADD TASK
01020          LD      HL,MSG3         ;MESSAGE 2
01030          CALL    @VDLINE         ;DISPLAY IT
01040          JP      @DOS            ;RETURN TO DOS
01050 ;
01060 ;
01070 ;
01080 PARMERR  EQU     $
01090          LD      HL,MSG2
01100          CALL    @VDLINE
01110          JP      @DOS
01120 ;
01130 MSG1     DM      'UPPER/LOWER CASE MONITOR   V1.0 7/82',CR
01140 MSG2     DM      'BAD PARAMETES...................',CR
01150 MSG3     DM      'U/L CASE MONITOR NOW ACTIVE.....',CR
01160 MSG4     DM      'U/L MONITOR DISABLED............',CR
01170 MSG5     DM      'U/L ALREADY ACTIVE.............',CR
01180 ;
01190 PRMTBL$  DB      'DIS   '
01200          DW      DPARM+1
01210          DB      'POS   '
01220          DW      PPARM+1
01230          DB      'P     '
01240          DW      PPARM+1
01250          NOP                     ;END TABLE
01260 ;
01270 ;
```

```
01280 ;****************************************
01290 ;**                                    **
01300 ;**        ACTUAL MONITOR ROUTINE      **
01310 ;**                                    **
01320 ;****************************************
01330 ;
01340 ;
01350 START   EQU     $
01360 ;
01370         PUSH    DE
01380         PUSH    HL
01390         PUSH    AF
01400         LD      HL,KFLAG$       ;KEYBOARD FLAG
01410         BIT     5,(HL)          ;TEST BIT 5
01420         JR      Z,$+11          ;IF 1 THEN LOWER CASE
01430         LD      A,5BH           ;LOAD UP ARROW TO REG A
01440 A1      LD      (3C00H),A       ;DISPLAY IT
01450         POP     AF
01460         POP     HL
01470         POP     DE
01480         RET
01490 LOWER   LD      A,5CH           ;LOAD DOWN ARROW TO REG A
01500 A2      LD      (3C00H),A       ;DISPLAY IT
01510         POP     AF
01520         POP     HL
01530         POP     DE
01540         RET                     ;RETURN TO SYSTEM
01550 ;
01560 ;
01570 LAST    EQU     $               ;LABEL FOR CALC LENGHT
01580         END     CASE
```

This is the hex code for the CASE monitor, and can be made into a /CMD file using the BINHEX program found later in this issue.

```
05 06 43 41 53 45 20 20 01 02 00 52 E5 CD C9 01 21 7C 52 CD 67 44 E1 11 23 53 CD 76 44
C2 73 52 01 3E 00 21 00 3C 09 22 49 53 22 52 53 01 00 00 78 B1 28 0E 3E 02 CD 13 44 21
E3 52 CD 67 44 C3 2D 40 2A 04 45 01 B5 45 ED 42 20 19 2A 49 40 01 1C 00 AF ED 42 22 49
40 23 22 FE 4D EB 21 3C 53 ED B0 C3 62 52 21 03 53 CD 67 44 C3 2D 40 11 FE 4D 3E 02 CD
10 44 21 C2 52 CD 67 44 C3 2D 40 21 A1 52 CD 67 44 C3 2D 40 55 50 50 45 52 2F 4C 4F 57
45 52 20 43 41 53 45 20 4D 4F 4E 49 54 4F 52 20 20 20 56 31 2E 30 20 37 2F 38 32 0D 42
41 44 20 50 41 52 41 4D 45 54 45 53 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E
2E 2E 0D 55 2F 4C 20 43 41 53 45 20 4D 4F 4E 49 54 4F 52 20 4E 4F 57 20 41 43 54 49 56
45 2E 2E 2E 2E 2E 0D 55 2F 4C 20 4D 4F 4E 49 54 4F 52 20 44 49 53 41 42 4C 45 44 2E 2E
2E 2E 2E 2E 2E 2E 2E 01 5A 00 53 2E 2E 0D 55 2F 4C 20 41 4C 52 45 41 44 59 20 41 43 54
49 56 45 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 0D 44 49 53 20 20 20 22 52 50 4F 53 20
20 20 15 52 50 20 20 20 20 20 15 52 00 D5 E5 F5 21 23 44 CB 6E 28 09 3E 5B 32 00 3C F1
E1 D1 C9 3E 5C 32 00 3C F1 E1 D1 C9 02 02 00 52
*1D
```

SOLEFIX -- Fix that GAT error!

Erik Ruf
202 Kenilworth Dr.
Akron, Ohio 44313
(216) 867-7575

As supplied, an LDOS double-density diskette will not boot on a Model I. This is because the ROM is expecting a single-density boot sector, and is thus unable to read the double-density boot routine on the disk. Overcoming this difficulty is simple -- you use the SOLE utility from Misosys. SOLE reformats cylinder 0 in single-density and

places its own boot routine on this track. SOLE also alters the diskette's directory by setting the allocation bits for all 3 granules of cylinder 0 so that the system won't attempt to write over the boot routines. Unfortunately, SOLE fails to allocate one of these these granules to a file; this creates a granule allocation table (or GAT) error.

Normally, this poses no problem for the user; as long as the granule is allocated, LDOS doesn't care what it is allocated to. However, some utilities (such as Super-Utility Plus) will detect this error, and will not proceed until it is fixed. Super-Utility Plus graciously fixes this error for you (whether you like it or not!), and thus allows the system to write on cylinder 0, granule 2; your boot routine will be ruined when this sector is overwritten.

The solution? Why not allocate all of cylinder 0 to BOOT/SYS? All one needs to do is to change one of the extent bytes in BOOT/SYS's directory entry. This will allocate the granules to BOOT/SYS. It's also a good idea to give BOOT/SYS the CREATE attribute so that the system won't be able to de-allocate this extra granule.

The quickest way to do this is with FED or any other disk file or sector editor. Use FED to load relative sector 2 of DIR/SYS. Then, change byte 01 from a 00 to an 80 (this will give BOOT/SYS the CREATE attribute) and change byte 17 from a 01 to a 02 (this will allocate all of cylinder 0 to BOOT/SYS). Write the buffer to the disk, and you're done. (Note: all data given in HEX).

This seems simple enough, but it is rather cumbersome when you have to patch about 50 disks. My first thought was to use a direct PATCH on DIR/SYS, so I tried it. BIG MISTAKE!! This led to an instant system crash; I spent a half-hour resurrecting the disk. Moral: never, NEVER use the PATCH utility on DIR/SYS! If you don't understand why, read the January 1983 LDOS Quarterly. (Oh, if mine had only arrived 3 days earlier...)

In order to take care of the problem once and for all, I wrote SOLEFIX/CMD (listed below as SOLEFIX/ASM), a small machine language routine that does the patching for me. SOLEFIX is entered by typing SOLEFIX (DRIVE=d) from the LDOS Ready level, where d is the drive # containing the diskette to be fixed. (As SOLEFIX uses information from the drive's Drive Code Table (DCT), be sure to log in the diskette by using DEVICE or LOG/CMD). After parsing the command line to get the drive #, SOLEFIX checks to make sure that the drive in question is a 5" floppy drive that contains a double-density diskette that has a single-density cylinder 0 (i.e. it has been SOLEd). If the drive and diskette pass the test, SOLEFIX uses the LDOS I/O primitives to load the offending directory sector, fix it, and write it back to the disk. SIMPLE! The actual read-fix-write is a very small part of the program; most of the code just checks for errors.

Hopefully, this routine will end the problems associated with SOLE's inherent GAT error. Happy disk-fixing!

```
00100          TITLE <SOLEFIX/ASM>
00110 ;
00120 ;        Program to fix the GAT error introduced by
00130 ;        SOLE by allocating the entire boot track to
00140 ;        BOOT/SYS and giving BOOT/SYS the CREATE
00150 ;        attribute so that none of its granules will
00160 ;        be deallocated. The program also checks to
00170 ;        ensure that the disk in question is a 5"
00180 ;        double-density floppy disk with a single
00190 ;        density track 0. As this program uses the
00200 ;        DCT, be sure to log in the drive before using
00210 ;        SOLEFIX.
00220 ;
00230 ;        syntax: SOLEFIX (DRIVEd) <ENTER>
00240 ;
00250 *LIST OFF                       ;suppress list of EQUates
```

```
00260 *GET EQUATE1/EQU               ;get system EQUates
00270 *LIST ON                       ;list back on
00280 ;
00290         ORG     6000H
00300 START   EQU     $              ;it all begins here
00310         PUSH    HL             ;save command line
00320         LD      HL,BANNER      ;addr for title
00330         CALL    @DSPLY         ;display title
00340         POP     HL             ;get command line
00350         LD      DE,PARMS       ;parm table address
00360         CALL    @PARAM         ;parse it, LDOS!
00370         JR      NZ,PARMERR     ;parse had troubles
00380 REPLACE EQU     $
00390         LD      HL,0FFFFH      ;replaced by parse
00400         LD      A,L            ;get drive #
00410         CP      8              ;check it
00420         JR      NC,BADNUM      ;bad or missing drive #
00430         LD      C,A            ;drive # in C
00440         CALL    GETDCT         ;drive info addr in IY
00450         LD      A,(IY)         ;get 1st DCT byte
00460         CP      0C9H           ;check enable/disable
00470         JR      Z,DIS          ;disabled drive!
00480         LD      A,(IY+3)       ;check drive type byte
00490         BIT     7,A            ;check for software WP
00500         JR      NZ,WPERR       ;write-protected disk!
00510         BIT     5,A            ;5" or 8"?
00520         JR      NZ,ERR8IN      ;8" drive!
00530         BIT     3,A            ;floppy or winchester?
00540         JR      NZ,WINCH       ;winchester drive!
00550         CALL    @CKDRV         ;check drive
00560         JR      C,WPERR        ;write-protected!
00570         JR      NZ,NREADY      ;drive not ready!
00580         LD      HL,MSG1        ;"checking drive"
00590         CALL    @DSPLY         ;display it
00600         LD      HL,SBUFF$      ;use system sector buffer
00610         LD      DE,0           ;cyl=0, sect=0
00620         CALL    RDSECT         ;read it
00630         JR      NZ,DOSERR      ;DOS error
00640         BIT     6,(IY+3)       ;check density
00650         JR      NZ,DDEN0       ;boot track must be SDEN!
00660         INC     D              ;cyl=1, sect=0
00670         CALL    RDSECT         ;read it
00680         JR      NZ,DOSERR      ;DOS error
00690         BIT     6,(IY+3)       ;density
00700         JR      Z,SDEN1        ;disk not DDEN!
00710         PUSH    HL             ;save buffer addr
00720         LD      HL,MSG2        ;"WRITING" message
00730         CALL    @DSPLY         ;display it
00740         POP     HL             ;get buffer
00750         LD      B,0            ;DEC for BOOT/SYS
00760         CALL    DIRRD          ;get directory sector
00770         JR      NZ,DOSERR      ;DOS error
00780         INC     HL             ;point to attrib byte
00790         SET     7,(HL)         ;CREATE attribute
00800         LD      DE,22          ;offset to alloc byte
00810         ADD     HL,DE          ;point to alloc byte
00820         LD      (HL),02        ;allocate 3 grans (cyl 0)
00830         CALL    DIRWR          ;write it back!
00840         JR      NZ,DOSERR      ;DOS error
00850         JP      @EXIT          ;Done! -- back to LDOS
00860 PARMERR LD      HL,ERR1        ;parameter error
00870         JR      ERRPRT         ;display it!
```

```
00880 BADNUM  LD      HL,ERR2         ;bad or missing drive #
00890         JR      ERRPRT
00900 DIS     LD      HL,ERR3         ;disabled drive
00910         JR      ERRPRT
00920 WPERR   LD      HL,ERR4         ;write-protected
00930         JR      ERRPRT
00940 ERR8IN  LD      HL,ERR5         ;not S" drive
00950         JR      ERRPRT
00960 WINCH   LD      HL,ERR6         ;not floppy drive
00970         JR      ERRPRT
00980 NREADY  LD      HL,ERR7         ;drive not ready
00990         JR      ERRPRT
01000 DDEN0   LD      HL,ERR8         ;DDEN cylinder 0
01010         JR      ERRPRT
01020 SDENl   LD      HL,ERR9         ;SDEN cylinder 1
01030 ERRPRT  EQU     $
01040         CALL    @DSPLY          ;display error msg
01050         JP      @ABORT          ;back to LDOS
01060 DOSERR  EQU     $
01070         AND     63              ;lop off bits 6+7
01080         JP      @ERROR          ;display & jump to LDOS
01090 PARMS   EQU     $               ;parameter table
01100         DEFM    'DRIVE '        ;parameter word
01110         DEFW    REPLACE+1       ;storage location
01120         NOP                     ;end of table
01130 ERR1    DEFM    'Parameter Error',13
01140 ERR2    DEFM    'Bad or missing drive #',13
01150 ERR3    DEFM    'Disabled Drive',13
01160 ERR4    DEFM    'Write-Protected disk',13
01170 ERR5    DEFM    'Not a 5" drive!',13
01180 ERR6    DEFM    'Not a floppy drive!',13
01190 ERR7    DEFM    'Drive not Ready',13
01200 ERR8    DEFM    'Track 0 must be SINGLE-DENSITY!',13
01210 ERR9    DEFM    'Diskette must be DOUBLE-DENSITY!',13
01220 BANNER  DEFM    28,31,'SOLEFIX - SOLE Boot Fix Routine  '
01220         DEFM    '(Erik Ruf, l/27/83)',10,10,13
01230 MSGl    DEFM    'Checking Diskette Format',13
01240 MSG2    DEFM    'Patching Diskette Directory',13
01250         END     6000H
```

This is the HEX listing for the SOLEFIX program:

```
05 06 53 4F 4C 45 46 49 01 02 00 60 E5 21 7B 61 CD 67 44 E1 11 B2 60 CD 76 44 20 6C 21
FF FF 7D FE 08 30 69 4F CD 8F 47 FD 7E 00 FE C9 28 63 FD 7E 03 CB 7F 20 61 CB 6F 20 62
CB 5F 20 63 CD B8 44 38 54 20 61 21 B3 61 CD 67 44 21 00 42 11 00 00 CD 77 47 20 63 FD
CB 03 76 20 4F 14 CD 77 47 20 57 FD CB 03 76 28 48 E5 21 CC 61 CD 67 44 E1 06 00 CD 10
4B 20 42 23 CB FE 11 16 00 19 36 02 CD 1F 4B 20 34 C3 2D 40 21 BB 60 18 26 21 CB 60 18
21 21 E2 60 18 1C 21 F1 60 18 17 21 06 61 18 12 21 16 61 18 0D 21 2A 61 18 08 21 3A 61
18 03 21 5A 61 CD 67 44 C3 30 40 E6 3F C3 09 44 44 52 49 56 45 20 11 60 00 50 61 72 61
6D 65 74 65 72 20 45 72 72 6F 72 0D 42 61 64 20 6F 72 20 6D 69 73 73 69 6E 67 20 64 72
69 76 65 20 23 0D 44 69 73 61 62 6C 65 64 20 44 72 69 76 65 0D 57 72 69 74 65 2D 50 72
6F 74 65 63 74 65 64 01 EA 00 61 20 64 69 73 6B 0D 4E 6F 74 20 61 20 35 22 20 64 72 69
76 65 21 0D 4E 6F 74 20 61 20 66 6C 6F 70 70 79 20 64 72 69 76 65 21 0D 44 72 69 76 65
20 6E 6F 74 20 52 65 61 64 79 0D 54 72 61 63 6B 20 30 20 6D 75 73 74 20 62 65 20 53 49
4E 47 4C 45 2D 44 45 4E 53 49 54 59 21 0D 44 69 73 6B 65 74 74 65 20 6D 75 73 74 20 62
65 20 44 4F 55 42 4C 45 2D 44 45 4E 53 49 54 59 21 0D 1C 1F 53 4F 4C 45 46 49 58 20 2D
20 53 4F 4C 45 20 42 6F 6F 74 20 46 69 78 20 52 6F 75 74 69 6E 65 20 20 28 45 72 69 6B
20 52 75 66 2C 20 6C 2F 32 37 2F 38 33 29 0A 0A 0D 43 68 65 63 6B 69 6E 67 20 44 69 73
6B 65 74 74 65 20 46 6F 72 6D 61 74 0D 50 61 74 63 68 69 6E 67 20 44 69 73 6B 65 74 74
65 20 44 69 72 65 63 74 6F 72 79 0D 02 02 00 60
*B6
```

Changing LBASIC defaults

This JCL file was submitted by Eric Ruf. Its basic purpose is to patch LBASIC (Version 5.1.3) to change the default for the EXT, BLK, and FILE parameters. To use this JCL, type in a DO command specifying the EXT, BLK, and F parameters as desired. DOing the file with no parameters will display the built in help message.

```
//IF -EXT&-BLK&-FIL
//ELSE if a parm was entered ...
//IF EXT
//ASSIGN EXT=FF
//ELSE
//ASSIGN EXT=00
//END
//IF BLK
//ASSIGN BLK=FF
//ELSE
//ASSIGN BLK=00
//END
//ASSIGN FSPEC=LBASIC/CMD.EZTO
PATCH #FSPEC# (D02,9E=#EXT# #EXT#:D11,25=#BLK# #BLK#)
//IF FIL
PATCH #FSPEC# (D02,A5=#FIL#)
//END
//EXIT
//END  (End the Help IF)
```

LDOS "CARD" UTILITY

by Canadian-Micro
119A Bathurst St.
London, ONT N6B 1P1

The purpose of this program is:

1. to execute multiple commands with a minimum of keystrokes.
2. to perform file copies based on mod flags.
3. to re-attrib files for easier directory management.
4. to rename files/extensions.
5. to delete files.

Most will find it particularly useful on large directories such as on double-sided 40 or 80 track drives.

To execute the program type: LBASIC RUN"CARD/BAS" <ENTER>.

- Display will prompt for (source or target) D r i v e # ?
  Type drive # <0-7>.
- You will then be prompted for directory parameters. For visible entries type:
  V <ENTER> , etc.
- Files will then be displayed one at a time for you to enter commands.

  eg. ==> SYS6/SYS:0 (isP) <?>

- First is the filename with extension and drive #.
- Inside the ( ) brackets is the file attributes:

  (s)=SYSTEM (v)=VISIBLE (i)=INVISIBLE (P)=PROTECTED

- Inside the < > brackets is where command is entered.

- Menu:

  <C>opy - will prompt for destination drive # (0-7). Destination file will have a mod
           flag unless original file is password protected, which causes "cloning".
  <A>ttrib - will prompt <V/I/X> and change attributes:
           V makes file visible;
           I makes file invisible;
           X removes all password protection.
  <R>ename - will prompt for new name, parameters are the same as "LDOS" rename.
  <D>elete - will ask OK?
           And then kill file, protected or not!
  <Q> key - will list full drive directory. To return to point of exit press any key.
  <-> key - after instruction is completed will backup to previous file for further
         commands.
  <SPACE BAR> - will advance to next file. (Nothing done.)
  <ENTER> - if first key pressed after 'C','A','R' or 'D' file will be re-listed.
        (Nothing done.)
  <@> key - when pressed for "D r i v e #" exits program. Pressed at any other time
        will return to start.

- Commands are executed as they are entered.

```
1 CLEAR3000:GOSUB49
2 ONERRORGOTO0:B$="  D r i v e   #   ? ":F=-1:GOSUB48:IFI$="@"THENCLS:CMD"S"
3 IFI$=""THEN2ELSEQ$=":"+I$:Z$="DIR/SYS.EZTO"+Q$
4 B$=" Parameters     <S/I/V>":F=1:GOSUB48:IFI$="@"THEN2
S IFI$=""THEN2ELSEY=ASC(I$): IFY<97THENY=Y+32
6 IFY=115ORY=118ORY=l05THENA$=CHR$(Y)ELSE4
7 OPEN"RO",1,Z$,32:ONERRORGOTO54:FIELD1,1ASM$,4ASL$,8ASN$,3ASE$,2ASJ$:R=LOF(1):E
=17:FORX=ETOR
8 GET1,X:Y=ASC(M$):IFYAND128THEN34
9 IFYAND64THENG$="s":IFN$="BOOT    "ORN$="DIR     "THEN34ELSE12
10 IFYANDl6THENG$="v"ELSE34
11 IFYAND8THENG$="i"
12 IFG$<>A$THEN34ELSEP$=CHR$(150)+"B":IFJ$=P$THENP$=""ELSEP$="P"
13 Z$=N$:GOSUB53:Z$=Z$+"/"+E$:GOSUB53:GOSUB52
14 B$=" "+Z$+Q$+" ("+G$+P$+")":F=1:GOSUB48
15 ON(INSTR(" CcAaRrDdQq-@".I$)+1)GOTO14,33,19,19,22,22,26,26,28,28,17,17,16,35
16 GET1,R:X=E:GOTO8
17 CLS:C$="DIR "+Q$:IFA$="v"THENCMD C$ELSEC$=C$+" ("+A$:CMD C$
18 C$=INKEY$:IFC$=""THEN18ELSECLS:GOSUB49:GOSUB52:GOTO14
19 C$="Copy ":PRINT@581,C$;:PRINT@610,CHR$(30);"to drive # ? <";:F=-
1:GOSUB36:IFI$="@"THEN35
20 IFI$=""THEN14ELSEIFI$=RIGHT$(Q$,1)THEN19ELSEK$=" :"+I$
21 IFP$="P"THENK$=K$+" (C=Y)":GOTO30ELSEK$=K$+" (C=N)" :GOTO30
22 C$="Attrib ":PRINT@579,C$;:PRINT@610,CHR$(30);"<V/I/X> ?
<";:F=1:GOSUB36:K$="":IFI$="@"THEN35
23 IFI$=""THEN14ELSEY=ASC(I$): IFY>96THENY=Y-32
24 IFY=86ORY=73THENK$=K$+" ("+CHR$(Y)+")":GOTO30
25 IFY=88THENK$=" (A=,U=,P=AL)":GOTO30ELSE22
26 C$="Rename ":PRINT@579,C$;:PRINT@610,CHR$(30);"  to  <";:F=12:GOSUB36:IFI$="@"THEN35
27 IFI$=""THEN14ELSEK$=" to "+I$:GOTO30
28 C$="Kill ":PRINT@579,"Delete ";:PRINT@610,CHR$(30);" OK ?
<";:F=1:K$="":GOSUB36:IFI$="@"THEN35
29 IFI$<>"Y"ORI$<>"y"THEN14
30 C$=C$+Z$:IFP$="P"THENC$=C$+".EZTO"
31 C$=C$+Q$+K$:PRINT@576,CHR$(30);TAB(10)"==> ";C$;
32 PRINT@785,CHR$(31);". . . E x e c u t i n g . . .":CMD C$
33 E=X
34 NEXTX
35 CLOSE1:GOSUB50:GOTO2
```

```
36 I$="":A=0:D=ABS(F):PRINTSTRING$(D,32);">";STRING$(D+1,24);CHR$(14);
37 W$=INKEY$:IFW$=""THEN37ELSEIFW$="@"THENI$="":D=1:GOTO47
38 IFD=ATHEN41ELSEIFF<0THEN40
39 IFW$<" "THEN41ELSEIFW$<CHR$(128)THEN47ELSE37
40 IFW$=CHR$(13)THEN46ELSEIFW$=>"0"ANDW$=<"7"THEN47ELSE 37
41 IFW$<>CHR$(8)THEN43ELSEIFA=0THEN37
42 PRINTCHR$(24);:I$=LEFT$(I$,(LEN(I$)-1)):A=A-1:POKE16418,32:GOTO37
43 IFW$<>CHR$(24)THEN45
44 IFA=0THEN37ELSEPRINTSTRING$(A,24);:GOTO36
45 IFW$<>CHR$(13)THEN37
46 PRINTCHR$(15);:RETURN
47 PRINTW$;:I$=I$+W$:A=A+1:IFD=1THEN46ELSE37
48 PRINT@576,CHR$(30);TAB(10)"==>";B$;:PRINT@617,"<";:GOSUB36:RETURN
49 CLS:X$=STRING$(34,58):PRINT@78,X$;:PRINT@142,":   LDOS   'C A R D'   UTILITY
:";:PRINT@206,":      by  Canadian-Micro        :";:PRINT@270,X$;:PRINT@389,"<C>opy
<A>ttrib        <R>ename      <D>elete";
50 PRINT@521,CHR$(31);STRING$(45,124);:PRINT@649,STRING$(45,124);
51 PRINT@983,"<@>  key to EXIT";:RETURN
52 PRINT@785,CHR$(31);"  <SPACE BAR> for next file";:PRINT@853,"<-> for previous
file":PRINT@918,"<Q>  for  directory";:GOTO51
53 IFRIGHT$(Z$,1)=" "THENZ$=LEFT$(Z$,(LEN(Z$)-1)):GOTO53ELSERETURN
54 PRINT@576,CHR$(30);TAB(10)"*****  Input Error  *****";:FORXX=1TO1500:NEXT:RESUMEl6
```

NewScript and The BASIC Answer

J.L. Latham l409 Evergreen Circle Midwest City, OK 73110

I'm sure you all are aware by now that NewScript from ProSoft is my own personal favorite word processor. There just isn't any job that I have asked it to do that it couldn't. Once again NewScript has come through like the true champ that it is.

For payment for my last journalistic adventure into The LDOS Quarterly, I received a copy of The BASIC Answer (TBA). A sweet package indeed. Without going into all the details, I'll just say that TBA provides you with a method of creating more readable source code for your BASIC programs. It pushes you toward a more structured programming method, and allows you to use either the standard BASIC interpreter OR a word processor to create your source code. Using a word processor for source code is the interesting part, and we will look at it here with NewScript in mind.

The TBA psuedo-compiler (well, I'm not sure what else to call something that takes ASCII text and converts it into interpreter executable code) insists that the source file be in "pure" ASCII format. The only characters with an ASCII value less than 32 allowed are those for the carriage return and line feed. A word processor such as Scripsit must be told specifically to save the text in that format, and the same goes for the BASIC interpreter. NewScript does not. NewScript saves all of its information in ASCII format without special prompting. Not having to remember any special commands is a real boon to such a a forgetful individual such as myself. Anytime I don't have to remember something it means I don't have to worry about forgetting it, and that's the way the whole world should work. After all, isn't that what computing is all about - letting a machine do the drudge work for you?

I began looking into the possibility of using NewScript with TBA almost immediately. I was a little surprised when I "compiled" a couple of their sample programs and ended up with a "direct statement in file" error on them. I then realized that they probably expected explicit carriage returns in the source code, so I used NewScript to edit the files, simply going to the end of certain lines and hitting the <ENTER> key. Once I had done this everything worked fine. As the TBA manual states, the source code must have a carriage return (read that as <ENTER> key) at least every 240 characters or the dreaded "direct statement in file" error will occur. That is an admonishment even to Scripsit users. Remember that one rule and almost nothing can go wrong.

Once I had determined that source code for TBA could be created effortlessly with NewScript, I then decided that calling up NewScript, editing and saving the text then going to the LDOS Command level and executing "TBA<ENTER>" to "compile" the code was a bit tedious, especially if I had an error in the code and needed to do some more editing. I found that it is entirely possible to edit your code with the NewScript EDITor, get a printout with SCRIPT, and "compile" the code to check for errors without ever leaving the protection of NewScript.

ProSoft has conveniently left slot #7 of their main menu unassigned to any specific task. In version 7.0, if you choose option #7 the program simply ignores your request. It then waits for another input from the keyboard by looping to line 70 of their menu driver program, which is written in BASIC. I have previously used slot 7 to perform such things as running other BASIC programs, and to access the G.E.A.P. Dot Writer program. I figured it wouldn't be too hard to get the program to access TBA for me either, and I was correct. TBA/CMD can be accessed via a CMD"TBA" statement from BASIC just as a CMD"DIR" could be.

Some minor program changes to the NewScript program NSINIT have to be done in order to accomplish this, but they are simple and are worth the time it takes.

In the October LDOS Quarterly I told you how to create a version of NewScript on a double density LDOS disk, and it turns out that those instructions also work with hard disks. So, if you haven't moved your NewScript files to a double density disk, get out the October issue and do it, and then proceed with the following modifications.

Boot your LDOS NewScript disk and when you are shown the main menu hit the <BREAK> key. You now have the NSINIT program in memory right in front of you. List the program from lines 100 on. My version ends with line 122. Add the following lines of code (my line numbers asssume that the program ends with line 122):

```
123 ONERRORGOTO125:CMD"TBA"
124 GOTO 126
125 RESUME 126
126 ONERRORGOTO0:GOSUB50:GOTO26
```

Now check out line 50. In my version of NSINIT it looks like this:

```
50 PRINT" Press <ENTER> TO CONTINUE...";
```

Scroll back up from that line until you find the line that ends with ON X GOTO 58,58,44,58,58,68,40,54,58. In my listing it was line 42. This is the line that controls the program flow based on your choice from the menu. The number we are interested in is the seventh one in the list. In my version it sends the program flow to line 40, as I indicated above. Change that seventh entry (the 40) to 123 and you will get everything headed the way it needs to be.

Believe it or not, one more change and we are ready. If your line numbers have been running with mine then look at line 32 of your program and you will see part of the the menu lising. One part of that line will be "7) (unassigned)". Change the "(unassigned)" to "PROCESS TBA SOURCE" and you are done with the required changes. Now SAVE"NSINIT/:0" to keep your modified version of NSINIT. Hope you enjoy the added convenience of being able to write and edit TBA source code, "compile" it, re-edit it, and even to write the documentation for it without ever leaving the confines of NewScript.

As a passing freebee I'll let those who want to add speed up board support to NewScript have that before I say goodbye. The changes are simple. At the end of line 1 of NSINIT add the OUT command that activates your speed up board. Look at the line mentioned above (line 42) that has the ON X GOTO statement in it and check the eighth entry in the line. In my listing that value is 54. Line 54 simply contains a CLS statement. Add the command that returns your clock to the normal CPU speed to the end of that line. Now when you return to the LDOS Command mode you will be back to the normal CPU speed.

This last change is of interest mostly to people who have boards (such as the Holmes which I use) that have several speed up (and slow down) commands other than a simple OUT254,0 0UT254,1 set of commands. You don't have to cut out the speed with the Holmes board, but some others may require it for proper disk operation.

EASY LSCRIPT

by James E. Bruckart

Like many TRS-80 computer owners, my first word processor was Radio Shack's cassette-based Scripsit. I used Lazy Writer with my first disk drive, but returned to Scripsit, albeit the LDOS patched version, to take full advantage of the LDOS and LScript.

Along the wandering path I developed an affection for a "logical" - mnemonic type - single character input for each word processing function. For example, type "I" to insert, "L" to load, and "H" for help. To obtain these functions and more, I use LSCRIPT with a special Key-stroke multiply filter "LSCRIPT/KSM" (listing 1). Most Scripsit functions are duplicated by depressing the <CLEAR> key and alphabetic key related to its name. LSCRIPT/KSM replaces Scripsit commands by passing the current LSCRIPT command value (i.e.- <CLEAR-1> for insert or hex character B1) when <CLEAR-I> is depressed. Likewise, "Load" is initiated by combining <SHIFT-ENTER>, <L>, and <Space Bar> (Spec Cmd, L, and space - Hex 1D, 4C, and 20).

LSCRIPT/KSM is created by typing the BUILD LSCRIPT/KSM (HEX) command and typing the response for each letter as noted in Listing 2. The letters "C", "F", "M", and "O" are unused and may be designated for "user-defined" functions. The Help feature (<CLEAR-H>) saves the current text in a file (TEXT/LCR) and displays the help file - HELP/LCR. After finishing with Help, <CLEAR-G> returns your text to the Scripsit buffer for processing.

To start word processing, I execute DO LSCRIPT. This JCL installs LSCRIPT/KSM, starts LSCRIPT, and allows an optional spelling check when LSCRIPT is finished. "1" selects spelling check, but any other key will finish JCL and return to LDOS Ready.

In summary, LSCRIPT/KSM lets you remove the stickers from your keyboard, and provides Help with a single keystroke. Most occasional users will find these mnemonic commands easier to remember, and each command - key combination more logical.

LSCRIPT/KSM Filter Data

```
A=>1B0D        I=>B10D        O=>0D          U=>B00D
B=>B50D        J=>AD0D        P=>1D50200D    V=>B60D
C=>0D          K=>B20D        Q=>1D51313B0D  W=>B40D
D=>B30D        L=>1D4C200D    R=>B80D        X=>B70D
E=>1D454E440D  M=>0D          S=>1D0D        Y=>B90D
F=>0D          N=>BA0D        T=>1D53200D    Z=>1A0D
G=>1D4C20544558542F4C43523B0D
H=>1D5320544558542F4C43523B201D4C2048454C502F4C43523B0D
```

This is the actual text file, HELP/LCR. Its contents are what will be shown on the screen when the <CLEAR><H> keys are pressed.

LSCRIPT Commands

```
A Top of Text  H Help         O              V Window
B Block        I Insert       P Print        W Word
C              J Page         Q Dir          X Exchange
D Delete       K Line         R Repeat       Y <--
E Exit         L Load         S Spec Cmd     Z End
F              M              T Save         BREAK Abort
G Get Text     N New Para     U -->
```

```
Comment: >* comment (must follow text boundary)
Document Info: Spec Cmd ? C ENTER - line # of cursor
                           L         length of text
                           M         available memory
Formatting text: >fc1, fc2, ...
                 PL = page length (1 to 90)
                 LM = left margin (0 to 131)
                 RM = right margin (1 to 132)
                 TM = top margin (1 to 89)
                 BM = bottom margin (2 to 90)
                 LS = line spacing (1 to 90)
                 PF = paragraph format (1 to 90)
                 J = justify text (Y or N)
                 C = center text (Y or N)
                 FR = print flush right (Y or N)
                 VC = center vertically (Y or N)
                 P = print text (Y or N)
                 WS = supress window (Y or N)
                 F = begin footer on page (1 to 65535)
                 H = begin header on page (1 to 65535)
                 PN = start page numbering (1 to 65535)
Global: Replace Text String - Spec Cmd R > text > new text
        Delete Text String - Spec Cmd D > text <ENTER>
        Find Text String - Spec Cmd F > text <ENTER>
Headers & Footers: Spec Char B, H, or F (Head or Foot), O, E, or S (odd, even, or all
pages), ENTER (terminate line), type text, ENTER, Block, End
Hyphenation: Block - Block End, at end of text - Spec Cmd H ENTER type number ENTER, to
remove - Spec Cmd H ENTER
Label Blocks: Block label Block End
Page Numbers: Block P # Block End
Printer: Control characters - >$ value, value,
         Pause comment - >#  comment string
Tab: Spec Cmd T = t1, t2, t3, ...
Video Info: Spec Cmd ? W - Video line width
            Spec Cmd W = (1 to 132) sets video width

Press <CLEAR> <G> to return to text
```

@PARAM , @DSPLY , @EXIT and INBUF$

Easy to use LDOS Routines for Everyone

by David Vinzant

LDOS provides users with several very useful and rather easy to use machine language subroutines. I had tried machine language several times but became frustrated with the amount of code required to do anything of real significance and restricted my programming to BASIC, which is slow but straight forward and easy to debug. After getting LDOS, learning the commands and the differences between LDOS and TRSDOS, I decided to find out what was in the Technical Information Section of the LDOS Manual. What I found was a description of all sorts of routines and memory locations that LDOS used and how to go about using them myself. The instructions looked so easy I decided to dig out my editor-assembler. I had always been intrigued with the speed of machine code, but had never taken the time to master it.

Several weeks earlier I had been talking with another new LDOS user and we were comparing notes on likes and dislikes. He had commented that he missed the HELP command that was available in TRSDOS 2.7DD. Having that comment in the back of my mind I quickly took notice of three routines and one memory location in the Tech. Section of the LDOS manual. These were the @PARAM, @DSPLY, @EXIT and INBUF$. With these I set out to write a HELP Command Utility. I decided on a format of HELP (command). I use the @PARAM routine to parse or look through my input line which was sitting in INBUF$. I

then built a table of the available commands for @PARAM to compare with my input line. This I would call TABLE1. The @PARAM routine requires the table to be in a special format. The command field must be 6 characters long, followed by a 2 byte field which tells @PARAM where to put the parameters into. @PARAM uses a 00H as an indicator of the end of the table. @PARAM can capture parameters in several ways. First if the parameter is entered in the format of PARAM=number, @PARAM will capture the number and put it into a specific place in memory. If the format is PARAM=yes/no/on/off, it will save 00H or FFH, 00H for off/no and FFH for on/yes. If the format is PARAM=string, it will save the location of the first byte of the string. If the format is PARAM with no equal sign, it will save FFH for that parameter. So I needed to set up a table for @PARAM to put its results in. This I called TABLE3.

The next routine I needed to look at was @DSPLY. With this routine, you simply load HL with the location of the first byte of the message and the when @DSPLY is called it will display memory starting at that location until it finds a 13 or 0DH as a terminator. So I needed a table of the locations of the first byte of each message. This I would call TABLE2. The program flows as follows:

1.Load the TABLE1 location into DE and the buffer location into HL.
2.Call @PARAM.
3.Clear INBUF$ so if next input has no parameter it is properly parsed.
4.Check for errors in the parameter entered (parameter which is not in the table).
5.Loop through TABLE1 and look for end of table (no parameter entered).
6.Loop through TABLE3 looking for an 0FFH (the parameter entered) (stored in TAB).
7.Load the location of the message into HL from TABLE2.
8.Call @DSPLY.
9.Call @EXIT (return to LDOS).

This was a great lesson in machine language programming for a beginner. First, I left all the complicated programming to LDOS. Second, I ended up with a program that was really useful. My final HELP Command Utility has all sorts of features including the following:

1. By entering HELP with no command the utility lists all the commands which HELP accesses.
2. The display for each command lists the command as it appears in the LDOS manual with all available parameters listed.
3. The display also lists the page number in the LDOS Manual where further details on the command can be found.

Listed here is a program for the beginner to use to get LDOS to print a line of data based on the command HELP (command) where command is either TEST1, TEST2, TEST3, no parameter or invalid parameter. This program can be assembled as any name but the name should be 4 characters in length. This is because I have offset the value of INBUF$ by 6 to get @PARAM to look at the first letter of the command entered (4 for HELP, 1 for space and 1 for parenthesis). It is also important to note that it is the <enter> , character 13 or 0DH, that @PARAM uses to terminate its look at the command line. In order to avoid a repeat of the last entry when a command is entered without a parameter, a 13 must be placed in the buffer. This was the only part of the LDOS documentation that I found missing. It didn't say where in the buffer to point to for @PARAM or what the @PARAM terminator was. It is also worth noting that @PARAM regards upper and lower case as the same, so it will find a match if your input is in lower case and your table is in upper case or the other way around. To get this program to run on a MODEL III you need only change @PARAM and INBUF$ to the values given in the manual for Model III remembering to offset INBUF$ by 6.

```
00100 @PARAM  EQU     4476H           ;define LDOS entry
00110 INBUF$  EQU     431DH           ;points and loc
00160 @DSPLY  EQU     4467H           ;for Model I
00170 @EXIT   EQU     402DH
00190         ORG     5200H
00200 ENTRY   LD      DE,TABLE1       ;table to look at
```

```
00210         LD      HL,INBUF$       ;buffer to look at
00230         CALL    @PARAM          ;parse command
00240         LD      A,13            ;0DH to clear INBUF$
00250         LD      (INBUF$),A      ;clear INBUF$
00410         JP      NZ,ERROR        ;check for bad parm
00420         LD      HL,TABLE3       ;indicator table
00430         LD      (TAB),HL        ;into TAB
00440         LD      DE,TABLE2       ;message table bc
00450         LD      HL,TABLE1       ;parse table bc
00460 LOOP    LD      BC,8            ;length of command
00470         LD      A,00H           ;check for end
00480         CP      (HL)            ;of parse table
00490         JP      Z,NOPAR         ;no param entered
00500         ADD     HL,BC           ;inc parse table
00510         PUSH    HL              ;save table bc
00520         LD      HL,(TAB)        ;load md table
00530         LD      A,0FFH          ;check for hit
00540         CP      (HL)            ;in @PARAM
00550         JP      Z,MSG           ;print message
00560         LD      BC,2            ;length of word
00570         ADD     HL,BC           ;inc indicator bc
00580         LD      (TAB),HL        ;save indicator bc
00630         EX      DE,HL           ;exch locations
00650         ADD     HL,BC           ;inc message bc
00660         EX      DE,HL           ;re-exch locations
00680         POP     HL              ;recover parse bc
00690         JP      LOOP            ;check next command
00840 ERROR   LD      HL,ERMSG        ;loc of error msg
00850         JP      MSG2            ;message displayer
00860 ERMSG   DEFM    'Invalid parameter'
00870         DEFB    13
00880 NOPAR   LD      HL,NOPARM       ;loc of no parm msg
00890         JP      MSG2            ;message displayer
00900 NOPARM  DEFM    'No parameter'
00910         DEFB    13
01690 MSG     LD      HL,TAB          ;load storage bc
01700         EX      DE,HL           ;into DE
01710         LD      BC,2            ;length of bc
01720         LDIR                    ;put msg bc in TAB
01730         LD      HL,(TAB)        ;loc of msg
01780 MSG2    CALL    @DSPLY          ;display message
01790         CALL    @EXIT           ;return to LDOS
01800 TABLEl  DEFM    'TEST1 '        ;parse table
01810         DEFW    TABLE3
01820         DEFM    'TEST2 '
01830         DEFW    TABLE3+2
01840         DEFM    'TEST3 '
01850         DEFW    TABLE3+4
02860         DEFW    0
02861 TABLE2  DEFW    WRD1            ;table of msg bc
02862         DEFW    WRD2
02863         DEFW    WRD3
02865 TABLE3  DEFM    'XXXXXX'        ;indicator table
02870 WRD1    DEFM    'Response to TEST1 input'
02880         DEFB    13
02950 WRD2    DEFM    'Response to TEST2 input'
02960         DEFB    13
02990 WRD3    DEFM    'Response to TEST3 input'
03000         DEFB    13
05780 TAB     DEFW    0               ;storage loc
05790         END     ENTRY
```

..... er .....

by Earle Robinson

The late appearance of the Quarterly fitted in well with my timing for this column. Plagued with landslides, and having moved houses and office, I really understand how much all those back issues of Byte & 80 Micro weigh(!) and appreciated the slimmer dimensions of 80 US. Speaking of 80 US, after having despaired as it seemed to float first from being an excellent technical oriented publication, then becoming a too elementary one, I am happy to note that the editors now have found a formula which should please most readers. They are also scrupulously avoiding the sort of editorials which have hurt 80 Micro's image, and reduced its readership and advertising.

I am writing this on the RS Model 100, a superb example of what a hand-held computer can and should be. It has an 8 line, 40 characters per line display. It also contains a built-in modem and text editor (which I am using) and which takes some getting use to! However, I was somewhat chagrined to learn that the acoustic cable, a necessity if you wish to use the built-in modem where there are no convenient phone jacks, which means 99 per cent of hotels and everywhere outside the U.S., will not be available until late Summer. Caveat emptor!

I shall upload it all into my trusty old Scripsit, not the newer Super one, but the old version 1.0 augmented by extensive patching under LDOS to have become and remain a very good text editor and even an adequate word processor. Please excuse a little immodesty there since I was one of the major contributors to the patches.

It is a funny thing, that though we all have at one time or another bitterly condemned the quality of the Shack's software, Scripsit has retained its value. And, the new SuperSCRIPSIT is a very useful program, too. It is a far more advanced word-processing program, of course. Yet, it is a darned easy one to get to know and use. It only has two major defects. First, the very lousy code takes up much more space than it should and runs very slowly on any system other than one equipped with a hard disk. Secondly, it still ain't all that reliable. For reliability, you can't beat WordStar.

WordStar is written in 8080 code and was the first fully functional word-processing program after the path was blazed by Electric Pencil. Alas, this program has yet to be released for the Model I or III. There are beta copies out, but MicroPro, the publisher, is uncertain if there is a viable means of marketing Wordstar to the TRS80 community. The program is very difficult to learn, I might add. There are some 180 different commands, almost none of which are logical to the keyboard. I use it for long documents because I can't yet rely on SuperSCRIPSIT, alas.

For those of you who do acquire SuperSCRIPSIT, may I advise you to be careful what printer you buy to run with it. If you do buy some little used brand of printer you may be reduced to having to write your own driver for it. Several times a week I receive enquiries if my firm offers a driver for such and such a printer, and I am obliged to say there is not enough demand to justify the time and expense in writing such a driver. Either buy a RS printer or get one which is used widely such as the ProWriter, Nec 1023a, the Epson MX series, or the C. ITOH F10 in the area of daisy wheels.

Speaking of printers, I had the opportunity of trying the 'son' of Mx80 the other day, called the FX80. It was a big disappointment. There is only one character set, though it can be used in 10, 12 pitch and proportional spacing, as well as in the so-called expanded & compression versions of each. But, the quality of the set is no improvement over the MX 80, which means that it is not as good as the ProWriter. A more serious weakness is that the micro spacing, when using the proportional option, precludes any known way of getting right justification in word processing. At first, I thought it was my own lack of imagination that prevented doing so. Then, I learned that the author of the widely admired Newscript has also given up and will not support this printer. I am surprised that Epson, after having created the first modern, low priced printer, has made such a disappointing successor to the MX line.

Some of you are also subscribers and regular users of the Compuserve and of the LDOS board there. For those who do not know of this board, I should like to suggest that you take a look at it. All LDOS owners who subscribe to the ESA, which means all who receive the Quarterly, may also join the LDOS board on Compuserve. A lot of valuable information gets released there first, there are interesting discussions...and some silly ones, too. However, it is a very valuable and rapid way of getting questions answered regarding not only the use of LDOS, but general and specialized programming or use problems.

Be forwarned, however, that the hourly charge of $5, which seems moderate can mount up rapidly if you become addicted to it. Try it anyway. At least, you don't have a large entry fee like Source which demands $100 to sign up. If you do use the LDOS board or others on Compuserve, try as much as possible to download messages into a file rather than printing them since transmission will be slowed somewhat. I should also recommend that you compose messages before accessing Compuserve, then upload them directly from disk, using whatever communication program, such as LCOMM, that you wish.

I am certainly no authority on the now becoming very fashionable 'C' language. Wishing to learn it, I first purchased the (in)famous Kernighan & Ritchie book, the bible to all 'C' users. Reading it is somewhat akin to reading the glosses to the Epistles of St. Paul, in Old Irish yet! However, help is at hand. The increasing interest in this language has brought forth a couple of new books. The best one, though outrageously overpriced, is the Plum book. Published by Plum Hall at $25, (see why I say it is overpriced?) directly from the publisher, or even more if you buy it elsewhere, as the publisher does not offer any discount for re-sale. It is a very good text to learn this language. However, a better bet for most of you may be 'The C Primer' published by McGraw Hill. It is well written and very accessible to those of who are either beginners to programming or who have relatively little experience. If you are a programming whiz, forget this book, and get the Plum and/or the Kernighan & Ritchie. As for the 'C Puzzle Book', it has errors but may interest you....., if you are a crossword puzzle fan.

Finally, a word of warning to assembly language programmers working with LDOS. The imminent release of LDOS 6.0, which will run on a screen of 80 x 24 and be a RAM based system, means that you should be careful to avoid direct peeks and pokes into display memory for you will risk compatibility problems when you convert programs to work under this new LDOS. Be aware, too, that 6.0 uses supervisory calls; you should convert whatever you are writing to use them rather than accessing the system vectors as we all have habitually done.

One more finally. My address is now 3WW Grenola, Pacific Palisades, CA 90272. Correspondence directed to the old address does reach me, but with some delay.

PARITY = ODD

Well, here I am again trying to sift through several thousand pounds of new software...if this keeps up, I can make a pretty good living just reselling the review material that is submitted to me! By the way, as of about 2 months ago, I've been using a LOBO MAX-80 almost exclusively for all of my computing chores. The more I use this machine, the more I feel I can't live without it. It is so FAST, you almost forget you're running a TRS-80 emulation. For those of you with MAXes, there is a new release of LDOS. The function keys are now recognized, and you hard disk owners will be able to boot from a HD partition. A few minor bugs were also cleaned up and the combination of LDOS and MAX-80 is a very solid, reliable one.

RESULTS OF THE PARITY=ODD Poll #1

Those of you that have been reading this column for a while remember that I asked for reader input on disk drive reliability several issues back. I finally managed to find time to wade through your responses and organize the data a little. First, let me say a big THANKS to all of you who wrote! Most of you not only included the drive

information, but a little about yourselves and how you use a TRS-80. All I can say is that the "hacker" is far from dead in the microcomputer industry.

It must be emphasized that the drive poll was not taken in a scientific manner, with controlled sample groups and so on. For this reason, don't take the results as "gospel". All kinds of sampling error can creep in a poll like this, and the intent was really just to get some general idea of how various drive brands compare. The numbers below show only the performance of 5" floppies. Some of you submitted 8" drive information, but there were not enough of you to include that information in the final data reduction.

Here once again are the general reliability categories established for the poll:

1 - Never Failed
2 - Failed after heavy use
3 - Failed once within 1st year of average use
4 - Failed more than once within 1st year of
average use.

AND FINALLY, the results (a drum roll please!)...

BRAND: MPI TOTAL NUMBER: 26	BRAND: TANDON TOTAL NUMBER: 19
CATEGORY #1: 15 57.69% OF 26	CATEGORY #1: 9 47.37% OF 19
CATEGORY #2: 1 3.85% OF 26	CATEGORY #2: 0 0.00% OF 19
CATEGORY #3: 7 26.92% OF 26	CATEGORY #3: 7 36.84% OF 19
CATEGORY #4: 3 11.54% OF 26	CATEGORY #4: 3 15.79% OF 19
BRAND: SHUGART TOTAL NUMBER: 18	BRAND: SIEMENS TOTAL NUMBER: 16
CATEGORY #1: 14 77.77% OF 18	CATEGORY #1: 13 81.25% OF 16
CATEGORY #2: 1 5.55% OF 18	CATEGORY #2: 1 6.25% OF 16
CATEGORY #3: 3 16.67% OF 18	CATEGORY #3: 2 12.50% OF 16
CATEGORY #4: 0 0.00% OF 18	CATEGORY #4: 0 0.00% OF 16
BRAND: BASF TOTAL NUMBER: 7	BRAND: TEAC TOTAL NUMBER: 2
CATEGORY #1: 2 28.57% OF 7	CATEGORY #1: 1 50.00% OF 2
CATEGORY #2: 1 14.29% OF 7	CATEGORY #2: 0 0.00% OF 2
CATEGORY #3: 4 57.14% OF 7	CATEGORY #3: 1 50.00% OF 2
CATEGORY #4: 0 0.00% OF 7	CATEGORY #4: 0 0.00% OF 2

One important thing to keep in mind, is that the fewer number of entries for a given brand, the less reliable the results are.

As I mentioned above, many of you added comments and questions to your entries to the poll. Several asked for help with specific problems ("How do I.....with LDOS and the tight writing schedule makes that impossible. A few of you also asked about my upcomming book on LDOS. It is not scheduled to be completed until later this year, and there will some delay after that getting it into print. When it does finally become available, rest assured you will be notified!

DEALS, DEALS, DEALS

As a public service, here are a few spectacular deals I've run across lately. If you're interested contact me on MNET or drop me a line.

********** FOR SALE **********

Speed-up kit for CRAY-I and CDC CYBER main CPU boards. Requires no trace cutting! Only 2500 solder connections to make.

TRS-80 Model I 4K With: Smal-LDOS, 8 Meg Winchester and Western Digital controller board, CP/M conversion, lower-case mod. installed, and RS232. Lots of software including communications software for async, bisync, HDLC, SDLC, and x.25 protocols.

Complete set of LDOS Quarterly magazines for the last 15 years. Includes ALL anniversary issues.

Am interested in starting a user's group to exchange ideas, programs, and hardware tips for ENIAC computers. Particularly interested in games, multi-user operating systems, real time control, and missle fire control software. Prefer originals with complete documentation. Also have some interest in developing user-friendly, menu driven, multi-tasking, systems software for Hollerith machines. Only serious replies please.

****

LETTERS AND CORRECTIONS DEPT.

In the last issue of this column I stated that Electric Webster does not check to see if a word you've entered as a correction is spelled correctly. Strictly speaking this is true. However, the manual does explain a way to get the program to do this. Personally, I think this ought to be a default condition. Speaking of Electric Webster, there is a new version of the product which works with virtual word-processors like SuperScripsit. Contact Cornucopia for more details.

Because of the disk drive poll, I've gotten a lot more mail than usual, and I thought I'd share a bit of it with all of you. Skip Blumenthal from New York comments, "Gold Plug 80's from EAP are a must." I agree! If you own a Model I, the 20 bucks or so for these gold edge connectors will save you a lot of accidental reboot problems. EAP advertises in the back of both 80-Micro and 80-U.S. so you'll have no trouble contacting them. I use a set of these between my Model I and LOBO LX-80 interface with excellent results. Richard Hertzel from Palm Springs writes that a SOLA constant voltage transformer solved many of his hardware related problems. Again, I can confirm this from personal experience. These transformers are on the expensive side, but you can sometimes get a good deal at a Hamfest or local surplus store. If you do buy one, be sure not to get a power rating too much higher than what you actually need. For example, if all your equipment dissipates a total of 400 watts, get the 500 watt model not the 1000 watt version. The reason is that this type of transformer works best when it is running near it's specified power rating. I've been using one here for about a year, and I don't even notice our "dirty" Chicago power anymore.

The most interesting letter came from Mike Johnson in British Columbia. He is the owner of an LX-80 and expresses his frustration with software that won't run on it. He suggests that I dedicate one of these columns to the LX-80. How about it LX-80 owners? Are there enough of you out there in reader land that would be interested? Let me know, and I'll be happy to do so. Mike also had a few pointed questions for me. In part he said:

"...I wonder sometimes about your objectivity...if a program is written by any BIG names it automatically gets a good review. A prime example is PMOD...the entire set of Powersoft utilities could be replaced by Trakcess at a fraction of the price and twice the performance. PRINT THAT!"

OK, Mike, I did! You may have a point. I see an awful lot of software in any given month. Sometimes I don't have the time to really dig into it in the detail I'd like. I suppose subconciously a "name" software author might get away with less scrutiny than an unknown. I try not to let that happen, and on occasion I HAVE taken very famous products to task for being poorly written or implemented or hard to use, or whatever (e.g. Electric Pencil 2.0, FORTRAN-80). In general, the approach I take when I do product reviews is threefold. First, does the product do what is claimed for it? Second, does it solve a real problem; does it DO something, or is it just another software "gadget"? Finally, is it being sold at a fair price? With regard to the Powersoft Utilities, I have several other thoughts. First, they run on almost every

LDOS system, Trackcess does not (Powersoft still hasn't gotten some of the utilties running properly on the larger hard disks). Second, Powersoft has just repackaged the utilties at a much more reasonable price - the original price WAS ridiculous. Finally, I always prefer products like the Powersoft utilties that work THROUGH the operating system rather than in spite of it.

REVIEW OF THE MONTH CLUB

This issue, I'd like to finish off the set of reviews on spelling checkers and perhaps draw a few comparisons between them.

The first one of interest is Proofreader from Aspen Software (now sold as the Random House Dictionary). This product is written entirely in machine language and consists of two parts, Proofreader and Proof-Edit. To correct a document you first pass it through Proofreader which in turn generates a list of unknown words. At this point, you can display the words on screen, send them to a printer, examine them and remove the ones which are not truly errors, create a file which contains the bad words, or exit to DOS. Typically, you first examine the words on screen, eliminate the ones which aren't bad, and then create a file of the remaining words. You then use Proof-Edit to interactively edit the words. Proof-Edit uses the bad word list created under Proofreader in conjunction with the file being edited itself. One nice feature here is that you can tell the program to accept an unusual word for the remainder of the session without having to permanently add it to the dictionary. As with the other spelling programs we've looked at, Proof-Edit allows you to look at words in context and correct them interactively. It does not check the spelling of your corrections however. Once you are through with the corrections, the program creates a new copy of your file with the corrections in it. The file name is the same as your original except that the last letter is incremented by one. TEST/DAT becomes TESU/DAT and so on. If by chance your filename ends in 'Z', then an error message gets displayed and you have to manually rename the source file and start all over again. This by the way apparently has some bugs. Proof-Edit kept telling me that a file called PARITY/SCR ended in a 'Z'!

This is an extremely well written product from a technical point of view. Of all the spelling programs I've looked at, it is the ONLY one that ran perfectly from day one under LDOS without bugs (other than mentioned above), patches, and other convolutions. It is reasonably fast and seems to do the job. However, the two part process of checking a document is very clumsy and is a serious enough flaw that I hesitate to recommend the product, especially for the novice user. Proofreader is available from:

Aspen Software
P.O. Box 339
Tijeras, NM 87059
(505) 281-1634

As of this writing I'm not sure what the present price is, so you'll have to contact Aspen for more details.

The last spelling checker I looked at is the Radio Shack Scripsit Dictionary. This is a very simple, straightforward product to use. It is intended to be used as both a stand-alone product for Scripsit users, as well as integrated with SuperScripsit. Since Tandy is still reworking the latter, I won't comment on how the integration works (Hint: Not very well with the original SuperScripsit release, but hopefully better when the reassembly is released!) The package works much like Hexspell in that you always correct errors interactively with the unknown word shown highlighted in context. When an unknown word is spotted, you can add the word to your list, correct it, or ignore it. If you add the word, you are asked if the letter 's' is an optional ending, and if so, that too is recorded. There are some interesting limitations with this program. First, you can only add up to 255 words to the user list in any one session. Moreover, that list is limited by the computer's memory, NOT available disk space. The upper limit is around 2000 words on a 48K system, which is probably enough for most people. Another limitation occurs when you are correcting a word. The Scripsit Dictionary will only let you key in alphabetic characters in the correction. No numbers or symbols are

permitted. Finally, you must have a X'00' terminator byte on your file, or this program "blows up" and recovers by returning to DOS Ready.

My feelings about this product are mixed. As usual, Tandy has ignored the DOS features, so your keyboard drivers will be unrecognized. On the other hand, the product does work with virtually no bugs. I'm told by some who have used this package more than I, that it misses some words, but I never noticed that problem. Scripsit Dictionary is available from Radio Shack stores for $149.

AND IN CONCLUSION

In the final analysis, my favorite of the bunch is Electric Webster from Cornucopia Software. It's fast, easy to use, and full of great features like displaying the dictionary. With a promise of a version for virtual word processors like SuperScripsit and Wordstar, I think it is the ideal choice for almost everyone. A very close second is Hexspell from Hexagon Systems. It IS capable of handling very large files. It would be my number one choice for applications like an office, in which the users were very unfamiliar with computers. This is principally because it is so easy to use and nearly impossible to "blow-up". In fact, the only objection I have to this product is that it is rather slow. Proofreader and Scripsit Dictionary are "me too" products. They do work, but you can do a lot better for less money. Remember, these are MY opinions, and I have been wrong (once or twice!). Before you buy, check around and talk with a few other people to get their thoughts. Well, that about does it for now. Now, back to the computer for some REAL hacking. Letsee, I left my LDOS master under that empty sixpack.....

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.

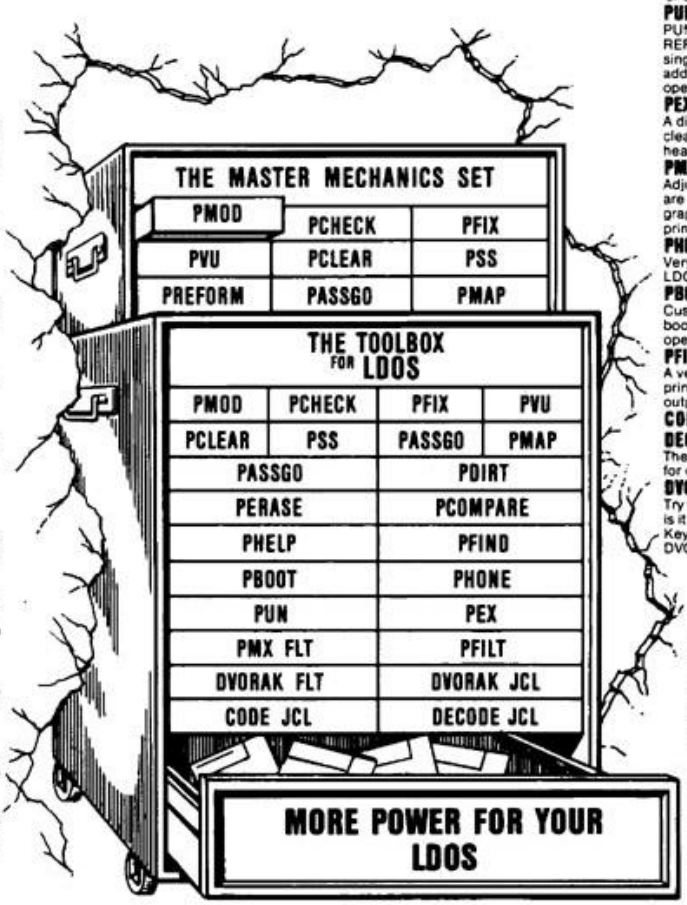
PSS/CMD
Sector Status. PSS allows you to easily identify which file is assigned to any sector or an entire diskette.

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
Read a TRSDOS Model III directory from LDOS.



PASSGO/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
Adjust 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
PASSGO/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

MISOSYS ANNOUNCES!

- Disassemble from disk / memory
- Disassemble to disk/printer/video
- Automatic output partitioning
- Full label generation
- Data area screening - generates DB, DW

- $40 + $2 S&H

MISOSYS
P.O. BOX 4848
ALEXANDRIA, VA. 22303
703-960-2998

FED

(The LDOS File Editor)

A NEW release of the LDOS File Editor now has built in Disassembler and direct disk modify by track and sector. It features an enhanced SEARCH mode as well as other new commands.

Original FED 5.1 ONLY - $19.00

FED II 5.1 - ONLY $39.00

FED 6.Ø - ONLY $49.00

Plus $3.00 S&H per package

The BASIC Answer

6.Ø version NEW RELEASE of The BASIC Answer. 5.1 version had overwhelming acceptance and praise by those who have become dedicated users. Designed to construct code in a structured manner. "Source" code is created with a word processor or text editor making use of powerful editing and movement features which are characteristic of such software.

Available for running with 5.1 LDOS for ONLY $69.00

NEW - version for running with LDOS 6.Ø - ONLY $79.00

Plus $4.00 S&H per package

BOTH PRODUCTS AVAILABLE FROM:

LOGICAL SYSTEMS INC.

LOGICAL SYSTEMS INC.

8970 N. 55th Street
P.O. Box 23956
Milwaukee, WI 53223
(414) 355-5454

What looks like COBOL, writes like a Word Processor, and runs like BASIC?

The BASIC Answer TBA

The BASIC Answer is a BASIC text processing utility. It is designed to allow the BASIC programmer to construct code in a structured manner. "Source" code is created with a word processor or text editor which allows the user to exploit with powerful editing and movement features characteristic to these types of software. Source code can also be created by means of a BASIC interpreter. TBA is then used to process this source code into ordinary interpretive BASIC code. TBA must be used exclusively with LDOS 5.1 operating systems.

TBA utilizes labels in lieu of line numbers. Branching in a program is accomplished by means of a descriptive label as opposed to an arbitrary line number. This means that blocks of code, subroutines, and procedures can be called and referenced by names which reflect their function, such as @ SORT.NAMES, @ FIND. MINIMUM, @ CALC.MEDIAN etc. Labels may be up to fourteen alphanumeric characters in length. This allows totally relocatable BASIC routines without the renumbering problems.

TBA supports variable names of up to fourteen significant alphanumeric characters. This means that cohesive descriptive names can be applied to variables in order to greatly augment program readability and comprehension, especially in the case of code which has not been examined for a long time.

For example, a typical TBA statement might be:

IF ACCNT. OVERDUE # > Ø THEN
GOSUB @ PRINT.WARNING
rather than
IFA1 # > Ø THENGOSUB51ØØØ

Clearly the first line contains a veritable wealth o' information when compared to the second.

TBA introduces the concept of "Conditional Translation". This feature allows co-existence of "machine-dependent" code within the same source. TBA can be instructed to ignore the irrelevant sections when processing.

TBA allows use of Global and Local variables (psuedo). Local variables are those which retain their value only in a unique subroutine. This means that variable tracking and conflict problems are minimized.

The BASIC Answer combines the self-documenting benefit of COBOL with the casual structure of BASIC in concert with the editing power of a word processor. Truly a timely combination.

TBA runs on the TRS-80 Model I or III and LOBO's MAX-80. Requires the LDOS 5.1 Operating System (LDOS must be purchased separately). Contact your local dealer or Logical Systems for more information. To get **The BASIC Answer** order catalog #L-21-030. $69.00 plus $4.00 shipping and handling.

* TRS-80 is a trademark of Tandy Corp. * MAX-80 is a trademark of Lobo Systems, Inc. QuizMaster is a product of Logical Systems, Inc. Price and specifications subject to change without notice.

QUIZMASTER

QuizMaster is an educational/information question and answer program that can also be used as a game. Its basic operation is to display a question and four possible answers. It scores the operator's response based upon the speed as well as correctness from one of three skill levels.

QuizMaster randomizes the order of the answer to prevent memorization. The question sequence is never the same. Extended play provides a "sudden death" feature for the skillful user.

QuizMaster comes with three subject files of 1ØØ questions each; U.S. Information, General trivia as well as Fantasy and Science Fiction trivia. These files can be increased or edited, or the user's own specialty files can be created and utilized. Each file can hold up to 255 question/answer sets and the only limit to the number of files is the number of diskettes you possess.

QuizMaster is educational, interesting and addictive. QuizMaster runs under the LDOS operating system (not included) to utilize maximum efficiency. The QuizMaster system includes all the facilities necessary to establish and maintain a series of multiple choice questions on any subject. The system is comprised of several machine language modules for fast access and response times.

WORD PROCESSOR-LIKE INPUT EDITOR

For ease of entry an "input editor" allows full transparent cursor motion along with insert and delete modes, type over and fast cursor positioning. This feature is found in both the "Add" and "Edit" modes.

FIVE SUPPORT PROGRAMS INCLUDED

Five support programs are provided to create, extend, edit, print and maintain question/answer files. Also included is a program to reconstruct a file that has been damaged by disk I/O errors or faulty disk media. A packing module allows files that have been heavily edited to be compressed and use disk space more efficiently.

All features are easy to use and easy to operate. Everybody loves trivia and now you can control it.

Other uses include:

- **classroom testing**
- **procedure quizzes**
- **product knowledge and**
- **group entertainment**

QuizMaster runs on the TRS-80 Model I or III and LOBO's MAX-80. Requires the LDOS 5.1 Operating System. (LDOS must be purchased separately.) Contact your local dealer or Logical Systems for more information. To get **QuizMaster** order catalog #L-51-500. $39.00 plus $4.00 Shipping and Handling.

For information or ordering call (414) 355-5454

LOGICAL SYSTEMS INC.

8970 N. 55th Street
P.O. Box 23956
Milwaukee, WI 53223

* TRS-80 is a trademark of Tandy Corp. * MAX-80 is a trademark of Lobo Systems, Inc. QuizMaster is a product of Logical Systems, Inc. Price and specifications subject to change without notice.

THE 'C' LANGUAGE (Part II)

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

Last time, in Part I, I gave a very brief history of C, discussed its use of storage (data or variable types) and introduced you to some other C language concepts.

In this article, Part II, I will discuss more on functions. Also introduced will be expressions and operators. A sample program will be presented to demonstrate some of these newly introduced concepts.

First, more on FUNCTIONS! Functions in C are analogous to subroutines in other programming languages. They conveniently group commonly used expressions together. Frequently used logic instructions, instead of being typed in multiple times throughout the body of a program, need only be typed in once as a function. The function can then be called whenever its logic process is needed. It is thus that functions can hide confusing details in the main body of the program. Following the main logic flow is then much easier. The programmer can see immediately what is going on and yet need not be concerned as to how things are being done (C lends itself well to structured programming techniques).

In C, a function, like a variable name, has associated with it a storage class and type. A C function with no explicit declaration is by default external (extern). External functions can be called from multiple source files. A function may also be declared as static. Remember, if declared as static, a function can only be called from within the source file where it is typed in (defined). The type of a function can also be specified. Are you thinking that functions are a collection of expressions, and wondering how can they be assigned a storage type? Well, actually, the type associated to a function refers to the value, if any, it returns. Functions, unlike variables, need not but can be declared before they are used. The C compiler knows the difference between an undeclared variable and a function by the left parenthesis '(' immediately following the function name. Here are some examples of declaring a function:

```
static char lnth(a,c);
            test();
```

The function lnth is declared to be only known within a single source module and optionally returns a character value. The function test is known amoung multiple source modules and optionally returns by default an integer value. It could have also been declared as follows:

```
extern int test();
```

The optional return statement is how the function returns a value back to its caller. It is optional; i.e., the function does not have to pass back a value. Even if the function does not return a value back to its caller, it is good programming practice to include the return statement. If no return is found, control of the function "falls thru" to the end of the function by reaching the ending right brace. Any expression (value) can follow the return statement. Examples of the return statement:

```
return;
return (0);
return ('x');
```

The first example of the return statement returns without passing back a value. The return (0); which could also be written as return 0; returns back the value zero. The last example returns the character 'x'. Although we haven't learned about the expression below, I'll let you ponder over it just to show how an expression can be used in the return statement:

```
return (a ? b : c);
```

Functions can also, optionally, be passed parameters. (Remember from last time that these parameters are passed by value rather than by reference. Except for arrays, and other parameters representing addresses of variables, each function gets its own private copy of the variable.) These parameters are enclosed by the mandatory parentheses denoting a function. It is the parentheses immediately following a variable name that denote it as a function rather than a variable (data type). The optional parameters passed to a function can be declared by default but it is a better programming practice to explicitly declare them.

The left and right brace {}, which enclose the expressions or logic performed by the function, are the next ingredients to completing a function. The expression(s) enclosed by the braces is/are called a block. (Blocks are not necessarily limited to use in functions. As we will see later, they are also used in the main body of a program. Enclosing braces are merely used to associate an expression or groups of expressions as a single entity.)

A complete example of a function is as follows:

```
prtval (c)
   int c;
   {
    printf("The parameter value passed was %d",c);
    return;
   }
```

The function prtval is passed a parameter "c" which is declared to be an integer. Within the braces of the prtval function, another function printf is called. The printf function is part of the standard C function library. It is passed two parameters in this example. The address of the string of characters enclosed in double quotes is the first parameter. The second parameter passed to printf is the value of c. (More will be discussed later about printf and standard C functions.) If necessary, functions can call themselves. This process is called recursion.

As mentioned in Part I, the main body of a C program is itself a function called main(). It can call functions as needed to perform designated tasks. This main function (the program itself) can have parameters passed to it. If parameters are passed to the C program, the following syntax is used to declare them and the main() function itself:

```
main (argc,argv)
  int argc;
  char *argv[];
  {
     ... program statements
  }
```

You are probably wondering, why only two parameters inside the () after main? Do I hear you asking what happens when I type in a command to tell the operating system to execute my program and I pass it more than 2 parameters? You should be asking! For example:

```
myprog p1 p2 p3 p4
```

Since this is a command line given to the operating system, there are actually 5 parameters or arguments. The program name itself is the first parameter and the 4 others are: p1, p2, p3 and p4. The main() function for the C program, myprog, would be coded as is done in the example shown above. The declarations for the variables argc and *argv suffice for all parameters passed to the C program in the command line. The variable argc contains the number of parameters passed (including 1 for the program name itself). The variable argv is actually a pointer to an array of pointers. Each

element of the array (one for the program name and one for each successive parameter) is actually a pointer to the parameter. The * before the variable argv is a unary operator. It says that the variable it preceeds contains an address. It uses this address to fetch the contents stored there. What it actually fetches depends on the variable's declared storage type. More will be said about arrays and pointers a little later after a few more concepts have been introduced. You'll see the argv and argc parameters in use in the sample C program below. Since we are talking about the command line, a very useful feature is available through the special characters < and > typed in on the command line. This feature of the C language is called I/O (input/output) redirection. It allows C programs (or C program users which take advantage of this feature) to be file and/or device independent. The C compiler provides a simple mechanism for character at a time input and output. It does so by providing a mechanism for "standard input" and "standard output". Each of these two mechanisms or "files" is usually, by default, set up to be the user's terminal. The term redirection is used when the default terminal standard input or output is redirected to or replaced by a file. The redirection occurs via the > and < symbols preceeding a file name. The < is used to redirect standard input. The > is used to redirect standard output. Two functions from the standard library are available to a C program to perform the character at a time input or Output. The getchar() function performs the input and the putchar() function performs the output. Here again is the previous command example (from above) with the I/O redirected:

```
myprog p1 p2 p3 p4 <infile >outfile
```

When myprog uses the getchar() function to get an input character, instead of getting it from the terminal, the character will come from the file "infile". Likewise, when myprog uses putchar() to output a character, the character will be output to the file "outfile" instead of the terminal. The number of parameters passed in the command line is the same, i.e., the I/O redirection strings "<infile" and ">outfile" do not count in argc and are not contained in argv! The C program, myprog, is not even aware that the redirection has taken place. (INDEPENDENCE!)

Let's move on to the next topic, that of variables, constants, expressions and operators. Variables in C, and in any language, are used to manipulate data in storage. The naming convention for variables, their storage class and their storage type was introduced in Part I. A variable name is composed of letters and characters and optionally the '_' character to improve readability of the name, as you can see:

```
char byte_of_storage;
```

Don't forget to define (declare) variables before you use them. Also, don't choose a variable name that is the same as a C reserved word (keyword or statement). If you do, don't worry, the compiler will tell you!

Constants are used in C, for the same reasons they are used in other languages. C allows for several types of constants, i.e., number, character and string constants.

It is usually good programming practice to use a special feature of the C compiler to define constants. As you'll discover, "equating" a constant to a name wjll make it easier to change later on. Just update it where it is "defined" and every occurence of it will also be updated automatically by the C compiler at compile time. Here are some examples of the C compiler directive #define:

```
#define MAXIMUM 1000
#define CLEAR  0x01C9()
#define CR  015    /* Octal value for a carriage return */
```

Note that constants, if given a name, are generally by convention represented by upper-case names. The #define compiler directive functions to the C compiler as an EQUATE directive functions to an assembler. (One line macros!) In the above examples, wherever MAXIMUM is found in the C program it is replaced with 1000, likewise CLEAR is replaced with the function call to address 0x01C9. Hex number constants are preceeded with a 0x

or 0X. Octal number constants are preceeded by just an 0. Another compiler directive can be used to include a file containing multiple #define statements. For example:

```
#include "file.ext"
```

This tells the compiler to include the contents (statements) in the file "file.ext". Usually there are many "standard" constants, declarations and/or expressions which you will include in most all of your C programs. This being the case, included with a C compiler is usually a "standard header" file of the most common constants. This file is what you will #include in most of your C programs.

A character constant is formed from a single character enclosed in single quotes. Certain special characters are represented with an "escape sequence" to denote it as a special character. Here are some examples:

```
'x'    /* single lower-case character x */       '\n'   /* newline    */
'A'    /* single upper-case character A */       '\r'   /* return     */
'\n'   /* newline   */                           '\0'   /* null       */
'\t'   /* tab       */                           '\015' /* return     */
'\\'   /* backslash */
```

The last example shows how to generate any character you want. It is the \ followed by 1 to three octal digits. A string of characters is represented by characters enclosed in double quotes. In C, a string is always terminated with a NULL or '\O' character. It need not be typed in the string itself since the C compiler adds it on automatically. An example?

```
"Earl C. Terwilliger"  /* My name as a string */
```

Note: the above string has a length of 19 and a size of 20. Don't forget the terminating NULL added by the C compiler!

Operators are the next topic. They specify what is to be done to variables and constants. Operators when combined with variables and constants are called 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.) The actual workings of each operator (with some sample expressions) will be the topic of Part III. Here is the chart:

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

Before I tell you goodbye until next time, I have a sample C program for you to look at. It is a multiple file KILL utility program. The syntax for running it is:

```
KILLEM file1/ext file2/ext file3/ext
```

Each file name passed to it will be a parameter. The KILLEM program will build the LDOS KILL command and pass it to LDOS to execute for each file name parameter. The commands built and passed to LDOS for the above example would be:

```
KILL file1/ext
KILL file2/ext
KILL file3/ext
```

As you look at the sample program below, review the concepts learned in Part II. Not all of the programs statements will be clear, but see how much you can understand. (Are you looking at the comments?) See you next time!

```
/* Multiple File KILL Utility                                  */
/* Author - Earl C. Terwilliger Jr.                            */
/* Written for LDOS and the LC C compiler                      */
#include stdio/csh            /* LC standard header file     */
#option INLIB                /* Special LC compiler option  */
#define CLEAR (0x01C9) ()    /* Rom call to clear screen    */
main(argc, argv)
     int argc;               /* Declare parameters          */
     char *argv[];
{
 int c, rc;                  /* declare AUTOMATIC variables */
 char buf[100];
 CLEAR;                      /* Call to CLEAR function      */
 puts("Multiple File KILL Utility  by Earl C. Terwilliger Jr.\n\n");
 if (argc < 2)               /* Must have at least 2 parms  */
  {
    puts("Syntax: KILLEM file1 file2 ...     \n");
    exit();                  /* Exit back to LDOS           */
  }
 while (argc > 1)            /* More parameters still?      */
  {
   buf[0] = 0x00;            /* Set string to NULL ""       */
   ++argv;                   /* Go to next parameter        */
   strcat(buf,"KILL ");      /* Concatenate these 2 strings */
   strcat(buf,*argv);        /* Concatenate "kill" and parm */
   puts(buf);                /* Display the string on video */
   rc = cmd(buf);            /* Execute LDOS cmd and return */
   printf("\nReturn Code was %d\n",rc);
   --argc;                   /* One less to deal with now!  */
  }
}
```

ITEMS OF GENERAL INTEREST

Generally interesting this month is the following information. In the technical documentation for the @RAMDIR call, the option to get the free and used space is documented as needing a 4 byte buffer. This routine will actually require 16 bytes of buffer space, with the space information being put in the 1st four bytes.

For MAX-80 owners, a patch to SYS0/SYS to tab past location 63 is (X'213B'=7F).

In keeping with the way things are on the Model III, the MAX-80 printer driver converts linefeeds to carriage returns. If you wish to change that, apply the following patch to SYS0/SYS: (X'03A8'=4F 28 0B 00 00).

Allocate files anywhere

We get so many requests to have LDOS allocate files from track 1 on up, rather than use the random allocation it now has, that the following patch is being released. This patch is identical for Model I, III, and the MAX80. In the patch, the "01" byte is the value, in hex, of the track where the system will start looking for free disk space. You can change it to be any track value desired.

```
D00,FE=2E 01 00 00 00 00
```

The following patches are for the 5.1.3 release of LDOS, Models 1 and 3 and the Max-80. If your file dates are 02/05/83 (Mod 1), 03/20/83 (Mod 3), or 03/05/83 (Max-80), then all of these changes have been made.

SECTION 1 - Model III and MAX80

Following are the patches applied to the 8/25/82, 9/20/82, or 10/15/82 version of LDOS to make it equal the 02/05/83 release. These patches are for both the LSI and the Radio Shack versions. If you have a version dated earlier than 8/25/82, you should send in for an update rather than applying these patches. NOTE - some of these patches appeared in the Jan '82 Quarterly.

MODEL III

```
. FDCDVRA/F33 - 01/24/83 - BS
. Patch SYS0/SYS.SYSTEM - Model III only, NOT for Max-80!
. This patch will inhibit a 6 ms restore rate.
.
D04,03=09
.
. This part will cause the disk driver to use seek with
. verify on a read or a write if the head is to be moved
.
D04,15=FD 7E 09 BA 30 02 CB E9 FD 7E 04 E6 0F B1 C1 D3
D04,25=F4 32 23 44 F1 D0 FD CB 03 56 CC D4 45 C5 06 7F
D04,35=CD 60 00 C1 C9 FD BE 05 C8 FD 72 05 06 1C C9
D04,64=06 18 CD DC 45
. EOP
***************

. FORMATB/F3X - 01/05/83 - RS
.
. This patch will cause format to properly use write
. pre-comp.
.
X'60E5'=8B 67
X'678B'=FD 56 05 C3 08 64
. This patch allow settling before step (1.4ms) and
. after each step (18ms).
X'6115'=91 67
X'6791'=CD 03 64 C5 01 44 00 CD 60 00 C1 C3 FE 63
X'611F'=00 04
. EOP
***************

. LCOMMB/F33 - for release "U"
.
. This patch will prevent buffer overflow during FS or FR.
.
D07, B3=12
. EOP
***************
```

```
 . LBASICB/F33 - 01/23/83 - for release "U"
 . This patch will cause the &H constant to allow spaces
 .
 D08,7F=D7 00
 . EOP
 ***************

 . SYS12D - 02/05/83
 . Corrects minor problems in RAMDIR
 .
 D00,98=87
 D00,D9=72
 D00,E6=00 00 00
 D01,2C=00
 . EOP
 ***************
```

SECTION 2 - Model III

For MODEL III with file dates earlier than 03/20/83, apply the following patches:

```
 . SYSTEM/FIX - 03/16/83 - Model III - Ver U
 . MODEL III patch to use Mod 4 Fast/Slow command
 . Not for the MAX-80!
 . Patch SYS7, and also apply SYS0B patch
 .
 D0D,A2=21 10 42 CB F6
 D0D,AE=21 10 42 CB B6 3A 10 42 D3 EC
 . EOP
 ***************

 . SYS0B/F33 - 03/16/83 - MODEL III - Ver U
 . Make Mod 3 use Mod 4 Fast/Slow command
 . SYSTEMB must also be applied to SYS7
 . Not for the MAX-80!
 .
 D0F,66=EC 78 21 10
 . EOP

 . LCOMMC/F33
 . Fix problem with buffer overrun in FR
 D05,1D=AF C9
 . EOP
 ***************
```

MODEL I

```
 . FORMATB/F3X - 01/05/83 - RS
 .
 . This patch will cause format to properly use write
 . pre-comp.
 X'60E5'=8B 67
 X'678B'=FD 56 05 C3 08 64
 .
 . This patch allow settling before step (1.4ms) and
 . after step (18ms)
 X'6115'=91 67
 X'6791'=CD 03 64 C5 01 44 00 CD 60 00 C1 C3 FE 63
 X'611F'=00 04
 . EOP
 ***************
```

```
. LCOMMB/F31 - for release "U"
.
. This patch will prevent buffer overflow during FS or FR.
.
D07,AE=12
.
. EOP
***************

. LBASICB/F31 - 01/23/83 - for release "U"
.
. Update the version and date message
D00,F9=33
D00,FE=35
.
. This patch will cause the &H constant to allow spaces
.
D08,6E=D7 00
.
. EOP
***************

. SYS12D - 02/05/83
. Corrects minor problems in RAMDIR
.
D00,98=87
D00,D9=72
D00,E6=00 00 00
D01,2C=00
. EOP
***************

. LCOMMC/F33
. Fix problem with buffer overrun in FR
D05,18=AF C9
. EOP
***************
```

Improved BINHEX/BAS

Following is a BASIC program called BINHEX/BAS. Is used to convert the hex code listings of programs into an executable file. The procedure to follow is:

1) Create an ASCII file containing the hex code as listed, leaving out the spaces. This can be done with the BUILD library command or any word processor that can save an ASCII file. Be sure to end each line with a carriage return, and do not have more than 254 characters per line.

2) The last byte in each listing should be the checksum. It will be proceded by an asterisk (*) This byte should not be typed into the ASCII file.

3) Run the BINHEX program, specifying the Hex to Binary mode. Compare the checksum generated by the BINHEX program to that in the listing. If they do not match, check the ASCII file to find the mistyped byte(s).

This program is SIMILAR to one listed in earlier Quarterlies. This version, however, does have the checksumming feature built in. Lines 240, 265, 270, 280, 300, 310, 320, 330, 465, 470, 515, 520, 530, 540 and 550 were added or changed.

```
10 REM -- Hex to binary/Binary to hex file converter
20 REM -- Tim Mann
30 CLS:PRINT:PRINT"Hex to binary/Binary to hex"
```

```
35 PRINT"    file converter":PRINT
40 CLEAR 5000
50 GOSUB 58000
100 PRINT "Type 1 to convert a binary file to hex"
110 PRINT "     2 to convert a hex file to binary"
120 PRINT:INPUT D
130 PRINT
140 ON D GOTO 400,200
150 GOTO 100
200 LINE INPUT "Hex file name: ";HF$
210 LINE INPUT "Binary file name: ";BF$
220 OPEN"I",1,HF$
230 OPEN"O",2,BF$
240 IF EOF(1) THEN 300
250 LINE INPUT#1,D$
255 IF D$="" OR D$="OK" THEN 240
260 FOR I=1 TO LEN(D$) STEP 2
265 DN=FND2(MID$(D$,I,2)) :SU=SU+DN
270    PRINT#2,CHR$(DN);
280    NEXT I:GOTO 240
300 IF SU>255 THEN CS=SU-INT(SU/256)*256 ELSE CS=SU
310 CS$=FNH2$(CS)
320 CLOSE:PRINT:PRINT"Done  -  Checksum = *"CS$:SU=0
340 GOTO 100
400 LINE INPUT "Binary file name: ";BF$
410 LINE INPUT "Hex file name: ";HF$
420 OPEN"RO",1,BF$,1
430 OPEN"O",2,HF$
440 FIELD 1,1 AS F$
450 FOR I=1 TO 30
455    IF EOF(1) THEN 505
460    GET 1
465    DN=ASC(F$):SU=SU+DN
470    PRINT#2,FNH2$(DN);
480 NEXT I
490 PRINT#2,
500 GOTO 450
505 PRINT#2,
510 CLOSE
515 IF SU>255 THEN CS=SU-INT(SU/256)*256 ELSE CS=SU
520 CS$=FNH2$(CS)
530 PRINT:PRINT"Done  -  Checksum - *"CS$:SU=0
550 GOTO 100
58000 DEF FNH1$(X)=MID$("0123456789ABCDEF",(X AND 15)+1,1)
58010 DEF FNH2$(X)=FNH1$(X/16)+FNH1$(X)
58040 DEF FND1(X$)=INSTR("123456789ABCDEF",LEFT$(X$,1))
58050 DEF FND2(X$)=FND1(RIGHT$(X$,1))+16*FND1(RIGHT$(X$,2))
58070 RETURN
60000 END
```

The VIDSAV program on Filter Disk #2 will abort with an error if a JCL file is used to install it. The following patch will correct this problem.

```
. VIDSAVA/FIX - 04/15/83
. Fix problem when installing w/JCL, PATCH VIDSAV/CMD
D00,1E=00 00 00
D00,45=00 00 00
```

The DIRCHECK program on Utility disk #1 could give a false error report on certain hard disks or double sided drives. Here is the fix.

```
. DIRCKA/FIX - 02/14/83
. Patch DIRCHECK/CMD on Utility Disk #1
. Fixes a problem with certain hard drives and
. double sided disks
D00,F3=FD CB 04 6E 20 15 FD CB 03 5E 28 12 5F FD 7E 07
D01,03=E6 E0 7B 28 09 CB 08 00 00
D01,10=00 00
```

The NODAM program published in the October '81 Quarterly would not work properly if an error was encountered when reading the source disk. The following is a corrected version. For those who are unaware of NODAM's function, it is a program to allow reading a TRSDOS 2.3 Model I disk on a Model III, LX-80 system, or a MAX-80 without the need to use REPAIR. To use it, issue the command NODAM :d, where "d" is the drive number containing the TRSDOS disk. Then, use the DEVICE command to log in the drive. You may then copy files directly off of the TRSDOS disk.

```
05 06 4E 4F 44 41 4D 20 01 02 00 52 E5 21 6B 52 CD 67 44 3A 25 01 FE 49 21 49 40 20 03
21 11 44 22 35 52 22 3E 52 E1 7E 23 FE 20 28 FA FE 3A 20 3D 7E FE 30 38 38 FE 38 30 34
D6 30 4F CD 8F 47 2A 00 00 01 2B 00 B7 ED 42 22 00 00 23 E5 21 E9 52 FD 7E 01 77 23 FD
7E 02 77 E1 FD 75 01 FD 74 02 FD CB 03 FE EB 21 D9 52 ED B0 C3 2D 40 21 C7 52 CD 67 44
C3 30 40 1F 4E 4F 44 41 4D 20 2D 20 4F 6C 64 20 44 61 74 61 20 41 64 64 72 65 73 73 20
4D 61 72 6B 20 72 65 61 64 65 72 20 2D 20 56 65 72 20 31 2E 30 61 0A 43 6F 70 79 72 69
67 68 74 20 31 39 38 32 20 62 79 20 47 61 6C 61 63 74 69 63 20 53 6F 66 74 77 61 72 65
2C 20 4C 74 64 2E 0A 0D 42 61 64 20 64 72 69 76 65 20 6E 75 6D 62 65 72 21 0D 18 08 00
00 05 4E 4F 44 41 4D FD CB 03 FE C5 CD 00 00 C1 C0 F5 7A FD BE 09 20 0E 78 FE 09 28 04
FE 0A 20 05 F1 3E 06 01 06 00 53 B7 C9 F1 C9 02 02 00 52
*25
```

LDOS: HOW IT WORKS - The REPAIR, CONV, and COPY23B Utilities

Moving files from other DOS'es discussed
or--- Yes, you can get there from here.

This month, we will discuss, among other things) the REPAIR, CONV, and COPY23B Utilities. First, let's look at the REPAIR Utility.

REPAIR (ALIEN)--- sounds pretty nasty, eh? Well, actually it's quite simple. LDOS maintains certain pieces of system information in different areas of the diskette. REPAIR (ALIEN) will modify these areas so that they conform to LDOS standards. The resulting diskette will be readable by LDOS. Immediately, you should make a backup to an LDOS formatted diskette, and retain the original as a master copy. When should it be used? Anyone with a Model III, a LX-80, or a MAX-80 will need to use it before transferring information from certain types of diskettes.

When is it necessary to use REPAIR?

Model I Owners: You will not normally need to use REPAIR for Mod I TRSDOS, but other Model I operating systems may require it. If you are having trouble reading a diskette, you may need to REPAIR it. To read Model III TRSDOS (assuming you have double density capability), use CONV, and do not use REPAIR.

Model III, LX-80, MAX-80 Owners: As a general rule, you will need to REPAIR any Model I non-LDOS diskette, and LDOS diskettes earlier than 5.0.2. Model III non-TRSDOS diskettes will probably also require repair. Model III TRSDOS should be CONVerted, not REPAIRed.

After REPAIRing a diskette, odds are that it will no longer be usable by its original operating system. If you have a Model I system, you can restore the diskette to its original condition-- see the article about OLDDAM, elsewhere in this issue. If you have one of the 'other' machines, and cannot afford to alter the diskette in question, see the artice about NODAM, also elsewhere in this issue.

Clear as mud, you say? Well, not to fear, as the General Conversion rules accompanying this article will clean up any confusion.

For example, let's say that we have a TRSDOS 2.3 diskette with a very valuable business program on it, and we need to move it to LDOS, for use on our Model III. First, boot with LDOS. Then, place the TRSDOS 2.3 diskette in drive 1. Type:

REPAIR :1 (ALIEN)<enter>

The TRSDOS diskette will now be readable by LDOS. Remember, however, that the diskette will no longer work as TRSDOS 2.3. COPY INVADERS/CMD:1 :0 (or other appropriate filespec) will move our valuable business program from the TRSOOS 2.3 diskette to our LDOS diskette in drive 0.

Now, how about CONV? CONV is used for the transfer of files from Model III TRSDOS 1.2 or 1.3 to LDOS. Model I owners must have some form of double density capability to use this utility.

Again, let's take an example. With our LDOS booted, and in drive 0, place the TRSDOS 1.2 or 1.3 in drive 1. Type:

CONV :1 :0<enter>

CONV will display the name of each of the files on the TRSDOS diskette, requesting permission to transfer it. A response of 'Y' will result in the transfer of the file, and 'N' will skip it. After all the filespecs have been displayed, control will be returned to LDOS.

But, you say, there I was, unaware of the ways of the world, running under Model III TRSDOS (before I knew about LDOS). I er, a, ...I created a data file on a TRSDOS diskette that's too big to fit on a LDOS minimum system diskette, and I only have two disk drives.

Oh? No problem-- a simple, little known use of the SYSTEM command will do it.

1) Place system files 1, 2, 3, 4, and 8 in high memory. First, clear out high memory. Eliminate all but absolutely necessary modules (PDUBL or RDUBL, for you Mod I users). Next, use the SYSTEM (SYSRES=1) command to place SYS1 in high memory. Repeat this for all previously named system files (see this month's JCL corner for an easy way to do this).

2) Format an LDOS data disk, and keep it handy.

3) With your LDOS system in drive 0, and the TRSDOS diskette in drive 1, type:

CONV :1 :0

4) When the filename is read, and you are prompted for permission to convert the file, remove the system diskette from drive 0, and replace it with the empty, LDOS data diskette. Respond Y <enter>.

5) At LDOS Ready, re-insert the LDOS system diskette.

That's it!

But what about COPY23B/BAS? Only use COPY23B/BAS on Model I TRSDOS 2.3B diskettes. Again, 'other' machine users must use REPAIR :1 (ALIEN), Model I users may proceed directly. LBASIC, COPY23B/BAS, and sufficient empty space must be present on your drive 0 diskette. Place the TRSDOS diskette in drive 1, and type:

```
   LBASIC RUN"COPY23B"<enter>
```

When prompted, give the source and destination filespecs. When 'Ready' appears, type CMD"S"<enter> to return to LDOS, or RUN<enter> to transfer another file.

General Conversion Rules:

Mod I TRSDOS	2.1	Mod I users may copy directly (write-protect first).
	2.2	Mod 3, LX-80, and MAX-80 users must REPAIR (ALIEN) the
	2.3	diskette first, and then copy any desired files. The diskette will no longer be usable by TRSDOS. See information about "Those Damn DAM's".
	2.3B	Use COPY23B/BAS (Model I ONLY!). If Model 3 version is present also, use that instead if possible. If TRSDOS 2.3B must be converted on a Model 3, LX-80 or MAX-80, you must REPAIR (ALIEN) first, and then use COPY23B/BAS.
	2.7DD	Get 2.8DD from Radio Shack.
	2.8DD	Copy to single density, 35 track diskettes, using the operating system in question, then treat as Mod I TRSDOS 2.3. Non-Mod I cautions apply.
DBLDOS		
DOSPLUS		
NEWDOS		
NEWDOS+		
NEWDOS80v	1.0	NOTE: must have 2 gran (single track) directory.
	2.0	
Mod 2 TRSDOS	2.0a	Use the READII/CMD program. See Utility Disk #1. Eight inch, double density drives required.
Mod 3 TRSDOS	1.2	Use the CONV/CMD program. Mod I users must have double
	1.3	density.
CP/M		See the CONVCPM product, catalog #M-35-220. Appropriate drive size and type required.

After transfer, all necessary modifications to ensure proper operation under LDOS must be installed by the user, and are the responsibility of the user. Some patches for Radio Shack software are available from LSI on our 'FIX Diskette'.

Roy's Technical Corner

Well, I'm back. For those that may have missed my column last quarter, rest assured that I was not idle. I'm sure that by now, everyone realizes that the LSI folks were hard at work designing and implementing the 6.0 version of LDOS. My job is to design the architecture of the DOS as well as oversee the implementation of critical modules. I also have a direct hand in implementing the kernel of the DOS and those modules that have a high degree of interfacing to that kernel. So you see, I have been busy. All this goes on while MISOSYS (my own company) activities take a back seat.

My wife, Brenda, has been handling the MISOSYS orders. If you have contacted us or ordered a product, you would probably have dealt with her. She should be on the job until our newborn arrives (expectation is late June). Very little software development has taken place at MISOSYS except, of course, the 6.0 project. We have found a little

time to convert products to run under the 6.0 system. If you are in commercial software development, you will find that 6.0 is very pleasant to work with. MISOSYS also had some software development activity handled by the local hackers - Washington D.C. area has quite a good stock of hackers. The latest piece of software is announced in this Quarterly - ZSHELL. This LDOS adjunct adds features that could become absolutely essential to your day-to-day operations. ZSHELL was written by Karl Hessinger and was inspired by some of the functions in LC.

Another new product is version III of the MISOSYS disassembler. This product has been long overdue. Since we wanted to be first with a disassembler running under LDOS 6.0, and since we did not want to release version II under 6.0, I was forced into pushing its development. Lo and behold, work went quickly. I am quite happy with the version III product. It supports direct disassembly from disk. It supports automatic partitioning of output files. It supports 100% labeling - before, after, and interstitially. It accepts a screening data input file to properly decode user selected regions as literals, byte data, or word data. We also kept the cost low.

I have also prepared the 6.0 compatible versions of EDAS-IV, PDS, and DSMBLR III, called PRO-CREATE, PRO-PaDS, and PRO-DUCE respectively. We hope to have a 6.0 compatible version of LC available soon - soon after the 1.0b version is finished.

Enough of this miscellaneous chit-chat. It just serves as a prologue to the primary discussion. Since I have found the conversion of 5.1 compatible software an easy task, I want to start presenting in this column, some of the design differences and interfacing differences between LDOS 6.0 and its predecessor series, 5.x. Please note that although the first announced 6.0 is named "TRSDOS", it was so named as a contractural obligation to the client - similarly to the IBM PC (PC-DOS versus MS-DOS). This column will continue to use the term "LDOS 6.x" as this publication is published by LSI. However, any reference to LDOS implies applicability to any LSI version 6.x DOS including TRSDOS 6.x.

Let me first pinpoint the primary design obligation since one key requirement was paramount throughout the entire design and implementation period. The goal of LSI that stands above all other goals is to ensure MEDIA COMPATIBILITY. This means that we need to be able to take a diskette and use it across all machines running LDOS - so long as the hardware permits that size diskette. Since day one, LSI has had a "standard" 5-1/4" structure - both single density and double density. We have also had a "standard" 8" diskette structure. The structure goes beyond just the format and allocation schemes - it covers the entire directory makeup. Thus, even though we may want to add a new feature (such as time-stamped directories), if it means changing the diskette so that it is unusable under 5.1 or 5.0, then it cannot be done. LDOS 6.x diskettes are directly usable under 5.x versions, and vice versa.

The media compatibility does not mean that programs executing under 5.1 will execute under 6.0 Lsee my later statements concerning SVC interfaced programs under 5.1]. The hardware architecture chosen for LDOS 6.0 is different than that running LDOS 5.x. LDOS 6.0 is designed to function on Z-80 based microcomputers with a minimum of 64K RAM and 80 by 24 video screens. The DOS is designed to operate starting from address X'0000' (page 0 origin). LDOS 6.0 is 100% SuperVisor Call (SVC) accessed. There is NO hard location for anything associated with the DOS except the hard locations for the RST Z-80 instructions and the NMI vector. Those data items felt to be needed by application software running under LDOS 6.x are available via SVCs and NOT by hard address. I cannot overemphasize this fact.

Functionally, LDOS 6.x is partitioned into seven regions: system low core (LOWCORE), Input/Output driver region (IOR), resident system (SYSRES), System Overlay Region (SOR), Library Overlay Region (LOR), User Program Region (UPR), and high memory region (HIMEM). The UPR extends from X'3000' through HIGH$. LDOS 6.x normally does not use HIMEM; however, certain user-specified requests must be satisfied by use of high memory. For example, SPOOL filter and buffer space use high memory. KSM filter and data space use high memory. These are uses corresponding to LDOS 5.x functions. Programs, similarly, must honor HIGH$ - which is available via an SVC.

The SVCs supported under LDOS 5.x are still supported - with three exceptions. The @CMD SVC (#23) has been dropped since its function was identical to @EXIT. The DIRCYL SVC (#83) has been dropped. Its function can be obtained by using SVC 81 which gets the Drive Code Table address for a logical drive into register IY which can then be indexed by nine to pick up the directory cylinder. The last difference is that the HIGH$ SVC (#100) has been altered to add a function code in register B. All other LDOS 5.x SVCs are supported. Device I/O handling has been changed so that the return code (Z/NZ indication) may mean something different. More on this later. More than seventeen SVCs have been added - some very significant ones.

Let me first discuss device I/O under LDOS 6.x. Since we are now controlling the entire device handling without a miscoded ROM in the way, device I/O has been improved. The return code used in byte I/O is identical across logical and physical devices. Thus, byte I/O via one of the logical devices (i.e. *KI, *PR, *SO, ...) is handled the same way regardless of the physical device identified in the Device Control Block (DCB) - be it physical keyboard, printer, or disk file. Applications are now expected to deal with return codes in all byte I/O. For example, A Z-status in response to an @GET indicates a character received without error (the same would be true for @KBD since that SVC uses @GET). An NZ-status with A=0 indicates no character received (i.e. NO ERROR).

Routing, filtering, and linking is now completely supported - devices may be routed to files and subsequently filtered and linked. A priority level has been established to the device TYPE byte bit assignments: file, NIL, route, link, and filter (file being the highest). The scheme employed in filter interfacing has been changed to improve the integration of filters into the device chain. Filters are assigned control blocks in the DCB table area. The DCB table now supports up to 31 entries. Each device driver and filter has its own entry. The establishment of a LINK also uses a DCB entry. Because of this, there is available technical documentation on the system which covers the device driver and filter templates to be used for user-written filters and drivers.

The primary console devices, keyboard and video, are generally not resident in memory-mapped space under LDOS 6.x. This means that applications cannot peek or poke the video screen or keyboard values. All *KI and *DO I/O is done through the system. In order to accomodate direct access of the video screen, a video control SVC (VDCTL) is available to PUT and GET characters by "row,column" notation. This would simulate the old peek and poke of the video. The VDCTL SVC also permits I/O transfer of the entire 2K block of video as well as cursor manipulations. Since keyboards can be matrix or ported, all key codes are input via the *KI device. This means a uniform set of codes for every application.

A significant change was introduced in the exit procedures of applications. Instead of just exiting via @EXIT or @ABORT, a single exit (@EXIT) is used with register pair HL containing a return code. A return code of zero indicates a successful execution. Any non-zero value in HL indicates an error exit. If executing from JCL, the exit condition is checked to determine if JCL should abort. All LDOS 6.0 supplied modules will exit with register pair HL set to either zero, the primary error number (1-63) if the exit is invoked because of a primary error, or negative one (X'FFFF') if some extended error has occurred. For the convenience of programmers, an @ABORT SVC is supported which just sets HL to X'FFFF' and goes to @EXIT. If an application properly maintains the stack pointer (SP), then it may exit via an RET instruction since LDOS pushes the @EXIT return address onto the stack before giving control to a program being executed from @EXIT or @CMNDI. LDOS LIBrary commands and most utilities also properly maintain the SP and are available for execution from within application programs by invocation via the @CMNDR SVC. If your applications are properly written, they too may be executed in this manner.

In order to prepare for hard-disk multiplexing, the directory maintains a "file open bit" for each file. This bit indicates that a file is already open for access greater than READ. Any subsequent attempt to OPEN a file with this bit set will force READ access (unless a lower access level is noted by the OPEN). An appropriate return

code is made to the OPENer. While I am discussing access, let me state that the terms "UPDATE" and "ACCESS" as applied to passwords have been dispensed with. They are replaced by the terms, "OWNER" and "USER". This is partly to minimize confusion with a new level of access which has also been introduced. Files may be protected with an "UPDATE" access level. This means that a file opened with UPDATE access may be READ from or WRITTEN to; however, the end-of-file (EOF) cannot be extended. Note that this also means that files opened with UPDATE or greater access - even if you only intend to read them - must be closed when you are finished with them. If not closed, the file-open bit remains set; thus subsequent OPENs get the "file already open" error. A LIBrary command, RESET filespec, has been added to "reset" that bit if a file was inadvertantly left open. Incidentally, a "global" password is not a part of LDOS 6.x. No longer can you use "RS0LT0FF" to access everything.

For those programmers interfacing from assembly language programs, the parameter scanner, @PARAM, has been improved. It now provides a response byte that indicates the type of user response to each parameter entered on the command line - be it numeric, string, or flag. If string, the length of the string is also returned. A new format is supported that provides a byte to indicate the length of each parameter word as well as the types of responses acceptable. A bit is assigned to indicate that a single character abbreviation is acceptable for the respective parameter. Thus, a more compact parameter table can be constructed to conserve space in your program. The LDOS 5.x table format is still supported via the same SVC.

LDOS 6.x now distinguishes between SYSTEM disks and data disks. Data disks may use all directory slots (except the two reserved for BOOT/SYS and DIR/SYS) for files. This provides for an additional fourteen files per DATA disk. SYSTEM disks are created during the process of backing up an existing SYSTEM disk to a DATA disk.

We now support a bank-switching SVC when the hardware implements memory bank switching. The SVC permits switching a memory segment (usually the top 32K) with up to seven auxiliary 32K memory banks. We also support the controlled transfer of execution to a location within the bank at the option of the user. The system maintains supervision of the resident bank to ensure that the standard bank (bank 0) is always resident during certain operations (byte I/O, disk I/O, and interrupt handling).

The bulk of LDOS 6.x is now machine independent. We expect to not alter LIBrary A, LIBrary B, or most utilities as 6.x is implemented on other machines. We have been able to accomplish this in large part due to the SVC interfacing structure. SYS8/SYS has now been made into a LIBrary module - LIB C. This incorporates members that are deemed to be machine specific code. For example, FORMS, SETCOM, SETKI, and SPOOL are members of this partitioned data set library. SYSGEN and SYSTEM are two other machine-dependent LIBrary commands. What this means to you as a programmer is that once you write an application to run under LDOS 6.x, it should run on any machine we put version 6. All you need do is utilize the standard interfacing procedures documented. Let the DOS do what an operating system is supposed to do - interface the application to the hardware.

The last thing to mention for now is that there is no KILL command in LDOS 6.x. I have perservered in my quest to make the system less violent. The command to get rid of unneeded files is "REMOVE". Enjoy LDOS 6.x. I will be discussing more details of this system in future issues of THE LDOS QUARTERLY.

THE JCL CORNER

By Chuck

Hello once again from the JCL corner of LDOS-land. This issue's column will cover a practical use for the compile phase of JCL, and also show an easy way to make backups with a JCL file - even though it may involve removing the disk containing the JCL file! As usual, I'll also answer some general interest questions received over the last 3 months.

To start things out, let me describe a practical use for a compiled JCL file, and explain how to build one. For those of you who are new to LDOS and JCL, the difference between an "execute" JCL file and a "compile" JCL file is as follows:

A file that contains no special JCL compilation macros can be executed directly. The most common type of execute file is one containing LDOS commands to set up a configuration or start up an application program.

A file that wishes to do logical testing, substitution, etc. using the JCL compilation macros MUST be compiled.

(See the DO Library command for instructions on how to compile a JCL file.)

Something that most of us with LDOS 5.1 or 6.0 have needed to do at one time or another is to reside certain system modules in memory. This is done with the SYSTEM (SYSRES=) command. However, if more than one module is to be resided, the SYSTEM command has to be repeated for each module. That takes many keystrokes and also introdcuces the chance for typographical errors to sneak in. Here, then, is a perfect candidate for a JCL file.

To start out, let's name this file S/JCL. Since the file is to be compiled, and there are many different system modules that can be resided, the //IF macro seems to be the choice for selecting desired modules. Briefly, the use of the //IF is:

```
//IF token        (if the statement is true)
these line(s) will go into the file
//END             (end of the //if checking)
```

The easiest way to build up the JCL file will be a series of //IF,//END blocks containing the SYSTEM command to sysres the desired module. Since one of the purposes of this JCL is to reduce keystrokes, some care should be taken when selecting the token names. For example, if you use tokens such as SYS2, SYS3, etc., then you would have to type in:

```
DO S (SYS2,SYS3,SYS8,etc.)
```

While this is easier than typing in separate SYSTEM commands, it can be made even easier if shorter token names are used. If you think that S2, S3, etc. is the shortest token that you can use - guess again. According to the definition of a token, it can consist of alphanumeric characters. Nowhere does it say that a token has to start with a letter! Thus, for ultimate ease of entry, our JCL file becomes:

```
//IF 1
system (sysres=1)
//END
//IF 2
system (sysres=2)
//END
etc.
```

Our DO command line can then be something like:

```
DO S (2,3,8,10)
```

In this example, it takes only 14 characters typed in to reside four system modules, versus 72 characters if done without a JCL file.

On to another slightly related subject. A user wrote in with the following question:

"I have a line in a JCL file that is BACKUP :1 :2 (DATE=#A1#). No matter what token value I specify for A1, I get a Parameter Error from BACKUP and the JCL aborts. I have tried A1=03/15/83- and A1="03/15/83-". What am I doing wrong?"

At first glance, the second substitution looks fine, where A1 is equal to the date enclosed in quotes. However, make a one line JCL file and compile it, listing the resultant SYSTEM/JCL file. It will show that NO substitution took place! Why, you may ask. As long as you asked, I'll tell you why. The value assigned to a token can consist of alphanumeric characters, as well as a slash (/), a period (.), and a colon (:). No other characters are allowed. This being the case, the solution for the problem is to rewrite the line as BACKUP :1 :2 (DATE="#A1#-"), and use the value of A1=03/15/83 on the command line.

As long as we're on the subject of the Backup utility, several requests have come for a method to do Backups with a JCL file, even though it requires removing the drive 0 disk or the disk containing the JCL file. Normally, this cannot be done, and will cause a system failure. The easiest solution to this problem is NOT to do the actual backup with a JCL line, but have JCL load a BASIC program to do the backup. For example:

```
LBASIC RUN"BACKUP/BAS"
//STOP

10 REM Make a Backup of the data
20 CMD"BACKUP :0 :1 (X)"
30 CLS:INPUT"All done - put original disk(s) back in, press <ENTER>";A$
40 CMD"S"
```

In this case, the BASIC program would be handling any prompting and/or error trapping. Although this is just a very simplified example, it will overcome the problem of using a JCL file to backup a data disk in a two drive system, etc.

JCL QUESTION OF THE QUARTER

The correct answer to last issue's question was that Doing the file with no tokens specified caused an abort. This was due to the fact that a //PAUSE line became the first line in the file, and we all know that is a no-no. The three winners were Ted Kingston (Strathmere, NJ), Richard Edgar (Madison, WI), and Al Brown (Tulsa, OK). By the way, I use a very scientific method to choose three winners from correct entries - I put them in a large trash bag, shake it up, and let three of the staff members reach in and pick out an entry. Anyway, here is this month's question. As usual, three of you who send in correct answers will get a free software package. The deadline will be around June 15th.

Here are several files, the purpose of which is to tell a tale. However, there seems to be something wrong...

```
//. Compiling the story...                    (This is the STORY/JCL file)
.
//IF TRSDOS
DO STORY (@WHAT,TRSDOS)
//END
//IF TRSDOS
@WHAT
.  I'm sorry - that option is NOT allowed!
DO STORY/JCL
//END
//INCLUDE LINE1/JCL
//INCLUDE LINE3/JCL

. Once upon a time, there were TRSDOS users,    (Build this as LINE1/JCL)
//INCLUDE LINE2/JCL

. who all got LDOS -                            (This is LINE2/JCL)

. and they all lived happily ever after,        (This is LINE3/JCL)
. using the LDOS JCL features!
```

As a footnote - something used in these examples is stated as NOT working in the JCL section of the manual. This is something that was fixed in version 5.1, but the documentation was not corrected. Can you find out what it is?

LES INFORMATION

by Les Mikesell

BYTE I/O (the easy way)

There are two different methods of file access available under LDOS, corresponding to the BASIC sequential or random access modes. The "sequential" type of access is somewhat slower, but is much simpler and within certain limits, the same routines may be used for both file and device I/O. As usual, the programmer has more control and more available functions when working in machine language rather than BASIC. This column will cover the techniques of sequential I/O, which may also be called "byte" I/O, since the characters are passed between the calling program a single byte at a time, and the operating system handles all of the "housekeeping" of buffering sectors and performing disk access when necessary.

With DE pointing to an open FCB or DCB:

```
        CALL  @GET
```

will return with a character in register A, and the status for the request in the Z flag.

```
        CALL  @PUT
```

will send the character in register A, and return with the status in the Z flag.

The following program will copy one file to another, using the @GET and @PUT calls. The filenames are passed on the command line.

```
@ERROR  EQU     4409H
@EXIT   EQU     402DH
@FSPEC  EQU     441CH
@GET    EQU     00l3H
@INIT   EQU     4420H
@OPEN   EQU     4424H
@PUT    EQU     001BH
@CLOSE  EQU     4428H
;
        ORG     5200H
;HL points to the next entry on the command line
BEGIN   LD      DE,FCB1       ;point DE at the source FCB
        CALL    @FSPEC        ;move to FCB
;HL now has moved to second entry
        LD      DE,FCB2       ;=>destination FCB
        CALL    @FSPEC        ;move name
;Set up to OPEN the source file
        LD      HL,BUFFER1    ;Pt to buffer
        LD      DE,FCB1       ;Pt to fcb
        LD      B,0           ;LRL
        CALL    @OPEN         ;open the file
        JP      NZ,IOERR      ;Jump on error
;Set up to INIT the destination
        LD      HL,BUFFER2    ;Pt to buffer
        LD      DE,FCB2       ;pt to fcb
        LD      B,0           ;LRL
```

```
        CALL    @INIT           ;init the file
        JP      NZ,IOERR        ;go if error
;files are open - start transferring data
LOOP1:  LD      DE,FCB1         ;source fcb
        CALL    @GET            ;input one character
        JR      Z,PUTIT         ;write it if read was OK
;got end of file or error if NZ - check error code
        CP      1CH             ;End of file?
        JP      NZ,IOERR        ;go if some other error
;end of source file, so transfer is complete
        LD      DE,FCB2         ;=>output fcb
        CALL    @CLOSE          ;must close to make directory entry
        JP      NZ,IOERR        ;go if error
        JP      @EXIT           ;all done, back to LDOS
;write to destination file..
PUTIT:  LD      DE,FCB2         ;=>output fcb
        CALL    @PUT            ;write byte (still in A)
        JR      Z,LOOP1         ;get the next one
;some error has  occurred - report it and quit
IOERR   SET     6,A             ;short msg/abort
        JP      @ERROR
FCB1    DS      32              ;space for file FCB'S & Buffers
BUFFER1 DS      256
FCB2    DS      32
BUFFER2 DS      256
        END     BEGIN
```

This program will accept a devicespec instead of a filespec for the destination of the copy, IF the output driver always returns with the Z flag set. This is not always true if the ROM printer driver is used for output, but if the PR/FLT or SPOOLer is active, the status conventions will be correct.

To load the contents of file into memory, the routine after the OPEN is simply:

```
        LD      HL,START        ;first address to store data
        LD      DE,FCB          ;=>fcb
INPUT   CALL    @GET            ;one byte from file
        JR      NZ,DONE         ;EOF or error
        LD      (HL),A          ;store in memory
        INC     HL              ;space for next byte
        JR      INPUT           ;get it
DONE    CP      1CH             ;was it end-of-file?
        JP      NZ,@ERROR       ;go if some other error
;file is loaded......
```

Since the status is tested after each input request, the system will always detect the exact end-of-file so the program does not have to check the FCB contents. (Assumming that the file was written via @PUT or the EOF offset byte was maintained properly in the FCB if sector I/O was used.) Of course, in a real application, it would be a good idea to check that HIGH$ is not overrun. The corresponding routine to write data from memory to a file would be:

```
        LD      HL,START        ;address of 1st byte to write
        LD      BC,COUNT        ;number of bytes to write
        LD      DE,FCB          ;=>fcb
OUTPUT  LD      A,B             ;check count of..
        OR      C               ;remaining bytes
        JR      Z,ALL           ;go if done (BC=0)
        LD      A,(HL)          ;load a byte
        INC     HL              ;point to next one
        DEC     BC              ;count down remaining
```

```
        CALL    @PUT            ;write it
        JR      Z,OUTPUT        ;continue if no error
        JP      @ERROR          ;quit if any error
ALL     CALL    @CLOSE          ;write is complete
```

If the program will deal only with files, the file positioning routines @POSN, @REW, and @PEOF may be used. For example, to make the above program append the data to an existing file, it would only be necessary to CALL @PEOF before the output. Writing to a file with @PUT will cause the end-of-file to be set at the last byte which was written when the file is closed, even if writing to an existing file which was previously larger. The positioning routines will generate an error if used with devices.

If it is necessary to determine whether a file or device is being accessed at runtime, the first byte of the FCB can be examined after a sucessful OPEN. Bit 7 will be a '1' if a file is open. Opening a device creates a DCB in the FCB space which is ROUTEd to the actual device control block (and thus will not have bit 7 set). Either a device or file may be accessed through the @GET and @PUT calls, but the status flag conventions at the return from the call are different. During file access, the Z flag will be set to indicate a good completion, while the NZ condition means the end of file was encountered or an error occurred. The LDOS output device drivers maintain the convention of returning with the Z flag set, but this is not a requirement for normal device operation and is not true for the ROM printer driver. Therefore, it is best to ignore the status after @PUT to a device:

```
        LD      DE,1FCB          ;may be for file or device
        CALL    @PUT             ;write a character
        JR      Z,GOOD           ;continue if status is OK
        EX      DE,HL            ;status is NZ, so check if..
        BIT     7,(HL)           ;writing to a file
        EX      DE,HL
        JP      NZ, @ERROR       ;abort if bad file write
GOOD    ; output was successful....
```

Allowing device and file interchangeability is slightly more complicated for input requests. In addition to the status conventions, there is the problem of determining when the input is completed, since the device will not give an end-of-file error, and will not always have a character available. Normally, input that would be desired from a device would be ASCII text, so some formatting conventions may be used to determine the end of an input (like a carriage return at the end). The LDOS drivers return with the Z flag reset (NZ) when a character is returned via @GET from a device. If the Z flag is set, no character was available. However, the ROM keyboard driver does not use this convention, so it is necessary to use an OR A to test if a valid character was returned. Otherwise, it would be necessary to require the use of KI/DVR to depend on the status returned from the driver. The following routine would input a line of data (terminated by a carriage return) from either a file or device:

```
CR      EQU     0DH             ;carriage return
        LD      HL,LINEBUFF     ;space to store input
        LD      DE,FCB          ;of OPEN file or device
INPUT   CALL    @GET            ;get (possible) character
        PUSH    AF              ;save returned char/status
        LD      A,(DE)          ;look at FCB
        RLCA                    ;is bit 7 set?
        JR      NC,DEVICE       ;go if not
        POP     AF              ;was file - restore char/status
        JR      Z,STORE         ;good status, keep character
        CP      1CH             ;EOF?
        JP      NZ,@ERROR       ;abort if disk error
        LD      A,CR            ;if EOF, put a...
        JR      STORE           ;carriage return in buffer
DEVICE  POP     AF              ;character / status
```

```
        OR      A               ;anything received?
        JR    Z,INPUT           ;no, try again
STORE   LD    (HL),A            ;put in buffer
        CP    CR                ;done?
        JR    NZ,INPUT          ;no, get next character
DONE    ; a complete line has been accepted...
```

Additional parsing might be added to this routine to discard invalid characters, process BACKSPACE (X'08'), limit the line size, echo to the video, and perhaps determine if the BREAK key was pressed. A "general purpose" input routine could be used to allow a program to accept certain inputs from a file or a device without needing to know ahead of time which will be used. For example, a file could be used to store control information used by a program. If the file does not exist, or the operator requests manual input, the program could simply OPEN *KI and use the same routines to get the information from the keyboard.

DISK I/O IN ASSEMBLER

by Doug Kennedy

This tale by Doug Kennendy describes his experiences in learning how to do sector disk I/O by using the LDOS system calls. The program he was writing turned out to be FED - the LDOS file editor.

In October of 1981, a few weeks after my initiation at Logical Systems (at that time Galactic Software), Bill Schroeder decided what my first big project was - a file editor. I knew that data was stored somewhere on disk, and that a disk was divided into concentric regions called tracks (cylinders), and each track was broken into 256 byte blocks called sectors, but that was it. I was a game writer, I had no idea how to access that information at a file level. When in doubt, READ THE MANUAL. That's what I did, but it didn't make much sense at the time. After writing the display layout & cursor positioning, the time had come for me to write the file handling routines.

First, I had to know what file to access. The best way I could think of was to prompt the user for the filespec (File specification). This was accomplished by using the @DSPLY message in conjunction with the @KEYIN keyboard input routine which inputs a line of data and transfers it into a buffer specified upon entry. My routine looked something like this:

```
FILEIN      LD      HL,PROMPT       ;Display "Filespec:" prompt
            CALL    @DSPLY
            LD      HL,FILEBUF      ;HL => Buffer for input
            LD      B,23            ;B => Max # of keys permitted
            CALL    @KEYIN          ;input filename/ext.password:d
            JP      C,@EXIT         ;return to LDOS if <BREAK>
```

After getting the filespec, it was then necessary to find out whether or not it was valid filespec. This is where @FSPEC comes in. This routine parses through a buffer (pointed to by HL) and indicates whether or not the contents of the buffer contain a legal filespec. If the filespec is legal, the Z flag will be set, and a copy of the filespec (converted to upper case) is placed into a 32 byte region (pointed to by DE) called a File Control Block (FCB). My routine looked like this:

```
CHECKFILE   LD      DE,FCB          ;DE --> File Control Block
            CALL    @FSPEC          ;Check filespec in FILEBUF
            JP      NZ,ILLEGAL      ;NZ - Display Illegal Filename
```

One neat feature of LDOS is the capability of default extensions. For example, to execute a command file such as FORMAT/CMD, one need only type FORMAT instead of FORMAT/CMD at LDOS Ready. This is because the LDOS command interpreter uses a default extension of /CMD. By using the @FEXT (Fetch EXTension) routine, a default extension

will be concantenated to the filespec in the FCB. To use the routine, point DE to the FCB with the filespec contained in it, point HL to the 3 byte default extension (in upper case) left justified with spaces. After calling the routine, the FCB would contain the filespec with the default extension unless one already existed or a slash "/" followed the filespec. If I wished for a default extension of /ASC, my routine would look like this:

```
PUTEXT      LD       HL,DEFEXT       ;HL => Default Extension
            LD       DE,FCB          ;DE => FCB containing filespec
            CALL     @FEXT           ;Fetch Extension
            RET                      ;NZ - irrelevant
DEFEXT DB   'ASC'                    ;Default extension in U/C
```

Once the complete filespec is contained in the FCB, one can attempt to access that file. But certain information is necessary for the Operating System to interface with that file. The logical drive number, diskette location and size of the file are just a few of the essential things needed to access the file. How does one get this information? OPEN the file! This is the process in which the system searches a disk's directory or multiple directories for a filespec and then sets up a 32 byte data region called an FCB. Of course, this assumes that the file is found (the LDOS Tech section contains the layout of the FCB). To open a file, simply point DE to the location of your FCB, which contains the filespec converted to upper case (placed there by @FSPEC). HL must contain the address of a 256 byte buffer which the system will use to READ or WRITE from. The B register must contain the Logical Record Length (LRL) of the file. Since I only performed Sector I/O (256 byte blocks), the LRL = 0. @OPEN, along with the other file handling routines return with the Z flag set if the operation was successful. If NZ, then the A register will contain the Error number (See the LDOS Error Dictionary). The code following the filespec check looks like this:

```
OPENFILE    LD       HL,IOBUFFER     ;HL => 256 byte buffer
            LD       B,0             ;B = 0 = 256
            CALL     @OPEN           ;Open the file
            JP       NZ,OPENERROR    ;NZ - Display Error
```

If @OPEN returns NZ, then the two most likely errors returned are : File Not Found, or File Access Denied. After successfully opening a file, the FCB no longer contains the filespec, but information relative to the file.

What if the application wished to create a new file or overwrite an existing file? The @INIT routine performs the same function as @OPEN except that if the file does not already exist, it will be created. The Z flag will be set after the @INIT call if successful. If the C flag is set, a new file was created.

Now that the program has access to the file, something can be done with the it. The file editor needed to READ records, make modifications, and WRITE changes. It was decided to make all record positioning using 256 byte records (Sector I/O). This is how the system performs all of its I/O. When record lengths other than 256 are used, the system must calculate where each record belongs. For example, a file with LRL = 100 would have its first record (record 0) on logical sector 0, relative byte 0, the second record on logical sector 0, relative byte 100, the third on logical sector 0, relative byte 200, the fourth on sector 1 relative byte 43. Since this editor uses sector I/O (LRL=256), the terms Record and Sector can be used interchangably for these purposes. FCB+12 and FCB+13 contain the next logical sector number (lsb,msb) to READ/WRITE from. If FCB+12/13 = 0, then the file has no sectors allocated. FCB+10/11 contain the next record number to get. Initially these bytes contain 0, indicating the next record to READ/WRITE is Record 0. The following code will read in the current record:

```
READCUR      LD        DE,FCB        ;DE => FCB of file
             CALL      @READ         ;Read the record
             JP        NZ,READERROR  ;NZ - Disk Read Error
```

After that @READ, the current record number would be incremented. If @READ returns NZ, the most probable errors are "Parity Error During Read", or "Record Number out of Range". Where are the contents of the record? There are two possibilities - 1) a user defined buffer for LRLs <> 256, or 2) the buffer that was specified when @OPEN was called. We'll discuss files having LRLs other than 256 in a while. FCB+3/4 contain the address of the buffer to store the data. These bytes can be manipulated to change the buffer address, so that one could read a record, add the LRL to the buffer address in the FCB, stuff back into the FCB, and read the next record, etc. What if one wishes to read some record in the middle of the file, or the end? @POSN positions the current sector number in the FC8 to the Record number contained in BC. Here's a simple routine to read a record:

```
READREC    LD       DE,FCB        ;DE => FCB of file
           LD       BC,(REC)      ;set BC = Desired record
           CALL     @POSN         ;Position to Record
           JP       NZ,OUTRANGE   ;NZ - Out of Range
           CALL     @READ         ;Read Record
           RET      Z             ;Z - successful
           JP       RDERROR       ;NZ - Display error
```

The same goes for writing:

```
WRITEREC   LD       DE,FCB        ;DE => FCB of file
           LD       BC,(REC)      ;set BC = Desired record
           CALL     @POSN         ;position to Record
           JP       NZ,OUTRANGE   ;NZ - Out of Range
           CALL     @WRITE        ;save changes to disk
           RET      Z             ;Z - successful
           JP       WR ERROR      ;NZ - Display error
```

The only difference between files having LRLs <> 256 and sector I/O, is that the data to READ/WRITE is placed in a user buffer pointed to by HL. This buffer should be adequate to hold 1 logical record, and cannot overlap the I/O buffer specified at @OPEN time otherwise disastrous results may occur. The READ/WRITE routines could be changed to handle any LRL as such:

```
READREC    LD       DE,FCB        ;DE => FCB of file
           LD       BC,(REC)      ;p/u Record number
           LD       HL,UBUFF$     ;HL => User Buffer
           CALL     @POSN         ;position FCB
           JP       NZ,OUTRANGE   ;
           CALL     @READ         ;Read Record
           RET      Z             ;Return if OK
           JP       RDERROR       ;else display error
```

After any modifications have been made, the directory must be updated in order to reflect the new changes. This is known as closing a file. The following routine will update a file's directory entry:

```
CLOSE      LD       DE,FCB        ;DE => FCB of file
           CALL     @CLOSE        ;Close the file
           JP       NZ,ERROR      ;Disk I/O Error
           RET                    ;Good - return
```

These routines perform all the basic file I/O through the operating system. Next issue I'll go through some of the more specialized file handling routines.

DK

LET US ASSEMBLE

by Rich Hilliard

In response to numerous requests, the LDOS Quarterly is starting an introductory assembly language section. The author is rather new to the subject and can still remember the pain involved so that the approach will be somewhat sympathetic. (Experienced persons will undoubtably remark that the first syllable of the word "sympathetic" be stricken.)

Assembly language requires not only careful planning and attention to detail but a certain psychological frame of mind. It is a discipline (some even say a religion) not unlike yoga or toilet training, which requires attention to several aspects of the person as a whole. Therefore, this series will attempt to develop the proper mental attitude as well as the "How to" approach.

The first aspect of the "expanded" mental outlook is that you must develop a perfectly monstrous ego which knows no limits whatsoever. Two exercises are recommended to augment meglomania. 1) Every morning, look in a mirror and repeat ten times: "I am a POWER in the universe". 2) While typing in code, take to humming Nobody Does It Better. Other things to work on are :

1. Take all criticism as an affront to the natural order of things.
2. Talk of nothing but programming until you have no friends.
3. Never encourage or assist persons with less knowledge, mock them instead.
4. Never examine code that is not yours without launching into endless monologues which extoll your prowess and defame all others.
5. Above all be extremely opinionated about everything.

Well, now that the fun is over let's get to work. The first notion to forget is that assembler is difficult. It isn't, it is merely tedious. For purposes of our discussion, a Z-80 based computer basically can add, compare and move information. However, it does this in a very rapid fashion. It is the various combinations of these functions which make things interesting.

Those three functions may not sound like much but keep in mind that there are only four basic compounds which encode the DNA molecule. Various groupings of these compounds form the "blueprint" for all life on this planet from microbes to humans. Therefore, a few functions, fairly worthless when taken alone, can cause some quite complicated things to occur.

If that doesn't impress you, then keep in mind that the computer can only count in a base two numbering system. (This is sometimes referred to as Binary but the more popular parlance was espoused by Lawrence Welk. E.g. "A one anda two, anda one anda two etc.) This, of course, is due to the fact that the two digits of Binary ("0" and "1") are used to bear a direct relationship to the electronic "off" and "on".

Now binary numbers are rather much for beings of higher intellect to deal with (Welk excepted). This is because humans, being enslaved by their visual cortex, have a rough time "seeing" a pattern of more than seven objects. Therefore, 01010000 which is one binary byte (8 bits) is rather rough to memorize or calculate with. It is even more difficult to readily decode a typical 16 bit address which is extremely prevalent in 8 bit micros. To wit, 1010101101111110 conveys very little information until our monkey brains break it up into recognizable patterns. Since 8 still violates our pattern recognition ability, the common custom is to use groups of four binary digits.

1010 1010 1111 1110 is now "visible" but takes up a lot of room so a simplification was in order. Leaving the history to college professors (who will also insist you know who Herman Hollerith was), a simple method of notation which uses base 16 numbers is used. The math involved can be derived from almost any introductory book on assembly language (it was probably the last thing you understood) so I will not encumber you with it. The lead number in this base 16 system (called hexadecimal or hex) would be AAFE. This

makes communicating about binary numbers much easier that the proliferation of ones and zeros. I will refer to hex numbers and give their decimal equivalents so there is no need to memorize or do base conversions. As you become proficient, hex numbers will merely become part of your vocabulary through use. The following chart may be handy for a while, however.

```
Decimal Binary   Hex

 0       0000    0
 1       0001    1  (pretty easy so far)
 2       0010    2
 3       0011    3
 4       0100    4
 5       0101    5
 6       0110    6
 7       0111    7
 8       1000    8
 9       1001    9  (now the fun begins)
10       1010    A
11       1011    B  (Since hex is base 16,
12       1100    C   six more "digits" are
13       1101    D   needed than for base 10.
14       1110    E   The letters A-F are
15       1111    F   drafted.)
```

Half a byte (4 bits) or one hex digit is sometimes referred to as a nibble.

The primary tool of the assembly language programmer is a program called an assembler. A program is a series of numbers in memory which is interpreted by the Z-80 chip. An assembler is merely a means of transforming the information from a humanly readable form into one which makes logical sense to the big Z. But before that happens, the code must make sense (at least some time) to us. Therefore, a set of symbols pertinent to Z-80 instructions, but written in human terms, must be learned. These are commonly called assembler mnemonics.

The mnemonics are entered into a text editing program and then processed into machine code by the assembler. In the case of many so called "Assemblers" the text editor is included for programming convenience. EDAS (for EDitor ASsembler) is one of these. The coding we will do will be done on EDAS. It is possible to make machine language programs without such a tool, just as it is possible to make a cake by glueing crumbs together, but who would want to?

There are several components to the editor which we will discuss as they are first encountered. In EDAS, just as in BASIC, there are various classes of instructions. The first class we will call editor commands. These are similar to BASIC commands in that they are something which is to be executed immediatly. In BASIC, these are things like EDIT, DELETE, LIST, RUN etc. In EDAS, each of these operations is handled by a command.

Some EDAS commands instigate a "mode" of operations. This is not a foreign concept to the BASIC programmer at all. Take the EDIT command from BASIC. Typing EDIT (line #) will enter the "edit mode" for that given line. The programmer is said to be in "edit" until the mode is voluntarily left. EDAS has several modes which work like that. A minor difference, is in BASIC the "add text" mode can be entered merely by typing line numbers. In EDAS, you must enter the "I"nsert mode which works in similar fashion to BASIC's AUTO mode.

The second class is instructions which comprise the actual assembly code. These are also divided into two groups, the mnemonics (often called op-codes) which will be rendered into machine language, and the instructions to the assembler which are called psuedo-ops. This latter group does not become part of the finished machine code but rather instructs the assembler about such things as where the program is to load, defines data, reserves space etc.

Do not dwell too much on this cursory explanation. You programmed in BASIC just fine without knowing whether you were typing a statement, a function, or a command. I only included this information to possibly clarify some buzz words you might have heard.

Now let's try a simple program. Remember the old saw:

```
10 PRINT "HELLO, I AM YOUR TRS-80 COMPUTER"
```

It probably was the first line of code that you ever wrote. (sniff, ah the innocence of $5000 ago). On the TRS-80 I/III series, there are several ways in which this can be done in assembly language. For now, we will take the easiest path. Turn to the Technical Section of the LDOS Manual, blow off the cobwebs, and find the page (the numbers are different in various versions) which explains the system entry point @DSPLY under the heading Video and Printer I/O routines. A system entry point is a place where, by establishing certain conditions, the programmer can do the equivalent of GOTO or GOSUB or IF.. THEN, to accomplish a job.

There is much debate on using system entry points. Some say that a novice should not use these things because the knowledge of how they are done is never learned. Really? I wonder if these same people built their first automobile. OK, granted someday you might have to write one of these routines yourself. Fine, toil with it then. Not to use what is provided NOW is a waste of your time and the computer's memory. It also has the added bonus of having to change relatively little in a program for other versions of the same operating system. We, therefore, will use them but also discuss other ways. The system entry point for @DSPLY says that it will display a message line terminated by a carriage return or end of text marker. The latter is equivalent to ending a BASIC PRINT command with a semi-colon after it.

Enter EDAS by typing its name at LDOS Ready. To enter the Insert mode type I and note the numbers 00100 at the left side of the screen. You are now in the equivalent of BASIC's AUTO command. There are some differences. Just as in AUTO you may use a starting number and specify the increment. (I100,10 is the default I1,1 would start at 00001 then 00002 etc.) The difference is that INSERT does not allow you to overwrite a line. Line numbers in assembly are useless for execution of the code and serve merely to organize the source text. NEVER refer to line numbers for branching.

Now tab over one zone by typing the right arrow. The source code is divided into zones and the zone immediately after the line numbers are RESERVED for labels, not instructions. Type in the word "ORG" and tab again.

ORG is one of the psuedo-ops referred to earlier. It tells the assembler where the program will load into memory. The assembler then works on all instructions using the ORG parameter as a relative starting point. LDOS requires memory up to and including address 20,991 (X'51FF' [X'' means a hex number with the value contained within the single quotes another way to indicate a hex number is by the suffix "H" as in 5lFFH]) and from HIGH$ up. HIGH$ will start at address 65,535 (X'FFFF') and progress down as you add various LDOS advanced features, so depending on how your system is configured it will be at various places. The MEMORY command can tell you where, but we will not be writing anything long enough to violate it for now. Based on this discussion, the lowest address we may ORG at is 20,992 (X'5200').

Addresses (or numbers) in EDAS may be decimal, hex (ending with H) or binary (ending with B)(in case WELK uses EDAS). It should be noted that hex addresses which start with a letter (A-F) MUST be preceeded by a zero so that the assembler can tell them apart from labels. This is covenient because it is not necessary to convert between number systems. Merely enter a number in hex if you have a hex number or in decimal if you don't. The parameter to ORG may be at any old place your heart desires between the low and HIGH$. The only requirement is that your eventual program length will not violate HIGH$ (this is illegal in some states). Since lots of room is good news let's go as low as possible and choose X'5200'. Press <ENTER> and be sure your code looks like this:

```
00100      ORG     5200H
00110
```

@DSPLY requires that the register pair HL be "pointed" to the first byte of the message. Registers are temporary storage areas within the Z-80 that are used to manipulate the information fetched from memory. How it is manipulated is up to the programmer. The regular registers in the Z-80 are named A,F,B,C,H,L,D, AND E. They each may hold a byte of information at a time. Three sets may be paired for 16 bits of information. These are the BC, HL, and DE pairs. To point a register pair at something simply means that the address of the "thing" is contained in a register pair.

Additional registers in the Z-80 which we will get into in later issues, are the I, R, IX, IY, SP, PC, and all the registers of the previous paragraph are duplicated in what is known as the "prime" set.

@DSPLY expects the address of the message to be in the HL register pair. The most common way to get information between memory and registers is with a LOAD instruction (mnemonic LD). EDAS has the ability to calculate addresses for us. We do this by labeling the address with an arbitrary name and then treat the label as if it were something real! This makes life very pleasant. Let's call our message "HELLO". To load the address of HELLO in HL type:

```
00110   LD      HL,HELLO
```

The comma is best read as "with", so we said LOAD HL with HELLO. Now HL is pointed to the message (I know that HELLO doesn't exist yet, so quit complaining.) This means that all of our entry requirements are met and we can vector to @DSPLY. Since we still wish to be in charge after the message prints, we will CALL @DSPLY (GOSUB). In the technical section of the manual observe the numbers after the word vector. Four digits eh? Pretty suspicious. In fact, this is the address where this routine is entered. Between the <> are the Model I addresses and between the [] are the Model III addresses (@DSPLY is the same for both). Whereas (I'm not a lawyer, really, other people do use that word.) in BASIC a GOSUB takes a line number, in assembler a CALL takes an Address. Your line 00120 is a CALL 4467H (17511).

The message called HELLO must be part of the program but should not be in a position where the message will execute. If we stuck it next, the Z-80 will try to run the collection of bytes which form the message. Therefore, the equivalent of GOTO can be used to jump around the message. Unlike BASIC, however, no time is lost by having the message at the end of a program. This also means a saving of some code since there is then no need to vector around the message. All of this is why it is customary to place most video output at the end of a program (or between unrelated parts).

We MUST however, return control of the Z-80 back to LDOS. (At least if you want to do more than print one lousy hello line). A few pages earlier in the technical section reveals a system entry point called @EXIT. Miracle of miracles, this doesn't even have any entry conditions. @EXIT cleans up after us and returns to the LDOS Ready prompt. This, if you will, is kind of a CMD"S" from BASIC. We do not want to CALL this since we are now giving up the ship. We need the equivalent of GOTO which is JUMP (mnemonic JP). The vector for both models is X'402D' (16429). Line 00130 reads "JP 402DH".

Since the program is over, we might as well compose a message. For this we need the psuedo-op DB (or DEFB) which stands for Define Byte. In EDAS, this means we get to string together up to 128 bytes. The assembler upon seeing a DB, merely places the subsequent series of bytes into memory. Anything contained within single quotes (') means that the characters are translated to their hex values of the ASCII number so that we humans need not bother with such demeaning and odious and vexing and more than likely error producing trivia. Line 00140 then reads like this:

```
00140 HELLO     DB      1CH,1FH,'HELLO I AM YOUR TRS-80 COMPUTER',0DH
```

Note that the different portions of the message are separated by commas. Looking in the Video Control Codes section or your TRS-80 manual provides an explanation of the two single byte codes preceeding the message. PRINT CHR$(28) moves the cursor to the upper left corner of the screen. Remember, we are doing the EQUIVALENT of PRINT. 28 is, of course, X'1C'. CHR$(31) (X'1F') means clear from cursor to lower right corner. For those of you still awake this is a CLS. The last byte of the message is X'0D' or decimal 13. This you will recall is required by @DSPLY to signal the routine that the printing is over for now.

Finally, we need to tell the assembler both that the source is over and what address to begin execution at. This is handled by the END psuedo-op. In most cases, the program begins execution at the first byte. This means that our execution address (sometimes called "transfer" address) will be the same as the ORG address.

To LIST the program use the "P" command. The "P" command works a little differently than BASIC's LIST. Typing P will print from the "current line" and 13 more lines below it. P line number, will print one line specified by number. P line number comma line number, will print starting at the first and ending at the second. A "#" means TOP of TEXT and an "0" means bottom. P#,0", therefore, is the same as typing LIST in BASIC. The program should look like this:

```
00100           ORG     5200H       ;comments are placed
00110           LD      HL,HELLO    ;behind semi-colons
00120           CALL    4467H       ;to clarify things
00130           JP      402DH       ;because lot of this
00140 HELLO     DB      1CH,1FH,'HELLO I AM YOUR TRS-80 COMPUTER',0DH
00150           END     5200H       ;tends to look the same
```

Press <BREAK> to exit the insert mode. Now let us assemble. If you ever need to look at the code again, the source file must be saved. This is done with the W (for write) command. So as not to overtax, call the file PRINT. Type "W print" and EDAS will save the Source Code in the file PRINT/ASM. If you supply your own extension the default will not append to the filename.

Before proceeding, it is a good idea to assemble without creating the object code just to see if all is well. To do this type "A -WE" which means assemble, and the dash W E means wait on error. If there is an error don't call me, you messed up. The assembler will pause on the error line and display an appropriate error message. Press any key to proceed after noting the line number of the error or press <BREAK> to get out of the assemble mode. You may enter the edit mode by typing "E" followed by the line number and a carriage return (ENTER). You will be pleased to note that the edit commands are the same as they are in Level II BASIC. If there is no error, the display will continue and the message 00000 errors will appear. For those of you who have errors be sure to save your source before leaving EDAS and after correction.

To assemble for real, add a filespec as in : "A PRINT-LP-WS". If you do not have a printer on line leave out the "-LP". These dash something or others are called switches because by specifying them you turn an option on or off. -LP means send the assembly to the line printer and -WS means assemble with a symbol table (which for this program is not real lengthy). The Printed Output should look like this:

```
5200            00100           ORG     5200H
5200 210952     00110           LD      HL,HELLO
5203 CD6744     00120           CALL    4467H
5206 C32D40     00130           JP      402DH
5209 1C         00140 HELLO     DB      1CH,1FH,'HELLO, I AM ETC.
     1F 48 45 4C 4C 4F 2C 20
     49 20 41 40 20 59 4F 55
     52 20 54 52 53 2D 38 30
     20 43 4F 4D 50 55 54 45
     52 0D
5200            00150           END     5200H
```

The column at the left contains the address for the bytes in the next column. At X'5200' will be an X'21'. At X'520l' will be X'09' and at X'5202' will be the byte X'52' etc. The second column is the object code which will be saved in load module format (/CMD) to the file PRINT/CMD. We will have more to say on object code next time when we use the LDOS system debugger to "see" the execution of this program.

Exit EDAS by giving the B command, and at LDOS Ready type "PRINT". If all went well your program should execute.

Next time, besides learning DEBUG, we shall explore direct to video memory implementations of a ripple and bubble sort both in basic and in assembly. We will call this segment "Sorts Illustrated" (groan).

Also we will begin moving a little faster because much of the groundwork has been laid. If you wish to explore certain subjects within this series, the author is definitely open to suggestion because he is not yet totally proficient in Assembly. After he gets good, he will, of course, send you a nasty gram for your impertenance.