LDOS QUARTERLY

January 1, 1983 Volume 2, Number 1

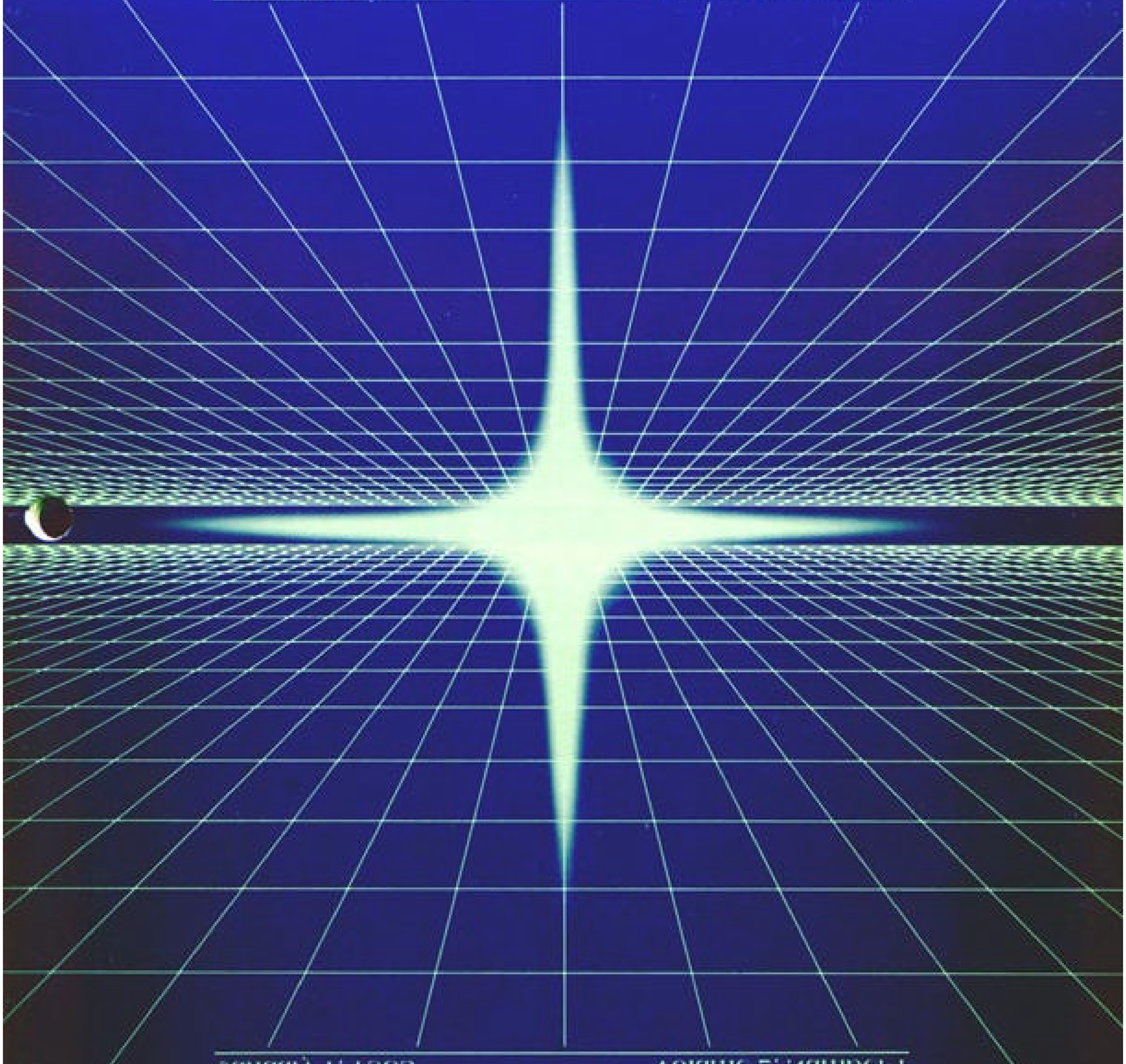


Table of Contents

= IMPORTANT NOTICE =

Logical Systems will be moving to its new location in April of 1983. To be sure your correspondence is handled promptly, please use the below listed address starting April 15th. Postal correspondence should use the PO Box number. Parcels sent UPS should use the street address only.

Logical Systems, Inc.
8970 N. 55th Street
PO Box 23956
Milwaukee, WI 53223

(414) 355-5454

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

Articles sent for consideration should be accompanied by typewritten or lineprinted copy. An ASCII text file or Scripsit file MUST accompany the printed copy! Please do not send in printed text without a disk, as it will NOT be considered for publication. Payment will be made in the form of a product from LSI, or $25.00 per page ("page" is defined as page in the then current newsletter format). The size of the article will determine the value of the product, although no reasonable request will be refused. Please include your name, address, telephone number and LDOS serial number with your submission. LSI is extremely interested in seeing submissions from our users, and is open to suggestion on any ideas for the Quarterly.

Submissions should be sent to:

LDOS Quarterly Editor
11520 N. Port Washington Rd.
Mequon, WI 53092

VIEW FROM THE BOTTOM FLOOR

by Bill Schroeder

Well, they interrupted my basket weaving and wallet making by handing me my crayons and a tablet. It must be time for another article for the LDOS Quarterly, so I'll give it a try.

For those who would prefer not to read through all my ramblings just to find out what the special offer is for this quarter, I'll start off with it. One of our most popular utilities is FED. This is the official LSI file oriented "ZAPPING" utility, which we believe to be the best of its kind for use with LDOS. From now until March 31, 1983 the price of FED will be dropped from $40 to JUST $25. After March 31, 1983 the price will go back to $40. Even better than that is my "FED with $50". This special package deal allows you to get FED for just $10, but there is a catch; you must order your "$10 FED" with some other LSI product(s) so the total value of the order is more than $60. This special offer is NOT available in conjunction with any other LSI special and is available only to LDOS owners that are on the Extended Support Agreement.

The LDOS Quick Reference Card is now being shipped. This card ended up to be 20 panels (10 per side) in length. It is 4 color and coated with clear varnish. This is without a doubt one of the best QRCs around. Any serious user should order one of these at just $8.95 (postage paid). It will be sent to you by first class mail. These cards are available directly from LSI and each registered owner will be limited to purchasing two (2) of these cards.

Lobo Drives is now shipping the MAX-80 computer in quantity. This is a super little machine running at a clock speed of over 5 MHz. There is an LDOS 5.1.3 (called MAX-LDOS around here) for use with this new machine. There is a lot more information on this machine elsewhere in this quarterly so I won't dwell on details. Let me just say that if you are in the market for a new Z-80 machine that runs LDOS and CP/M, has a real time clock, is 128k capable, has the SIO dual RS232 set, handles 8" and 5" drives and can run most Model III software you should look at the MAX. I think you will be quite surprised. If all this is not enough to pique your interest, consider that the MAX sells for only $820. That is not a misprint - it is only $820.

The MAX-80 from LOBO is a truly astounding machine. It has all the hardware capabilities of a Model-I, Model-II and Model-III and has more hardware features than all three put together. Yes, it will take some time for the full potential of this machine to be utilized from a software standpoint, but at this time it is without a doubt the best thing that has come along in the TRS-80 compatible area. When you consider the price of just $820 it's hard to believe. You of course must buy a monitor (about $150) and a couple of drives (about $600), but this still leaves a total of only about $1500 to $1600 for a complete state of the art Z-80 system which is CP/M and LDOS capable. The only negative consideration would be the service aspect as LOBO is the only service location at this time and they are in California. This draw back is offset, however, by the fact that the MAX-80 is designed with this in mind and contains a very low component count on a simple board design. This is much to the credit of Kirk Hobart, the design engineer of the MAX-80. We at LSI have yet to have a MAX-80 fail or even glitch in any way. If you or anyone you know is considering a TRS-80 type computer, the MAX-80 certainly deserves a serious look. Also, you should keep an eye on LOBO for many neat things in the future.

For those that are interested, I have been running the Radio Shack hard drive (26-1130) for the last 5 months or so, and I am very pleased with both the reliability and the very low noise level of this drive. If you are thinking of a hard drive for your system, you should look at the Radio Shack 5-MEG. It retails for just $2495 and is available through your local RS store. One more new item from the Shack is their doubler for the Model I. The RS double density modification is easy to install, very reliable, fully supported by LSI and competitively priced.

Let's touch on a subject that is near and dear to my heart (well more like a knife in the back). That, of course, is software piracy; in particular, piracy of LSI software. We are continuing to see illegal copying of our software at an increasing rate. With this in mind I am now asking for your help. If you as an LDOS user purchase (you must have paid for it) any software product that contains any LSI code, or you believe it contains LSI code please let us know, when where, how and from whom you bought it. No one has the right to duplicate any LSI code as LSI does NOT license any of its products at this time. Your name will not be used and if the information you provide (diskette, receipt, testimony or the like) result in the successful prosecution of the pirate I will pay you a reward of $10,000 dollars. I really hope that I am able to pay this reward many times, as this may slow down a problem that costs every legitimate software customer quite a bit of money each year and is seriously jeopardizing the cottage software industry. If it does not stop soon, the industry as a whole will suffer greatly and software will rise in price faster than any other commodity. So please help us and yourself to draw the line and at least stop piracy for profit. This $10,000 reward will be paid above and beyond any out of pocket expenses that you may incur in assisting us.

80-US magazine has now gone "slick". The January issue of 80-US has a whole new look. If you don't get this publication, you should. Just call or write to Mike Schmidt at 80-US, 3838 5. Warner St. Tacoma, Wa. 98409. He will be happy to send you a FREE (check that FREE) copy of 80-US so you can see what you are missing. There is no obligation at all, but you have to tell Mike that you are an LDOS owner and heard about this offer here. Enjoy.

There is now a new section on the LDOS MicroNET board. This is the "C" users group. This group will promote the use and understanding of the "C" programming language. It is set up as Section #1 on the LDOS board; Section #0 remains the main LDOS section. Those who would like to get into the "C" language should consider getting the "LC" package, available through LSI or MISOSYS at just $150. This package is a "C" compiler and EDAS 4.1 editor assembler. For more information on this package contact LSI or MISOSYS. We will now be trying to have an article or two relating to "C" and its uses in each quarterly. Hopefully, Earl Terwilliger will be heading up this effort. Earl writes well and is a much appreciated addition to our regular quarterly contributors.

Speaking about LDOS Quarterly contributors, articles from our users on LDOS, TRS-80s, Radio Shack, related software, or even editorial in nature are of great interest to LSI. We want our users to hear from other users. There is one big advantage in publishing in the LDOS Quarterly that you are not likely to see in any other of our industry publications, that is NO EDITING OF AN ARTICLE'S CONTENT. There are no "Sacred Cows" at LSI. We will publish relevant, well written and constructive articles, even if a few "not-so-nice" things have to be said. For a specialized publication we also pay rather well (we think). Our new author's rate is $40 per published page (this will change if we start typesetting). Because we do not typeset the Quarterly, we only get about half as many words on a page as does a regular magazine like BYTE. This would be like them paying $80 to $100 dollars per page. So if you have something to say, write it and send it in - it could pay off.

In my last column I mentioned the get together that we had here in Mequon and listed the attendees. Unfortunately I listed most of the people by name and finished with the statement "and staff". Well, needless to say the "staff" was justly offended by this oversight. So for clarification here is the technical definition of "and staff": Gil, June, Linda, Lorly, Lynette and Mark. I certainly apologize to these most valuable members of our company.

LSI now has a complete catalog of products available through LSI. This catalog is yours free for the asking, so please call or write and we will be happy to send you one. Our LSI hotline is now operational at (414) 241-4100. It is available 24 hours a day, 7 days a week. There is some interesting info on there and usually a "hotline special" is also mentioned that could save you some bucks on LSI products.

Many LDOS users and dealers have inquired about our desire to published or distribute software created by other than LSI. The fact is that WE ARE ACTIVELY LOOKING FOR SOFTWARE TO PUBLISH. LSI is looking for software of all types; UTILITY, LANGUAGE, APPLICATIONS, even ENTERTAINMENT. We are able to make royalty or lump sum deals. We are able to pay from 10 to 25 percent of the revenues on a product or up to $50,000 in a lump sum purchase of a copyright. We are also willing to negotiate any reasonable arrangement and will consider both exclusive and non-exclusive contracts. If you have authored or own the copyright/marketing rights of a product that works on LDOS, we would like the opportunity to review the product and consider it for LSI publication.

LSI has several rules regarding submissions for review, so to avoid disappointment please follow these guidelines. 1) All submitted software must be accompanied by complete documentation. 2) Your submission should include a letter clearly explaining the mission of the software package and the intended audience. 3) You must be willing to provide proof that you hold the rights to offer the product to LSI. 4) You must be willing to modify the product as may be deemed necessary by our analysts. 5) If payment is by royalty, you must be available and willing to maintain the product. 6) Your submission will NOT be returned to you. 7) You must allow 6 to 8 weeks for LSI to evaluate your product and its potential market. 8) If LSI publishes your product you must be willing to provide LSI complete source code. 9) Submissions should be sent to LSI Attention software acquisitions manager.

The LDOS 6.0 project is doing very well and we hope to have a Model-II version of this RAM based system released as an LSI product no later than mid 1983. We will publish a feature list of this new system in the April issue of the LDOS quarterly. We are looking at a retail price of $200 to $400 for this system, depending on the configuration purchased. We are considering offering 2 or three different packages containing different enhancements and utilities, something on the order of "USER", "ADVANCED USER" and "PROGRAMMER". This is all pure conjecture at this time as final marketing decisions have yet to be made. I will try to provide more definite information in the next quarterly.

We now have OFFICIAL LSI "T-SHIRTS". These are heavy cotton shirts in either Dark Brown or Sand with our Logical Systems logo imprinted in the opposite color on the upper left chest. They are available in Small, Medium, Large and X-Large for just $9.00 postage paid, or just $5 if ordered with any other LSI merchandise. Be sure to specify size (S, M, L, XL) and color (Dark Brown, Sand) when ordering.

LSI is moving to a much larger facility. We have purchased a 10,000 square foot building and will be moving our entire operation there sometime in April. The new LSI address will be 8970 N. 55th St. Brown Deer, WI. 53223 or PO Box 23956, Brown Deer, Wi. 53223. The new phone number for all services will be (414) 355-5454. These changes should become effective sometime in the month of April. We are still growing very rapidly and hope to open an LSI branch in another state in 1984.

LSI has become a very successful company, but we are cognizant of the fact that this success is directly attributed to the ongoing support and enthusiasm of YOU, our customers. Through the support and word of mouth advertising that you have provided, we have been able to use our resources for things like this newsletter and better documentation and testing for LSI products instead of lavish advertising. For this I thank you all very much, and you should thank yourself as the end result is more for your money when buying LSI software. I will see to it that YOU the customer are never forgotten as the real reason for the overwhelming success of the LSI operation.

USERS GROUP DIRECTORY

In response to the offer in the last issue, the following LDOS users groups sent in information about themselves. Please note that these are independent groups and have no official connection with LSI.

The Cincinnati TRS-80 User's Group has an active LDOS group.

CINTUG - LDOS SIG
Alan R. Moyer, Chairman (513) 868-0248
993 San Angelo Dr., Hamilton, OH 45013

or

CINTUG
Karl Wiedamann - President (513) 871-0073
Ed Fairman - Secretary (513) 542-5028
5380 Bahama Terrace, Apt 8, Cincinnati, OH 45223

For users in the Madison, WI area, the Madison LDOS User's Group meets the second Thursday of each month at one of the members homes. You can contact Scott Loomer at (608) 233-0488 for details.

The Adelaide user's group in South Australia has several sub-groups depending on your particular area of interest. They also publish a monthly newsletter. For information, contact:

Adelaide Micro User's Group
Rod Stevenson, Secretary
36 Sturt St.
Adelaide 5000, South Australia

Telephone 08 51 5241 (bus) 08 337 6682 (home)

TCUG - LDOS SIG

For those of you around the Washington D.C area, there is an LDOS special interest group which is part of the TCUG users group. They meet the 1st Wednesday of every month. For further information, contact Carl Hessinger at (301) 474-8486.

MAXIMUL - MAX-80 User's Group

For MAX-80 owners, the MAX-80 Independent Microcomputer Users League, or MAXIMUL for short, has been started by Doug Hogarth and Bill Vermillion. You can contact Bill on Compuserve under number [70270,214]. They currently have several newsletters available under that number in ACCESS on Compuserve. For more information, send a self addressed stamped envelope to:

MAXIMUL
PO BOX 19525
Orlando, FL 32814

CUSTOMER SERVICE TIPS

As the years have flown by, the staff here at Logical Systems has become increasingly efficient in the customer service area. This not only makes things easier internally for us, but also gives you much faster and better service. In the past months it has become necessary for certain groups of our staff to handle specific aspects of the broad services we provide. There is usually someone or some way for us to offer you assistance with most questions or problems. If we can do nothing else, we try to steer you in the proper direction to get what you need.

Recently we have evaluated the types of services we have been providing, either via the phone, or through the postal service. Both aspects have been categorized and our staff of experts has been broken-up, departmentalized, and will try to specialize in what each group does best.

Below are examples of how we'd like you to classify either your letters, or your phone calls to us.

If you are calling or writing about:

1. System programs (LDOS), specs on updates, syntax questions, hardware questions or requirements or anything you feel only a good programmer or technician could answer.

The Syntax is:

> Customer Service Department Please.

*Please do not try explaining in length any technical question or problem to our receptionist. She will only end up re-routing your call and you'll have to go through your explanation all over again. The same is true with the person who handles the mail. If its marked Attn: Technical Support Department, it will be enroute to the correct department a little faster.

2. Update questions (in general) "Have we received it, or has it gone back out to you?", product orders and shipments, Dealership questions, and requests for our product catalogs are handled by:

> Order Processing Department

3. Warranty Card Registrations (all products), Extended support agreement Subscriptions, address change information and warranty status information are handled by:

> Subscription and Registration Department

4. Specific questions about products we sell, compatability, and policies surrounding our product sales are handled by:

> Product Sales Department

5. Special services like, Custom software, technical programming, contract purchases or OEM Dealer questions should be directed to:

> Contract Sales Department

6. Articles, suggestions, or ad placement for the LDOS Quarterly are handled by:

> LDOS Quarterly Editor, or Ad Manager

** All other questions that just don't fall into the above categories will be best handled if you start with a BRIEF description of what you are looking for. This means both your correspondence or you calls.

Remember, our technicians currently do not answer the phone nor do they open and date stamp the mail. We're trying to streamline our services to you but we need your help.

CUSTOMER SERVICE (TECHNICAL SUPPORT)- PHONE HOURS

The normal hours for technical support are:

9:00am - 12:00 noon and 4:00pm - 5:00pm Central Time

The hours for Customer Service in general are:

9:00am - 5:00pm Central Time

Between 12:00pm and 4:00pm our technical staff is scheduled for meetings crucial for new product development and review of current product status. To give these people time to bring you the best possible software it became necessary to lock them away for a few hours each day to do their thing.

Regular (non-technical) Customer Service and all of our other activities continue through the afternoon except the tech support service. So if you can, try to place your calls during the appropriate times and you will find that your questions will be handled much smoother. If you call and speak with our Customer Service department for Technical Support, they will try to help you, but if they are unable to give you a complete & accurate response they will write down your name, number & question(s) and get back to you with the appropriate answer.

Our present phone number 414/241-3066 has 5 lines designated for sales and service. When we relocate our phone number will be 414/355-5454 and there will be many more lines for your calls. This new phone number will be effective April 15th, 1983. Remember, our Customer Service staff can only handle a certain amount of calls at the one time. So don't be surprised if you hear a sweet voice come back to you and say "All those lines are busy, would you like to hold". Don't fret though, if you choose to hold, your call will be handled in the order which it was received. You will be taken care of.

UPDATES

This is an item which is of great importance and at times is gravely misunderstood. How exactly are updates at LSI handled?

Your disks really don't go into never never land on their way here, when we receive them, or on their way back to you. Though it may seem that way to some of you, some of the time.

We process a great deal of master disks for update, but recently due to the increased volume we've experienced calls where people expect to see their updated disks returned to them much quicker than is possible. Also, we have had some disks lost in the mail either on their way here, or on their way back to you. (That's never never land!)

The best way for you to insure that we have received any parcels you send us would be for you to send these Certified mail, or UPS. When we receive yours disks, be it an LDOS disk or any other masters for updating, we have had an approximate turn around time of about one week (from receipt). Then your disks should be on their way back to you. We normally use first class mail when returning these.

We process so many updates that it becomes a real task to try and locate the status of a disk we should have received four weeks earlier.

Each of our departments is here to supply you with its special administrative, or software talents and we appreciate your patience and acceptance of our changes as they occur. Please bear with us during these changes, we have to adjust too.

AGAIN - our departments are as follows

* Customer Service Department
* Order Processing Department
* Subscription & Registration Department
* Product Sales Department
* Contract Sales Department
* LDOS Quarterly Department

Your calls or correspondence will be handled to your advantage if it gets to the right place the first time.

APL*PLUS/80 - A SYSTEM OVERVIEW

****

by Daniel J. Lofy & Lee C. Rice

I. APL ON THE RISE

As a marketable programming language, APL seemed doomed from the start. It was first created by Kenneth Iverson as a noncomputer language for mathematics at Harvard. Iverson later moved to IBM, and continued to develop the language, first as a hobby and later for possible programming applications (specifically for the IBM 360). His first book (A PROGRAMMING LANGUAGE, published by Wiley and Sons, 1962) stirred some interest in APL among only a few programmers and teachers.

The original implementation of APL required special terminals and printers able to handle its formidable set of Greek characters. In addition to the fact that it had poor string handling facilities and was not user friendly, its keyboard layout was also different from the standard keyboard; which meant that it also failed to attract computer hobbyists. Rumor has it that IBM originally published the implementation as a joke. In addition to everything else which it had going against it, it was also an interpreted language (rather than a compiled one); which meant that it was particularly unattractive in data processing environments where CPU time was a precious commodity.

APL had a few things going for it, however. The language is enormously productive in terms of programmer time, and permits the construction of swift and elegant code for many applications which require ten times the code in other languages. With the advent of minicomputers and microcomputers, CPU time became cheap and programming time became a precious commodity; so that, by the middle of the 1970's, APL had achieved a small but loyal following. In the years which followed, its growth can only be described as phenomenal.

Scientific Time Sharing Corporation (STSC) is one of no fewer than forty companies now devoted to APL software and publications, and the producer of APL*PLUS - an enhanced version of APL which contains stronger string handling capabilities and extremely portable APL code.

They produce and support versions of APL*PLUS for a wide variety of mainframes and minis. They entered the micro market in 1982, choosing the TRS80-Model III for their first implementation. Of equal significance, APL*PLUS/80 was designed to run under either TRSDOS or LDOS. "Designed to run" means that the manuals contain full instructions for LDOS implementation, and that APL*PLUS/80 takes full advantage of the power and flexibility of the LDOS operating system.

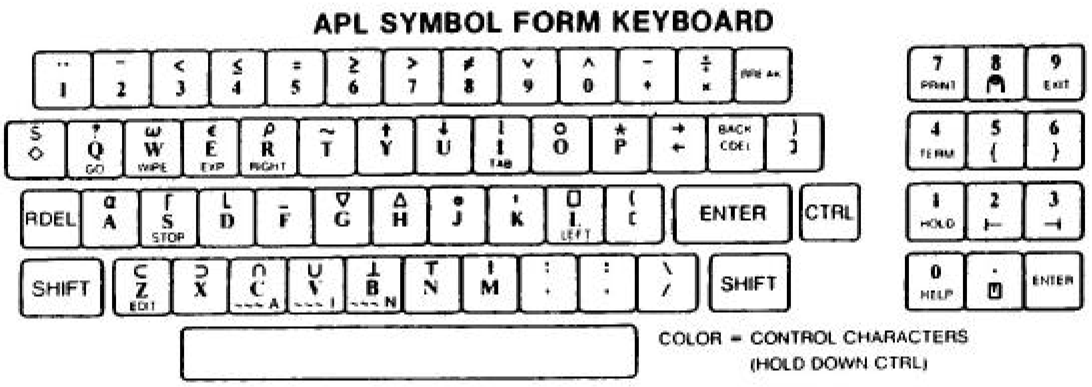
STSC also provides a chip which will permit the Model III CRT to display the APL character set, and a label set to mark the special symbols on the keyboard. We installed the chip in less than twenty minutes (no soldering required) thanks to a particularly clear set of instructions - having first been assured by the local Radio Shack repair shop that they could service the machine fully with the chip installed. For those who want APL but choose not to relearn keyboard positions, APL*PLUS/80 also contains a keyword option: mnemonic keywords replace the APL function symbols, and the keyboard retains its ASCII layout. Both the keyword option and the symbol option are filters, so that programs typed in (and saved to disk) in one format can always be displayed in the other. You can be working in one format, and change to the other with a simple command.

Like LISP and a number of other modern interpretive languages, APL is a functional programming language. It comes stocked with a rich set of system functions, and new functions can be produced as combinations of existing ones. In fact, a program is just a large function built out of smaller functional modules. Unlike many interpretive languages (like LISP), APL*PLUS/80 also provides full access to the operating system from WITHIN the APL workspace. The programmer has in fact two I/O options. Individual functions and programs can of course be saved to (or read from) disk, but another option is that of saving an entire workspace (which will typically contain many programs or functions). Access from one workspace to another is also possible.

Most importantly, entering an APL workspace offers no limitations in accessing other data. You can read in as data any files or programs - whether they are written in BASIC, FORTRAN, or virtually any other ASCII code. Lack of transportability of data and text files from one micro language to another is a very serious problem for implementations of most computer languages at the micro level. In APL*PLUS this problem has been eliminated once and for all.

II. PROGRAMMING IN APL

Whether you elect to install the APL character generator chip or to use the keyword forms (a set of mnemonics which replace the standard APL character set), APL will require some patience in becoming acclimated to an almost entirely new keyboard layout.



The better you typed before, the more practice you may need to purge ingrained habits. The time lost, however, is more than compensated by the fact that APL programs (called "functions") are typically much shorter than their counterparts in BASIC, FORTRAN, or just about any other language. Further, although it may seem a formidable task in memorization to learn all of the weird symbols, you'll soon discover that many of the most used characters are almost iconographic. See, for example, if you can guess what the following do:

```
A←2.0
B←⎕
⎕←B
⌊(3.1415)
```

The first expression assigns the value 2.0 to the variable A, while the second prompts the user for numeric input and assigns it to B. The third writes B to the terminal; and the fourth writes the floor (largest integer smaller than the floating point number) to the terminal.

One reason why APL functions are so short, and the language is so efficient, is that many of its functions extend to arrays as well as scalars. This reduces the number of loops required to implement most algorithms. Additionally, APL also offers the programmer the opportunity to write functions composed of other functions, all of which can be saved in a single workspace. This not only facilitates logical design of programs; but, since separate functions can be tested separately, debugging time is dramatically reduced.

For those who may feel uneasy with the concept of an array, it reduces simply to the notion of a list of elements of some kind. For example, the daily sales of John Jones on the working days of January might be stored within the single APL variable A as the array $110.00 $204.40 $160.59 etc. Now suppose we need to know Jones' sales for the entire month. Most programming languages would require us to loop thrpugh the array one element at a time. APL, however, provides a technique called reduction which inserts operators between each element and also performs the desired operation. In our example, Jones' sales for January could be computed by typing: +/A. This reduction of A under addition could also be used as a single piece of still another function. Operations such as reduction extend immediately to arrays of more than one dimension. If instead of one salesperson we had five whose sales were stored in a matrix B, we could type +/B and we would get a list of five numbers, each representing the total sales for one salesperson. Reduction is only one of the many ways provided by APL to avoid frequent use of loops, and to make functional expressions strikingly short and quite readable.

Another such operator is "rho", which is useful in determining the number of elements in an array. For example, instead of using a loop to count the number of days that Jones worked, we can simply type:

```
ρA
```

APL provides two modes of operation. The mode in which operations are immediately performed is called "desk calculator mode", and is illustrated by all examples so far. The "function definition" mode permits the user to combine APL intrinsic functions into user defined functions. To make the retail store example slightly more complicated, suppose we want a function to determine the average daily sales (to the nearest penny!) for the store for an arbitrary month. Let's also accommodate the fact that, in addition to not specifying the number of working days in the month, some months may require more salespeople on hand than others.

If we add one simplification to the above example by assuming that any salesperson who works one day works each working day, we could present the situation using the following BASIC program:

```
10 DIM MO!(20,5) REM MO is workdays by salespeople
20 TS%=0
30 FOR I = 1 TO 20
40 FOR J = 1 TO 5
50 TS! = TS! + MO!(I,J)
60 NEXT J
70 NEXT I
80 AV% = INT((TS!/100) + .5) REM Divide & round
```

Compare this to the following APL function:

```
        ∇ DAILYSALES MONTH
[1]       (⌊(0.5+(+/+/MONTH÷ρ,MONTH)×100))÷100
        ∇
```

To conclude that the APL is better because it does in one line what BASIC does in eight would be mistaken. At best, the above APL function offers a good example of the abuse of power and only accounts for part of the strength of APL. The better method would be to use a function such as the following.

```
        ∇ DAILYS1 MONTH;A;B
[1]       A←+/+/MONTH÷ρ,MONTH
[2]       B←(⌊(0.5+A×100))÷100
[3]       ⎕←B
        ∇
```

Notice that the above function could also have been defined as:

```
        ∇ DAILY2 MONTH;A;B
[1]       A←AVERAGE MONTH
[2]       B←ROUNDOFF A
[3]       ⎕←B
        ∇
```

where AVERAGE is

```
        ∇ R←AVERAGE MAT
[1]       R←+/+/MAT÷ρ,MAT
        ∇
```

and ROUNDOFF is

```
        ∇ X←ROUNDOFF Y
[1]       X←(⌊(0.5+Y×100))÷100
        ∇
```

Clearly, this last is not required for such a simple case; but, for more complex functions, nested definitions offer both readability, fast debugging, as well as the fact that they can be easily used in other functions. This example illustrates the fundamental advantage of the use of workspaces. Not only can the months be stored together as variables in a single workspace, but so can a variety of functions needed to process them. These functions can be copied separately into other workspaces should they be needed again, allowing the user to build a library of his or her most needed functions.

III. STSC'S APL*PLUS/80 SYSTEM

The complete system provided by STSC contains no fewer than three user manuals, two books, reference cards, APL character chip and keyboard labels, warranty agreement, and a product overview and ordering information for STSC's other systems and many publications available on APL. The APL system is provided on a both sides of a double density diskette.

The first side is to be backed up to a system diskette(TRSDOS or LDOS), and should be used in drive #0: there is JUST enough room for the APL system on an LDOS system disk. The second side of the diskette contains a variety of APL workspaces (to be described below); and, as a data diskette in one of the other drives, it has room for user created workspaces.

Complete backup information is provided, and the user may make as many backups as required. To run APL under LDOS, no user patches are necessary. STSC has had the foresight to provide a self-executing program ('LDOSAPL/JCL') which does all the necessary interfacing between LDOS and APL (and need only be executed once per backup). There are system commands in APL for accessing LDOS disk directories, and for reading into a workspace files not created under APL. Nor are users limited by the workspace configurations provided by STSC. The COPY command under APL will copy individual functions from disk workspaces into the active workspace; so the creation of new workspaces from existing ones is smooth and easy.

The manuals consist of (1) a Formatting User's Guide, (2) a Shared File System User's Guide, (3) a Programmer's Reference Manual, and (4) a Computer Operation User's Guide. The books provided are (1) the second edition of the classic APL textbook, APL: AN INTERACTIVE APPROACH, and (2) a tutorial APL IS EASY. It is NOT necessary to wade through the 1200 pages of documentation, since many of the workspaces are interactive, and STSC also provides suggestions as to what to read or skim and what to skip, depending on familiarity with APL and/or with the operating system (LDOS or TRSDOS). For even the novice user, about two hours of reading should be enough to begin hands-on APL work.

Sixteen workspaces are provided, all with the default suffix of '/AWS'. One of these, 'DESCRIBE', provides a description of the others. Most workspaces are NOT mentioned in the manuals and documentation. They are intended for hands-on work, and for later user modification. Once you have loaded a particular workspace, there is also a function/command called 'DESCRIBE' which will provide detailed information on what is in the workspace, and suggestions for use.

Three of the workspaces ('DEMO', 'DEMOAPL', 'BIGDEMO') are simple demonstrations of APL graphics and functions. They include such functions as calendars, tipping hats, bargraphs, snowflake designs, and interactive spelling contests. They are designed, not to teach APL, but to make it user friendly, and to give the beginner a glimpse of its power.

There are two tutorial workspaces. APLCOURS/AWS is a simple and interactive drill on basic APL functions - it provides a scoring technique at the end, so that users can pace their progress and identify trouble spots. SHAPE/AWS is a more advanced tutorial which tests user knowledge of the structure of the fundamental functions. Both workspaces are fully interactive, totally user friendly; and, like all other workspaces, they can be run in either keyword mode (with mnemonics) or using the APL symbol set.

There are two additional workspaces designed to be used with the books provided. CLASS/AWS provides all of the programs used in APL: AN INTERACTIVE APPROACH. The reader going through this book as a self tutorial will find hours of typing time have been saved. LESSONS/AWS contains all the programs used in the book APL IS EASY, and again saves hours of typing time. We should note that, especially in the case of CLASS/AWS, these programs are not merely drill exercises. Many are powerful examples of APL programming, and users will want to move some of them into their self created workspaces for later use too.

Two workspaces provide formatting information and functions. EDIT/AWS contains prototype functions for inputting and editing text. These functions are of limited value in the form provided, but suggestions are given for modification to suit individual text editing needs. FORMAT/AWS contains formatting functions mentioned in the Formatting User's Guide, and will again save the user hours of typing time.

Five additional workspaces contain miscellaneous utilities. SYSTEST/AWS provides a complete on line test of all APL library functions, and can be used on entry to APL (in much the same way as the TRSDOS program MEMTEST/CMD is used). CHARSET/AWS fills the screen with the APL Symbol Forms, and can be used for ready reference by those using these forms (or learning them) rather than keyword forms.

EPSONMXS/AWS has prototype functions for Epson printers with Graftrax, and provides a driver routine to allow the printer to reproduce APL symbols in bit graphics mode. The techniques used are more important than the actual functions, and readers with other printers can create their own custom print driver routines in APL using these as models. FILEAID/AWS provides functions for advanced users who wish to write-protect, read-protect, or execute-protect programs within a workspace (full password implementation is provided). The authors have access to two mainframe time-sharing systems, and we should note that APL*PLUS provides a system of security which rivals the big systems, for those in need of it. SENDRCV/AWS is a workspace containing powerful functions for communicating with a host mainframe equipped with APL*PLUS. Programs may be downloaded or uploaded between micro and mainframe, and full communications protocol is provided. Finally, UTILITY/AWS provides an additional set of APL programs for communications, and operates through the RS232 interface to allow the user direct control of BAUD rates, duplexing, and terminal modes.

Many of these workspaces will probably not be accessed by the beginning user, but it is surely a comfort to know that they are there and waiting. Readers who balk at the $300 pricetag for APL*PLUS should note that they are not buying a language, but a system. The authors presently have the following languages up and running under LDOS: FORTRAN, COBOL, UOLISP, LISP1.5, EDTASM, two data base systems, and three statistical packages, as well as Compiler BASIC. In each case, putting up the language has been only the first step. Interfacing with the system (LDOS) is the next, and providing utility support packages follows. If these packages are available from vendors, they are never cheap and seldom bug-free; and, if they are not, months of programming time must be committed to their production. There are no such problems with APL*PLUS: the support utilities are ready and waiting. Customization may be desirable or even necessary, but this can be done in APL itself, thanks to its complete interface with LDOS. We have had three occasions to write STSC with questions. In two cases they telephoned the day they received the letter, and in the third case, we received information in the mail within a week. In short, on a value-received basis, APL*PLUS is both powerful and dirt cheap!

IV. CONCLUDING NOTES
****

Once having experimented with the workspaces (a generous number of them!) provided on the distribution disks, the user may choose to establish his/her own account. This allows one to begin in either a clear workspace, or one which has built-in functions ready for execution. For those sharing their TRS80 with others, data files may also be protected under different account numbers.

The STSC implementation of APL offers many advantages, and these are well documented (with many examples). The book, APL: AN INTERACTIVE APPROACH, is an excellent overview of the language, and also details the differences between APL and APL*PLUS.

In addition to generous documentation and support packages provided by STSC with APL*PLUS/80, the company has also been most responsive to questions. In our first effort at installing the system under LDOS, we encountered an inability to execute the command)WSLIB, which gives a listing of workspaces on a disk. They were back to us with the appropriate patch the next day.

Equally reassuring was it to find out that our local Radio Shack repair shop knew about the APL character chip. For reasons not connected with APL or the chip, our TRS80 had to go in for service. They accepted the machine with no questions after being informed that the seal had been broken to install the APL chip; and, when it was returned, the APL chip had been left installed.

APL, like BASIC, is an interpreted language and is therefore quite slow. In fact, don't be too surprised if the first time you execute a function of considerable length you fear that you went into an infinite loop. What the language lacks in speed it makes up in its ease of programming, its large and powerful set of primitives functions, its ability to combine these primitives, and its modes of operation. If you already own a MODEL III, STSC's APL*PLUS/80 will turn it into a desk calculator which exceeds any you could buy before you even tried to write a function. Furthermore, it is a relatively easy language to learn especially given the almost 1100 pages of information STSC provides. If you're looking for a new and interesting language for your MODEL III we recommend this one.

THE "C" LANGUAGE

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

LOGICAL SYSTEMS has set up an "interest section" on the LDOS SIG for the "C" programming language. Wonder what all the fuss about "C" is? I can hear you saying "NOT ANOTHER LANGUAGE!." Well, yes, another language! However, "C" is not just another language. It is different than BASIC, FORTRAN, PASCAL, Z80 ASSEMBLER, etc. How different, do I hear you asking? In a series of articles, let's take a look at "C" and I will show you just how simple but powerful the "C" language can be.

In this first article, I would like to give a brief history of "C" and tell a little about it through its use of variables, storage and its use of functions.

"C" was developed by Brian W. Kernighan and Dennis M. Ritchie of BELL LABORATORIES. "THE C PROGRAMMING LANGUAGE" authored by Kernighan and Ritchie is the ultimate reference document for the "C" language. A wide range of computers now have "C" compilers. (TRS80, IBM PC, PDP11, IBM/370's, etc. "C" runs on various operating systems.) "C" has been called a systems programmers language since it is useful in writing operating systems. In fact, the UNIX operating system (a trademark of BELL LABS) was written in "C". Frequently, any given implementation of the "C" language references a particular UNIX version. (You don't, however, have to write an operating system to take advantage of "C".)

"C" lends itself well to "structured programming". Structured programming does not mean the absence of the GOTO statement. In fact, the "C" language does implement the goto statement. However as Kernighan and Ritchie (K&R) state in their book, the GOTO is "infinitely-abusable". When I discuss the control and flow of a "C" program I'll talk more on the topic of structure. (Not to be confused with the C concept of structures, i.e., records. Although, I will want to also talk about how C groups variables into a record or structure. LATER!)

"C" was designed to be a portable language. Its not very heavy at all! Many "C" programmers will say, it is light in some features. Most programmers are amazed when they see how relatively few identifiers (statements or keywords) that "C" has. Here is all there is:

```
int     extern   else       char      register
for     do       double     static    while
struct  goto     switch     union     return
case    long     sizeof     default   short
break   entry    unsigned   continue  auto
if      float    typedef    enum      entry
```

As you can see, many of the "C" statements are the same as those found in other languages. (For examples: if, for, goto, etc.) Did someone say they can't find the keyword enum in the back of K&R's book? I'll pretend I didn't hear that for now. New things come later!

"C" hereinafter referred to as C (I got tried of typing ""), is characterized as a "low level" language. A C source program is compiled (assembled) into an executable machine language program. This is not, however, why it is called a low level language. It is called that because it deals with data items much the same way as machine instructions do. Machine instructions deal with characters (bytes), numbers and addresses. These objects are what C deals with also. To illustrate this, we'll take a look at C's data types.

There are only four data types:

```
        char      - a single byte (one character)
        int       - an integer
        float     - floating point (single precision)
        double    - floating point (double precision)
```

The int data type can be further qualified to be short int, long int or unsigned int. The length of these number data types is specific to the machine for which the C compiler was designed.

You might be wondering, at this point, about how "high level" tasks are performed. Or for that matter, how is any task accomplished which other languages perform but for which you don't see a C vocabulary word? You say you don't see any read, write (I/O), string manipulation, or array processing statements in C's above listed vocabulary? I expect not! Such things are specific to a particular machine. You can of course invent functions to do these "high level" tasks and explicitly call them in C. Usually you don't have to invent them, however, since they are provided as functions in the C run time or the installation (computer specific) library. The installation specific functions, run time functions and functions of your own design aid in creating a more structured program. One might expect (and rightly so) that the main body of the structured program is also a function! It is in fact and is called main(). (The parentheses () are used to denote main as a function and can optionally enclose variable names representing parameters passed to the function.) There must always be a main() function in your C program as it is always the entry point. Braces {} enclose all of the statements that make up a function. C statements are expressions, such as x = 2 followed by a semicolon. The semicolon is used as a statement terminator. Statements may be, if you wish, grouped together (compounded) into blocks. A block is a statement or many statements which are enclosed in braces {}. This block or compound statement is treated as a single statement.

Braces surrounding the statements of a function, such as main() also form a block or compound statement.

Functions are invoked by their name followed by an optional argument list in parentheses. Taking a look at a simple C program, the use of the all these symbols () {} ; can be illustrated:

```
    main()
    {
        /* C sample program to print HELLO!   */
        /* and return back to the operating system */
        printf("HELLO!");
        exit(0);
    }
```

Note that comments in a C program are delimited by a /* and an */. In this sample program are 3 functions; main, printf and exit. The exit function passes back to the operating system the return code of 0. This function is unnecessary in this sample program since the program would "fall through" and end normally at the last bracket without it. (It is usually a better practice to return a value upon ending a function.)

The printf function prints the values passed to it in the format specified. (Its counterparts in the FORTRAN language are the WRITE and FORMAT statements.)

The arguments of a function are passed as copies of the values of the arguments. This is "call by value" versus "call by reference". The function gets its own copy of the variable and can't change the original passed value held by the caller. "Call by reference", that is, passing the addresses of the variables can be achieved in C if pointers (addresses) of the variable arguments are passed. Having the address of the variable, the called function can then change the value. This "call by reference" is actually how array names are passed. When an array name is passed as a variable, it is actually the address of the first element of the array. (It would be impractical to copy the entire contents of an array and pass these separate copies as is done with other variables.)

Any of the 4 data types (int, char, float, double) can be represented in C by a variable. Numeric and character constants are also used in C as data values. A variable is symbolically represented by a name. This name is composed of letters and digits. The first character of the name must be a letter and although large names are allowed, only the first 8 characters of the name are significant. A C convention is to use lower case for variable names and upper case for symbolic constants. NOTE: C keywords (reserved words) must be in lower case.

Variables, before used must be declared. In declaring a variable the data type the variable is to represent is stated.

For examples:

```
  char c,d,e;
  int  x;
  short b;
  string[100];
```

c, d, e are declared to be each a single character. x is declared as an integer. b is declared as short precision and string is a character array of 100 characters. (C numbers the elements starting at 0. So this array has elements 0 through 99, each 1 character in length.)

Variables not only have a type associated with them but also a storage class. The scope ("lifetime") of a variable is the part of, or range of the program in which the variable is defined. There are basically 4 types of storage classes for variables: extern, auto, static and register. An important concept to introduce before explaining these 4 types is the difference between the declaration and the definition of a variable. When a variable is declared, properties of the variable are assigned (type, size, etc.). When a variable is defined (done only once) storage is assigned. Except for use with external variables these terms (definition and declaration) are almost synonymous.

Automatic (auto) variables are variables "local" or "internal" to a single function only. This is the default for variables declared within a function. Auto variables have a scope or "lifetime" only within the braces {} or block in which they are defined. Variables declared as or defaulted to auto, appear as a function is called and disappear when the function ends. Since they have this dynamic "local" nature they are said to be "automatic".

External (extern) variables are "global" in nature. They are permanent and are accessible throughout the entire range of a C program. They can be shared between functions of a single C program and between many C source programs. Only one of the C programs would define the variable, the others would have to only declare it as extern. An external variable definition appears outside of any function. This actually defines the variable as external (extern). External variables are defined outside of any function and declared in the function which uses them.

Static (static) variables are stored in a fixed memory space. They can be external or internal in nature. When declared to be internal in nature they are like auto variables except they remain in existence in their fixed space. External static variables are global in nature but are only accessible within a single C source program. Static variables represent private permanent storage.

Register (register) variables are stored in machine registers. They are used to store heavily used variables in order to improve performance of the C program.

Here is a sample of some variable definitions and declarations:

```
        auto int c;
        extern char b;
        register r1;
        static   x1;
```

As you can see, the storage class and data type can be used to specifically determine the properties of a variable.

That's it for this issues article. Next issue, we'll discuss more about expressions, functions and learn about operators. I will have a sample program and we'll take a look at its design.

WHAT? you ask, stopping now? Well, we can't learn it all in one lesson! Besides, I wouldn't have anything for you to read next issue. See you then, in the mean time my COMPUSERVE ID number is 70575,1330. Send me a message on the LDOS SIG in the C Interest Section!

ALCOR PASCAL

by Scott Loomer

Product: Alcor System's Pascal for the TRS-80 Model I & III

Price: $199

This review of the Alcor Pascal System consists of two parts; first, a general overview of the Alcor package and second, tips on how to maximize its effectiveness under LDOS.

The Alcor Pascal System

The introduction of a complete implementation of the Jensen and Wirth standard Pascal is exciting news for the TRS-80 community. Pascal is a high level programming language that is now widely used in the instruction of programming and the creation of highly maintainable programs. This review will not attempt to teach you much about Pascal. I'll assume that you know enough about the language to be interested in using it on your system. Any good book store will have several texts on the Pascal language which would provide an introduction.

The Alcor Pascal system is currently available on the TRS-80 Model I & III and will soon be available for the Apple II, IBM PC and CP/M according to Alcor. Any programs which do not use the specific system installation library (as described below) will be transportable between the systems.

The Alcor Pascal system comes on two double-density disks for the Model III or three single-density diskettes for the Model I. The system contains several components, each of which will be described below. The components are the text editor, two versions of the Pascal compiler, the linking loader, a string function library, a TRS-80 system installation library, the Pascal run-time interpreter and a set of tutorial files.

The editor (or another text editor) is a necessary first step as the Pascal code must first be prepared. The Alcor editor was written in Pascal and does the job, but as I will discuss later in this article, there are better alternatives. Alcor Pascal is designed to run under T**DOS and therefore has to provide the capability to generate the full ASCII character set that T**DOS doesn't support. The Alcor editor (referred to as the Blaise editor in honor of Blaise Pascal) uses the CLEAR key for several functions such as generating characters that are not directly available on the TRS-80 keyboard. This use of the CLEAR key collides with the LDOS keyboard driver. Another disadvantage of the editor is that it is large (25k) and not very memory efficient. You have only about 13k of buffer space on a 48k machine. If you do use the Alcor editor, do not use the LDOS keyboard driver. The editor has an extensive repertoire of features including HELP commands.

The compilers are the heart of the system. After the Pascal source code has been created, it is compiled into P-code by the compiler. Alcor provides two versions of the compiler. They are identical except that one resides entirely in memory and the other is overlayed. The in-memory version can compile a program of up to about 1000 lines while the overlayed version can handle up to about 4000 lines. To put these line counts in the proper perspective, the Pascal compiler itself had about 8000 lines of source code (as explained by Pascal). Operation of the compiler consists of identifying the source file, object file and listing device or file. Normally, the compilation is displayed on the monitor line by line as it occurs. Errors detected by the compiler are identified by a code number and a pointer to the position where the error occurred. At the conclusion of the listing, each error code identified is explained. The compilation is quite fast; about one line of source per second.

The run-time interpreter is used to execute the P-code. Alcor Pascal would best be described as a semi-compiled language. The P-code produced by the compiler is a compact, tokenized version of the source code. The program is executed by using the run-time interpreter. The resultant execution speed is faster than BASIC but slower than a machine language program. The run-time interpreter is about 16k in length.

An alternative to using the run-time interpreter is provided by the linking loader. The loader allows several object files to be combined into a single program. This can include the run-time support normally provided by the interpreter. The merged file can then be written out as a stand-alone command file. Alcor places no restrictions on the distribution of files created this way other than to require that they be identified as being created by the Alcor Pascal system. The drawback to this type of file is that your small Pascal program (maybe about 3k) has now become 18k+ due to the included run-time support. The best solution for using Pascal programs on your own system is to leave them as P-code and have the run-time interpreter on your system to execute them.

The Alcor system supports extensions to the Pascal standard as defined by Jensen and Wirth. Some of these are in the compiler and others are supported by two system libraries. A few of the compiler supported extensions include common variables, the ability to compile procedures independently and the OTHERWISE clause for the CASE statement. The string library is (will be) standard on all Alcor Pascal implementations.

The strings are dynamically created and can be disposed of to reclaim memory (no garbage collection). The string functions supported essentially duplicate those available in BASIC. The second library supports features unique to the TRS-80. The functions include graphics support, peek and poke, inp and out, inkey and two machine language interface routines. There is the equivalent of the user function in BASIC and a second more powerful routine that allows all primary registers (except the flag) to be initialized and/or read. Any program written that does not use the TRS-80 library functions will be transportable to any Alcor Pascal system.

The final portion of the system is the tutorial. This consists of a 70 page section of the manual and dozens of sample programs. The programs progress through the Pascal language and conclude with a 500 line data base program example. Incidentally, the names of the tutorial files are keyed to the figures in the tutorial section of the manual. The tutorial section provides a good illustration of how Pascal and the Alcor system work.

A few comments about the manual. It is quite extensive, consisting of over 200 pages. There is an introductory section and sections on the text editor, the system installation, the tutorial, as well as a reference section and an index. The reference section is a detailed description of the language. A quick reference card is also provided. Be sure to examine the manual carefully as some important information is not repeated in the section where you might look for it. For instance, the description of how to generate the non-standard characters in the editor are in a section on enhancements to earlier versions of the system.

Alcor Pascal on LDOS

Alcor Pascal is supplied on T**DOS formatted disks and must be patched to run under LDOS. The patches are provided in a special Alcor format which must be applied using the Alcor patch utility (provided). The patch utility needs to be renamed to avoid conflict with the system patch utility; try PPATCH. Alcor recommends applying the patches under T**DOS. If you do so, make sure you are using version 1.3 as earlier versions will not access the entire patch file. This isn't mentioned in the manual, probably because Alcor isn't aware of the problem. It did provide me with a chance to test their customer support. Since the effect appeared to be a glitched file, I called their support number. I explained the problem and they said they would have to research it. They called back within an hour (on their dime) and read me the balance of the patch (it's short) so that I could get started. I sent the disks back and they were replaced (new disks, same serial number).

The weakest portion of the Alcor Pascal system is the text editor. If you do not now possess a text editor or word processor, it'll get the job done. If you have a text editor such as LED, I'd recommend using it as you are already familiar with it. LED works extremely well as a Pascal source editor. I have a Pascal system disk which contains a copy of LED that has been zapped with FED to change the default extension to /PCL. The D patch is:

```
     .Patch to LED to change the default extension
     D05,08="PCL"
     .End of patch
```

LED is compact (6k) and provides all of the features of the Alcor editor. It is also faster since it is machine language. The buffer capacity is nearly 40k! A particularly useful feature of LED is the automatic indent. This creates the same indent on subsequent lines that is traditional in structured code.

Alcor Pascal supports some I/O redirection. When the system prompts for a file name, :C for CRT/keyboard or :L for line printer may be substituted. The standard INPUT and OUTPUT channels in Pascal are automatically prompted for prior to execution (with defaults to the CRT/keyboard).

This means that a simple program that reads from INPUT and writes to OUTPUT will allow you to type to your printer, list a file to the screen, type to a file, etc. Two deficiencies in this I/O redirection (from an LDOS viewpoint) are that you can't easily access devices such as *CL, *SI or *SO and the filespecs MUST BE IN UPPER CASE. Since it takes a few moments for the compiler to load in, it isn't desirable to have it abort because you forgot to use upper case. One solution for this or any command program in which you want to force upper case is to apply the following X patch using the LDOS patch utility:

```
    .Patch to set the caps lock switch on the Model III:
    X'429F'= 20
    .This is the patch for Model I:
    X'4423'= 20
    .That's all there is to it
```

The reverse patch (to force lower case) is:

```
    .Patch to reset the caps lock switch on the Model III:
    X'429F'= 0
    .This is for the Model I:
    X'4423'= 0
    .That's all there is to it
```

I'd recommend applying the first patch to PASCAL/CMD, PASCALB/CMD, RUN/CMD and LINKLOAD/CMD since all of these will abort if a filespec is entered in lowercase. Use the second patch on your editor if you like to create source in lower case. The compiler treats lower case source as if it were upper case.

The Alcor run-time interpreter has to be renamed as RUN is a system command; try PRUN.

Since the normal development of a Pascal program will have you jumping back and forth between the editor and compiler, use KSM to save some time. If you use a standard name for the file you are currently working on such as WORKFILE try the following KSM definitions:

```
   E - led workfile/pcl
   C - pascal;WORKFILE/PCL;;WORKFILE/OBJ;
   R - prun WORKFILE/OBJ
   L - linkload;L;
```

A <CLEAR><E> gets you into the editor and a <CLEAR><C> compiles it for you. <CLEAR><R> then executes the program. <CLEAR><L> gets you into the linking loader and set to specify the file to load.

In summation, let me say that I am very pleased with Alcor Pascal. It is a comprehensive implementation that seems to be bug free except for the following: subrange specifications for integer variables don't work unless the subrange starts at 0. The Alcor folks are aware of this and promise to correct it. Since subranges are not absolutely necessary for any application, this isn't a serious problem. The UPPER CASE restrictions are annoying but not fatal. I've addressed this problem to Alcor and I'm sure if others did as well it would be changed. One other item that should also be requested is access to the flag register in the TRS-80 library machine language interface routine. Alcor also advertises an advanced development package that consists of a P-code optimizer (said to reduce the length of the P-code 10-30%) and a native code generator. The CODEGEN program converts the P-code to Z-80 machine language which will run faster, but expands the length of the file. I plan on examining this package and will report on it in the future.

If you've wanted Pascal for your TRS-80, your wait is over - Alcor is here.

Please address any comments on this review to:

Scott A. Loomer
315 Palomino Lane
Madison, WI 53705
608-233-7739 or MicroNet[70075,1033]

Since I first wrote the review of Alcor Pascal (in August) a couple of things have happened that I wish to pass on to you.

First, the initial copy of the Alcor Newsletter came out. What a pleasant surprise! With the exception of the LDOS Quarterly, I've become resigned to trivial newsletters (if any at all) from software manufacturers. Such is not the case with Alcor. Their first effort is 62 photo-reduced pages of good, solid information. Included in the newsletter is a brief mention of an enhanced version of the Pascal system due out soon (inexpensive upgrade to current owners). Two enhancements listed include random access file support and an include capability. Other enhancements are alluded to. The newsletter included Pascal procedures to allow use of random access files in the interim. This is a good illustration of the power of Pascal and the Alcor implementation. Try writing random access routines for BASIC in BASIC! Other articles describe how to pick up command line arguments in a Pascal program and how to use the machine language interface capabilities of Alcor Pascal to provide bitwise logical operators. All in all, good stuff!

The second item was a mailing from Alcor that corrected a slight error in the random access procedures and also fixed an error in the linking loader. This prompt attention to user-reported errors does credit to Alcor.

LDOS AND PASCAL-80

by D. E. HALL

The power of Standard Pascal with the ease of use of interpretive BASIC (well, almost). And it fits on a Model I or III, is now LDOS-compatible, and is inexpensive, to boot. It's New Classics Software's Pascal-80. With this Pascal, there's no tedious swapping of diskettes or programs to progress from entering the program statements to compilation to execution and back to editing. It's all there, run through a menu of simple commands, in one program, and still leaves you with 23K bytes of memory in a 48K machine. An additional 9K is available once the program has been compiled and stored on disk. This may surprise some of you, but Pascal-80 was written by Phelps Gates, the author of a TRS-80 version of APL, and if he can make that fit, he can make just about anything fit.

An earlier version of Pascal-80 was reviewed by Rowland Archer in the December 1981 issue of Byte. At that time, it was not compatible with LDOS, but it has now been enhanced to be so. According to the manual, the current version has been tested with and will work with DOS Plus, DoubleDOS, LDOS, NewDOS, NewDOS 80 (including version 2), and TRSDOS. It even comes with its own DOS, in case you don't like any of these -- TDOS, from Micro-Systems Software. I have found no compatibility problems with LDOS. Pascal-80 does use its own keyboard driver, so, according to Gates, various "special features" will not work. I have not tried any that didn't work, but Pascal-80 is complete enough to not need much extra.

Archer had several complaints about the version of Pascal-80 which he was using, although he thought it was the best of the TRS-80 Pascals he had tried. Most of these complaints have been taken into consideration and remedied by Mr Gates in the new version. These include the limited DOS compatibility, the lack of an INCLUDE facility (to direct the compiler to compile procedures from a disk file), the lack of an ASCII source file format (so a powerful word processor can be used to create the source program), the lack of an equivalent to (L)BASIC's SET and RESET functions, and minimal documentation. All these complaints have been remedied, leaving an excellent implementation of Standard Pascal for the TRS-80.

Archer also thought that the full-screen editor supplied was minimal for the creation of large programs, but I found it quite acceptable. However, several limitations do still exist in Pascal-80. My main complaint is that Pascal-80 disk file names must be specified at compilation; you cannot specify a file name during execution. It may be possible to get around this using LDOS's ROUTE and LINK commands, but I haven't succeeded yet. Also, Pascal-80 is not quite a full Standard Pascal, but it's very close. The following functions of Standard Pascal are not implemented:

Variant Records

WITH statement

Pointer variables

Structures of files (ARRAY OF FILE, etc.)

Procedures PACK & UNPACK (all structures are packed)

File window (buffer) variables.

Many extensions to Standard Pascal have been built in. These include, but are not limited to, the ability to assign a string which is shorter than the declared array length to an ARRAY OF CHARACTER variable; arrays of CHARACTER may be printed with a single statement; REAL variables may have either 14-digit or 6-digit precision (to save space); a predefined file (along with INPUT and OUTPUT), LP, for the printer; and various other routines specific to the TRS-80. Variable names may be any length, with all characters significant; calculations are done to 14 digit accuracy. A full complement of variable types are available: Boolean, Integer, Char, Real, Real6, and Text. Others may be defined by the user, as well.

Pascal-80's compiler is fast, and the compiled code executes faster than an equivalent BASIC program would; Gates says that execution is generally four to five times the speed of TRS-80 interpretive BASIC, except for the slowing down in calculations caused by Pascal-80's 14 digit accuracy. My experience so far is that it is quite noticeably faster in execution than LBASIC, at least in the word processor I'm writing in both LBASIC and Pascal-80. Even with Pascal's limited string-handling functions, it takes Pascal-80 about 2 seconds to send a justified 60-character line to the printer, to LBASIC's 10 seconds or so. The Pascal version is now roughly 500 lines long, and there has been no indication that I'm near the end of memory. Archer has calculated the upper limit of source lines to be about 1180, depending upon the programmer's style. Though a substantial program can be written in Pascal-80, it is not designed to handle very large disk data files. It can only SEEK (Pascal-80's random access mode) up to the 65,535th byte in a file.

Error messages detected during compilation are pretty good, and whenever an error is found, the offending line is displayed with an arrow under the character that the compiler was looking at when it got confused. The actual error is frequently in the previous line. If after a compilation error is found, you select to edit the program, you are automatically located at the offending line -- no need to search for it.

Unlike some compilers, you are shown only one error at a time, but with the ease of switching from compilation to editing, this is no problem. And you needn't be bothered by "errors" caused by an earlier mistake. Remember those good old FORTRAN programs you tried to run, with one error and 17 error messages?

The procedure to use Pascal-80 is very similar to that used to run LBASIC. You first boot the system, enter any LDOS commands (such as SYSTEM (LOWER) or to enable the printer filter), and enter PASCAL, just as you would LBASIC. Pascal-80 will then display a menu on the screen, allowing you to load a source program from disk, edit a source program in memory, create a new program, append a source disk file to the program in memory, save the source program to disk, compile the source program in memory, run the program in memory (it will automatically be compiled first, if necessary), erase the program in memory, write the compiled code to disk, or execute compiled code that has been stored on disk. Part or all of the source code can be listed on the printer from within the editor. You generally must run PASCAL before you can execute a Pascal-80 program (just as you must first run LBASIC to be able to execute an LBASIC program). However, Gates has included on the Pascal-80 disk two programs which enable you to convert a compiled object program into a /CMD file which may be run from outside Pascal-80. New Classics has kindly allowed registered users to distribute the /CMD versions of Pascal-80- compiled programs, as long as several conditions are met. You may contact them for details.

Pascal-80 is available from New Classics Software, 239 Fox Hill Road, Denville, NJ, 07834, and costs about $100. -- pretty good, considering what many Pascals are being advertised for. I haven't tried the more expensive Pascals, but I expect to stick to Pascal-80. Why meddle when you're satisfied?

Contributed by David Hall, 2343 Wallen Rd, Moscow, ID 83843

PDS - Standard and other types of uses

By Scott Loomer

The Partitioned Data Set (PDS) utility from MISOSYS is probably the most powerful and least exploited utility for LDOS. Let me try to explain, as a user, what PDS can do for you. This article is divided into three parts; first, an explanation of what PDS is designed to do, second, some examples of what it can be made to do, and finally, the future for PDS.

Partitioned Data Set Standard Usage

A partitioned data set is a collection of programs that can be accessed and executed independently. Those are my words and not intended to represent how anyone who knows something about computer science would describe a PDS. Does my description sound like anything you're familiar with? How about the LDOS system libraries, SYS6 and SYS7? They, too, are a collection of programs that can be executed individually and are, in fact, PDSes (PDSi?). Look at the number of separate library functions in SYS6; there are 18. Now since the minimum disk allocation on a Model III is 1.5k (1 granule) that means that SYS6 is at least 18 * 1.5k or 27k in length, right? Wrong. A quick look at the directory shows us that SYS6 is 13.5k long. So ..... that must mean that programs (let's call them members) in a partitioned data set must be allocated just the space they require. That is the first of two major advantages of a POS; programs occupy just the space they need. The second advantage of a PDS can also be illustrated with SYS6. How would you like to have all 18 of those programs stored individually AND showing up in the directory. That would certainly clutter up the display during DIRectories, wouldn't it?

The PDS utility allows you to create your own library files. The utility is, itself, a PDS. It consists of eight programs (members) that are used to create and manipulate PDSes. Use of the PDS directory command on PDS shows us the following:

```
PDS - DIR Library Module - LDOS Version 1.0
Copyright (C) 1981, Roy Soltoff, All rights reserved

PDS: PDS/CMD      07/31/82    Size:   9K Members:  8/ 10
append   P 15-Jan-82  1688    build    P 15-Jan-82   1067
copy     P 15-Jan-82  1031    dir      P 15-Jan-82   1297
kill     P 15-Jan-82   545    list     P 15-Jan-82   1109
purge    P 28-Jan-82  1511    restore  P 15-Jan-82    595
```

The first line of the directory names the file, the date, and the size in both total length of members and the number of members in use vs. the PDS member capacity. Each member is then listed alphabetically, identified as either a program (P) or data (D), and the member's length is given in bytes. If you add up the 8 entries in the directory the total is 8,843 bytes which agrees with the 9k size. An examination of the system allocation directory shows that PDS/CMD occupies 10.5k. The 1.5k difference? Well, there is some overhead for a PDS in the form of a front end loader and internal directory. Together, these total about 400 bytes and are sufficient in this case to bump PDS into another granule. But examine the alternative: on a Model III each of the members would occupy a granule if saved separately with APPEND occupying two granules. That would be a total of 9 granules or 13.5k. Thus, this PDS has saved us about 30% in disk storage space. This is typical of the space you can save by combining several programs in a PDS.

The eight utility programs that comprise PDS accomplish the following:

BUILD - Create a new PDS. Initially it will only occupy the space necessary for the front end loader and the directory. It will expand as members are:
APPEND - Adds a new member to a PDS.
COPY - Copies a member out of a PDS and into an external file.
DIR - Gives a directory of the members of a PDS.
KILL - Deactivate a PDS member.
LIST - Provides the same function as the LDOS system LIST command, but for PDS members.
PURGE - Reclaims the space occupied by KILLed members and compacts the PDS.
RESTORE - The reverse of KILL if the member hasn't been PURGEd.

Two types of members are supported by PDS. These are executable programs (command files) and all else which are treated as data. To execute a program member, you type:

"PDS name(membername)"

Only enough of the member name to be unique is required. You can use the BUILD member of the PDS utility by typing "PDS(BUILD)" or "PDS(B)" or "PDS(BU)". This is another of the features of PDS that I appreciate. I like program names to be as descriptive as possible (within the 8 character limit) but I don't like to type long names. The solution? Store the programs in a PDS with a short name (like 'L' for library), using their full names as the memberspec. If you have done this with the HITAPE/CMD program and there are no other members that start with 'H', it can be invoked with "L(H" - three keystrokes! Programs that require parameters can also take advantage of this abbreviated accessing; to invoke a mirror image backup on my system, I type "L(B) :0 :1". A PDS directory listing of 'L' will give the full name of 'REPAIR' so you can have your cake and eat it too.

Another significant advantage of PDS is its speedy access to the member that you wish to execute. The entire PDS file (which could be as large as available disk space) does not load into memory. Rather, only the short front end loader and directory are brought in. The front end loader determines from the directory where on the disk the start of the member is and directs LDOS to begin execution of the file at that point. The member is then brought into memory and executed. PDS uses an ISAM key to select the appropriate portion of the file to execute.

PDS also provides the ability to have multiple entry points into a single program. Using the MAP parameter of the PDS APPEND command, you can specify several different member names and entry points into a single program being added to a PDS. The individual portions of the program are executed by specifying the appropriate member name.

Non-standard Applications of PDS

Even with a utility as inherently useful as PDS, I am seldom satisfied to leave my hands off. Therefore, I have developed some very useful (to me at least) applications for PDS that are not exactly what it is designed for.

The first of these applications is the HELP utility (available from MISOSYS for $25). The HELP utility is a series of command files that explain features of LDOS when invoked with HELP(command name). The explanations are quite detailed showing allowable abbreviations and default values for parameters as well as the function and syntax of the command. The HELP files are all PDSes. I wanted to have an easy, quick means of displaying the HELP screens. The solution was to create each HELP command display as a command file origined to the video display screen. Placing the HELP member into a PDS provided compact storage of many displays and handled the overhead of accessing a particular entry. Only one problem remained. Since the HELP screens are directly origined to the video display, they overlay what was previously on the screen. Rather than make the HELP members unnecessarily large to blank all portions of the screen, I came up with the following patch which causes the screen to be cleared before the member is executed:

```
. PDSHELP/FIX
. Patch to PDS/CMD utility to clear screen before
.   executing member
. Jump to patch
D00,2F=C3 D7 52
. Change "PDS member required" message to make room
D00,FE=6E 65 65 64 65 54 21 0D
. Patch to clear the screen
D01,06--CD C9 01 7E 23 FE 28 C3 04 52
. End of patch
```

The HELP utility includes this patch as well as instructions for creating your own custom HELP files using PDS. A BASIC program is also included to convert text files created with a word processor into the load module format required for executable PDS members. A point that should be made here is that any LDOS system can use a PDS file so you do not need the PDS utility to use HELP. The utility is only required to create and modify PDS files.

A second non-standard application was more involved. I wanted to create a telephone dialer to use with my Hayes SmartModem. To accomplish what I had in mind, I needed to be able to do some pre-processing (create a blank display with titles), execute the PDS member (which would fill in the display blanks with names and numbers) and finally do some post-processing (dial the number). The PDS utility would give the capability to maintain the phone list. My requirements indicated that I would have to write my own front end loader.

The PDS BUILD command allows you to specify use of your own front end loader (FEL) instead of the standard one provided by PDS. An examination of the standard FEL shows it to be one of the tightest, and most efficient (and consequently confusing) pieces of code that I've encountered. So much for writing my own FEL since I would need to duplicate and extend what is accomplished by the standard FEL. Well, not exactly.... there was a coward's way out. I decided to use the standard FEL and add my own routines. The FEL is a command file itself so that it is possible, using CMDFILE, to append other routines to it and then change the transfer address to execute the new code before jumping back to the standard FEL. A general procedure for creating a semi-custom FEL is as follows:

1. Isolate the standard PDS FEL by:
   a. Create a temporary PDS with the command:
      PDS(BUILD) TEMP
   b. Run CMDFILE. Load your temporary PDS. The load address reported will be 5200 to 52E0 with a transfer address of 5200. CMDFILE has loaded in just the front end loader from the PDS.
   c. Save the front end loader as STANDARD/FEL.
2. Add your new code to the standard FEL by:
   a. Assemble your new code with an origin of 52E1H. After you accomplish what you need to, end your new code with a jump to 5200H to execute the original FEL. If you need to do something after the member executes, have the member's code jump back into the code you are adding to the FEL. You should then terminate this post-processing code with a jump to @EXIT at 402DH.
   b. Run CMDFILE. Load in STANDARD/FEL and then your new object code. Save the merged code with a new transfer address of 52E1H and a suitable name (with an /FEL extension).
3. There is a pointer in the standard FEL that must be adjusted to make your new FEL work correctly. This byte, located in record 0 at relative byte 70H, points to the relative byte after the last byte of your FEL. The easiest way to determine the new value is to use FED. Load in your new FEL and use the <E> command to go to the last byte of your FEL. Move the cursor one byte forward and note the relative byte address displayed by FED. This is the new value that must be inserted into the original FEL code. Use FED to change the byte at record 0, relative byte 70H from 14H to the value you just determined and save the changed file. You have finished the creation of your custom FEL.
4. To create a PDS using your new FEL the command is as follows:

```
PDS(BUILD) filespec (LOADER="NEWFEL/FEL",MEMBERS=dd)
```

If any of you are in possession of both the PDS utility and a Hayes SmartModem, contact me if you would be interested in the autodialer.

The Future of PDS

The future for PDS utilization is bright. Some of the current or near term enhancements or applications are:

1. The directory command in LDOS 5.1.3 now indicates PDS files with an asterisk in the attributes column.

2. The newly arrived enhanced EDAS version IV allows the user to create PDS libraries of assembler source code for standard applications. The EDAS command *SEARCH will cause EDAS to search the referenced PDS for all members that will resolve undefined references in the source. This powerful capability allows you to create standard routines and access them by name.

3. The C compiler, LC, also released recently, makes use of PDS. The standard library and installation library for LC is provided in the form of PDS files. Any references in the C source code to these library functions is automatically resolved at compilation time. Note that you will not need to have PDS to use LC. Any PDS file can be used on any LDOS system without the PDS utility which is only needed to create the file. If you do have the PDS utility, however, you will be able to create your own C support libraries of your personal routines.

4. Lastly, it is possible that if interest in PDS warrants the development, we will see PDS expanded to allow read only access from any application program. The PDS members could then be Scripsit documents, BASIC data files, etc. and be accessed from within the application by specifying the PDS filespec (memberspec).

Please direct any comments concerning this article to:

Scott A. Loomer
315 Palomino Lane
Madison, WI 53705
MNet [70075,1033]

NEWSCRIPT 7.0 and REFLEX.

Some Further Notes.

Gordon B. Thompson,
Bell-Northern Research
P.O. Box 3511, Stn C,
Ottawa, Ontario. K1Y 4H7,
Canada

Since the original article on The Communicating Micro appeared in the October 1982 issue of the LDOS Quarterly, a solution to running Prosoft's very powerful word processor, NEWSCRIPT 7.0, with REFLEX has been found. The result is a very powerful word processor that can fully communicate with another of its kind in the fully cloned fashion described in the previous article, can transfer the contents of its working memory, can boot in double density on Model I, and can easily reconfigure for different printers, etc.

NEWSCRIPT has its own powerful drivers for screen, printer and keyboard. Although this driver package is not relocatable, it is, however, tailored to suit the user's needs, and so may vary in length. The address stored in the system keyboard DCB at H4016 & 7, is a vector pointing to both the NS/CMD keyboard driver and to NEWSCRIPT's communal scratch memory area. Throughout the various component parts of NEWSCRIPT frequent use is made of the value stored in the DCB. As a result, installing REFLEX/FLT, which changes the contents of the DCB, altered, causes no end of trouble. The solution is to fool NS/CMD by installing the filter and then rethreading the control sequences so NS/CMD never knows there is a filter active.

NS/CMD's keyboard driver begins by immediately jumping over the adjacently located scratch memory area. If instead of jumping deeper into NS/CMD, control is sent to REFLEX, and then passed back to the spot where that jump was to have landed, the original NS/CMD address can be left in the DCB. This change is accomplished without altering the disk copy of NS/CMD by simply patching RAM with RAMPATCH/BAS, Listing 1, and capturing the result with a SYSTEM (SYSGEN) command.

Unfortunately, the jump to REFLEX involves jumping too many bytes for a two byte jump instruction. Consequently, one byte of NEWSCRIPT's scratch memory must be usurped to accommodate the third byte of the absolute jump command. I have yet to run into problems with this strategy. It seems as if this particular memory location is rarely used.

REFLEX was intended to be controlled from the LDOS keyboard driver. As NEWSCRIPT's keyboard driver is different, a new way of controlling REFLEX must be established. This is accomplished with RAMPATCH/BAS. NS/CMD's special instruction mode, the <SHIFT><CLEAR> command, is changed so that the two commands that switched the special character sets of the Model III become REFLEX's commands instead. These commands were unused in Model I.

To use REFLEX with NEWSCRIPT, it is necessary to relocate REFLEX's mode indicator letters from the top right screen corner to the bottom right corner elsewise they will be interpreted by the EDITOR package as edit instructions! REFLEX/FLT's references to 3C38 should be changed to 3CFF.

NEWSCRIPT's NS/CMD carries its own JCL feature, using the file STARTUP/MIN for its instructions. Prosoft provided this feature to boot the system. The LDOS configure feature is very much quicker for everyday booting, so the NS/CMD JCL function can be saved for generating fresh configurations. STARTUP/MIN is shown in Listing 2. Although the published version of REFLEX did not load under LDOS's JCL routines, it does load with NEWSCRIPT's JCL function.

For the Model III, the procedure is very simple. An LDOS system disk with the RS232T driver, REFLEX/FLT, NS/CMD, STARTUP/MIN, NSINSTAL/BAS and EDIT, SCRIPT and NSINIT, is prepared. NSINSTAL/BAS is run. This operation sets NS/CMD for your particular printer and typing style. Once this has been done, NS/CMD is run. This invokes STARTUP/MIN file, which loads the proper drivers and filters, runs RAMPATCH/BAS to rethread control, and finally, captures this configuration with a SYSTEM(SYSGEN) command.

For the Model I, the procedure is similar, except that a single density disk carrying all the requisite drivers, including PDUBL and NS/CMD, STARTUP/MIN, NSINSTAL/BAS must be assembled and it is used to build new configurations. A double density working disk is put together that carries NEWSCRIPT's EDIT, SCRIPT, NSINIT and SOLE2/CMD. A two sided disk works very well, one side for the single density builder, and the other for the double density working material. Once the new configuration has been captured on the double density disk, and "SOLE2d" in, the double density side can boot directly. The single density side is only needed when the configuration is being altered in a major way. Additional details on the original construction double density NEWSCRIPT disks, see "Using NEWSCRIPT 7.0 with Model I Double Density" in October 1982 LDOS Quarterly, page 55.

Two minor points: If AUTO LBASIC RUN"NSINIT is used to make the disk automatically get into NSINIT, NEWSCRIPT's superMENU, then the first key stroke is lost. Secondly, the continuous delete mode is not sent over the REFLEX link, only a string of ddd's. The toggle delete must be used instead.

EDIT will send the contents of its workspace out the communications line by merely saving to *SI in place of a filename. Text received from the communications line can be directly written into EDIT's workspace by being in REFLEX and in an EDIT input mode.

Listing 1.

RAMPATCH/BAS

```
10 '       ***  RAMPATCH FOR NS/CMD  ***
   11 '      REFLEX/FLT MUST BE ALREADY INSTALLED.
   12 '      Reference: "The Communicating Micro"
   13 '          October 1982 Issue, LDOS Quarterly
   14 '      +++  VARIABLE ASSIGNMENT:  +++
   15 '    R    Entry point for REFLEX/FLT
   16 '    N    Entry point for NS/CMD's keyboard driver
   17 '    D    Target of Jump Relative at start of
   18 '               NS/CMD's keyboard driver.
2$ 'Find values for R, N and D:
   21 R1=PEEK(&H4016):R2=PEEK(&H4017):R=256*R2+R1-65536
   22 N1=PEEK(R+1):N2=PEEK(R+2):N=256*N2+N1-65536
   23 D=N+66:D2=INT((D+65536)/256):D1=D+65536-256*D2
3$ 'Rethread command control sequence:
   31 POKE (&H4016),N1:POKE (&H4017),N2
   32 POKE N,&HC3:POKE N+1,R1:POKE N+2,R2
   33 POKE R+1,D1:POKE R+68,D1
4$ 'Patch NS/CMD so the ?? s and j commands are replaced
   41 '   with new commands that control REFLEX:
   42 '  <SHIFT><CLEAR> i   Puts REFLEX in LOCAL mode.
   43 '  <SHIFT><CLEAR> u   Puts REFLEX in REFLEX mode.
   44 I  <SHIFT><CLEAR> y   Puts REFLEX in TERMINAL mode.
   45 DIMV(20):FOR K=1TO19:READ V(K)
46 POKE(N+185+K),V(K):NEXT K
49 DATA 254,105,40,08,254,117,40,04,254,121,32,20,198,128,201,00,00,00,00
50 CMD"S
```

Listing 2

STARTUP/MIN

```
SET *SI RS232T(DTR=Y,RTS=Y)
FILTER *KI REFLEX
LBASIC RUN"RAMPATCH/BAS
SYSTEM(SYSGEN)
```

For Model I, the appropriate double density driver must be installed. The line calling it can be placed at the beginning of STARTUP/MIN. Also, the RS232R driver should be used for Model I. Otherwise, Models I and III are identical.

VISICALC and REFLEX.
Some Further Notes.

The patch given in "THE COMMUNICATING MICRO" article in the October 1982 issue of the LDOS Quarterly for VISICALC to make it send and receive the contents of its workspace over the communications line may appear to be faulty in that the receiving machine flashes horizontal bars instead of accepting the incoming code.

The problem is that VISICALC requires a finite time to to organize its memory. The difficulty is overcome by having the receiving machine move to the lower right hand corner of the sheet that will be sent, and enter something in that cell. Now, incoming data can then be accepted without pause.

When VISICALC is loading from disk, you may have noticed how it accepts a token gulp, pauses, and then gets going again. It's the same thing.

MISOSYS MISOSYS MISOSYS MISOSYS MISOSYS MISOSYS MISOSYS MISOSYS MISOSYS MISOSYS MISOSYS MISOSYS

LC Compiler

The "LC" Compiler provides a substantial subset of the C programming language with:

- o Integer subset of C; has access to floating point ROM routines via functions
- o All statements supported except: SWITCH-CASE, GOTO, TYPEDEF, STRUCT, UNION.
- o All operators except "->", ".", SIZEOF, (TYPENAME).
- o Standard I/O redirection with device independence.
- o Input using FGETS or GETS functions support JCL.
- o Dynamic memory management.
- o Sequential files open for: READ, WRITE, and APPEND.
- o LC Generates Z-8Ø EDAS-IV source code as output.
- o Z-8Ø "source" libraries in ISAM-accessed PDS files.
- o Compact, one-line compiler invocation for easy use.
- o Compiled programs run on both Model I and Model III
- o IN/LIB accesses graphics and LDOS entry points.
- o LC/LIB includes: FPRINTF, PRINTF, ALLOC, FREE, SBRK, and String functions.
- o LC: The Mod I/III version includes: LC/CMD, LC/LIB, FP/LIB, IN/LIB, EDAS-IV, XREF, and more than 2ØØ pages of documentation. Requires 2-drive 48K LDOS.

Mod 1&3 for LDOS: $15Ø+$4S&H

MISOSYS MISOSYS MISOSYS MISOSYS MISOSYS

LDOS, Version 5.1 is the Ultimate in Operating Systems for the TRS-80 Models I and III. MISOSYS is your East Coast Headquarters for LDOS, the documented system! Version 5.1 is priced at $129.ØØ + $5.00 S&H per system. Deduct $35 if ordering both.

LDOS 5.1

GRASP

Finally, user customized character sets for your MX-8Ø Graftrax and MX-1ØØ printers. With **GRAphic Support Package** you create character sets usable from any and all applications. Create character sets of single-width, double-width, & 12-pitch, using the **GRASP** character editor, ALTCHAR. **GRASP** comes supplied with 7 character sets. Print Mod3 special characters on your Epson! Set MX options from your keyboard. Invoke underlining! Written by K.A.Hessinger & S.A.Loomer. **GRASP: $5Ø+$2S&H**

VISA

MISOSYS - Dept. M5
P.O. Box 4848
Alexandria, VA 223Ø3-Ø848
7Ø3-96Ø-2998
Dealer's Inquiry Invited

MasterCard

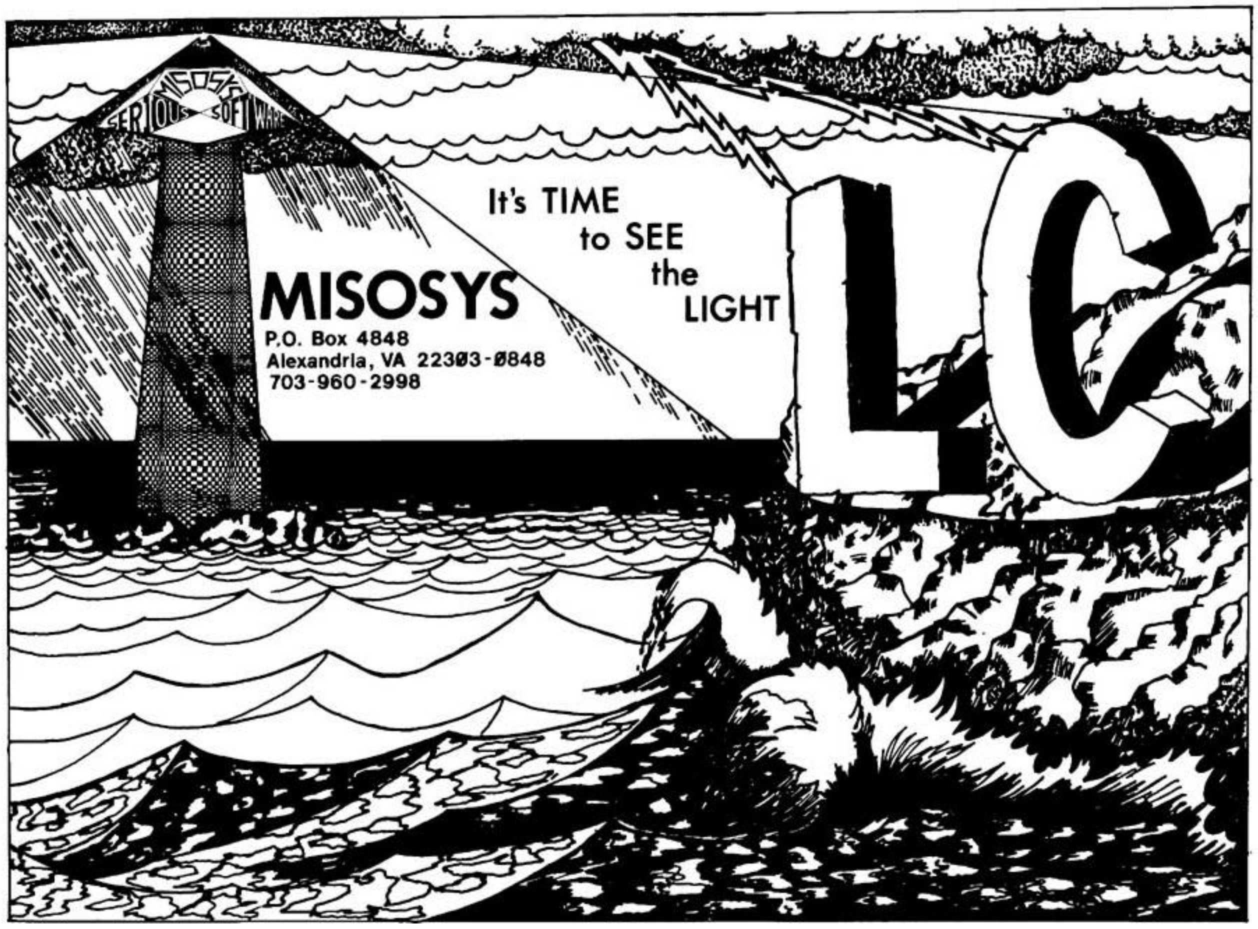
~123

EDAS-IV

EDAS, Version IV is the most fantastic absolute address assembler, bar none! It has:

- o Assemble to disk or memory from multiple source files nestable to FIVE levels.
- o Assemble conditional code with IF, IFLT, IFEQ, IFGT, IFDEF, IFNDEF, IFREF to 16 levels with IF-ELSE-ENDIF.
- o Automatic search of SOURCE subroutine libraries saved in ISAM-accessed structure to resolve references left undefined (requires PDS).
- o One-level MACROs support parameter substitution by position and by keyword.
- o Local labels in both MACRO expansions & PDS searches.
- o Supports +, -, *, /, .MOD. .AND., .OR., .NOT., .XOR.
- o Constants can be declared as base 2, 8, 1Ø, & 16 or string, with more than one value on a single line.
- o 15-char labels including special chars: @, ?, $, _
- o Extensive cross-reference utility & EQU generation.
- o Enter source in upper case or lower case. Line editor has COPY, CHANGE, and MOVE
- o Pseudo-OPs LORG, COM, PAGE TITLE, SUBTTL, SPACE.

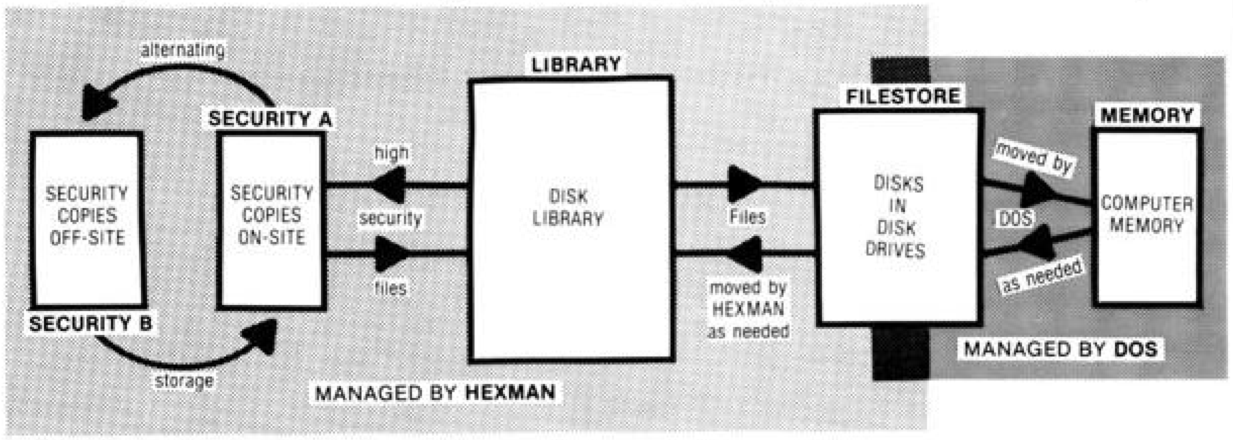
Model I/III for LDOS: $1ØØ
Model II for TRSDOS: $2ØØ
Add $4 S&H + 4% VA tax.



HEXMAN D.M.S.

DISK MANAGEMENT SYSTEM

Computerized file storage and retrieval for your whole disk library.



Your Disk Operating System makes a fine job of managing the files that are in your disk drives. No need to worry about where your old files are kept or where to store new files, your DOS takes care of it. Now HEXMAN takes over where your DOS leaves off, and extends worry free file storage and retrieval to your whole disk library.

Storage Management

Only a small portion of your disks can be in your disk drives at any one time. These disks form the "Filestore" to which your computer has immediate access, and the remaining disks form your disk "Library". Ideally the "Filestore" should contain the most active files, while inactive files are kept on "Library" disks stored near the computer.

HEXMAN comes as close to this ideal as it can by monitoring file activity and keeping the most frequently used files in the Filestore. If previously inactive files are needed, they are moved in from the Library. If the Filestore becomes full, the least active files are moved out to the Library. The net result is the files you are most likely to use are ready and waiting.

HEXMAN also performs other storage management chores such as daily backups of modified files, on-site and off-site storage of security copies, and file growth monitoring.

File Retrieval

HEXMAN manages up to 8000 files (2000 originals with up to 4 copies of each). If remembering that many eight letter file names gives you a headache — don't worry. HEXMAN gives you two easy ways to find the files you need. With HEXMAN you find files by function e.g. "Get all the files I need for the end of month Sales Reports" or by description e.g. "Find the letter I sent to Radio Shack about my disk drives". HEXMAN finds your files in seconds. Once you have found them you may Review them, Load them or Delete them as needed.

Requirements

Model III — 48K, 2 drives.

Model I — 48K, 2 drives. Double Density adaptor. Lower case modification.

LDOS operating System. If you do not have LDOS, please add $35 for a copy of the smal-LDOS operating system.

US $169

90-day trial - just $20.

We are sure HEXMAN will become a "must have" program for all serious TRS-80 users. But check it out for yourself. Purchase HEXMAN now and try it out. If you feel you can live without it (unlikely!), return it for a prompt refund less a $20 evaluation charge.

Offer valid for direct sales only - dealer terms may vary.

WARNING — TIMELOCKED SOFTWARE

This product is timelocked. Unregistered copies will expire at a future date. Registered owners will receive an extension patch when needed.

A NOTE ON CANADIAN MAIL — please allow 5 to 10 days in each direction for mail delivery. For faster service leave your orders or enquiries on our 24-hour answering machine.

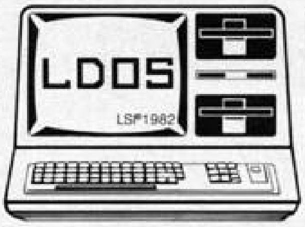
HEXAGON SYSTEMS

P.O. Box 397, Station A
Vancouver, B.C. Canada V6C 2N2
Telephone (604) 682-7646
Electronic Mail-Micronet 70235,1376

LDOS MICRO REVIEW

Volume 1 No. 1 ★ SPECIAL EDITION ★★★ December 1, 1982

The BASIC Answer



You'll think you've made the DOS strike of the decade when you turn your micro on to LDOS. You'll find a bonanza of features like full keyboard type-ahead; a true background spooler; file backup by date, class, and between different drive types; hard disk support; data transportability between Model I and III; and a complete communications utility including disk file send and receive. Support for Radio Shack's Doubler and selected others is also provided.

With our Job Control Language, you get true "hands off" running of your application programs — give a single command and then walk away. The 400 page manual includes examples of all commands and utilities. The Operator's Guide gives you step by step instructions on how to use LDOS with your applications. Stop running with only "half" a computer! Let LDOS provide the missing features to speed up and simplify your TRS-80 computer system! Visit a dealer or contact LSI for more information on the most popular sophisticated operating system for your TRS-80.

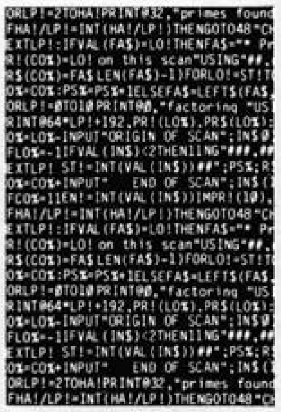
LDOS is available worldwide through thousands of dealers for just $129.

The BASIC Answer is a BASIC text processing utility. It is designed to allow the BASIC programmer to build code in a structured manner. "Source" code is written with a word processor or text editor which allows the user to exploit the powerful editing and movement features characteristic to those types of editors. Source code can even be created by your own BASIC interpreter. **The BASIC Answer** is then used to process these files into normal interpretive BASIC code.

Free Yourself from Line Numbers

The BASIC Answer allows substitution of labels for **line numbers!** This means that your BASIC code now can read like a novel. Instead of the typically undescriptive "GOSUB 1000", a label such as "GOSUB @Search.Name" is used. Imagine yourself reading code filled with such descriptive branches and understanding it at a glance, even years later. This feature even allows totally relocatable BASIC routines without the renumbering problems.

TRS-80 is a trademark of Tandy Corporation. LDOS is available for the TRS-80 Model-I and Model-III. Prices and specifications subject to change without notice. LDOS and The BASIC ANSWER are products of Logical Systems, Inc.



```
ORLP!=2TOHA!PRINT@32,"primes found
FHA!/LP!=INT(HA!/LP!)THENGOTO48"CH
EXTLP!:IFVAL(FA$)=LO!THENFA$="* Pr
R!(CO%)=LO! on this scan"USING"##,
R$(CO%)=FA$LEN(FA$)-1)FORLO!=ST!T
O%=CO%:PS%=PS%+1ELSEFA$=LEFT$(FA$,
ORLP!=0TO10PRINT@0,"factoring "US
RINT@64*LP!+192,PR!(LO%),PR$(LO%):
O%=LO%-INPUT"ORIGIN OF SCAN";IN$0
FLO%=-1IFVAL(IN$)<2THENIING"###,##
EXTLP! ST!=INT(VAL(IN$))##";PS%;R
O%=CO%+INPUT"   END OF SCAN";IN$(
FCO%=1IEN!=INT(VAL(IN$))IMPR!(10),
FHA!/LP!=INT(HA!/LP!)THENGOTO48"CH
EXTLP!:IFVAL(FA$)=LO!THENFA$="* Pr
R!(CO%)=LO! on this scan"USING"##,
R$(CO%)=FA$LEN(FA$)-1)FORLO!=ST!T
O%=CO%:PS%=PS%+1ELSEFA$=LEFT$(FA$,
ORLP!=0TO10PRINT@0,"factoring "US
RINT@64*LP!+192,PR!(LO%),PR$(LO%):
O%=LO%-INPUT"ORIGIN OF SCAN";IN$0
FLO%=-1IFVAL(IN$)<2THENIING"###,##
EXTLP! ST!=INT(VAL(IN$))##";PS%;R
O%=CO%+INPUT"   END OF SCAN";IN$(
ORLP!=2TOHA!PRINT@32,"primes found
FHA!/LP!=INT(HA!/LP!)THENGOTO48"CH
```

A New Concept in Variable Usage

The BASIC Answer allows variable names to be as long as 14 characters and ALL 14 are significant. Imagine reading:

"IF ACCNT.OVERDUE #> 0 THEN GOSUB @PRINT.DUN"

rather than

"IFAO#>0THEN GOSUB52130"

Which would you rather read? It also introduces to BASIC the concept of Global and Local variables. This feature circumvents the tedious problem of variable tracking because a Local variable is only viable in its own subroutine!

NOW AVAILABLE
LDOS 5.1 Quick Reference Card
$5.95.

End the Multiple Machine Hassle

The BASIC Answer introduces the concept of **"Conditional Translation."** This feature allows the programmer to place different "machine dependent" code simultaneously into the same Source Code. **The BASIC Answer** can be "switched" when processing to ignore the unwanted or include extra code! No more multiple master programs to confuse maintenance. All the masters could now be rolled into the same program. Modify the one master and you've modified them all. Process the same code with different switches set, and get two or more versions from the same source.

The BASIC Answer combines the self-documenting power of COBOL with the relative ease of BASIC together with the power of a word processor.

The BASIC Answer is available for just $69.00.

LOGICAL SYSTEMS INC.

11520 N. Port Washington Rd.
Mequon, WI 53092
(414) 241-3066

M E M O R Y   M A P - A L P H A B E T I C   L I S T I N G

This memory map section is provided to allow quick lookup of a memory address corresponding to an LDOS system label. An asterisk marks those addresses which are different on the <Mod I> and [Mod III].

```
   @ABORT----<4030>,[4030]      @PAUSE----<0060>,[0060]      JFCB$-----<4358>,[4265]*
   @ADTSK----<4410>,[403D]*     @PEOF-----<4448>,[4448]      JLDCB$----<43C0>,[42C2]*
   @BKSP-----<4445>,[4445]      @POSN-----<4442>,[4442]      JRET$-----<430C>,[4222]*
   @CKDRV----<44B8>,[4209]*     @PRINT----<446A>,[446A]      KFLAG$----<4423>,[429F]*
   @CKEOF----<444B>,[4458]*     @PRT------<003B>,[003B]      KIDCB$----<4015>,[4015]
   @CLOSE----<4428>,[4428]      @PUT------<001B>,[001B]      KIJCL$----<43BE>,[42BE]*
   @CMD------<4400>,[4296]*     @RAMDIR---<4396>,[4290]*     KISV$-----<43B8>,[42B8]*
   @CMNDI----<4405>,[4299]*     @READ-----<4436>,[4436]      LDRV$-----<4308>,[4427]*
   @CTL------<0023>,[0023]      @REW------<443F>,[443F]      MFLAG$----<442F>,[ N/A]*
   @DATE-----<4470>,[3033]*     @RMTSK----<4413>,[4040]*     MULTEA----<4B6C>,[4B6B]*
   @DEBUG----<440D>,[440D]      @RPTSK----<4416>,[4043]*     OSVER$----<403E>,[441F]*
   @DIV------<44C4>,[4451]*     @RREAD----<4454>,[445E]*     OVRLY$----<430E>,[4414]*
   @DODIR----<4463>,[4419]*     @RUN------<4433>,[4433]      PDRV$-----<4309>,[4423]*
   @DOKEY----<44BE>,[4285]*     @RWRIT----<4457>,[4461]*     PRDCB$----<4025>,[4025]
   @DSP------<0033>,[0033]      @SKIP-----<4460>,[4464]*     PRSV$-----<43BC>,[42BC]*
   @DSPLY----<4467>,[4467]      @TIME-----<446D>,[3036]*     RDSECT----<4777>,[4777]
   @DVRHK----<4033>,[4033]      @VER------<443C>,[443C]      RDSSEC----<4B45>,[4B45]
   @ERROR----<4409>,[4409]      @WEOF-----<444E>,[445B]*     RSELCT----<4759>,[4759]
   @EXIT-----<402D>,[402D]      @WHERE----<000B>,[000B]      S1DCB$----<43D8>,[42D4]*
   @FEXT-----<4473>,[444B]*     @WRITE----<4439>,[4439]      S2DCB$----<43E0>,[42DA]*
   @FNAME----<44BB>,[4293]*     CFCB$-----<4480>,[4485]*     S3DCB$----<43E8>,[42E4]*
   @FSPEC----<441C>,[441C]      DATE$-----<4044>,[421A]*     S4DCB$----<43F0>,[42E6]*
   @GET------<0013>,[0013]      DAY$------<4047>,[4417]*     S5DCB$----<43F8>,[ N/A]*
   @ICNFG----<4303>,[421D]*     DBGSV$----<405D>,[405D]      SBUFF$----<4200>,[4300]*
   @INIT-----<4420>,[4420]      DCT$------<4700>,[4700]      SEEK------<475E>,[475E]
   @KBD------<002B>,[002B]      DCTBYT----<479C>,[479C]      SELECT----<4754>,[4754]
   @KEY------<0049>,[0049]      DFLAG$----<441F>,[4289]*     SFCB$-----<44A0>,[42A1]*
   @KEYIN----<0040>,[0040]      DRCYL-----<4B65>,[4B64]*     SFLAG$----<430F>,[442B]*
   @KILL-----<442C>,[442C]      DIRRD-----<4B10>,[4B10]      SIDCB$----<43C8>,[42C8]*
   @KITSK----<4300>,[4285]*     DIRWR-----<4B1F>,[4B1F]      SODCB$----<43D0>,[42CE]*
   @KLTSK----<4419>,[4046]*     DIVEA-----<4B7B>,[4B7A]*     SVDAT1$---<4306>,[442F]*
   @LOAD-----<4430>,[4430]      DODCB$----<401D>,[401D]      SVDAT2$---<4307>,[4457]*
   @LOC------<445A>,[446D]*     DOSV$-----<43BA>,[42BA]*     TCB$------<4500>,[4500]
   @LOF------<445D>,[4470]*     GETDCT----<478F>,[478F]      TIME$-----<4041>,[4217]*
   @LOGER----<447E>,[428D]*     HIGH$-----<4049>,[4411]*     TIMER$----<4040>,[4288]*
   @LOGOT----<447B>,[428A]*     INBUF$----<4318>,[4225]*     USTOR$----<4DFE>,[4DFE]
   @MSG------<4479>,[4402]*     INTIM$----<404B>,[4473]*     VERSEC----<4772>,[4772]
   @MULT-----<44C1>,[444E]*     INTMSK$---<404C>,[4474]*     WRPROT----<4768>,[4768]
   @OPEN-----<4424>,[4424]      INTVC$----<404D>,[4475]*     WRSECT----<4763>,[4763]
   @PARAM----<4476>,[4454]*     JDCB$-----<430A>,[4220]*     WRTRK     <476D>,[476D]
```

Communication Host

by James F. Bruckart

The computer revolution has introduced computers and terminals into the workplace at an ever increasing pace. As home computer owners, terminal programs and modems have given us access to mainline information systems and community bulletin boards. Until the introduction of LDOS, most TRS-80 applications required special programming to function in anything except a terminal mode. But the device independence of LDOS has introduced the ability to use your TRS-80 has a host computer.

As a military physician, I perform histories and physicals on all patients I admit to the hospital, and compile data related to their complaints and diagnoses for future research. Such data manipulation is perfectly suited for a TRS-80 database, but frequent movement in the hospital and among hospitals makes it inconvenient to transport my computer. However, I have found that any computer terminal provides a link to my database.

The problems of maintaining a host system for terminal access breakdown to operating software, system configuration, and system security. LDOS solves the first problem. Using the links described in the LDOS manual (LINK *KI *CL and LINK *DO *CL) your computer is in a host mode. Any characters you type at a distant terminal will be treated as if you typed them at the keyboard, and characters displayed on your video display will appear on the terminal display. In this mode your computer will run the familiar LDOS commands and Basic programs as if you're sitting at home at the keyboard.

Getting started will require LDOS operating on a computer with an RS232 interface (or similar hardware) and an autoanswer modem. The JCL file in listing 1 will configure your system for initialization of the host mode. Turning the date switch off and setting the DO command to AUTO will cause the system to reconfigure with power-up or reset. I plug my CPU into an appliance timer that turns the computer off for a minute at noon each day. This way the computer recovers at noon if I crash the system in the morning.

A problem you may encounter when using most dumb terminals; the carriage return (CHR$(13)) causes a return and linefeed on the TRS-80, but only a carriage return on the terminal. Each new line of text will simply be printed overtop the previous line. ADDLF/FLT is a filter for the communication line that solves this problem by issuing a line feed (CHR$(10)) after each carriage return.

The final problem of running a host system is security. Others will learn that your computer is "on-line" and one day your valuable files may be lost. My Basic program COMMPASS/BAS can prevent unauthorized persons from roaming into your data and operating system. After calling your computer, type enter or return and the computer will cue you for a password. Incorrect entries and the time they occur will be logged on your line printer. After 5 incorrect entries, the computer issues a null break (should disconnect the other users modem), and returns to wait for your call. When you issue the correct password, the computer returns to the LDOS Ready prompt. End your session by typing LBASIC RUN"COMMPASS/BAS" or BOOT from LDOS Ready. Special features of the Compass program include initial checking for the line printer and disabling of the break key (to prevent the unauthorized user from "Breaking" out of the program).

In summary, LDOS provides the key to initializing your personal host computer system. Configuring the computer (after purchasing an autoanswer modem) is simply performed with a JCL file loaded at power-up. The problem of system security is relegated to a simple Basic program where you specify the password.

Hints to those getting started:

1. Try each program at the keyboard and watch the modem lights. What you type is equivalent to modem input, and modem output will be indicated by the transmit data light.

2. Programs which displace the keyboard driver or use the RS232 will probably not work with this configuration (i.e.- Scripsit or Profile). You or a friend should patch these programs to use the LDOS drivers.

```
.Communication Host JCL Initialization
.Set Modem to on and auto-answer
SET *CL TO RS232x/DVR
. - fill in the proper driver and parameters
. for your hardware
FILTER *CL using ADDLF/FLT
SYSTEM (DATE=OFF)
AUTO DO = COMMHOST
LINK *KI *CL
LINK *DO *CL
LBASIC RUN"COMMPASS"
//STOP
```

```
10 REM ** Communication Password Screen
20 CLEAR 500: CMD"B","OFF": CLS: PRINT TIME$: B$=""
30 PRINT@256,"Printer not ready": LPRINT CHR$(13): CLS: PRINT@256,"Computer ready for
communications": REM Checking if printer ready - if you don't have a  printer omit
lines 30 and 80.
40 A$="":INPUT A$
50 IF A$="" THEN GOTO 60 ELSE GOTO 40
60 FOR I=1 TO 5: PRINT: LINEINPUT "Password: ";B$
70 IF B$="PASSWORD" THEN GOTO 100: REM INSERT YOUR OWN PASSWORD
80 LPRINT TIME$,B$
90 PRINT "Incorrect - your error has  been logged": B$="": NEXT: PRINT STRING$(100,1)
:GOTO 20: REM EXECUTING MODEM BREAK
100 PRINT "Correct - returning to LDOS Ready"
110 CMD"B","ON": CMD"S"
```

```
00110 ;****  ADDLF/FLT - adds linefeed after carriage return
00120 ;
00130 ;    see Filters and Drivers section of LDOS
00140 ;    Technical Information in Owner's Manual
00150 ;
00160 LF      EQU     10
00170 CR      EQU     13
00180 EXIT    EQU     402DH
00190 ABORT   EQU     4030H
00200 DSPLY   EQU     4467H
00210 ;
00220 ;       Model III equates as comments
00230 ;       Change If using a Model III
00240 ;
00250 HIGH    EQU     4049H           ;4411H for Mod 3
00260 LOGOT   EQU     447BH           ;428Ah for Mod 3
00270 ;
00280         ORG     5200H
00290 ;
00300 ENTRY   LD      A,(DE)          ;Device must have output
00310         AND     2
00320         JR      Z,NOGOOD
```

```
00330          PUSH    DE              ;Save DCB
00340          LD      HL,MSG
00350          CALL    DSPLY           ;Show sign on msg
00360          POP     IX              ;IX => DCB of *CL
00370          LD      HL,(HIGH)       ;Now ready to move
00380          LD      BC,LAST-START   ; filter into high mem
00390          XOR     A
00400          SBC     HL,BC
00410          LD      (HIGH),HL       ; so move HIGH$ down
00420          INC     HL
00430          LD      A,(IX+1)        ;Old *CL entry pt.
00440          LD      (PUTBYT+1),A    ;Store in our filter
00450          LD      (GETBYT+1),A
00460          LD      A,(IX+2)        ;Also old MSB of *CL
00470          LD      (PUTBYT+2),A
00480          LD      (GETBYT+2),A
00490          DI
00500          LD      (IX+1),L        ;Put our new entry pt.
00510          LD      (IX+2),H        ; into *CL DCB
00520          EX      DE,HL
00530          LD      HL,START
00540          LDIR                    ;Move our filter up
00550          EI
00560          JP      EXIT            ;All Done
00570 ;
00580 NOGOOD   LD      HL,ERRMSG
00590          CALL    LOGOT
00600          JP      ABORT
00610 ;
00620 MSG      DEFB    LF
00630          DEFM    'This filter will add a '
00640          DEFM    'line feed to <CR>'
00650          DEFB    CR
00660 ERRMSG   DEFM    'This filter is for output only!'
00670          DEFB    CR
00680 ;
00690 ;        Actual filter code to move to high memory
00700 ;
00710 START    JR      C,GETBYT        ;If Carry, is input request
00720 PUTBYT   CALL    0               ;Old *CL driver
00730          CP      CR              ;Did we just send a CR?
00740          RET     NZ              ;Nope
00750          LD      C,LF            ; Else yes, so send LF
00760          JR      START
00770 GETBYT   JP      0               ;Old *CL driver for input
00780 LAST     EQU     $
00790          END     ENTRY
```

Using the EDAS 4.1 "Z" Command

By Earl C. Terwilliger

The new version of MISOSYS's editor assembler is now available, EDAS-IV. GREAT STUFF! The original version was used to assemble LDOS itself. Its new features and functions (too numerous to mention here) give the assembly language programmer ALL that can be asked. There are so many functions, it is sometimes hard to remember them all. Ah ha, you say! Does it have a HELP facility? Well, since you asked, you can build a help file and display it with the <V>iew facility. However, if there are multiple "screens" in your HELP file, scrolling to the right place may not be so easy!

EDAS-IV has a patch space available for the user to implement a function. I decided to use this patch area to implement a HELP display facility. In EDAS-IV, at address X'5809' is a vector pointing to X'5DAF'. At X'5DAF' is the 50 byte patch area. To start executing the code at X'5DAF' type in a Z and a carriage return. This is the ZCMD function. Originally as shipped, EDAS-IV has an X'C9' at address X'5DAF'. This is the Z80 opcode for a RETURN. I designed the following program to be PATCHed in starting at the X'5DAF' location and thus replace the RETURN. The program simply opens the file EDASHELP/TXT and lists it on the video screen. (The PATCH code created from the assembled program is also shown below.) If multiple HELP "screens" are desired, simply precede each "screen" of data with an X'0C'. When the HELP function detects an X'0C' as part of the data it will pause until any key is depressed. When a key is then depressed, the next screen of data is displayed! Before each "screen" is displayed, the video screen is cleared. This is done so that each "screen" does not have to fill the video display. To exit the HELP facility (or ZCMD function) without having to display all of the screens, depress the BREAK key. The source program and its PATCH version are as follows:

```
;       EDAS <Z> COMMAND PROCESSOR     (ZCMD/ASM)
;               HELP FACILITY
;       WRITTEN BY EARL C. TERWILLIGER JR.
;
;       ADDRESS X'5809' =--> X'5DAF' =  X'C9'
;       FOR THE TRS80 MODEL I
;       REPLACE WITH THE FOLLOWING
;
        ORG     05DAFH          ;PATCH AREA
START   EQU     $
        LD      HL,4200H        ;SBUFF$ SYSTEM BUFFER
        LD      DE,FCB          ;FILE NAME ADDRESS
        LD      B,0             ;LRECL = 0
        CALL    4424H           ;@OPEN - OPEN FILE
        RET     NZ              ;OPEN OK?
CLS     EQU     $
        CALL    01C9H           ;CLEAR THE SCREEN
GET     EQU     $
        LD      DE,FCB          ;FILE ADDRESS
        CALL    0013H           ;@GET - CALL GET A BYTE
        JR      NZ,CLOSE        ;CLOSE IF END OF FILE
        CP      0CH             ;CODE TO WAIT?
        JR      NZ,DSP          ;NO. DISPLAY
        CALL    0049H           ;@KEY - WAIT FOR INPUT
        CP      01H             ;BREAK?
        JR      Z,CLOSE         ;END IF YES.
        JR      CLS             ;GO CLEAR SCREEN
DSP     EQU     $
        CALL    033H            ;CALL @DSP
        JR      GET             ;GET
CLOSE   EQU     $
        LD      DE,FCB          ;FILE NAME ADDRESS
        CALL    4428H           ;@CLOSE - CLOSE FILE
        RET                     ;RETURN
LAST    EQU     $
LENGTH  EQU     LAST-START
        ORG     4480H           ;CFCB$ FCB FOR COMMANDS
FCB     EQU     $
        DEFM    'EDASHELP/TXT
        END
```

```
.
. PATCH FILE FOR EDAS/CMD  EDAS-IV
. IMPLEMENTS THE <Z> COMMAND FOR THE TRS80 MODEL I (LDOS)
. WHEN THE <Z> COMMAND IS ENTERED IT LISTS THE FILE
. EDASHELP/TXT ON THE VIDEO SCREEN.
.
X'5DAF'=21 00 42 11 80 44 06 00 CD 24 44 C0 CD C9 01
X'5DBE'=11 80 44 CD 13 00 20 12 FE 0C 20 09 CD 49 00
X'5DCD'=FE 01 28 07 18 E8 CD 33 00 18 E6 11 80 44 CD 28 44 C9
X'4480'="EDASHELP/TXT                     "
.        <-------- 32 BYTE FCB --------->
.
. END OF PATCH
```

The file name EDASHELP/TXT can, of course, be changed to one of your own choice. The special pause code of X'0C' in the data file stream can be changed to one of your own choosing also. Be sure and change the program code to match the file name and pause code you choose! MODEL III LDOS owners will want to change to the appropriate LDOS addresses used in the program.

A possible EDASHELP/TXT file, one that I use is as follows:

```
.................. EDAS-IV  HELP  MENU ....................

     <A>  ASSEMBLE            <B> BRANCH       <C> CHANGE/COPY
     -CI  CORE IMAGE          <D> DELETE       <E> EDIT
     -IN  IN MEMORY           <F> FIND         <G>
     -LP  LINE PRINTER        <H> HARD COPY    <I> INSERT
     -NC  NO CONDITIONAL      <J>              <K> KILL A FILE
     -NE  NO EXPANSION        <L> LOAD FILE    <M> MOVE BLOCK
     -NH  NO HEADER           <N> RENUMBER     <O>
     -NL  NO LISTING          <P> PRINT        <Q> QUERY DRIVE
     -NM  NO MACRO            <R> REPLACE      <S> SWITCH
     -NO  DUMMY               <T> TYPE SOURCE  <U> UTILIZATION
     -WE  WAIT ON ERROR       <V> VIEW FILE    <W> WRITE
     -WO  WRITE OBJECT        <X> EXTEND       <Z> HELP
     -WS  WITH SYMBOLS        1   ALTER LINES PER PAGE
     -XR  XREF FILE           .   JOB LOG MESSAGE
```

Well, I'll stop now and go back to creating some MACROS to store in my PDS library for use with EDAS-IV. Bye for now!

........................**er**........................

By Earle Robinson

I have been so involved with printers and drivers for the new SuperSCRIPSIT (tm) from Tandy, that much of this quarter's article will consist of some very negative comments about printers and their manufacturers. Working with some of these machines is some what like living in the New York City subway. The noise is deafening. At one time I had 5 printers scattered around my little office. Thank God I wasn't running more than one or two at a time.

First of all, I'll discuss the various daisy wheel printers that I have had the opportunity to work with while writing the drivers that my company, softERware, is distributing. The DW 2 from Tandy has been a real work-horse for many, and its merits are undeniable. Unfortunately, every time I have ever used one, my neighbors run over to learn if there has been an earthquake. In California that is no laughing matter! It is also a very ugly machine, resembling the modern designs that the World's Fair of 1939 projected for the future. The 'concentration camp' blue-grey doesn't help things much.

Nevertheless, it can provide very good print quality with a variety of different wheels. The real problem in today's environment is its high price compared with what some of the competition is offering. The new RS daisy wheel, the DWP 410, is much quieter, no prettier, and slower. It is also cheaper.

Next the F10, aka 'Starwriter' or 'Printmaster'. The F10, available in 40 cps and 55 cps models is a reasonably quiet and non-vibrating machine which offers the same print quality as the DW 2, but at a significantly lower cost. If you shop around, you can pick one up for about $1300-1350. Add $300 for the faster model. One weakness that it has compared to the DW 2 is that it supports only one proportional space wheel, the Theme 10 Pt. The spacing for that wheel is locked into its electronic circuitry. This means that when you are in proportional mode, the circuit provides the spacing as each letter is printed. If you use another proportional wheel, and it doesn't have the same spaces as the Theme, tough cooky! There is an option, according to the #&$% manual, but the technical people at the importer have never figured out how to use it! On the plus side, the F10 will accept any of the Qume or Diablo wheels. This gives you a very wide palette for whatever printing needs you may have.

Finally, the manual! If you think that Digital Research writes poor manuals or that government documents are obtuse, you must read, or try to read the F10 manual. It is the worst written piece of you know what that I have ever come across. Even the first documentation for my 'discatER' was better written and clearer. The dip switches are described in two different sections. There is no way to know which one is correct. This is especially important since the front end has no switches for varying pitch, line spacing. Only switches for select/deselect, line feed, form feed and a set-page. Further, the manual speaks of 'serial' and 'line' mode. You might think that your parallel printer won't work in the serial mode. Wrong. Serial here means a sort of logic-seeking mode which offers more features than the line mode. So, why even consider line mode? I don't know. I could go on forever, which is how long it took me to figure it all out.

Next the Qume 5 Series. My neighbors thought that there might have been an earthquake when the DW 2 was running. They knew that there had been one when the Qume was in action. This printer, which weighs about 14 tons, probably has vacuum tubes in place of circuit boards. It is a solid monster. It was the first daisy wheel distributed by Radio Shack, and under its own name has been vibrating and printing away for years. It also deserves mentioning because nearly all other printer manufacturers use most of the software control codes that it does. Note that I said nearly all. More later.

Then there are the various Diablos, the newer Qumes. All resemble in one way or another the F10, except they are more expensive. One exception is the NEC Spinwriter. This is the Rolls Royce of printers. As you probably know, NEC uses a thimble as the printing element. Since it is closer to the paper, it provides more precise printing.........at a price. If you are looking for a daisy wheel, I would go for the F10, if you are either a programmer or if there is a printer driver for the WP program with which you would use it. Reasons are its low cost, relatively little noise, and the wide availability of wheels and ribbon cartridges. I am told that there is a new manual in the offing, too. Regarding wheels, be prepared for problems in finding the proportional ones or those that are not the standard. Also shop around for prices. My local Computerland charges $35 for a Theme wheel while I can get it elsewhere for $6 or less.

Just a few words about the low cost Brothers and Smith Corona daisy wheels. Cheap they are, compared to the others. But, they lack many features of the better printers, most particularly the lack of a backspace. They are also very slow. My advice is to save up for a true daisy wheel or just save money and buy one of the dot matrix printers described below. I also saw a new Dutch manufactured printer at Comdex. Very nice. But what about service if they can't compete in the market?

It was the Epson MX printers that revolutionized the dot matrix printer market and provided for the first time good print quality at an affordable price. I have had one for over three years and it has never failed, nor have I even had to change the printing head. The documentation, written by David Lien is outstanding, an example that I wish other manufacturers would follow. The print quality in today's market is, however, no longer competitive for the same price, in spite of the qualities of the new ROM's. It is also quite slow when using the features such as emphasized printing to improve appearance. Nevertheless, the Epson provides a clean print out, and is notably reliable.

The ProWriter, aka NEC 8023a, aka TEC, is actually manufactured by TEC. Available at about the same price as the Epson, it is perhaps a half a generation in advance. It is faster, offers Greek letters, and other characters not on the Epson, and proportional spacing. It also has a 12 pitch font, a definite advantage over the Epson which only offers 10 pitch as standard. The Epson compressed is a little too small for my taste. The manual, from the same importers who brought us the F10 is better than that, but still poor. It comes with both pin-feed and friction feed. Pin feed is often called tractor feed. It is not the same thing. For heavy duty usage, pin feed is definitely inferior.

As for the new DMP 200, 400 and 500 printers from the Shack, well, they do offer the considerable technical support that Tandy provides. Enough said.

Now, the star of the new generation of dot matrix printers, the Toshiba P1350. I understand that RS will also be distributing it under the name of DMP 2100. It is fast, runs at 160 cps (that means characters per second) for draft printing and between 80 and 120 for the various other optional printing modes and fonts. The Toshiba offers 10 and 12 fixed pitch fonts and their so-called expanded versions of 5 and 6 pitch as well as proportional font. Frankly, I prefer the 'Prestige Elite' font. It is almost as good as daisy wheel printing. The manual is poor but adequate. And, at least they do use the Qume control codes which will simplify a programmer's task. The price is high, $2200 list; there is no discounting yet. Add another $250 for a tractor feed. If you can afford it and/or need it, wait a few months for competition to force the price and choose this gem of a printer. The new Anadex printer, not yet out, is also reported to be outstanding. For an economical printer, try the ProWriter in one or another of its garbs. There are many other dot-matrix printers out there, too. My advice is that you ask yourself if the manufacturer will still be in business a year after you buy one. If the answer is no, don't buy it. There are many people who bought the Base2, to name only one defunct printer, and they have no way to get it serviced.

I firmly believe that dot matrix, or a new technology will eventually replace the daisy wheel and spinwriters of today. If you need a printer for business correspondence or are submitting manuscripts to publishers, go for a daisy wheel or a spinwriter. There are people out there who look at letters with a magnifying glass to see how it is typed. If they see dot-matrix, into the round file it goes. If you are already rich like Tim Daneliuk then you can get away with anything you want, even a scratchy pen. Otherwise, take note of the above.

I have recently had the opportunity to try two programs, both of which are very impressive. The TBA package, from the same Mequonians who brought you LDOS and this quarterly, is a fine piece of software. It will permit you to structure your Basic programs and then read them afterwards, months afterwards. You can even create a sort of macro type of capability from within Basic with TBA. I think I'll do a little more work in Basic in the future.

The other program, UTILZAP, was written by Bob Bowker a television creator who likes to twiddle bits in his spare time. Utilzap is a 'superzap' type of program which offers two things that SU+ doesn't: double sided support and use with an LX80. SU+ has many more features, of course. But, I can't use it with my LX80 nor with my 8" drives. So, I have turned to Utilzap. I prefer FED for some of the things that the other will do. But, Utilzap has direct access to disk cylinders and tracks.

Here's a little quick, very quick way to copy sectors from one file to another from within a program. First of all open the files with the LRL as 256. Then load DE with the file to be read, and HL with the file to be written. The rest of the code is as follows:

```
AGAIN   CALL    @READ
        EX      DE,HL    ; Exchange FCB'S
        JR      NZ,OUT   ; EOF. Get out
        CALL    @WRITE   ; Or use @VER
        JR      AGAIN    ; Read next sector
OUT     CALL    @CLOSE   ; Close the new file
        RET
```

I use this little routine to make a backup of re-opened files from within SuperSCRIPSIT, an enhancement which should have been there to begin with.

By the time you all read this Christmas and Hanukah will have passed. I wish everyone (except the author of the F10 manual) a most happy and prosperous New Year!

PARITY - ODD

(C) 1982 Tim Daneliuk
T&R Communications Associates

FIRST, A WORD ABOUT TANDY...

There has been a noticeable upsurge of the "Tandy is a bad guy" mentality appearing lately in the popular microcomputing press. While I certainly have no reason to defend the folks in Fort Worth (I don't even own Tandy stock!), I think that a lot of the editorializing being done is poorly thought out, and sometimes just plain ridiculous. So, to start out this column, I've decided to do a little editorializing myself. I hasten to point out that this section is necessarily biased and represents only my own rather narrow-minded opinions!

ACCUSATION NUMBER 1: Tandy doesn't know how to market it's new computer products correctly.

This is partly right and partly wrong. It is true that some of their recent marketing decisions have made less than great sense (introducing the Model 16 with no supporting software). On the other hand, you can hardly fault the marketing practice of a company whose revenues went up about $300 million during a year when the rest of the economy withered away. They must be doing SOMETHING right! Tandy has made mistakes, but to paraphrase Mark Twain, "Reports of their demise are greatly exaggerated". Whether Apple, IBM, or Atari like it, Tandy is still a dollar volume leader in the microcomputer market. I wouldn't count them out just yet on the marketing front!

ACCUSATION NUMBER 2: The Radio Shack computer salespeople are poorly trained.

Score 100% on this one. I have seen some of the most amazing demonstrations of TRS-80s in computer stores ("Yessir, you just stick the disk in drive 1, and soon you'll be running accounts receivable for all 5000 of your stores...") It is unbelievable how little about their product line many RS sales people know. However, blame can't be laid primarily at Tandy's doorstep. This is an industry-wide problem ('Been in a COMPUTERLAND or BYTE Shop lately?? Its not really any different.). This problem was bound to happen so long as the growth rate of the product outstripped the nation's ability to assimilate new technologies. Ever try explaining what a "byte" of data or a "disk sector" is to the complete novice? It can be REAL frustrating. I think the only solution to this one is time.

Eventually, personal computers will become as integral to our lives as automobiles or television, and the learning gap will narrow. One final thought here: You can't expect a computer salesman to give you in-depth explanations of how his particular flavor of computer is designed, along with a tutorial explanation of a boot ROM disassembly! You don't go to your local GM dealer and demand to know why they use one spark plug gap versus another. Why? They didn't design the car. The dealer is there to sell cars and provide normal warranty and post-warranty service on them, not custom design them to your specifications. Likewise, you cannot reasonably expect your local dealer to provide hours of consultation to give you a free education in computer science.

ACCUSATION NUMBER 3: Tandy doesn't provide adequate after-sale support for their hardware/software products.

This is the most common gripe I've heard, both in the media, and by TRS-80 owners. I may be sticking my neck out on this one, but in my estimation, this complaint has LITTLE OR NO BASIS IN REALITY WHATSOEVER! Tandy has repeatedly provided hardware and software updates for nothing, yet people complain. Sure, I'd like them to admit that some of their software has problems, and fix it the day I report the problem. On the other hand, you get what you pay for. As an example, take the case of LDOS. I've heard LDOS users grumbling that they wished Tandy were as responsive as LSI in dealing with problems. This is silly. LSI can be responsive 'cause they charge you for every update. You can get extended support for a year, or just pay the ten bucks every time you update a disk. The point is, that our friends in Mequon have revenue which helps pay for their support activities. Tandy does it for nothing, and then gets a lot of grief from the user community because they don't provide instant service. Personally, I think that the Radio Shack Microcomputer News magazine should be expanded to encompass a nationwide support network, much like LSI's extended support agreement for LDOS. With each serious user kicking in $20 or so a year, sufficient revenue would probably be generated to set up an independent division whose purpose would be to provide timely support on all major software and hardware items. This would also weed out the casual users who think that their twenty or thirty bucks invested in a game gives them the right to tie up a $40,000 a year systems analyst with ridiculous questions (these folks would NEVER consider spending an extra nickel for support, let alone $20).

The really amazing thing about this whole issue, is that many of the biggest complainers are also the biggest software pirates around. They steal the software, and then have the gall to be outraged when their software and/or hardware has problems. In a capitalistic economy like ours, the "bottom line" makes the corporate boardroom decisions. A company that is losing revenue due to lost sales from pirates, is not likely to go out of their way to improve customer support! Whether you believe it or not, piracy ultimately hurts YOU, the computer user.

As Bill and many of the people at LSI will attest, when you give support away, people loose sight of the value they are getting for their money. When LDOS was still being sold with free 800 line support, the overwhelming majority of calls were from people who couldn't be bothered to read the manual. I think this kind of thing is the reason Tandy recently announced the abolition of it's 800 lines. If you get the idea that I'm really in Tandy's corner on this issue, you're absolutely right. I used to work in a service industry, and I know from long hard experience that as long as you give your time and effort away, it is NEVER appreciated. Tandy has made it clear that they realize their problems and deficiencies, and that they really want to support the user community. The fact that LDOS is being sold in Radio Shack stores, the fact that Tandy has an official policy of attempting to repair TRS-80s which have been modified, the fact that Tandy is setting up a division which encourages third party software, all point to an active corporate policy of trying to provide quality products and keeping the customer happy.

If you think this is a lot of opinionated nonsense, here's a little survey you can conduct. Find yourself a half dozen or so computer users with machines other than the TRS-80 (if there are that many!). Try to pick people with abilities comparable to your own. Now, compare notes on how each person's respective hardware/software vendor treated them. You'll probably be surprised. Tandy may just turn out to be the best of the bunch! Oh, one more thing: Compare notes on what each person's total investment has been in real dollars (i.e. actual costs incurred to buy the system and keep it running) versus what they've been able to accomplish with their systems. Be prepared for suprise #2.

NOW, A FEW ODDS -N- ENDS

The first PARITY=ODD poll on disk drives is going very well. You're response has been gratifying, to say the least (I didn't know that many people read this column!). If you haven't sent your entry in yet, please do it NOW. The information concerning the poll as well as where to send your entry is in the last issue of the LDOS Quarterly.

As some of you may know, I served as a field test site for the new computer from LOBO, the MAX-80. I hope to have a chance to comment in depth on it in some future installation of this column. For now though, let me say that this is a fine machine well worth the price. You'll have to go a long way to find a more cost effective microcomputer. If you haven't heard, the MAX-80 comes standard with 5", 8", and hard disk interfaces, two RS232 ports, a parallel printer port, and a TRS-80 type expansion bus. It also has a lot of little "goodies" like a real time clock with battery backup, 64K memory, a programmable video character generator, etc., etc. Best of all, the MAX runs BOTH CP/M and LDOS (a soon-to-be-released version of 5.1.3), yet costs a paltry $820! If you're thinking about adding a new machine to your collection, or would like to upgrade to something new, give the MAX-80 a close look. It promises to be a popular and well supported machine. Speaking of LOBO, they've just reduced the price of all their hard disk products, making them substantially more competitive with other hard disk packages than they used to be. I've been running the large 8 MEG version (the Model 1850) here, and I've found it to be a tremendous piece of hardware. You can't fully appreciate the elegance and power of LDOS till you've seen it running a hard disk system! Here's LOBO's address:

LOBO Drives International
358 5. Fairview
Goleta, CA 93117

(800) 235-1245

The final ODD -N- END, concerns Model I owners who are using their machines in critical applications. Since Tandy is no longer manufacturing our dear beloved "I", you may want to consider buying a used one as a backup in case of dire emergency (i.e. the CPU blows-up). AEROCOMP has, from time-to-time, a few reconditioned Model I, 16K, Level II, keyboard units. They also have a good selection of Model I parts like printed circuit boards, cases, keyboards, and so on. If you're a hardware hacker or just need a spare machine, you might want to check with them:

AEROCOMP
Redbird Airport
Building 8
P.O. Box 24829
Dallas, TX 75224

(800) 824-7888

FINALLY, THE GOOD STUFF!

I've looked at two software products for this issue of the Quarterly, the first of which is a spelling checker called Electric Webster. I was particularly anxious to look at this product, because it's predecessor, Microproof, was notoriously incompatible with LDOS. Well, things have changed. Webster runs just fine under LDOS 5.1.3 (I didn't check it under earlier versions). The only feature which I didn't actually test, was calling it from within the LSCRIPT environment. One word of caution: Be sure you have the latest version of Webster and ALL the patches from the manufacturer before trying to run it under LDOS and LSCRIPT. I understand that there were a few problems with some of the early releases, so check with Cornucopia if your copy is fairly old.

Webster is written entirely in machine language. Once it has loaded, you are prompted for the name of the file you want to check. That file is loaded, and Webster proceeds to check for spelling errors. The program will display the total number of words in your file as well as the number of unique words. Then all the unrecognized words are listed to the screen. Now, Webster takes you through that list, one word at a time. You can interactively add the word to your dictionary, ask to see it in context, or correct it's spelling. There is one real handy feature for those of us who can almost spell. At any time, you can examine the part of the dictionary the unknown word would go in if it were added to the list of known words. This helps you to find the correct spelling of a word even if you can't remember EXACTLY how to spell it. For example, if you use the word "evident", Webster will flag it as unknown. When you ask to see the dictionary, a list of 15 words or so will be displayed, all of which start with "ev...". By going through the list, you can find the correct spelling of "evident", and correct your text accordingly. After all the text has been processed, Webster will load a previously patched version of LSCRIPT, bring in the corrected text, and turn control over to the word processor. You can now print, save, or edit the corrected text as usual.

Electric Webster runs very nicely under LDOS. It seems well integrated into the operating system, and I had no difficulty using it on a variety of hardware including 8" drives and a hard disk. It is also one of the fastest (if not THE fastest) of the spelling checkers I've tested so far. I found it easy to use, and the documentation for it quite useful.

There are a few problems with Webster, however. First, you cannot use lower-case to input the name of the file you want to check. This is not particularly serious, but irritating nonetheless. Secondly, the size of file you can check is limited to what can fit in memory. With disk oriented word processors like SuperScripsit becoming popular, this is a fairly severe limitation. Thirdly, if you correct a word, Webster doesn't check to see if your correction is properly spelled. So, if you are correcting "evident" and change it to "evident", Webster will let it go by unnoticed. This is a big problem, in my judgement, and one that ought to be fixed pronto!

All in all, I really like Electric Webster. Once you get used to it's idiosyncracies, it is a powerful spelling checker that will be useful to almost anyone. At $149 it's no giveaway, but if you write a lot, the price is easily justified. I only hope that Cornucopia comes out with a disk oriented version to cope with word processors like SuperScripsit. Electric Webster is available from:

Cornucopia Software Inc.
P.O. Box 5028
Walnut Creek, CA 94596

(415) 524-8098

Before I go on to the second product, The BASIC Answer from LSI, I should mention one thing. I generally try to include one LSI/Galactic/Misosys product in each of these columns. This does NOT mean that these products get an automatic positive review.

The "powers-that-be" at LSI have given me complete latitude in doing reviews, including the right to negatively review their products, so long as I can substantiate my views (they must really think their stuff is good!). Rest assured that I look at the LDOS support products from these companies with the same rigor as I do anyone else's software.

TBA (The BASIC Answer) is a BASIC text processor whose function is to make programming in BASIC simpler by allowing you to write line-independent, structured code. You do this by writing your programs in a syntax that refers to other parts of the program by label. For example, instead of using GOSUB 1000, you might use GOSUB @SCREEN.DUMP. Similarly, variables can have much longer names than in the usual LBASIC language (up to 14 significant characters). TBA also lets you maintain variables as being "Local" in a subroutine, or "Global" and accessible to the whole program. The idea behind all this is that programs written this way will be 1) More readable than regular LBASIC code, and 2) Easier to maintain in the future. For example, let's say you use an important subroutine that normally starts at 1000. If you ever modify it so that it starts at some other line number, you will have to change all GOSUB 1000 references in the entire program. With TBA you don't have to change anything because all your references to that subroutine were something like GOSUB @SCREEN.DUMP, which is completely independent of the actual line numbers.

Once you've written a program using TBA's "language" (using a text editor like LED or LSCRIPT, or even the LBASIC editor), you use TBA to translate the program into normal LBASIC syntax. In fact, so long as you don't use the LBASIC extensions to standard Tandy Disk BASIC, you can use BASCOM or other similar compiler products to compile TBA-generated programs. TBA also has many options built-in including complete cross-referencing facilities, and assembly language type pseudo-ops like TITLE, LIST ON/OFF, and PAGE.

As usual, LSI has included complete documentation which has sections for both the beginner and advanced BASIC programmer. The package sells for $69 and will run on either the Model I or III.

I have to be honest: I really don't care much for BASIC. In fact, except for Snapp's extended BASIC for LDOS, I'd never seen any BASIC utility that I thought was worth much. Well, we live and learn. TBA is terrific! Its just plain indispensable if BASIC is your language of choice. TBA almost forces you to write self-documenting, structured programs. You really have to go out of your way to write sloppy code with this product, and I recommend it wholeheartedly. In fact, just between you and me, I've started programming in BASIC again. In fact, I may even publish some TBA "source" in this very column. But please, don't tell anyone...

ITEMS OF GENERAL INTEREST

The patch to enhanced Visicalc version 150Y0-T83 in the October Quarterly suffered a character loss during editing. In the line that starts X'A720', there is a single "3" as one of the character pairs. If that pair is changed to a "3A", the patch will work properly. Thanks to Revd. Michael Bootes of West Sussex, England for bringing this to our attention.

The following patch was part of the article "LSCRIPT patches add versatility" in the October 1982 issue of the Quarterly. Unfortunately, there was a problem with the patch 4 section, and we did not get the corrected version into the newsletter. The correct code is as follows:

```
.
.Patch 4 - resolve conflicts between Scripsit & PR/FLT
.  The following patch allows Scripsit to peacefully coexist
.  with PR/FLT. The patch will zero the PR/FLT parameters on
.  entry to Scripsit and restore them on exit. This will only
.  work for LDOS 5.1.2 and later. This patch was written by
.  Scott Loomer.
D00,48=C3 F5 63
D12,3D=E5 3A 25 01 FE 49 28 07 21 1F 44
D12,4C=3E 42 18 05 21 89 42 3E 43 32 33 64 22 A1 62 7E
D12,5C=CB 5F 28 23 DD 2A F6 4D DD 7E 19 32 3B 64 DD 36
D12,6C=19 00 DD 7E 1A 32 3C 64 DD 36 1A 00 3A 2A 40 32
D12,7C=3D 64 3E 00 32 2A 40 E1 C3 69 63 00 00 00
D13,E4=C3 A0 62
DL0,E4=3A 00 00 CB 5F 28 16 DD 2A F6 4D 3A 3B 64 DD 77
DL0,F4=19 3A 3C 64 DD 77 1A 3A 3D 64 32 2A 40 C3 DD 63
.End of Patch
```

The LDOS Library commands LOAD (X) and RUN (X) are documented as working with files that start above X'5300'. This is not correct, and should be documented as working with files that load at or above X'5400'.

Following is a patch to FED/CMD, the LDOS File Editor. It fixes a problem that can occur with the L command.

```
     . Patch FED/CMD, corrects L command
     D1B,81=0B 20 01 04
     . EOP
```

Early versions of the WRTEST/CMD program on the Utility Disk #1 has a problem when running on the Model III. The following patch will correct it.

```
     PATCH WRTEST/CMD (D01,6A=9A 52)
```

The MAP/CMD program on the first utility disk would not always show the proper sector count for double sided drives. Apply the following patch to correct this problem.

```
     . Patch for MAP/CMO, utility disk #1
     D03,D5=E4
     D03,FB=C6 00 B8 DA EB 55
     D04,2E=E4
     D04,41=E4
     D04,54=E4
     D05,79="4"
     . EOP
```

Model III PROFILE, Ver 3.4, will sometimes come up with a "Disk full or write protected" error when initializing a new system. The following patch to the INIT program will cure the problem:

```
     PATCH INIT/ (X'7062'=1A)
```

LCOMM on version 5.1.3, on Model I and III and MAX-80, has a problem when trying to send a disk file that is larger than the memory buffer. To fix it, apply the following patch:

```
. MODEL I patch for LCOMM file send - 12/15/82
D07,AE=12
. EOP

. MODEL III and MAX-80 patch for LCOMM file send - 12/15/82
D07,B3=12
. EOP
```

For Version 5.1.3, both Models I and III, the following patch to FORMAT/CMD will correct the use of precompensation when formatting in double density.

```
. Model I & III TRS-80 patch for FORMAT w/precomp
X'60E5'=8B 67
X'678B'=FD 56 05 C3 08 64
. EOP
```

The following patch for LBASIC prevents a syntax error if a space is used after a hex constant.

```
. LBASIC/CMD fix for use with hex constants, MODEL I only
D08,6E=D7 00
. EOP

. LBASIC/CMD fix for use with hex constants, MODEL III, MAX-80
D08,7F=D7 00
. EOP
```

The PR/FLT program uses a CHR$(6) in a special manner to reset its internal line counter. This can interfere with programs that do dot addressable graphics or proportional printing. The following patches will prevent a CHR$(6) from being trapped by PR/FLT.

```
. MAX-80 PR/FLT patch, disable CHR$(6) feature
D03,DF=00 00 00 00
. EOP

. Model III PR/FLT patch, disable CHR$(6) feature
D03,C0=00 00 00 00
. EOP

. Model I, 5.1.3 PR/FLT patch, disable CHR$(6) feature
D03,E7=00 00 00 00
. EOP

. Model I, 5.0.3A PR/FLT patch, disable CHR$(6) feature
D03,2F=00 00 00 00
. EOP
```

The XLATE parameter in LED does not function correctly. This can be fixed by applying the following patch:

```
. Patch to LED, all versions, to correct XLATE parm
D03,41=BA
. EOP
```

When using the Extended Visicalc and the LDOS KI/DVR program, the key sequence <SHIFT><CLEAR> rather than just <CLEAR> is needed to perform a backspace. Also, the ^ character used to denote exponetiation can be generated by <CLEAR><;>.

MAX-80 LDOS users may run into a problem at times when attempting make a backup using a single drive. To correct the problem, apply the following patch to the FORMAT/CMD program:

```
. FORMATXA - Fix FORMAT/CMD for single drive backups - MAX-80 ONLY!
D10,79=00
D11,7C=00
. EOP
```

When using both the MINIDOS and KSM filters, the MINIDOS filter normally must be installed AFTER the KSM. If the order is reversed, the "R" function of MINIDOS will not work. From time to time we get requests from users who want to sysgen Minidos and use KSM files only for certain applications. Right now the only way to do this is to sacrifice the "R" function of Minidos.

As a solution, Roy has developed the following patch to MINIDOS/FLT. Once this patch is applied, the Minidos function "R" will only work if KSM is installed after Minidos, not before it. It is recommended that you rename the copy of Minidos after you patch it to avoid confusion.

```
. Patch to MINIDOS/FLT Ver 5.1.3 Model I
D02,92=14
D02,DD=2l 12 00 ED 7A F9 E1 C9 00 00
. EOP

. Patch to MINIDOS/FLT Ver 5.1.3 Model III
D02,8E=14
D02,D9=2l 12 00 ED 7A F9 E1 C9 00 00
. EOP
```

FIX Disk update

Since the last newsletter, the following files have been added to the FIX disk. The current date on the disk is 12/14/82.

***** RCOBOLA/FIX - Patch for Radio Shack RUNCOBOL for use
with ISAM files.

***** RSCOBOLA/FIX - Patch for RSCOBOL to prevent an error if
the compiler is entered without using a filename.

***** RCOBOLB/FIX - Patch for RUNCOBOL to fix a problem with
OPEN-EXTEND mode using non-ISAM files.

```
. RCOBOLA/FIX
. This fixes a problem with Opening and immediately
. closing an ISAM file. Patch the RUNCOBOL/CMD module.
.
X'5220'=CD 42 44 C8 FE 1C C0 78 B1 C8 3E 1C B7 C9
X'AE09'=20 52
. EOP
. **RSCOBOLA/FIX**
. Patch for RSCOBOL/CMD to prevent an error when no
. filename is used when entering the compiler
.
X'9B02'=CD 04 52
X'5204'=16 00 78 B7 CA 2D 40 C3 51 9C
. EOP

. **RCOBOLB/FIX - 12/13/82**
. Fixes Open/Extend problem in RUNCOBOL/CMD
.
X'522F'=CD 48 44 C8 FE 1C C0 AF C9
X'9A40'=2F 52
. EOP
```

LDOS for the MAX-80

For those of you who are not familiar with the MAX-80 computer from Lobo Drives, it is a Z-80 based 64K RAM single unit computer that has built in 5" and 8" drive controllers, parallel printer port, dual RS232 ports, runs at 5 MHz, and can be configured with an additional 64K of RAM. For more information on the computer itself, see the add in the last Quarterly or contact Lobo directly at 805-683-1576.

We are receiving many calls and letters asking questions about the MAX-80 and LDOS. Here are the most common ones along with our usual answers.

**1) What version of LDOS will be used for the MAX-80?

LDOS Version 5.1.3, emulating a Model III. This means that assembly language entry points and storage areas will be the same as the Model III version of LDOS. These addresses are shown in the "Alphabetic Model I/III Memory Map" near the end of this newsletter.

**2) Will my programs have more memory to run in since there is no ROM?

No. Since the LDOS is version 5.1.3, it needs the ROM code in low memory from X'0000' to X'2FFF', the same as a Model I or III. Lobo has a license to use the Microsoft code, so that part of memory is loaded from disk when booting. The memory available will be the same as when running a Model III.

**3) I own a Model I. Will my present software run on the MAX-80?

That depends... If the software would run on a Model III, it will probably run on the MAX. We also have many patches providing Model I/III compatibility available on our FIX disk.

**4) Will my Model III software run on the MAX-80?

Probably. Since the hardware is not a true image of a Model III and since the ROM area is not an exact duplicate, there could be conflicts. However, we did use the MAX-80 to write the MAX-80 operating system. We know that programs such as Scripsit, EDAS, the Microsoft M80, L80, and EDIT all run with no changes.

**5) Will LDOS use the optional 64K of memory?

No, not for any standard system features. We will be developing several utilities for the MAX-80, the first of which will be a RAMDISK for the alternate memory. This should be available in February if all goes as planned.

**6) What specific differences are there between the MAX-80 LDOS and the Model III version?

First of all, the entire front end of the ROM code was re-written. However, the Documented entry points were all maintained. Programs that use ROM calls should work as long as the call was documented in the Radio Shack Model III manual. The character set on the MAX-80 does not have the special character set that is normally found at locations 192 through 255. Instead, a graphics character is displayed. Since the character set is programmable, this graphics character can be changed to the special character if necessary (this function is not provided by LDOS, and would require the user to program the change). The MAX-80 version of LDOS has two utility programs to set the hardware clock/calendar. No power-up prompting for date or time will be done as the hardware with its battery backup always supplies the correct date and time. Also, programs such as BACKUP and FORMAT no longer display the message about the real time clock not being accurate, as the hardware clock runs constantly and is a REAL time clock. The entire keyboard driver is resident in low memory, so setting KI/DVR no longer takes any user memory. Two replacement programs, RS232M and MAX80/DCT take the place of their Model III counterparts to make up for the hardware differences.

**7) The MAX-80 runs at 5 MHz which is 2 1/2 times faster than the Model III. Will this make a difference in my programs?

Yes and no. Programs will definitely execute faster on the MAX-80! However, things such as the interrupt rate, the keyboard delay and repeat, and the documented call for delay (@PAUSE) have been adjusted to keep timing and delay loops the same as on a Model III.

**8) Will LDOS use an 80 column display?

No. LDOS version 5.1.3 is designed to use a 16x64 video display, and therefore does not have an 80 column mode..

**9) What about the documentation and support for the MAX-80?

LDOS for the MAX-80 comes with the same 5.1.3 manual as Model I and III LDOS, but has an addendum describing the differences in existing commands and new programs. A normal registration card and Extended Support Agreement are provided with the manual. Questions about the MAX-80 hardware should be directed to Lobo. LDOS questions can be answered by either Lobo or by Logical Systems. There is also a MAX-80 users group, MAXIMUL, as described in the Users Group article near the front of this newsletter.

**10) What type of programs will definitely not work?

Self-booting disks will not boot on the MAX-80. Utilities that access the Model III disk controller or printer port directly will not work. Communications programs that have their own RS-232 drivers included will not work because the MAX-80 uses the SIO chip from Zilog rather than the UART that is used in the Model III. Programs that used the ROM area of the Model III as a "bit bucket" or a garbage dump area will probably crash the system eventually. Programs that use tape I/O of any kind. Other than that, any program that uses only documented ROM or system calls should be OK.

**10) Do you sell the MAX-80?

No. To keep the price low, the MAX-80 is available only from the manufacturer - Lobo Drives International.

LDOS: How it works - The PATCH Utility

From time to time we get requests for in-depth explanations of certain LDOS library commands and utilities. As an experiment, the staff of the Quarterly has decided to start up this column. If you have a specific request for a "How it works" article, please send it in writing to LSI, attention Quarterly Editor. This issue's column is about PATCH, as requested by Mr. Ian Hawke of England.

These first few paragraphs will describe the different type of patches and when they should be used. Then, creating a patch and applying it to a file will be shown with several examples. Before continuing, one very important point should be made. NEVER patch the only copy of a program or data file! If for some reason the patch does not apply properly, you can always start over and try again... as long as you have another copy of the original file!

The LDOS Patch utility is used to make changes to existing programs or data files. To use it, you must tell Patch the name of the file you want to change and then give it the new information to put into the file. For this discussion, we will consider files to belong to one of two groups, the first being a load module program and the second being any other type of program or data file. Patch can be used with one of two formats depending on the type of file being patched - a patch by load address or a patch to a direct sector and offset in the file. Load module programs can be patched in either format. Any other type of program or data file must be patched using the sector and offset format.

Now that you know the two types of programs and the two types of patches, lets define some of the terms used in the description.

Load module programs are programs that load into a specific area of memory. They are usually identified by a /CMD extension, but this is not always true. Generally speaking, any program that you execute by typing in its name at the LDOS Ready prompt is a load module type program. With LDOS, the /FLT and /DVR programs are also load module files, and can be patched using either format.

BASIC programs, source code files for assemblers and compilers, word processor text files and program data files are all examples of non-load module files. This type of file is not normally changed with the Patch utility, but it can be done by using the direct sector and offset mode of Patch. However, it is generally easier to make corrections to most of these types of files by using the editor that originally created them.

Patch by load address is referred to as an "X format" type of patch. Patch by direct sector and offset is called a "D format" patch.

When using the X format patch on a load module file, the new code will be applied to the end of the file, making the file larger. The actual new code will be in the format:

```
  X'nnnn'=bb bb bb bb
```

The "X'nnnn'" represents the memory location where you wish the new code to start loading, while the "bb bb bb" represent the hexadecimal values to load at those locations. Since the new code is at the end of the file it will load last, thereby overlaying anything that might have been loaded previously at the specified memory location. Since files such as BASIC programs and data files do not use load addresses, you can see why this type of patch is reserved for use with load module files.

The D format patch is used to change one or more bytes starting at a specific sector, and at a specific offset into that sector. This type of patch would be in the format:

```
     Dss,oo=bb bb bb
```

where "ss" is the sector number, "oo" is the offset into the sector, and "bb bb bb" represent the hexadecimal values to be written to those locations. Since this type of patch is directly overlaying existing locations in the file, it will not make the file larger. Because it does not reference any particular load address as an X format patch does, it can be applied to any type of file. NOTE: ss and oo are offset from zero. Thus, D00,00 refers to the first record, and the first byte in that record.

Creating a Patch

In this issue of the Quarterly, the "Items Of General Interest" section (hereafter called IOGI to conserve space) contains patches to several programs, including one for the PR/FLT program. Referring to the IOGI section you should see that there are three separate patches, one for each LDOS version. All versions of the patch contain 3 lines - an identifying comment, the actual patch code, and an ending comment. You will notice that the patch is in the D format, with the Model 3 patch code line looking like:

```
   D03,C0=00 00 00 00
```

Examining this code, you should see that the patch will be applied to relative record 3 of the file (D03), starting at a relative offset of X'C0' in the record (C0=), and that four bytes of the file will be changed to zeros (00 00 00 00).

The two lines that start with periods are comment lines that are used to identify the name of the file and the end of the patch code. They will have no effect on the patch file, and may be omitted entirely if desired. Their only purpose is to let the user see what the patch file is for, and to assure that no lines were left out when the file was built.

Since this is a very short patch, there are two ways it can be applied to PR/FLT. The first method is to build the patch as an ASCII file using the BUILD library command or a word processor. Since you all have BUILD, I will use it in the following example. To follow along, get a backup copy of your LDOS disk with PR/FLT on it.

To create the patch file, type in the following lines from the LDOS Ready prompt, pressing <ENTER> and <BREAK> as indicated:

```
  BUILD PR/FIX<ENTER>
```

Your disk drive(s) will now access, and the system will open a file called PR/FIX. When the disk drives stop, you can type in the proper 3 lines of code as listed in the IOGI section, pressing <ENTER> after each line. When you have typed in the lines, press <BREAK> to close the file. Very simple so far! To actually apply the patch, enter the following line from the LDOS Ready prompt:

```
  PATCH PR/FLT.GSLTD USING PR/FIX
```

If all goes well, the disks will access, the Patch sign on message will appear, and shortly thereafter the message 'Patch(es) successfully installed" will be shown. If, however, trouble ensues, read this next section for the most common error messages and their causes and cures. After typing in the command line, the disks spin and the message "Program not found" appears WITHOUT the Patch sign on message showing up. This indicates the PATCH/CMD utility is not on your disk. Get a copy of it off your master and try again. After typing in the command the disk spins and "File not in directory" comes on the screen. This means you are missing either the object (PR/FLT in this case) or the patch file (PR/FIX). If you see the message "Bad hex digit encountered", chances are you mistyped the patch code. The line that Patch thinks is bad will be displayed on the screen, so you can check it against the written version. If it all looks correct, you may have left a space after the last digit pair in the line.

This may not be readily apparent from the display on the screen. The error message "Patch input format error" indicates that some patch code line was typed in incorrectly. The best way to correct this error is to list the /FIX file and compare it to the original printed listing. If you get an error message "Load file format error", you are trying to patch a non-load module file with an X format patch.

Since the above patch code is very short, it may be applied directly to the file without the need to create the separate PR/FIX file. To do this, you could use the Patch command as follows:

```
  PATCH PR/FLT.GSLTD (D03,C0=00 00 00 00)
```

If the patch code is small enough, it may be applied directly to the target file by placing it in parentheses on the command line. Files that contain more than one line or a very long line cannot be applied in this manner and will have to be built into a file as previously described.

The same rules apply for X format patches as well as D format ones. The patch code may be applied from the command line or can be built into a file. Again referring to the IOGI section, you will see an example of a command line X format patch that is for the Profile INIT program, a D format patch for the LDOS utility LCOMM, and several multi-line X format patches for programs in the Radio Shack COBOL compiler package.

The LDOS FIX disk is a collection of /FIX files for some of the more popular TRS-80 application programs. If you have recently purchased it and are unsure about how to patch programs with it, you can use the following format:

```
  PATCH object file USING fix file
```

The "object file" in this example is the name of the actual file you want to apply the patch to, such as RUNCOBOL/CMD, SCRIPSIT/LC, etc. The "fix file" is the name of the patch file on the FIX disk, such as RCOBOLA/FIX, LSCRIPT/FIX, etc. Once you have gone through the procedure a couple of times, you should find that applying patches is a very simple procedure.

One final note: Although the directory of a disk is accessible as a file called DIR/SYS, it should NEVER be patched! Doing so will make certain parts of the directory unreadable to LDOS. If you need to make changes to the directory, use the extended debugger disk read/write function or a program such as the FED file editor program.

THE JCL CORNER - By Chuck

Hello once again from the DO area of LDOS-land. No, I am no longer questioning if anyone reads this section of the newsletter. The reader response cards from the last newsletter have proved that many people are interested in using JCL and read this column and the other examples of JCL usage submitted by LDOS owners. HOWEVER . . . the #1 comment concerning JCL was something like "Have more simple JCL articles" or "I still don't understand how to use JCL". Assuming that the responses mean "What the heck is JCL good for and how do I use it", I will in the future try to dedicate a portion of this column to answering specific questions received from readers about the use of JCL as well as describing the general features. To start it off, let's go back to the basics for a second and define what exactly JCL is in its simplest form.

JCL : Job Control Language - Sometimes described as "A FILE CONTAINING A SERIES OF OFTEN REPEATED COMMANDSU.

JCL : Job Control Language - Sometimes described as "AN AUTOMATIC START-UP FUNCTION FOR APPLICATION PURPOSES".

JCL : Job Control Language - Sometimes described as "A PRE-DEFINED FILE OF KEYBOARD ENTRIES THAT ANSWER PROMPTS IN AN APPLICATION PROGRAM".

Before I talk about the above definitions, I want to stress that JCL shouldn't be considered a separate "language". Instead, it should be viewed as a method to create a series of commands that will be passed to the system JUST AS THOUGH YOU TYPED THEM IN ON THE KEYBOARD!

How can you figure out what a JCL file is good for? One way is to keep track of anything you do every day or more than once a day. These will usually be the first things to be incorporated into JCL files. Once you've found a function to automate, you're set to create a JCL file. No, creating a JCL file is not impossibly difficult, and in fact is extremely simple. Until you are experienced at it, I recommend that you use a pen(cil) and paper to do the following.

> Actually perform the operation, writing down exactly what you type in on the keyboard.

Not hard so far, eh? Well, the rest is just as easy. To actually construct the JCL file, you need to put what you have on paper into an ASCII file on your disk. Several methods are available. The BUILD library command is available to all LDOS owners and can be used as follows. From the LDOS Ready prompt, type in a command such as:

```
BUILD filename
```

where "filename" is the name of the JCL file you wish to create. Now, looking at the commands you have written down on the paper, type in each line and press <ENTER>. After all the lines are typed in, press <BREAK> to end the build. All done.

Word processors can also be used to create JCL files. Since my experience is limited to Scripsit (in one of its many LDOS patched versions), I'll use it as an example of creating a JCL file. This should be sufficient so that users of Newscript, Pencil, Lazy Writer, etc. can figure out the proper procedure. For this example, I am going to create a JCL file to copy three files from drive 0 to drive 1 and then do a directory command to see the results. First, I type in LSCRIPT to enter the LDOS patched version of Scripsit. Next I type in the actual commands, pressing <ENTER> after each line:

```
COPY JCL21/SCR:0 :1
COPY SUGGEST/SCR:0 :1
COPY CONTEST/SCR:0 :1
DIR :1
```

Now I have the actual commands I want the JCL file to execute. The next step is to save them as an ASCII disk file which I will call COLUMN/JCL. To do this, I go to LSCRIPT's command mode and type:

```
S,A COLUMN/JCL
```

> ** With LSCRIPT and possibly with other word processors, un-needed spaces may exist at the end of the file. Before saving the file, you should position to just after the last carriage return in the file and then issue the appropriate command to Delete Line. With LSCRIPT, this is a <CLEAR><3><CLEAR><2> sequence.

Text editors can also be used to create JCL files. If I were using LED (the LDOS Editor) to create this file, I would type LED COLUMN/JCL at the LDOS Ready prompt. Once in LED, I would type in the four command lines, delete any extraneous spaces after the last line, and then use the command <CLEAR><SHIFT><=> to save the file and exit.

Heeeeey, this stuff is SIMPLE!

Yup, that's all there is to it. To make this JCL file execute, I would type in the command DO =COLUMN at the LDOS Ready prompt, and then sit back and watch as the three files were copied and the directory of drive 1 was displayed. The nice thing about JCL files is that once correctly built, they never make typographical mistakes when issuing commands. The longer and more involved the command series is, the more you will appreciate the "error free" feature of using a JCL file to do a series of commands.

Use the LCOMM utility? Create a JCL file consisting of SET *CL RS232/DVR (parameters if needed), LCOMM *CL, //STOP. Use LBASIC? Create a JCL file consisting of LBASIC (parameters if needed), RUN"program name", //STOP.

OH OH . . . he's using some funny JCL stuff (//STOP) . . .

Not to fear, really. I can explain it in one small paragraph, plus a couple of examples. There are two common ways to end a JCL file. The first is with a //EXIT macro, the second is with a //STOP macro ("macro" being a JCL buzzword). If neither one of the macros is used, //EXIT is assumed. The big difference is that:

//EXIT puts you back where you came from.

//STOP leaves you where you are.

Normally, any JCL file such as the COLUMN/JCL file example that starts at the LDOS Ready level and also ends there will not use either macro at the end, and thus will be treated the same as if an //EXIT had been used - it starts at LDOS Ready and ends up there. However, if you are entering an application such as LBASIC, you will want to use the //STOP to remain in LBASIC when the end of the JCL file is reached. If you didn't use a //STOP, you would see a "Job Done" message and return to the LDOS Ready level as soon as the last line in the JCL file was executed.

Ask me, I'll tell you - responses to USER QUESTIONS:

"When I use the command DO FILENAME, I end with an extra file on my disk - a file called SYSTEM/JCL. What gives ?"

> When DOing a JCL file there are two commands - DO FILENAME and DO =FILENAME. The LDOS JCL contains provisions to do logic and substitution, and therefore has a "compile" phase. If the "=" is not used on the command line, the JCL file will first be compiled and then executed. The compilation phase examines each line and writes the results to a file called SYSTEM/JCL which it then executes. If your JCL file contains only executable commands and execution macros such as //STOP or //EXIT, using the "=" in the command line will prevent the SYSTEM/JCL file from being created. A file such as the COLUMN/JCL example described earlier is such a file and does not need compiling.

"When I use a BASIC compiler, the resulting program executes much faster than the original program. Why isn't this true with JCL, and how can I speed up the execution. Can the MEMDISK utility program help?"

> Unlike language compilers such as a BASIC compiler, the JCL compile phase does not create a machine language program. Instead, it is used to check for logical operators and user defined substitution fields, etc. The result is still an ASCII file used to substitute lines in the file for keyboard responses. To speed up execution, you can use (version 5.1) the SYSTEM (SYSRES=11) command to reside the JCL execution module in memory. For those of you who have the MEMDISK utility, creating a MEMDISK and then copying the JCL files to it will speed up execution. To provide optimum speed, JCL files that are used extensively can be copied to a small MEMDISK and then SYSGENed.

** This is a paraphrase of letters/telephone calls:

"I don't understand JCL, and I don't want to call or write with my questions because I'm afraid you'll laugh/talk over my head/tell me to read my manual/etc." "I was afraid to call before this because I thought you would /or/or/."

> My name is Chuck. My number is 414-241-3066 (after we move in March or April 414-355-5454). I will never /or/or/. Due to circumstances beyond my control, I was involved with the TRS80 since its inception in 197? (my memory is hazy as to the exact date). As an employee of Radio Shack at the time (repair center technician), I did such things as call Ft. Worth and ask "How do I make a copy of the disk you sent me. FORMAT? What is FORMAT? BACKUP? What is BACKUP?" This was followed by questions such as "RS232? What the heck does that mean?" If I did it, so can you. Really - I (and our entire support staff) are always ready to help you with your LDOS questions, and I will always answer any JCL questions, as I appreciate the challenge. I use both JCL and the new TYPEIN utility daily, and I don't know how I ever got along without them.

<u>JCL HINTS AND TIPS:</u>

I have heard complaints that screen format of JCL leaves something to be desired when using the //PAUSE, //INPUT, etc. macros, as the macro itself appears on the screen. Remember the "%" symbol (Ver 5.1)? Try a line such as:

```
     //PAUSE %1DPress <ENTER> to continue
```

The %1D means "move the display cursor to the beginning of the line." This will cause the following message to cover up the //Macro. For those of you concerned with screen format, this should help.

<u>JCL QUESTION OF THE QUARTER:</u>

Last issue's winner was Jaques Yerby - the only person to mail in an answer to the question. The correct answer was that entering a command to DO the file with no parameters would NOT produce any executable lines. Therefore, any existing SYSTEM/JCL file would be executed instead! Consider the case where the last JCL you compiled and executed was to format a disk. Well, that same format command would be in the SYSTEM/JCL file, and would execute. Of course, unexpected formats are very often fatal to the information on the target disk! A good way to safeguard against this occurrence is to start every JCL file with an execution comment, even if it is just a period followed by a carriage return.

This issue's question is more straightforward than the last. Again, free software is available for three lucky people. To be eligible, send in your answer to the question before March 15th, 1983. A drawing will be held to determine the winners. A FED, LED, TBA, Filter disk or Utility disk will be the prize.

Question: What is wrong with the following example, and how can it be fixed?

```
//. January question JCL now compiling. . .
//IF A
//ASSIGN A=1
//END
//IF B
//ASSIGN B=2
//END
//PAUSE  Press <ENTER> to see the results
//IF A+B
. A or B was used as a parameter
//ELSE
. Neither A or B was used as a parameter
```

```
//END
```

To keep the examples used in these questions of some practical value, I'll try to base them on actual customer service calls or letters concerning JCL. For those of you who want to see a practical use for a JCL, try and talk your Radio Shack Computer Center into listing out the INITHD JCL file that comes with the Model I/III hard disk system.

LES INFORMATION by Les Mikesell

The LDOS SYSTEM (FAST) and SYSTEM (SLOW) commands are implemented to work only with hardware that uses a value of 1 output to port FE to increase the CPU speed and a value of 0 to go back to normal speed.

An image of the value last output to port FE is maintained in the system. The SYSTEM (FAST) and (SLOW) commands pick up this stored byte, merge in the appropriate setting for bit 0, output the new value to port FE, store the new value, and set the FAST bit in SFLAG$ accordingly. The SFLAG$ bit is used by the system in certain timing loops, especially the "motor-on" delay time for the disk drives, and this byte is saved in the CONFIG/SYS file by the SYSTEM (SYSGEN) command.

When SYS0 initializes at boot-up, a value of 0 is output to port FE to slow system down until the config file is loaded (if there is one). After the config file loads, the bit in SFLAG$ is tested to determine the correct state for the CPU speed, and the value to activate the fast speed is output if the bit is set.

There are modifications currently available that use different port addresses and values to control the speed. The following addresses may be patched to change the CPU speed control function to work with different hardware. The first list is the memory address and normal contents with a description of the function performed. The second list contains the disk locations for a 'D' type patch to change the values or port address used.

```
---------------------------------------------------------
Model 1, version 5.1.3
SYS0/ SYS
;
Address- values          function
X'4427'=00          ;image of value output to port

X'4E1D'=3E 00       ;LD A,0
X'4E1F'=D3 FE       ;OUT (FEH),A -initial slowdown at bootup

X'5072'=$1 FE 01    ;LD BC,01FEH -set up value/port address

X'507B'=ED 79       ;LD (C),B  - output (if FAST bit is set after config file loads)

Model 1 5.1.3
SYS7 ... SYSTEM (FAST or SLOW)
(sets bit 3 of SFLAG$ accordingly)

X'521F'=F6 01       ;OR 1 - Set FAST by setting bit 1

X'522B'=E6 FE       ;AND 0FEH - set SLOW by resetting bit 0

X'5230'=D3 FE       ;OUT (FEH),A  - output value
```

Locations for 'D' patches (note that the values shown are the normal contents and should be changed according to the hardware modifications)

```
SYS0/SYS:
D0C,D8=00      <= SLOW value
D0C,FE=FE      <= Port address
D0F,35=FE      <= Port address
D0F,36=01      <= FAST value

SYS7/SYS:
D0D,9B=01 <=Value to OR with image byte to produce FAST value
D0D,A7=FE <=Value to AND with image byte to produce SLOW value
D0D,AC=FE <=Port address
```

```
Model 3, version 5.1.3
SYS0/SYS
;
Address- values          function
X'42A0'=00         ;image of value output to port
;
X'4E49'=3A A0 42   ;LD A,(42A0H)   - get initial value (a 0)
X'4E4C'=D3 FE      ;OUT (FEH),A    - output for slowdown

X'5062'=01 FE 01   ;LD BC,01FEH - set up value/port address

X'506B'=ED 79      ;LD (C),B  - output (if FAST bit is set after config file loads)


Model 3 5.1.3
SYS7 ... SYSTEM (FAST or SLOW)
(sets bit 3 of SFLAG$ accordingly)

X'521F'=F6 01         ;OR 1 - Set FAST

X'522B'=E6 FE         ;AND $FEH (set SLOW by resetting bit 0)

X'5230'=D3 FE         ;OUT (FEH),A  - output value
```

Locations for 'D' patches (note that the values shown are the normal contents and should be changed according to the hardware modifications)

```
SYS0/SYS:
D02,23=00      <= SLOW value
D0D,48=FE      <= Port address
D0F,66=FE      <= Port address
D0F,67=01      <= FAST value

SYS7/SYS:
D0D,A6=01 <=Value to OR with image byte to produce FAST value
D0D,B2=FE <=Value to AND with image byte to produce SLOW value
D0D,B7=FE <=Port address
```

Some users have reported frequent I/O errors with certain double headed disk drives. The patch reported in an earlier issue of the Quarterly solved the problem by changing the "SEEK" command issued by the disk driver to "seek with verify". This causes a delay on every sector access which makes the drives function properly, but slows the system down much more than necessary. The following program will add the delay needed by the drive only after the head has been moved and requires additional settling time.

Assemble the program with the name "SLOSTEP/DCT" and install it with the command SYSTEM (DRIVE=n,DRIVER) (where n is the number of the drive). When the system prompts for the name of the driver, enter SLOSTEP, and the program will be installed. If the delay is required for more than one drive in the system, the procedure must be repeated for each drive.

```
;
;SLOSTEP/DCT
;An addition to the LDOS disk drivers to provide an
;Increased delay for head settling whenever the disk
;Function would cause the head to move.
;This may provide greater reliability with certain
;Double-headed drives
;
DELAY1  EQU     100      ;Delay before stepping
DELAY   EQU     1350     ;<=countdown delay for head settling
;
;SYSTEM ENTRY POINT DEFINITIONS...
@DSPLY  EQU     4467H
@EXIT   EQU     402DH
@PAUSE  EQU     60H
DCT$    EQU     4700H
HIGH$   EQU     4049H
HIGH3   EQU     4411H
;
;
        ORG     6000H
;On entry to a /DCT program called by the
;SYSTEM (DRIVE=n,DRIVER), DE contains the address of the
;Specified DCT
;
ENTRY:  PUSH    DE              ; Save DCT pointer passed by SYSTEM
        LD      HL,LOGON        ; =>Logon message
        CALL    @DSPLY          ; print it
        LD      HL,(HIGH$)      ;Pick up available memory pointer
; CHECK FOR MOD 3...
        LD      A,(125H)        ; TEST FOR MOD III
        CP      'I'             ; match if mod III
        JR      NZ,MODI         ; go if Mod I
; IF THIS IS A MOD III
        LD      HL,HIGH3        ;Mod 3 HIGH$
        LD      (NMEM),HL       ;Save for later
        LD      HL,(4411H)      ;Get contents
;
MODI:   LD      (MYMEM),HL      ;Store away for relocating
        LD      (OLDHI),HL      ;And in header
;
        POP     IY              ; put DCT pointer in IY
; IY=> DCT FOR THIS DRIVE
        LD      H,(IY+2)
        LD      L,(IY+1)        ;HL=Driver address
        LD      (DVR1),HL       ;Stuff into program..
        LD      (DVR2),HL       ;Everywhere needed
        LD      (DVR3),HL
        LD      DE,$-$          ;<=HIGH$ value
MYMEM   EQU     $-2
        LD      HL,LAST         ;Relocate working parts
        LD      BC,LEN          ;Byte count to move
        LDDR                    ;Move to high memory
        LD      (HIGH$),DE      ;Lower HIGH$ to protect
NMEM    EQU     $-2             ;<=self modifying for mod 1/3
```

```
        INC     DE              ;Point to driver start
        LD      (IY+1),E        ;Store new driver address
        LD      (IY+2),D        ;In DCT for drive
        JP      @EXIT
;
LOGON:  DB      'Stepping delay for double headed drives',0DH
;
;LDOS type header for hi-memory module:
START:  JR      DRIVER          ;Branch around header
OLDHI:  DW      0               ;<=pointer to previous HIGH$
        DB      5,'STEPS'       ;Length / name
;
;Actual routine
;Trap only functions that require head movement
DRIVER: LD      A,B             ; GET FCN CODE
        CP      0AH             ; VERIFY?
        JR      Z,CHECK
        CP      0DH             ; WRITE SECTOR?
        JR      Z,CHECK
        CP      0EH             ; WRITE SYSTEM SEC?
        JR      Z,CHECK         ; SAME
        CP      9               ; READ SECTOR?
        JR      Z,CHECK
        CP      6               ;SEEK?
        JR      Z,CHECK
        CP      5               ;STEPIN?
        JR      Z,STEPIN
        CP      4               ;RESTORE?
        R       Z,STEPIN        ;Same for RESTORE
OLDDVR: JP      $-$             ;Go direct to driver if other function
DVR1    EQU     $-2
;
STEPIN: PUSH    BC
        LD      BC,100
        CALL    @PAUSE
        POP     BC
        PUSH    BC
        JR      CMDOUT          ;B already has correct fcn number
;
;Check if head is going to move for these commands...
 CHECK: LD      A,D             ;Desired track
        CP      (IY+5)          ;Current head position
        JR      Z,OLDDVR        ;Go direct if not moving head
;
;If head is going to move
GOSLO:  PUSH    BC
        LD      BC,100
        CALL    @PAUSE          ;Slight pause, then..
        LD      B,6             ;Seek track first...
;
CMDOUT: CALL    $-$             ;Using old driver
DVR2    EQU     $-2
        LD      B,7             ;Tstbsy
        CALL    $-$             ;Check disk status/wait till done
DVR3    EQU     $-2
        LD      BC,DELAY        ;Oelay countdown
        CALL    @PAUSE          ;Let heads settle
        POP     BC              ;Get back function code
        LD      A,B             ;Was it a seek, stepin or restore?
```

```
01240         CP      7
01250         JR      NC,OLDDVR       ;Complete function if read or write
01260         XOR     A               ;Set Z flag for good completion
01270         RET                     ;Done if seek or stepin
01280 ;
01290 LAST    EQU     $-1             ;Symbols for assembly
01300 LEN     EQU     $-START
01310 ;
01320         END     ENTRY
```

LATE BREAKING NEWS. ETC.

Following are bits and pieces received too late to be put in the middle of the newsletter.

With Scripsit and LSCRIPT (the LDOS patched version of Scripsit), doing an L or S command will load or save a file. If no filename is given, the last entered filename will be used. This is normally convenient as it saves having to type in the filename when saving the file to disk. However, if you have made changes and do not want to overwrite the original document, you must remember to save the file under a different name. Just entering S rather than S FILENAME will save it under the name you used to load it, thereby overwriting the original file. Just so everyone has an option, the following patch to LSCRIPT will make it ignore any previously entered filename and force you to specify the filename to load or save.

```
. Patch for LSCRIPT to force filename to be specified
. on a Save command.
.
D0B,D0=28 06 CD 14 63 CD FE 53 3E 10 CA
.
. This next part is to force a filename to be specified
. on a Load command.
.
D0B,37=13 42 6C 6F 63 6B 20 74 6F 20 65 78 63 68 61 6E
D0B,47=67 65 3F 20 CD FF 53 C0 C3 A8 5D 00
D0B,59=CD 1B 5D
. EOP
```

LDOS PROBLEM REPORT FORM

```
====================================================================================
   Date ___/___/___    Customer Name _____________________________________________

   Serial # _______________________  Version ___________  Model_______________________
====================================================================================
   Address ___________________________________________________________________________

   City _____________________________  State _____________  Zip ______________________

   Country ____________________________  Phone # (_____) _______-______________________
====================================================================================
CPU:  (_____) TRS-80    (____) LNW    (____) PMC   (____) Video Genie

MODEL I (____)  III (_____)     (____) Other _____________________________________

LOWER CASE:  (_____) RS    (______) PENCIL  (_____) Other ______________________
====================================================================================
E.I.: (____) RS     (____) LX80     (____) LNW     (____) OMIKRON

      (____) Other __________________________________________

DDEN: (____) Percom     (____) DOC    (____) LNW   (____) LNW 5/8   (____) RS

      (____) Other _____________________________________
====================================================================================
CLOCK SPEED UP (____) _____________________________  RATE in MHZ __________________

CLOCK/CALENDAR (____) T-TIMER    (____) TCHRON   (____) METHUSELAH  (____) TIK-TOK

               (____) Other __________________________________
====================================================================================
MODEM: (____) ______________________________  RS232 _________________________________
====================================================================================
VIDEO: (____) STANDARD    (____) Other ______________________   HI-RES (____)
====================================================================================
DRIVES: 5'' (_____) ______________________    8'' (_____) __________________________

        5'' (_____) ______________________    8'' (_____) __________________________

        5'' (_____) ______________________    8'' (_____) __________________________

        5'' (_____) ______________________    8'' (_____) __________________________

HARD DRIVE:  5''(_____)  8''(_____)  Controller ________________________________
====================================================================================
PRINTER:  Parallel (_____)  Serial (_____)  Brand _____________________________
====================================================================================
```

(OVER)

LDOS PROBLEM REPORT FORM

This form is provided to report problems that occur when using the LDOS operating system. In the space below, please list any system configuration you are using along with a description of the problem. Be sure to indicate whether the problem is with an LDOS file, an application program, or a combination of both. On the back of this sheet is a place for your name and address, as well as a checklist of hardware and other information. It is extremely important to mark your particular hardware, especially if it is not-standard.