$4

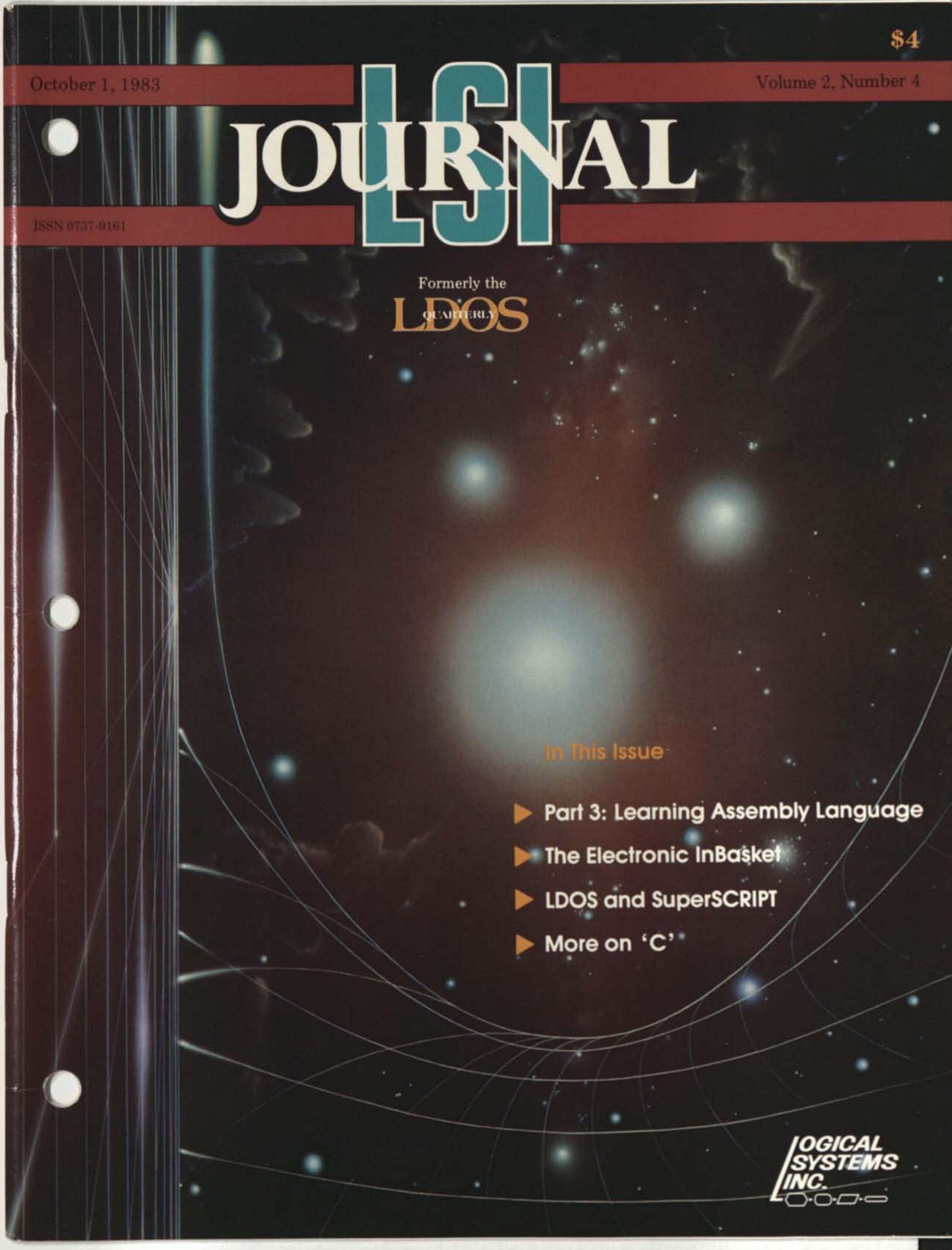
October 1, 1983

Volume 2, Number 4

LSI JOURNAL

ISSN 0737-9161

Formerly the LDOS QUARTERLY



In This Issue

- Part 3: Learning Assembly Language
- The Electronic InBasket
- LDOS and SuperSCRIPT
- More on 'C'

LOGICAL SYSTEMS INC.

ANTHOLOGY ONE

SIX ISSUES TOGETHER AT LAST!

Now Available — Volume One of the LDOS Quarterlies. Including the July 1981 through October 1982 issues. Over 325 pages of information sure to benefit all LDOS users.

LOGICAL SYSTEMS INC.

Only **$19.00**
Plus $4.00 Shipping and Handling

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

The Next Generation

SUPERLOG

ADVANCED ELECTRONIC NOTEBOOK
BY **KSoft**

Over the past two years, LOG Electronic Notebook has quietly been creating a revolution in personal information management. Designed to emulate a familiar pencil and notebook, LOG Electronic Notebook can do for random information what a spreadsheet program does for numbers.

Now, even the best has been improved! KSoft is pleased to announce SUPERLOG, the next generation of the LOG family. SUPERLOG is not a patch! It is a totally rewritten version of the original LOG concept, fully compatible with the LDOS 5.1.3 operating system currently endorsed by Tandy.

SUPERLOG retains all of the versatile features of LOG while adding many new options requested by professional users: Floppy or hard disk. Any number of LOG files per diskette. 1 to 32767 pages per file. Password protection and error checking. New text editing commands include automatic text Wrap-Around. Expand and Delete for entire lines, a Page Copy command, and an Undo key to reverse editing changes. Cursor motion is more flexible with new key commands plus a Forms simulator. The SEARCH function is greatly enhanced with a Wild-Card character, case-independent search, and multiple word search at 10 pages/second.

Also Note: SUPERLOG is now fully interrupt activated; it may be accessed from practically any foreground task including LDOS Utilities, LBASIC, LSCRIPT, EDAS, etc. with non-destructive return to the foreground program. No other information management program is this versatile!

Write or call Today! We'll be glad to tell you about SUPERLOG and what it can do for you!

SUPERLOG Specify Model I or III. $119.95
LDOS 5.1.3, 48K, and 2 Drives required.
(Model IV version to be offered soon.)
LOG TRSDOS versions, Models, I, III still available.

KSoft
318 Lakeside Drive
Brandon, MS 39042
(TRSDOS is a trademark of Tandy Corporation)

(601) 992-2239
Mastercard and Visa accepted.
Add $5.00 for shipping and handling.
(LDOS is a trademark of Logical Systems Inc.)

THE LSI JOURNAL October 1983 Volume 2, Number 4

CONTENTS

INTRODUCTION FROM LSI:

ARTICLE SUBMISSION POLICY .... Page 2
VIEW FROM THE BOTTOM FLOOR .... Page 3
A CASE OF MIS-ALLOCATION .... Page 8
NEW PRODUCT ANNOUNCEMENTS .... Page 14

FROM OUR USERS:

The Electronic InBasket .... Page 15
Fast Graphics for 'LC' .... Page 16
Using Interrupts and SVCs in FORTRAN .... Page 19

REGULAR USER COLUMNS:

... er ... Earle Robinson at large .... Page 28
* PARITY = ODD * Tim Daneliuk .... Page 31
'C' What's Happening - Earl Terwilliger .... Page 34

FROM THE LDOS SUPPORT STAFF:

Items of General Interest .... Page 38
updates, patches and clarifications

LET US ASSEMBLE - Rich learns yet more assembler .... Page 4Ø

LDOS: HOW IT WORKS - Using non-relocatable code .... Page 5Ø
"Foreign" machine code, and other stuff in high memory discussed

THE JCL CORNER - by Chuck (sort of) .... Page 52
Automatic Chaining with JCL .... Page 52

Letters from the Customer Service Mailbag .... Page 54

LDOS and SuperSCRIPSIT .... Page 55

MAX-8Ø Memory Map .... Page 58

Performing DATE Conversions in BASIC .... Page 6Ø

LES INFORMATION - by Les Mikesell .... Page 61

View From Below the Bottom Floor .... Page 64

Copyright © 1983 by Logical Systems, Incorporated
897Ø N. 55th Street P.O. Box 23956
Milwaukee, Wisconsin 53223
Main switchboard: (414) 355-5454
LDOS Hotline: (414) 355-4463

The LSI Journal policy on the submission and payment for articles is as follows:

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

1. A cover letter, summarizing the content and intent of the article
2. A printed hardcopy (lineprinted or typewritten) of the article. Desired print effects and formatting should be indicated where necessary.

A diskette with--

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

Please do not send in printed text without a diskette, as it will NOT be considered for publication. Payment will be made in the form of an LSI product, or $4Ø per published page in the current LSI Journal format. The size of the article will determine the value of the LSI product available as payment.

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

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

Submissions should be sent to:

The LSI Journal Editor
c/o Logical Systems, Inc.
897Ø N. 55th Street
P.O. Box 23956
Milwaukee, Wisconsin 53223

VIEW FROM THE BOTTOM FLOOR

by Bill Schroeder

Yes, there is now an LDOS 5.1.4!

OK, what did LSI do to 5.1.3 to make it 5.1.4? From the standpoint of the DOS itself, 5.1.4 is simply a needed change in the version number after a dozen or so patches. This means that 5.1.4 contains all patches to date, some are VERY IMPORTANT, others are less important, and some that are just plain arbitrary. New functions have been added that make this a very valuable update for our LDOS users and a must for Model 4 users.

A BIG FEATURE OF LDOS 5.1.4 IS THAT THE ORIGINAL FED (LSI FILE ZAPPER) IS NOW INCLUDED WITH THE LDOS 5.1.4, AT NO EXTRA CHARGE!!!

FED is the famous LSI utility that allows simple maintenance and updating of all LDOS-type files. FED originally sold for $4Ø and was recently reduced to $19 with the introduction of FED-II. Sure, there is an ulterior motive. First, I would like all of our users to have the power of a FED-type program, and then again there is FED-II. I believe that when our users find out how handy FED is, they will become purchasers of our FED-II product. I may be wrong, but in any case, the LDOS user benefits.

A NEW HIGH SPEED BACKUP UTILITY (QFB) COMES WITH LDOS 5.1.4

I have often been asked if LSI could provide a FASTER method of creating a "mirror-image" duplicate of an LDOS-type disk. Well, 5.1.4 has this feature in a new program called "QFB" (for Quick Format and Backup). This new utility will duplicate a disk in about half (or less) of the time that a FORMAT/BACKUP sequence takes. QFB and its documentation are provided at no additional charge with a 5.1.4 update. Please note that QFB is for mirror-image backups of standard LDOS-formatted diskettes only.

IMPORTANT NOTE:

```
|==================================================|
| FED and QFB are NOT included with smal-LDOS 5.1.4 |
|==================================================|
```

The cost to update to 5.1.4 is just $1Ø ($5 with ESA). This is an important and valuable update, so please send in your master disk (disks in the case of Model 1) and let LSI update your system. The official date of 5.1.4 is September 1, 1983. There are still many 5.1.3 systems available from LSI dealers and from Radio Shack and these can and should be purchased without worry that they are not 5.1.4 versions. ANYONE WHO PURCHASES THE FULL LDOS 5.1.3 SYSTEM AFTER AUGUST 1, 1983 IS ENTITLED TO A FREE 5.1.4 UPDATE AT ANY TIME! Note: Proof of purchase date is required and must accompany master disks sent to LSI for this "FREE" update.

Another item worthy of comment is the <u>NEW</u> name for the LDOS Quarterly. It is now called the "**LSI JOURNAL**". The name change was necessary, because our publication will be addressing many wide and varied topics in the future. For the present, LDOS and its related products will be the main thrust of this publication but in the future many other topics and products will be discussed. The new name more aptly describes the future route of our publication. For those perceptive folks, you may realize that the removal of the word QUARTERLY from the publication could mean that a more frequent rate of publication may be in the offing. You never can tell ....

More "news" is that the cost to receive the LSI JOURNAL has been reduced. Effective immediately, a four issue subscription is just $14.95! Subscriptions to the LSI JOURNAL are now available to ANYONE. That's right, there is no requirement to be a registered owner of any LSI product. LSI will continue to bring you this publication, full of straight forward technical and user information, from professionals, and with very little advertising material.

The ESA program is no longer available but all existing ESA agreements will be HONORED TO THE LETTER until expiration.

We have opened up the LDOS SIG (Special Interest Group) on microNET/CompuServe to the public so that all LDOS and TRSDOS 6.x users, or anyone else for that matter, can now access this bulletin board service. Note: You must be a current member of CompuServe to use this service.

Occasionally, I get a letter from an LSI customer, who complains about LSI offering a new generation of a product or a drop in the price of an existing product. I would like to address this issue.

First of all, let's discuss purchased products and updates. Updates are just that. They are intended to correct defects or oversights in a released product. When we enhance a product, it is not the same thing. For some reason, some software purchasers have placed the software industry in a category previously unknown in a capitalistic economy. An industry where a purchase by a customer creates a permanent responsibility to provide future enhancements or version changes at little or no cost to that customer. I can't think of another industry that is ever expected to provide this type of customer support or service. There is no way that a non-cottage industry company could function on this basis. Witness the many, many software companies that are no longer in business due to their attempts to satisfy these unrealistic expectations.

To bring the issue very close to home, look at the TRS-8Ø software industry. I guarantee you that as you look through the software you have purchased over the past two years, you will find that over half of the providers of those products are no longer around to provide any type of support, enhancements, or upgrades. Tough industry to stay afloat in-- don't you think?

Now look elsewhere in the industry. I have purchased many printers for LSI, often to find that within months (in one case days) that the manufacturer dropped the price, enhanced the product, or in most cases did BOTH at the same time. Was I upset or did I feel used or abused? Of course not, things change, markets change, competition changes, production costs and techniques change, and on and on. I want to see things progress and I am quite willing to pay the price for that progress. The advancements in computer hardware and software have to be paid for and the companies involved must make a profit.

About nominal cost upgrade to new versions of a product I can only say... HOW? Some letters I receive indicate that an upgraded or enhanced version of a product should be provided for free or at nominal cost because the customer supported the original effort by purchasing the product. COME ON NOW! I purchased an MX-8Ø when it first came out. I have also purchased a radar detector, pocket calculators, digital watches, 35mm cameras, color TVs, air conditioners, new cars, FM personal radios, VCRs and dozens of other items shortly after they were introduced for consumption. Within a year or so of purchasing every one of these items, I could have bought a BETTER VERSION, FOR LESS MONEY! Why? Simply because the manufacturer was able to continue to upgrade and enhance the original product to create better, cheaper, more desirable versions of the product. It should be clearly noted that NOT ONE of the companies that manufactured or sold me these products offered any type of TRADE-IN, UPGRADE, UPDATE, or even a token discount for support of their original version. Nor would I have been so naive as to have expected one.

I believe that I got what I paid for with every one of these products. I bought the item as it was designed, for the price as marked. No one held a gun to my head or forced me to buy the first, second or third generation of a product. When and what to buy was MY choice. Yes, a later generation of a product is usually better and costs less, but I had the use of the item for the ensuing time period until that next version became available. I certainly did not expect the company to let me trade up to a new VCR or SLR camera at a nominal update fee. If I wanted the newer model, I sold, gave away or threw away the one I had and BOUGHT the NEW version. However, whenever I purchased a product that did not function as I was lead to believe, I insisted that it be repaired, replaced, upgraded or as a last resort that I be allowed to return it for a full refund. In all fairness though, I would only expect these options within a reasonable time frame.

If one is not willing to accept progress in technology as the "way of things" then, it would be safest to never buy anything, unless it has been out of production for ten or more years. In that case, you probably won't ever have a new version of the product to feel bad about.

Which brings me to another point. The customer who decides to purchase a new, probably incompatible computer. I own a VHS format video tape recorder. If I decide to go out and buy a BETA recorder, knowing full well that these two formats are incompatible, I certainly can't tell the stores were I bought my "VIDEO SOFTWARE" (tapes) that they should or have to give me BETA-type copies of the movies at a nominal charge. I'd be thrown out of the store. Alternatively, if for some strange reason the store did do this type of thing, it would soon be bankrupt.

In spite of this, some of our customers think LSI should "GIVE" them Model 4 versions of our products. Why? Well, because they once bought the Model-III version. Now I ask you, did Radio Shack "GIVE" you a Model 4 computer on the basis that you have one coming because two years ago you were nice to them and bought a Model-III? I doubt it.

Yet, with all this as common knowledge to ALL of the American consumers, there is still a strange perception about software suppliers. It's almost as though some of our customers think that writing software is a very simple process and, therefore, is worth very little. It is very expensive to design, write, test, debug, document, publish, advertise, and support GOOD computer software! Many of the software suppliers have not made the financial grade and are no longer around for disgruntled customers to complain to. Maybe they should not have used as much of their resources providing upgrades to working products, each of which lost them money.

It is your choice to buy a MAX-8Ø or a Model 4, or any other computer that LSI is supporting or is going to support in the future. It is your choice to get CP/M or any other OS, and yes, some of our products are coming out on CP/M, XENIX and MS-DOS. We do not intend to offer upgrades or updates or trade-ins under these circumstances. LSI policy will be to update ONLY the same product title, same product series for the same computer and operating system. These updates will be for ERROR corrections only. Enhanced or second generation products and the same product for a different environment, must be purchased.

I often hear from customers and other people in the micro industry how terribly profitable the software industry is. That is plain ignorance speaking (or maybe too much belief in what Wayne Green has said in 8Ø-Micro). If this industry was all a "bed of roses" then I ask you, why is the failure rate of young software companies SO high? It is not common for companies that are making high profits to go out of business, with many unpaid debts. Watch the magazines. Watch the software companies come and go. Better yet, if you are one of the people who thinks this industry is so easy to make money in, you should take the Great American Alternative and start your own software company, or try to make a living as a independent software author. The odds are very much against success, but some will succeed and become financially strong. These are the ones with FIRM, FAIR and PROFITABLE policies regarding their goods and services.

Take FED and FED-II as examples. FED is one of LSI's most popular utilities. The original FED sold hundreds of copies but has not yet even paid for its development. FED-II cost over twice as much to develop. We priced the original FED at $4Ø, when we introduced FED-II we dropped the price of FED to $19 and offered the new version FED-II, at the $4Ø price. Yes, a better product for the same money! Now we have included the original FED on LDOS 5.1.4 for (almost) FREE. Well and good, but all this does not change the fact that FED and many of our other products are efforts to support our operating systems and are not likely to become profitable products to LSI. Some day, with our ongoing enhancements to FED and rewriting it in "C", for use on other machines, LSI may break even on the product, or possibly even make some profit on it, but I doubt it.

The FED example is not unique. LSI has in the past and will in the future, invest tens of thousands of dollars on support items for our product lines. Some of these will recover their costs, some will make a profit, but alas, most will be "losers" as stand alone products. Many LSI products are slated as non-profit products when they are started. They are created to support our products and our most valued asset, our customers.

Software economics and the profitability of a software company is hard to understand and/or control. Most software companies and customers have very close personal ties to the computer industry in one fashion or another. Most are programmers and computer enthusiasts first and business people second. Not a good ordering of things if the company is going to be here some years from now.

LSI intends to be in this industry for a long time to come. We will be fair with our policies, but not to the overall detriment of LSI. That would be a detriment to our customers as well. The ULTIMATE in bad support is having the company you buy a product from go out of business. We doubt that this will happen at LSI, because we do not intend to give away our products. Many software companies have failed for this reason, and this reason alone.

I should make one thing very clear-- I do not believe that a defective product is the responsibility of the customer. The software provider has a responsibility to provide updates, at a modest handling cost, to repair defects found in any version of a product. The customer has the right to expect that. But if the product is purchased by the customer and it functions as advertised and documented at the time of purchase, then the customer has gotten what he agreed to pay for at the time of purchase and should not expect more.

I have had many questions about our software protection plans for the LSI 6.x products.

First of all, I must say that we have no intention of changing our stated RIGHT TO PROTECT OUR PROPERTY. Those who disagree and feel that by purchasing an LSI product they should have complete ownership rights to that product, with the right to duplicate it whenever and for whomever they like, are morally, legally, and ethically incorrect.

Secondly, we do NOT have the intention at this time of protecting more than a selected few of our products. We will be protecting products that we have found to be the most likely to be pirated. We do get calls daily from thieves, some of them openly admit to having pirated copies of our products, and justify this, with some assinine reason why THEY have the right to STEAL FROM US. I for one am just plain sick of it. Most other companies in this industry are taking a similar stand and with good reason. VISICORP, MICROSOFT, TANDY and many others are now protecting at least SOME of their products from illegal duplication. So whether users like it or not, they will have to get used to it. Even the much acclaimed and probably the most owned utility in the TRS-80 industry is FULLY BACKUP PROTECTED. This is of course SUPER UTILITY from Powersoft.

In the last issue I spoke of our plans to provide SOME of our products for the Model 4 (6.x OS) on disks that only allow a limited number of backups. This is certainly our right and we are providing several items in this manner. We are under no obligation to provide unlimited reproduction of our property. The number of backups that can be made from these products will vary from 3 to 25 depending on the nature of the product and how often someone offers to sell me a pirated copy of my own software. We are not going to indicate when and if a product will be protected or the type of protection that will be used. The packaging of protected products will carry a clear warning (visible without breaking the package seal) indicating the number of backups, if any, that can be made from the master. If this is unacceptable to the customer, and the product has not been opened, the product may be returned for an immediate refund.

At this time, only Model 4 products will be subject to limited backup protection. Future LSI products (for all environments) may be provided on backup-proof media. If we elect to take this step, we will provide two or three MASTER copies with each package.

When you buy a computer, YOU GET ONE! At LSI we have many products which we purchase for use on XENIX, CP/M, MS-DOS and LDOS, some of which are provided on backup-proof media. We have no problem with this concept or the use of these products. We feel this is the right of the software producer. Of course, it is the right of the customer not to buy the product. If you enter the world of the IBM-PC, you will find that programs on protected media are very common.

One decision that we are quite sure of is that LSI WILL BE SUPPORTING MS-DOS, CP/M and XENIX with future products. This is a market of the future and LSI will be there.

We feel that many of our products and our superior after sale support make many LSI products good bargains.

In the interest of fairness to our customers, we wish to do something that most retailers would never do, tell them well in advance of planned price increases. Effective with our January 1984 catalog, you will see rises in the retail price of MOST of our products. The increases will vary from little or nothing to as much as several times the present price. You have plenty of time to take advantage of our present pricing and acquire the items that are of interest to you. I recommend that you do so now, because as of the first of next year, you can expect to pay more for those items.

LSI IS NOW LICENSING THE LDOS 6.x SYSTEM TO THIRD PARTY IMPLEMENTORS

As of September 1983, LSI began making available FULL licenses to the LDOS 6.x system! LSI will provide COMPLETE LDOS 6.x SOURCE CODE to outside companies so that the LDOS 6.x system may be quickly made available on many diverse machines. Our licensing terms are much more liberal than those of other OS licensing companies, and a lot less expensive. There are many, many details involved in these licenses and it would not be appropriate to go into them at this time. But, if there is a computer that you think the LDOS system "should" be on, let the manufacturer of that machine, and LSI know.

These implementations will all be media and software compatible with other LDOS 6.x systems within the limits of the hardware. For instance, almost any Radio Shack Model 4 software should run on these other machine implementations, without change!

This is, of course, due to the nature of the 6.x SVC system. Any programmer writing for the 6.x version need only use the SVC structure in the documented manner, and the software will run on other systems with no problems. The beauty of the 6.x architecture is that there are NO HARD ADDRESSES, STORAGE LOCATIONS or VECTORS, at all!

There are several OEMs and some well known OS implementors either considering this program or already in it. In the next issue of the LSI JOURNAL, I will be announcing some of the machines being supported and the companies who are doing the implementations. Many of our customers will be very pleasantly surprised.

ABOUT SOME NEW PRODUCTS (and SPECIALS)

By the time that you read this, you should be able to run down to your nearest Radio Shack store and pick up a copy of the MODEL 4 TECHNICAL MANUAL, Catalog Number 26-2110. This is without a doubt one of the best manuals ever published by Radio Shack. We have heard many negative comments about the superficial nature of the Model 4 user's manual. The "USER'S" manual was intended as just that. No programming or system interface information was provided (or intended). The release of this technical manual should clear up all questions regarding the 6.x system and its functions. This document is the ONLY TRSDOS 6.x (LDOS 6.x) official specification.

Every effort will be made to maintain full upward compatibility to this spec as the 6.x system is expanded. Any use of the system in ANY WAY that is not stated in this spec will cause program compatibility problems. There will be other publications that may provide incorrect information regarding things that we have every intention of changing. So, **beware**! DO NOT USE ANY FUNCTION or SVC THAT IS NOT DOCUMENTED DIRECTLY BY LSI or RADIO SHACK. There are many changes planned for the future of our 6.x product. Don't become the victim of unofficial information. THINK! Get a copy of the official Model 4 Tech Spec from Radio Shack and treat it as the "BIBLE TO 6.x". You'll be happier in the long run.

Now how about a year end special just for Model 4 owners? We have LS-FEDII at $49, LS-FM at $49, and LS-QFB/COMP at $39, for a total of $137. But from now until December 31, 1983, you can get all three for just $98. That's right, for the price of LS-FM and LS-FEDII alone, you get the LS-QFB/COMP package thrown in. Don't pass this one up. Beat the price increases. The savings amount to OVER 28% OFF the combined retail prices. When ordering this special offer specify the "LSI MODEL 4, TOOLKIT SPECIAL" and you will get all three products for just $98 plus $5 shipping and handling.

CP/M for your Model 4 is now available. CP/M 2.2 has been adapted to your Model 4 by a company called MONTEZUMA MICRO. We have a copy here for review and it seems to br fully functional. For complete information or ordering this product, contact MONTEZUMA MICRO, CP/M group, P.O. Box 32Ø27, Dallas, TX 75232 or call (214) 339-51Ø4 and ask for John, John or John (I think you must be named JOHN to work for these people). With this package your Model 4 will be able to function as a true CP/M machine. Why wait, and wait, and wait, and wait... You can have CP/M now (if you need it).

The number one TRS-8Ø author, BILL BARDEN has outdone himself once again and produced a book entitled, "HOW TO DO IT ON THE TRS-8Ø". This is the ultimate in TRS-8Ø reference material and covers the MODs 1,2,3,CoCo,etc. No TRS-8Ø owner, whether user, programmer, or novice should be without this fantastic book. To this end, LSI has purchased a fair quantity of these "fellers" and is making a special offer to our LDOS users. The book normally sells for $29.95, but while supplies last (and we gots, lots) you can get this great and USEFUL book for just $25 plus $4 shipping and handling if ordered alone, or $2Ø plus $2 if ordered with more than fifty dollars worth of other LSI products. I am so confident that you will agree that this book is the most valuable reference guide in your library, that if you don't agree, SEND IT BACK and we will cheerfully refund your money.

By the end of 1983 LSI hopes to have released several of our products on PC/MS-DOS. We have made a rather large commitment to enter and support the IBM-PC and MS-DOS environment. We will bring to those products the quality and support that you have come to expect from LSI products. We want to stress however, that we intend to continue our support of the Z-8Ø market as we move into the 8Ø88/86 and 68ØØØ markets. We are going to try very hard to continue in both the 8 and 16 bit machines.

A CASE OF MIS-ALLOCATION

by Bill Schroeder

Getting a file onto a diskette is not a simple process. Let's take a closer look. When an operating system is asked to create a file, it must first go to the directory and establish a "directory entry". This is done by placing the filespec and other assorted information into the directory.

Now that the file exists in the directory, data will probably be written to the file. To write to a newly created file, the operating system must now find a piece of storage area on the disk (a granule) at which to start writing the data. Sounds simple so far... well, maybe not.

There are many ways in which an operating system can select just where to start this file. I will describe and discuss a few of them, starting with the method used in LDOS 5.1.3 and before.

This method is to randomly select a cylinder on the disk in a fairly balanced pattern ranging from the first cylinder on the disk (#Ø) through the highest numbered cylinder on the disk. The first granule on the selected cylinder is examined. If the granule is found to be in use or flawed, the system will begin to move upward through the cylinders and through each cylinder's set of granules (e.g. toward the inner tracks). The system will check each granule on the way, looking for a vacant usable one. As soon as an acceptable granule is found, it will be assigned in the GAT and in the file's directory entry as the first granule of that file.

Other variations on the above theme usually involve some type of weighted selection of where to start looking for space on the disk. I will now refer to the random allocation methods as the "RANDOM" method. The algorithm used to select this random location can be written to force the selection to be a number, below the directory, above the directory, on the lower 1/3 or 1/4 of the disk, and on and on. In the case of LDOS 5.1.3, the algorithm attempts to select a cylinder randomly from the entire disk.

Another method of selecting file space is very simple. I will call this method the "CONTROLLED" allocation method. This method simply starts to search at a given spot on the disk and progresses upward through the cylinder numbers until an acceptable space is found. Usually the search will start at cylinder #Ø or #1. Space will then be allocated solidly from the low numbered cylinders (e.g. the outer tracks) to the high numbered cylinders.

Now, some "systems analysts" have taken sides on exactly which method of space allocation is most efficient and reliable for micro floppy disk usage. I must first state that when I started to check out the "real world" implications to the user, I did not think that it could make much of a difference one way or another. But after some experimentation, significant results were obtained. Programs were written, and the results were obtained from actual testing. It became quite apparent that one side of this argument is totally wrong! One method is far superior to the other in all tested cases.

Listed below are the main points that were posed to LSI in a letter from a prominent systems programmer. He is very much in favor of RANDOM allocation techniques. His opinions have been very highly regarded by the TRS-8Ø user's community. These points were brought to LSI's attention as a result of our decision to alter the methods of allocation used in current and future LSI operating system products. No supportive facts, mathematical analysis, or test results were provided to support this individual's position.

1) Random allocation provides more uniform wear of the media surface.

2) Random allocation minimizes the average access time across the media.

3) Random allocation minimizes the number of extents more so than most other methods

4) Random allocation will cause the use of less directory records. Since the number of extents that can be retained in each directory record is finite (four), a new directory record (extended directory entry) must be used when the primary directory entry is filled. In LDOS this happens when a file contains five or more extents.

The letter went on to strongly protest even the consideration of such a radical idea as no longer having RANDOM allocation in LDOS OS products. It even stated this RANDOM technique to be the HEART of the LDOS file handling system. After testing, analysis, and careful consideration of these claims and assumptions, I now believe them to be totally inaccurate.

Now let us review each of these claims for the RANDOM methods:

POINT #1, makes no sense at all. The reason is very simple. All operating systems that do not retain the entire disk directory in memory MUST make frequent directory accesses to locate, OPEN, CLOSE and access records in files. Also, with an overlay based system like LDOS, the operating system itself must go to the directory every time it is necessary to change the overlay which is currently in memory. What all this points to is simple. The directory cylinder will fail from wear long before any other cylinder on the disk wears out. To validate this statement, tests were performed to make actual counts of the accesses of each cylinder on a disk under varying circumstances. These tests proved that under all uses of the LDOS operating system the directory cylinder was accessed five to fifty times more often than any other cylinder.

It is also known that the use of higher cylinder numbers (inner tracks) is much more likely to cause disk faults such as parity or CRC errors. This is because the bits of data are packed much closer together on the inner cylinders. The data density on the inner tracks approaches the maximum reliable resolution that current floppy systems can attain.

POINT #2. The concept of RANDOM placement has little bearing on file access timing. If the file starts at a RANDOM cylinder on the disk it has the same probability of starting at cylinder Ø, as say, cylinder 39. It does not sound like faster access would be attained on these files.

That's just the beginning of the problem. If a file should start on one of the upper cylinder numbers (say above 35 on a 4Ø track disk), it will have a very good chance of wrapping around to a low numbered cylinder. This results in the file being broken and another extent created. A similar action will occur every time a file needs to acquire another gran and the next physical gran on the disk is unavailable. If RANDOM allocation is beginning to sound a little silly, well hang on, IT GETS WORSE.

Because most user's diskettes are between 6Ø and 8Ø percent filled with files, I must assume that any allocation technique other than full RANDOM is better for rapid access. A pure RANDOM technique has the possibility of using both extremes on a disk and nothing near the directory. CONTROLLED allocation will progress toward the directory until the disk is half full, then it will continue on the other side of the directory. The two extremes (cylinders Ø and 39 used) cannot occur until the disk is almost 1ØØ% filled.

It is very important that an operating system tries to keep a file as contiguous as possible. Based on our actual testing, I have found that this could be a valid reason to have RANDOM allocation... if there is only one file on a disk, and the file is not very large, and the RANDOM starting granule is near the directory.

Just to verify these statements, I created files using a test program, using both the RANDOM and CONTROLLED methods. Then I OPENed, READ and CLOSEd the files in a manner which assured that a similar number of files were accessed and the same number of records were read. Surprise! Here too, CONTROLLED access was consistently faster. In general, the disk access time was about 15-2Ø percent faster.

POINT #3, a real bummer here. My actual tests were quite conclusive in this regard. The average number of file extents generated by the RANDOM method was about 2.Ø extents per file. The average from the controlled method was much better at about 1.3 extents per file.

POINT #4, My answer to point number three said enough. However, certain other things were tabulated during my tests. As pointed out earlier, a directory entry can only hold a finite number of extents. In the case of LDOS 5.1, this is four. If the system needs a fifth extent it must use an FXDE (extended directory entry). This entry will use up an additional directory slot, and extended directory entries are very time consuming to the system, under all conditions. Therefore, I counted the number of files that were stored with five or more extents during the testing.

Again, the test results proved my contention that RANDOM allocation is not the best method. When using RANDOM allocation, nineteen files (or more than 15%) required five or more extents. The CONTROLLED allocation method, on the other hand, produced only one file with five extents.

In the spirit of fairness, I would like to point out one condition under which the RANDOM technique may be a better method. This is during a period of alternate extension of two files that are open at the same time (or opened alternately). If this case should arise with CONTROLLED allocation, the two files will occupy alternate granules as they expand. This would be a very inefficient file layout. If a program does extend files alternately, the RANDOM technique would serve this type of program better than the CONTROLLED method. However, most programs that use multiple large files tend to capture large blocks of space at a time. "Space capture" techniques that are used by well written applications tend to eliminate this problem. This concept is generally referred to as file pre-allocation. LDOS does make provision for pre-allocating files (see the "CREATE" library command in your LDOS manual).

I do not find this point to be serious for three reasons. First, there are very few programs that use a file creation procedure that would induce this problem. Second, if this situation does occur, it is very easy to correct the space usage under CONTROLLED allocation by a simple BACKUP $:S :D. This will cause any fragmented files to become contiguous, providing the backup is to an empty disk. Last is the reliability factor. I strongly believe that not utilizing the inner tracks on a floppy disk for as long as possible is far more important than this benefit of RANDOM allocation.

If a disk full of files has become badly fragmented, it can easily be put "back in order" under CONTROLLED allocation. With a RANDOM system, good luck. Every time files are written (backup by class and copy are no exceptions) the RANDOM technique is used. Therefore, the disk will almost always have fragmented files no matter what is done. With the CONTROLLED type of system, a simple backup by class will place all the files on the disk into a minimum fragmentation state (the most efficient).

How many times does a PARITY ERROR or DATA RECORD NOT FOUND ERROR occur? In most cases, this is the fault of the hardware, and is most likely to occur on the inner disk cylinders. While is the fault of the hardware or media, the operating system could help! If the OS uses CONTROLLED allocation starting at the low numbered cylinders and proceeds inward, the use of the most error prone areas of a floppy disk is avoided. The inner cylinders will be used only as the disk is almost full. If you have a drive that is slightly out of alignment or a head with low output, it is possible not to fill the disks over 3/4. This would keep all the files below track 3Ø. This type of control is not available in a RANDOM system. Since we have begun using CONTROLLED allocation (starting at track #1), we have had a marked decrease in the occurrence of disk faults! This in itself is a perfectly valid reason for the change.

I have included my test results in chart format. For those of you who are ambitious, I am making the test program itself available. If you are a die-hard believer in RANDOM, don't take my word for it. Examine the test program and run it yourself. I have the results and will never again allow the use a RANDOM allocation scheme in any LSI operating system.

At LSI we try to do things correctly and pride ourselves in the technical accuracy of our products. If we make a mistake, we will admit it and we will fix it. On the very important concept of allocation, however, it seems that we did not take as much care in deciding on a method as we should have. We fell for the hype that the RANDOM concept was the greatest thing since the Z-8Ø. For this oversight, I sincerely apologize. I will make every effort to see to it that any future design decisions are made purely on facts, and not on heresay or personal opinions or "guesses".

All 5.1.x MASTER duplication disks at LSI will be altered to incorporate the CONTROLLED allocation method as of disks with FILE DATES of September 1, 1983. NOTE: See VIEW FROM THE BOTTOM FLOOR for complete UPDATE details.

In TRSDOS 6.1 and any future LDOS 6.x or LDOS 5.x operating system, all allocation searching will begin at track number 1. The patch to allocate from track number 1, on all LDOS 5.1 products, is as follows:

```
.PATCH TO ALLOCATE STARTING AT CYLINDER #1
. apply to SYS8/SYS.SYSTEM on LDOS 5.1.x
```

```
DØØ,FE=2E Ø1 ØØ ØØ ØØ ØØ
.
. EOP
```

This can be applied as a command line patch by entering the following at LDOS Ready:

```
        PATCH SYS8/SYS.SYSTEM:Ø (DØØ,FE=2E Ø1 ØØ ØØ ØØ ØØ) <ENTER>
```

The 'Ø1' in the above patches is the track number at which all allocation searches will begin.

If you have any intention of changing back and forth between allocation schemes (I can't see any reason for RANDOM, however) or might want to be changing the starting track number that your system will use, then JCL is what you need. JCL will let you create custom commands for your system. This one is fairly simple but shows the basic ability of JCL. This JCL file will create a command that has the following syntax: 'DO TK (RND)' or 'DO TK (TK=nn)' or just 'DO TK'. The 'nn' is of course, a user selectable track (TK) number.

```
. ALLOCATION SETTING JCL - FOR USE WITH LDOS 5.1 - Ø6/Ø1/83
. Set SYS8/SYS to Allocate from track TK.
. If TK is not specified it will be set to Ø1
.
//IF -TK
//ASSIGN TK=Ø1
//END IF
.
. If RANDOM (RND) not requested then patch in CONTROLLED
.
//IF -RND
PATCH SYS8/SYS.SYSTEM:Ø (DØØ,FE=2E #TK# ØØ ØØ ØØ ØØ)
//EXIT
//END IF
.
. If RND is specified, Allocation will be set to RANDOM.
.
PATCH SYS8/SYS.SYSTEM:Ø (DØØ,FE=D5 CD 4E 44 D1 6C)
//EXIT
```

A test program was created to perform the test. The function of the program was to create files as a user might and then kill them. The program is not really important though, as long as the same program is used to test each mode of allocation. The program's creation, deletion and access activities are not intended to duplicate the wide variance of actual system usage. Instead, they are intended to provide a yard stick by which the two allocation methods can be compared. I DO NOT claim that this program accurately represents ANY particular type of actual disk usage.

What I did was run the test program with the operating system set for RANDOM and then again with the operating system set for CONTROLLED (SYS8 properly altered). The computer was a stock Radio Shack Model III. The operating system was LDOS 5.1.3 (Ø6/1Ø/83). Scotch diskettes were used. The step rate of the test drive was set at 6ms. For speed I SYSRESed overlays 2,3,8,1Ø.

Note: Make sure that SYS8 is patched to the desired test method before SYSRESing it! Patching SYS8 on the OS will not alter the one that is already resident in high memory. It is also NOT advisable to SYSRES SYS8 if the ACCESS SPEED test is to be performed because this will alter results in favor of the RANDOM mode. This is because more extents and, therefore, additional accesses to SYS8 are likely to be required by the RANDOM mode.

Speaking of using the SYSRES command, here is a simple /JCL to handle multiple SYSRESes for you.

```
. RES/JCL - Ø6/Ø1/83 - Simple MULTI-SYSRES command
. To USE ENTER "DO RES (1,2,3...)<ENTER>"
. 1,2,3...etc. are the SYS modules to be "RESed"
//if 1
system (sysres=1)
//end if
//if 2
system (sysres=2)
//end if
//if 3
system (sysres=3)
//end if
//if 4
system (sysres=4)
//end if
//if 5
system (sysres=5)
//end if
//if 8
system (sysres=8)
//end if
//if 9
system (sysres=9)
//end if
//if 1Ø
system (sysres=1Ø)
//end if
//if 11
system (sysres=11)
//end if
//if 12
system (sysres=12)
//end if
```

Now back to the subject at hand. The maximum file size was set to 6ØØ records for all runs. Each run was done THREE times. The runs with the lowest and highest average ratios were discarded. This, in effect, tends to remove the extremes. The remaining "median" runs are tabulated below.

FILE ALLOCATION TEST RESULTS

TEST RUN#	AVERAGE NUMBER OF EXTENTS PER FILE AT COMPLETION RANDOM ALC.	CONTROLLED ALC.
1	1.818	1.397
2	1.945	1.367
3	1.794	1.342
4	2.145	1.275
5	1.95Ø	1.315
6	1.867	1.262
7	2.Ø75	1.288
8	2.347	1.386
9	1.895	1.365
1Ø	2.212	1.325
Overall Average	2.ØØ5	1.333

Due to space constraints, the program listings are not included in this issue. Copies of the listings may be obtained for no charge by sending a <u>LARGE</u> self-addressed stamped envelope (6 by 9, with postage for two ounces).

NEW PRODUCTS

Many people have requested a high level HOST/TERMINAL environment to use with the 6.x system. We have created such an program, called FourTALK, which will be available shortly. This package includes a complete HOST system that works in conjunction with the TERMINAL portion of the package. The TERMINAL portion is set up look like a Radio Shack DT-1, emulating an ADDS-25 terminal. Full cursor positioning, reverse video, etc. are supported. Another feature of this program is the availability of all the features of "COMM", to make the emulation of the ADDS much more powerful than just a terminal. Full upload and download are available for file transfer as well as spooled printer support and all the other high level functions of COMM while maintaining an ADDS-25 protocol for the handling of the video and keyboard.

The HOST will allow you to remotely operate your Model 4, with full cursor positioning in the ADDS-25 mode. With this package you can use a DT-1 as a remote work station to your Model 4 or another Model 4 as an ADDS-25 terminal, for use with the Model 12/16 under XENIX, or as a terminal to a Model 4 running the HOST portion of the package.

The ever popular WordStar word processing system is now available for use on LDOS 5.1. It is supplied on smal-LDOS, and is available for the Models 1 and 3 (or 4, in the 3 mode). The special introductory price of just $249 plus $5 shipping and handling will be in effect until December 31, 1983. The regular price will be $395. If you always wanted the power of WordStar on your TRS-8Ø, then NOW IS THE TIME. Order W-37-Ø1Ø for Model 1, and W-37-Ø2Ø for Model 3 and MAX-8Ø. We are working on the popular MailMerge option, and it should be available shortly.

The LDOS QUARTERLY ANTHOLOGY, LSI Catalog Number L-49-11Ø, which contains all of Volume #1 of the LDOS QUARTERLY is now available for just $19 plus $3 for shipping and handling. This is a complete reprint of all the issues in volume #1, just as they were originally sent to our LDOS users. The anthology is provided "three hole drilled", ready for placement in any standard three ring binder.

LS-TBA, LS-FEDII, LS-FM and LS-HELP for the Model 4 running under TRSDOS 6.x (LDOS 6.x) are now being shipped.

DiskDISK is a brand-new package that we should be be shipping by the time you read this. With this package, you can create a file on a disk that appears to the system to be another disk drive! That's right, if you have a double-sided 4Ø or 8Ø track drive, or a hard drive, you can create "logical drive partitions" as files on the physical drive. ALL system functions are available as though this file were an actual physical drive. Now you can have a disk within a disk. These DiskDISKs can be used as though they are just another disk drive. Even mirror-image backups function in a normally.

This concept is much more versatile and powerful than the use of "partitioned data set" type files. There are no restrictions on reading and writing to these DiskDISK files as there are with other concepts. You can now partition your hard drive as "logical" floppy drives. This keeps groups of related files together, and makes backup and file maintenance a breeze. DiskDISK is just $99 plus $3 shipping and handling. DiskDISK, LSI Catalog Number L-35-211 is for LDOS 5.1 systems, and LS-DiskDISK (L-35-212) is for LDOS/TRSDOS 6.x. For hard drive users this is a "MUST HAVE" item.

Another NEW PACKAGE for the Model 4 6.x system is LS-QFB/COMP 6.x. This is the new high speed "Quick Format and Backup" utility and our popular disk and file compare utility combined into one package for just $39 plus $3 shipping and handling. With COMP you can compare two FILES or two DISKS and find out about every byte that does not match between the two. Output is provided to the video or optionally to your printer. With QFB you can create byte for byte duplicates in about half the time that format and backup would do the same job. This neat package will make your Model 4 even more pleasant to use. Order LSI Catalog Number L-32-Ø1Ø. NOTE: QFB will NOT create copies of backup limited or protected disks.

THE ELECTRONIC INBASKET

by Gordon B. Thompson, 5 Bay Hill Ridge, Stittsville, Ontario, Canada KØA 3GØ
Telephones: Voice (613) 836-3554 and Electronic Inbasket (613) 836-5578

The Electronic Inbasket is a simple BASIC program that makes a Model I or III act as a message collector. It sits there, on the end of a modem, and waits for any incoming calls. Upon the detection of carrier by the modem, the program responds with a suitable banner, and then collects the message and displays it on the screen, character by character. When the caller disconnects and the carrier vanishes, the message is appended to a disk file, all formatted for printing out in your favourite word processor.

This program makes use of the REFLEX filter, which appeared in "The Communicating Micro" in the LDOS Quarterly for October 1982. With REFLEX, the INKEY$ instruction can capture the data coming in over the modem. For this service, input must be character by character with a check for carrier between each character to make the program safe from hanging due to not receiving a carriage return character or logoff from the caller.

The use of REFLEX also allows the keyboard of the TRS-8Ø dedicated to this service to be used while a message is being received, should one wish to communicate with the sender in real time. No command is necessary, just start typing, and the local machine running this program will act just like a half duplex terminal. The characters typed this way are sent to the distant party by the REFLEX filter. The Electronic Inbasket operates in half duplex, and does not echo the incoming data stream. If it were to echo the incoming stream, characters entered at the local keyboard would get sent twice, once by REFLEX and once by the echo routine. The disk routine can be activated from the local keyboard by typing a "EOM" or CONTROL D, <SHIFT DWN ARROW><D>. This will not cause the communications link to sever, however, as no break signal is sent to the caller's modem.

Output to the communication line makes use of the device independent features of LDOS. An OPEN statement is used to open the file "*SI", and all outgoing data is sent via the PRINT # command to that "file". These two tricks, the use of REFLEX for incoming data, and the OPEN"O",Ø,"*SI" statement, greatly simplify the handling of the communications line in this particular case.

Because of differing screen formats, and other vagaries, the program is designed to put its major output onto a disk along with the requisite print control statements so the output can be formatted by a word processing package. My favourite word processor is NEWSCRIPT, so the ELECTRONIC INBASKET contains NEWSCRIPT's control words. Other word processors, like SCRIPSIT, would require other control sequences.

For stability purposes, the program only opens the disk file at the end of a message, and then immediately closes it again, clears all variables, and returns to its waiting state with a clean slate. Not all messages are short, or simple.

In normal use, a second telephone line is needed to run this service. A Model I, a single disk drive and an auto-answer modem complete the hardware requirements. The communication specification will be 3ØØ baud, even parity and half duplex if the RS232 driver is set as indicated.

The new Model 1ØØ is simply great at sending messages to the older Models I or III running this ELECTRONIC INBASKET program, and the 1ØØ's TELECOM specification should be set to M7E1D for this purpose. However, not until we learn how to make the Model 1ØØ detect the presence of carrier, as opposed to mere incoming data, and how to receive that data character by character with an interleaving carrier detect routine, can the 1ØØ replace its older brothers in this Electronic Inbasket service.

```
1ØØ ' *  *  * Electronic Inbasket *  *  *
11Ø ' Note:
12Ø ' Requires LDOS with the following configuration:
```

```
130 '        SET *KI KI(TYPE)
140 '        SET *SI RS232R(DTR=Y,RTS=Y) for Model I, or
150 '        SET *SI RS232T(DTR=Y,RTS=Y) for Model III.
160 '        FILTER *KI REFLEX/FLT
170 '  * * *  Communications Specification:  * * *
180 '      300 Baud, Even Parity, Half Duplex.
190 '
200 '  REFLEX/FLT uses different locations for storing its
210 '  flag in Models I and III.  This routine determines
220 '  which model, and then sets the REFLEX flag to on.
230 M=PEEK(&H125):'  See if it is a Model I or III.
240 IFM=73 POKE&H4413,1:GOTO270:'  It's a three.
250 POKE&H401A,1:'  It's a one.
260 '     Setup routine.
270 CLS:CLEAR 5000:DIM C$(100):PRINT"Started at "+TIME$
280 PRINT"Waiting...":OPEN"O",1,"*SI":' Reentry point.
290 '     Main or Waiting Routine.
300 POKE&H3FFF,32:FORN=1TO50:NEXT:GOSUB460:IFB=0GOTO330
310 POKE &H3FFF,&H2A:FORN=1TO50:NEXT:GOTO300
320 '       Opening Banner Routine.
330 FORN=1TO50:NEXTN:PRINT#1,CHR$(13)+CHR$(10)+"* * THE ELECTRONIC INBASKET *
*"+CHR$(10)
340 PRINT#1,"Date and time are: "+TIME$+CHR$(10):PRINT#1,"Please leave your message and
then disconnect."+CHR$(10)
350 PRINTTIME$:'  Screen print of message arrival time.
360 PRINT#1,CHR$(13)+CHR$(10)+">";:PRINT">";
370 '          Get Character Routine.
380 '  With REFLEX on, it can come from either *KI or *SI.
390 A$=INKEY$:GOSUB460:IFB<>0THENGOTO480ELSEIFA$=""GOTO390
400 PRINTA$;:B$=B$+A$:L=LEN(B$):IFA$=CHR$(8)B$=LEFT$(B$,L-2)
410 IFA$=CHR$(04)GOTO480ELSEIFA$<>CHR$(13)GOTO390
420 '      Line End Routine.
430 C$(K)=B$+CHR$(13):K=K+1
440 A$="":B$="":GOTO360:' Go back for next message line.
450 '         Modem Carrier Detect Subroutine.
460 A=INP(&HE8):B=AAND(&H20):RETURN
470 '     End of Message and Disk Routine.
480 OPEN"E",2,"MESSAGE/TXT":PRINT#2,TIME$
490 '          Word Processor commands are for
500 '     Prosoft's NEWSCRIPT.  Change them to suit.
510 PRINT#2,".br":' NEWSCRIPT control word.
520 FORL=0TOK:PRINT#2,C$(L);:NEXTL
530 PRINT#2,".sk 2":'  NEWSCRIPT Control Word.
540 PRINT:' Insert a blank line on screen display.
550 CLOSE:CLEAR5000:DIMC$(100):GOTO280
```

FAST GRAPHICS FOR 'LC'

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

One of the fine features of the 'LC' is its library of graphics routines. In addition to the set, reset and test functions familiar in BASIC, there are routines for line, box and circle drawing. These routines are primarily the work of Karl Hessinger and Karl is to be commended for a fine job. The line drawing (and consequently the box routine) is very fast, but the circle routine is slowed by its use of calls to the floating point routines in ROM. This article introduces a circle routine which replaced the LC library circle routine in version 1.1. This routine is approximately twenty times faster than the current one. Note that there are no floating point math or trancendental functions used. Also presented is a routine that will fill in an area that is continuously bounded. The FILL routine is non-recursive which means that it does not use calls to itself. It is very regular in the manner in which it fills an area and is, therefore, more pleasing than the typical recursive approach.

These routines were adapted from algorithms presented in the book "Fundamentals of Interactive Computer Graphics" by Foley and Van Dam (published by Addison Wesley, 1982). This book is a must for anyone interested in computer graphics. Most algorithms in the book are demonstrated in Pascal which is easy to convert to 'C'.

The new graphics functions that are listed here may be added to a user library by following the instructions in the LC manual.

Circle Routine

```
/* CIRCLE - fast circle plotting by Scott A. Loomer
   Adapted from Fundamentals of Interactive Computer Graphics by Foley and Van Dam
   This routine will plot circles using a fast, non-floating point algorithm. Syntax
   and return codes are identical to the circle function in the LC IN/LIB.
*/
#option INLIB

circle(funcod,x1,y1,r1)
int funcod,x1,y1,r1;
{  int d,dx,dy;
   if (funcod > 1 || funcod < Ø) return(-3);
   dx=Ø;
   dy=r1;
   d=3-2*dy;
   while (dx<dy)
   {  circlepoints(funcod,x1,y1,dx,dy);
      if (d<Ø) d=d+4*dx+6;
      else
      {  d=d+4*(dx-dy)+1Ø;
         dy--;
      }
      dx++;
   }
   if (dx==dy) circlepoints(funcod,x1,y1,dx,dy);
   d=r1/2;
   if (((x1+r1)>127)||((x1-r1)<Ø)||((y1+d)>47)||((y1-d)<Ø))
      return(-1);
   else return(funcod);
}

circlepoints(funcod,x,y,dx,dy)
int funcod,x,y,dx,dy;
{  int tdy,ty;
   tdy=dy/2;
   pixel(funcod,x+dx,y+tdy);
   pixel(funcod,x+dx,y-tdy);
   pixel(funcod,x-dx,y-tdy);
   pixel(funcod,x-dx,y+tdy);
   tdy=dx/2;
   pixel(funcod,x+dy,y+tdy);
   pixel(funcod,x+dy,y-tdy);
   pixel(funcod,x-dy,y-tdy);
   pixel(funcod,x-dy,y+tdy);
   return;
}
```

Non-recursive Fill Routine

```
/* Non-recursive Fill Algorithm - by Scott A. Loomer
   Adapted from Fundamentals of Interactive Computer Graphics by Foley and Van Dam
   When given an x,y coordinate in an area bounded by contiguous border or the screen
   edge, this algorithm will change all internal pixels to the border value.
   calling syntax: fill(funcod,x,y) where:
      funcod - is the value of the fill character, which must
         also be the value of the border character, Ø = reset,
         1 = set
      x,y    - is a coordinate in the area (must not be equal
         to border char)
*/

#option INLIB

int top,sx[128],sy[128];

fill(funcod,x,y)
char funcod;
int x,y;
{  int idx,max,min,xl,yl;
   top=1;
   while (test(funcod,++x,y))
   {;}
   push(--x,y);
   while (pop(&x,&y))
   {  max=x;
      do pixel(funcod,x,y);
      while (test(funcod,--x,y));
      min=x++;
      for (idx=1;idx>=-1;idx-=2)
      {  yl=y+idx;
         xl=max;
         while (xl > min)
         {  if (test(funcod,xl,yl))
            {  while(test(funcod,++xl,yl))
                 {;}
                 push(--xl,yl);
                 while(test(funcod,--xl,yl))
                 {;}
            }
            --xl;
         }
      }
   }
   return;
}

test(bord_char,x,y)
char bord_char;
int x,y;
{  int ret;
   return(((ret=point(x,y))!=bord_char)&&(ret!=-1));
}

push(x,y)
int x,y;
{  sx[top]=x;
   sy[top++]=y;
}

pop(x,y)
int *x,*y;
{  *x=sx[--top];
   *y=sy[top];
   return(top);
}
```

Test Program

```
/* Test of the new CIRCLE and FILL routines
   Assumes that CIRCLE and FILL have been placed in a library called USER/LIB
*/
#option INLIB
#option USERLIB

main()
{  int x,y;
   pmode(1);
   fill(1,Ø,Ø);
   circle(Ø,Ø,Ø,24);
   fill(Ø,5,5);
   circle(Ø,96,24,24);
   circle(Ø,96,24,12);
   fill(Ø,96,15);
   box(Ø,24,24,36,36);
   fill(Ø,Ø,47);
   fill(1,Ø,47);
   exit(Ø);
}
```

Using INTERRUPTS and SVCs in FORTRAN

by J. Gary Bender, PO Box 773, Los Alamos, New Mexico 87544

"WHY?" Why would anyone want to write an interrupt routine in Fortran... after all, interrupt level stuff is certainly the realm of assembly language. Those were exactly my initial thoughts. On the other hand, the more I can get done in a high level language is usually the more that I get done ... period. So I was motivated to try it.

By using SVC calls from Fortran, you can do many things that may surprise you. You can, for example, avoid the FORLIB I/O routines. The subject of this article is how to have the LDOS interrupt handler execute a Fortran subroutine. A few other examples of SVC use from Fortran are included.

SVC's are not really required. A direct call to @ADTSK and @RMTSK would work just as well ... for the moment. With all the potential new LDOS machines on the horizon, it is time to make the SVC table a normal part of your system CONFIG. The SVC's allow you to get closer to the operating system than Fortran normally permits and still remain compatible with the Model I, III, or Max 8Ø.

There are occasions when you may need, or at least could use, an interrupt driven subroutine in an application level program. The "dead man" timer in this example is one of the more useful, yet simple, capabilities. Besides the timed input shown here, it can permit you to write programs with uncluttered logic. I use it in an automatic message exchange program that dials and communicates with a remote host. Every time an expected "event" occurs, such as receiving a character, the timer is reset. While waiting for an event to occur, the timer is included in the loop. If it counts down before something else happens, something went wrong and I can take some action to abort the exchange. Most of the program does not have to be concerned with the multitude of things that can go wrong during the exchange. Also, I am not confined to the primary loop. I can wander off and do anything I want and have control over whether or not the deadman timer resets, stops, or continues to countdown. The conventional Fortran technique would require the use of do-loops -- all over the place -- to insure the program did not make an 8 hour long distance phone call waiting for a crashed host to respond.

Let's look at how this works from Fortran. The three elements of an interrupt procedure are: the routine itself, a Task Control Block that points to the routine, and a facility to install/remove the routine in the LDOS interrupt task table.

The routine that does "something" upon interrupt is a standard Fortran SUBROUTINE ending with a RETURN statement. It can call other Fortran subprograms. It should NEVER access any routines in the "mainline" program. (Remember, you do not know when it will execute. If it were to change a local value in the mainline, there is no telling what could happen.) The subroutine is included with your other subroutines, but it is never CALLed. It will execute independent of the mainline program, however, while it is installed in the LDOS interrupt task table.

Since it is not directly called by the Fortran program, arguments cannot be used. That means it must communicate with the rest of the program through COMMON. As long as the routine is included with one of your Fortran source files, LINK-8Ø will load it, even if it is not referenced. Actually, it is referenced as an EXTERNAL.

The TCB is easier to handle than may be apparent at first. All the TCB is is an INTEGER*2 Fortran variable that contains the address of the interrupt subroutine. The only problem is that you have no way of knowing where the subroutine will ultimately reside in memory. Fear not, it is quite easy to determine the address of a subroutine at runtime. Due to the calling conventions used by Microsoft FORTRAN-8Ø, a SUBROUTINE LOC (ARG) that does nothing but RETURN will return the address of the ARG argument (regardless of ARG's type) if it is accessed as an INTEGER*2 FUNCTION.

Shall I try that one again? This is what happens: a function reference such as IADDR = LOC(DEADMN) puts the address of DEADMN in the HL register ... the first argument's ADDRESS is always passed in the HL register by convention. LOC itself does nothing at all, it just returns. Also by convention, INTEGER*2 functions return their VALUE in the HL register. Since the calling Fortran program thinks LOC is an integer function, it uses the value in HL as the function value. The value in HL is the address the caller just put there as the argument. It is a little roundabout, but it works.

To install and remove the interrupt subroutine, you need access to the @ADTSK and @RMTSK SVCs. The interlude routine must be written in assembler, since Fortran cannot directly call the routines with the proper register settings.

The following program demonstrates two other SVC calls. @KBD, similar to the INKEY$ function in Basic, and @DSP, which displays one character on the screen. The example program checks for a character from the keyboard. If nothing happens in about 16 seconds, it times out. The deadman counts down in increments of 4 seconds because both the Model I and Model III interrupts are very close to "even" if you count interrupts for 4 seconds. The Model I has 2Ø "ticks" and the Model III has about 15. For most deadman type requirements you do not need exact timing, but if you tell the user there is a 6Ø second timeout, don't zap him in 45 seconds just because he is on a Model I.

A precaution must be observed when using interrupts: YOU MUST REMOVE THE TASK BEFORE EXITING THE FORTRAN PROGRAM! You will almost certainly have a system crash if you leave the Fortran program while the interrupt task is still running. If the Fortran program bombs out without first removing the task, you should re-boot the system. If the program is susceptible to abnormal, uncontrolled termination, you should install and remove the interrupt task as needed.

Let me briefly discuss the individual routines in the example. I'll start with the assembly language SVC interlude routines. You must use MACRO-8Ø to assemble the routines in order to have a LINK-8Ø formatted relocatable module.

DSP$ loads a character from a Fortran variable into the C register and calls the @DSP SVC. It displays one character at a time at the current cursor position and advances the cursor over one position. KDB$ (or INKEY$) does a little more work since LDOS returns more information. My objective when writing KBD$ was to make all the information easy to use (or ignore) at the Fortran level. Besides the character itself, which is normally all that is needed, the first 4 bits of the INFO argument tell you everything you are apt to want to know about the input character. KDB$ does not wait for a key. If there was no key depressed, INFO and ICHAR will both be zero. The routine is set up so you can call it as a subroutine, an INTEGER*1 function, or an INTEGER*2 function. Either of the function calls will return the character (or zero).

ADTSK$ and RMTSK$ are the routines that install or remove the interrupt driven subroutine in LDOS' interrupt table. ADTSK$ needs the address of the subroutine to be executed by the interrupt handler and the slot number to assign it to. See the LDOS manual for slot assignments. The example uses slot Ø, a low priority slot that executes 5 times per second on the Model I. RMTSK$ only needs the slot number.

The Fortran subroutines include a couple additional goodies. CLS clears the screen using DSP$ and control characters. DISPL is a great time saver for me. It will display a string enclosed in any pair of delimiters. An array can be used for the string, but it must still include the delimiters. When using a string constant to call DISPL, as in the example, remember to enclose everything, including the string delimiters, between single quote marks. The single quotes tell the compiler it is a string, they are not part of the string itself.

TINP$R is the Timed INPut subroutine. The version I normally use returns a Ratfor string, which is where the "$R" comes from. For this example there is no need to get into an alternate string convention. TINP$R has six arguments which are documented before the CALL in the Main program. The routine displays a non-blinking cursor and loops calling KBD$. When KBD$ indicates that a character was depressed, TINP$R first checks that it was not an ENTER or BREAK key, puts the character into the INBUF array, echos the character to the screen, and moves the underscore cursor over.

In a non-timed routine, this would go on until the maximum characters were typed, or an ENTER or BREAK. The timed routine adds the deadman counter. When TINP$R is called, it starts the DEADMN interrupt routine by calling SETDM (.TRUE.,MAXSEC), i.e. turn the deadman on and set it for MAXSEC. Each time through the loop it checks for a timeout. Each time a character is typed, the deadman counter is reset to MAXSEC. When the input loop is exited for any reason, the deadman is shut off. This also removes the task, which is safer than waiting until the program ends.

DEADMN is just sitting there in the middle of the program and is never called by the mainline program. When installed as an interrupt task, however, it will execute about 5 times per second. All that it does is count down 2Ø ticks (or 15) and then subtract 4 seconds from the main counter DMKONT. If everything gets counted down to zero, it stops decrementing. It is up to the mainline program to check for a timeout by examining the value in COMMON.

There is no reason that DEADMN cannot maintain several counters or call other subroutines. Just avoid using any subroutines in the "other program." Think of the DEADMN task as a separate program running concurrently with the mainline and sharing some of its memory.

SETDM will install or remove the DEADMN task depending on the truth of the first argument (.TRUE. == ON) and initializes/resets the counter to the number of seconds specified. SETDM is part of the mainline program. It controls DEADMN by inserting values into COMMON and by making the actual calls to add or remove the task. It does not change the deadman counter when removing the task. That lets you remove the task before you check for a timeout. Notice that all that is needed to get the address of DEADMN are the statements EXTERNAL DEADMN and DMTCB = LOC (DEADMN). In a critical application, it would be advisable to disable interrupts while SETDM is setting values since a counter may be decremented midway through the setting process. For normal applications it should not matter.

TIMEDO is an easy way to check for a timeout by using a logical function call. NTIMED checks for a NOT timed out condition. It cheats a little by just returning .NOT. TIMEDO.

LOC is just what I promised before. A very handy "do-nothing" routine. I also use it to have my Fortran programs use the @PARAM SVC --- command line options with LDOS doing all the work!

All the Main program does is set up the demo, call TINP$R and tell you what happened.

Before using this program, you MUST have the SVC table installed: SYSTEM (SVC) is all that is necessary. If you use the names TIMEDI/FOR and TIMESUBS/MAC for your source code, then the following will compile/assemble the program:

```
F8Ø TIMEDI=TIMEDI
M8Ø TIMESUBS=TIMESUBS
```

Link the program with:

```
L8Ø TIMEDI,TIMESUBS,TIMEDI-N-E
```

Do not be concerned if you have $IOERR and/or $LUNTB show up as unsatisfied references after the link and load. The warning is extraneous and caused by FORLIB loading blocks of code rather than individual routines.

Take a good look at the size of the program ... about 16ØØ bytes! You've written a Fortran program that does I/O and used less than 6K! Maybe there IS something worthwhile in using SVCs from Fortran ....

When typing the following listing, do not include the "/*" comments. F8Ø does not permit that style of comment.

```
C  TIMEDI/FOR   Demonstrate Interrupt driven timed input
C  JG Bender   24 Jul 83
C
C
      PROGRAM   TIMEDI
C
      INTEGER*2 ACTCHR
      BYTE      SCRATC(64)
      LOGICAL   BROKE, TIMOUT
      LOGICAL   TINP$R
C
      INTEGER*2        DMKONT, DMTPSS, DMSECS, DMTICS
      COMMON /DEADCM/ DMKONT, DMTPSS, DMSECS, DMTICS
C
      DATA DMKONT/Ø/, DMSECS/4/, DMTICS/Ø/
C
C   set the DeadMan increment counter (low priority ticks / 4 secs)
C   ( for a Model III, set  DMTPSS = 15 )
C
      DMTPSS = 2Ø
C
C   Announce the program
C
      CALL CLS
      CALL DSP$ (X'ØD')            /* a carriage return
      CALL DISPL ('/ You have 16 seconds to type something: /')
C
C   In the following TINP$R call:
C      SCRATC  <= buffer to receive input characters
C      1       => maximum number of characters to accept.
C      16      => number of seconds to wait
C      ACTCHR  <= number of characters returned
C      BROKE   <= .TRUE. if user hit .BREAK. key
C      TIMOUT  <= .TRUE. if user timeout during input
```

```
      CALL TINP$R (SCRATC,1,16,ACTCHR,BROKE,TIMOUT)
          IF (TIMOUT)  GOTO 7ØØ
          IF (BROKE)  GOTO 8ØØ
C
      CALL DSP$ (X'ØD')
      CALL DISPL ('/You made it!/')
      GOTO 99Ø
C
7ØØ   CALL DSP$ (X'ØD')
      CALL DISPL ('/You took too long !/')
      GOTO 99Ø
C
8ØØ   CALL DSP$ (X'ØD')
      CALL DISPL ('/You QUIT/')
      GOTO 99Ø
C
99Ø   CONTINUE
      END
C
C   ===================================================
C   ===================================================
C
C   SUBROUTINES
C
C
C   Contents:
C       CLS         Clear screen
C       DISPL       Display a delimited string, no .CR.
C       TINP$R      Timed input
C       DEADMN      Dead Man countdown timer
C       SETDM       Set DeadMan counter ON, and initialize count
C       TIMEDO      Check if deadman counted down to Ø
C       NTIMED      Check if deadman DID NOT timeout(== TIMEDO-)
C
C
C   ===============================================
C       CLS         Clear Screen
C   -----------------------------------------------
C
      SUBROUTINE CLS
C                       Assume Model I/III control chars
C
      CALL DSP$ (X'1C')          /*  home cursor
      CALL DSP$ (X'1F')          /*  clear to end-of-frame
      RETURN
      END
C
C   ===============================================
C       DISPL       Display a delimited string on screen
C   -----------------------------------------------
C
      SUBROUTINE DISPL (DSTRNG)
C                                 all strings will be <= 127 chars
C                                 no carriage return for this routine
      BYTE      DSTRNG(1), DELIM
      INTEGER*1 I
C
C   the first character of the string is the delimiter
C
      DELIM = DSTRNG(1)
C
C    the following is a 'FOR' loop in RATFOR
C
```

```
      I = X'Ø2'
23ØØØ IF (DSTRNG(I) .EQ. DELIM .OR. I .GE. X'7E')  GOTO 23ØØ2
         CALL DSP$ (DSTRNG(I))
         I = I + X'Ø1'
         GOTO 23ØØØ
C
23ØØ2 CONTINUE
      RETURN
      END
C   ======================================================
C     TINP$R    Timed Input
C   ------------------------------------------------------
C
      LOGICAL FUNCTION TINP$R (INBUF,MAXCHR,MAXSEC,ACTCHR,BROKE,TIMOUT)
C
C             .TRUE. if input recvd, .FALSE. if timed out w/ no data
C
      BYTE        INBUF(1), ICHAR
      LOGICAL     BROKE, TIMOUT, NTIMED, TIMEDO
      BYTE        KBLANK, KBSPAC
      INTEGER*1 MAXCHR, I, IONE, I126, IMAX
      INTEGER*2 MAXSEC, ACTCHR, INFO
C
      DATA IONE/X'Ø1'/, I126/X'7E'/
      DATA KBLANK/X'2Ø'/, KBSPAC/X'18'/
C
C       install the deadman interrupt routine
      CALL SETDM (.TRUE.,MAXSEC)
      BROKE  = .FALSE.
      TINP$R = .FALSE.
      TIMOUT = .FALSE.
C
C   range check
      IMAX = MAXCHR
      IF (MAXCHR .LT. IONE)  IMAX = I126
C
C   display an underscore and a backspace
C
      CALL DSP$ ('_')
      CALL DSP$ (KBSPAC)
C
      I = X'ØØ'
23ØØ5 IF (TIMEDO(Ø) .OR. (I .GE. IMAX)) GOTO 23Ø1Ø  /* end loop
         CALL KBD$ (ICHAR,INFO)
         IF (INFO .EQ. Ø)  GOTO 23ØØ9     /* no char --> loop
            TINP$R = .TRUE.
            IF (ICHAR .EQ. X'ØD' .OR. ICHAR .EQ. X'Ø1')  GOTO 23ØØ6
C                          .CR.                 .BREAK.
               I = I + IONE             /*  next char in string
               INBUF(I) = ICHAR         /*  put the chr into string
               GOTO 23ØØ8               /*  --> loop
C
23ØØ6       IF (ICHAR .EQ. X'Ø1')  BROKE = .TRUE.
            GOTO 23Ø1Ø                  /*  end looping -->
C
C           reset deadman and display the character just typed
23ØØ8       CALL SETDM (.TRUE.,MAXSEC)
            CALL DSP$ (ICHAR)
            CALL DSP$ ('_')
            CALL DSP$ (KBSPAC)
C        bottom of loop
23ØØ9    GOTO 23ØØ5
C
```

```
C     exit loop to here
23Ø10 CALL SETDM (.FALSE.,MAXSEC)          /* kill the deadman
      IF(TIMEDO(Ø)) TIMOUT = .TRUE.
      ACTCHR = I
      CALL DSP$ (KBLANK)                   /* remove the cursor
      CALL DSP$ (KBSPAC)
      RETURN
      END
C
C   ==================================================
C      DEADMN      Dead Man, interrupt timer, count down
C   --------------------------------------------------
C
      SUBROUTINE DEADMN
C                  ONLY called by the Interrupt driver
C                  All set/reset of values must be done by
C                     calling routine
C                  Install in a low-priority task slot
C
      INTEGER*2 DMKONT    /*  number of seconds to countdown
      INTEGER*2 DMTPSS    /*  nr ticks before need decrement DMKONT
      INTEGER*2 DMSECS    /*  nr seconds per DMTPSS ticks
      INTEGER*2 DMTICS    /*  current tick countdown
C
      COMMON /DEADCM/ DMKONT, DMTPSS, DMSECS, DMTICS
C
C     countdown a tick for each call
      DMTICS = DMTICS - 1
      IF (DMTICS .GT. Ø)  GOTO 23Ø14  /*  still counting ticks -->
C       reset local countdown
        DMTICS = DMTPSS
        IF (DMKONT .LE. DMSECS)  GOTO 23Ø12    /*  TIMED OUT -->
C         decrement a chunk of seconds from timer
          DMKONT = DMKONT - DMSECS
          GOTO 23Ø14
C
23Ø12   DMKONT = Ø
C
23Ø14 CONTINUE
      RETURN
      END
C
C   ==================================================
C     SETDM       Set up DeadMan counter
C   --------------------------------------------------
C
      SUBROUTINE SETDM (ONOFF,SECS)
C                                # Install/remove D/M in slot Ø
C
      INTEGER*2 SECS, LOC
      INTEGER*2 DMTCB     /*  Task Control Block for DeadMan
      INTEGER*1 ISLOT     /*  task slot to use
      LOGICAL   ONOFF     /*  caller instruction to turn
C                             the deadman ON or OFF
      LOGICAL   DMISON    /*  flag if DeadMan IS ON
C
C     it is necessary that this routine have the following EXTERNAL:
      EXTERNAL  DEADMN
C
      INTEGER*2     DMKONT, DMTPSS, DMSECS, DMTICS
      COMMON /DEADCM/ DMKONT, DMTPSS, DMSECS, DMTICS
C
      DATA DMISON/.FALSE./, ISLOT/X'ØØ'/
```

```
C
      IF (.NOT.(ONOFF))  GOTO 23Ø16      /* .T. == turn it on
C        reset counters
         DMKONT = SECS
         DMTICS = DMTPSS
         IF (DMISON)  GOTO 23Ø15     /*  task already running ?
            DMTCB = LOC (DEADMN)     /*     no, install it
            CALL ADTSK$ (DMTCB,ISLOT)
            DMISON = .TRUE.
23Ø15    GOTO 23Ø18
C
C    ONOFF is .F., shutdown the deadman ...
C       DO NOT reset the DMKONT.  A programmer may remove the
C       interrupt task before checking for a timeout.
C
23Ø16 IF (.NOT.(DMISON)) GOTO 23Ø18              /* is it running ?
         CALL RMTSK$ (ISLOT)                     /*  yes, remove it
         DMISON = .FALSE.
C
23Ø18 CONTINUE
      RETURN
      END
C
C   ===================================================
C       TIMEDO      Check if DEADMN timed out
C   ===================================================
C
      LOGICAL FUNCTION TIMEDO (IDUMMY)
C
      INTEGER*2      DMKONT, DMTPSS, DMSECS, DMTICS
      COMMON /DEADCM/ DMKONT, DMTPSS, DMSECS, DMTICS
C
      TIMEDO = .FALSE.
      IF (DMKONT .LE. Ø)  TIMEDO = .TRUE.
      RETURN
      END
C
C   ===================================================
C       NTIMED      See if DEADMN did NOT timeout
C   ---------------------------------------------------
C
      LOGICAL FUNCTION NTIMED (IDUMMY)
C
      LOGICAL TIMEDO
      NTIMED = (.NOT. TIMEDO (IDUMMY))
      RETURN
      END
C
C   ===================================================
C       LOC     return arg address as I*2 function
C   ---------------------------------------------------
C
      SUBROUTINE LOC
      RETURN
      END
C
C  this is the end of the Fortran code.


; File: TIMESUBS/mac
;
;       ******************************************
```

```
;
; DSP$/mac
; JGB    25 Feb 83
;
S_V_C   EQU     X'28'   ; RST vector for SVC call
DSP_    EQU     2       ; @DSP  SVC
;
;       FORTRAN usage:
;
;       CALL DSP$ (char)
;
;               char => FORTRAN variable with character to
;                       send in low-order byte.
;
DSP$::  LD      C,(HL)
        LD      A,DSP_
        RST     S_V_C
        RET
;
;       *****************************************
;
; KBD$/mac
; JGB   14 Feb 83
;
KBD_    EQU     8       ; @KBD  SVC
;
;       FORTRAN usage:          ( BYTE function )
;                       ( may be typed BYTE or INTEGER*2 )
;
;       CALL KBD$ (ICHAR,INFO)          INKEY$ == synonym
; -or-  JCHAR = INKEY$ (ICHAR,INFO)
;       (byte)          {HL}  {DE}
;       Returns K/B char WITHOUT waiting.
;
;       (byte)  ICHAR  <= Ø or byte from K/B  -- FORTRAN variable
;       (I*2)   INFO   <= Ø if no key pressed -- FORTRAN variable
;                bit:     Ø if key pressed
;                bit:     1 if Control-key also down
;                bit:     2 if CLEAR key also down
;                bit:     3 if char == BREAK
;       Note high bit of ICHAR will be on if CLEAR key was
;            also down
;
KBD$::  NOP             ; see LDOS manual for @KBD
INKEY$::                ; synonym
        PUSH    DE      ; ->info
        PUSH    HL      ; ->char variable
        LD      A,KBD_
        RST     S_V_C
        POP     HL
        LD      (HL),A  ; return char in Low byte of arg.
        POP     HL      ; ->INFO
        LD      (HL),Ø  ; clear the INFO byte
        JR      Z,KBD4  ; there was NO character -->
        SET     Ø,(HL)
        JR      NC,KBD2 ; no Shift-Down-Arrow -->
        SET     1,(HL)
KBD2:   OR      A       ; clear flags
        BIT     7,A
        JR      Z,KBD3  ; no CLEAR key
        SET     2,(HL)
KBD3:   CP      X'Ø1'   ; == BREAK key ?
        JR      NZ,KBD4
        SET     3,(HL)
KBD4:   INC     HL      ; ->high order byte of info
        LD      (HL),Ø
        LD      H,Ø     ; return in {A} and {HL}
        LD      L,A     ;           [byte] [Int*2]
        RET
;
;       ******************************************
;
;
; ADTSK$/mac
; JGB   25 Feb 83
;
ADTSK_  EQU     29      ; @ADTSK  SVC
;
;       FORTRAN usage:
;
;       Call ADTSK$ (task_tcb, slot_no)
;
;               task_tcb => FORTRAN INTEGER*2 variable containing
;                           the ADDRESS of the subroutine to install
;                           in the LDOS interrupt task handler.
;               slot_no  => task slot number to use
;
ADTSK$::
        EX      DE,HL   ; tcb in DE
        LD      C,(HL)  ; slot number in C
        LD      A,ADTSK_
        RST     S_V_C
        RET
;
;       ******************************************
;
;
; RMTSK$/mac
; JGB   26 Feb 83
;
RMTSK_  EQU     3Ø      ; @RMTSK  SVC
;
;       FORTRAN usage:
;
;       CALL RMTSK$ (slot_no)
;
;               slot_no => FORTRAN INTEGER*1 or INTEGER*2 variable
;                          containing slot number to remove the
;                          task from.
;
RMTSK$::
        LD      C,(HL)  ; slot number into C
        LD      A,RMTSK_
        RST     S_V_C
        RET
        END
```

....ER....

by Earle Robinson, 3ØØ Grenola, Pacific Palisades, CA 9Ø272

As promised last time, here are some remarks about the 'in' operating system, UNIX. Since there appear to be articles, and new books appearing every day, I won't try to fully explain UNIX, except to very briefly tell you what it is.

UNIX is a multi-user and multi-task operating system which was originally developed for the Digital Equipment PDP-7 by Bell Labs engineers. Written in assembly, it was later re-written in a low level von Neumann language called C so it became somewhat more portable between different machines than it had been. The origins of UNIX, in 1969, are reflected in its somewhat difficult syntax. Remember that in those days teletypes were used as terminals and every letter was precious in a command. Consequently, to get a directory listing of files, you type 'ls' plus any of the various switches which are available.

Most other commands are equally confusing to the new user. And, if you think the LDOS manual is long, try the 2 volume UNIX Programmers' manuals! They total over 1ØØØ pages!

The UNIX system itself is noteworthy for three main characteristics: a hierarchical file structure, I/O redirection and Pipes. Under UNIX, the directory structure is like an upside down tree with branches. From the so-called root directory you may create one or more sub-directories, and each of these may have sub-directories, and each of those, etc. etc. As you can imagine, this will permit you to have as many files in your storage medium as memory will permit. Thus, with a hard drive, you are not obliged to partition at all; the sub-directories provide their own dynamic partition. As for I/O redirection, all LDOS users should be reasonably familiar with this feature; it is one of the bases of the operating system. The device concept as implemented by the author of LDOS' precursor, VTOS, further refined and developed by LSI, was undoubtedly drawn from UNIX itself. However, where UNIX really excels is in the use of pipes and filters. This permits the concurrent processing of several tasks. For example, you could have a file sorted, written to another file and finally printed out on your printer with a single command line expression. UNIX makes full use of memory and creates temporary files, erased at the end of the task processing, to accomplish this. As you can imagine, such intensive I/O using poor little 5 1/4" disk drives would be painfully slow. And, even with the use of hard disk, things can be slowed down a great deal, especially if there are several users accessing the drive at the same time. Further, the maximum memory on most 8 bit machines, 64 K, limits their ability to effectively use UNIX.

One further constraint with UNIX is that it requires several megabytes, that is several million bytes of accessible disk storage to use it and the many utilities which accompany it. As you may begin to imagine, UNIX is not a system that you are likely to ever run on your Model I, III, MAX-8Ø, or even the Model 4. These machines just do not have sufficient RAM to handle such a bulky system. Whatever computer you may use, a hard disk of at least 1Ø megabytes should be employed. Also, a minimum of 192 K of RAM is required; in fact, most systems need at least 512 K.

Most of us are also confused when UNIX is discussed because of the myriad versions and forms it may take. To begin with, would you imagine that System V was introduced after Version 7, and that there are still implementations using Version 6 and System III, which itself appeared after Version 7? And, then there is the quarrel about whether one should be using a true UNIX or a UNIX-compatible system. The latter are often cheaper for the user because there are no license fees collected by Bell and can be better supported with further extensions. Most implementations are for multi-user systems though some single-users ones are offered, too.

The final problem with UNIX in the micro world is that it is even less user-friendly than CP/M, LDOS or any other known system. For this reason, a 'shell' is often put around the operating system so that the user is faced with menus and sub-menus until he reaches the application he intends to run. UNIX may be a programmer's delight, but end users will require a full scale shell implementation to use computers employing it. At present, the Bourne shell appears the most widely used, though it still is confusing to many unsophisticated users.

Whatever its strengths and weaknesses, you are likely to hear and see a lot about UNIX in the next few years. And, if good enough shells are used, it may well survive as the standard operating environment on 16 bit machines. Whatever the system is, however, one thing is quite certain: operating systems, and most utilities, will be written in 'C' or another high level language. Assembly language will be less and less used except for narrow uses where speed must be improved. This is because maintenance of programs will be easier and portability obtained, not to mention the lower cost for software houses in writing and debugging.

I have been having a wonderful time with the new version of FED, called FED II. It has many new features and is faster. I only regret that I don't have such a program for my IBM PC, too. I have also had a great deal of fun using the latest version of Super Utility Plus V.3 on my MAX-8Ø. This is a terrifically useful utility program. The manual is well-written and quite clear. But, I suppose that most of you have it already.

Speaking of the IBM, you may be interested in some initial comments about the machine which is sweeping most of the competition into Chapter XI, or at least into some steep operating losses. Hardware-wise, the PC is superbly built. There were a few design flaws, however. There are too few slots in the mother board and the power supply is barely adequate. The XT version, which contains a hard drive, has rectified those problems. The keyboard is controversial. You either like it or detest it. As for the monochrome monitor, it is a beauty, the easiest to read and use that I have seen on a micro.

Software? Well, that is another story. Let's begin with the 2 principal operating systems offered, PC-DOS and CP/M-86. The former, called MS-DOS on IBM compatible machines, is a creation of Microsoft, and was originally very close to CP/M (ugh!). The latest version, 2.Ø, is already a big improvement and is approaching Xenix, little by little. There are hierarchical directories, I/O redirection, and Pipes and Filters. It is not as fast as LDOS, and in some ways more cumbersome. As for CP/M-86, it appears to have lost out in this market, partly because IBM pushed the Microsoft DOS more. Market studies show that DOS has over 95% of the market. CP/M-86 has some nice features, but it is more cumbersome to use, perhaps because it is still so much like CP/M, its antecedent. It was also reported to be full of bugs, at least in the initial releases.

Most of the original software offered for the PC was merely CP/M stuff which was cross compiled to the 8Ø88 microprocessor used in the IBM machines and its copies. Alas, this meant that the code was far from optimized and ran even slower than on 8 bit machines. However, since the market has grown so much, virtually everyone is rushing to introduce products. There are now literally dozens of word processing packages, several spread sheet programs, and much more. When you see the massive advertising for some of these programs you can understand that the stakes are high in the IBM world, and that large capital is required to obtain a market share. The two major magazines, PC Magazine, and PC World, are the thickness of a large city's telephone book. They are so full of ads that it is hard to find the editorial matter.

In the realm of utilities for the PC, there is still surprising little, certainly nothing like the products offered by LSI and others for the Radio Shack line. On the other hand, there is a wonderful choice in programming languages, nearly a dozen full implementations of C, a couple of Pascals, Lisp, Modula 2, and others.

Many people criticize the Intel chip used in the IBM and its so-called clones because it is not as powerful as the 16 bit offered by Motorola or National Semiconductor. They cite the need to use segmented addressing and the fact that IBM uses the 8Ø88 rather than the 8Ø86 and that the former uses only an 8 bit data bus which slows everything down. Naturally, the Tandy 16 & the Fortune 16/32 are faster and more elegant machines. But, there are probably a million or more IBM's and its clones already out there while I doubt that more than 1ØØ,ØØØ of all the Motorola based machines have been sold. What the IBM lacks today is multiuser and multitasking capabilities. This will come, probably when IBM uses a newer version than the 8Ø88 from Intel, and when Xenix or IBM's in house operating system is finally put on the market.

You may not know it yet, but you can now have a telex number and send and receive telexes without getting one of those awful teletypes, and without any outlay at all........if you already have a modem and half-way decent communications program. In fact, Lcomm will serve very nicely. Here's how. RCA will be glad to give you a telex number which will permit sending telexes direct to either RCA or Western Union terminals, send overseas telexes and telegrams. It is not necessary have a line and a computer tied up either. You have incoming messages routed to RCA's store and forward memory bank. Then, you merely have to dial an 8ØØ number to retrieve messages whenever you wish. And, you send telexes from your computer using an 8ØØ number, too. I do not recommend tying up a line & computer for another reason than the obvious one. To receive messages directly on-line your program must be designed to respond to a Ctrl-E sent from RCA (and Western Union uses the same), i.e. the ASCII ENQ character. You have one (1) second to return the answer back when requested by the Ctrl-E. Otherwise, the line is dropped immediately. I also find that it is very economical to compose messages off-line, using Scripsit, LED, or another text editing program of your choice, then to upload it when the recipient's terminal replies. If any of you are interested in further details, drop me a line.

A couple of people at LSI have made a bet whether I would ever do an article without mentioning printers or word-processing programs. So, I shall NOT mention I recently heard that Harv Pennington has written (himself!!) an IBM PC version of Electric Pencil. I shall also NOT mention that...........

**** Parity=Odd ****

by Tim Daneluik, 4927 N. Rockwell St., Chicago, IL 6Ø625

RUMORS DEPT.

Fall is upon us, and as always I have more products to look at than time to do it in! Not only is there more new software coming out every week, but a whole host of new machines are expected to be announced this quarter. Herewith is a list of completely UNSUBSTANTIATED rumors that I've heard, but my sources are impeccable (I know a janitor at the Tandy towers!):

RUMOR #1 - The IBM "PEANUT" is supposed to be using an INTEL IAPX 186 (8Ø186) as its main processor. This processor is a real power-house with built in programmable timers, DMA controller, and has true 16 bit capability. The rumor mill also has it that the PEANUT will come with 64 K of memory, 1 floppy drive, and a keyboard for under $8ØØ! If all this comes true, you can bet that PC sales will fall off, since about the only thing the older machine will offer is more expandability (i.e. the PC-XT for example). Personally, I hope IBM builds a little performance into the PEANUT, because the PC is way too slow to justify its $3ØØØ+ price tag. They certainly have the processor it takes to do this. The 8Ø86 family of machines isn't as elegant perhaps as a 68ØØØ or one of its derivatives, but the Intel machines are plenty powerful in their own right. The poor performance of the PC is more a function of the "hurry up and get it done" mentality of its designers, than it is the basic choice of processor.

RUMOR #2 - Tandy is supposedly working on a version of the Model 4 with a built-in 5 Meg. hard disk. That's no big surprise, and makes a lot of sense for the market the 4 is directed at. Given today's pricing structure, the machine ought to sell for about $3ØØØ - $35ØØ. I've been playing with a Model 4 on and off, and with a hard disk in it, I may even buy one myself!

RUMOR #3 - This one isn't so much rumor as it is a not yet released product. A replacement printed circuit board for the TRS-8Ø Model I will shortly be available which turns your trusty old 'I' into a Model III! EVERYTHING is built into this board including the disk controller and expansion edges, so you don't even need an expansion interface! The whole business fits inside the Model I keyboard housing, and uses the Model I power supply and video display. I've gotten a look at the thing, and it is VERY well built - probably better than the original Tandy Model III electronics. Hopefully, by the time the next issue of this magazine appears, I will have one here for review, and will be able to tell you more about it. The company producing this appears to be a real business, not a "maw and paw" garage operation, so it looks like this might be a big seller. By the way, the price should be under $4ØØ retail for this goodie!

LOBO, THE LX-8Ø, AND WHY YOU CAN'T RUN ALL YOUR SOFTWARE

The Model I is apparently far from dead! I've gotten several letters from people asking me to devote a whole column to the LX-8Ø. Unfortunately, there weren't THAT many letters. However, a few comments are in order. LOBO, the people who build the LX-8Ø, have always taken the position that their designs should outperform the Tandy hardware they replace. The MAX-8Ø computer, for example, has no rivals in performance or reliability in the 8-bit line of Radio Shack machines. (For that matter, the MAX-8Ø has NO 8-bit peers that I've found under about $4ØØØ-$5ØØØ!)

For those of you new to the TRS-8Ø, the LX-8Ø is a high performance interface unit designed for the Model I computer as a replacement for the infamous Expansion Interface. In many ways, the LX-8Ø had a lot to do with LDOS coming into being. LOBO designed the interface to have many features not normally present in a standard EI. They included double-density, 5" and 8" drive interfaces, two RS-232C ports, and a built-in power supply. In short, the LX-8Ø overcame every deficiency ever present in the Radio Shack interface. Unfortunately, the price paid in the design, was that the LX-8Ø was not hardware compatible with the EI. This meant that many of the device dependent portions of the operating system for the Model I had to be rewritten. LOBO decided to also offer a new DOS for their interface, and contracted someone to write LDOS. Unfortunately, that someone never finished the job....so, LOBO came to Galactic Software (now LSI) and contracted them to finish the job. In the process, LSI came into being, and eventually ended up owning LDOS. The rest, most of you know. Bill Schroeder and his bunch of "not-ready-for-prime-time programmers" went hog-wild and wrote LDOS not just for the LX-8Ø, but for the Model I and the new Model III as well!

The net result of all this is that the LX-8Ø/Model I system runs LDOS just fine, but some existing pieces of software won't work. This software falls into three general categories:

1) The software is "self-booting" and uses no operating system.
2) The software is written specifically for a DOS other than LDOS.
3) The software ignores the DOS and tries to do physical I/O (Input/Output) to the hardware itself.

Software in the first category usually consists of things like games, utilities (Super-Utility, for example), and almost all forms of protected media. The LX-8Ø has its own special "boot ROM" (Read Only Memory) that is used to initially load the operating system. This ROM is substantially different than the ROM used when booting a "standard" Radio Shack interface. This is because the LX-8Ø supports things like booting in double-density, booting from an 8" disk, or even booting from a hard disk, and these special procedures have to be implemented in the boot ROM. Self-booting media expects to use a standard boot procedure, and invariably fails to work when used in an LX-8Ø environment. Even if you could get the program to boot on the LOBO interface, it probably still wouldn't work. These programs typically use no operating system, and access the hardware directly. Since the "innards" of an LX-8Ø are different than the EI, many self-booting programs, especially those involving disk I/O, CRASH on an LX-8Ø.

Little needs to be said about programs in the second category. Programs specific to another DOS usually can be modified to run under LDOS, and therefore the LX-8Ø. However, this requires some ability on your part, and is not always a simple thing to do. Now that LDOS is an accepted standard within Radio Shack, hopefully this kind of software will cease to be written (or at least purchased!).

Those of you who read this column regularly (all four of you) will remember my soapbox some time ago on software which does its own physical I/O. To repeat, very rarely if EVER should applications software deal with the hardware itself. The LX-8Ø is a perfect example of why. So long as software uses LDOS to "talk" to hardware, the operating system is able to accommodate differences in the hardware itself. Once the application bypasses the DOS, there is no guarantee that it will be able to run on other TRS-8Ø compatible systems. For example, the printer port on the LX-8Ø returns slightly different status bits than a regular EI does, even though the printer interface on the LX-8Ø is still mapped to X'37E8'. An application which uses LDOS calls to print data works just fine on the LX-8Ø. An application which goes directly to X'37E8' may or may not work. Again, some patching may get this kind of software to work, but it is usually more trouble than it is worth.

Here then, is a short and by no means complete listing of software which will/will not work on the LX-8Ø:

WILL NOT WORK

Any DOS other than LDOS
Stand-Alone Machine Language Monitors
Almost all self-booting disks
Disk-drive timing programs
Disk-drive diagnostic programs

WILL WORK

SNAPP Extended BASIC
Microsoft EDTASM+ for disk
POSTMAN
discatER
ALCOR PASCAL
MACRO-MON (mostly, some minor problems)
LSI and MYSOSIS utilities/languages
LAZY-WRITER (mostly, some functions like directories, and RS232 won't work)
ELECTRIC-PENCIL 2.Ø (mostly, some functions don't work right, like getting directories)

SCRIPSIT w/LSI patches
MACRO-8Ø " "
FORTRAN-8Ø " "
BASCOM " "
ZCAT
POWER-MAIL
LDOS TOOL BOX

One final thing, you cannot "Un-Repair" an LDOS disk on an LX-8Ø so that the disk will be readable under TRS-DOS 2.3. This is because of the Floppy Disk Controller chip used in the interface. If any of you LX-8Ø owners out there have patches for software to make it LDOS/LX-8Ø compatible, please send it to me. I'll publish them here, for everyone's benefit - besides, you get your name in print that way!

MODEL 4 / TRSDOS 6.Ø CORNER

Although the Model 4 is relatively new, several pieces of software are already available for it. Logical Systems has both FM and FED in 6.Ø formats. (Don't forget LS-Technical Help 6.x! ed.) FM is a sophisticated file backup and purge utility which will be of special interest to those of you with hard disks (whenever Tandy gets around to putting the hard disk on the Model 4!). FEDII is the latest iteration of the most useful utility a machine language programmer can have. It is made to edit any sector, or any file on an LDOS disk directly. You can make changes in ASCII or hex, and many search features are also supported. FEDII lets you step through /CMD files by load module (forward and backward), and has an in-line disassembler built in. You can disassemble byte-by-byte, or an instruction at a time. As with the original FED, FEDII has on-line help in the form of a command menu. Be aware of the fact that many LSI products for the 6.x operating systems are "limited backup masters". This means that there is a limit to the number of copies of the product you can get from the distribution diskette. The products I have seen provide for 25 total copies, which seems adequate for almost everyone.

Misosys has also released many of their products to run under TRSDOS/LDOS 6.Ø. So far, I've seen 6.Ø versions of EDAS, DSMBLR III, PDS, MEMDIR, PARMDIR, and DOCONFIG. The last three are included in one package similar to the MSP-Ø1 package for LDOS 5.1.3. A new program, SWAP, is included in this package. SWAP allows you interchange any two logical drives in the system. For example, SWAP :1 :2 exchanges the DCT (Drive Code Table) information for logical drives 1 and 2.

PowerSoft in Dallas is also introducing several products for the Model 4. Power-Mail, which is just about the best mailing list program I've ever used, is available now, and other products will follow.

As you've probably noticed, all these products are versions of existing LDOS 5.1 software packages. This means, that for the first time, a generally popular personal computer is being supported with mature second-generation software. Even though LDOS has gone through a major new implementation, the general design and concept of the system survives! If I were betting on the market, I'd say Tandy is gonna sell a LOT of Model 4s, and that this machine is going to have some of the best and most compatible software ever seen in this industry. If this does happen, it will be in no small measure because Tandy chose to adopt the most powerful DOS in the 8-bit market as their new standard.......

<u>THE "C" LANGUAGE (Part IV)</u>

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

Hello! Nice to 'C' you again! The topic for PART IV is logic, control and flow. The specific C language vocabulary words that will be used in this part are:

```
    for      while    if         else
    switch   break    continue   do        goto
```

In previous parts, statements and blocks were mentioned. In conjunction with the above logic, control and flow vocabulary words, statements and blocks of statements accomplish the tasks designed into a C program. Let's take a look at these C vocabulary words and their use in a C program.

But first, a quick reminder about (expressions) statements and blocks! Remember, a C statement is an expression followed by a semicolon. For examples:

```
    a = 24;
    c = getchar();
    printf("%d \n";e-18);
```

These are all examples of C statements. Each expression is ended with a semicolon. It is used in C as a statement terminator rather than a separator. (You might also note in the above example with the printf() function a general rule in C. Wherever it is permitted to use the value of some type of variable, it is also permitted to use an expression of that type. Hence the e-18 expression is used instead of having to assign it to some intermediate variable. You can save a lot of coding using this rule, but be careful! You can also make your program confusing!)

Whenever it is necessary to group statements (declarations, etc.) and treat them as one, they can be enclosed in braces {}. This creates a "block" or "compound statement". This block enclosed by the {} braces is not followed by a semicolon even though the enclosed statements are treated as one. The need for blocks or compound statements will be seen as the C logic, control and flow vocabulary words are explained. Shall we begin as K&R does with the if statement?

The general format (syntax) of the if statement is:

```
    if (expression)    statement_1
       else            statement_2
```

(You will note the importance of differentiating between a statement and an expression!) The if-else statement is used to make decisions. The expression is evaluated. If it is true (i.e., has a non-zero value) then statement_1 is executed. The else is optional. If it is present and the expression is false (i.e., has a zero value) then statement_2 is executed. Since the else is optional and can be omitted, you could be confused by the following:

```
if (a = 2)
    if (c = 2)
        d = 2;
else
    d = 4;
```

The rule in C is that the else is associated with the closest previous else-less if. The way the above compound if statement is indented you may be led to falsely believe that the else should be paired with the if it is aligned with. Another important point to mention here deals with indentation. It is generally practiced to have the else aligned with the if to which it belongs. Thus the following is more readable:

```
if (a = 2)
    if (c = 2)
        d = 2;
    else
        d = 4;
```

If the else was in actuality to be paired with the first if, then the {} can be used to force the proper association as follows:

```
if (a = 2) {
    if (c = 2)
        d = 2;
}
else
    d = 4;
```

The else is thus paired with the first if. The second if is contained in a "block" and is the statement_1 referenced in the general format of the if statement. Of some note also is the placement or "style" of placing the {} and their alignment in the above if else statement. Each C programmer develops a way of placing and or aligning if-else, else-if and the {} braces. Consider the following two examples:

```
/*   EXAMPLE  1              */
if      (expression) statement
else if (expression) statement
else if (expression) statement
else                 statement

/*   EXAMPLE  2              */
if   (expression)
     statement
else if (expression)
     statement
else if (expression)
     statement
else
     statement
```

Both examples work the same but are of different styles. Perhaps the most popular or common style (used in the K&R book) is represented via the second example. Example 1 may look nice too, but consider how long the actual expressions and statements may be. If they are quite long, the style of example 2 may appear nicer. Whichever style (method) you choose, it is a good rule to be consistent.

If you noticed, the above two examples demonstrate a generalized way of writing a multi-way decision. If any expression is true, its associate statement is executed and the whole else-if chain is ended. If none of the expressions are true then the statement after the last else is executed. This represents the "default case". Any of the statements can be a block of statements in the {} braces. The last else could also be missing and there would be no default statement executed.

Another way of making a multi-decision in C is with the switch statement. The syntax for the switch statement is:

```
        switch (expression)  {
        case  constant:
              statement
              break;
        case  constant:
        case  constant:
              statement
              break;
        case  constant:
              statement
              break;
        default:
              statement
              break;
        }
```

(Notice again the style used to place the {} braces!) The switch statement is followed by an integer expression and a block enclosed in braces. The logic of the switch statement is to evaluate the integer expression and compare its value to the constant case values. Each case is "labeled" by a constant expression (usually an integer or character constant). If a case matches the value of the expression, that case begins the execution. Statements after that case are then executed. If a break statement is encountered the switch statement (block within {} braces) is exited. If no cases match the expression then the default case begins the execution. The default case is optional. The cases and default can occur in any order, but the cases must all be different. If no cases match and no default case is present, nothing happens at all. (Nothing happening at all has been described as "the sound of one hand clapping"). It is good programming practice to put the break statement at the end of a case. If a break is not present, execution "falls through" to the statements which follow. This may not be the desired action! An example of the switch statement follows:

```
     switch (answer) {
     case 'y':
     case 'Y':
          printf("The answer was YES!");
          break;
     case 'n':
     case 'N':
          printf("The answer was NO! ");
          break;
     default:
          printf("Enter only Y or N! ");
          break;
     }
```

The above switch statement could possibly be used to test for a Y<es> or N<o> reply. Note that it uses a case for the upper or lower case possible responses. You are no doubt asking what happens if the default case is executed and you want to allow another response until Y or N is entered? Well, you could use the C statements which allow looping! Looping (executing a statement or groups of statements a given number of times) can be accomplished in C via four basic ways: for, while, do-while and goto.

The syntax of the while statement is:

```
    while (expression) {
      statement
  }
```

If the expression after evaluation is true, the statement is executed. The expression is then re-evaluated and if true statement is executed again. This process is repeated until expression is false (zero).

The syntax of the for statement is:

```
    for (expression_1; expression_2; expression_3) {
        statement
    }
```

Expression_1 and expression_3 are typically assignments or function calls and expression_2 is an expression to be evaluated as true or false (a relational expression).

Another way to write the for statement using while is shown as follows:

```
    expression_1;
    while (expression_2) {
        statement
        expression_3;
    }
```

From the explanation of the while, you can see how the for statement works. In the for statement the expressions could be multiple expressions separated by commas. For example:

```
      for (i=Ø,j=Ø; (s[i] != Ø); ++i) {
            if  (s[i] == 'a') ++j;
      }
```

Whether you use the while or the for statement is just a matter of choice. Typically the for is used for simple initialization and re-initialization. It is analogous to the FORTRAN DO loop or BASIC for-next statements.

The syntax of the do-while is :

```
    do
      statement
    while (expression);
```

The difference between the do-while and while is a subtle one. With the do-while, the statement is always executed at least once. The expression is evaluated at the bottom of the loop instead of at the top.

Remember the break statement from the switch? It can also be used in the for, while or do-while to exit. Another statement, the continue statement is related to break. It does not exit from a for, while or do-while statement but causes the next iteration of the enclosing loop to happen. An illustration for you to ponder:

```
  for (i=Ø,j=Ø; (s[i]); ++i) {
    if (s[i] == 'a')  continue;     /* Skip this character */
    if (s[i] == '\n') break;        /* Exit for if new line */
    ++j;
  }
```

In the above for statement, the only ways for it to end are if s[i] equals Ø or the new line character. Note that the relational expression is (s[i] != Ø) but it can be and is shortened in this example to (s[i]).

With the above new C language commands, you can perform various logic patterns, and control the flow of a C program. Another flow control C statement is the goto. The object of the goto is a label. A label has the same form as a variable name but is followed by a full colon. The goto and the label to go to must be in the same function. The use of the goto is not recommended, except for possibly branching out of some heavily nested logic.

Next time, in PART V, the topic will be initialization, more on blocks, pointers and arrays. C you next time!

GENERAL INTEREST

It has been reported that the "Active Variable Analyzer" in the last issue works as listed only with the old "Memory Size" Model 1 ROMs. Mr. C. E. Clayes reports that if the 9B at the 21st row down, and the 18th column across in the BINHEX listing is changed to a 7C, the resulting program will work on the new "MEM SIZE" ROMs. Another LDOS user reports that a change to 6F should work on both ROM types.

Some people have been reporting difficulties with the Radio Shack Double Density adapter. Remember-- the RS DDen unit requires at least LDOS 5.1.3,and will not function with any earlier releases. Also, the proper driver to use is RDUBL, <u>not</u> PDUBL. Lastly, this adapter should only be installed by a competent computer technician, as it requires alignment when it is installed. If you just "plug it in", it may seem to work, but reliability of disk I/O will be questionable.

In regards to "Fix that SOLE GAT error" in the April '83 LDOS Quarterly, Mr. R. D. Greet reports that there is an easier method. He has supplied the following patch:

```
. Patch for SOLE2/CMD
. This patch modifies SOLE2 so that Directory 'fix' programs
. do not generate a GAT error for track Ø on DDen boot disks
X'53D5'=CD CB 57
X'57CB'=3C 32 17 58 3A Ø1 58 CB FF 32 Ø1 58 C9
. END OF PATCH
```

Mr. Greet has a Percom-type DDen adapter. This patch may work with the RS-type also.

He also has supplied the following patch to correct existing directories. If you patch DIR/SYS, you must use REPAIR :d (ALIEN) or the extended debugger to re-write the system DAM on the directory track. Do NOT work on a disk in drive Ø.

```
                    PATCH DIR/SYS.SYSTEM (DØ2,Ø2=8Ø:DØ2,17=Ø2)
```

<u>The following are mandatory patches to LSI products:</u>

In 5.1.4, the Date and Time prompts were changed to accept a wide range of delimiters between digits, rather than just "/" and ":". However, the Time prompt will now NOT accept a colon (oops). To remedy that, apply the following patch to SYSØ:

```
. Patch SYSØ/SYS.SYSTEM - MOD 3 ONLY!        . Patch SYSØ/SYS.SYSTEM - MODEL 1 ONLY!
DØE,A5=3B                                    DØE,63=3B
. EOP                                        . EOP
```

```
. FMA/FIX - Ø7/14/83
. This patch is to the 5.1 version of FM to correct problems in moving system files
D19,62="a"
DØ1,Ø9=11 8Ø 58 C3 DE 66
DØE,3C=CD C8 59
DØ1,ØF=11 4Ø 58 7E E6 5Ø FE 5Ø CØ F1 C3 7D 5E
DØ5,B2=CD CE 59
```

```
DØ1,1C=CD 9Ø 5A 36 22 C9
D27,A3=C3 DB 59
. EOP


. TBA51B/FIX - Ø7/22/83
. This patch is to the LDOS 5.1 version of The BASIC Answer
. It fixes the local variable DC problem.
DØ6,BØ=EB 1A CD 6E 6C BE 2Ø Ø4 23 C3 C6
DØ6,CF=13 1Ø DF 22 41 5F C3 B3 5F
DØ5,64=Ø4 48 7E CD Ø7 5E 1Ø FA
DØ5,Ø5=C3 ØE 5E CD 6E 6C 12 23 13 C9
D1B,9D="b"
. EOP
```

<u>The following are optional patches</u>

```
. MAXPR - Auto LF patch to SYSØ
. This patch is for SYSØ/SYS on the MAX-8Ø. It provides for permanent
. linefeed after carriage return for use with printers that need this,
.  and eliminates the need to set the PR/FLT (ADDLF).
X'Ø4Ø1'=CD Ø9 Ø1
X'Ø1Ø9'=DD 34 Ø5 FE ØD CØ CD 22 Ø4 2Ø FB 3E ØA 32 E8 37 C9
. EOP
```

The following patch has been requested. This patch will "back-off" the patch to Model 3 LDOS that allows use of the faster clock rate of the Model 4. This should only be used on Model 3 machines with speed-up kits. The resulting configuration will match the information published in the Jan '83 article on speed-up kits.

```
. Reverse of Mod 4/3 mode clock patch.  This patch is for Model 3 LDOS ONLY!.
. Patch SYS7/SYS and also apply the SYSØ/SYS patch.
DØD,A2=3A AØ 42 F6 Ø1
DØD,AE=3A AØ 42 E6 FE 32 AØ 42 D3 FE
. end of patch

. Reverse of Mod 4/3 mode clock patch.  This patch is for Model 3 LDOS ONLY!.
. Patch SYSØ/SYS and also apply the SYS7/SYS patch.
DØF,66=FE Ø1 21 AØ
. end of patch
```

The following patch was supplied by Mr. W. Fields

High/Fix - This modification to the HIGH utility from Utility Disk #1 causes HIGH to pause at the end of each screen and prompt the user to press any key to continue. This will prevent the information from scrolling off the screen if more than six modules are in high memory. This patch will also correct a bug in the display of "UNKNOWN's".

```
. HIGH/FIX
. This fix is for the version of HIGH
. that has a modification date of
. 16-FEB-83. (Version 1.Ø.1 in output
. headings)
.
. William Fields
. Post Office Box 112Ø
. Glendale Heights, Ill 6Ø137
.
. First we patch each significant call
. to @dsply to go instead to the patch
. area code first.
.
X'52Ø7'=9D 53
X'521F'=A1 53
X'5233'=A5 53
X'528C'=A5 53
X'52B1'=DØ 53
.
. Now free up a byte for a counter.
.
X'5312'=1Ø Ø4 DD E1 C9 ØØ
.
. Patch area code here.
.
. The following code creates the screen pauses
.
X'539D'=3E Ø4 18
X'53AØ'=Ø6 3E Ø2 18 Ø2 3E Ø1 F5 CD 67 44 F1 21 17 53 86
X'53BØ'=FE ØE 3Ø Ø4 32 17 53 C9 21 D7 53 CD 67 44 CD 49
X'53CØ'=ØØ CD C9 Ø1 21 4A 53 3E ØØ 32 17 53 CD 9D 53 C9
X'53DØ'=CD 33 ØØ 3E Ø1 18 D5
X'53D7'="Press any key to continue."
X'53FØ'=Ø3
.
. The following code corrects the address display for "UNKNOWN's"
x'5283'=C3 F2 53
x'53F2'=E5 2B EB CD E5 52 C3 88 52
.END OF PATCH
```

LET US ASSEMBLE

by Rich Hilliard

Welcome back! Last time we discussed various cutesy screen displays of famous sorts. While no criticism from you was apparent, we may have moved too quickly into the land of nod. Keep in mind that the purpose of this column is to be of assistance to you, the viewer. Therefore, if you want something specific discussed, please write in and tell me and we will work on it. So far, we have had a very interesting suggestion from Mr. Woodson of Atlanta, which is an assembler program to compute moving averages. This type of program takes a long time in BASIC. We will start the preliminary work on this project next time.

By the way, this is exactly the type of subject which is ideal for learning assembler. If you have a BASIC program of your own which lasts as long as an an all day sucker, why not submit it?

And now on to the task at hand, number base conversion. Oh no! Not number base conversion ... anything but base conversion .... please, please get us out of this!!! Holy bovine fecal matter, batman, calm down. Conversions are our friends (just like dogs and fire, however, they can do us harm if abused). Actually, they are not bad at all. Amaze your friends, write a conversion program for LDOS.

Number base conversion is often present in assembler programming because the stupid computer can only calculate in binary. Meaningful numbers (decimal) must be obtained from the ten-digit monkey running the machine, converted so that the stupid two-digit computer can deal with it, and then the result converted back to monkey. The three most used number bases in our little corner of the universe are (in alphabetical order): binary, decimal, and hexadecimal which are respectively the computer's, ours, and our method of looking at the computer's.

Base conversion is very simple in itself but we have (of course) a further problem to deal with. The computer is quite content to honk along without ever telling us what is going on. Furthermore, it has no use whatsoever for English. After all we made all that up in order to get fed (and keep from being a meal). Since most computers do not eat, they have no use for our language. The need does exist for us to know what our little inventions are doing, and from time to time, to send this information to other devices or computers. To do this, a standard (ho, ho, ho) code was established called ASCII. Like every other standard that I know of in this industry, it isn't.

The purpose behind ASCII is so that when a byte (in English) is sent to a printer or another machine, the character sent is understood at the other end. Where this breaks down is as follows: ASCII only accounts for seven bits out of the eight bits in a byte. This means that while the values Ø through 127 are more or less accounted for, the numbers 128 through 255 are up for grabs. In fact, even within a single manufacturers product line, that manufacturer seldom is sooth (this last for D & D fans) regarding their purpose. As an example, in a Model III, 128-191 are used for graphics, and 192 to 255 are space compression codes. An "alternate set" can be switched in which wipes out space compression and gives you the greek alphabet and other assorted junk. (On the Mod 4, reverse video occupies these codes as yet a third alternative.)

This puts an additional conversion sequence into any code because the number "3" when typed at the keyboard is not represented by the value ØØØØ ØØ11, but by the value ØØ11 ØØ11 (ASCII). (Can you guess what must be done to convert it?)

I want you to understand that these conversions are standard in every applications program written in assembler that obtains input. Therefore, let us establish a series of subroutines necessary to convert all this stuff. BASIC handles much of this automatically (see "&H"), especially the ASCII conversion. But consider, when the statement INPUT A is encountered, BASIC already knows that the information coming from the keyboard will be ASCII decimal numbers only (English - you see), and it rejects any non-decimal characters. A better appreciation of our problem is seen by the statement LINEINPUT A$. Now BASIC merely accepts a character stream until the <ENTER> key is pressed or 255 characters have been received.

In assembler, all of our keyboard inputs are exactly like that. We have no idea what characters are coming in, so we must examine every character for its relevance and act accordingly. Most of the needed conversion routines can be found in a program which takes in a number of any of the mentioned bases, and displays the converted results in all three bases.

Let's define the program in English:

1. Take in an ASCII binary, decimal, or hexadecimal number.
   - A. A number with suffix "B" is binary
   - B. A number with suffix "H" is hex
   - C. A number with no suffix or suffix "D" is decimal
2. Convert the input from ASCII to binary.
   - A. ASCII-binary to binary
   - B. ASCII-decimal to binary
   - C. ASCII-hex to binary
3. Display the ASCII representations.
   - A. Binary to ASCII-binary
   - B. Binary to ASCII-decimal
   - C. Binary to ASCII-hex

Rather than duplicate this process in BASIC, I will simply include it in the comments. To make life easy on us, we will demand suffixes for the declared number. I do NOT recommend doing things "the easy way", but we must walk before we decathalon.

It can be seen that we need a main line program which takes a number from the keyboard and whips it to one of three subroutines to get it into binary. We could then write code which determines which was input and not convert it, but why bother? The results are calculated so quickly that little (if any) time will be lost. Therefore, we will simply convert the number through all three "back to ASCII" routines, one of which, admittedly, need not have been done. We are going to use system vectors @KEYIN, @EXIT, and @DSPLY. So the first lines of code are as follows:

```
        00100 @EXIT      EQU        0402DH                    ;Normal Exit vector
        00110 @DSPLY     EQU        04467H                    ;PRINT subroutine
        00120 @KEYIN     EQU        00040H                    ;LINEINPUT routine
        00130            ORG        05200H                    ;start code at X'5200'
```

The @KEYIN system vector requires the HL register pair point to a spot in memory where the input from the keyboard will be stored (be sure to look up @KEYIN in your LDOS manual that you may know what secrets are written therein). This buffer will be of a maximum length as determined by the contents of the B register plus one. Let's set a perfectly arbitrary limit to the size of the converted number to be two bytes long. If this were entered in binary it would be sixteen characters in length. Finally we need a suffix of one character, so the buffer length required is eighteen.

By the way, it is good to write (communication - what a concept!) down things that need to be done later in the program so that they are not forgotten. Right now write down that we need to define an input buffer of 18 characters named NBUFFER.

Now, the stupid computer won't tell us what is going on - so we better inform the user what he is in and what to do about it. To do that, we print the message labeled "SIGNON" to the video. This routine will be labeled because we will come here until told to leave or if an erroneous input is detected. Write down that we need to compose SIGNON. Remember lesson 1 and code the message printing as follows:

```
        00140 START      LD         HL,SIGNON                 ;greet the masses
        00150            CALL       @DSPLY                    ;print it
```

Since the program is quick, dirty and user hostile, our complete "documentation" is contained in the message we just printed and we can now take in the desired input:

```
        00160            LD         B,17                      ;maximum chars allowed
        00170            LD         HL,NBUFFER                ;stick them in memory
        00180            CALL       @KEYIN                    ;GOSUB LINEINPUT
```

@KEYIN does not come back until either the <BREAK> or the <ENTER> key is pressed. If the maximum number of characters is reached, @KEYIN will not allow any other keys to work except <BREAK>, <ENTER> or the backspace. When control comes back to us, the B register contains the number of characters received and if <BREAK> was pressed the C flag of the F register is set. <BREAK> will be our signal to stop executing. This is somewhat of a PROBLEM because if the user has set SYSTEM (BREAK=N) or CMD"B","OFF" from LBASIC, then we simply never leave our program. A way to prevent the hang-up would be to alter our routine to examine for some other key, or to make certain that the system does handle <BREAK> by checking in the SFLAG$ vector, but as I said this is user hostile. Anyway, you can always blame the user because running with the <BREAK> key disabled in <u>ALL</u> software is not a good plan. Anyway, we check C flag and jump to @EXIT. One other thing-- some putz will always press the <ENTER> key by itself so we will check that the B register contains a not-zero value. Let's be kind and jump back to the prompt if this happens.

```
        00190            JP         C,@EXIT                   ;leave if <BREAK> pressed
        00200            INC        B                         ;TEST for zero characters
        00210            DEC        B                         ;If B was zero this sets Z
        00220            JR         Z,START                   ;& we start over
```

Okay- we got something in the input buffer! We must determine which of our three conversion SUBS to CALL. Remember that we needed an "H", "B", or "D" at the end of our input. We are also assuming that if no "H" or "B" is present that "D" is assumed. HL is still pointing to the front of the character string we called NBUFFER. Well sir, we know the length of the string and where it starts so all we have to do is point HL at the last character, load it into A and do a mess of compares. We need to find the equivalent of MID$(NBUFFER,LEN(NBUFFER)-1,1). Then we look for the suffix. If it is there we must also lop it off before going to the subroutines. This is done by decreasing the length by one. We will point HL to the proper location by placing the length of NBUFFER (from B) into the DE pair and adding it to HL. It would be slick if we could add B directly but such luck is not with us:

```
        00230            LD         E,B                       ;put B contents into DE pr
        00240            LD         D,0                       ;note the order
        00250            DEC        E                         ;adjust for zero
        00260            PUSH       HL                        ;save first char
```

```
ØØ27Ø           ADD     HL,DE           ;HL => last char
ØØ28Ø           LD      A,(HL)          ;put pointed to in A
ØØ29Ø           RES     5,A             ;Convert to upper
ØØ3ØØ           CP      'B'             ;Binary suffix?
ØØ31Ø           JR      NZ,TESTH        ;GOTO TESTH line if <>
ØØ32Ø           DEC     B               ;ELSE drop the "B"
ØØ33Ø           POP     HL              ;update pointer
ØØ34Ø           CALL    ASCBIN          ;GOSUB ASCBIN
ØØ35Ø           JR      PRINTEM         ;and GOTO PRINTEM
ØØ36Ø TESTH     CP      'H'             ;Hex suffix?
ØØ37Ø           JR      NZ,TESTD        ;GOTO TESTD if <> "H"
ØØ38Ø           DEC     B               ;loose the "H"
ØØ39Ø           POP     HL              ;update pointer
ØØ4ØØ           CALL    ASCHEX          ;GOSUB ASCHEX
ØØ41Ø           JR      PRINTEM         ;and GOTO PRINTEM
ØØ42Ø TESTD     CP      'D'             ;decimal suffix?
ØØ43Ø           JR      NZ,TESTD1       ;remove 'D' if present
ØØ44Ø           DEC     B
ØØ45Ø TESTD1    POP     HL              ;update pointer
ØØ46Ø           CALL    ASCDEC          ;GOSUB ASCDEC
```

Well that certainly was a boatload. You can see that it only gets to one of the three ASCII to binary subroutines. Actually, there is no reason to have three different subs in this instance, but if we stay universal, the same three subs can be used again and again. Save them as separate modules and then merge them into any program. In line 29Ø, notice the RES instruction. This resets bit 5 of the A register. The reason for this manipulation is that the suffix received may be in lower case. Observe bit 5 in the following chart:

Character	Upper case ASCII	Lower case ASCII
B	Ø1ØØ ØØ1Ø	Ø11Ø ØØ1Ø
D	Ø1ØØ Ø1ØØ	Ø11Ø Ø1ØØ
H	Ø1ØØ 1ØØØ	Ø11Ø 1ØØØ

You will note that the bit pattern for upper and lower case is identical except for bit 5, which is set for lower case alphabetic characters. Therefore, to force upper case RES bit 5, or to force lower case SET bit 5 of the byte in question. Our program converts any lower case character in the A register to upper case. Otherwise, to be user friendly, we would have had to make six compares instead of three. If any of "BDH" is the last character of the string note that the length of the string is decreased so that there is no interpretation of the last character. The subroutines will have to be written to detect characters outside the allowable range for the type of conversion being done. If such an illicit character is encountered, we will print an error and start over. The purpose of saving HL with PUSH and then POPping it back is so that the leftmost character of the string is pointed to by HL when entering each subroutine. The ASCII-binary to binary routine allows two characters 48 (X'3Ø' or ØØ11 ØØØØ or Ø) and 49 (X'31' or ØØ11 ØØØ1 or 1). Remember where HL is? Thanks to judicious forethought it is pointing at the first character of the string because when we decreased the length we remembered the pointer. What if some wise guy punched B, D, or H as the only thing? Don't worry, we will blow him away with range checking! First let's write the rest of the main body of the program.

```
ØØ47Ø PRINTEM   LD      HL,(NBUFFER)    ;get binary number
ØØ48Ø           CALL    BINASC          ;convert Binary to ASCII
ØØ49Ø           CALL    HEXASC          ;convert hex
ØØ5ØØ           CALL    DECASC          ;convert decimal
ØØ51Ø           LD      HL,PBUFFER      ;show results to video
ØØ52Ø           CALL    @DSPLY
ØØ53Ø           JR      START           ;& back to the top
```

Well, if you've been writing notes correctly, you know that we must define two buffer areas, write six subroutines, and compose a message - imagine trying to remember all that! Since all that follows will be subroutines, why not finish the this section with our messages and buffers and headers (lions and tigers and bears, oh my).

```
ØØ54Ø ERROR     LD      HL,ERRMESS      ;say bad job
ØØ55Ø           CALL    @DSPLY
ØØ56Ø           JR      START
ØØ57Ø SIGNON    DB      ØAH,'Itty Bitty Base Converter :',ØAH
```

```
ØØ58Ø           DB    'Enter number to convert - end hex in H - binary in B',ØAH
ØØ59Ø           DB    'and decimal in D - - press <BREAK> to quit',13
ØØ6ØØ ERRMESS   DB    ØAH,'Number out of correct Range',13
ØØ61Ø NBUFFER   DS    18
ØØ62Ø PBUFFER   DB    ØAH,'       Binary            Hex          Decimal',ØAH
ØØ63Ø BBUFFER   DB    'ØØØØ ØØØØ ØØØØ ØØØØ     '
ØØ64Ø HBUFFER   DB    'ØØØØ             '
ØØ65Ø DBUFFER   DB    'ØØØØØ            ',13
```

We will jump to ERROR (54Ø) whenever an input or out of range error occurs. This simply prints the string ERRMESS to the video and starts over. DS is an EDAS psuedo-op which merely defines an 18 byte gap in the code. This means that whatever was in memory at that location will not be overwritten by loading our program. Strange stuff can occur if you rely on default strings coming from areas created by DS. That is why [BHD]BUFFERs are defined as ASCII zeros and spaces. Defining the buffer in this manner allows us to use these buffers for the conversion back to ASCII, and then by pointing to the string PBUFFER include all four strings (lines 62Ø-65Ø) with one CALL to @DSPLY. This is a quick way to format the output. Remember that tabs are not recognized by either @DSP or @DSPLY. We must either format our own spaces our write a tab generator. For small stuff like this program, it is cheaper codewise to imbed the spaces within the program code as above. Obviously, for variable text formatting or long outputs this would be the ultimate in tacky (not including the IRS). Well, you may cross out a few things from your list. Now we need the six conversion subroutines. Here is the code for the ASCII-binary to binary conversion which we have named ASCBIN:

```
ØØ66Ø ASCBIN    LD      DE,Ø          ;DE will hold the binary
ØØ67Ø ABLOOP    LD      A,(HL)        ;char into A
ØØ68Ø           CP      3ØH           ;is A < ASCII Ø?
ØØ69Ø           JP      C,ERROR       ;IF yes THEN GOTO ERROR
ØØ7ØØ           JP      Z,AGAIN       ;If A=Ø THEN GOTO Again
ØØ71Ø           CP      31H           ;= ASCII 1?
ØØ72Ø           JP      NZ,ERROR      ;see 52Ø
ØØ73Ø           LD      C,1           ;store 1 the one in C
```

The byte values in NBUFFER must be either 3ØH or 31H, which are ASCII "Ø" and "1" respectively. HL is pointing to NBUFFER. We will use the DE pair to hold our binary number. We will examine the string in NBUFFER one character at a time until the string is exhausted. The maximum width of NBUFFER when we get here is 16 characters. Therefore, it is impossible to enter a binary value beyond our two byte limit. To effectively trap errors, we need only check that the digits are either zero or one. The accumulator is loaded with an NBUFFER character in 67Ø. We test for a "Ø" in 68Ø. If the character found is LESS than "Ø", we GOTO ERROR and quit (tsk, tsk, tsk - more on this later). If it equals "Ø" we do nothing with it. Why? Remember that all 16 bits in the DE pair are already zero (line 66Ø). There is no need to convert a "zero" to a real zero, that is the default. If the byte is not "Ø" then it MUST BE a "1" or somebody typed a "B" suffix by accident. Therefore, error city. Now that we know we must ignite a bit somewhere in DE, which bit do we flip? We will manipulate the bit position in C. We start by loading it with one and process thusly:

```
ØØ74Ø           LD      A,B           ;determine placeholder
ØØ75Ø           DEC     A             ;adjust
ØØ76Ø           CP      8             ;determine high or low byte
ØØ77Ø           JR      C,INTOE       ;if A <= 8 then E register
ØØ78Ø           PUSH    BC            ;save the counter in B
ØØ79Ø           SUB     8             ;reset bit position
ØØ8ØØ           JR      Z,SKIP1       ;do not rotate last bit
ØØ81Ø           LD      B,A           ;set up inner loop
ØØ82Ø ABLOOP1   SLA     C             ;shift C left, B times
ØØ83Ø           DJNZ    ABLOOP1       ;for B times
```

The B register contains the count. Whatever B's value is the bit position in DE which must be set to one. Note that bits are numbered 15-Ø (left to right) and that the count in B will be 16 to 1 (garbage odds) so that we must adjust by subtracting one. We load A with B (74Ø) and then DEC A (A=A-1). Now the value in A is the bit position which we want to set to 1. We cannot deal with the DE pair on a bit level, but we can deal with either D or E on a bit level. We must determine which register the desired bit is in. If A is 15 through 8, we deal with D. If A is 7 through Ø, we deal with E. Lines 76Ø and 77Ø determine which path. Obviously, the D register alone does not have a bit greater than 7. We adjust for this by subtracting 8. A special case arises if the result of the subtraction is zero. We do not wish to adjust the C register at all so we skip right to the "stuff the bit in D" gizmo located at SKIP1. Otherwise we get the bit into the correct relative position by shifting it left, for the number of times of the value in A.

A neat little one byte loop is possible in Z-8Ø code. It involves looping by the number contained in the B register. We are going to use it a lot. Put the desired number of loops in B, and set up the junk to do between it by establishing a label where you want the the routine to repeat. This is akin to the first BASIC statement after a FOR ... TO line. The NEXT equivalent is the mnemonic DJNZ (Decrement and Jump-relative if Not Zero). In this case line 82Ø will repeat until B is zero. SLA (Shift Left Arithmetic) moves all bits in the named register 1 position to the left and then puts a zero into the rightmost bit. (Anything dropping off bit 7 is lost.) For example, the contents of C at the start are always ØØØØ ØØØ1. If A were 4, the result in C would be ØØØ1 ØØØØ. We now have the bit in the desired position (bit 4, you will note). All that remains is to get it into D or E, fetch the next byte from memory and process for as long as there are characters.

To get C into D or E we cannot use the load instructions. This is because we are in the process of flipping bits one at a time. A LD in this case would simply wipe out the previous work. To flip the correct bit we use boolean algebra - WAIT! Don't throw up. I'm sorry I used that term. Besides you use it all the time in BASIC. It's just that nobody ever buzzed you with it before. (For those who are interested - it was named after George Boole.) (Buzz off, George. I hate people who name things after themselves). In BASIC, it is sometimes called Hilliardian algebra (but not by many). In BASIC such statements as:

```
                    IF A=Ø AND B=1 THEN GOTO BLAZES ... and
                    IF A=Ø OR B=1 THEN GOSUB MARINE
```

are really balgebra (stick it, George) statements. In assembler, these operations are often used to alter the register contents one bit at a time according to the following tables:

AND

	Ø	1
Ø:	Ø	Ø
1:	Ø	1

OR

	Ø	1
Ø:	Ø	1
1:	1	1

XOR

	Ø	1
Ø:	Ø	1
1:	1	Ø

So to set D or E with the bit in C we will OR D,C. It would be nice, but balgebra works ONLY with A. So we xfer C into A and then OR away. Here is the rest of ASCBIN:

```
ØØ84Ø SKIP1     LD      A,C             ;place bit into A
ØØ85Ø           OR      D               ;merge with current D
ØØ86Ø           LD      D,A             ;update D
ØØ87Ø           POP     BC              ;recover count from 78Ø
ØØ88Ø           JR      AGAIN           ;& goto AGAIN
ØØ9ØØ INTOE     PUSH    BC              ;save the counter in B
ØØ89Ø           CP      Ø               ;If bit zero, skip
ØØ91Ø           JR      Z,SKIP2         ;it and do another
ØØ92Ø           LD      B,A             ;set up inner loop
ØØ93Ø ABLOOP2   SLA     C               ;Shift C left
ØØ94Ø           DJNZ    ABLOOP2         ;for B times
ØØ95Ø SKIP2     LD      A,C             ;bit is in A
ØØ96Ø           OR      E               ;merge with E
ØØ97Ø           LD      E,A             ;update E
ØØ98Ø           POP     BC              ;recover count from 9ØØ
ØØ99Ø AGAIN     INC     HL              ;point to next char
Ø1ØØØ           DJNZ    ABLOOP          ;and do it again
Ø1Ø1Ø           LD      (NBUFFER),DE    ;store conversion
Ø1Ø2Ø           RET                     ;exit subroutine
```

Now the perceptive out there are probably asking themselves, "why did we need the C register at all?" We didn't. We could have loaded A with 1 after determining whether to use D or E, and loading the bit count into B. It would have saved us another step in lines 84Ø and 95Ø. I simply thought that the method we did use was less confusing. Note that we end the loop by reloading NBUFFER with the converted binary value and then RETurning.

This program is interesting to watch under DEBUG. Set the display to 52F3 and single step with various values. To watch the output conversion, set the display to 5332. By the way, when the PC points to CD 4Ø ØØ, do a C (not an I) and enter the number you wish to convert. I suggest we finish the program first, however.

With the ASCHEX subroutine we also start with the leftmost character and again store the converted result in DE. Our binary input could only deal with a bit at a time. Every hex digit, however, represents 4 bits (called a nibble - which is half a byte). This means that four operations on D and E will convert the whole mess. We do have another problem. Hex digits are comprised of the arabic numerals Ø-9 (ASCII 3ØH to 39H) and the letters A-F (ASCII 41H to 46H). Note that they are not contiguous. This means we have to check two ranges for valid characters. Also, the values represented by A-F hex are not reached by simply ignoring the high nibble (sounds like the leader of some rubber-chicken club) as we can do for the digits Ø-9. A common way to convert the decimal numbers is to subtract 3ØH from their ASCII value. The A-F's are converted by subtracting 37H. Since we are learning balgebra, we will use AND to strip off the high nibble. Now here is ASCHEX:

```
      Ø1Ø3Ø ASCHEX    LD       DE,Ø              ;reset DE
      Ø1Ø4Ø           LD       A,B               ;test for >= 5 digits
      Ø1Ø5Ø           CP       5
      Ø1Ø6Ø           JP       NC,ERROR          ;too many
      Ø1Ø7Ø           CP       Ø                 ;test for Ø
      Ø1Ø8Ø           JP       Z,ERROR
      Ø1Ø9Ø AHLOOP    LD       A,(HL)            ;get character
      Ø11ØØ           CP       3ØH               ;A < Ø
      Ø111Ø           JP       C,ERROR
      Ø112Ø           CP       3AH               ;A < 9
      Ø113Ø           JR       NC,CONAF          ;convert if <= 9
      Ø114Ø           JR       STUFFIN
      Ø115Ø CONAF     RES      5,A               ;convert to upper
      Ø116Ø           CP       'A'               ; A < 65?
      Ø117Ø           JP       C,ERROR
      Ø118Ø           CP       'G'               ; A >= G?
      Ø119Ø           JP       NC,ERROR
      Ø12ØØ           SUB      7                 ;convert from alpha
      Ø121Ø STUFFIN   AND      ØFH               ;mask high nibble
      Ø122Ø           LD       C,A               ;save value
      Ø123Ø           LD       A,B               ;determine which nibble
      Ø124Ø           CP       3                 ; 4 & 3 go in D
      Ø125Ø           JR       C,INTE
      Ø126Ø           PUSH     BC                ;save iteration
      Ø127Ø           BIT      Ø,A               ;if zero lsn
      Ø128Ø           JR       NZ,LOWN           ;else msn
      Ø129Ø           LD       B,4
      Ø13ØØ AHLOOP1   SLA      C                 ;shift C left 1 bit
      Ø131Ø           DJNZ     AHLOOP1           ;for four times
      Ø132Ø LOWN      LD       A,C
      Ø133Ø           OR       D                 ;put into D
      Ø134Ø           LD       D,A
      Ø135Ø           POP      BC                ;restore count
      Ø136Ø           INC      HL
      Ø137Ø           DJNZ     AHLOOP            ;do another
      Ø138Ø INTE      PUSH     BC                ;save iteration
      Ø139Ø           BIT      Ø,A               ;if zero lsn
      Ø14ØØ           JR       NZ,LOWN1          ;else msn
      Ø141Ø           LD       B,4
      Ø142Ø AHLOOP2   SLA      C                 ;shift C left 1 bit
      Ø143Ø           DJNZ     AHLOOP2           ;for four times
      Ø144Ø LOWN1     LD       A,C
```

```
          Ø145Ø         OR       E                    ;put into D
          Ø146Ø         LD       E,A
          Ø147Ø         POP      BC                   ;restore count
          Ø148Ø         INC      HL
          Ø149Ø         DJNZ     AHLOOP               ;get another
          Ø15ØØ         LD       (NBUFFER),DE         ;store result
          Ø151Ø         RET
```

No new concepts are in ASCHEX. Note that the BIT tests in lines 127Ø and 139Ø are used to determine whether A is odd or even. This will deal with the high nibbles if even or low nibbles if odd. Note the use of AND in line 121Ø. Since 1 AND Ø results in zero, this type of maneuver is called a mask. The byte ØFH looks like this ØØØØ 1111. The contents of A are altered. Anything that WAS in bits 7-4 of A is reset to zero (1 AND Ø = Ø). If the bits 3-Ø were one, they remain one and if they were zero they remain zero. You can see that the high nibble was "masked" off. What is left in the low nibble (after adjusting for A-F) is the value of the ASCII representation.

ASCDEC presents us with much of the same approach but we have more math to do. Since both binary and hex are conducive to shifting and shafting we could zip around in nib fashion slipping and sliding bits and nibbles about with a certain audacity and flair indicative of a bon vivant attitude. This is not a big surprise. Multiplying and dividing by 1Ø in a decimal system involves slipping and sliding the decimal point around in exactly the same fashion. Now, however, we not only convert from ASCII to a real value, but from base ten to base two.

We will do this by starting from the left of the string, converting the ASCII value to binary. If there is another decimal number, we first multiply the old number by 1Ø, because we know that its value is ten times the next number. Then we add the old number (how old is it?) to the value of the new number so that the last digit read is always the least significant (which reminds me of this explanation). For example, say that the string in NBUFFER is "2345". We will store the transient numbers in IX which has an initial value of zero. OK, here goes ; multiply IX by ten [1Ø * Ø = Ø] ; add the first value (2) to IX which now equals 2 (ØØ1Ø). If the string were only one character long we would be finished. Since there is another character we loop through the routine again. Multiply IX by ten [1Ø * 2 = 2Ø] ; then add the 3, IX now equals 23. Note that the 2 became the "tens" digit. (IX is really ØØØ1 Ø111). There is another digit. Multiply IX by ten [1Ø * 23 = 23Ø] ; then add the 4, IX now equals 234 (111Ø 1Ø1Ø). There is another digit. (You are in a maze of twisty passages, all alike). Multiply IX by 1Ø [1Ø * 234 = 234Ø] ; then add the 5, IX now equals 2345 (ØØØØ 1ØØ1 ØØ1Ø 1ØØ1). Wallah! As you can see, the process can be carried through all 16 bits of IX.

Two really minor problems. (I knew it.) The Z-8Ø has no multiply function. Wait! There's more. Multiplication is actually only shorthand addition. The A, HL, IX and IY registers are the ones capable of math. A can do it all, but is only eight bits wide. HL can add and subtract but can do nothing else. IX and IY can only add. HL is being used to fetch our string, so that leaves IX and IY for the 16 bit arithmetic. Ten is not a very handy binary number, but two is. Can we think of a good method of using two in succession to exploit the power of two and still be ten? Would I bother writing all of this broohaha if there weren't? When we add a number to itself, we get that number doubled (X + X = 2 * X). We now have multiply by two. Remember that. Add the result to itself and we have 4 * X. Add that result to itself and we have 8 * X. (Almost there!) Add that to itself and we've gone too far but - I say but - if we "memorized" the first doubling (2 * X) we can add it to the third doubling (8 * X) and we have TEN times the original number! We did it! (the crowd roars)

Here is the uncensored text for ASCDEC:

```
          Ø152Ø ASCDEC  LD       DE,Ø                 ;reset DE
          Ø153Ø         LD       IX,Ø                 ;reset IX
          Ø154Ø         LD       A,B                  ;test for >= 6 characters
          Ø155Ø         CP       6
          Ø156Ø         JP       NC,ERROR
          Ø157Ø         CP       Ø                    ;test for zero char
          Ø158Ø         JP       Z,ERROR
          Ø159Ø ADLOOP  LD       A,(HL)               ;get char
```

```
        Ø16ØØ           CP          3ØH                     ;test for < Ø
        Ø161Ø           JP          C,ERROR
        Ø162Ø           CP          3AH                     ;test >= ":"
        Ø163Ø           JP          NC,ERROR
        Ø164Ø           AND         ØFH                     ;mask high nibble
        Ø165Ø           ADD         IX,IX                   ;multiply IX by two
        Ø166Ø           JP          C,ERROR
        Ø167Ø           PUSH        IX                      ;save product
        Ø168Ø           ADD         IX,IX                   ;IX = IX*2 (4 * start)
        Ø169Ø           JP          C,ERROR
        Ø17ØØ           ADD         IX,IX                   ;IX = IX*2 (8 * start)
        Ø171Ø           JP          C,ERROR                 ;>65535
        Ø172Ø           POP         DE                      ;retrieve doubled IX
        Ø173Ø           ADD         IX,DE                   ;IX = IX * 1Ø from start
        Ø174Ø           JP          C,ERROR                 ;>65535
        Ø175Ø           LD          D,Ø                     ;put amt into IX
        Ø176Ø           LD          E,A                     ;picked up digit into IX
        Ø177Ø           ADD         IX,DE                   ;add into buffer
        Ø178Ø           JP          C,ERROR                 ;>65535
        Ø179Ø           INC         HL
        Ø18ØØ           DJNZ        ADLOOP                  ;do another if necessary
        Ø181Ø           LD          (NBUFFER),IX            ;stuff in buffer
        Ø182Ø           RET
```

Notice all the jumps to ERROR. This is because the carry flag is set if a bit falls off bit 15 of IX. Since this can happen in so many places, there are many checks. It means that the decimal number entered was greater than 65535. The other error traps involve range checking for the digits Ø-9. Actually, nothing fatal happens if all the JP C,ERROR statements from 166Ø on, are removed. Entry of numbers 65536 through 99999 will return a modulo 65535 result.

Well, where are we? We now have any number input from the keyboard into NBUFFER in a binary state. Using NBUFFER we now convert back to ASCII for each of the three bases. No new concepts. Here is the rest of the code in subroutines DECASC, HEXASC, and BINASC:

```
        Ø183Ø DECASC    PUSH        HL                      ;save for next one
        Ø184Ø           LD          DE,DBUFFER              ;point to decimal in mem
        Ø185Ø           LD          BC,1ØØØØ                ;# of ten-thousands
        Ø186Ø           CALL        DECASC1                 ;GOSUB find decasc1
        Ø187Ø           LD          BC,1ØØØ                 ;# of thousands
        Ø188Ø           CALL        DECASC1
        Ø189Ø           LD          BC,1ØØ                  ;# of hundreds
        Ø19ØØ           CALL        DECASC1
        Ø191Ø           LD          BC,1Ø                   ;# of tens
        Ø192Ø           CALL        DECASC1
        Ø193Ø           LD          BC,1                    ;# of ones
        Ø194Ø           CALL        DECASC1
        Ø195Ø           POP         HL                      ;recover nbuffer
        Ø196Ø           RET
        Ø197Ø DECASC1   XOR         A                       ;cheap way to ld a,Ø
        Ø198Ø ALOOP     OR          A                       ;clear carry flag
        Ø199Ø           SBC         HL,BC                   ;find how many times BC is
        Ø2ØØØ           JR          C,ADD                   ;in hl, if neg put it back
        Ø2Ø1Ø           INC         A                       ;place count in accumulator
        Ø2Ø2Ø           JR          ALOOP
        Ø2Ø3Ø ADD       ADD         HL,BC                   ;restore the one too far
        Ø2Ø4Ø           ADD         A,3ØH                   ;make it ASCII
        Ø2Ø5Ø           LD          (DE),A                  ;put digit in buffer
        Ø2Ø6Ø           INC         DE                      ;point to next place
        Ø2Ø7Ø           RET
```

We simply take the number in HL (NBUFFER) and subtract the highest possible multiple of ten that could be in it. We set BC to 1ØØØØ and subtract. If the subtraction results in a positive number (NC), we subtract again. When the result is negative, we have gone too far, so we ADD back the last minuend to restore the positive value. We also counted the number of successful subtracts in A. The number in A is then the number of ten-thousands that was in the original number. To convert this value to an ASCII number, ADD 3ØH or OR 3ØH. You know that we can print the contents of A to the video through @DSP. Here is the best illustration of our problem. Assume A has 6. If we CALLed @DSP with 6 in A, NOTHING would print because the ASCII value for 6 is an unprintable (not obscene) character used to ACKnowledge in serial communications. To get the number "6" to print on the video, we must first modify it to 36H.

In DECASC, we use DBUFFER to build the string of ASCII characters one byte at a time. Returning from the sub DECASC1, we try each diminutive power of ten until the number is Ø. At that time, DECASC returns to the main body which calls HEXASC:

```
Ø2Ø8Ø HEXASC   PUSH   HL              ;save count
Ø2Ø9Ø          LD     DE,HBUFFER      ;point to buffer with DE
Ø21ØØ          LD     A,H             ;convert high half of H
Ø211Ø          AND    ØFØH            ;mask off bits 3-Ø
Ø212Ø          CALL   SHIFT           ;shift it down to lsnibble
Ø213Ø          CALL   CONASC          ;make it ASCII
Ø214Ø          LD     A,H             ;convert low nibble of H
Ø215Ø          AND    ØFH             ;mask off 7-4
Ø216Ø          CALL   CONASC
Ø217Ø          LD     A,L             ;convert L as we did H
Ø218Ø          AND    ØFØH
Ø219Ø          CALL   SHIFT
Ø22ØØ          CALL   CONASC
Ø221Ø          LD     A,L
Ø222Ø          AND    ØFH
Ø223Ø          CALL   CONASC
Ø224Ø          POP    HL
Ø225Ø          RET
Ø226Ø SHIFT    LD     B,4             ;loop 4 times
Ø227Ø SHLOOP   SRL    A               ;shift right 1 bit
Ø228Ø          DJNZ   SHLOOP
Ø229Ø          RET
Ø23ØØ CONASC   ADD    A,3ØH           ;make it an ASCII
Ø231Ø          CP     3AH             ;does it surpass arabic
Ø232Ø          JR     C,OK1           ;IF no THEN ok1 ELSE
Ø233Ø          ADD    A,7             ;offset for A-F range
Ø234Ø OK1      LD     (DE),A          ;stuff in buffer
Ø235Ø          INC    DE              ;point to next position
Ø236Ø          RET
```

This is the easiest because it just involves adding 3ØH to the values Ø-9 and 37H to the values A-F (1Ø-15) for each of the four nibbles. The sub SHIFT moves information obtained from the high nibble to the low nibble that it might undergo the services of CONASC which adds 3ØH. If the sum is greater than 39H ("9") then another 7 is added. The string is concantenated in HBUFFER and HEXASC returns control to the main body which calls BINASC.

```
Ø237Ø BINASC   PUSH   HL              ;save HL
Ø238Ø          EX     DE,HL           ; DE <==> HL
Ø239Ø          LD     HL,BBUFFER      ;point to buffer
Ø24ØØ          LD     B,2             ;establish loop of 2 bytes
Ø241Ø BLOOP    LD     A,8ØH           ;set up bit marker in A
Ø242Ø BLOOP2   LD     (HL),3ØH        ;set bit to zero
Ø243Ø          PUSH   AF              ;save bit position
Ø244Ø          AND    D               ;see if position is set
Ø245Ø          JR     Z,ZEROD         ;if not - skip it
Ø246Ø          LD     (HL),31H        ;ELSE put a "1" in buffer
Ø247Ø ZEROD    INC    HL              ;point to next "bit"
Ø248Ø          POP    AF              ;restore bit position
Ø249Ø          SRL    A               ;shift it right
Ø25ØØ          PUSH   AF              ;save it
Ø251Ø          BIT    3,A             ;test for nibble's end
Ø252Ø          JR     Z,NOSKIP
Ø253Ø          INC    HL              ;skip over space
```

```
        Ø254Ø NOSKIP  POP     AF          ;restore bit position
        Ø255Ø         JR      NZ,BLOOP2   ;go til nibble is depleted
        Ø256Ø         INC     HL          ;skip space between bytes
        Ø257Ø         LD      D,E         ;place E's pattern in D
        Ø258Ø         DJNZ    BLOOP       ;do the other byte
        Ø259Ø         POP     HL
        Ø26ØØ         RET
        Ø261Ø         END     52ØØH
```

This routine tests each bit to see if it is on or off. It always writes an ASCII "Ø" to the string in BBUFFER. Then it tests for 1 and if it is a one, writes 31H to the string. It writes the zero first to alter the contents of any residual string left from a prior conversion. Also, this procedure skips a space every 5th character to bust up the 16 character string into 4 four character segments for readability.

Notice the error trapping in the last three subroutines is non-existent. This is because the information is coming from the computer, and is not subject to an error which the program can deal with. (You try to program around a dead RAM bank or a power glitch.) Extensive error work is mandatory whenever getting information from a monkey (Monkeys like to type between the keys). Always remember that most operators are merely incompetent or inadequate but be prepared to deal with the sadistic.

Start by assuming that whatever instructions you provide will be ignored except in crisis. Then be prepared to be the object of abuse because your instructions simply tell the operator what to do and not every possible combination of what NOT to do. Above all else a friendly program must not rely on written instructions in lieu of error traps because if you tell somebody that such and such a thing will produce bad results, then that is EXACTLY what they are going to proceed to do in order to observe the disastrous results. (It's kind of like saying, "Whatever you do, don't ever press that red button".)

The problem with the error traps employed in the six routines above is that eventually a system crash would occur if enough errors were repeatedly made. Why? All of the error traps are in subroutines. A CALL instruction PUSHes the return address onto the stack. A RET POPs the address back into the PC. You can see that we JP out of a sub back to the start of code. This leaves an address on the stack which is in deplorable taste. If enough litter was dumped on the stack (which continues to build down in memory), it eventually starts overwriting things it shouldn't because it has finite space to operate in. These kinds of problems are really up to the programmer to solve. It would be unlikely, in fact, it would almost have to be deliberate (see sadist) for this to be encountered, but....

The JP to @EXIT by the way, restores the stack to its correct level, so if we make it back to LDOS Ready we're OK. And now for the LET US ASSEMBLE CONTEST: correct the stack problem mentioned. (hint : see if LD SP,HL and LD (xx),SP can help) Secondly, rewrite ASCBIN and ASCHEX to eliminate use of the C register. Three people will receive a FREE Technical HELP package, for correct replies.

LDOS: HOW IT WORKS

Configuring with non-relocatable code on floppy and hard disk discussed
or--- What's up there anyway?
by Joseph J. Kyle-DiPietropaolo

The LDOS operating system uses high memory for many different reasons. That does not mean, however, that LDOS always uses high memory. The "base" LDOS system does not use any high memory, but also does not allow the use of special LDOS features. Many of these LDOS special features (the KI/DVR, Hard Disk operation, KSM, ...), do use some high memory. Each item that uses high memory can relocate itself to any available area of high memory. Unfortunately, many programs that are not distributed by LSI were not written to these standards. These programs require a certain area of memory to be available, and this may not be the case for any given LDOS configuration.

Does this mean that such a program can't be used on LDOS? Not necessarily-- if the program is otherwise compatible with LDOS, the solution is relatively simple. Within LDOS, there is provision to tell the system not to use a specified area of memory. This area can then be used by whatever program requires it.

Let's look at a specific example. Suppose you have a program that requires the memory from X'E7ØØ' to the top of memory (you may insert the appropriate address from your program).

1) Boot up your system with the <clear> key held down. This will prevent any existing configuration file from loading.

2) Use the MEMORY command to protect the area to be used by your program. In this example, you would type "MEMORY (HIGH=X'E7ØØ')". LDOS will now "avoid" this area.

3) Add any LDOS features you may wish to use at this time (KI/DVR, PDUBL, RDUBL, ...).

4) Use the "SYSTEM (SYSGEN)" command to save this configuration on disk. It will automatically load each time you boot up this diskette.

If you are running a hard-disk system, things are a bit more difficult. At this point, you must set-up your hard disk drivers. This example will show how to set-up the RS 5 Meg hard disk, but the principles will be the same for any system.

1) Use the SYSTEM (DRIVE=n,DRIVER,... command to set up the system.

2) To do so, remember the following facts:

   a) The primary disk drive is I/O select #1
   b) Logical drives Ø-3 will be heads 1-4

3) Type the following "SYSTEM (DRIVE=1,DRIVER,DISABLE)". The driver is "TRSHD3", and the I/O select is 1. There are 153 cylinders on this drive. Number of heads for the partition is 1, and starting head number is 2.

4) same, but DRIVE=2. Starting head number is 3.

5) same, but DRIVE=3. Starting head number is 4.

6) same, but DRIVE=4. Starting head number is 1. The hard disk is now set-up, but drive Ø is still the floppy. The following sequence will finish things up.

7) Enter "SYSTEM (SYSTEM=4)". Floppy disk #Ø will now be logical drive #4, and the hard disk is drives Ø-3. If you have two floppies, use the following command to set-up the second drive. "SYSTEM (DRIVE=5,DRIVER)", and the driver is MOD3 (or MOD1). Physical I/O drive select is #2.

8) Use SYSTEM (SYSGEN) to store this set-up on the hard drive, and COPY CONFIG/SYS.CCC:Ø :4 to move this configuration from the hard disk to the boot-up floppy.

Errata:

In the last issue, this column indicated that 14 + 9 = 25. Please note that 14 + 9 actually equals 23, not 25. I'm sure that nobody is interested in the long and twisted chain of events that led to this error.

In the April '83 Quarterly, the LDOS topic was moving files between DOSes. Super Utility Plus version 3.x will move files between various different operating systems with little or no trouble (on single-sided diskettes). If you have a lot of files to move, the savings in effort alone could easily be worth $79, not counting future use of the program.

THE JCL CORNER

by Chuck (sort of)

This month's Corner will be a bit different than most, in that I'm not going to write it (except for this part, of course). Instead, a guest author will be presented. But first, here is the correct answer to, and the delayed announcement of the winners of the last JCL contest, held in the April issue.

The point that the JCL question was trying to make is that a label will be found even if it is in the middle of a false //IF conditional block. There are several different things that can be done to correct it, but they all boil down to putting the label somewhere else. The three winners drawn out of the bag were Byron Nate of Alberta, Robert Wright of Georgia, and Mark Vasoll of Oklahoma. Congrats on the luck of the draw!

Ever hear the phrase "only limited by your imagination"? Well, to show that this is truly the case with JCL, read the following article by Jim Kyle. By the way, I'd like to see more articles of this kind for inclusion here, so if you have a favorite JCL procedure, send it in now! If worse comes to worse, I may even someday include our JCL (sort of) procedure used to build SYSØ for the MAX-8Ø.

AUTOMATIC CHAINING WITH JCL

by Jim Kyle, 121Ø1 Western View, Oklahoma City, OK 73132
CIS 731Ø5,165Ø (4Ø5) 728-3312

LDOS JCL has uses limited only by your imagination. Here's one more way to use it. Perform the same editing operations on a whole set of files with only one line of keyboard entry.

The task which spawned this idea was to convert a number of rather large files into EDAS-compatible source language. The files themselves were generated by a mix of Fortran, Macro-8Ø, and EDAS modules, and variables were not named consistently in the original source programs. Therefore I used DSMBLR-III to disassemble each of the large files into a set of EDAS source files -- but this lost the mnemonic names completely.

To restore the mnemonic names, and at the same time introduce total consistency of names across the whole set of file sets, I created a simple JCL file which took as its input argument the name of the file to be edited, then invoked EDAS, loaded the file, globally changed each of the desired references, wrote the file back in its original position, and //EXITed. It worked perfectly, but when one original file expanded to 6 or more *GET files during disassembly it required constant attention to invoke the JCL for the next file in sequence. Enter the brainstorm.

Years ago, I was working with an interpreter which passed arguments from function to function, and developed a means of passing a set of arguments one at a time: If the function actually worked with ARG1, I made it accept a sequence such as "ARG1, ARG2, ARG3,... ARGN", then ended it with a call back to itself which passed only "ARG2, ARG3,... ARGN". At the entry, it checked for ARG1="", and if no ARG1 was present, assumed the job was complete and therefore quit. The effect was that by typing the whole set of arguments on the first call, they were processed one at a time and each one that had not yet been processed moved over one place on the next call. (You'll see a resemblance to normal recursive-calling techniques here; that's what led to the idea in the first place.)

The same thing works with LDOS JCL. The //IF - //ELSE - //END construct provides a filter which determines how many arguments were passed to this invocation. When the filter detects that six arguments were passed to DO this time, for instance, it then generates a DO to the same /JCL file, passing only the last five of the arguments to the new DO (the first one has already been used by now and is no longer required). The filter is created by nesting another IF-ELSE-END construct inside the ELSE clause of this one; you wind up, for a 7-argument filter, with a string of 7 //END statements in a row just before the //EXIT statement.

Here's my EDIT/JCL file as the example. The items in angle brackets are comments and should not be included in your JCL. Note that I also use the //INCLUDE statement to get the actual editing commands for EDAS. This makes the outer shell JCL work unchanged for any editing sequence; I just pass the name of the file to be INCLUDEd as one more argument. It does, however, require a second filter to remove the INCLUDE file name.

To start the sequence, just type:

```
DO EDIT (INCL=editfile,FI=filename,A=a,B=b,.....G=g)
```

and stand back. It's fully automatic from there until the set of 8 files has been edited. If you have more than 8 files in your set, this JCL will do only 8 at a time. For the 9th and later ones, type:

```
DO EDIT (FI=filename,FS=H,A=I,B=J,.....G=O)
```

There's no need for the INCL since you won't need to copy the edit sequence again, and including the FS argument keeps the first file from being processed again.

I welcome comments and/or constructive criticism. You can find me on the LDOS SIG of CIS most any night; if I'm not there, leave a message
...jim...

```
//. EDIT/JCL - July 12, 1983
//if -fi                                         <error check for filename>
//. Must define base file name FI=
//quit
//end
//if incl                                        <then copy new INCLUDE file>
copy #incl#/edt incl/edt
//if g                                           <and repeat DO without INCL>
do edit (fi=#fi#,a=#a#,b=#b#,c=#c#,d=#d#,e=#e#,f=#f#,g=#g#)
//else
//if f
do edit (fi=#fi#,a=#a#,b=#b#,c=#c#,d=#d#,e=#e#,f=#f#)
//else
//if e
do edit (fi=#fi#,a=#a#,b=#b#,c=#c#,d=#d#,e=#e#)
//else
//if d
do edit (fi=#fi#,a=#a#,b=#b#,c=#c#,d=#d#)
//else
//if c
do edit (fi=#fi#,a=#a#,b=#b#,c=#c#)
//else
//if b
do edit (fi=#fi#,a=#a#,b=#b#)
//else
//if a
do edit (fi=#fi#,a=#a#)
//else
do edit (fi=#fi#)
//end                                            <these unwind the first nested filter>
//end                                            <of //if b>
//end                                            <of //if c>
//end                                            <of //if d>
//end                                            <of //if e>
//end                                            <of //if f>
//end                                            <of //if g, and filter>
```

```
//exit              <never reached because DO redoes SYSTEM/JCL>
//end                             <of //IF incl nesting>
edas (jcl,abort)
//if fs
L #fi##fs#                        <concatenate FilenameSuffix to Filename>
//else
L #fi#                            <use Filename only>
//end
//include incl/edt                <enables any sequence to be used>
//if fs
W #fi##fs#                        <same as when Loading>
//else
W #fi#
//end
b                                 <get out of EDAS>
//if g                            <start of shift-over filter>
do edit (fi=#fi#,fs=#a#,a=#b#,b=#c#,c=#d#,d=#e#,e=#f#,f=#g#)
//else
//if f
do edit (fi=#fi#,fs=#a#,a=#b#,b=#c#,c=#d#,d=#e#,e=#f#)
//else
//if e
do edit (fi=#fi#,fs=#a#,a=#b#,b=#c#,c=#d#,d=#e#)
//else
//if d
do edit (fi=#fi#,fs=#a#,a=#b#,b=#c#,c=#d#)
//else
//if c
do edit (fi=#fi#,fs=#a#,a=#b#,b=#c#)
//else
//if b
do edit (fi=#fi#,fs=#a#,a=#b#)
//else
//if a
do edit (fi=#fi#,fs=#a#)
//else
. EDIT RUN COMPLETE
//end                             <begin unwinding the filter>
//end
//end
//end
//end
//end
//end                             <of //if g, as before>
//exit                            <at completion of stacked jobs>
```

Letters from the Customer Service Mailbag

A new feature here in the LSI Journal, Letters from the Customer Service Mailbag will present some of the most frequently posed questions, and questions of topical interest to all LDOS owners.

Q: I just got SuperScripsit from Radio Shack. How can I use it under LDOS?

A: The latest version of SuperScripsit from RS is version Ø1.Ø2.ØØ. This version comes with a disk file called "HARDDISK/JCL". Performing this "DO" file will apply the necessary LDOS patches. If this file is not on your diskette, or you have the Model 1 version, contact your vendor to get the proper version and/or complain. Also, see the article on SuperSCRIPSIT later in this issue.

Q: I want to use my (Scripsit) or Microsoft (FORTRAN) or (MACRO-8Ø) or (BASCOM) on LDOS, but I can't find the version that matches the patches on your "FIX Disk".

A: Send us your original master program diskette for proof of purchase. This would be the diskette from Microsoft or Radio Shack, with their original label. Also include $1Ø, or a blank diskette and $5. We will send back your original diskette unaltered, and also send back a diskette containing a LDOS-compatible copy of the appropriate package. Remember, that's $1Ø or a blank diskette and $5 (per program package).

Q: I want to use Profile 3+ on LDOS. What should I do?

A: The original version of Profile 3+ will not function under LDOS. You must get the "Hard Disk Profile 3+" from Radio Shack, Cat. # 26-1593. For existing owners, Cat. #7ØØ-62Ø3 is available as an update.

Q: I'm trying to run my Profile 3+ with JCL, and it's not working right. What's the deal here?

A: Profile 3+ uses a combination of input systems, and most of these inputs will not accept data from JCL files. We have had some success here using "TYPEIN", a utility on our Utility Disk #1. Utility Disk #1 is available directly from LSI for $39, plus $3 shipping and handling.

Q: When I do a LINK *DO *PR to get output on both the video and printer, my printer starts underlining/(insert appropriate print effect here). What's wrong with LDOS?

A: Well, there's nothing really wrong with LDOS, it's just that your printer is responding to the normal video display control codes. One solution is use the PR/FLT, with a parameter of XLATE=X'ØFØØ'. This will effectively remove the control code that causes the problem. If you have an application requiring more sophisticated and/or multiple translations, see our Filter Disks #1 and #2. The Filter disks are available for $29 each plus $3 shipping and handling per disk.

LDOS and SuperSCRIPSIT

by Joseph J. Kyle-DiPietropaolo

LDOS and SuperScripsit is a powerful combination, but some preparation is necessary to ensure success. First, make sure that you have the latest version of SuperScripsit (henceforth known as "SS"). As of Ø9/1Ø/83, this was version Ø1.Ø2.ØØ. For the Model 3, this version includes the LDOS patches for SS in a file called HARDDISK/JCL. DOing this JCL file will apply the patches to SS. As of Ø9/1Ø/83, Radio Shack has not issued patches for the Model 1 version of SS. Some testing has been done, and it seems to work pretty well as-is on the Model 1 under LDOS. The first listing below is a patch to allow directory query from inside SS when running on LDOS. Please don't call asking about use of the Model 3 Dictionary, because LSI is not working on it. Contact Radio Shack with any other questions regarding Model 1 usage.

The next four listings are for the use of Model 3 SS on the MAX-8Ø. These patches will "point" the SS printer driver to the proper MAX-8Ø address, and provide for special character usage. The last two listings are for the modification of the DW2 driver to allow the limited use of LDOS drivers and filters on the *PR device. including the RS232 driver. Similar patches could be made to any other driver, based on the information given here. These patches should work on both Mod 1 and 3 SS, but they have only been tested on the Mod 3 version.

One last note-- if you are attempting to use the ASCII to SS convert function, and can't seem to get SS to read your file, try adding a HEX ØDØØ sequence to the end of your file. If you don't have any sort of file editor, use BUILD with the HEX and APPEND parameters to add these two bytes to your text file.

```
. SCR17M1/FIX
. PATCH for SuperSCRIPSIT version Ø1.Ø2.ØØ MODEL 1 ONLY!!
. This patch will provide directory query from the main SS
. menu, as option <D>. This will not, however, appear on the
. menu, as there is no room.
.
. patch SCR17/CTL
DØØ,36=38
DØØ,3C=D6 3Ø 4F Ø6 ØØ
DØØ,42=63
DØØ,DØ=37
DØ2,4C="LDOS  "
DØ2,A2=44 5Ø 8D
. end of patch
```

```
. SSFIXES/JCL
. PATCHES TO CORRECT SUPERSCRIPSIT PRINTER DRIVERS Ø1.Ø2.ØØ
. FOR MAX-8Ø ONLY!!
PATCH DMP21ØØ/CTL (DØ1,Ø6=32 E8 37)
PATCH DMP21ØØ/CTL (DØ2,95=3A E8 37)
PATCH DW2/CTL (DØ2,21=32 E8 37)
PATCH DW2/CTL (DØ3,35=32 E8 37)
PATCH DW2/CTL (DØ2,Ø3=3A E8 37)
PATCH DWP41Ø/CTL (DØ3,11=32 E8 37)
PATCH DWP41Ø/CTL (DØ3,25=32 E8 37)
PATCH DWP41Ø/CTL (DØ1,F3=3A E8 37)
PATCH LP4/CTL (DØØ,DØ=32 E8 37)
PATCH LP4/CTL (DØ1,C4=3A E8 37)
PATCH LP8/CTL USING LP8/FIX
PATCH DMP4ØØ/CTL USING DMP4ØØ/FIX
.  all 32 E8 37 sequences are replacing D3 F8 ØØ
.      3A E8 37                         DB F8 ØØ
```

```
. LP8/FIX
. Patch for LP8/CTL SuperSCRIPSIT printer driver Ø1.Ø2.ØØ
. TO RUN ON MAX-8Ø ONLY!!!!
.
. CORRECT FOR NEW TOP OF DRIVER POINTER
DØØ,A4=22 BE
. PATCH IN VECTOR TO OUTPUT DATA
DØØ,EF=C3 1E BE
. ALTER STATUS INPUT TO PROPER LOCATION
DØ1,E6=3A E8 37
. ADD NEW OUTPUT CODE
X'BE1E'=32 E8 37 C9
```

```
. DMP4ØØ/FIX
. Patch for DMP4ØØ/CTL SuperSCRIPSIT printer driver Ø1.Ø2.ØØ
. TO RUN ON MAX-8Ø ONLY!!!!
.
. CORRECT FOR NEW TOP OF DRIVER POINTER
DØØ,A4=46 BE
. PATCH IN VECTOR TO OUTPUT DATA
```

```
DØØ,FD=C3 1E BE
. ALTER STATUS INPUT TO PROPER LOCATION
DØ1,F4=3A E8 37
. ADD NEW OUTPUT CODE
X'BE42'=32 E8 37 C9

. Patch to MAX-8Ø SYSØ/SYS for 5.1
. This patch will change certain graphic characters to the
. special characters used by Mod 3 SuperSCRIPSIT.  These graphic
. characters will no longer be available for normal use, so only
. patch a special disk for use with SuperSCRIPSIT.
.
. insert "delta"
DØ6,9F=ØØ ØØ Ø8 14 22 7F ØØ ØØ
DØ8,A4=ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
.   "copyright"
DØ6,EF=3C 42 9D A1 A1 9D 42 3C
DØ8,F7=ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
.   "paragraph"
DØ6,FF=3E 4A 4A 3A ØA ØA ØA ØA
DØ9,Ø7=ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
.        "-(?)"
DØ7,5B=FF E3 DD F3 F7 FF F7 FF
DØ9,63=ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ


. ROM/JCL
. This JCL will create an additional driver that uses the
. *PR DCB vector to allow limited use of LDOS DRIVERS and
. FILTERS.  Main use is to capture a "PRINTER IMAGE FILE".
. Invoke with "DO ROM".
COPY DW2/CTL TO ROM/CTL
PATCH ROM/CTL USING ROM/FIX
//EXIT


. ROM/FIX
. Patches to SuperScripsit 1.2 DW2 printer driver
. these changes make the DW2 driver use the system
. printer driver call to provide the 'hooks' into
. the system.
.
.  Correct for new top of driver, as SuperScripsit
.  maintains this pointer at load address X'BB73'
DØØ,8D=3D BF
.  previous contents were 35 BF
.
.  Ignore printer ready check, as system driver will wait
.  on printer not ready, and we don't want mass quantities
.  of zeros in disk files. This will cause the system
.  to hang on *PR device not ready
DØ2,Ø3=3E 3Ø ØØ
.  previous contents were DB F8 ØØ
.
.  Insert calls to patch area. This driver happens to have
.  two output sequences.
DØ3,21=CD 35 BF
DØ3,35=CD 35 BF
.  previous contents were D3 F8 ØØ   (in both cases)
.
.  Now let's add the call to @PRT. This is an X-PATCH so
.  that it extends the file.  The address will depend on
.  the value found in the pointer at X'BB73', here is the
. correct address for the DW2/CTL driver.
.
X'BF35'=D5 F5 CD 3B ØØ F1 D1 C9
. end of patch
```

To create a disk file of ASCII output---

ROUTE *PR to FILESPEC/EXT
enter SuperScripsit...
print the document (proportional will function, but if used the destination printer must also be a DW2, and the file will be inordinately large.)
Now, you may exit to LDOS and RESET *PR

This patched driver could be in place at all times, but then the system would hang on printer not ready. Don't forget to "block-adjust" if changing drivers.

MAX-8Ø MEMORY MAP - by Chuck

or "Hey... where'd that go??"

The primary design criteria of LDOS for the MAX-8Ø was to emulate a Model III running LDOS. Therefore, all of the documented system entry points and storage areas HAD to remain in the same place as on the Model III. This included the places in the Model III ROM, even though that area is RAM on the MAX-8Ø. Many of you have asked where we put things, and if there are "safe" areas of memory still unused by the system and available to the user. This article should describe where things are, where they aren't, and where there is nothing.

To make things easier, let's define a couple of terms to indicate the different areas of memory on the MAX. LOWROM will mean the area of memory from ØØØØH to 2FFFH. This is the area normally occupied by the I/O drivers and the BASIC code on a Model I or III. HIROM will mean 3ØØØH to 3BFFH. On a Model III, this is used for various things. On a Model I, this area was partially unused, with certain memory mapped addresses defined here and there. VIDRAM is the area from 3CØØH to 3FFFH, and represents the memory mapped video on both the TRS8Ø's and the MAX-8Ø (more on this later). SYSRES will refer to the area from 4ØØØH to 4DFFH.

To start things off, hardly anything was changed in LOWROM from Ø7Ø8H to 2FFBH, as this area contained the BASIC code licensed from Microsoft. The only thing done was to change the cassette I/O entry points to provide an immediate return. However, the area from ØØØØH to 7Ø7H (containing all the I/O drivers, SET and RESET, and some other small routines) was radically changed, because this area in the Models I and III ROM is copyrighted by Tandy. Suffice it to say that we put the necessary code in the right places to make the machine work like a Model III.

Currently, there are scattered areas in the LOWROM area that are not used by the system. However, these cannot be documented because they are the most likely areas to be changed from version to version, and were during the development that produced the current Ø9/Ø1/83 master. Anyone (other than us) that attempts to use these areas is crazy.

The interesting part of the MAX-8Ø is the HIROM area. Since we didn't need to worry about keeping anything in any particular place, there was almost 2.5K of useable memory available for us to play around with. Ecstasy! Wild dreaming! What to do with all that RAM?? Well, here is what came about.
3ØØØH-3ØFFH
This area contains all the routines to access the hardware clock/calendar in the MAX. Near the very beginning is a short vector table to handle the entries to the keyboard driver, @DATE and @TIME. The LBASIC TIME$ code is at the very end of this area.

31ØØH-31FFH
This area contains about half of the floppy disk driver. The other half is up in the normal place, starting around 4585H, just like on a Model III. The code does not go all the way to the end, but it come so close that there is not really any spare RAM that should be considered available for users.

32ØØH-35FFH
Here lives the keyboard driver, the type-ahead and the JKL screen print, plus the type-ahead buffer. This is why no extra memory is used when KI/DVR is set on the MAX-8Ø. There is spare memory near the end of this block. The type-ahead buffer stops around 35BFH. This leaves approximately 64 bytes available.

36ØØH-36FFH
This is an area that is unused by the system, except for two bytes at 36FEH and 36FFH. It is normal RAM, available for the user up to 36FDH.

37ØØH-37FFH
This is "slow" RAM, and contains the memory mapped I/O locations as documented in the MAX-8Ø technical manual. None of the non-I/O locations in this area are used by the system.

38ØØH-38FFH
As defined in the MAX-8Ø technical manual, this area represents the keyboard matrix, and is used as such. This area is the same as the Models I and III, except for the additional keys provided on the MAX-8Ø.

39ØØH-3A6CH, 3A6DH-3BFFH
The first part of this area is sort of strange. It is used during booting, but can later be used as regular RAM. It is not used by the system once the boot has finished. The second half starting at 3A6DH holds the driver for the LOBO hard disk controller.

It appears that the safe spare areas still available for the user are from around 35CØH to 36FDH, and from 39ØØH to 3A6CH. However, be cautioned that if any patches to the routines in the HIROM area need to be done, any patch code that has to be added will go in this free area. Also, there have already been some utilities written by MAX-8Ø owners that use this region, such as MEMDISK programs. If you are using one of these utilities, check with the author before putting your own code in this region.

Now comes the large gray area referred to as SYSRES. This is an all encompassing area that contains the LDOS resident system and areas used by BASIC and LBASIC. Very little had to be changed in this area on the MAX-8Ø. Most of what is different deals with the interrupt handling. Like the Models I and III, there is NO free space in this area.

One special area on the MAX-8Ø is the first area of HIROM, from 3ØØØH to 33FFH. Although this is normal RAM under 5.1.3, it is also the same location that the second half of video memory occupies. This video memory is not used in LDOS, because the 16x64 format of the screen only requires 1K of video RAM, and 3CØØH to 3FFFH can be used. When using an 8Øx24 video driver, the real RAM must be temporarily switched out and the extra video memory switched in to access the screen. From the earlier description of what normally is kept there by LDOS, you can see the conflict. Those writing their own drivers should take this into consideration.

Performing DATE conversions in BASIC

by Dick Konop

There have been several requests to discuss date conversions in BASIC. In particular, converting a date in the form MM/DD/YY to its corresponding day of the year. The following routine will accomplish this type of date conversion. It will also take a julian date (in the form -YY/DDD) and convert it to the corresponding date in the form MM/DD/YY.

The routine is relatively straight-forward, and the Remark statements serve as documentation. For those of you who are not interested in a full blown date conversion process, consider lines 21Ø and 22Ø. These two lines will determine the day of the year using the RAM Storage assignment of DAY$. Note that this date determination process is only valid on LDOS-5.1, while the other date routine can be used with any version of LDOS.

```
1Ø 'This is the init routine. It must be run prior to using the date subroutine.
2Ø '
3Ø DIM D(12):D(Ø)=Ø:D(1)=31:D(2)=28:D(3)=31:D(4)=3Ø:D(5)=31:D(6)=3Ø
4Ø D(7)=31:D(8)=31:D(9)=3Ø:D(1Ø)=31:D(11)=3Ø:D(12)=31
15Ø '
2ØØ 'Routine to compute current julian date; DC=day of the year. This date is taken
2Ø2 'directly from the system. Note that Model I owners should use the addresses 2Ø3 '
X'4Ø47' and X'4Ø48'
2Ø4 '
21Ø IF(PEEK(&H4418) AND 1) THEN DC=256 ELSE DC=Ø
22Ø DC=DC+PEEK(&H4417)
23Ø '
232 'The following routine replaces the ever popular CMD"J" command. The source value
236 'is passed in the variable JD$. It may be in the form "mm/dd/yy", in which case the
238 'day of the year will be passed back from the subroutine. It may also assume the
24Ø 'form "-yy/ddd", in which case the subroutine will pass back the date in the form
242 'MM/DD/YY. Note that the value passed to the subroutine must adhere to the syntax
244 'rules, otherwise the subroutine will return the string "*".
252 'The value of the subroutine (or error value) will be returned in the variable JC$
254 '
262 'To use this subroutine, the following sequence of commands can be used:
265 '
27Ø LINEINPUT"Enter date string (either MM/DD/YY or -yy/ddd) ";JD$
275 GOSUB 3ØØ
28Ø PRINT JC$:END
285 '
3ØØ 'The first thing that is needed is to determine the type of value being processed
3Ø2 ' (i.e. mm/dd/yy or -yy/ddd)
3Ø5 LY=Ø 'reset leap year to "off"
31Ø IF LEFT$(JD$,1)<>"-" THEN 5ØØ 'goto 5ØØ if mm/dd/yy
311 '
312 'Lines 32Ø - 37Ø check to see that a valid date string was passed to the
314 'subroutine, and return an asterisk (*) if a proper date string is not passed.
315 '
32Ø IF MID$(JD$,4,1)<>"/" THEN JC$="*":RETURN
325 IF LEN(JD$)<5 THEN JC$="*":RETURN
33Ø YR$=MID$(JD$,2,2):CK$=YR$:GOSUB 1ØØØ
34Ø IF CK=-1 THEN JC$="*":RETURN
35Ø DY$=MID$(JD$,5):CK$=DY$:GOSUB 1ØØØ
36Ø IF CK=-1 THEN JC$="*":RETURN
365 IF INT(VAL(YR$)/4)=VAL(YR$)/4 THEN LY=1
37Ø IF VAL(DY$)=Ø OR VAL(DY$)>365+LY THEN JC$="*":RETURN
371 '
372 'LY=1 if leap year. February (D(2)) must be adjusted accordingly
373 '
```

```
375 D(2)=D(2)+LY
38Ø DY=VAL(DY$)
381 '
382 'DY contains the day of the year passed to the subroutine. The month is determined
383 'by subtracting the number of days in each month from this value until it is less
385 'than or equal to the number of days in the next month. DY will contain the day of
386 'the month, while L represents the month.
387 '
389 FOR L=1 TO 12
39Ø IF DY<=D(L) THEN 4ØØ
395 DY=DY-D(L):NEXT L
396 '
397 'Lines 4ØØ-45Ø form the date string given the year (YR$), the month (L), and the
398 'day of the month (DY).
399 '
4ØØ JC$="  /  /  "
4Ø5 MID$(JC$,7)=YR$
41Ø VT$=MID$(STR$(L),2):IF LEN(VT$)=1 THEN VT$="Ø"+VT$
42Ø MID$(JC$,1)=VT$
43Ø VT$=MID$(STR$(DY),2):IF LEN(VT$)=1 THEN VT$="Ø"+VT$
44Ø MID$(JC$,4)=VT$
445 D(2)=28
45Ø RETURN
455 '
46Ø 'Lines 5ØØ-63Ø determine the day of the year given the date in the form MM/DD/YY.
47Ø '
472 'Lines 5ØØ-6ØØ perform a check to see that a valid date string was passed to the
476 'subroutine, and return an asterisk (*) if an improper date value was passed.
478 '
5ØØ IF LEN(JD$)<>8 THEN JC$="*":RETURN
51Ø FOR L=3 TO 6 STEP 3:IF MID$(JD$,L,1)<>"/" THEN JC$="*":RETURN
52Ø NEXT L
53Ø MM$=MID$(JD$,1,2):DD$=MID$(JD$,4,2):YY$=MID$(JD$,7)
54Ø CK$=MM$:GOSUB1ØØØ:IF CK=-1 THEN JC$="*":RETURN
55Ø MM=VAL(MM$):IF MM<1 OR MM>12 THEN JC$="*":RETURN
56Ø CK$=YY$:GOSUB1ØØØ:IF CK=-1 THEN JC$="*":RETURN
57Ø IF INT(VAL(YY$)/4)=VAL(YY$)/4 THEN LY=1
58Ø D(2)=D(2)+LY
59Ø CK$=DD$:GOSUB1ØØØ:IF CK=-1 THEN D(2)=D(2)-LY:JC$="*":RETURN
6ØØ DD=VAL(DD$):IF DD<1 OR DD>D(MM) THEN D(2)=D(2)-LY:JC$="*":RETURN
6Ø2 '
6Ø3 'After checking is done, day of the year is calculated, and returned in var JC$
6Ø5 '
61Ø JC=Ø:FOR L=Ø TO MM-1:JC=JC+D(L):NEXT L
62Ø JC=JC+DD:JC$=MID$(STR$(JC),2)
63Ø D(2)=D(2)-LY:RETURN
89Ø '
9ØØ 'This routine checks to see if all characters in a string are numeric, and returns
91Ø 'a -1 in CK if non-numeric characters are found.
1ØØØ CK=Ø:FOR LL=1 TO LEN(CK$)
1Ø1Ø A=ASC(MID$(CK$,LL,1)):IF A<48 OR A>57 THEN CK=-1:RETURN
1Ø2Ø NEXT LL:RETURN
```

LES INFORMATION

by Les Mikesell

This column will cover the differences the @PARAM function between LDOS 5.1 and TRSDOS/LDOS 6.x, and also explain a few details about how the system accesses a disk drive the first time when the system is powered up.

Under LDOS 5.1, the system parameter scanner (@PARAM) will read a list of input values, typically from the command line, and store the parsed value of each input at a specified location. The use of the function is as follows:

```
        HL => opening parenthesis of input list
        DE => table of parameters
        CALL @PARAM
        The Z flag will be set if successful.
```

The input list is in the familiar syntax of all LDOS command parameters, the parameter name optionally followed by an 'equals' sign and a value. Numeric values may either be decimal numbers or hex values using the X' notation. The values ON, Y, YES, (or the name with no value specified) return an X'FFFF or TRUE for the response. OFF, N, or NO will return X'ØØØØ as the response. String values enclosed in quotes return the address of the first character of the string. If a parameter is not given, the value stored for the response is unchanged.

The table of parameters is arranged in the following manner:

```
Parameter name (Uppercase and padded to 6 characters with spaces)
Address to store response value (2 bytes)
....repeat for all parameters...
X'ØØ at end of list
```

TRSDOS/LDOS 6.x versions will also support an identical type of @PARAM using the SVC functions.

```
        HL => inputs
        DE => table
        LD  A,@PARAM
        RST 28H
```

The 6.x version can also use a different type of parameter table structure which can be more compact and gives more information about the type of input. If the first byte of the parameter table is X'8Ø, the alternate structure is used:

```
X'8Ø  indicate alternate structure
--------
Type and length byte:
bits 5-7 indicate type of response desired
bit 4  if set, accept abbreviated response
bits Ø-3 indicate length of parameter name

parameter name follows (uppercase)

Response byte - filled by parameter scanner
Bit 7 set indicates numeric value found
Bit 6 set indicates flag parameter found (yes/no/on/off)
Bit 5 set indicates string parameter found
Bits Ø-4 = length of parameter found

2 byte address to store the parameter value
----------------
repeat for all parameters
X'ØØ  to indicate end of list
```

The setting of bits 5-7 in the type and length byte will not cause an error if the wrong type of input is given, but does form a convenient mask to test the response byte after the @PARAM SVC is executed. Using the newer type of parameter table allows the program to determine the type of input given at run-time, which means that numeric values of X'ØØØØ and X'FFFF can be distinguished from the 'flag' type of response, and a program can be made to handle string or numeric inputs for the same parameter. The 'accept abbreviation' bit in the type byte means that the entry does not have to be repeated to allow an abbreviated form of the same entry, and the 'length' field avoids the wasted space of padding the entries to a fixed number of characters. The bit fields of the type and length byte are easily constructed using the logical 'OR' function of an assembler to merge the fields.

For example, to accept a numeric value for a parameter called SIZE, and allow abbreviation, the assembler listing could be:

```
ABB     EQU     1ØH     ; define bit 4 for abbreviation
NUM     EQU     8ØH     ; define bit 7 for numeric input
TABLE   DB      8ØH     ; <=indicate start of table
        DB      NUM.OR.ABB.OR.4 ;construct type & length byte
        DM      'SIZE'
SRESP   DB      Ø       ; <=response type byte
        DW      SPARM   ; <= address to store response value
        DB      Ø       ; <=indicate end of table
;
SPARM   DW      Ø       ; <=response value will be placed here
```

The above syntax is for the EDAS assembler; others may use different notation for the 'OR' function. After using the @PARAM SVC, the program can check the contents of SRESP. If bit 7 is set (indicating a numeric response), then SPARM will contain the value that was given. If anything other than the parameter SIZE (or an abbreviation) and its value is found in the input list, an error will be generated and the function will return with the Z flag reset.

Careful observers may note that the first access to a disk drive (other than drive zero) is generally much slower than subsequent accesses. The reason for this effect lies in the fact that the disk controller must be told the current head position as well as the desired destination track in order to position the head for a read or write. The head position for each drive is one of the values stored in the system drive code table (DCT). However, on the first access to a drive after the system has been re-booted, there is no way to determine the current head location. Thus, the controller will generally be given the wrong information, and will not find the requested sector on the first attempt.

When this occurs, the disk driver will automatically issue a RESTORE command to the disk controller, which will force the head to go to track zero, regardless of the current position. Then, once the actual position is established, the controller is able to calculate the correct number of steps to reach the desired track on the next try. An addition complication is introduced by the auto-density recognition built into the disk drivers. This is accomplished by performing re-tries after an error in alternating densities. Thus, the first attempt after establishing the head position may be done in the wrong density, and another re-try will be required. The correct settings are logged into the DCT, so subsequent accesses will be correct.

The TRSDOS 6.Ø system attempts to avoid this problem by issuing the RESTORE command to each drive that is enabled when the system is booted. (This may be made optional in later releases.) This ensures that the current head position is known at all times. However, the RESTORE command takes a significant amount of time to complete if it is issued for a drive that is not actually connected. Since the default setting for the system is to have 4 drives enabled, and most machines are only equipped with 2, there is a very noticeable delay as the system is booted. This is easily avoided by using the SYSTEM (drive=d,disable) command to disable the drives which are not available. Then SYSGEN this setting (along with any other desired configuration), and boot-up will occur without the delay. Disabling the unused drives will also speed up global searches where a drive number is not specified for a file.

View From Below the Bottom Floor

Since we did not write the TRSDOS 6.x manual ourselves, it turns out that there are several undocumented features that users may be interested in. We'll throwing in some other optional patches to the newer TRSDOS 6.1 release at the same time. Also, a patch for the 5.1.4 disk driver, as explained later.

To start things off, 6.x has a built-in Repeat Last DOS Command. <CTRL><R> will reissue the last DOS command. This is only valid at the DOS Ready prompt.

Want to do a directory display of more than one drive, but not all drives? With 6.x, try the syntax:

```
DIR :2-      Show all drives 2 or higher.
DIR :1-:3    Show drives 1, 2, and 3.
DIR -:1      Show drives Ø and 1.
```

Here is a patch to the FDC driver for both 6.1 and 5.1.4 that will help alleviate the 3ØØ RPM sync problem, and give smoother disk I/O. However, it disables the interrupts longer, so things like type-ahead, LCOMM, and the spooler will not work quite the same during disk I/O. For TRSDOS 6.1 **ONLY**, NOT for the original 6.Ø release!

```
. Patch BOOT/SYS.LSIDOS
DØC,7D=F3 DB FØ A3 28 FB ED A4
FØC,7D=DB FØ A3 28 FB ED A2 F3
DØD,D1=F3 DB FØ A3 28 FB ED A3
FØD,D1=DB FØ A3 28 FB ED A3 F3
. EOP
```

For 5.1.4 for the Model III, use the following:

```
. Patch SYSØ/SYS.SYSTEM
DØ5,5B=F3
. EOP
```

This patch will correct the FREE map display for TRSDOS 6.x when viewing a hard drive:

```
. Patch SYS7/SYS.LSIDOS
DØ5,38=C5 CD 4F 26
FØ5,38=CD 4F 26 C5
```

This patch to TRSDOS 6.1 will lengthen the drive check timing to be sure of finding a specified file on the first pass of a drive:

```
. Patch SYS2/SYS.LSIDOS
DØØ,E6=1F
FØØ,E6=15
```

For those of you who would like TRSDOS 6.1 to normally display the directory in the allocation (wide) format, use the following patch to SYS6:

```
. Patch SYS6/SYS.LSIDOS
DØ4,BØ=FF FF
FØ4,BØ=ØØ ØØ
```

And last but not least... We have had many requests for a patch to change the REMOVE Library command back to the old familiar KILL. For all you do, this patch's for you:

```
. Patch SYS1/SYS.LSIDOS
DØ1,CB=4B 49 4C 4C 2Ø 2Ø
FØ1,CB=52 45 4D 4F 56 45
```

THE WAIT IS OVER

CP/M®

EXCLUSIVELY FOR THE

TRS-80® **Model 4**

Now, for the first time, unleash the powerful features resident in your Model 4 computer. Open up the vast store of CP/M software such as WordStar®, dBASE II and Multiplan™, along with thousands of others.

- Includes INTERCHANGE ™, a utility that allows reading, writing and copying 20 different manufacturers' disk formats such as IBM, KAYPRO, OSBORNE, XEROX, etc.
- Includes MEMLINK™, a unique feature that uses the optional 64K RAM memory as a fast disk drive.
- Complete with all these CP/M utilities; ASM, DDT, DUMP, ED, LOAD, PIP, STAT and SYSGEN.
- Operates at the 4Mhz clock in the standard Model 4 mode.
- NO HARDWARE MODIFICATIONS. Just insert the disk and boot.
- NO COPY PROTECTION. Backups may be made for your own use and protection.
- The CONFIGURATION program supports a full range of 5-1/4'' disk drives: 35, 40, 77 and 80 tracks, single and dual sided in any combination as well as the standard Model 4 drives.
- Includes MODEM 7, a powerful public domain communications program for file transfer and remote data base access such as CompuServ and the Source.
- Supports 80 x 24 video, reverse video, direct cursor addressing and more.
- Utilizes the Model 4 function keys and allows user defined keys.
- Auto Execute command for turnkey applications.
- FORMAT utility permits up to 52 disk formats to be constructed, all menu driven.
- Fast backup routine with verify for mirror image copies.
- All support programs are menu driven for ease of use.
- Ready to run in the standard 64K Model 4. The additional, extra cost, 64K RAM upgrade not required.
- Complete with over 250 pages of comprehensive user documentation.

AVAILABLE NOW FOR IMMEDIATE SHIPMENT.......................................$ 199.95

The full line of MicroPro software is now available formatted for the Model 4 using our CP/M. Each disk is already configured and ready to run. Just install the printer of your choice and go.

WordStar® Fast memory mapped version	$250
MailMerge® Multi-purpose file merging program	125
SpellStar® 20,000 word proofreader on a disk	125
StarIndex™ Creates index and table of contents	95
WordStar Professional. All the above for only	450
InfoStar™ Advanced DBMS	$250
ReportStar™ Report generator & file manipulator	175
DataStar™ Data entry and retrieval package	150
SuperSort® Fast and flexible sorting is yours	125
CalcStar™ Advanced electronic spreadsheet	95

ORDER INFORMATION

Call now and your order will be shipped at once from our Dallas warehouse. We accept American Express, MasterCard, Visa, and most any other form of payment known to man. Credit cards are not charged until your order is shipped. Add $4 UPS surface shipping and handling on orders within the 48 States. No State Sales Tax on software or shipments delivered outside of Texas. No refunds. Defective items are replaced upon return, postpaid.

ORDER NOW . . . TOLL FREE

800-527-0347
800-442-1310

The Toll Free lines are for orders only.
Specifications subject to change without notice.

128K MEMORY UPGRADE

Our upgrade includes 64K of 150nsec RAM, genuine PAL® chip and instructions for installation. This kit will upgrade your 64K Model 4 to 128K and allow the use of our MEMLINK and TRSDOS 6.x MEMDISK. Comes with a full 1 year guarantee.

A BARGAIN AT ONLY $99.95

AD #883

MONTEZUMA MICRO

214-339-5104
Redbird Airport, Hangar #18
P.O. Box 32027
Dallas, Texas 75232

"WE KEEP YOU RUNNING"

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

BULK RATE
U.S. POSTAGE PAID
Permit No. 36
Mequon, WI 53092

POSTMASTER: DATED MATERIAL DO NOT DELAY

January 1, 1984

Volume 2, Number 5

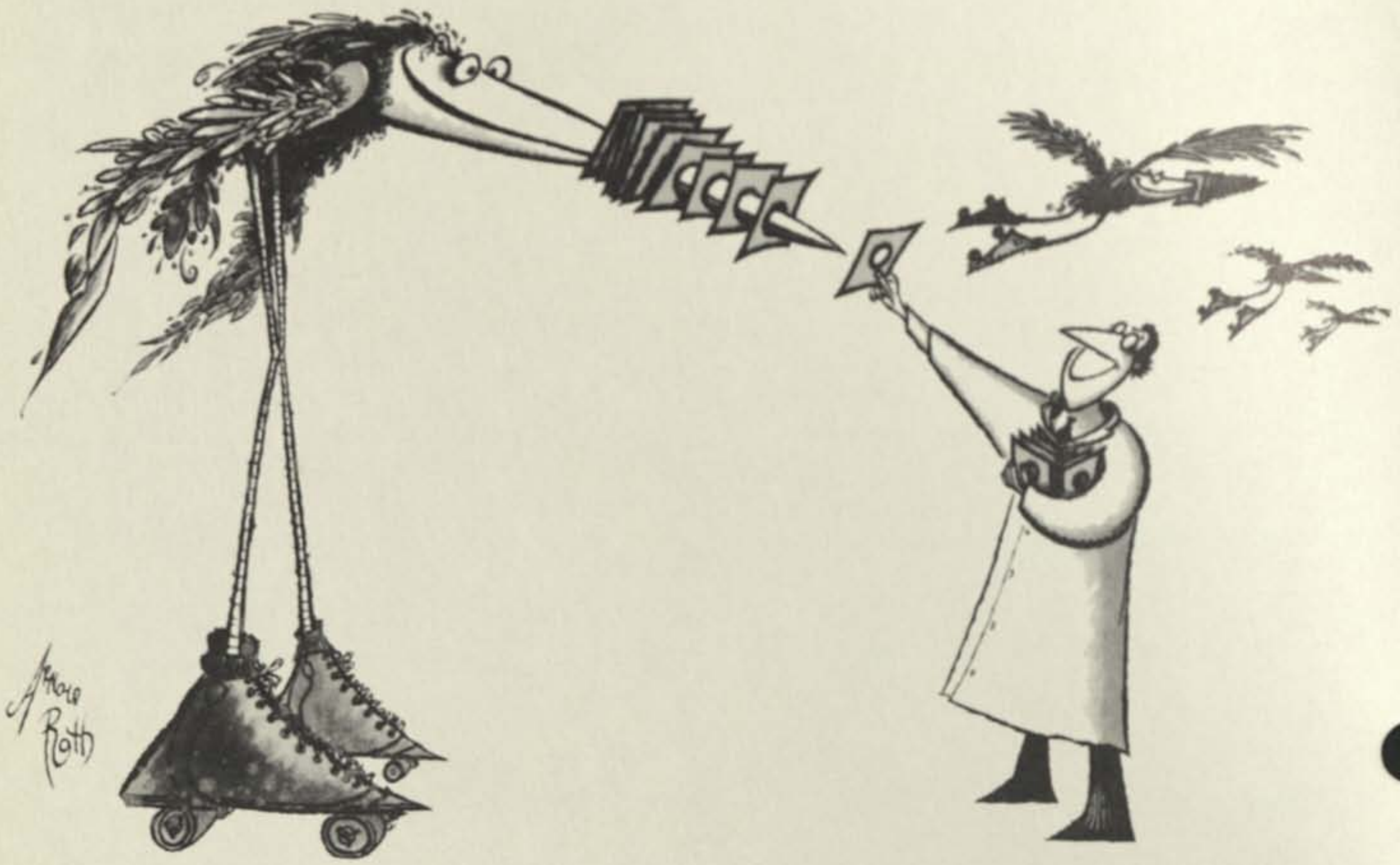
LSI JOURNAL

ISSN 0737-9161

$4

Formerly the LDOS QUARTERLY

Smallware.™



Our software is making a name for itself.

Smallware. That's what we've named our unique software designed for microcomputers. Smallware offers much more than ordinary software: high quality, customer support and a complete product line. You can buy software anywhere. But for the special features of Smallware, The Small Computer Company is your one and only source.

The Small Computer Company is known to many as the company who developed the filing system software Profile III Plus—LDOS and TRSDOS versions—for Radio Shack.

Now, whether you're a microcomputer end-user, dealer or manufacturer, you can order our Smallware directly from us.

Here are just some of the enhancements we offer to Model III LDOS and TRSDOS users:

FORMS: If you prepare forms that require several lines of data, from invoices to shipping instructions, Forms is invaluable. It allows you to print individual forms (up to 13" x 11") with graphics, trademarks, logos, underlining, subscript and superscript functions $125

ARCHIVE: Lets you maintain up-to-the-minute, clean files by removing inactive records and transferring them to a pre-determined list or file; split an existing data base into any number of specialized files; free substantial disk storage space $150

PROPACK™: A tool that lets BASIC programmers more easily customize Profile systems. The resulting programs are shorter, easier to write and faster running. Propack also gives the BASIC program indexed access to Profile data $75

We also have other enhancements for the TRSDOS version of Profile III Plus and for Profile Plus, which runs on the Models II and 12. Call or write for our complete product line brochure.

The Small Computer Company does more than create award-winning Smallware. Our commitment to the customer extends to custom design as well as system consultation.

To order any of these products, call our toll-free sales number, 1-800-847-4740 (in New York state, call 1-212-398-9290).

The Small Computer Company, Inc.

230 West 41st Street, Suite 1200, New York, New York 10036

80 readers' choice micro 1982

Table of Contents

INTRODUCTION FROM LSI:

ARTICLE SUBMISSION POLICY .... Page 2
VIEW FROM THE BOTTOM FLOOR .... Page 3
NEW PRODUCT ANNOUNCEMENTS .... Page 6

FROM OUR USERS:

EZEDIT/FLT - The LDOS command line editor .... Page 7
PRPARM - Change those PR/FLT parameters .... Page 13
SYNONYM - The LDOS command synonym processor .... Page 14
Mod 3 RTC Interrupts - discussed and modified .... Page 22
Profile One Plus - A special treat for Mod 1 users .... Page 26

REGULAR USER COLUMNS:

*** PARITY = ODD *** - Tim Daneliuk .... Page 28
'C' What's Happening - Earl Terwilliger .... Page 31

FROM THE LDOS SUPPORT STAFF:

ITEMS OF GENERAL INTEREST .... Page 35
updates and clarifications

LDOS: HOW IT WORKS - The BACKUP utility .... Page 36
Who, what, where, why, when (and how) discussed

Sending Printer codes - A special application .... Page 38

THE JCL CORNER - by Chuck .... Page 39

Special Feature:

The LSI Journal and LDOS Quarterly Index .... Page 39
by Scott Loomer

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

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

1. A cover letter, summarizing the content and intent of the article.
2. A printed hardcopy of the article. Desired print effects and formatting should be indicated where necessary.

A diskette with--

3. A 'plain vanilla' ASCII text file containing the article. The text should be free-form (without "hard" carriage returns), but any tables or other structured data should be formatted as 87 characters per line. Do NOT send SuperSCRIPSIT or Newscript files. Also, please do not embed print effects.
4. If the article involves assembly language programs, include both the source code, and the object code.
5. Any other necessary files or patches should also be supplied in machine readable form.

Please do not send in printed text without a diskette, as it will NOT be considered for publication. Payment will be made in the form of an LSI product, or $4Ø per published page in the current Quarterly format. The size of the article will determine the value of the LSI product available as payment.

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

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

Submissions should be sent to:

The LDOS Quarterly Editor
c/o Logical Systems, Inc.
897Ø N. 55th Street
P.O. Box 23956
Milwaukee, Wisconsin 53223

VIEW FROM THE BOTTOM FLOOR

by Bill Schroeder

Well, another year has come to an end, and I thank you all once again for being supporters of LSI, and the LSI product line. With your support LSI has made it through yet another year.

1984 will be a year of rapid and massive change in the microcomputer industry. Here are a few of my quickie predictions for the year ahead:

Tandy will try their hand at competing with IBM (no easy task there).

AT&T will enter the market place with a whole new concept

Commodore will continue to dominate the low end market.

Digital Research will increase its presence in the systems software arena.

The REAL "Peanut" will be introduced by IBM.

TeleVideo will become a major force in the market

Removable cartridge hard drives will become more prevalent.

Microsoft will fall slightly from its present lofty position.

Apple will begin to have serious trouble.

Over one hundred small computer product manufactures will go out of business.

The Coleco ADAM will prove to be a failure

Over one-half of the TRS-8Ø software suppliers will falter or fail.

Over 25 microcomputer publications will fail.

It is quite probable that none of the above events will take place, but they are interesting possibilities - - -

Now, on to what I know will be changing at LSI in 1984:

LSI will not be publishing the LSI Journal, but the people at BASIC COMPUTING will be! That's right, starting with the April issue of BASIC COMPUTING, the LSI Journal will be incorporated as a special section in that magazine. This arrangement with these folks was made so that LSI could get out of the business of publishing a "magazine". All the same authors will be writing for us, and we will be providing the material to BASIC COMPUTING for publication, on at least a quarterly basis. All existing subscribers will have their subscriptions filled by BASIC COMPUTING. If you already subscribe to BASIC COMPUTING, your subscription will be extended the proper number of issues. For those who are not subscribers to either publication, a subscription order card is included in this issue.

This means that this is the final issue of the LSI Journal published by LSI. The major bulk of technical and other information will now be imparted through BASIC COMPUTING. But--- there will be a new publication from LSI. This will be the "LSI Newsletter". It will be published for the purpose of announcing new products, special offers and the like to our valued customers. All registered LSI customers will receive this newsletter at NO CHARGE. It will be published on an as required basis, but we expect at least several times per year. It will contain little, if any, technical information.

The LSI Hotline phone number has been disconnected due to lack of interest. There have been very few phone calls, far less than we expected. I can't justify the cost of the phone line, the answering equipment and the creation of the content for such a small audience. This service was discontinued effective January 1st. To those out there who called and appreciated the LSI Hotline, my apologies.

S U P E R S A L E S a n d V A L U E S

DEAL #1: While our supplies last, any product LSI has in stock that is not manufactured or published by LSI will be sold-out for 4Ø% OFF of the suggested retail price. Don't miss this chance. LSI will no longer be selling software that is not published and/or manufactured by LSI. We are liquidating our inventories of these products, to your benefit. All <IN STOCK> products from MISOSYS, MICROPRO, POWERSOFT, MOLIMERX, ... are to be sold at 4Ø% off of suggested retail. We will NOT fill backorders or give rain checks for these products. We have dozens of some items, and only a few of others. These will be shipped on a first-come, first-served basis, and any orders for items that are already sold-out will be promptly returned/refunded. To take advantage of this offer, you must indicate that you are taking advantage of Deal #1 on your order or over the phone. Please note that the special introductory offer for WordStar and MailMerge is over, and that they have returned to their regular price of $395 and $249, respectively.

Products we have that are not listed in the LSI catalog:

PowerMail Plus, Model 1/3 version by PowerSoft	Sugg. retail $15Ø
PowerMail Plus, Model 4 version by PowerSoft	Sugg. retail $15Ø
ZSHELL, for LDOS 5.1 from MISOSYS	Sugg. retail $ 4Ø

The 6.x versions of EDAS (PRO-CREATE) and DSMBLR III (PRO-DUCE)

For the most part, these and all the other discontinued products are fine products from excellent companies. Our decision to discontinue, marketing, promotion, support, endorsing, etc. of non-LSI products should in no way reflect on the quality of these products or companies.

DEAL #2: Special LSI 3Ø% discount. For orders postmarked between March 1st, 1984 and March 15, 1984, take a 3Ø% discount on any LSI-manufactured product. This includes LDOS, FED II, LED and all our other excellent LDOS support products (don't forget diskDISK). This offer is not good in conjunction with any other offer, but you may have both Deal #1 and Deal #2 items on the same order. This offer is also good on phone orders placed in this period, but again you must indicate Deal #2 over the phone (or on your order).

DEAL #3: FREE LSI Journal Issues. With any order totaling $5Ø or more (net amount after discount), get the previous Volume 2 LSI Journal/LDOS Quarterly issues at no charge. This includes Volume 2, numbers 1 through 4. Number 4 is in short supply, so if we run out, you will get only numbers 1 through 3. Take advantage of this order quickly if you need all four. Again, you must note this special offer on your order (or over the phone). This offer is good until we run out of back issues.

DEAL #4: How about the full LDOS 5.1.4 operating system at 1/2 price? That's right, <u>FULL</u> LDOS 5.1.4 for just $64.5Ø. This special offer is available to CONVERTS from <u>both</u> NEWDOS and DOSPLUS. Now is the time for all your friends to convert to the power of LDOS. Here's how it works:

From February 1, 1984 until June 3Ø, 1984, LSI will provide the complete LDOS 5.1.4 system for the price of $64.5Ø, plus $5.ØØ shipping and handling, to anyone that trades in their NEWDOS-8Ø or DOSPLUS 3.4/3.5/4 operating system. To take advantage of this special, just send your original MASTER disk and MANUAL to LSI along with $69.5Ø, and we will rush a fresh LDOS 5.1.4 operating system right out. This offer is not good in conjunction with any other offer. Don't forget to mark your order as Deal #4 or the LDOS Trade-In offer. Oh, one more thing... I will even take our own LDOS 5.Ø systems in trade!

DEAL #5: We have several extra Radio Shack Hard Disk systems. These are in good, used condition, but are being sold on an as-is basis due to the fact that we do not have any of the manuals or cables (other than the power cable). Because of this, we recommend these to experienced users who already own a RS HD system. The price, you ask? Well, these are a steal at $995 for a primary or $695 for a secondary, plus shipping and handling. None of the above discounts apply, but we will throw in Deal #3.

All these "Deals" are good only directly from LSI, and <u>not</u> from any of our dealers.

In the future, LSI will again market software not created by LSI, hopefully by the middle of 1984. When we do, these will be very carefully selected products that are manufactured and supported by LSI, even though originally written outside. We will no longer offer products that do not bear the LSI name.

We had originally planned a new catalog for January '84, but due to needed product line changes, including the changes outlined above, we have decided to wait until June of 1984 to publish our new catalog. This will be sent to all registered LSI customers at no charge. We think our new catalog will be well worth the wait. At that time we will be implementing many new policies and pricing changes along with the revamped product line. For the time being, (the first half of '84) all LSI prices and policies will remain as stated in out current catalog.

Here's an interesting thought: "Beware of pre-release software". When a software company sends out BETA test (the second test phase) copies of software, they sometimes appear on the "Underground Software Exchange" overnight. Beware-- if anyone offers you a test copy of a new or updated product, you may be getting a free time bomb. That product is in BETA testing because the producing company is still in the process of locating and correcting errors. I know of one incidence of a fellow that received a clandestine copy of an update to a popular spread-sheet program. He was very excited about this find because this was the product he used everyday in his business, and now he had "THE HOT NEW VERSION". (Note: Apparently he never purchased the original version in the first place.)

After about three days of playing with it, he had managed to destroy all of his existing data files (several hundreds of hours of work). He called the company and they very nicely told him to go to hell. He was not a legitimate owner of their product in the first place, had no right to have the BETA product he obtained, and found out that the product was expected to destroy his files. After all, it was a BETA TEST version and not a released product. So beware, it may not be a good idea to be the first on your block with a neat, new product, if that is before it is even released.

For those who are interested, the current version of the TRSDOS 6.x system for the Model 4 and 4P is Ø6.Ø1.Ø1, and should be available from your local RS store. I think its a freebie. They should also have the hard disk drivers and formatter for TRSDOS 6.X (it is available off-the-shelf in the Midwest Computer Centers, anyway). The latest version of LDOS for the Model 1, 3 and MAX-8Ø is 5.1.4 with file dates of 1Ø/Ø1/83.

The Model 2/12 version of TRSDOS 6.x is complete (and in final testing) as of this writing. This will be known as LS-DOS 6.2.Ø (or TRSDOS if purchased through Tandy). The pricing and public availability of this product are still in question. If you are interested in this product, please contact LSI in March of 1984 for complete and final details. Whatever the final decisions may be, you WILL be able to purchase this product in April of 1984.

This new product will allow for complete transportability of software between the Models 4/4P and 2/12, along with any other machine that has 6.x implemented for it. A bit of caution here--- you MUST have used ONLY the official information contained in the Model 4 technical manual (26-211Ø) from Tandy, or information distributed by LSI when interfacing to the 6.x system to achieve this compatibility. Use of any other source of information on the 6.x system will most likely result in current or future incompatibilities. At this time, LSI and Tandy are the only official sources regarding technical specification for the TRSDOS 6.x and LS-DOS 6.x systems.

New Product Announcements

Remember LED, the LDOS EDitor? This is the official LDOS 5.1 Text Editor, and is used here at LSI for program source code maintenance, KSM file editing, and many other editing needs. Well, LED has been vastly enhanced and is now one of the best full screen editors available for your TRS-8Ø Model 4/4P (and in the future 2/12). LED is available now for $99 plus $3 shipping and handling as LS-LED for the 6.X operating system ONLY. We will NOT be doing a Model III version of this product (there just isn't enough room). If you use your Model 4 for programming, you will love this powerful text editor.

Note that LS-LED is not a word processor, but is a flexible, easy to use screen oriented text editor. LS-LED is capable of doing most word processor type functions, but there are no print formatting facilities, or indeed any printing capabilities (Of course, the DOS library command LIST with the "print" parameter can be used for hardcopy, if desired). Two major features that have been added to LS-LED are: writing a marked block to disk, and the insertion of the contents of a disk file at the current cursor position. With these two commands, the production and maintenance of subroutine libraries and other forms of "boiler-plate" operations become a snap.

Of course, the original version of LED for LDOS 5.1 is still available for $29 plus $3 shipping and handling.

LSI is now shipping LS-Host/Term, a comprehensive communications utility for the 6.X operating system. This package allows a Model 4 running under 6.x to emulate an ADDS-25 terminal, and provides for error-free file transfer between 6.x systems, or other systems supporting the Modem7 protocol. The host system may be unattended during the transfer. Speaking of hosting, the host capabilities of this package are very sophisticated, and include password protection and remote cursor positioning using two different protocols. LS-Host/Term is only $199 plus $3 shipping and handling.

FED, the most popular "zapping" program around for the LDOS system will be available for the IBM (or any MS-DOS 2.x, PC compatible) machine in March or April of 1984. The price will be $99, plus $3 shipping and handling. This is a very valuable tool for any serious user of an MS-DOS machine.

LSI will also introduce the most versatile data handling and management system available for the PC-DOS (and MS-DOS) world. This product will be announced in June of 1984, and more details will be available at that time.

LSI Quick Hint #1

Question: How can I move a file from LDOS 5.1 or TRSDOS 6.x to TRSDOS 1.3?

Quite simple, actually. Here is the procedure: First, under LDOS 5.1 or TRSDOS 6.x, format a five inch diskette as thirty-five track, one side, single-density. Now, copy the desired file to this special diskette. You may now re-boot under TRSDOS 1.3, and the TRSDOS 1.3 "CONVERT" utility will read this disk just as though it were a Model 1 TRSDOS 2.3 diskette.

Here's the catch-- some Model 4 computer systems have a newer type of controller board, and will not format properly in single-density with LDOS 5.1.3 or before, or TRSDOS 6.Ø. This problem was "bypassed" in software, and these machines will format correctly with LDOS 5.1.4 and TRSDOS Ø6.Ø1.Ø1.

EZEDIT/FLT - a command line edit utility

by Graham M Brown, 2Ø Paddock Close, Castle Donnington, Derby DE7 2JW, England

How many times have you typed in a long command line from DOS, only to find that you spelled a parameter incorrectly, and DOS asks you to do it all again? Well certainly I have done it enough times, and it is a nuisance. So EZ-Edit was born. Imagine typing:

```
FILTER *PR PR(M=1Ø,I=1Ø,C=8Ø,L*6Ø,P=66,F,T)
```

There is an error in the "L" parameter. With EZ-Edit, just type in <SHIFT><CLEAR><O> and your command comes back on the screen - position the flashing cursor over the character in error, correct it, press <ENTER>, and bingo!

EZ-Edit is a keyboard filter which intercepts a <SHIFT><CLEAR><O> and allows you to edit the previously entered command line. The left and right arrows move through the line, and extend it if required. Any other key will overtype existing text. <ENTER> will cancel editing, and process the command as shown. <SHIFT><CLEAR> will delete the command from the cursor to the right, and then execute the remainder. <BREAK> will return to DOS ready. In the above example, if the cursor was over the first open bracket, pressing <SHFT><CLR> would result in the command **FILTER *PR PR.**

Just as KSM does not work properly with MINIDOS (see LDOS Quarterly, Jan '83), EZ-Edit has a similar effect on the "R" command of MINIDOS. I have developed a similar patch for MINIDOS when EZ-Edit is running. EZ-Edit will perform this alteration as a temporary patch in memory when it loads. This program will, of course, work without MINIDOS. EZ-Edit takes less than 256 bytes when relocated in high memory.

EZ-Edit requires KI/DVR to be present and active. This program should work under all 5.1 releases of LDOS, but has only been tested on 5.1.3 and 5.1.4. Originally, the program was written to accept <clear><shift><E>, but this was changed to prevent a conflict with KSMPLUS. If desired, the value on line 234Ø may be changed to any other valid character. The value (EF) is also underlined in this BINHEX listing.

```
Ø5Ø6 4544 4954 2Ø2Ø Ø1Ø2 ØØ52 D5DD E13A 25Ø1 FE49 2Ø48 2111 4422 A252 22AA 5221 2B44
229Ø 5222 E152 2125 4222 F253 228F 5422 9F54 22F6 5221 BE42 2298 5222 FØ52 2199 4222
A254 2189 4222 6252 218A 4222 1253 21D5 ØØ22 7852 212Ø Ø122 7F52 2115 4ØAF ED52 C217
5321 1C53 CD67 4421 1F44 CB66 CAØE 53CB 6E28 1E21 AC53 CD67 442A FA4D E511 D9ØØ 1936
12E1 1124 Ø119 EB21 Ø453 Ø1ØA ØØED BØED 5B16 4Ø3A ØF43 CB6F 28Ø4 ED5B BE43 ED53 D953
Ø1E1 ØØ2A 494Ø 5D54 AFED 4222 494Ø 23E5 ED53 CE53 11Ø9 ØØ19 22Ø8 5422 1754 221B 54E1
E511 ØAØØ 1922 ØD54 2232 54E1 E511 ØBØØ 1922 1154 2214 54D1 D521 CC53 EDBØ E1F3 3AØF
43CB 6F2Ø Ø8DD 75Ø1 DD74 Ø218 Ø322 BE43 FBØ6 3E21 1843 3E2Ø 7723 1ØFC 3EØD Ø1Ø2 ØØ53
77C3 2D4Ø 211Ø ØØED 7AF9 E1C9 ØØØØ 2175 53CD 7B44 C33Ø 4Ø21 8D53 18F5 455A 2D45 4449
542Ø 2D2Ø 4C44 4F53 2Ø63 6F6D 6D61 6E64 2Ø6C 696E 652Ø 6564 6974 6F72 2E2Ø 5665 7273
696F 6E2Ø 352E 31ØA 2863 292Ø 3139 3833 2Ø62 792Ø 472E 4272 6F77 6E2E 2Ø41 6C6C 2Ø72
6967 6874 732Ø 7265 7365 7276 6564 ØDØA 4B49 2F44 5652 2Ø6E 6F74 2Ø69 6E73 7461 6C6C
6564 2Ø21 ØDØA 4669 6C74 6572 2Ø4F 4E4C 592Ø 7573 696E 672Ø 2A4B 492Ø 6465 7669 6365
21ØD ØA4D 494E 4944 4F53 2Ø64 6574 6563 7465 642Ø 2D2Ø 7Ø61 7463 6869 6E67 2E2E 2EØD
18ØA ØØØØ Ø445 6469 74ØØ ØØØØ CDØØ ØØFE EFCØ E5D5 C5DD E5ED 5B2Ø 4Ø7B E6CØ 5FED 532Ø
4ØD5 D521 1843 7EFE ØD28 Ø623 CD33 ØØ18 F53E Ø1AF ØØ54 ØFCD 33ØØ D13E 5F32 D553 6F1A
32D6 53AD 32D7 5321 D753 3AD5 53AE 32D5 5312 Ø15Ø Ø1D5 CD2B ØØD1 B72Ø Ø5ØB 78B1 2ØF3
28E3 F53A D653 12F1 FEØ8 2823 FEØ9 2815 FEØD 2835 FE1F 2821 FEØ1 285A FE2Ø 38B7 FE8Ø
3ØB3 127B E63F FE3F 3ØØ1 1318 A87B E63F FEØ1 38Ø1 1B18 9E3E 1ED5 ED53 2Ø4Ø CD33 ØØ3E
ØFCD 33ØØ D17B F63F 5F1A 1BFE 2Ø28 FA13 13ED 532Ø 4Ø7B E63F 4FØ6 ØØE1 1118 43ED BØ3E
ØD12 CD33 ØØDD E1C1 D1E1 2118 43C3 Ø544 E1DD E1C1 D1E1 C32D 4ØØ2 Ø2ØØ 52
```

```
ØØ1ØØ          TITLE   '<EZEDIT/FLT by: G M Brown>'
ØØ11Ø ;*=*=*
ØØ12Ø ;Some code alteration is necessary in MINIDOS/FLT for
ØØ13Ø ;the "R" command to function properly. So EZ-Edit
ØØ14Ø ;should be installed AFTER MINIDOS . Should MINIDOS be
ØØ15Ø ;installed after EZ-Edit, then the "R" command is not
ØØ16Ø ;guaranteed to work at all.
ØØ2ØØ @KBD     EQU     ØØ2BH
```

```
ØØ21Ø @DSP     EQU     ØØ33H
ØØ22Ø KIDVR    EQU     4Ø16H
ØØ23Ø CURSOR   EQU     4Ø2ØH
ØØ24Ø @EXIT    EQU     4Ø2DH
ØØ25Ø @ABORT   EQU     4Ø3ØH
ØØ26Ø HIGH1$   EQU     4Ø49H
ØØ27Ø HIGH3$   EQU     4411H
ØØ28Ø SFLAG1$ EQU     43ØFH
ØØ29Ø SFLAG3$ EQU     442BH
ØØ3ØØ INBUF1$ EQU     4318H
ØØ31Ø INBUF3$ EQU     4225H
ØØ32Ø KIJCL1$ EQU     43BEH
ØØ33Ø KIJCL3$ EQU     42BEH
ØØ34Ø @CMNDI1 EQU     44Ø5H
ØØ35Ø @CMNDI3 EQU     4299H
ØØ36Ø DFLAG1$ EQU     441FH
ØØ37Ø DFLAG3$ EQU     4289H
ØØ38Ø @DSPLY   EQU     4467H
ØØ39Ø @LOGOT1 EQU     447BH
ØØ4ØØ @LOGOT3 EQU     428AH
ØØ42Ø ;The first part of the coding loads an initial message
ØØ43Ø ;checks that *KI has been referenced, and  for KI/DVR
ØØ44Ø ;and MINIDOS being active.
ØØ46Ø          ORG     52ØØH            ;A good place to start
ØØ47Ø          PUSH    DE               ;Put *KI DCB into
ØØ48Ø          POP     IX               ;IX register for later
ØØ5ØØ ;        This section of code corrects all model specific
ØØ51Ø ;        references in the code.
ØØ53Ø          LD      A,(Ø125H)
ØØ54Ø          CP      'I'              ;it's a 3 if true
ØØ55Ø          JR      NZ,CONT          ;go if Mod 1
ØØ56Ø          LD      HL,HIGH3$
ØØ57Ø          LD      (M1),HL
ØØ58Ø          LD      (M2),HL
ØØ59Ø          LD      HL,SFLAG3$
ØØ6ØØ          LD      (M3),HL
ØØ61Ø          LD      (M4),HL
ØØ62Ø          LD      HL,INBUF3$
ØØ63Ø          LD      (M5),HL
ØØ64Ø          LD      (M6),HL
ØØ65Ø          LD      (M7),HL
ØØ66Ø          LD      (M15),HL
ØØ67Ø          LD      HL,KIJCL3$
ØØ68Ø          LD      (M8),HL
ØØ69Ø          LD      (M9),HL
ØØ7ØØ          LD      HL,@CMNDI3
ØØ71Ø          LD      (M1Ø),HL
ØØ72Ø          LD      HL,DFLAG3$
ØØ73Ø          LD      (M11),HL
ØØ74Ø          LD      HL,@LOGOT3
ØØ75Ø          LD      (M12),HL
ØØ76Ø          LD      HL,ØØD5H
ØØ77Ø          LD      (M13),HL
ØØ78Ø          LD      HL,Ø12ØH
ØØ79Ø          LD      (M14),HL
ØØ8ØØ CONT     LD      HL,4Ø15H         ;See if *KI specified
ØØ81Ø          XOR     A                ;in the filter command
ØØ82Ø          SBC     HL,DE
ØØ83Ø          JP      NZ,WRONGDV       ;error if not.
ØØ84Ø          LD      HL,LOGMSG        ;Point to message....
ØØ85Ø          CALL    @DSPLY           ;..and print it.
ØØ86Ø          LD      HL,DFLAG1$       ;System flag area
ØØ87Ø M11      EQU     $-2
```

```
00880          BIT     4,(HL)           ;Test for KI/DVR, and
00890          JP      Z,KINOTON        ;error if NOT set
00900          BIT     5,(HL)           ;Test for MINIDOS, and
00910          JR      Z,MDISOFF        ;bypass zapping if NOT there.
00930 ;If MINIDOS active, then it has to be zapped, otherwise
00940 ;this bit is skipped over.
00960          LD      HL,ZAPMSG        ;Point to message....
00970          CALL    @DSPLY           ;...and display it.
00980          LD      HL,(4DFAH)       ;Get MINIDOS address
00990          PUSH    HL               ;and save
01000          LD      DE,00D9H         ;Offset for zap
01010 M13      EQU     $-2
01020          ADD     HL,DE            ;get actual address, and
01030          LD      (HL),12H         ;zap MINIDOS
01040          POP     HL               ;restore start address
01050          LD      DE,0124H         ;for next offset
01060 M14      EQU     $-2
01070          ADD     HL,DE            ;get actual address, and
01080          EX      DE,HL            ;put in DE register pair
01090          LD      HL,TABLE         ;Point to zap table
01100          LD      BC,0AH           ;10 bytes to zap, so..
01110          LDIR                     ;...do it !
01130 ;The next part of the code intercepts the *KI driver
01140 ;address, and stores EZ-Edit's start there. The old
01150 ;contents are loaded into EZ-Edit for continuation.
01170 ;High memory is also altered.
01190 ;EZ-Edit does not bother to indicate to the system (or
01200 ;other programs) that it has loaded. Indeed, by finding
01210 ;the proper header, it would be possible to get EZ-Edit
01220 ;to locate itself at the same address after a *KI reset.
01230 ;This has been left for you to do if you require.
01250 MDISOFF LD     DE,(KIDVR)       ;Get *KI driver address
01260          LD      A,(SFLAG1$)      ;See if JCL is
01270 M3       EQU     $-2
01280          BIT     5,A              ;active at present
01290          JR      Z,NO_DO          ;skip if not, else
01300          LD      DE,(KIJCL1$)     ;change DE to KIJCL$
01310 M8       EQU     $-2
01320 NO_DO    LD      (VECTAD),DE      ;Store in EZ-Edit
01330          LD      BC,LAST-START    ;Get length of filter
01340          LD      HL,(HIGH1$)      ;HL = Present High Memory
01350 M1       EQU     $-2
01360          LD      E,L              ;Put into register
01370          LD      D,H              ;pair DE
01380          XOR     A                ;Clear the carry
01390          SBC     HL,BC            ;get new High memory,
01400          LD      (HIGH1$),HL      ;and set it.
01410 M2       EQU     $-2
01420          INC     HL               ;Point to EZ-Edit start
01430          PUSH    HL               ;and save it
01440          LD      (OLDHI),DE       ;Put old HIGH$ in filter
01450          PAGE    OFF
01480 ;Relocate calls and jumps in the filter
01490 ;Restore *KI DCB driver address, and relocate
01500 ;EZ-Edit to below HIGH$
01520          LD      DE,ONOFF-START   ;Get offset
01530          ADD     HL,DE            ;add to new start
01540          LD      (MODIFY2),HL     ;and modify
01550          LD      (MODIFY6),HL
01560          LD      (MODIFY7),HL
01570          POP     HL               ;recover start
01580          PUSH    HL               ;and keep going
01590          LD      DE,CONTENT-START
```

```
        ADD     HL,DE
        LD      (MODIFY3),HL
        LD      (MODIFY8),HL
        POP     HL
        PUSH    HL
        LD      DE,XORVAL-START
        ADD     HL,DE
        LD      (MODIFY4),HL
        LD      (MODIFY5),HL
        POP     DE              ;DE = Start of EZ-Edit
        PUSH    DE              ;save it.  DE = To
        LD      HL,START        ;and HL = From
        LDIR                    ;BC = Count, so move it.!
        POP     HL              ;recover EZ-Edit's start
        DI                      ;Don't interrupt a minute
        LD      A,(SFLAG1$)     ;Test for JCL again
M4      EQU     $-2
        BIT     5,A
        JR      NZ,DO_ON        ;Skip if ACTIVE, else
        LD      (IX+01H),L      ;Load KIDCB with address
        LD      (IX+02H),H      ;of EZ-Edit
        JR      OUT
DO_ON   LD      (KIJCL1$),HL    ;JCL, so load KIJCL$
M9      EQU     $-2
OUT     EI                      ;Interrupts on, and
        LD      B,62            ;how many spaces
        LD      HL,INBUF1$      ;point to inbuf$
M15     EQU     $-2
        LD      A,20H           ;space
HERE    LD      (HL),A          ;store it
        INC     HL              ;point to next
        DJNZ    HERE            ;go until done
        LD      A,0DH           ;CR
        LD      (HL),A          ;put at end of buffer
        JP      @EXIT           ; now back to DOS
;Table of zaps for MINIDOS zapping routine
TABLE   DB      21H,10H,00H,0EDH,7AH,0F9H,0E1H,0C9H,00H,00H
;Error if KI/DVR not established
KINOTON LD      HL,KIOFFMS        ;Point to message
GO_OUT  CALL    @LOGOT1           ;LOG and DISPLAY
M12     EQU     $-2
        JP      @ABORT            ;Abnormal exit to DOS
;Error if *KI not referenced
WRONGDV LD      HL,NOTKIMS        ;Point to message
        JR      GO_OUT            ;Log, display, and DOS
;               Messages
LOGMSG  DB      'EZ-EDIT - LDOS command line editor. Version 5.1',0AH,'(c) 1983
by G.Brown. All rights reserved',0DH
KIOFFMS DB      0AH,'KI/DVR not installed !',0DH
NOTKIMS DB      0AH,'Filter ONLY using *KI device!',0DH
ZAPMSG  DB      0AH,'MINIDOS detected - patching...',0DH
;EZ-Edit : The actual relocated filter
;The filter conforms to the LDOS standard header.
START   JR      BEGIN
OLDHI   DW      0000H
        DB      4,'Edit'
;Local storage area
ONOFF   DB      00H     ;Flash character store
CONTENT DB      00H     ;The contents of DE before flash
XORVAL  DB      00H     ;The XOR character for the flash
BEGIN   CALL    0000H   ;Call to KI/DVR, and MINIDOS
VECTAD  EQU     $-2
        CP      0EFH    ;See if SHIFT+CLEAR+O
```

```
Ø235Ø          RET     NZ          ;Ret if not
Ø237Ø ;Got a SHIFT+CLEAR+O here
Ø239Ø          PUSH    HL                  ;Save what we use
Ø24ØØ          PUSH    DE
Ø241Ø          PUSH    BC
Ø242Ø          PUSH    IX
Ø243Ø          LD      DE,(CURSOR)         ;Get the cursor location
Ø244Ø          LD      A,E                 ;and make sure that
Ø245Ø          AND     ØCØH                ;its at the far left of
Ø246Ø          LD      E,A                 ;the screen.
Ø247Ø          LD      (CURSOR),DE
Ø248Ø          PUSH    DE                  ;Save the address of the
Ø249Ø          PUSH    DE                  ;line - TWICE
Ø251Ø ;Get the last command from INBUF$, and put on screen
Ø252Ø ;Turn the cursor off afterwards.
Ø254Ø          LD      HL,INBUF1$          ;Point to input buffer
Ø255Ø M5       EQU     $-2
Ø256Ø GETLOOP  LD      A,(HL)              ;and get all characters
Ø257Ø          CP      ØDH                 ;up to but NOT including
Ø258Ø          JR      Z,GOT_CR            ;a CR, and put on the
Ø259Ø          INC     HL                  ;screen
Ø26ØØ          CALL    @DSP
Ø261Ø          JR      GETLOOP
Ø262Ø GOT_CR   LD      A,ØFH               ;Cursor OFF now
Ø263Ø          CALL    @DSP
Ø264Ø          POP     DE                  ;Restore line address
Ø265Ø          PAGE    OFF
Ø267Ø ;Scan for an input, and flash the character with the
Ø268Ø ;cursor (CHR$(95)). Allowable inputs are:
Ø269Ø ;LEFT ARROW  - backspace without erase
Ø27ØØ ;RIGHT ARROW - cursor right without erase
Ø271Ø ;SHIFT+CLEAR - erase all from cursor to the right and
Ø272Ø ;              process the command to the left.
Ø273Ø ;ENTER       - process the line as is. Characters to the
Ø274Ø ;              left and right of the cursor make up the
Ø275Ø ;              command
Ø276Ø ;BREAK       - Back to DOS
Ø278Ø FLASH    LD      A,5FH               ;Cursor character
Ø279Ø          LD      (ONOFF),A           ;store it
Ø28ØØ MODIFY2 EQU      $-2
Ø281Ø          LD      L,A                 ;and save in L
Ø282Ø          LD      A,(DE)              ;Get character in DE
Ø283Ø          LD      (CONTENT),A         ;and save for later
Ø284Ø MODIFY3 EQU      $-2
Ø285Ø          XOR     L                   ;XOR to get flash value
Ø286Ø          LD      (XORVAL),A          ;and save it.
Ø287Ø MODIFY4 EQU      $-2
Ø289Ø TEXTLP   LD      HL,XORVAL           ;Point to store area
Ø29ØØ MODIFY5 EQU      $-2
Ø291Ø          LD      A,(ONOFF)           ;Get one character
Ø292Ø MODIFY6 EQU      $-2
Ø293Ø          XOR     (HL)                ;XOR
Ø294Ø          LD      (ONOFF),A           ;save back for flash
Ø295Ø MODIFY7 EQU      $-2
Ø296Ø          LD      (DE),A              ;and put on screen
Ø297Ø          LD      BC,15ØH             ;Timer for scanning
Ø298Ø LOOP     PUSH    DE                  ;save DE
Ø299Ø          CALL    @KBD                ;and look at the keyboard
Ø3ØØØ          POP     DE                  ;get DE again
Ø3Ø1Ø          OR      A                   ;Any input ???
Ø3Ø2Ø          JR      NZ,INPUT            ;Jump if so, else....
Ø3Ø3Ø          DEC     BC                  ;see if BC = Ø
Ø3Ø4Ø          LD      A,B
```

```
Ø3Ø5Ø         OR      C
Ø3Ø6Ø         JR      NZ,LOOP          ;Loop if not, else...
Ø3Ø7Ø INPUT   JR      Z,TEXTLP         ;no input = back to flash
Ø3Ø8Ø         PUSH    AF               ;save input
Ø3Ø9Ø         LD      A,(CONTENT)      ;restore contents to
Ø31ØØ MODIFY8 EQU     $-2              ;original character
Ø311Ø         LD      (DE),A
Ø312Ø         POP     AF               ;recover input, and
Ø313Ø         CP      Ø8H              ;is it LEFT ARROW ??
Ø314Ø         JR      Z,LARROW
Ø315Ø         CP      Ø9H              ;RIGHT ARROW ??
Ø316Ø         JR      Z,RARROW
Ø317Ø         CP      ØDH              ;ENTER ??
Ø318Ø         JR      Z,ENTPRES
Ø319Ø         CP      1FH              ;SHIFT+CLEAR ??
Ø32ØØ         JR      Z,CLEARPR
Ø321Ø         CP      Ø1H              ;BREAK  ??
Ø322Ø         JR      Z,BREAK
Ø323Ø         CP      2ØH              ;Is it ASCII 2ØH-7FH
Ø324Ø         JR      C,FLASH
Ø325Ø         CP      8ØH
Ø326Ø         JR      NC,FLASH
Ø327Ø         LD      (DE),A           ;if so, then put on
Ø328Ø RARROW  LD      A,E              ;the screen. Test for
Ø329Ø         AND     3FH              ;extreme right, else
Ø33ØØ         CP      3FH              ;increment DE (type to
Ø331Ø         JR      NC,NOADD         ;the right).
Ø332Ø         INC     DE
Ø333Ø NOADD   JR      FLASH            ;Back for next input.
Ø335Ø ;Deal with the input. Editing allowed on one line only.
Ø337Ø LARROW  LD      A,E              ;Test for column 1
Ø338Ø         AND     3FH              ;If less, then cannot
Ø339Ø         CP      1                ;move further left.
Ø34ØØ         JR      C,NODEC
Ø341Ø         DEC     DE               ;else DECrement
Ø342Ø NODEC   JR      FLASH
Ø343Ø CLEARPR LD      A,1EH            ;Erase to end of line
Ø344Ø         PUSH    DE               ;Save DE for a moment
Ø345Ø         LD      (4Ø2ØH),DE       ;tell the cursor where,
Ø346Ø         CALL    @DSP             ;and clear the line.
Ø347Ø         LD      A,ØFH            ;Cursor OFF
Ø348Ø         CALL    @DSP
Ø349Ø         POP     DE               ;and restore DE
Ø351Ø ;How long is the command ?? - go to right of line, and
Ø352Ø ;find first non blank character on the left.
Ø354Ø ENTPRES LD      A,E              ;Point DE to end of
Ø355Ø         OR      3FH              ;the line
Ø356Ø         LD      E,A
Ø357Ø BKLOOP  LD      A,(DE)           ;and look back....
Ø358Ø         DEC     DE
Ø359Ø         CP      2ØH
Ø36ØØ         JR      Z,BKLOOP
Ø361Ø         INC     DE
Ø362Ø         INC     DE               ;DE = space after command
Ø363Ø         LD      (CURSOR),DE      ;load the cursor
Ø365Ø ;Move the new command into INBUF$, and execute via
Ø366Ø ;@CMNDI address.
Ø368Ø         LD      A,E              ;Get column number into A
Ø369Ø         AND     3FH
Ø37ØØ         LD      C,A              ;Load length of the
Ø371Ø         LD      B,Ø              ;command into BC
Ø372Ø         POP     HL               ;Point HL to line start
Ø373Ø         LD      DE,INBUF1$       ;DE = Destination
```

```
Ø374Ø M6      EQU     $-2
Ø375Ø         LDIR                    ;Move the command
Ø376Ø         LD      A,ØDH           ;finish off with a CR
Ø377Ø         LD      (DE),A          ;in the buffer, and....
Ø378Ø         CALL    @DSP            ;on the screen.
Ø379Ø         POP     IX              ;Restore original
Ø38ØØ         POP     BC              ;register values
Ø381Ø         POP     DE
Ø382Ø         POP     HL
Ø383Ø         LD      HL,INBUF1$      ;Point to new command
Ø384Ø M7      EQU     $-2
Ø385Ø         JP      @CMNDI1         ;and execute it.
Ø386Ø M1Ø     EQU     $-2
Ø387Ø BREAK   POP     HL              ;Keep the stack tidy
Ø388Ø         POP     IX
Ø389Ø         POP     BC
Ø39ØØ         POP     DE
Ø391Ø         POP     HL
Ø392Ø         JP      @EXIT           ;and exit to DOS
Ø393Ø LAST    EQU     $
Ø394Ø         END     52ØØH
```

Mr. Brown also sent us the following program to alter PR/FLT parameters "on the fly". Unfortunately, we don't have room to run the source code. If you desire a hardcopy of the source, send us a request along with a self-addressed, stamped envelope (large, with 37 cents postage).....

Although PR/FLT as supplied with LDOS is a very fine general purpose printer filter, I have found it somewhat restricting in that there is no facility for changing any of the parameters once set. In fact, should you wish to change the margin, it would be necessary first to issue a RESET *PR, followed by re-filtering PR/FLT with your new parameter values in the command line (as normal). This is all very well, but if you had your printer output routed, or indeed had other printer filters installed, the RESET would ruin everything. Certainly from my point of view, I needed a program that would find where PR/FLT was in memory and then change any of the parameters that I wanted.

The program is called from DOS ready by entering the command:

```
   PRPARM (parm,parm,parm,...)
```

where parameters are nearly the same as for PR/FLT:

MARGIN	number of spaces for the left margin
INDENT	-:- -:- indent on line wrap
CHARS	number of characters per line
LINES	number of printed lines per page required
PAGE	page length in lines (at 6 lines per inch)
XLATE	use the format XLATE=X'aabb' (see the PR/FLT documentation)
FF	issue a form feed

Abbr: MARGIN=M,INDENT=I,CHARS=C,LINES=L,PAGE=P,XLATE=X

Example: PRPARM (M=3Ø,C=5Ø,FF)

Run PRPARM and reset the left margin to 3Ø and only print 5Ø characters on each line. Issue a form feed character to the *PR device as well.

A typical display would be:

```
PRPARM - parameter modifier for PR/FLT. Version 5.1.3
Copyright (c) 1983 by Graham M Brown.
```

```
The current PR/FLT parameters are :
MARGIN = Ø1Ø spaces
INDENT = Ø36 spaces
CHARS  = Ø82 characters per line (Ø = no count made.)
PAGE LENGTH = Ø66 lines
LINES/PAGE  = Ø66 lines
ØØØ (dec) is being translated to ØØØ (dec).
```

By typing in PRPARM only, the current parameters as stored in PR/FLT are displayed and no alterations are done. A check is made for LDOS 5.1.x and also that PR/FLT is in fact installed. The program is written for the Model 1 and 3. Here is the BINHEX code:

```
Ø5Ø6 5Ø52 5Ø41 524D Ø1Ø2 ØØ52 E521 6353 CD67 443A 25Ø1 FE49 2Ø31 211F 4422 4Ø52 2189
4222 4852 2154 4422 6C52 3EØ1 32C8 5221 8A42 225E 533A FF52 3D32 FF52 32B7 523A Ø853
3D32 Ø853 32BA 523A 3E4Ø FE51 C256 5321 1F44 CB5E CA5A 53FD 2AF6 4DE1 7EFE ØD28 Ø7FE
2Ø2Ø ØB23 18F4 3EØD 32Ø7 55C3 CC52 116B 55CD 7644 C252 53DD 2125 4Ø21 FFFF CD1E 5328
Ø3FD 7717 21FF FFCD 1E53 28Ø3 FD77 1921 FFFF CD1E 5328 Ø3FD 771A 21FF FFCD 1E53 28Ø3
DD77 Ø521 FFFF CD1E 5328 Ø3DD 77Ø3 21FF FF23 7CB5 28Ø7 2BFD 746D FD75 7121 ØØØØ 7CB5
289E 3EØC CD3B ØØ3E ØØDD 77Ø4 26ØØ FD6E 1711 A554 CD2F 53FD 6E19 11BE 54CD 2F53 FD6E
1A11 8C54 CD2F 533A 284Ø 6F11 FE54 CD2F 533A 2A4Ø 6F11 3Ø55 CD2F 53FD 6E6D Ø1Ø2 ØØ53
113F 55CD 2F53 FD6E 7111 6Ø55 CD2F 5321 5854 CD67 4421 1D55 CD67 44C3 2D4Ø 2324 252Ø
2E7C B5C8 2B7D C964 ØØØA ØØØ1 ØØDD 2129 53AF DD46 Ø1DD 4EØØ B7ED 4238 Ø33C 18F9 Ø9C6
3Ø12 1379 FEØ1 C8DD 23DD 2318 E2E1 21Ø7 54DD 21C9 53DD 21E7 53CD 7B44 C33Ø 4ØØA 5Ø52
5Ø41 524D 2Ø2D 2Ø7Ø 6172 616D 6574 6572 2Ø6D 6F64 6966 6965 722Ø 666F 722Ø 5Ø52 2F46
4C54 2E2Ø 5665 7273 696F 6E2Ø 352E 312E 33ØA 8383 8383 8383 2Ø2Ø 2Ø43 6F7Ø 7972 6967
6874 2Ø28 6329 2Ø31 3938 332Ø 6279 2Ø47 7261 6861 6D2Ø 4D2Ø 4272 6F77 6E2E ØDØA ØA5Ø
6C65 6173 652Ø 7573 652Ø 4C44 4F53 2Ø35 2E31 2E33 2Ø6F 6E6C 792E ØDØA ØA5Ø 522F 464C
542Ø 6973 2Ø6E 6F74 2Ø79 6574 2Ø69 6E73 7461 Ø1Ø2 ØØ54 6C6C 6564 2Ø21 ØDØA ØA5Ø 6172
616D 6574 6572 2Ø65 7272 6F72 2Ø2D 2Ø74 7279 2Ø61 6761 696E 2Ø21 ØA41 6C6C 6F77 6162
6C65 2Ø7Ø 6172 616D 6574 6572 732Ø 6172 653A 2Ø4D 2C49 2C43 2C4C 2C5Ø 2C46 462C 616E
642Ø 584C 4154 45ØD ØA54 6865 2Ø63 7572 7265 6E74 2Ø5Ø 522F 464C 542Ø 7Ø61 7261 6D65
7465 7273 2Ø61 7265 2Ø3A ØAØA 2Ø2Ø 2Ø2Ø 2Ø4D 4152 4749 4E2Ø 3D2Ø 5858 582Ø 737Ø 6163
6573 ØA2Ø 2Ø2Ø 2Ø2Ø 494E 4445 4E54 2Ø3D 2Ø58 5858 2Ø73 7Ø61 6365 73ØA 2Ø2Ø 2Ø2Ø 2Ø43
4841 5253 2Ø2Ø 3D2Ø 5858 582Ø 6368 6172 6163 7465 7273 2Ø7Ø 6572 2Ø6C 696E 652Ø 283Ø
2Ø3D 2Ø6E 6F2Ø 636F 756E 742Ø 6D61 6465 2E29 ØA2Ø 2Ø2Ø 2Ø2Ø 5Ø41 4745 2Ø4C 454E 4754
482Ø 3D2Ø 5858 Ø1D6 ØØ55 582Ø 6C69 6E65 732Ø 2D2Ø 544F 462Ø 6861 732Ø 6265 656E 2Ø72
6573 6574 ØD2Ø 2Ø2Ø 2Ø2Ø 4C49 4E45 532F 5Ø41 4745 2Ø2Ø 3D2Ø 5858 582Ø 6C69 6E65 73ØA
2Ø2Ø 2Ø2Ø 2Ø58 5858 2Ø28 6465 6329 2Ø69 732Ø 6265 696E 672Ø 7472 616E 736C 6174 6564
2Ø74 6F2Ø 5858 582Ø 2864 6563 292E ØD4D 2Ø2Ø 2Ø2Ø 2Ø8C 524D 4152 4749 4E8C 5243 4841
5253 2Ø81 5243 2Ø2Ø 2Ø2Ø 2Ø81 5249 4E44 454E 5476 5249 2Ø2Ø 2Ø2Ø 2Ø76 524C 494E 4553
2Ø97 524C 2Ø2Ø 2Ø2Ø 2Ø97 525Ø 4147 452Ø 2ØA2 525Ø 2Ø2Ø 2Ø2Ø 2ØA2 5246 462Ø 2Ø2Ø 2ØBC
5258 4C41 5445 2ØAD 5258 2Ø2Ø 2Ø2Ø 2ØAD 52ØØ Ø2Ø2 ØØ52
```

SYNONYM -- The LDOS Command Line Synonym Processor

By Henry Melton, 2511 Dovemeadow Drive, Austin, TX 78744

The world of the professional programmer can get positively confusing at times. In the course of a single business day, I am required to deal with a half-dozen or so different operating systems, some quite modern and some positively ancient. And then I come home to work with my LDOS system . . . Not only do I have to constantly adjust to differing keyboards and screen formats, but more significantly, I have to remember which operating system command does what. Just to get a listing of files, my fingers might type LIST, DIR, FILES, CAT, LISTC, or L. Some of the newer systems are helping us poor users by allowing command synonyms to be used, so that more than one reserved word might be used for the same function. I decided to add that same capability to my LDOS.

The SYNONYM utility is an addition to the error trapping system of LDOS that allows for extended flexibility in operator actions for Model I and Model III TRS-8Ø systems running under LDOS 5.1. SYNONYM locates itself into high memory following the standard LDOS convention and is SYSGENable. Once loaded, it intercepts any PROGRAM NOT FOUND errors generated from keyboard or JCL command line inputs. Once the error is detected, a special text file SYNONYM/TXT is read and the match-and-substitute record lines in it are used to determine if the rejected command line has a valid substitute. If so, the reconstructed command line is placed in the command buffer and re-executed.

The result of SYNONYM processing is to add another level to the two existing levels of command interpretation. Under LDOS, the first word of a command line is first checked for an LDOS reserved word. If none of the reserved words match, then the word is treated as the name of a command file to be executed. With SYNONYM, if the first two levels fail, this third possibility for command information exists.

Like the JCL language, much of the utility of SYNONYM is up to the imagination of the user. The synonym library file can be created and modified with any word-processor that will produce a plain, un-numbered file of text lines.

Each record line in the file contains three items of information:

The first character of each line is a single numeric digit in the range of 1 through 9. This is the minimum number of characters that must be the same to indicate a match.

The second item is a word that is allowed as a valid synonym.

The third item is the remaining text on the line. This is the text that will replace the rejected command line. The special character '&' may be used to represent the text of the original command line after the first word.

Here are some examples:

Command line	SYNONYM record	Resulting Command line
fi /vc	2 files dir &	dir /vc
f	1 free free &	free
calc 365/12	4 calc lbasic ?&:cmd"S"	lbasic ?365/12:cmd"S"
cØ1 syn/mac	3 cØ1 copy &:Ø :1	copy syn/mac:Ø :1
dØ	2 dØ dir &:Ø	dir :Ø
dl sample	2 dl dir &:1	dir sample:1
bk work	2 bk copy & bk&:3	copy work bkwork:3
a syn	3 asm do it(@asm,name=&)	do it(@asm,name=syn)
ban	3 banner lbasic run"banner"	lbasic run"banner"

If there is no acceptable synonym, then the PROGRAM NOT FOUND error is displayed normally. If the resulting synonym command is itself invalid, the whole process repeats using the new command line as input.

Synonyms can be used within JCLs and synonyms may call JCLs. In fact, using a DO command line as a synonym may easily be the most powerful use of this utility. My personal JCL file (IT/JCL) is quite large, but remembering all the parameters and syntax considerations has been my main trouble. When I want to invoke a function of the system, I really don't want to puzzle over whether I am invoking a system function, a /CMD program, a LBASIC program or a JCL -- I want type the word and have it done. SYNONYM makes this a reality.

Another use of synonym is to execute one-liners from LBASIC. See CALC and REVTOF in the sample SYNONYM file to see what I mean. LBASIC is powerful. The ability to use its power at LDOS Ready is very handy at times. The limitations on this use is that the function has to fit within LBASIC and :CMD"S, leaving only 5Ø characters to get the job done. Anything longer has to route through a JCL or use a real /BAS program file. To keep screen clutter to a minimum, I patched LBASIC on my Model I system to eliminate the sign-on banner when LBASIC is executed.

```
. Patch to remove the execution banner from LBASIC - Model 1
X'541E'=ØØ ØØ ØØ ØØ ØØ ØØ

. Patch for Model 3
X'5446'=ØØ ØØ ØØ ØØ ØØ ØØ
```

Sample SYNONYM/TXT file -- Total line length should not exceed 8Ø characters. The resulting command line should not exceed 63 characters.

```
2 dirt dir &
2 files dir &
1 asm do it(@asm,name=&)
2 debug debug (e
3 out debug (n
3 off system (drive=3,disable)
3 onn system (drive=3,enable)
1 v dir /vc
1 d dir /aaa
1 bye do it(@back)
1 free free &
2 d1 dir :1 &
2 dØ dir :Ø &
3 device device
2 looker lbasic run"looker
3 cØ1 copy &:Ø :1
3 cØ3 copy &:Ø :3
1 hex list & (H,LRL=256)
3 nosynonym memory (add=x'4ØØd',word=X'4bcd')
5 synoff nos
2 peek memory (add=x'&')
3 zap purge & (S,I,Q=N)
4 hide attrib & (inv)
3 unhide attrib & (vis)
4 calc lbasic ?&:cmd"S"
3 tof lbasic lprint chr$(12);:cmd"s"
6 revtof lbasic for i=Ø to 65:lprint chr$(27)+chr$(13);:next i:cmd"S
3 jmp memory (go=X'&')
3 cls jmp 1C9
3 erase kill &
3 delete kill &
2 type list &
3 catalog dir &
4 page tof
3 num list & (num,A8
3 tab list & (num,tab,A8
3 slow filter *do dospeed
4 fast reset *do
3 bØ1 backup &:Ø :1
3 bØ3 backup &:Ø :3
5 nameØ attrib :Ø (name="&")
5 name1 attrib :1 (name="&")
7 format1 format :1 (name="&",dden,abs,mpw="PASSWORD")
4 sys1 system (system=1)
6 sysgen system (sysgen)
8 clonesys do it(@clonesys)
```

Here is the BINHEX code for SYNONYM. If you wish the "processed" DOS command to be displayed on execution, replace the ØØØØ ØØ with CD67 44.

```
Ø5Ø6 5359 4E4F 4E59 Ø1Ø2 ØØ52 21AD 52CD 6744 3A2D 4ØFE C32Ø Ø921 FC52 CD67 44C3 2D4Ø
3A25 Ø1FE 492Ø 2F21 1144 2251 5222 7952 2285 5221 1D42 2294 5222 A252 2322 9A52 22A8
5221 2542 229E 5322 7B54 2B22 F953 2199 4222 A453 DD21 2E53 2A49 4Ø22 5C53 1156 56B7
ED52 444D 3E16 DD6E ØØDD 66Ø1 235E 2356 EBØ9 EB72 2B73 DD23 DD23 3D2Ø E9ED 5B49 4Ø21
5656 Ø1FD Ø2ED B8ED 5349 4Ø2A ØD4Ø 226B 5321 6453 22ØD 4Ø3A Ø343 3292 542A Ø443 2293
543E C332 Ø343 218C 5422 Ø443 C32D 4Ø53 594E 4F4E 594D 2Ø2D 2D2Ø 4C44 4F53 2Ø63 6F6D
6D61 6E64 2Ø6C 696E 652Ø 7379 6E6F 6E79 6D2Ø 7Ø72 6F63 6573 736F 722Ø ØA43 6F7Ø 7972
6967 6874 6564 2Ø31 3938 332C 2Ø48 656E 7279 2Ø4D 656C 746F 6EØD 4F6E 6C79 Ø1Ø2 ØØ53
2Ø76 616C 6964 2Ø61 742Ø 4C44 4F53 2Ø52 6561 6479 2Ø2D 2Ø69 6E73 7461 6C6C 6174 696F
```

```
6E2Ø 6162 6F72 7465 6421 ØAØD 8D52 8A52 8C54 9652 EØ53 9C52 A452 8553 8A53 8F53 9653
9953 A653 A953 AC53 B553 B853 CD53 DØ53 EB53 2954 7754 18Ø8 ØØØØ Ø553 594E 4F4E F5FE
8628 Ø4F1 C3ØØ ØØ33 3333 33F1 FE5F 28Ø9 F53E 863B 3B3B 3B18 EA3B 3B3B 3B3B 3BCD A653
2ØDF CDCD 532Ø DACD EB53 28Ø2 18F4 CD25 54CD 7754 F121 1843 ØØØØ ØØC3 Ø544 CDB5 5321
C754 1195 54Ø6 ØØCD 2444 C921 C153 1195 54Ø1 ØCØØ EDBØ C953 594E 4F4E 594D 2F54 5854
ØD21 Ø756 1195 54CD 13ØØ CØ77 FEØD C8FE 2Ø38 F1E5 1157 56ED 52E1 28E8 2318 E521 Ø756
7ED6 3Ø47 237E FE2Ø 28FA 1117 4313 1AFE 2Ø28 Ø197 ØØ54 FA1A AEE6 DFCØ 2313 1ØF7 1AFE
ØDC8 FE2Ø C87E FE2Ø 28ØD FEØD 28Ø9 1AAE E6DF CØ23 1318 E7B7 C9DD E5D5 E521 C755 D11A
FE2Ø 28Ø3 1318 F813 1AFE 2Ø28 FA77 FEØD 2833 FE26 28Ø5 2313 1A18 F2DD E1DD E5DD 7EØØ
FE2Ø 28Ø6 381A DD23 18F3 DD23 DD7E ØØFE 2Ø38 ØD28 F577 23DD 23DD 7EØØ FE2Ø 3ØF5 2B18
D1D1 DDE1 C921 C755 1118 43Ø6 3F7E 12FE 2ØD8 2313 1ØF7 3EØD 12C9 2164 5322 ØD4Ø C9ØØ
ØØØ1 Ø557 5648 4A4D Ø2Ø2 ØØ52
```

```
ØØ11Ø ; SYNONYM -- Command line synonym processor for LDOS
ØØ12Ø ;      Copyright 1983 by Henry Melton
ØØ14Ø ;  The synonym processor resides in high memory and is linked into the error
ØØ15Ø ;  reporting chain. When the error code 95 appears, indicating that a Program not
ØØ16Ø ;  found error is to be flagged, SYNONYM reads the command buffer and uses the
ØØ17Ø ;  first word of the contents as a substitution key for an acceptable alternate
ØØ18Ø ;  command string.  A file with the name SYNONYM/TXT is needed that contains the
ØØ19Ø ;  substitution data.
ØØ46Ø ; Synonym records are searched sequentially til either a match or EOF occurs. If
ØØ47Ø ;  the new command line fails, then the processor tries again with this new
ØØ48Ø ;  input. When & substitution occurs, there is the possibility of the resulting
ØØ49Ø ;  command line exceeding the 64 character command buffer. If so, the line will
ØØ5ØØ ;  be truncated to 64 characters.
ØØ53Ø ECHO      EQU       1              ;Display the generated command = 1
ØØ55Ø @DSPLY    EQU       4467H
ØØ56Ø @EXIT     EQU       4Ø2DH
ØØ57Ø @GET      EQU       ØØ13H
ØØ58Ø @OPEN     EQU       4424H
ØØ59Ø ; Model 1 Equates
ØØ6ØØ HIGH1$    EQU       4Ø49H
ØØ61Ø INBUF1$   EQU       4318H
ØØ62Ø @ICNFG1   EQU       43Ø3H
ØØ63Ø @CMNDI1   EQU       44Ø5H
ØØ64Ø ; Model 3 Equates
ØØ65Ø HIGH3$    EQU       4411H
ØØ66Ø INBUF3$   EQU       4225H
ØØ67Ø @ICNFG3   EQU       421DH
ØØ68Ø @CMNDI3   EQU       4299H
ØØ69Ø ;
ØØ7ØØ           ORG       52ØØH
ØØ71Ø START::   LD        HL,BANNER       ;Display the program name and copyright
ØØ72Ø           CALL      @DSPLY
ØØ73Ø           LD        A,(@EXIT)
ØØ74Ø           CP        ØC3H            ;is it a 'jump'?
ØØ75Ø           JR        NZ,OKGO         ;if so, we aren't at LDOS Ready
ØØ76Ø           LD        HL,ABORTMS      ;and can't install/modify the system
ØØ77Ø           CALL      @DSPLY
ØØ78Ø           JP        @EXIT
ØØ79Ø OKGO:     LD        A,(Ø125H)       ;pick up type of machine flag
ØØ8ØØ           CP        'I'             ;will be "I" if Model 3
ØØ81Ø           JR        NZ,MOD1         ;continue if Mod 1
ØØ82Ø           LD        HL,HIGH3$       ;Pick up Mod 3 locations and
ØØ83Ø           LD        (MODCH1),HL     ;adjust all references
ØØ84Ø           LD        (MODCH2),HL
ØØ85Ø           LD        (MODCH3),HL
ØØ86Ø           LD        HL,@ICNFG3
```

```
ØØ87Ø         LD      (MODCH4),HL
ØØ88Ø         LD      (MODCH6),HL
ØØ89Ø         INC     HL
ØØ9ØØ         LD      (MODCH5),HL
ØØ91Ø         LD      (MODCH7),HL
ØØ92Ø         LD      HL,INBUF3$
ØØ93Ø         LD      (MODCH8),HL
ØØ94Ø         LD      (MODCH11),HL
ØØ95Ø         DEC     HL
ØØ96Ø         LD      (MODCH1Ø),HL
ØØ97Ø         LD      HL,@CMNDI3
ØØ98Ø         LD      (MODCH9),HL
Ø1ØØØ ; all Mod 1/3 conversion is done, let's get on with the good stuff...
Ø1Ø2Ø MOD1:   LD      IX,RELOTB       ;Using the relocation table and the
Ø1Ø3Ø         LD      HL,(HIGH1$)     ;High Memory location, patch the
Ø1Ø4Ø MODCH1  EQU     $-2
Ø1Ø5Ø         LD      (STORE),HL      ;load module for operation in high
Ø1Ø6Ø         LD      DE,LAST-1       ;memory.
Ø1Ø7Ø         OR      A
Ø1Ø8Ø         SBC     HL,DE
Ø1Ø9Ø         LD      B,H
Ø11ØØ         LD      C,L
Ø111Ø         LD      A,ENTRYS
Ø112Ø PTLOOP: LD      L,(IX+ØØH)
Ø113Ø         LD      H,(IX+Ø1H)
Ø114Ø         INC     HL
Ø115Ø         LD      E,(HL)
Ø116Ø         INC     HL
Ø117Ø         LD      D,(HL)
Ø118Ø         EX      DE,HL
Ø119Ø         ADD     HL,BC
Ø12ØØ         EX      DE,HL
Ø121Ø         LD      (HL),D
Ø122Ø         DEC     HL
Ø123Ø         LD      (HL),E
Ø124Ø         INC     IX
Ø125Ø         INC     IX
Ø126Ø         DEC     A
Ø127Ø         JR      NZ,PTLOOP
Ø129Ø         LD      DE,(HIGH1$)     ;Move the module to its high memory
Ø13ØØ MODCH2  EQU     $-2
Ø131Ø         LD      HL,LAST-1       ;location
Ø132Ø         LD      BC,LAST-FIRST
Ø133Ø         LDDR
Ø135Ø         LD      (HIGH1$),DE     ;Put the new HIGH$ value in storage
Ø136Ø MODCH3  EQU     $-2
Ø137Ø         LD      HL,(4ØØDH)
Ø138Ø RELØ:   LD      (VECTOR),HL     ; Plug the primary RST 4Ø
Ø139Ø REL1:   LD      HL,BEGIN        ; vector with the SYN processor
Ø14ØØ         LD      (4ØØDH),HL      ; address.
Ø141Ø         LD      A,(@ICNFG1)     ; Do the same for the CONFIG
Ø142Ø MODCH4  EQU     $-2
Ø143Ø RELX:   LD      (CMD),A         ; initialization chain so the
Ø144Ø         LD      HL,(@ICNFG1+1)  ; SYN processor can be SYSGENed
Ø145Ø MODCH5  EQU     $-2
Ø146Ø RELY:   LD      (ADDRES),HL
Ø147Ø         LD      A,ØC3H
Ø148Ø         LD      (@ICNFG1),A
Ø149Ø MODCH6  EQU     $-2
Ø15ØØ RELZ:   LD      HL,INIT
Ø151Ø         LD      (@ICNFG1+1),HL
Ø152Ø MODCH7  EQU     $-2
Ø153Ø         JP      @EXIT           ;end of the loading operation
```

```
BANNER:   DEFM     'SYNONYM -- LDOS command line synonym processor '
          DEFB     ØAH
          DEFM     'Copyrighted 1983, Henry Melton'
          DEFB     ØDH
ABORTMS:  DEFM     'Only valid at LDOS Ready - installation aborted!'
          DEFB     ØAH
          DEFB     ØDH
RELOTB:   DEFW     REL1      ;Table of addresses to relocate to HIGH$
          DEFW     RELØ
          DEFW     INIT
          DEFW     RELX
          DEFW     RELXX
          DEFW     RELY
          DEFW     RELZ
          DEFW     INTERCPT
          DEFW     LOOP1
          DEFW     REL2
          DEFW     PROCESS
          DEFW     REL3
          DEFW     OPENSYN
          DEFW     REL4
          DEFW     REL5
          DEFW     LOADFCB
          DEFW     REL7
          DEFW     READLN
          DEFW     LOOP2
          DEFW     COMPARE
          DEFW     REL8
TBLEND:   DEFW     MOVELN
TBSIZE    EQU      TBLEND-RELOTB+2
ENTRYS    EQU      TBSIZE/2
FIRST:    JR       BEGIN
STORE:    DEFW     Ø                   ;to receive the old HIGH$ value
NAME:     DEFB     BEGIN-TEXT
TEXT:     DEFM     'SYNON'
BEGIN:    PUSH     AF
          CP       86H                 ;A true error code has
          JR       Z,CHECK2            ; an 86H here.
QUIT:     POP      AF                  ; so if not, go about
          DEFB     ØC3H                ; your business.
VECTOR:   DEFW     Ø
CHECK2:   INC      SP
          INC      SP
          INC      SP
          INC      SP
          POP      AF
          CP       95                  ; A PROGRAM NOT FOUND
          JR       Z,INT               ; error will have a 95
          PUSH     AF                  ; here.
          LD       A,86H
          DEC      SP
          DEC      SP
          DEC      SP
          DEC      SP
          JR       QUIT
INT:      DEC      SP                  ; make sure the stack
          DEC      SP                  ; is ordered properly
          DEC      SP                  ; to minimize side effects
          DEC      SP
          DEC      SP
          DEC      SP
INTERCPT: CALL     OPENSYN             ; open SYNONYM/TXT
          JR       NZ,QUIT
```

```
Ø22ØØ LOOP1:    CALL    READLN          ; read in a line
Ø221Ø           JR      NZ,QUIT
Ø222Ø REL2:     CALL    COMPARE         ; test for a match
Ø223Ø           JR      Z,PROCESS       ; if so, process it
Ø224Ø           JR      LOOP1           ; or else loop
Ø225Ø PROCESS:  CALL    BUILDLN         ;build the replacement
Ø226Ø REL3:     CALL    MOVELN          ; command line and move
Ø227Ø           POP     AF              ; it back into the
Ø228Ø           LD      HL,INBUF1$      ; command buffer and
Ø229Ø MODCH8    EQU     $-2
Ø23ØØ           IF      ECHO                     ; execute it.
Ø231Ø           CALL    @DSPLY
Ø232Ø           ELSE
Ø233Ø           NOP
Ø234Ø           NOP
Ø235Ø           NOP
Ø236Ø           ENDIF
Ø237Ø           JP      @CMNDI1
Ø238Ø MODCH9    EQU     $-2
Ø24ØØ OPENSYN:  CALL    LOADFCB
Ø241Ø REL4:     LD      HL,DSKBUF
Ø242Ø REL5:     LD      DE,FCB
Ø243Ø           LD      B,Ø
Ø244Ø           CALL    @OPEN
Ø245Ø           RET
Ø247Ø LOADFCB:  LD      HL,FILENAME
Ø248Ø REL7:     LD      DE,FCB
Ø249Ø           LD      BC,CR-FILENAME+1
Ø25ØØ           LDIR
Ø251Ø           RET
Ø253Ø FILENAME: DEFM    'SYNONYM/TXT'
Ø254Ø CR:       DEFB    13
Ø256Ø READLN:   LD      HL,LINEBF       ;Read a line of text into
Ø257Ø LOOP2:    LD      DE,FCB          ; a local buffer until
Ø258Ø           CALL    @GET            ; a CR, skipping all nulls
Ø259Ø           RET     NZ              ; or other control characters.
Ø26ØØ           LD      (HL),A
Ø261Ø           CP      13
Ø262Ø           RET     Z
Ø263Ø           CP      2ØH
Ø264Ø           JR      C,LOOP2
Ø265Ø           PUSH    HL
Ø266Ø RELXX:    LD      DE,LAST         ; Make sure the text does
Ø267Ø           SBC     HL,DE           ; not exceed the line
Ø268Ø           POP     HL              ; buffer.
Ø269Ø           JR      Z,LOOP2
Ø27ØØ           INC     HL
Ø271Ø           JR      LOOP2
Ø273Ø COMPARE:  LD      HL,LINEBF       ;Get the match count
Ø274Ø           LD      A,(HL)          ; from the first
Ø275Ø           SUB     3ØH             ; character in the
Ø276Ø           LD      B,A             ; synonym record line.
Ø277Ø LOOP3:    INC     HL
Ø278Ø           LD      A,(HL)          ; skip spaces in the
Ø279Ø           CP      2ØH             ; syn record
Ø28ØØ           JR      Z,LOOP3
Ø281Ø           LD      DE,INBUF1$-1
Ø282Ø MODCH1Ø   EQU     $-2
Ø283Ø LOOP4:    INC     DE
Ø284Ø           LD      A,(DE)          ; skip leading spaces
Ø285Ø           CP      2ØH             ; in the original
Ø286Ø           JR      Z,LOOP4         ; command line
Ø287Ø LOOP5:    LD      A,(DE)
```

```
Ø288Ø          XOR     (HL)              ;compare loop.
Ø289Ø          AND     ØDFH              ; ignoring upper/lower
Ø29ØØ          RET     NZ                ; case distinctions,
Ø291Ø          INC     HL                ; compare for the full
Ø292Ø          INC     DE                ; match count, rejecting
Ø293Ø          DJNZ    LOOP5             ; for any mismatch.
Ø294Ø LOOP5A:  LD      A,(DE)
Ø295Ø          CP      13                ; After the count,
Ø296Ø          RET     Z                 ; if the original ends
Ø297Ø          CP      2ØH               ; first, match is okay.
Ø298Ø          RET     Z
Ø299Ø          LD      A,(HL)            ; But if the SYN key
Ø3ØØØ          CP      2ØH               ; ends first, then the
Ø3Ø1Ø          JR      Z,NOMATCH         ; match is rejected.
Ø3Ø2Ø          CP      13
Ø3Ø3Ø          JR      Z,NOMATCH
Ø3Ø4Ø          LD      A,(DE)
Ø3Ø5Ø          XOR     (HL)
Ø3Ø6Ø          AND     ØDFH
Ø3Ø7Ø          RET     NZ
Ø3Ø8Ø          INC     HL
Ø3Ø9Ø          INC     DE
Ø31ØØ          JR      LOOP5A
Ø311Ø NOMATCH: OR      A
Ø312Ø          RET
Ø314Ø BUILDLN: PUSH    IX                ;Save IX on general principles.
Ø315Ø          PUSH    DE                ;Save the pointer into INBUF$
Ø316Ø          PUSH    HL                ;Save the pointer into the SYN record
Ø317Ø REL8:    LD      HL,NEWLN          ;Start at the beginning of the
Ø318Ø          POP     DE                ; new line buffer,
Ø319Ø LOOP6:   LD      A,(DE)            ; skip until a space
Ø32ØØ          CP      2ØH
Ø321Ø          JR      Z,LOOP7
Ø322Ø          INC     DE
Ø323Ø          JR      LOOP6
Ø324Ø LOOP7:   INC     DE                ; skip spaces until replacement
Ø325Ø          LD      A,(DE)            ; text is encountered.
Ø326Ø          CP      2ØH
Ø327Ø          JR      Z,LOOP7
Ø328Ø LOOP8:   LD      (HL),A            ; copy replacement text
Ø329Ø          CP      13                ; until EOL
Ø33ØØ          JR      Z,BLDEX
Ø331Ø          CP      '&'               ; if '&' is in replacement
Ø332Ø          JR      Z,SUBSTIT         ; text then substitute.
Ø333Ø RETUR:   INC     HL
Ø334Ø          INC     DE
Ø335Ø          LD      A,(DE)
Ø336Ø          JR      LOOP8
Ø338Ø SUBSTIT: POP     IX                ;Pick up the pointer to the
Ø339Ø          PUSH    IX                ; remainder of the original
Ø34ØØ LOOP9:   LD      A,(IX)            ; command line, and copy it
Ø341Ø          CP      2ØH               ; to the new command line.
Ø342Ø          JR      Z,LOOP1Ø
Ø343Ø          JR      C,NOSUB           ; Skip leading spaces.
Ø344Ø          INC     IX                ; If no no-space characters,
Ø345Ø          JR      LOOP9             ; then & vanishes.
Ø346Ø LOOP1Ø:  INC     IX
Ø347Ø          LD      A,(IX)
Ø348Ø          CP      2ØH
Ø349Ø          JR      C,NOSUB
Ø35ØØ          JR      Z,LOOP1Ø
Ø351Ø LOOP11:  LD      (HL),A
Ø352Ø          INC     HL
```

```
         INC     IX
         LD      A,(IX)
         CP      2ØH
         JR      NC,LOOP11
NOSUB:   DEC     HL
         JR      RETUR
BLDEX:   POP     DE
         POP     IX
         RET
MOVELN:  LD      HL,NEWLN
         LD      DE,INBUF1$          ; copy up to 63 text
MODCH11  EQU     $-2
         LD      B,63                ; characters and one
LOOP12:  LD      A,(HL)              ; CR to the original
         LD      (DE),A              ; command buffer.
         CP      2ØH
         RET     C
         INC     HL
         INC     DE
         DJNZ    LOOP12
         LD      A,13
         LD      (DE),A
         RET
INIT:    LD      HL,BEGIN            ; This is the CONFIG
         LD      (4ØØDH),HL          ; initialization routine.
CMD:     DEFB    ØC9H
ADDRES:  DEFW    Ø
FCB:     DEFS    5Ø
DSKBUF:  DEFS    256
NEWLN:   DEFS    64                  ; Result line buffer
LINEBF:  DEFS    8Ø                  ; Syn line buffer
LAST:    DEFM    'HJM'
         END     START
```

Modifying the Model 3 Real-Time-Clock Interrupts

by Andrew Gransden, c/o 68 St Annes Grove, FAREHAM, Hampshire, England, UK TS15 9TB

This article is primarily aimed at those TRS-8Ø Model III owners, living outside North America, with machines adapted to work with 5Ø Hertz mains power. This is not to say that other readers will not be interested in the experiences described below. (With some modification, this approach could be used to correct the clock on a Model 4 running in the Model 3 mode under 5.1.4 at the 4MHz speed - ed.)

This all started by replacing my unreliable Mod I-type machine (Video Genie {UK}/PMC-8Ø {USA}) with a Mod 3 from Tandy. I stayed with the TRS-8Ø line only because I am hooked on LDOS, and on the fact that I could use all my LDOS compatible software (with only minor patching) and disks on my new machine. I was extremely pleased with my Mod 3 operating under LDOS apart from the annoying fact that the Real Time Clock (RTC) lost 1Ø seconds every minute. As I had always used the RTC in my programs, and with system functions like JOBLOG, I set forth to find a means of correcting this problem.

The internal clock must run at the proper frequency to ensure that the display is not affected by jitter or flickering. In the Mod 3, the internal clock frequency is divided to yield a RTC interrupt frequency half that of the mains. In a 6ØHz machine the RTC interrupt occurs at 3ØHz, or every 33.33 milliseconds, while in the 5ØHz machine it occurs at about 25Hz (25.381Hz to be precise) or approximately every 4Ø milliseconds. Unfortunately, TANDY did not compensate for this difference by replacing the System ROMs. TRSDOS 1.3 can be patched, but under LDOS the clock management is left to the ROM. This means that the RTC goes uncorrected when using LDOS.

Having established the cause of the problem, I now had to find a way of applying my own correction. With reference to the Technical Information section of the LDOS Manual, the

Model III Technical Reference Manual (Cat No. 26-21Ø9) and the excellent reference work 'Model III ROM Commented' (a fully commented disassembly of the System ROMs) and armed with EDAS 4.1, DSMBLRII and the LDOS DEBUGger, I set about my task.

One solution would have been to design, build, and install an additional crystal-driven clock operating at 3ØHz. This would have been easy enough, but with the result that the warranty on my Mod 3 would have been voided. Another solution would have been to order, at some expense, an external battery-powered clock. The third, and the solution I chose was to correct the clock error using software. The following is a description of the Mod 3/LDOS RTC interrupt control chain, and how I solved the problem.

When the RTC pulse occurs, it interrupts the CPU and sets Bit 2 of the Interrupt Status Port (X'EØ') to zero. An image of this Status Port is kept by LDOS in INTIM$ (X'4473'). The jump vectors relevant to each Interrupt Status Bit are held in INTVC$ (X'4475' - X'4484'). In LDOS, the vector relating to Bit 2 of INTIM$ points to X'44A5' in SYSØ, which in turn jumps to X'3529' in ROM where the RTC routines can be found. This routine decrements the system clock 'heartbeat' (X'4216') from 3Ø (X'1E') down to zero. When 'heartbeat' reaches zero, the time is incremented and, if selected, displayed in the corner of the screen. What I needed to do was to re-write this routine to operate properly at the actual 25Hz interrupt rate instead of the expected 3ØHz. To totally re-write the ROM routine would have used a lot of high memory, and would have duplicated a lot of code. The solution was to write a relocatable routine which would apply a correction to the clock count, and then hand control back to the ROM RTC routine. Results from several experiments proved that I needed to apply a double correction using 2 counters. The final affect is to stretch every 2Øth second by approximately a third. The accuracy of the corrected RTC is within 3 seconds a day.

The program first loads into low memory; locates the current RTC interrupt vector and saves it; modifies the internal references; installs the routine in high memory below the current HIGH$; and then reduces HIGH$ to protect the routine. The correction routine uses only 87 bytes of high memory and obeys the 'front-end' protocol as defined by LSI. The program carries out a series of checks to ensure that you are using a Mod 3 with LDOS (Version 5.1.x) and you have not previously applied the correction. The program will abort with a suitable error message if any of these tests fail.

Once installed the routine can be SYSGENed to load every time your Model III is booted. This is because SYSGEN saves all system vectors including those contained in INTVC$ as well as the high memory area above HIGH$. One word of warning: as should be the case with all programs used with LDOS, HIGH$ should be respected, otherwise your system will crash FASTER than ever before (within 25 ms of clobbering the correction routine).

Below is the BINHEX dump for those of you without an Editor/Assembler.

```
Ø5Ø6 5449 4D45 4249 Ø1Ø2 ØØ52 2196 52CD 6744 3A25 Ø1FE 49C2 1853 2A13 4Ø3E A5BD C21D
533E 44BC C21D 532A 7944 3E4F BCDA 2253 2A11 44DD 21C9 53DD 75Ø1 DD74 Ø2DD 21CD 53DD
75Ø1 DD74 Ø2DD 21D6 53DD 75Ø1 DD74 Ø23E 1577 2BDD 21DE 53DD 75Ø1 DD74 Ø2DD 21E2 53DD
75Ø1 DD74 Ø2DD 21EB 53DD 75Ø1 DD74 Ø23E 2477 2A79 4422 ØA54 2A11 4422 B953 Ø157 ØØAF
ED42 2211 4423 F322 7944 EB21 B753 EDBØ FB21 F752 CD8A 42C3 2D4Ø 5449 4D45 353Ø 2Ø2D
2Ø4C 444F 532Ø 5265 616C 2Ø54 696D 652Ø 436C 6F63 6B2Ø 353Ø 487A 2Ø43 6F72 7265 6374
696F 6E2Ø 5574 696C 6974 792Ø 2D2Ø 5665 722E 342E 31ØA 436F 7Ø79 7269 6768 742Ø 2843
292Ø 3139 3833 2Ø2Ø 412Ø 572Ø 4772 616E 7364 656E ØD54 494D 4535 3Ø2Ø 696E Ø1BB ØØ53
7374 616C 6C65 642Ø 616E 642Ø 6F7Ø 6572 6174 696F 6E61 6CØD 2131 5318 Ø821 5Ø53 18Ø3
218A 53CD 8A42 21A7 53CD 8A42 C33Ø 4Ø46 6F72 2Ø54 5253 2D38 3Ø2Ø 4D6F 6465 6C2Ø 4949
492Ø 7573 652Ø 4F4E 4C59 21ØD 436F 7272 6563 7469 6F6E 2Ø77 7269 7474 656E 2Ø74 6F2Ø
776F 726B 2Ø75 6E64 6572 2Ø4C 444F 532Ø 5665 7273 696F 6E2Ø 352E 312E 782Ø 4F4E 4C59
21ØD 436F 7272 6563 7469 6F6E 2Ø61 6C72 6561 6479 2Ø69 6E73 7461 6C6C 6564 ØD54 494D
4535 3Ø2Ø 4162 6F72 7465 6421 ØD18 Ø9Ø1 53BB 53Ø6 5449 4D45 353Ø 3A16 42FE 162Ø 353A
ØC54 3D32 ØC54 FEØØ 2ØØA 3E15 32ØC 543E 1E32 1642 3AØD 543D 32ØD 54FE ØØ2Ø 153E 2432
ØD54 3EØ1 5F3A 1642 93FE Ø63Ø Ø23E Ø632 1642 3A16 42EE Ø62Ø Ø43C 3216 42C3 ØØØØ Ø2Ø2
ØØ52
```

```
ØØ1ØØ ;*       TIME5Ø/ASM - Version 4.1 - 28 Oct 83
ØØ11Ø ;*       TITLE:  TIME5Ø 5ØHz Real Time Clock Correction Routine
```

```
;*      AUTHOR: Andrew W. Gransden
;*              c/o 68 St Annes Grove
;*              FAREHAM, Hampshire
;*              PO14 1JW   England  UK
;*      Copyright (c) 1983 A W Gransden
;*      TIME50 routine corrects for errors in the
;*      the TRS-80 Model III Real Time Clock
;*      interrupt handling routine when operating
;*      on a 50Hz Version Machines under the
;*      LDOS Disk Operating System Version 5.1.x.
;*      Variable and Label Declarations
LF      EQU     0AH     ;linefeed
CR      EQU     0DH     ;carriage return
SEC1    EQU     21      ;coarse correction count
COR1    EQU     8       ;coarse correction value
SEC2    EQU     36      ;fine correction count
COR2    EQU     1       ;fine correction value
ROMCHK  EQU     0125H   ;ROM check for Model III
INTVEC  EQU     4012H   ;interrupt vector
@EXIT   EQU     402DH   ;LDOS return entry
@ABORT  EQU     4030H   ;abnormal program exit to LDOS
HBEAT$  EQU     4216H   ;clock heartbeat counter
HIGH$   EQU     4411H   ;highest useable memory
@DSPLY  EQU     4467H   ;display message
@LOGOT  EQU     428AH   ;display & log message
RTCVEC  EQU     4479H   ;jump vector to RTC routine
;*      start of TIME50 installing routine                          *
        ORG     5200H
ENTRY   LD      HL,MSG1                 ;point to message 1
        CALL    @DSPLY                  ;and display it
        LD      A,(ROMCHK)              ;test for Model III
        CP      49H
        JP      NZ,ERROR                ;go if not
        LD      HL,(INTVEC+1)           ;load interrupt vector
        LD      A,0A5H                  ;load A with known jump
        CP      L                       ;and compare with HL
        JP      NZ,ERROR1               ;exiting if incorrect
        LD      A,44H                   ;to error handing abort
        CP      H
        JP      NZ,ERROR1
        LD      HL,(RTCVEC)             ;load RTC vector
        LD      A,4FH                   ;load max DOS area addr
        CP      H
        JP      C,ERROR2                ;go if greater
        LD      HL,(HIGH$)              ;get current high mem loc
        LD      IX,P1                   ;set pointer to allow
        LD      (IX+1),L                ;correct addressing of
        LD      (IX+2),H                ;storage in high memory
        LD      IX,P2
        LD      (IX+1),L
        LD      (IX+2),H
        LD      IX,P3
        LD      (IX+1),L
        LD      (IX+2),H
        LD      A,SEC1                  ;set count to ? seconds
        LD      (HL),A
        DEC     HL
        LD      IX,P4                   ;set pointer as above
        LD      (IX+1),L
        LD      (IX+2),H
        LD      IX,P5
        LD      (IX+1),L
        LD      (IX+2),H
```

```
ØØ8ØØ          LD     IX,P6
ØØ81Ø          LD     (IX+1),L
ØØ82Ø          LD     (IX+2),H
ØØ83Ø          LD     A,SEC2
ØØ84Ø          LD     (HL),A
ØØ85Ø          LD     HL,(RTCVEC)            ;get present int vector
ØØ86Ø          LD     (EXIT+1),HL            ;and save
ØØ87Ø          LD     HL,(HIGH$)             ;reduce HIGH$ by
ØØ88Ø          LD     (NXTMEM),HL            ;store old high memory
ØØ89Ø          LD     BC,LAST-START          ;length of routine
ØØ9ØØ          XOR    A                      ;clear carry flag
ØØ91Ø          SBC    HL,BC                  ;calculate new HIGH$
ØØ92Ø          LD     (HIGH$),HL             ;protect routine
ØØ93Ø          INC    HL                     ;point to new start
ØØ94Ø          DI                            ;disable interrupts
ØØ95Ø          LD     (RTCVEC),HL            ;break into Int chain
ØØ96Ø          EX     DE,HL                  ;transfer new START to DE
ØØ97Ø          LD     HL,START               ;load address of routine
ØØ98Ø          LDIR                          ;move routine to high ram
ØØ99Ø          EI                            ;enable interrupts
Ø1ØØØ          LD     HL,MSG2                ;point to message
Ø1Ø1Ø          CALL   @LOGOT                 ;display & log
Ø1Ø2Ø          JP     @EXIT                  ;return to LDOS
Ø1Ø4Ø ;*       Messages                                                 *
Ø1Ø6Ø MSG1     DB     'TIME5Ø - LDOS Real Time Clock 5ØHz '
Ø1Ø7Ø          DB     'Correction Utility - Ver.4.1',LF
Ø1Ø8Ø          DB     'Copyright (C) 1983  A W Gransden',CR
Ø1Ø9Ø MSG2     DB     'TIME5Ø installed and operational',CR
Ø111Ø ;*       error handling                                           *
Ø113Ø ERROR    LD     HL,ERMSG               ;point to message
Ø114Ø          JR     EREXIT                 ;jump to error exit
Ø115Ø ERROR1   LD     HL,ERMSG1              ;point to error message
Ø116Ø          JR     EREXIT                 ;jump to error exit
Ø117Ø ERROR2   LD     HL,ERMSG2              ;point to error message
Ø118Ø EREXIT   CALL   @LOGOT                 ;display message
Ø119Ø          LD     HL,ERMSG3              ;load abort message
Ø12ØØ          CALL   @LOGOT                 ;display & log
Ø121Ø          JP     @ABORT                 ;jump to abort routine
Ø123Ø ;*       error messages                                           *
Ø125Ø ERMSG    DB     'For TRS-8Ø Model III use ONLY!'
Ø126Ø          DB     CR
Ø127Ø ERMSG1   DB     'Correction written to work under '
Ø128Ø          DB     'LDOS Version 5.1.x ONLY!'
Ø129Ø          DB     CR
Ø13ØØ ERMSG2   DB     'Correction already installed'
Ø131Ø          DB     CR
Ø132Ø ERMSG3   DB     'TIME5Ø Aborted!',CR
Ø134Ø ;*       actual TIME5Ø routine to be placed in high ram           *
Ø136Ø START    EQU    $
Ø137Ø          JR     J1                     ;skip protocol block
Ø138Ø NXTMEM   DS     2                      ;high mem addr of next bk
Ø139Ø          DB     Ø6H                    ;6 bytes of protocol blk
Ø14ØØ          DB     'TIME5Ø'               ;routine title
Ø141Ø J1       EQU    $
Ø142Ø          LD     A,(HBEAT$)             ;get heartbeat count
Ø143Ø          CP     1EH-COR1               ;just reset?
Ø144Ø          JR     NZ,TEST2               ;go if not
Ø145Ø P1       LD     A,(COUNT1$)            ;get seconds count
Ø146Ø          DEC    A                      ;decrease by one
Ø147Ø P2       LD     (COUNT1$),A            ;save
Ø148Ø          CP     ØØH                    ;... seconds gone
Ø149Ø          JR     NZ,P4                  ;go if not
Ø15ØØ          LD     A,SEC1                 ;reset seconds count
```

```
Ø151Ø P3       LD    (COUNT1$),A      ;save reset count
Ø152Ø          LD    A,1EH            ;increase heartbeat
Ø153Ø          LD    (HBEAT$),A       ;save reduced heartbeat
Ø154Ø P4       LD    A,(COUNT2$)      ;carry out fine
Ø155Ø          DEC   A                ;correction
Ø156Ø P5       LD    (COUNT2$),A      ;and save
Ø157Ø          CP    ØØH
Ø158Ø          JR    NZ,TEST2         ;when COUNT2$ reaches Ø
Ø159Ø          LD    A,SEC2           ;reset correction count
Ø16ØØ P6       LD    (COUNT2$),A
Ø161Ø          LD    A,COR2
Ø162Ø          LD    E,A
Ø163Ø          LD    A,(HBEAT$)       ;get heart beat count
Ø164Ø          SUB   E                ;remove correction
Ø165Ø          CP    Ø6H              ;finished?
Ø166Ø          JR    NC,J2            ;go if less
Ø167Ø          LD    A,Ø6H
Ø168Ø J2       LD    (HBEAT$),A
Ø169Ø TEST2    LD    A,(HBEAT$)       ;get heartbeat count
Ø17ØØ          XOR   Ø6               ;at new bottom?
Ø171Ø          JR    NZ,EXIT          ;go if not
Ø172Ø          INC   A                ;heartbeat=1
Ø173Ø          LD    (HBEAT$),A       ;save reduced heartbeat
Ø174Ø EXIT     EQU   $
Ø175Ø          JP    ØØØØH            ;jump to RTC routine
Ø176Ø COUNT1$  DS    1                ;reserve 2 bytes
Ø177Ø COUNT2$  DS    1                ;for correction counts
Ø178Ø LAST     EQU   $
Ø179Ø          END   ENTRY
```

Profile ONE Plus ??

by E. R. Sturiale, SASSCO Microcomputer Services
133 Falmouth St., Rochester, NY 14615 (716) 865-1622

Has Radio Shack forgotten about the Mod 1 user? If their marketing of Profile III+ is any indication, they have. Fortunately, when they made the wise decision to use LDOS as their hard disk operating system, the possibility arose to develop patches for Mod 1. Since we run a small software consulting firm which uses both Mod 1's and 3's, it became necessary to have compatibility between machines for Profile data bases.

As you can probably tell from the number of patches, the conversion of machine language programs is not simple. Also, there are just enough hardware differences between machines to generate Excedrin headaches 256 through 1Ø23. The calls to the operating system were relatively easy to change after the required 896 pages of disassembling and cross referencing to decode the Profile modules.

Applying the patches

1) The minimum requirements to install these patches and run Profile I+ are:
   Double Density (any LDOS-supported form)
   Two Disk Drives
   PROFILE III+ HD (RS number 26-1593)

2) BACKUP your RS Profile III+ distribution diskette onto a working patch DATA disk. This can be accomplished after formatting by using the LDOS Backup utility with (X,VIS) or any other copy by file option. By all means leave the write protect tab on the distribution diskette.

3) Create a "CLEAN" LDOS system diskette with at least 5Ø K of free space.

4) Type in all the FIX and JCL files using the BUILD command (or some ASCII text editor). Store them on the "CLEAN" LDOS system diskette and double check your typing.

5) Place the Backup (working) PROFILE diskette in Drive 1 and the system disk with the fix files in Drive Ø.

6) NOTE : If your distribution diskette is labeled Version Ø1.ØØ.Ø1, or if you have already applied the patches supplied by Radio Shack, then skip to Step 7.

   Now type: DO RSPATCH

   If the messages indicate all is O.K., then continue else check your RSPATCH/JCL file and then return to Step 2

7) This JCL will apply all the patches necessary for Profile 1 + to operate. The procedure may take as long as five minutes. Please watch the screen for errors in patching. If one does occur, then check the FIX files and go back to Step 2.

   To start the patch procedure, type: DO PROFIX

8) If you are going to use the system for Hard Disk operation, then copy all your Profile /CMD programs from the working disk on Drive 1 to your HD and begin hacking. If you want to use the system on floppy, then continue. This set of patches will add prompts for diskette swaps.

   Now type: DO PROMPT

9) At this point, create two more LDOS system diskettes and label the first "CREATION DISKETTE" and the other "RUNTIME DISKETTE"

   Copy the following patched files to the CREATION diskette:
   EFC1/CMD EFC2/CMD EFC3/CMD EFC4/CMD EFC5/CMD EFC6/CMD EFCE/CMD EFCM/CMD CM/CMD

   Copy the following patched files to the RUNTIME diskette:
   EFC7/CMD EFC8/CMD EFC9/CMD EFCA/CMD EFCB/CMD EFCC/CMD EFCD/CMD EFCF/CMD RM/CMD

   If all goes well, you should be ready to dive into the manual and get started.

There are several differences in PROFILE I+ that I should mention.

1) The Profile I+ programs will try to write the working modules such as screen formats on Drive Ø. When working with the floppy version, it is usually more convenient to have these files somewhere else. The best way to do this is to use the LDOS SYSTEM (DRIVE=Ø,WP=ON). This forces the files to be written on the next available drive and not your CREATION diskette.

2) The cursor character on the DEFINE SCREENS option is different than standard. The only problem that may occur is if you try to use the special character from the <shift> @ display which is the same as the cursor. If the cursor passes over this character the special character will be erased. Since there are lots of other special characters that look like the cursor, you should not have any problems selecting another one.

3) Part of the patching procedure disables the Model III scroll protect option since the Model I does not have that feature. The easy way to do that was to change the memory loads required to a place where they will not do any damage. I chose location 3ØØ1H which is in non-existent memory in the Model I. If you are using a memory side-car that resides in that area then either un-plug it, or change all the patches in the FIX files that are "Ø1 3Ø" to a location that is not being used. By the way, I have not noticed any difference in operation or screen presentation by eliminating the scroll protect feature. Other than some

characters being different, due to the different character sets, all screen presentations seem to be O.K.

4) Using KI/DVR with the (TYPE) option is recommended. Use of Profile I+ with the other drivers or filters (except the HD drivers, PDUBL and RDUBL) has not been evaluated. If you wish to try some others, experimentation may be required. When using KI/DVR with Profile I+, any prompts that call for the <clear> key should be replaced with <shift><clear>.

5) None of the other modules that are offered by Small Computer Company have been patched or tested with Profile I+. We have plans to purchase them so if patches are necessary, you may see them in future LSI Journals.

6) After Profile I+ was working for awhile, we noticed that the programs gave excessive PRINTER NOT READY messages when everything was in fact O.K. Patches were added to lengthen the delay time the programs wait for the printer to respond to accommodate such functions as double striking or fast CPU's.

The patches are available in XAØ of the LDOS SIG on CompuServe, and are presented here starting on page 47.

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

by Tim Daneliuk, T&R Communications Assoc., 4927 N. Rockwell St., Chicago IL 6Ø625

Well, Winter is upon us. If that isn't bad enough, my friendly LSI Journal editor decided that he needed TWO columns absolutely ASAP. He said "I need two PARITY=RIDICULOUS columns by the deadline. Can you do it?" Well, I barely make it to work on time. So friends, I had to get moving. Translated, this means that what you are about to read is probably not up to the high journalistic standards you've come to expect from me. I must be held blameless, as I simply cannot rush two columns out in 12 weeks and be expected to maintain the excellence and humility you've all come to know and love...

CP/M and the TRS-8Ø

Let's face it. CP/M is not a great operating system. It's not really even a GOOD operating system. There are too many systems out there which use CP/M as the DOS (?) to ignore "Pa Kildall's" favorite product. Even the folks at Radio Shack have announced their intention to bring out CP/M+ (aka CP/M 3.x) for the Mods 4, 4P and 12. So, as a public service, let's discuss a few CP/M-8Ø related products.

In all fairness, I should point out that CP/M was FIRST! For its day, it was a fine product which served the 8-bit micro industry well. Much of the popularity of early microcomputers was partly due to the existence of CP/M. Since it was first, CP/M is among the best outside vendor and user-supported operating system on the market. There are products which ONLY run under CP/M, so it's a good idea to be familiar with it.

First, let's look at Montezuma Micro's (hereafter known as MM) CP/M for the Model 4. Tandy announced CP/M availability this spring, and to date has not yet released it. The MM CP/M is a full blown CP/M 2.2 for the Mod 4. It incorporates all the usual CP/M commands (all 4 of 'em) and utilities, with extras. The folks at MM have done a real nice job on this implementation. The BIOS (Basic Input/Output System) is written to emulate an ADM-3A. The keyboard driver allows you to input special characters (curly braces {}, brackets [], and backslash \) which are not normally available on the Model 4 keyboard. MM has gone as far as cleaning up those hideous CP/M error messages. Now after an error, you get the choice of retrying the operation which caused the error, letting CP/M handle the error, or rebooting the system. Instead of the usual "BDOS ERROR" message, the MM CP/M tells you things like "Record Not Found" and the like.

The highlight of this CP/M implementation is that it is chock full of useful utilities. The most notable is INTERCHG.COM, a utility which reads "alien" disk formats. The only "standard" CP/M format is eight inch, single-sided, single-density. As a result, there is no "compatibility" of the CP/M system whenever you're using other types of media. This isn't so bad with eight inch floppy disks because if you can write anything else, you can almost always create single-density single-sided disks also.

With five inch CP/M, each manufacturer specifies the media format as they choose. This has created a wonderful mess of incompatible five inch CP/M disks. INTERCHG reads over 2Ø popular five inch CP/M media formats, including Osborne, Xerox, Lobo, Zenith, and NEC disks. I tested INTERCHG on Lobo and NEC PC-8ØØ1 five inch disks. I was able to read both with no difficulty whatsoever.

MM has also included the popular public-domain MODEM.COM program. This gives you communications ability under CP/M. MEMLINK.COM is a "RAMdrive" program which can use the extra 64K memory bank as a logical drive, like TRSDOS 6.x. The only thing missing in this CP/M implementation is the ability to use the Shack's hard disks under CP/M. On the other hand, why would anyone stop using LDOS and start using CP/M on a hard disk?

In the several months I've used the MM CP/M, it has run flawlessly with one exception. When you do a lot of disk I/O among several different drives, you get Record Not Found errors. I've detailed this bug to the manufacturer. By the way, if you're worried about support, fear not! Montezuma Micro is run by the same people who run Aerocomp. Aerocomp is one of the most reputable and helpful mail-order houses in the business. The Model 4 CP/M package comes with 36 pages of documentation about the implementation, as well as Dave Cortesi's EXCELLENT book "INSIDE CP/M". The latter is a 5ØØ+ page discussion of CP/M. MM CP/M costs $199 and is available from:

Montezuma Micro, P.O. Box 32Ø27, Dallas, TX 75232 (214) 339-51Ø4

Thanks to a local CP/M "guru", I have been introduced to a fabulous new product called MPC. If you fiddle around in assembler, you've probably had the urge at some point to write you're own operating system. No, you're not crazy! (Well, maybe a little crazy...) Of course, such a project is a large undertaking, and would require more time than the average individual has available. The next best thing would be to look at the source code for someone else's DOS. Unfortunately, most DOS authors frown on distributing the source for their system, and quite understandably!

You may have noticed that MPC is CPM reversed, and that's exactly what this product does. It "reverses" (disassembles) the code which makes up the CP/M kernel and produces a FULLY COMMENTED source file. If you're interested in how the BDOS (Basic Disk Operating System) or CCP (Console Command Processor) work, you can read and study this file. The comments alone are worth their weight in gold and will give you great insight as to how CP/M actually works. MPC costs only $35 and is available from:

CC Software, 2564 Walnut Blvd. #1Ø6, Walnut Creek, CA 94598, (415) 939-8153

THINGS FOR THE MAX-8Ø

The folks at Powersoft are at it again! Apparently they really like the MAX-8Ø 'cause they keep bringing out new products for it. First, a "MAXed" version of SU+, and now SETMAKER/SETWRITER. If you own a MAX, you already know that the video system on the MAX is quite versatile. The character set is programmable - i.e. you can make each character look as you wish. Unfortunately, this is a messy proposition involving time, effort, and assembly language.

SETMAKER allows you to create custom character fonts and graphics characters on the MAX-8Ø. These can be saved as on disk or they can be directly loaded into LDOS itself.

You can boot the system with your own fonts and graphics in place. Powersoft has also included several examples of customized character fonts and graphics.

SETWRITER is a companion product to SETMAKER, and allows you to print your custom fonts and graphics on an Epson MX-8Ø or FX-8Ø, just as you see them on the screen! If you're using an MX-8Ø you must have GRAFTRAX. I was not able to test SETWRITER since I don't have either of these printers. Judging from SETMAKER (which ran flawlessly), I don't hesitate to recommend these programs. They're in machine language, and are about as "bullet proof" as can be. There are plenty of on-line menus-- the documentation is almost unnecessary.These are $29.95 each, or $5Ø for both. They're from:

PowerSOFT, 115ØØ Stemmons Freeway Suite 125, Dallas, TX 75229 (214) 484-2976

MODEL 4 TOPICS

Speaking of Powersoft, they've also just released their LDOS utilities for TRSDOS 6.x on the Model 4. It's the "Toolbelt for TRSDOS 6" and costs $49.95. Considering the (more than 15) useful programs you get, this has got to be the best bargain in town. I use versions of these utilities under LDOS 5.1 and TRSDOS 6, and find them to be excellent products. Contact Powersoft for more details.

A new word processor from Anitek called "LeScript" is available. One version runs on Mod 3, 4, or MAX (the package also includes a Mod 1 version). It supports 8Ø col on the latter two, even when running the Mod 4 in Mod 3 mode! LeScript also uses the extended memory available in the Mod 4 and MAX as text buffer. When you fire LeScript up on a MAX, for example, you're greeted with the pleasant sight of around 8ØK of text space.

LeScript also supports virtually every printer known! If you have a printer from RS, Epson, NEC, C. Itoh, ... you'll find its features implemented in LeScript. For example, I was able to use italics, underline, super- and sub-scripts on my MX-1ØØ. LeScript even lets me print in proportional mode by using the Epson's bit-image graphics.

One especially delightful aspect of LeScript's operation is that it runs just great with LDOS 5.1 / TRSDOS 6. Although the program doesn't ordinarily use the system's DCBs (ahem!) it DOES know enough to not interfere with operating system features. For example, you can leave type-ahead on when you enter LeScript, and when you're done you won't be greeted with lines of garbage.

For $129 you'll be hard-pressed to find a better overall word processing product. Though LeScript isn't virtual (the maximum text you can edit at any one time is limited by memory), it should accommodate the vast majority of word processing chores you can dream up. If I sound enthusiastic, I am! You will be too when you see this product. For more information, contact:

ANITEK Software Products, P.O. Box 361136, Melbourne, FL 32936 (3Ø5) 259-9397

MACHINE WARS

It never fails, almost invariably someone asks the question, "What should I buy, a Model 4 or a MAX?" I've used both quite a bit and I'm ready to give my informed opinion - "It depends!" From a pure performance point of view, the MAX wins hands down. In some cases, the MAX runs rings around much more expensive machines like the IBM-PC or the TRS-8Ø Mod 12. I also prefer the versatility of the MAX. It gives me two serial ports, runs eight inch floppies, and can boot from any type of disk drive, including hard disks.

There's another side to this story. It's clear that new TRS-8Ø software will be for LDOS/TRSDOS 6. LDOS 6.x is not now available for the MAX-8Ø, and may never be, due to hardware conflicts. If you use your machine in a commercial application and need to be compatible with the rest of the world, it seems that the MAX is not a viable choice unless you intend to use CP/M.

What about the Mod 4? Frankly, the Mod 4 never impressed me much. It has the same sleazy video that Tandy is notorious for, has no 8" drive capability, and only one RS-232 port. Worse yet, early Mod 4s apparently had flakey disk controllers and used wait states when accessing memory. The latter made this "4 MHz" machine run as much as 25% slower. (I am told that these problems have been resolved, and that current production Mod 4s work just fine.) But ... just wait 'till you see the Mod 4P (P=Portable)! It has great video, no wait states, and is built like no TRS-8Ø I've ever seen. I spent a day with a 4P, and as jaded as I am, I was impressed! This is a wonderful machine, and shows that Tandy IS paying attention to the market. I still miss 8" floppies because an 8" disk drive is an excellent compromise between price and storage capability. Other than that, the 4P is a "dream" machine.

Another consideration is support. Though LOBO has one of the very best warranties, as a mail-order operation they're not in the position to give instant help. Radio Shack, on the other hand, can help you locally and is in a better position to answer questions.

All things considered, the bottom line is this: If you're a software "tinkerer" who is reasonably knowledgeable and a performance hound, get a MAX. You'll love it! If you're fairly non-technical, and need a lot of "hand-holding" and support, buy from Radio Shack. What do I use? A MAX-8Ø! Though I enjoy using the Mod 4, I find the MAX a consistently overall better performer. Still, the Mod 4P is really tempting! Mebbe if I save my lunch money for a few years...

THE WRAP UP

That's it for now. Unfortunately, I've had to delay the review of the Model I to III upgrade I mentioned in last time. Hopefully I'll get another crack at it later in 1984.

The "C" Language (Part V)

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

INITIALIZATION, BLOCKS, POINTERS, ARRAYS

As you can tell from the C commented title for Part V, the subjects for discussion are blocks, pointers, how variables can be initialized and an introduction to arrays. Shall we start with more on blocks? (Were you expecting a choice?)

Several computer languages are block-structured in the sense that they allow functions to be defined within other functions. C does not allow this. In C, functions are always "external" since they are not inside of other functions. I am alluding to the fact that functions are blocks of C code. Remember from previous parts that a block is enclosed via {}. These braces {} enclose functions and other blocks. After the { comes variable definitions, if any. Variables in C, are thus defined in a block-structured manner.

Variables can be declared following the { that begins any compound statement. Also after the { that begins a function, variables can be declared (defined). If more variables need to be declared, later in the function, they can be, by declaring them after the left brace which begins a block. These variables can even have the same name as other variables. Their declarations "supersede" the identically named variables in outer blocks. They exist only within the block in which they are declared. Don't forget or confuse what you have learned previously about variable storage class and what you are learning now. The above comments on variables declared within blocks hold true for external variables too. Now can we look at how variables can be initialized? (No freedom of choice, is there?)

If you would like to assign an initial value to a variable when it is defined, C will allow it. As an interesting point, C does initialize certain variable classes for you. If you do not specifically assign an initial value to an external or static variable, C will initialize them to zero for you. However, automatic and register variables are not initialized automatically for you by C. So, don't count on them containing anything worthwhile unless you specifically initialize or assign a value to them. An equals sign and a constant expression are used to initialize simple variables. (Arrays and structures are initialized differently, as we shall C later.) Here are some examples of simple initialization:

```
int a = 5;
int b = c = d = e = Ø;
char g = 'x', h, i = 'y';
char f = 'f';
int d = 45 * 67;
```

As you can imply, this initialization saves "extra", sometimes unnecessary, assignment statements which assign a value to a variable. K&R call this shorthand for assignment statements. Remember what was just learned about blocks and how variables can be declared within them? Well, variables declared within these blocks (or functions) can also be initialized. This initialization takes place each time the function or block is "entered". External and static variables are initialized only once. (Are you wondering why this is? External and static variables are of different storage class and scope than automatic and register variables. Think about how and when these variables come into existence and when they go out of existence (if they do)?) Also, for automatic and register variables, the initialization can be done via any valid expression. This initializer is not limited to a constant expression.

Before I discuss how arrays can be initialized, shouldn't I discuss what they are and how they are declared (defined) ? For example:

```
int number[1Ø];
```

This declares an array of size 1Ø. In essence, this is a "block" of 1Ø integers together. Likewise:

```
char name[12];
```

declares a block (an array) of 12 characters. Each member of the array is called an element. Each element is numbered or indexed. In C the index starts at zero. In the number array above, the elements can be referred to individually via number[Ø], number[1], ..., thru number[9]. C also supports multi-dimensional arrays. For example:

```
int a[1Ø][2Ø];
```

This declares a two dimensional (rectangular) array. Elements of a multi-dimension array are stored by rows. Viewing storage as linear, elements of the array are seen in storage order if the right most index varies fastest. Now, how can arrays be initialized?

Arrays are initialized differently than other variables. Only external and static arrays can be initialized, automatic arrays can not be initialized. External and static arrays are initialized as shown in this example:

```
static int numbers[1Ø] = { Ø,1,2,3,4,5,6,7,8,9 };
```

Remember, in the absence of explicit initialization, all elements of external and static arrays are initialized automatically to zero.

In initializing external and static arrays, fewer initializers can be used than there are elements. In this case, the remaining elements will be zero. C also disallows more initializers than elements. Wouldn't it be nice to be able to repeat an initializer or just to initialize specific elements and ignore others? Well, sorry, C does not provide a means to do that.

Here is an example of a character array and its initialization:

```
                        /*  ....5...1Ø...15...2Ø...25 */
static char me[] = "Earl C. Terwilliger Jr.";
```

Quick! How many elements does the array me have? (Use the comments ruler line to help you count.) Did you guess correctly with 24? Each character between the quotes is an element plus the \Ø which is added by the C compiler to terminate the string. Did you notice that the size of the array, i.e., the number within the [] was omitted? If you do not include it, C will compute the size of the array for you based on the number of initializers. Another way to initialize a character array is as follows:

```
char name[] = { 'E', 'A', 'R', 'L', '\Ø' };
```

Notice that it is so much easier to use:

```
char name[] = "EARL";
```

Are you thinking that the initialization of a character array is like a "string copy"? If so, be careful in your evaluation of the following statements:

```
static char msg[5];
msg = "TEST";
```

This is not a string copy! C does not provide any operator for string copying or dealing with an entire string of characters as a single unit. Also, msg is the name of an array, it is a constant. It is not an lvalue and the above expression using it as such is ILLEGAL! How then can elements of an array be assigned values? The answer is by individually assigning values to each element. To "blank out" a character array, examine the C code which follows:

```
char message[2Ø];
...
for (i=Ø, i<2Ø, ++i) {
    message[i] = ' ';
    }
```

Also, note that the message array does not necessarily have to be external or static. It could be an automatic array!

Next, onward to pointers! A pointer is a C variable which contains the address of another variable. I can hear you thinking! You are no doubt asking, how does the pointer get the address? The unary operator & mentioned in an earlier part gives the address of its object. The & operator applies only to array elements and variables. Consider the following:

```
char a;
char *ptr;
...
a = 25;
ptr = &a;
```

In the expression: ptr = &a, ptr is assigned the address of a. By the way, there is no such thing as just a pointer. In C, pointers are always pointers to a particular data type. As shown above ptr is a pointer to type character. The * operator denotes indirection, it treats its operand as an address. It accesses this address to obtain the contents stored there. For example:

```
char *ptr, a, b;
b = 'x';
ptr = &b;
a = *ptr;
```

In the above examples, b is assigned the value 'x'. ptr is assigned the address of b. a is assigned the value of the character pointed to by ptr, which is 'x'. *ptr is a C mnemonic declared in this example to be a character. The combination of the * and ptr denote a character just like the above variable b does. When a pointer is declared, the type of data it points to is stated. The pointer is limited to point to data of that type. Also, pointers and pointer references are lvalues and can appear on the left side of assignment statements. Above, the pointer ptr is seen appearing on the left of an assignment statement. Below, *ptr is shown on the left of an assignment:

```
                char *ptr, a, b;
                b = 'x';
                ptr = &a;
                *ptr = b;
```

After the above statements are executed, a will contain the same value as b! *ptr is a pointer reference. In the case above it actually references a. ptr contains the address of a and *ptr references the character stored at the address in ptr.

Having the address of a variable is very useful. Remember from a previous part that C passes copies of variables as arguments to a called function. This is "call by value". The called function can not alter a variable in the calling function. (Actually, it could if the variable used in both functions was an "external" variable.) Now that you have learned about the & operand, you can use it to pass, as parameters to a function, addresses of (pointers to) variables. The called function can declare the arguments passed as pointers and alter the referenced data!

Looking back over the discussion on arrays, do you remember the problem of assigning values to an array? Consider this, now that you are familiar with arrays and pointers:

```
                char *myname;
                myname = "Earl C. Terwilliger Jr.";
```

This also is not a string copy! But it is a valid expression. myname is a pointer and it is assigned the address of the string! Comparing these two C statements with the ones shown to illustrate arrays, you should be wondering about the relationship between an array and a pointer. Actually an array name is a pointer expression. However, keep in mind that a pointer is a variable but an array name is a constant. If an array name is passed as an argument to a function, what is actually passed is the location (address) of the beginning of the array. (Using the & operator on just an array name is invalid. C does however, allow the & operator to take the address of an array element, for example &myname[4]. The & operator applies only to variables and array elements!)

A called function, when passed an array name as an argument, can declare the argument as a pointer and reference thru the elements of the array. Would you like an example?

```
        main() {
             static char myname[] = "Earl C. Terwilliger Jr.";
             char a;
             a = 'l';
             printf("%d\n",scount(myname,a));
        }
        scount (ptr,ch)
             char *ptr, ch;
        {
             int c = Ø;
             while (*ptr != '\Ø') {
                  if (*ptr++ == ch) ++c;
             }
             return (c);
        }
```

The function scount will return the number of occurrences of a given character in a given character string (array). The two parameters passed to it are the address of the string to search and the character to search for. If you follow the logic, pay particular interest to the *ptr++ expression. The value printed after the above code is executed should be 3! (What? You don't believe me? Type in the code and try it out on your favorite C compiler.)

Next time, you will see more on pointers and arrays. Structures will be introduced and I will point out some of the most common errors found in C programs.

Items of General Interest

Here are corrections and additional information regarding subjects raised last time:

Page 12:

The patch to restore "random" allocation should have been listed as for Model 3 and MAX-8Ø, LDOS 5.1.x only. The correct patch for Model 1, LDOS 5.1.x is:

```
PATCH SYS8/SYS.SYSTEM:Ø (DØØ,FE=D5 CD C1 44 D1 6C)
```

Page 14:

The new name for the TRSDOS 6.x communications software package is LS-Host/Term, Catalog number L-35-281, $199 plus $3 shipping and handling.

Page 55:

It has been reported that the following patch will correct a "Ø left" error with Model 1 SuperSCRIPSIT, Version Ø1.Ø2.ØØ: PATCH SCRIPSIT/CMD (X'7E22'=FC)

Page 57:

Here is the equivalent of the ROM/CTL patch, but for the Model 1. Comments are the same as the Model 3 patch:

```
. ROM/FIX
. Patches to the Model 1 SS 1.2 DW2/CTL driver for system DCB usage
DØØ,91=3D BF
DØ2,ØB=3E 3Ø ØØ
DØ3,2D=CD 35 BF
DØ3,45=CD 35 BF
X'BF35'=D5 F5 CD 3B ØØ F1 D1 C9
. End of patch
```

Page 64:

In the FDC driver patch, the first line should have ended <u>ED A2</u>, not <u>ED A4</u>. The SYS2 patch for drive timing is already present on most release versions of 6.1. The byte position of the SYS1 patch (for changing REMOVE back to KILL) should have been <u>C8</u>, not <u>CB</u>.

Patches, patches, patches ...

The following patch to LBASIC (versions prior to Ø9/31/83) will correct the operation of RUN"filespec",V for large programs:

```
. Patch to LBASIC/CMD (Model 1 ONLY!)
. corrects operation of RUN"",V
DØC,58=5E 64
D13,89=ED 62 39 D9 CD 4D 1B D9 F9 C9
. End of patch

. Patch to LBASIC/CMD (Model 3 and MAX-8Ø ONLY!)
. corrects operation of RUN"",V
DØC,6F=75 64
D13,AØ=ED 62 39 D9 CD 4D 1B D9 F9 C9
.End of patch
```

The following patch to FM will correct a problem with not moving certain files:

```
. Patch to FM 5.1 to correct not moving certain HIT positions
DØF,4E=Ø2
D19,62=62
. End of patch

. Patch to FM 6.x to correct not moving certain HIT positions
DØF,67=Ø2
FØF,67=ØØ
D19,C6=63
F19,C6=62
. End of patch
```

The following patch to QFB (all 5.1 versions) will provide for proper operation on double-sided media, and prevent a conflict with READ4Ø source drives:

```
. Patch QFB/CMD (5.1.x)
. corrects two-sided & READ4Ø operation
DØ2,26=ØØ ØØ ØØ
X'5AAE'=CD C6 6Ø
X'6ØC6'=CD 96 5C FD CB Ø3 A6 C9
. End of patch
```

The following patch corrects a problem in the version of XMODEM provided with the LS-Host/Term communications package:

```
. Patch to XMODEM from LS-Host/Term
. corrects problem with setting 8 bit word mode
DØ1,9E=C9
FØ1,9E=28
. End of patch
```

The following patch corrects an error in the SVC table for Model 1 LDOS 5.1.3 and 5.1.4

```
. Patch to correct SVC table entries for Model 1 LDOS 5.1.3 & 5.1.4
. this patch is to SYS7/SYS
D11,8B=44 3Ø 44 33 44
. End of patch
```

LDOS: How it works - The BACKUP utility discussed

BACKUP functions and procedures discussed
or--- You can never have too many backups
by Joseph J. Kyle-DiPietropaolo

Long, long ago in a galaxy far, far away... oops- sorry. When the idea of a BACKUP utility was first implemented in a TRS-8Ø type DOS (Model 1 TRSDOS), the only designated purpose was to produce exact duplicates of existing diskettes. The BACKUP utility on LDOS 5.1.x and TRSDOS 6.x, however, wears many hats to serve a variety of purposes.

The first is, of course, to produce exact duplicates. One major difference between this mode of LDOS/TRSDOS 6.x BACKUP and the original variety is that LDOS/TRSDOS 6.x BACKUP (henceforth known as BACKUP) requires that the destination diskette be FORMATted first. The reason for this is simple: LDOS can handle many different disk drive setups. BACKUP can handle all of these, but only if the diskette was previously processed and made usable by LDOS through the FORMAT utility.

To produce an exact duplicate of a diskette, several things must be true about both the source and destination disks.

1) Both drives must be the same type. That is, they must both be five inch or both eight inch, the same density (single or double), and have the same number of sides.

2) Neither the source nor destination drive can be a hard disk system.

3) The destination drive must have an equal or greater number of cylinders than the source drive. For most people, a cylinder is the same as a track, but double-sided drives and hard disk systems actually have cylinders. A cylinder is a collection of tracks grouped together as one logical unit.

4) If the destination diskette has flaws (indicated during the FORMAT process), they cannot be on a cylinder that is occupied on the source drive. Generally, flawed diskettes should be discarded in any case.

When these conditions are met, and none of the special BACKUP parameters are specified (as described further on) BACKUP will be able to do what is called a "Mirror-image BACKUP". This is a misnomer-- the data is not reversed, as it would be in a mirror, but is copied identically from the source to the destination cylinder by cylinder.

This is the most common type of BACKUP. All other forms of BACKUP operations fall into the category of "BACKUP-by-class". If one or more of the above conditions are not met, then the BACKUP is done by copying each file on the source drive to the destination drive, one at a time. During this type of BACKUP, the BACKUP utility will display messages to indicate the type of BACKUP. "Backup-by-class invoked" means that the process was caused by a specification on the part of the user. "Backup-reconstruct invoked" means that BACKUP detected that one of the above conditions was not true.

Well, you may ask, what is a "BACKUP-by-class" good for? For this we must dig a little deeper into the parameters and specifications of BACKUP. One useful specification is the "partspec". A partspec is a portion of a normal LDOS file specification. For example, to move all files with the extension of /BAS, and that begin with "M". The command "BACKUP M/BAS:Ø :1" could be used. The special partspec of $ means "all files".

To move groups of files, parameters can be used. For instance, "BACKUP :Ø :1 (MOD)" would copy all files that had been modified since they were last backed up. In a DIR, this modified condition is indicated by a "plus" (+) sign next to the file.

Other parameters are available to backup files based on dates, visibility status, protection level, and whether or not the file already exists on the destination drive. With this introduction, all users should be able to use BACKUP more efficiently.

But what about frequency of backups? As a general rule, backups should be made at any significant break in a data processing procedure. That means at least every day a diskette is used. If a lot of processing is done, it wouldn't be a bad idea to perform backups more often, perhaps at mid-day in addition to at the end of the working day.

And how many sets of diskettes? Three is considered the absolute minimum. The sets should be used in a rotation system. For example, let's label the sets of disks A, B, and C. On the first day, set A is used. At the end of the day, set A is backed up onto set B. The next day, set B is used for processing. Set B is then backed up onto set C. Set C is used the next day, and at the end of the day, set C is backed up onto set A, and the cycle continues. In this manner, no set is used two days in a row, and new work is always done on the backup to ensure its integrity.

Many companies use five sets, labeled Monday through Friday. This helps prevent confusion as to what set is to be used, and provides additional backup protection. Each set is used on its labeled day, then backed up onto the next day's set. Six sets could be used if work is to be done on Saturdays.

What about re-formatting? Many people advocate periodically bulk-erasing and re-formatting the destination disk before a backup. This is a good idea, as this would be the only time that currently unused portions of the diskette would be checked for potential flaws.

What about diskettes themselves? Diskettes should be labeled with the date they are put into service. After a period of time, typically six months, they should be replaced with new diskettes, even if no difficulties were noted in their operation. The cost of even a premium diskette is trivial when compared to the value of the data it stores.

Sending Characters to a Printer Via the Keyboard with a Single Keystroke (Whew!)

by Dick Konop

An interesting customer service request prompted this article. The nature of the dilemma goes something like this: How can one pass an um-teen character control sequence to a printer directly from the keyboard with a single keystroke? One answer to this problem can be found in the use of KSM and MINIDOS.

The MINIDOS filter has a command (<CLEAR><SHIFT><P>) which will allow a (two character) hex byte to be entered from the keyboard. This byte is then sent to the printer. The KSM filter allows multiple keystrokes to be defined as a single key (each of the keystrokes <CLEAR><A> through <CLEAR><Z> can represent a different sequence of characters). To attain our final goal, a KSM file can be created which will invoke the MINIDOS filter, and pass it the hex control bytes.

The best way to illustrate this is by example. Let us assume that the bytes X'1B' and X'ØF' need to be sent to the printer to produce the desired result. First, create a KSM file. One method of producing a KSM file is with the BUILD library command. If the BUILD command is used, the HEX parameter must also be specified. For example:

```
BUILD MOOSE/KSM (HEX)
```

After issuing the BUILD command, the prompt A => will appear on the screen (if the extension for the filespec was /KSM). Respond to this prompt by entering the following characters (note that the spaces are for readability only, and must not be entered).

```
FØ 31 42 3B FØ 3Ø 46 3B ØD
```

Once the KSM file has been built, the KSM and MINIDOS filters must be applied to the keyboard. The order in which the filters are applied is important. The KSM filter must be applied first, followed by the MINIDOS filter. For example:

```
FILTER *KI KSM MOOSE
FILTER *KI MINIDOS
```

After this has been done, depressing the <CLEAR><A> key sequence will cause the characters X'1B' and X'ØF' to be sent to the printer.

A total understanding of what is happening is not required to use this concept. It is important to note that for each byte sent to the printer, four bytes are needed in the KSM file. The first byte will always be X'FØ'. This is the character that will cause the MINIDOS "P" function to be activated. The next two bytes in the KSM file are the hex values corresponding to each hex digit in the byte being sent to the printer. That is to say, the "31" and "42" are the hex representations for the characters "1" and "B", respectively. These form the byte that will be sent to the printer (in this case X'1B'). The last byte in the four byte sequence will always be X'3B'. This is a semicolon character, and is translated by KSM into an <ENTER> (X'ØD'). Finally, there must be a terminating X'ØD' byte at the end of the KSM key assignment. This acts as a terminator for the KSM key definition. The X'ØD' marks the end of all assignments made to the <CLEAR><A> KSM key.

Please note that when a KSM printer control key is pressed, the actual MINIDOS commands will appear on the screen (just as if they had been typed in). This type of printer control should be useable from within any program that allows use of KSM and MINIDOS.

THE JCL CORNER - by Chuck

For the last two years, I have attempted to use this column to shed some light on the subject of Job Control Language. Through examples both of my own design and also those of other readers, the many aspects of JCL have been examined and described. Rather than rehash all of this material again, I would rather devote this space to answering specific questions about the application of JCL procedures.

One particular question about using JCL procedures keeps coming to the attention of our customer service department. The question, "How can I use JCL to run (a particular program)?", can't always be answered with a simple set of instructions. Some programs as written can be run and controlled via a JCL procedure, while others can't. There are those that can be partially controlled, but still require some user keyboard input.

Future columns will attempt to deal with both previously mentioned subjects; answering specific user questions, and explaining how existing programs can be controlled with a JCL procedure. In addition, I will attempt to explain how a program can be designed to allow a JCL procedure to control it from start to finish. If any of you have questions, comments, or interesting uses for JCL, send them to LSI, attention "JCL Chuck".

Cumulative Index to LDOS Quarterly Volumes 1 and 2 - Subject

"Active Variable Dump for LBASIC" - Alan Moyer		v2n3p15	v2n4p38
"Alcor Pascal" - Scott Loomer			v2n1p18
Allocation, disk			v2n4pØ8
AM Electronics, disk controller			v1n3pØ3
"APL*PLUS/8Ø - A System Overview" - Daniel Lofy & Lee Rice			v2n1pØ9
"Article, An" - Charlie Butler			v1n4p4Ø
"ASCII File Listing Utility for The BASIC Answer" - Jeffrey Brenton			v2n3p19
Assembly language basic concepts	v2n2p58	v2n3p46	v2n4p4Ø
Assembly language patching		v1n6p38	v2n1p46
Assembly language programming tips		v1n5p14	v1n6p64
"At Large" - Earl Terwilliger			v1n6p5Ø
"Automatic Chaining with JCL" - Jim Kyle			v2n4p52
"BASIC and File Structure - A Beginner's View" - Wes Goodnough			v1n6p2Ø
"BASIC Concepts - The RUN,V Command" - Dick Konop			v1n6p69
"Beta Tester, I was an LDOS" - Tim Daneliuk			v1n3p16
Byte I/O	v1n3p38	v1n6p76	v2n2p52
under LDOS 6.x			v2n3p56
'C' graphics			v2n4p16
"'C' language, The" - Earl Terwilliger:			
general introduction			v2n1p15
functions, variables, constants, expressions			v2n2p35
operators			v2n3p39
logic, control & flow			v2n4p34
initalization, blocks, pointers, arrays			v2n5p31
"'Card' Utility" - Paul Tonini			v2n2p18
"Case of Mis-Allocation, A" - Bill Schroeder			v2n4pØ8
Case mode indicator			v2n2p11
Changing operating systems			v1n4p4Ø
"Clock Speedup Kits with LDOS, Using" - Tim Mann			v1n1p13
Cobol			v1n6p32
"Color Comes to the TRS-8Ø's" - Scott Loomer			v2n3p28
"Communicating Micro, The" - Gordon Thompson		v1n6p12	v2n1p28
Communications		v1n5p4Ø	v1n6p83
"Communication Host" - James Bruckart			v2n1p35

Cumulative Index to LDOS Quarterly Volumes 1 and 2 - Subject

"Confessions of a Machine Language Addict" - Ray Pelzer v1n6p38
Configuring with non-relocatable code v2n4p5Ø
Customer service tips v2n1pØ7
Cylinder term explained v1n1pØ4
DAM's, old v1n5p19
"Data Address Marks" - Roy Soltoff v1n1pØ5
Date conversions v2n4p6Ø
"Device I/O and Independence, LDOS" - Roy Soltoff v1n3p38
Disk allocation schemes v2n4pØ8
Disk byte I/O v1n3p38 v1n6p76 v2n2p52
Disk controller, AM Electronics v1n2pØ3
Disk drive first access delay v2n4p63
"Disk Drive Control Linkages, LDOS" - Bob Bowker v1n6p43
Disk drives (8") on the Model III v2n3pØ8
Disk drives, non-Radio Shack v2n3p55
Disk drive poll v2n2p27
Disk drive select time hardware fix (Mod I) v1n3p18
"Disk I/O in Assembler" - Doug Kennedy v2n2p55
Disk speed (3ØØ RPM delays) v1n4p11
"Double Sided Drives with LDOS" - Tim Mann & Roy Soltoff v1n2p37 v1n3p56
"Easy LScript" - James Bruckart v2n2p22
"Easy VisiCalc" - James Bruckart v2n3p23
"EDAS IV 'Z' Command" - Earl Terwilliger v2n1p37
Editor, LDOS command line v2n5pØ7
Electric Webster with Newscript, LDOS and Sole v1n6p55
"Electronic Inbasket, The" - Gordon Thompson v2n4p15
"ELSIE - The Contented Compiler" - Jim Frimmel v1n3p21
Epson MX-8Ø tips v1n5p18
"..... er ....." - Earle Robinson:
  assembly language efficiency v1n6p64
  printers, TBA, UTILZAP v2n1p39
  printers, word processors, MNet, 'C', LDOS 6.x v2n2p26
  'C' book, SS drivers, Model 4, PROMPT/CMD v2n3p33
  UNIX, IBM PC, Telex v2n4p28
Expansion interface v1n4p39
EZEDIT - The LDOS command line editor - Graham Brown v2n5pØ7
"Fast Graphics for 'LC'" - Scott Loomer v2n4p16
File listing utility for TBA v2n3p19
File structure, BASIC v1n6p2Ø
Filter linkage v1n3p47
"Fortran, Cobol and LDOS JCL" - Glen Rathke v1n6p32
Fortran with interrupts and SVC's v2n4p19
"Greek to Me, LDOS - It's" - Charles Knight v1n5p15
"Hayes Smartmodem, LDOS and a" - John Mullin v1n6p34
High memory, avoiding memory conflicts v1n4p13
High memory module header v1n3p46
Host, communications v2n1p35
Index, LDOS Quarterly/LSI Journal - Scott Loomer v2n5p39
"Inside the Expansion Interface" - Earle Robinson v1n4p39
"Interrupts and SVC's in Fortran, Using" - J. Bender v2n4p19
JCL chaining v2n4p52
"JCL Corner" - Chuck Jensen:
  clear screen v1n1pØ3
  file copying v1n2p3Ø
  compilation macros v1n3p51
  hex codes, more compilation macros v1n4p46
  keyboard input v1n5p34
  logical operators, keyboard filters, LScript patch v1n6p72
  creating and defining JCL procedures v2n1p54
  backups v2n2p49
  JCL with FORTRAN and Cobol v1n6p32

LBASIC:
- active variable dump v2n3p15 v2n4p38
- "CMD"O" implementation and possible uses" v1n4p15
- default extension patch v1n5p19
- file structure v1n6p2Ø
- notes (items missing from 1st printing of the LDOS manual) v1n1p09
- RUN,V command v1n6p69
- USR routines (relocating to high memory) v1n3p47

LDOS 5.1.4, new features v2n4pØ3

LDOS 6.x
- announced v2n2p46
- licensing v2n4pØ7
- technical manual v2n4pØ7
- patches v2n5p35 v2n4p64

"LDOS: How it Works" - Joe Kyle-DiPietropaolo:
- PATCH utility v2n1p52
- REPAIR, CONV and COPY23B utilities v2n2p44
- non-Radio Shack disk drives v2n3p55
- configuring with non-relocatable code v2n4p5Ø
- BACKUP utility v2n5p36

"Les Information" - Les Mikesell:
- communications v1n5p4Ø
- RS232 drivers v1n6p83
- SYSTEM (FAST) and (SLOW) commands v2n1p58
- byte I/O v2n2p52
- byte I/O under LDOS 6.x, CTLP/FLT v2n3p56
- @PARAM under LDOS 6.x v2n4p61

"Let Us Assemble" - Rich Hilliard:
- the basics v2n2p58
- using DEBUG, sorts v2n3p46
- number base conversion v2n4p4Ø

"Library, The" - Earle Robinson v1n4p28

Library commands:
- LOAD documentation correction v2n1p47
- SYSTEM (FAST) v1n1p13 v2n1p58
- SYSTEM (SLOW) v2n1p58

Limited backup policy explained - Bill Schroeder v2n3pØ4 v2n4pØ6

"Linking to LDOS in Assembly" - Roy Soltoff v1n1p12

"LISP Implementations for the Z-8Ø" - Lee Rice & Daniel Lofy v1n6p26

Listing utility for TBA v2n3p19

LScript made easy v2n2p22

"LScript Patches to Add Versatility" - Scott Loomer v1n6p45 v2n1p47

Load module structure v1n4p42

Lower case lock v1n4p11

LSI Journal submission and subscription policies v2n4pØ2

MAX-8Ø LDOS described v2n1p5Ø

"MAX-8Ø Memory Map" - Chuck Jensen v2n4p58

Magazines v2n3p35

Manual story, with corrections to 1st edition v1n1p16

Memory map v2n1p34

Minimum configuration disk v1n2p29

"Mixing Newscript, Electric Webster, LDOS and Sole" - Jerry Latham v1n6p55

Modifying Mod 3 RTC Interrupts - Andrew Grandsen v2n5p22

"My BASIC Answer" - E. Cheatham (TBA review) v2n3p17

"Newscript 7.Ø and REFLEX" - Gordon Thompson v2n1p28

"Newscript and The BASIC Answer" - Jerry Latham v2n2p2Ø

"New Version - EDAS IV, A" - Marc Leager v1n6pØ9

Number base conversion v2n4p4Ø

Old DAM's v1n5p19

Parity errors v1n1pØ4

Cumulative Index to LDOS Quarterly Volumes 1 and 2 - Subject

"*** PARITY = ODD ***" - Tim Daneliuk:
  introduction, DATAENTR 2ØØ & ISAM 2ØØ v1n4p17
  tips for better programming, DISCATER, Filter Disk v1n5p11
  BASF drives, drive poll, TAS, MODEM8Ø, HEXSPELL II, Utilities, HELP v1n6p59
  Tandy, MAX-8Ø, AEROCOMP, Electric Webster, The BASIC Answer v2n1p42
  disk drive poll, gold plugs, LX-8Ø, Proofreader, Scripsit Dict. v2n2p27
  magazines v2n3p35
  rumors, LX-8Ø software compatibility, TRSDOS 6.x software v2n4p31
  CP/M, LeScript, M4 vs. MAX-8Ø v2n5p28
"Partitioned Data Sets, MISOSYS Announces" - Roy Soltoff v1n3p23
"Pascal 8Ø, LDOS and" - D. Hill v2n1p22
Passwords for LDOS 5.1.x v1n1pØ3
Patch, how to v2n1p52
"PDS - Standard and Other Types of Uses" - Scott Loomer v2n1p24 v2n3p45
"Performing Date Conversions in BASIC" - Dick Konop v2n4p6Ø
PR/FLT, changing parameters v2n5p13
Printer codes, sending v2n5p38
Printers v2n1p39
Profile III Plus with LDOS v2n4p55 v2n5p26
Profile One Plus (Pro3+ on the Mod 1) - E. Sturiale v2n5p26
PRPARM - changing PR/FLT parameters - Graham Brown v2n5p13
Quarterly reader survey results v2n2pØ6
Radio Shack v2n1p42 v2n3pØ3
Recursive v2n5p42
"Relocating Code for LBASIC USR Routines" - Chuck Jensen v1n3p47
Reviews:
  Alternate Source (TAS) - Tim Daneliuk v1n6p6Ø
  APL*PLUS/8Ø (STSC) - Daniel Lofy & Lee Rice v2n1pØ9
  CHROMAtrs (South Shore Computer Concepts) - Scott Loomer v2n3p28
  DATAENTR2ØØ & ISAM2ØØ (Johnson Associates) - Tim Daneliuk v1n4p18
  EDAS IV (MISOSYS) - Marc Leager v1n6pØ9
  Electric Webster (Cornucopia) - Tim Daneliuk v2n1p45
  HELP (MISOSYS) - Tim Daneliuk v1n6p63
  HEXSPELL II (Hexagon) - Tim Daneliuk v1n6p61
  LDOS Utilities (Powersoft) - Tim Daneliuk v1n6p62
  LISP (Supersoft) - Lee Rice & Daniel Lofy v1n6p29
  MAX-8Ø (Lobo) - Tim Daneliuk v2n1p44
  Microcomputer Math book (SAMS) - Earle Robinson v1n5p1Ø
  MODEM8Ø (LSI) - Tim Daneliuk v1n6p6Ø
  Pascal (Alcor) - Scott Loomer v2n1p18
  Pascal-8Ø (New Classics) - D. Hill v2n1p22
  PDS (MISOSYS) - Scott Loomer v2n1p24
  Printers - Earle Robinson v2n1p39
  Profile III+ (Radio Shack) - Sam Goldberg v2n3p25
  Proofreader (Aspen Software) - Tim Daneliuk v2n2p3Ø
  Scripsit Dictionary (Radio Shack) - Tim Daneliuk v2n2p3Ø
  Structured BASIC Translator (Acorn) - Sue Ratkowski v1n4p21
  The BASIC Answer (LSI) - E. Cheatham v2n3p17
  The BASIC Answer (LSI) - Tim Daneliuk v2n1p46
  UOLISP (Far West) - Lee Rice & Daniel Lofy v1n6p3Ø
Routing a device v1n3p38
"Roy's Technical Corner" - Roy Soltoff:
  load module structure
  task processor v1n4p42
  error handling during byte I/O, @CKDRV, KFLAG$, @ICNFG, @KITSK v1n5p2Ø v1n6p76
  LDOS 6.x v1n6p76
RS-232 drivers v2n2p46
RTC Interrupt, Model 3 v1n6p83
"Running 8-Inch Drives on the Model III Under LDOS" - Peter Simon v2n5p22
Sector I/O v2n3pØ8
Sending Printer codes v2n2p55
v2n5p38

Cumulative Index to LDOS Quarterly Volumes 1 and 2 - Subject

"SOLEFIX - Fix that GAT Error" - Erik Ruf v2n2p14 v2n4p38
Speedup kits v1n1p13
SYNONYM - The LDOS command synonym processor - Henry Melton v2n5p14
Sysgen and type-ahead v1n1pØ4
Sysres, Quickly with JCL v2n4p12
System routines:
  @ADTSK, @RMTSK, @KLTSK, @RPTSK - task proc. v1n5p31 v1n6p76 v2n4p19
  @CKDRV - determining 5.1.2 vs 5.1.3 for @CKDRV adjustment v1n6p78
  @CKDRV - moved v1n5pØ6
  @CMNDI - command interpreter v1n1p12
  @CTL, @GET, @PUT - byte I/O v1n3p38
  DAY$ - day of the week v1n4p11
  @GET and @PUT under LDOS 6.x v2n3p56
  @ICNFG - configuration interfacing v1n6p81
  @KITSK - keyboard task v1n6p82
  KFLAG$ - keyboard scanner v1n6p78
  "@PARAM, Using" - Roy Soltoff v1n2p31
  @PARAM under 6.x v2n4p61
  "@PARAM, @DSPLY, @EXIT and INBUF$ for Everyone" - David Vinzant v2n2p23
  @RAMDIR documentation correction v2n2p39
  SVC's and Fortran v2n4p19
Tandy v2n1p42 v2n3pØ3
Task processor v1n5p2Ø v1n6p76
"T-Timer, LDOS Supports the" - Roy Soltoff v1n4p12 v1n5p18 v1n6p66
TRSDOS (Mod I) to LDOS (Mod III) transfer without REPAIR v1n4p58 v2n2p44
TRSDOS 6.x (see also LDOS 6x.) v2n2p46
Update policy explained - Bill Schroeder v1n2pØ2 v2n4pØ4
Upper case lock v1n4p11
Users group directory v2n1pØ6
Utilities:
  CONV v2n2p44
  COPY23B v2n2p44
  PATCH v2n1p52
  REPAIR v2n2p44
Version number explanation v1n2pØ2
VisiCalc made easy v2n3p23
"VisiCalc with LDOS, Using" - Roy Soltoff v1n2p2Ø

Cumulative Index to LDOS Quarterly Volumes 1 and 2 - Programs

ADDLF/FLT - James Bruckart - add a line feed after carriage return v2n1p36
ALIVE/ASM - Earl Terwilliger - task processing demo v1n6p53
BEEP/FLT - Tim Mann - beep cassette port on each keystroke v1n1p22
BINHEX/BAS - Tim Mann - convert hex to binary v1n1p21 v1n4p56
  Improved version with checksum v2n2p42
CARD/BAS - Paul Tonini - file utility v2n2p18
CASE/CMD - Rick Tobias - Mod I case mode indicator v1n2p11
CIRCLE/CCC - Scott Loomer - fast graphic circle routine v2n4p16
CLFLT/FLT - Les Mikesell - general purpose communications filter v1n6p87
CLS/CMD - clear screen command for use in JCL (very short!) v1n6p67
COM/BAS - Earl Terwilliger - short BASIC dumb terminal program v1n6p5Ø
COMM1/ASM - Earl Terwilliger - machine language dumb terminal program v1n6p51
CONFIG8/CMD - Peter Simon - 8" drives on the Model III v2n3pØ8
CTLP/FLT - Les Mikesell - link printer and display v2n3p57
DATE/BAS - Dick Konop - date conversions v2n4p6Ø
DCAL/BAS - Tim Mann - disk drive speed check on Mod I v1n1p26
DELETE/CMD - Renato Reyes - multiple file killer v1n4p56
DIRMAP/BAS - Dave McLaughlin - granule map of disk allocation v1n2p41
DISKMAP/BAS - Bill Fields - alphabetized disk file mapper v1n4p5Ø
EZEDIT/FLT - Graham Brown - LDOS command line editor v2n5pØ7

Cumulative Index to LDOS Quarterly Volumes 1 and 2 - Programs

FAST/CMD - Tim Mann - fast clock on the LX-8Ø v1n1p24
FASTSPL/ASM - fast despooling for Model I 5.Ø.3 v1n6p68
FILL/CCC - Scott Loomer - non-recursive graphic fill routine v2n4p16
INBASKET/BAS - Gordon Thompson - communications message collector v2n4p15
IOTA/FLT - Charles Knight - modification to SLASHØ/FLT v1n5p16
KILL/CCC - Earl Terwilliger - multiple file killer v2n2p39
LBASIC/OV4 - Alan Moyer - active variable dump for Model I v2n3p15
LIST/TBA - Jeffrey Brenton - file lister for TBA v2n3p19
MINIDOS/FLT - Roy Soltoff - MiniDOS for 5.Ø.x Mod I users v1n2p44
MODCMD/CMD - John Mullin - send commands to a Hayes modem v1n6p34
MX8Ø/FLT - Ken Roser - block graphics on MX-8Ø w/o Graftrax v1n1p22
MX8Ø/FLT - improved version of above v1n4p59
NODAM/CMD - Chuck Jensen - read non-LDOS disks without REPAIR v1n4p58 v2n2p44
ONLINE/FLT - John Mullin - carrier detection filter v1n6p37
PASSWORD/BAS - James Bruckart - password check for host systems v2n1p36
PENCTL/FLT - Tim Mann - for Electric Pencil users v1n1p22
PDUBL/CMD - original Percom doubler driver (for illustration only) v1n1p24
PROMPT/CMD - Earle Robinson - change the 'LDOS Ready' prompt v2n3p35
PRPARM/CMD - Graham Brown - changing PR/FLT parameters v2n5p13
REFLEX/FLT - Gordon Thompson - communications program v1n6p16
RENAME/BAS - Charles Knight - BASIC file renamer v1n5p42
SALVAGE/BAS - transfer data from TRSDOS 2.3B disks v1n2p15 v1n4p62
SLASHØ/FLT - print slashed zeroes v1n2p41
SLOSTEP/DCT - Les Mikesell - head settling delay for some DS drives v2n1p6Ø
SOLEFIX/ASM - Erik Ruf - fix SOLEd disk for compatibility v2n2p14
SYNONYM/CMD - Henry Melton - command synonym processor v2n5p14
TCHRON/ASM - Roy Soltoff - time board support routine v1n6p66
TERMINAL/BAS - dumb terminal program v1n6p86
TIME5Ø/CMD - Andrew Grandsen - modify M3 RTC processing (5ØHz) v2n5p22
TIMEDI/FOR - J. Bender - Fortran interrupt and SVC demo v2n4p22
UPCASE/FLT - Tim Mann - convert characters to upper case v1n1p24
UPDATE/BAS - Tim Mann- update system disks (replaced by BACKUP) v1n1p23
USRFREE/CMD - Chuck Jensen - LBASIC free memory command v1n1p24

Cumulative Index to LDOS Quarterly Volumes 1 and 2 - Patches

BASIC compiler (RS) patch v1n4p62
COBOL patch v1n4p62
DEBUG ASCII mode patch v1n5p45
DIRCHECK/CMD (LSI Utility Disk #1) patch for hard or DS drives v2n2p44
DTERM patch - Peter Trenholme v1n5p18
EDAS IV "Z" command patch - Earl Terwilliger v2n1p37
FED patch v2n1p47
File Manager patch for moving HIT entries v2n5p36
File Manager patch for moving system files v2n4p38
HIGH patch for screen full pause v2n4p39
I/O bus disable for boot on Model III v2n3p46
KI/DVR patch to extend keyboard debounce time v2n3p45
LBASIC default extension patch v1n5p19
LBASIC remove execution banner v2n5p15
LBASIC RUN"",V fix v2n5p35
LED xlate parameter patch v2n1p48
Lower case driver forced on boot (Mod I) v2n3p46
LScript patch to require filespecs for S or L v2n1p62
LScript JCL patch v1n6p73
LScriptX patches - Scott Loomer v1n6p45 v2n1p47
MAP (LSI Utility Disk #1) v2n1p47
MAX-8Ø patch to prevent printer driver from converting LF's to CR's v2n2p39
MAX-8Ø patch to set LF after CR v2n4p39
Microsoft patches v1n2p16

Cumulative Index to LDOS Quarterly Volumes 1 and 2 - Patches

Microsoft MACRO-8Ø patches - Richard Deglin			v1n6p96
MiniDOS patch to allow installation before KSM			v2n1p49
Mod 3 speedup patch			v2n4p39
Newscript patch to use with REFLEX - Gordon Thompson			v2n1p3Ø
Password (master) disable patch			v1n5p19
PDUBL patch to write old DAM			v1n5p19
PR/FLT patch to disable chr$(6) trapping			v2n1p48
Profile patches - Dick Yevich			v1n4p59
Profile patch to fix buffer problem			v2n1p47
Profile 3 Plus, HD version - for use on Mod 1 - E. Sturiale			v2n5p26
PDUBL patch to write old DAM			v1n5p19
QFB - Quick Format and Backup fix for READ4Ø and DS drives			v2n5p36
SOLE patch to allow use of Super Utility - Erik Ruf			v2n2p14
SuperScripsit patch - Model 1 version Ø1.Ø2.ØØ			v2n5p35
SuperScripsit patches - Dennis Brent & Renato Reyes			v1n6p92
SuperScripsit patches - Joseph Kyle-DiPietropaolo			v2n4p55
SYSTEM(SLOW) and (FAST) patch			v2n1p58
TCHRON, TIMEDATE8Ø & TRSWATCH time & date boards patch		v1n4p12	v1n6p66
The BASIC Answer patch for local variable DC			v2n4p39
T-Timer patches	v1n4p12	v1n5p18	v1n6p66
TRSDOS patch to read REPAIRed LDOS disks on Mod I			v1n2p13
VIDSAV/CMD (LSI Filter Disk #2) patch to allow JCL installation			v2n2p43
VisiCalc patches		v1n2p2Ø	v1n5p43
Mod III enhanced - Ray Pelzer	v1n5p43	v1n6p38 v2n1p46	v2n3p45
RS232 transfer			v1n6p19
WRTEST (LSI Utility Disk #1)			v2n1p47
XMODEM - from LS-Host/Term - patch to fix 8-bit init			v2n5p36

Cumulative Index to LDOS Quarterly Volumes 1 and 2 - Authors

Bender, J. - "Using Interrupts and SVC's in Fortran"	v2n4p19
Bowker Bob - "LDOS Disk Drive Control Linkages"	v1n6p43
Brent, Dennis - SuperScripsit Patches	v1n6p92
Brenton, Jeffrey - "An ASCII File Listing Utility for TBA"	v2n3p19
Brown, Graham - "EZEDIT - The LDOS command line editor	v2n5pØ7
"PRPARM - Changing PR/FLT parameters"	v2n5p13
Bruckart, James - "Communication Host" (PASSWORD/BAS & ADDLF/FLT)	v2n1p35
"Easy LScript"	v2n2p22
"Easy VisiCalc"	v2n3p23
Butler, Charlie - "An Article" (changing operating systems)	v1n4p4Ø
Cheatham, E. - "My BASIC Answer"	v2n3p17
Daneliuk, Tim - "I was an LDOS Beta Tester"	v1n3p16
"*** PARITY = ODD ***": DATAENTR2ØØ & ISAM 2ØØ	v1n4p18
"*** PARITY = ODD ***": Message to Hackers	v1n5p11
"*** PARITY = ODD ***": TAS, HEXSPELL II, MODEM8Ø, HELP, Utilities	v1n6p59
"*** PARITY = ODD ***": Tandy, MAX-8Ø, Electric Webster, TBA	v2n1p42
"*** PARITY = ODD ***": disk drive poll & spelling checkers	v2n2p27
"*** PARITY = ODD ***": magazines	v2n3p35
"*** PARITY = ODD ***": rumors, LX-8Ø, TRSDOS 6.x software	v2n4p31
"*** PARITY = ODD ***": CP/M, LeScript, M4 vs. MAX-8Ø	v2n5p28
Deglin, Richard - Microsoft MACRO-8Ø patches	v1n6p96
Fields, Bill - DISKMAP/BAS	v1n4p5Ø
Frimmel, Jim - "ELSIE, The Contented Compiler"	v1n3p21
Goldberg, Sam - "Review of Profile III+ for LDOS"	v2n3p25
Goodnough, Wes - "BASIC and File Structure - A Beginner's View"	v1n6p2Ø
Hall, D. - "LDOS and Pascal-8Ø" (New Classics review)	v2n1p22
Knight, Charles - "LDOS, It's Greek to Me" (Greek translation)	v1n5p15
RENAME/BAS	v1n5p42
Kyle, Jim - "Automatic Chaining with JCL"	v2n4p52
Latham, Jerry - "Mixing Newscript, Electric Webster, LDOS and Sole"	v1n6p55

Cumulative Index to LDOS Quarterly Volumes 1 and 2 - Authors

Latham, Jerry - "Newscript and The BASIC Answer" v2n2p20
Leager, Marc - "A New Version - EDAS IV" (MISOSYS review) v1n6p09
Lofy, Daniel - "APL*PLUS/80 - A System Overview" (STSC review) v2n1p09
"LISP Implementations for the Z-80" (Supersoft & Far West reviews) v1n6p26
Loomer, Scott - "Alcor Pascal" (review) v2n1p18
"Color Comes to the TRS-80's" (CHROMAtrs review) v2n3p28
"Fast Graphics for 'LC'" v2n4p16
"LScript Patches to Add Versatility" v1n6p45 v2n1p47
"PDS - Standard and Other Types of Uses" (MISOSYS review) v2n1p24
Melton, Henry - "SYNONYM - LDOS command synonym processor" v2n5p14
McLaughlin, Dave - DIRMAP/BAS v1n2p41
Moyer, Alan - "Active Variable Dump for LBASIC" v2n3p15
Mullin, John - "LDOS and a Hayes SmartModem" (w/programs) v1n6p34
Pelzer, Ray - "Confessions of a Machine Language Addict" v1n6p38 v2n1p46
Ratkowski, Sue - Structured BASIC Translator review (Acorn) v1n4p21
Rathke, Glen - "Fortran, Cobol and LDOS JCL" v1n6p32
Reyes, Renato - DELETE/CMD v1n4p56
SuperScripsit patches v1n6p92
Rice, Lee - "APL*PLUS/80 - A System Overview" (STSC review) v2n1p09
"LISP Implementations for the Z-80" (Supersoft & Far West reviews) v1n6p26
Robinson, Earle - "Inside the Expansion Interface" v1n4p39
"The Library" v1n4p28
"Review: Microcomputer Math" (Barden, SAMS) v1n5p10
"..... er .....": assembly language programming tips v1n5p14
"..... er .....": efficient assembly language programming v1n6p64
"..... er .....": printers, word processors, MNet, 'C', LDOS 6.x v2n2p26
"..... er .....": 'C' book, SS drivers, Model 4, PROMPT/CMD v2n3p33
"..... er .....": UNIX, IBM PC, Telex v2n4p28
Roser, Ken - MX80/FLT v1n1p22
Ruf, Erik - "SOLEFIX - Fix that GAT Error" v2n2p14
Simon, Peter - "Running 8-Inch Drives on the Model III Under LDOS" v2n3p08
Sturiale, E. - "Profile One Plus" v2n5p26
Terwilliger, Earl - "At Large" (communications program & ALIVE) v1n6p50
"The 'C' Language" v2n1p15 v2n2p35 v2n3p39 v2n4p34 v2n5p31
"Using the EDAS IV 'Z' Command" v2n1p37
Thompson, Gordon - "The Communicating Micro" (& REFLEX/FLT) v1n6p12 v2n1p28
"The Electronic Inbasket" v2n4p15
Tobias, Rick - CASE/CMD, a Mod I case mode indicator v2n2p11
Tonini, Paul - CARD/BAS, file utility v2n2p18
Vinzant, David - "@PARAM, @DSPLY, @EXIT and INBUF$ for Everyone" v2n2p23
Yevich, Dick - "Profile on LDOS" v1n4p59

Developing an index turns out to be a very subjective procedure.

- apologies are hereby offered to anyone that feels slighted through omission or misstatement; it was unintentional -- Scott Loomer

LSI Quick Hint #2

With LDOS 5.1.4, using the library command "DIR partspec:n (A=N)" will provide the old-style "multiple-across" directory display without any patches to the system.

```
.CPROMPT/FIX
X'75F8'=CC 3A 7C CA 63 76 FE 6D CA 63 76


.RPROMPT/FIX
X'76BE'=Ø2 7D
X'7DØ2'=CD 5Ø 7C 21 2D 71 CD E5 7C 21 1C 7D 7E Ø2 23 CD 33
X'7D13'=ØØ 7E FE Ø3 28 61 C3 ØE 7D ØD ØD ØD ØØ 2Ø 2Ø 2Ø 2Ø
X'7D24'=2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 4D
X'7D35'=6F 75 6E 74 2Ø 43 52 45 41 54 49 4F 4E 2Ø 44 69 73
X'7D46'=6B 65 74 74 65 ØD 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
X'7D57'=2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 5Ø 72 65 73 73 2Ø 45 4E
X'7D68'=54 45 52 2Ø 54 6F 2Ø 43 6F 6E 74 69 6E 75 65 2Ø 2Ø
X'7D79'=Ø3 AF CD 49 ØØ FE ØD 2Ø F8 C9


.RSPATCH/JCL
PATCH EFCM/CMD (X'886A'=C3 84 86)
PATCH EFCM/CMD (X'883C'=CD 78 86)
PATCH EFCM/CMD (X'8684'=AF 32 14 42 C3 2D 4Ø)
PATCH EFCM/CMD (X'8678'=3E 2Ø EB 2B BE CA 7B 86 23 36 ØD C9)
PATCH RM/CMD (X'716C'=31)


.PROFIX/JCL
. Auto patching for Profile + HD model III
.to operate on the Model I with either floppy or HD.
.Be sure the patch disk is in drive Ø and the
.disk containing the profile modules is in drive 1
//PAUSE Press <ENTER> to Begin
PATCH CM/CMD USING CM/FIX
PATCH RM/CMD USING RM/FIX
PATCH EFC1/CMD USING EFC1/FIX
PATCH EFC2/CMD USING EFC2/FIX
PATCH EFC3/CMD USING EFC3/FIX
PATCH EFC4/CMD USING EFC4/FIX
PATCH EFC5/CMD USING EFC5/FIX
PATCH EFC6/CMD USING EFC6/FIX                  .CM/FIX
PATCH EFCM/CMD USING EFCM/FIX                  X'7BC2'=67 44
PATCH EFCE/CMD USING EFCE/FIX                  X'74B8'=18 43
PATCH EFCF/CMD USING EFCF/FIX                  X'7652'=18 43
PATCH EFC7/CMD USING EFC7/FIX                  X'765E'=18 43
PATCH EFC8/CMD USING EFC8/FIX                  X'74BB'=19 43
PATCH EFC9/CMD USING EFC9/FIX                  X'7661'=Ø5 44
PATCH EFCA/CMD USING EFCA/FIX                  X'7832'=Ø5 44
PATCH EFCB/CMD USING EFCB/FIX                  X'761A'=49 4Ø
PATCH EFCC/CMD USING EFCC/FIX                  X'761E'=49 4Ø
PATCH EFCD/CMD USING EFCD/FIX                  X'7617'=Ø1 3Ø
                                               X'782C'=Ø1 3Ø
                                               X'7ABE'=Ø1 3Ø
.PROMPT/JCL                                    X'7B3F'=Ø1 3Ø
.This patch allows FLOPPY usage                X'7151'="ONE"
PATCH CM/CMD USING CPROMPT/FIX                 X'7172'=31
PATCH RM/CMD USING RPROMPT/FIX                 X'74F6'=63
```

```
.EFC1/FIX
X'87FA'=67 44
X'846A'=Ø5 44
X'8464'=Ø1 3Ø
X'86F6'=Ø1 3Ø
X'8777'=Ø1 3Ø
X'8397'=C2
X'89BØ'=6Ø

.EFC2/FIX
X'7E12'=67 44
X'7A82'=Ø5 44
X'74BE'=22 4Ø
X'7ØC3'=5B
X'4Ø22'=8F
X'4Ø23'=ØØ
X'7E9Ø'=6Ø

.EFC3/FIX
X'81Ø9'=67 44
X'7D79'=Ø5 44
X'7CA6'=C2
X'8187'=6Ø

.EFC4/FIX
X'7FE9'=67 44
X'7C59'=Ø5 44
X'7C53'=Ø1 3Ø
X'7EE5'=Ø1 3Ø
X'7F66'=Ø1 3Ø
X'8Ø67'=6Ø

.EFC5/FIX
X'7D66'=67 44
X'79D6'=Ø5 44
X'79DØ'=Ø1 3Ø
X'7C62'=Ø1 3Ø
X'7CE3'=Ø1 3Ø
X'81A1'=6Ø

.EFC6/FIX
X'7AE8'=67 44
X'7758'=Ø5 44
X'7B66'=6Ø

.EFC7/FIX
X'6Ø76'=67 44
X'5CE6'=Ø5 44
X'57D2'=5D 44
```

```
X'58Ø1'=5D 44
X'5825'=5D 44
X'5849'=5D 44
X'588E'=5D 44
X'59ØD'=5D 44
X'5AØ3'=5D 44
X'6ØF4'=6Ø

.EFC8/FIX
X'56F2'=67 44
X'543C'=Ø1 3Ø
X'5668'=Ø1 3Ø
X'56E9'=Ø1 3Ø
X'7Ø7B'=Ø1 3Ø
X'7ØAD'=Ø1 3Ø
X'7BAF'=Ø1 3Ø
X'7BE7'=Ø1 3Ø
X'7BDE'=18 43
X'7BED'=18 43
X'8DE7'=18 43
X'8DFE'=18 43
X'8E51'=18 43
X'5442'=Ø5 44
X'7ØD4'=Ø5 44
X'7BFØ'=Ø5 44
X'8DD9'=49 4Ø
X'973Ø'=49 4Ø
X'6DF8'=4C 44
X'6EØ6'=4C 44
X'5764'=5A 44
X'564C'=5D 44
X'74C9'=6Ø

.EFC9/FIX
X'E441'=67 44
X'BFA3'=7Ø 44
X'E39C'=Ø5 44
X'7B7E'=18 43
X'C5F8'=5D 44
X'7B74'=ØØ ØØ ØØ
X'E6B1'=6Ø

.EFCA/FIX
X'7Ø33'=67 44
X'6758'=7Ø 44
X'6C9D'=Ø1 3Ø
X'6F2F'=Ø1 3Ø
X'6FBØ'=Ø1 3Ø
X'5B55'=18 43
X'5B73'=18 43
X'6CA3'=Ø5 44
X'6A7B'=6Ø
```

```
.EFCB/FIX
X'6681'=67 44
X'62F1'=Ø5 44
X'62EB'=Ø1 3Ø
X'657D'=Ø1 3Ø
X'65FE'=Ø1 3Ø
X'5E4B'=C1 44
X'5E46'=C4 44
X'56ED'=18 43
X'57ØB'=18 43
X'66FF'=6Ø

.EFCC/FIX
X'56E7'=67 44
X'9625'=67 44
X'97E9'=67 44
X'9FFD'=67 44
X'AØ1F'=67 44
X'565D'=Ø1 3Ø
X'56DE'=Ø1 3Ø
X'73B1'=Ø1 3Ø
X'73D5'=18 43
X'73DD'=18 43
X'9228'=18 43
X'9248'=18 43
X'9298'=18 43
X'73EØ'=Ø5 44
X'74Ø6'=Ø5 44
X'9224'=49 4Ø
X'587B'=6Ø

.EFCD/FIX
X'5C4Ø'=67 44
X'58AA'=Ø1 3Ø
X'5B3C'=Ø1 3Ø
X'5BBD'=Ø1 3Ø
X'8Ø5D'=18 43
X'8Ø74'=18 43
X'8Ø82'=18 43
X'58BØ'=Ø5 44
X'5CBE'=6Ø

.EFCE/FIX
X'7D8Ø'=67 44
X'79FØ'=Ø5 44
X'791D'=C2
X'79EA'=Ø1 3Ø
X'7C7C'=Ø1 3Ø
X'7CFD'=Ø1 3Ø
X'81B7'=6Ø
```

```
.EFCF/FIX
X'88AB'=67 44
X'7ADC'=18 43
X'7AF3'=18 43
X'7BØ1'=18 43
X'851B'=Ø5 44
X'8Ø1B'=49 4Ø
X'8515'=Ø1 3Ø
X'87A7'=Ø1 3Ø
X'8828'=Ø1 3Ø
X'82F3'=6Ø

.EFCM/FIX
X'8CFF'=67 44
X'7379'=96 43
X'76Ø1'=Ø5 44
X'8843'=Ø5 44
X'8986'=Ø5 44
X'76EC'=C2
X'7423'=18 43
X'75D2'=18 43
X'75FE'=18 43
X'8835'=18 43
X'884Ø'=18 43
X'882B'=Ø1 3Ø
X'898Ø'=Ø1 3Ø
X'8BFB'=Ø1 3Ø
X'8C7C'=Ø1 3Ø
X'8D7D'=6Ø

.RM/FIX
X'771A'=Ø5 44
X'777Ø'=Ø5 44
X'779C'=Ø5 44
X'7956'=Ø5 44
X'7598'=18 43
X'77ØF'=18 43
X'7717'=18 43
X'7765'=18 43
X'776D'=18 43
X'778E'=18 43
X'7796'=18 43
X'759B'=19 43
X'75EØ'=63
X'7C63'=Ø1 3Ø
X'7BE2'=Ø1 3Ø
X'795Ø'=Ø1 3Ø
X'76CE'=Ø1 3Ø
X'714B'="ONE"
X'7CE6'=67 44
```

SUPERSCRIPSIT PRINTER DRIVERS $49/$59 (Dep. on printer)

With an ALPS printer driver, it is possible to attach non-Radio-Shack-supported printers to the powerful SuperSCRIPSIT Word Processing System. **More than 80 different printers are now supported** with customized drivers, which automatically handle the interface to the printers. Each driver supports all the normal text processing (margins, tabs, centering, etc.), plus whatever features the printer can support. The printer driver is a software product, which is easy to install and use.

FOUR MODEL 4 UTILITIES (FOR MODEL III MODE) $19

With the following ALPS Utilities, you can easily access some of the Model 4 hardware features while in Model III mode. The following utilities are designed to run on a TRS-80 Model 4 disk system equipped with either 64K or 128K of RAM memory, and executing in Model III mode. These utilities can be used on any operating system.

CPUFAST can be used to speed up processing speed. It sets the CPU clock speed to 4 Mz. This will cause programs to execute up to 50 percent faster than normal. The CPUFAST command need only be typed once. The Model 4 will remain in high speed mode until is is rebooted, or until the CPUSLOW command is executed. The CPUSLOW program allows returning processing speed back to the normal 2.0 Mz. rate (Model III compatible rate) without re-booting the system.

RAMTEST will test Model 4 memory by writing and reading data from each RAM location. It will continue endlessly until the BREAK key is presssed or until it encounters a bad location. The following areas of RAM are verified:
Std. RAM, ROM Overlay, First & Second 32K expansion bank (if 64K exp installed).

CLEARMEM will set all Model 4 memory (as described in RAMTEST above) to the value specified in BYTE='xx'. It no BYTE= parameter is specified, all memory is initialized to zero. This routine is useful for system developers.

BEEP will cause a beeping sound to be generated by the Model 4 sound generator circuitry. The optional LOUD parameter results in a harsher, louder beep. The optional CYCLES= parameter specifies the number of times that the beep cycle is repeated. Useful applications of BEEP are in /JCL files to signal an operator for keyboard input or to audibly indicate when a job has completed. BEEP can also easily be used from within BASIC programs via the BEEPSUB callable subroutine. The file BEEPTEST/BAS is a sample program which contains an example of the required calling sequence.

BASIC PROGRAM SEARCH UTILITY $19

Search for any string within a BASIC program with an easy CMD command. Display all occurrences on screen. With this Search Utility, you can scan your program automatically, rather than visually for greater accuracy. LDOS only. Model I/III/4(III mode).

Great for all BASIC programmers!

For example: **CMD "BS /QC/"** will reveal whether or not you have used the name "QC" in your BASIC program. Great way to prevent duplicate variable names.

For example: **CMD "BS /AB/"** will find all occurrences of the variable AB if you want to change the name to another name. Any missing lines that you know should be listed would indicate a mistyped line.

MEMDISK (MODEL 4 IN MODEL III MODE) $39

Easy to use. (LDOS only). Use MEMDISK as your System Drive or load other files into memory for Instant Access. Use **80K** of 128K memory. The MDISK utility creates a RAM memory disk drive using 80K of expansion memory avail- able in a Model 4 equipped with the 128K upgrade option. It operates under LDOS Version 5.1 in Model III mode only. MDISK creates a single 80K logical drive which acts nearly identical to a floppy disk but at greatly improved access speed. This should support all applications which do not directly interface with the floppy disk controller hardware, or do not depend on the 180K physical capacity of a floppy disk.

CREATE option is used to define a single 80K MDISK logical drive. The logical drive will be automatically formatted and can be used immediately after creating.

FLOPPY option is used to create a floppy disk logical drive. An existing MDISK drive could be redefined to its original floppy configuration using this command.

LOAD=... option is used to quickly build an MDISK from the contents of a file created by a previous SAVE. Note that a **High Speed MDISK System Drive** can be created. MDISK :0 (LOAD="SYS0") will load an MDISK drive saved with Filespec name "SYS0/MDK" directly into drive 0.

NULL option removes a logical drive completely from this system. An MDISK drive, if NULLed, can be recovered using the WARM option.

SAVE=... option is used to quickly save, in the form a single file, an image of the current MDISK drive. Only allocated cylinders of the MDISK drive are written to disk which greatly reduces the size of the generated file for MDISKs which are not full.

WARM option allows recovery of an MDISK drive when standard RAM memory has been changed, but expansion memory remains unaltered. This command can be used after a system reboot to recover a pre-existing MDISK drive.

MEMORY DUMP UTILITY $24

HEX/ ASCII screen display of memory pages (Incl. exp. 64K, low & std. RAM, ROM) with one easy command. For Model 4 (Mod III mode), any oper. sys. **(80X24 screen).**

LOADMAP UTILITY $19

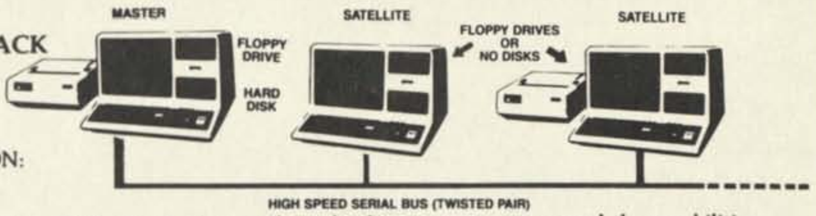
Obtain a graphic screen display of object program load address ranges. One command. Model III or Model 4 (Mod III mode), any operating system.

All of the above software products are immediately available with full documentation. You may order either by telephone or mail. Mastercard and Visa are accepted. (For UPS COD, add $3). Orders are sent via First Class Mail, unless otherwise requested.

A L P S, 23 Angus Road, Warren, New Jersey 07060
201-647-7230

Networking FOR RADIO SHACK
MODELS III & 4...NOW!!

TYPICAL CONFIGURATION:


MASTER
FLOPPY DRIVE
HARD DISK
SATELLITE
FLOPPY DRIVES OR NO DISKS
SATELLITE
HIGH SPEED SERIAL BUS (TWISTED PAIR)

After comprehensive field testing in medical offices, PSM, Inc. is releasing the technology necessary to expand the capabilities of your Model III/4 to MULTI-USER, MULTI-PROCESSING, SHARED DATA APPLICATIONS using Winchester 5.25" hard disks. Each workstation is a complete, independent computer which is tied to the hard disk through a high-speed serial bus. Workstations DO NOT NEED ANY DISK DRIVES; they will boot up properly through the serial bus from the hard disk. It will be running under LDOS as if the hard disk was its drive: 0 !

FEATURES

- Protects existing software investment
- Data file sharing
- Compatible with LDOS 5.1.4
- Shared hard disk [5 to 20 megabytes]
- File/record lockout capability
- Used in active medical offices since 2/83.

PSM-W5:	Model 4 w/64k RAM, 40trk DS Floppy, built-in 5 Mbyte hard disk, master network controller, LDOS 5.1.4, Technical Manual	$4,975.00
PSM T/0:	Model 4 w/64k RAM, boot PROM, satellite network controller, Tech Manual	$1,875.00
PSM-U5:	PSM upgrade of your Model III/4 w/64k RAM to PSM-W5: incld's 5Mb hard disk, hard disk controller, master network controller, LDOS 5.1.4, power supplies, Tech Manual	$2,625.00
PSM-U0:	PSM upgrade of your Model III/4 w/64k RAM to PSM T/0: incld's boot PROM, satellite network controller, Tech Manual	$665.00
PSM-NM:	Technical Manual: Incld's complete description of PSM networking, and application implementation Tech Manual purchase may be credited toward later hardware purchase	$20.00

PSM programming and systems management, inc.

All shipping and handling charges are the customers' responsibility. Technical consultation services available. Modification to Model III/4 to implement networking will void the Tandy warranty.

3866 INDIAN RIPPLE RD., DAYTON, OH 45440 [513] 426-8644

LOGICAL SYSTEMS INC.

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

BULK RATE
U.S. POSTAGE PAID
Permit No. 36
Mequon, WI 53092

In This issue:

- Revealed –
  The Future of the LSI JOURNAL
- Change PR/FLT "on the fly"
- "C" – Part 5
- EZ-EDIT –
  LDOS Command Line Editor
- The LDOS QUARTERLY/
  LSI JOURNAL Index

POSTMASTER: DATED MATERIAL DO NOT DELAY

April 1984

Volume 2 Number 6

ISSN 0737-9161

$4.

LSI JOURNAL

Formerly the LDOS QUARTERLY

EMERGENCY COVER

(Originally Scheduled to Appear in the April Issue of Basic Computing.)

We Now Have A Toll Free Orderline! (800) 248-3535

LOGICAL SYSTEMS INC.

NEW!

OUTLINE CAMEO Pretorian
Rotunda LOMBARDIAN celtic
Nostalgia INCISED TRAJAN Playbill
Olde English CHAINED
LIQUID CRYSTAL CLIMBING
MOON LITE Pump ANTIQUE
BANNER Elegant REV BAN
SHADOW ROMAN Mini Cubes
Chancery Medium JULY 4 Small Boldface
Small Bold Italics

These are printed by DOTWRITER just as you see them

See What You Can Do With DOTWRITER 4.0!

Now available for the Model 4, too!

This new, fast version of DOTWRITER is just what you need to turn your dot-matrix printer into a versatile typesetting machine. Written entirely in "machine language," our latest release offers even more features to help you produce beautiful, eye-catching results.

What Is DOTWRITER?

DOTWRITER is a full-function text printing program. It lets you print distinctive letterheads, brochures, flyers, catalogs, invitations, or even a book. It does superb right-justified proportional printing, including "kerning" (tucking small letters under big ones to achieve a really professional result). DOTWRITER handles type sizes from ⅛ to 1 inch, can magnify text until each letter fills the page, intermixes type styles, and even does reversals (white on black).

After writing your text with any popular TRS-80 Word Processor, such as NEWSCRIPT, just insert the necessary layout commands, save it to disk, and DOTWRITER will do the rest.

What's Included?

DOTWRITER includes the printing program and fourteen complete sets of type faces (60 to 90 characters in each set). The 60-page manual has a step-by-step tutorial, a Table of Contents, and an Index. And, of course, you will have on-going support directly from PROSOFT.

DOTWRITER Can Grow With You

15 "Font disks," each with 3–12 complete typeface sets (60 to 95 characters in each set), are available separately, and three more will be released soon. These disks cost less than $25.00 each, and may be purchased at any time. We'll be happy to send you a free sample of all our typefaces on request.

Design Your Own Typefaces

If you just want to use some of the many typefaces we carry, then DOTWRITER is all you need. If you want to be able to modify our typefaces or even design new ones, then you will also want to order the **"Letterset Design System."** We offer it at a reduced price when you order it along with DOTWRITER.

Supports Models I, III, and 4

One version of DOTWRITER 4.0 is for the TRS-80 Models I and III (also LNW and MAX-80), and another is for the TRS-80 Model 4 (**yes, in native Model 4 mode**). At least 48K and two disk drives are needed.

The Letterset Design System works only on Models I and III, but it can run on a Model 4 in Model III mode.

Versions are available for the Epson MX-80 with Graftrax, MX-100 with Graftrax-Plus, RX-80, and FX-80; the C. ITOH 8510/1550; the Microline 84/92/93; and Radio Shack's DMP series 200/400/500, and 2100. Please specify your printer and computer when ordering!

How to Order

Limited ad space allows us to show you only a few of the 120 DOTWRITER fonts, but free print samples are available on request. If you want the best in graphics printing, we suggest you order DOTWRITER today, toll-free.

DOTWRITER 4.0 (Models I, III)	**$79.95**
DOTWRITER 4.0 (Model 4)	**99.95**
Letterset Design System (LDS)	**39.95**
Special: DOTWRITER and LDS for Models I, IIII	**99.95**
for Model 4 (Model 3 LDS)	**119.95**
Additional Fonts (4–12 per disk)	**17.95 and 24.95**

ORDER NOW, TOLL-FREE
(800) 824-7888, oper. 577

PROSOFT®

Dept. C, Box 560, No. Hollywood, CA 91603
(818) 764-3131 Information and Same-Day Processing
TERMS: VISA, MC, checks, COD. Please add $3.00 shipping in U.S. or Canada, $15.00 overseas, sales tax in Ca. Most orders filled within one day.

The LSI Journal April 1984 Volume 2, Number 6

CONTENTS

INTRODUCTION FROM LSI:

LATE BREAKING NEWS .... Page 2

NEW PRODUCTS ANNOUNCEMENTS .... Page 3

ITEMS OF GENERAL INTEREST .... Page 4
patches and update information

Profile III Plus and LDOS (revisited) .... Page 7

LSI Journal/BASIC Computing Insert:

View from the Bottom Floor .... Page 9

Locating high memory routines under LDOS/TRSDOS 6.X .... Page 11

REGULAR USER COLUMNS:

*** PARITY = ODD *** - Tim Daneliuk .... Page 12

'C' What's Happening (Part 6) - Earl Terwilliger .... Page 14

FROM THE LDOS SUPPORT STAFF:

LDOS: HOW IT WORKS - an introduction to COMM and LCOMM .... Page 16

Les Information - Fast Machine Language Disk I/O .... Page 17

LSI Journal and LDOS Quarterly Index - by Scott Loomer .... Page 22
(Please note that due to the lead times involved,
the index in Vol2, #5 is actually more complete.)

LATE BREAKING INFORMATION

By Bill Schroeder

Yes ... this is the issue of the LSI Journal that was to have appeared in the pages of the magazine BASIC Computing. Alas, it seems that this will never come to pass.

About the first week of March, I was informed that there would never be an April issue of BASIC Computing. Citing ongoing and growing losses, Irv Schmidt informed me that BASIC Computing was going to cease publication. At this writing, they have stated that they are making arrangements to have another publication fufill their subscription obligations as they are financially unable to do so. If you have any questions about your subscription, contact BASIC Computing at 3838 S. Warner St., Tacoma, Washington, 98409 - (2Ø6) 475-2219.

On March 9th, 1984, the material that was to have been published in the April issue of BASIC Computing was returned to us. We did a mad scramble to put the issue together ourselves, and this is the result. Most of the material is presented as it would have appeared in BASIC Computing. We are sending this issue to all registered LDOS owners to explain the situation. We sincerly apologize for this development. Please understand that we had NO KNOWLEDGE WHATSOEVER of the upcoming demise of BASIC Computing.

LSI ANNOUNCES TOLL FREE ORDER SERVICE

(8ØØ) 248-3535

For your convenience, LSI has installed Toll-Free phone service to our order desk. Now you can place orders with LSI "on our nickel". The number will be (8ØØ) 248-3535 and will go into service about March 3Øth, 1984. We hope you will enjoy the convienience of this new service. Please remember that this number is for orders only. Customer Service calls will continue to be taken at the main switchboard number of (414) 355-5454.

LSI "Such a Deal" Update

Here is the current status of the Deals offered in the last Journal:

DEAL #1:

For those of you that did not receive the previous issue of the Journal, LSI was closing out all products not manufacturered by LSI. All of the non-LSI software is gone, except for MicroPro's MailMerge, Captain 747 from Molimerx and the IJG book "How to do it on the TRS-8Ø", by William Barden.

For those who own the LDOS version of WordStar, now is the time to pick up a copy of MailMerge. For details on the capabilities of MailMerge, see your WordStar manual (the MailMerge documentation is included in it). There are less than twenty copies left, so place your order right away. Under Deal #1, MailMerge is $149.4Ø, plus $3 shipping and handling. We also have a limited quantity of extra WordStar/MailMerge manuals available for $2Ø each plus $4 shipping and handling (binder not included).

"How to do it on the TRS-8Ø" by William Barden is an excellent book for all TRS-8Ø owners and LDOS users. This book talks about both hardware and software (including LDOS) in a very readable manner. Mr. Barden has always received great praise for his books on microcomputers and assembly language programming, and has done it again with this book. This is a must for all TRS-8Ø owners. Whether you buy it from us at a discount, or pay full retail, don't boot up without it.

The price, you ask? Well, Deal #1 was only for software products, but "Such a deal we have for you". The normal retail price is $29.95, but order it from LSI for just $25 by itself, or for $2Ø with any order over $5Ø in other merchandise. Enclose your check with your order, and LSI will pick up the added shipping charges. Otherwise, add $2 for shipping in the United States. Now how many do you want?

DEAL #2:

is over! Please note that the new LSI catalog will be available Real Soon Now. We will probably start mailing it out by the beginning of June. Unfortunately, along with the new catalog will come some price increases (except for LS-LED, which will go down to $49). This makes *now* the time to place that order you've been holding off on. Orders postmarked before June 1st, 1984, or placed by phone before that date will be honored at the old prices.

DEAL #3:

continues! We said "until we run out" and we meant it (and still do!). With any order totaling $5Ø or more, LSI will include a set of Volume 2 Journals/Quarterlies, Numbers 1 through 4 at no extra charge. There is still a reasonable supply left of all four issues. Even if you already have all these issues, please take advantage of this offer and give them to a less fortunate friend.

DEAL #4:

Speaking of that less fortunate friend, how many of <u>your</u> friends are not using LDOS on their TRS-8Ø? Tell them about Deal #4. They can take advantage of it and trade in that "other" operating system. Until June 3Øth, 1984, LSI will take either NEWDOS8Ø/v1/v2 or DOSPLUS 3.4/3.5/4 in trade towards the purchase price of the LDOS 5.1 operating system. Send your old MASTER diskette and MANUAL to LSI, and pay only $64.5Ø plus $5 shipping and handling for a brand-new copy of the LDOS 5.1.4 operating system (specify model). The only operating system/version other than those mentioned above that qualifies for this Deal is 5.Ø Model 1 LDOS. It may be traded in on LDOS 5.1.4 for the Model 1.

DEAL #5: Sorry, all the extra Radio Shack Hard Disk Systems are gone.

NEW PRODUCT ANNOUNCEMENTS

LSI Publishes DOS Source Code

Logical Systems, Inc. announces publication of the complete, commented, assembler source code for the LS-DOS/TRSDOS 6.2 operating system. This may be the first time that the complete source code for such a sophisticated operating system has been made available to the public at a reasonable price.

The publication is entitled LS-DOS/TRSDOS 6.2 "THE SOURCE", and is published in three 8 and 1/2 by 11, soft bound volumes:

Volume #1	THE SYSTEM	L-6Ø-Ø11
Volume #2	THE LIBRARIES	L-6Ø-Ø12
Volume #3	THE UTILITIES	L-6Ø-Ø13

Each volume is available for $99, or the complete set of three as L-6Ø-Ø2Ø for $249. Delivery of all volumes will begin in June of 1984.

New TRSDOS 6 BASIC Utilities

Logical Systems, the authors of the TRSDOS 6.x operating systems licensed to Tandy/Radio Shack for the Model, 4 have announced a new product package for use by BASIC programmers, called "BSORT/MOD324".

This package includes a very powerful SORT utility, called "BSORT", which is executed from BASIC to sort arrays. BSORT is written entirely in assembler using advanced sorting technics for super fast operation. Tag arrays, index arrays, string and numeric arrays, mid-string sorts, accending and decending sorts and much, much more are supported. BSORT requires TRSDOS 6.1.2 or 6.2.Ø for proper operation. The TRSDOS 6.1.2 release includes a new version of BASIC, and should be available from Radio Shack by the time you read this. Radio Shack's catalog number for TRSDOS 6.1.2 is 7ØØ-2246.

A BASIC program conversion aid called MOD324 completes the package. Again, MOD324 is written totally in assembler for speed. This product will convert a Model 3 BASIC program to a Model 4-type format. MOD324 is even capable of adjusting print locations on the screen ("PRINT @" and "PRINT TAB") as well as pointing out the lines that need further attention after the convert. BSORT/MOD324 is available as L-32-210 for $49.

LSI Announces FED86

FED86 is the LSI all-purpose File and disk EDitor for the IBM-PC and PC-compatible machines (*not* the Tandy 2000) running under MS-DOS or PC-DOS (version 2.X required).

Any byte in any given file or disk can be displayed and/or altered. The display information in file mode includes: 256 byte record display, file name and drive number, record number and relative byte number within the current sector. In addition, the value of the byte under the cursor is displayed in hex, decimal and binary. In disk mode, the disk relative sector is displayed instead of the file name.

FED86 also includes commands for: searching for hex or ASCII strings, a case-independent text search, sending a record or group of records to the printer and modification of the displayed data (either in hex or ASCII).

All in all, FED86 is a tool that no MS-DOS user should be without. Even if these operations sound difficult, FED86 makes them a snap!

LSI Announces the LS-Utiltiy Disk

The LS-Utility Disk can be considered as "the best of 5.1 for 6.X". It includes the majority of the most popular filters and utilities from our LDOS 5.1 Filter Disks #1 and #2, and our Utility Disk #1, reconfigured for use under LDOS/TRSDOS 6. Projected availability is June 1984, and the programs included with the LS-Utility Disk are expected to be:

PRCODES/FLT	On printers that will backspace, provides for easy control of boldface and underlining, and also provides slashed zeros.
TRAP/FLT	Traps and throws away any user-defined character.
MAXLATE/FLT	A complete translation filter system for input or output devices. Includes tables for EBCDIC and DVORAK translation. Custom tables are easily built for any application, and a string of characters may be specified to replace a "match".
CALC/FLT	*KI filter - HEX/DEC/Binary conversion and HEX add/subtract.
KSMPLUS/FLT	Same as KSM, but allows key re-definition "on-the-fly" from the keyboard. Also includes "command repeat" and more.
RDTEST/CMD	Non-destructive forced read of an entire diskette.
READ40/CMD	Allows read-only access to forty cylinder diskettes in a eighty cylinder drive by double-stepping (96 TPI only).
TYPEIN/CMD	Combines the functions of JCL and KSM. Allows most programs that won't normally accept JCL to be controlled automatically, if they honor the device I/O structure of LDOS/TRSDOS 6.

Items of General Interest

The current release of LDOS for the TRS-80 Models 1, 3 and the LOBO MAX-80 is 5.1.4, with file dates of 10/01/83.

The latest (known) release version of TRSDOS 6 is 06.01.02. In this release, there are no operating system changes from 06.01.01, but there is a new version of Microsoft BASIC supplied by Radio Shack. This version can be obtained from any Radio Shack Computer Center upon proof-of-purchase (of a Model 4 or 4P). Your old 06.00.00 to 06.01.01 DOS disk should serve. The Radio Shack stock number for this update is 700-2246. Given the heavy demand for this item, your Computer Center may not have it in stock, but can order and/or reserve one for you.

Radio Shack Hard Disk Owners who have not yet received the hard disk drivers to allow the use of the hard disk under TRSDOS 6 should get a copy of 7ØØ-2247. This is TRSDOS Ø6.Ø1.Ø1 along with the hard disk drivers (and formatter). You will also need to get the above mentioned 7ØØ-2246 so that you have the new version of Microsoft BASIC. Again, the Computer Center may not have a 7ØØ-2247 in stock for you, but can order it.

The following patches have been requested to alter the time/date prompts and commands to accept a period instead of a colon or slash as a delimiter. This patch accepts a period only (not almost any character as in 5.1.4). The first patch is for the boot-up prompts, and should be applied to SYSØ. The second is for the DATE and TIME commands, and should be applied to SYS7.

```
.T611SYSØ/FIX
. Patch to TRSDOS Ø6.Ø1.Ø1 as distributed by Radio Shack
. Use the BUILD command or an editor to type in this patch
. as shown, and then PATCH SYSØ/SYS.LSIDOS T611SYSØ/FIX
.
. This patch forces boot-up date and time prompts to accept
. the period as a delimiter instead of the slash or colon
.
. This part changes the date prompt
DØE,B3=2E
FØE,B3=2F
.
. This part changes the time prompt
DØF,9E=2E
FØF,9E=3A
. End of patch

. T611SYS7/FIX
. Patch to TRSDOS Ø6.Ø1.Ø1 as distributed by Radio Shack
. Use the BUILD command or an editor to type in this patch
. as shown, and then PATCH SYS7/SYS.LSIDOS T611SYS7/FIX
.
. This patch forces the DATE and TIME commands to accept
. the period as a delimiter instead of the slash or colon
.
. This part changes the date command
DØ1,EB=2E
FØ1,EB=2F
.
. This part changes the time command
DØ3,Ø5=2E
FØ3,Ø5=3A
. End of patch
```

The Model 1 to Model 3 upgrade/replacement board mentioned previously by Tim Daneliuk in his *** Parity = Odd *** column is available from a company called Northern Technology. They are located in Elk Grove Village, Illinois, and their phone number is (312) 86Ø-1772. By now you may have already seen their ads in the TRS-8Ø magazines.

The following corrections to the article "Profile One Plus" in the January '84 LSI Journal were prepared by Joseph J. Kyle-DiPietropaolo:

The patch files for RM, CM, EFC8, EFCC and EFCM needed correction. Using the new patch files on the next page with the rest of the previously published patch files will result in a properly operating version of Profile One Plus. Please note that these patches are for Ø1.ØØ.ØØ or Ø1.ØØ.Ø1 only. With version Ø1.Ø1.ØØ, all patches except EFCM/FIX seem correct, but have not been tested extensively. With all versions, PR/FLT should be installed with the SLINE=1 parameter to ensure proper report pagination:

```
FILTER *PR PR/FLT (SLINE=1)<enter>
```

```
.CM/FIX *
X'7BC2'=67 44
X'74B8'=18 43
X'7652'=18 43
X'765E'=18 43
X'74BB'=19 43
X'7661'=Ø5 44
X'7832'=Ø5 44
X'761A'=49 4Ø
X'761E'=49 4Ø
X'7617'=Ø1 3Ø
X'782C'=Ø1 3Ø
X'7ABE'=Ø1 3Ø
X'7B3F'=Ø1 3Ø
X'7151'="ONE"
X'73C5'=DE 41
X'74F6'=63
-------------
.RM/FIX *
X'771A'=Ø5 44
X'777Ø'=Ø5 44
X'779C'=Ø5 44
X'7956'=Ø5 44
X'7598'=18 43
X'77ØF'=18 43
X'7717'=18 43
X'7765'=18 43
X'776D'=18 43
X'778E'=18 43
X'7796'=18 43
X'759B'=19 43
X'75EØ'=63
X'7C63'=Ø1 3Ø
X'7BE2'=Ø1 3Ø
X'795Ø'=Ø1 3Ø
X'76CE'=Ø1 3Ø
X'714B'="ONE"
X'7CE6'=67 44
X'7591'=DE 41
X'7799'=DE 41
-------------
.EFC8/FIX *
X'56F2'=67 44
X'543C'=Ø1 3Ø
X'5668'=Ø1 3Ø
X'56E9'=Ø1 3Ø
X'7Ø7B'=Ø1 3Ø
X'7ØAD'=Ø1 3Ø
X'7BAF'=Ø1 3Ø
X'7BE7'=Ø1 3Ø
X'7BDE'=18 43
X'7BED'=18 43
X'8DE7'=18 43
X'8DFE'=18 43
X'8E51'=18 43
X'5442'=Ø5 44
X'7ØD4'=Ø5 44
X'7BFØ'=Ø5 44
X'8DD9'=49 4Ø
X'973Ø'=49 4Ø
X'6DF8'=C4 44
X'6EØ6'=C4 44
X'5764'=5A 44
X'564C'=5D 44
X'8DD2'=DE 41
X'74C9'=6Ø
-------------
.EFCC/FIX *
X'56E7'=67 44
X'9625'=67 44
X'97E9'=67 44
X'9FFD'=67 44
X'AØ1F'=67 44
X'565D'=Ø1 3Ø
X'56DE'=Ø1 3Ø
X'73B1'=Ø1 3Ø
X'73D5'=18 43
X'73DD'=18 43
X'9228'=18 43
X'9248'=18 43
X'9298'=18 43
X'73EØ'=Ø5 44
X'74Ø6'=Ø5 44
X'9225'=49 4Ø
X'5641'=5D 44
X'5759'=5A 44
X'71ØØ'=C4 44
X'71ØC'=C4 44
X'587B'=6Ø
-------------
.EFCM/FIX *
X'8CFF'=67 44
X'7379'=96 43
X'76Ø1'=Ø5 44
X'8843'=Ø5 44
X'8986'=Ø5 44
X'76EC'=C2
X'7423'=18 43
X'75D2'=18 43
X'75FE'=18 43
X'8835'=18 43
X'884Ø'=18 43
X'882B'=Ø1 3Ø
X'898Ø'=Ø1 3Ø
X'8BFB'=Ø1 3Ø
X'8C7C'=Ø1 3Ø
X'8772'=DE 41
X'8D7D'=6Ø
-------------
```

BUSINESS CHECK LEDGER +
ADVANCED BOOKKEEPING SYSTEM
$99.50

Checks ledger, expense accounts ledger, deposit accounts ledger, deposit ledger, accounts payable ledger, payee ledger, bank file, add — edit — display — print for each file.

Handles three banks with three series of checks numbers per bank. Can split checks into nine different expense accounts. List checks by check number, payee, or expense. Reconciles bank balance by clearing checks and deposits. Requires two disk drives, printer and 48K RAM. Integrates with RS disk payroll 26-1556. Model I, III, 4 or MAX80. Manual contains step-by-step procedures for all features and entries.

Data Stored in Business Check Ledger +

BANK FILE <Max 3 banks>
(1) Name of bank (2) address of bank (3) city, state, zip of bank (4) balance (5) first check number entered in computer (6) last check number entered in computer (7) last check number cleared

CHECK FILE <Max 9 — three banks and three series each bank>
(1) Check number (2) check ID <1=current 2=cleared 3=split expense 4=void> (3) amount of check (4) payee code <payee file number> (5) expense code <expense file number> (6) date of check

DEPOSIT ACCOUNT FILE <Max 100>
(1) Deposit account number (2) deposit name (3) quarter to date <QTD> (4) year to date <YTD>

EXPENSE ACCOUNT FILE <Max 100>
(1) Expense account number <general ledger account number> (2) expense name (3) quarter to date <QTD> (4) year to date <YTD>

PAYEE FILE <Max 998>
(1) Payee name (2) payee address (3) payee city, state, zip (4) expense account number normally associated with payee (5) quarter to date <QTD> (6) year to date <YTD>

DEPOSIT FILE <No max>
(1) Deposit ID <1=uncleared 2=cleared> (2) deposit amount (3) date of deposit (4) account deposited to

ACCOUNT PAYABLE FILE
(1) Date due (2) amount due (3) payee code (4) expense code

FOR DETAILS WRITE

JAMES RUSSELL

110 Beechmont Dr., Carmel, IN 46032, (317) 846-8553

The following optional patch for LDOS 5.1.3 or 5.1.4 on the Model 1 will allow the system time and date routines to utilize a hardware clock such as the Alpha Products Newclock8Ø or Timedate8Ø, and most other clock devices that use the MSM5832 clock chip. A similar patch is not possible on the Model 3 as the real-time-clock service routines are in ROM. A new service routine could be written as a RTC Task, however.

This patch assumes that the clock is addressed as ports BØ through BF. If your particular clock module is addressed differently, the underlined bytes may be changed. For instance, if your clock uses ports CØ to CF, change the underlined B5 to C5, and alter the rest of the underlined bytes in a similar manner.

```
. Patch to SYSØ/SYS.SYSTEM to allow use of a clock device
. Fix the timer interrupt routine
DØ4,Ø8=D2 45 ED 78 ØD CD A6 47 ED 78 ØD E6 ØF 85 12 1B
DØ4,18=C9 11 43 4Ø Ø1 B5 Ø3 CD C3 45 1Ø FB
. Fix the date initialization on BOOT
DØD,58=21 46 4Ø Ø1 BA Ø1 CD C6 4E Ø6 Ø3 CD C6 4E Ø1 BC
DØD,68=ØF CD C6 4E EB D6 5Ø 47 DB BB E6 Ø3 11 45 4Ø 21 48 4Ø 2Ø
DØD,7B=26 CB FE 18 22 ED 78 ØD AØ Ø7 57 Ø7 Ø7 82 57 ED
DØD,8B=78 ØD E6 ØF 82 77 2B C9
. End of patch
```

Using Profile 3 Plus under LDOS (revisited)

by Joseph J. Kyle-DiPietropaolo

Profile 3 Plus and LDOS are a very powerful combination. There are, however, several things that can cause trouble if you are not careful. Let's take them one at a time.

First, what version should you buy? Well, the only version that will run under LDOS is the "Hard Disk" version. Don't let that phrase scare you, as it works quite well on floppy disk also. At least two double-density drives are required, and if you are going to do any fancy sorting, three would be better. Double-sided and/or eighty cylinder drives can be used if you have them.

The "Hard Disk" version is $1ØØ more than the "floppy" version, but this is more than offset by the fact that PROSORT is included. PROSORT is Small Computer Company's disk virtual sort and enhanced selection module, which normally sells for $15Ø by itself. If you already own the "floppy" version, Radio Shack will sell you an upgrade to the "Hard Disk" version for $1ØØ and proof of purchase for the "floppy" version. Their catalog number for the upgrade package is 7ØØ-62Ø3.

In terms of actually running Profile 3 Plus, there are three things to keep in mind:

1) If you have the *KI driver installed, all references to the <clear> key in the Profile documentation should be changed to <shift><clear>. If you are using math fields, do not use type-ahead, as Profile does not seem to calculate properly if <shift><clear> is struck before the calculation is complete.

2) Profile checks the printer status by looking directly at the hardware. This means that if you are using the spooler, the printing rate will not really improve much. This can be corrected by searching through all the EFC programs (and RM/CM) for the byte sequence 3A E8 37 and replace it with 3E 3Ø ØØ. This is a good excuse to learn how to use FED (included with the LDOS 5.1.4 update). Making these changes will disable the check of printer status. Since the system printer driver (which contains its own "Printer *BUSY* test") is used to actually output the data, there will be no conflicts. Also, you will be able to route printed reports to a disk file without having a printer ready and on-line.

3) Lastly, contrary to what the manual says, Profile will *not* work with "DO files" under LDOS. This is because of the way keyboard input is requested by Profile. This difficulty is neatly fixed by using the TYPEIN utility from the LSI Utility Disk #1 (Order L-32-Ø7Ø, $39 direct from LSI).

When creating a file of commands to be used with TYPEIN and a Profile user menu, put the entire command sequence in the data file. This includes the EFC line. Build the file by putting in the exact keystrokes you would type if performing the procedure by hand. Then, invoke the procedure with TYPEIN. Do not attempt to pass the name of a DO file to the EFC module by placing it in parenthesis as stated in the Profile manual, as it will not work.

For example, let's say that we want to be able to expand our existing data file by 1ØØ records by pushing one key. First, create a user menu that has an entry called "Expand the MAIL file by 1ØØ Records". Set up the user menu so that this entry is executed by pushing "E". When "E" is pushed, it should execute the command: TYPEIN EXPAND/KYS. The TYPEIN data file you would have to build (with the name EXPAND/KYS) is given below. Note that <enter> means the depression of the key marked "ENTER".

Contents of the file EXPAND/KYS:

```
EFC7<enter>
MAIL<enter>
1ØØ<enter>
```

LSI's LED, the LDOS text EDitor (L-3Ø-Ø2Ø, $29) is an excellent tool for generating and maintaining these keystroke files for TYPEIN. LED also has a special HEX mode that will allow you to insert the <shift><clear> code (X'1F') directly into a TYPEIN keystroke file. This character is needed to terminate "extended mode" functions. Without LED, it can be done but it is not easy.

If you must use BUILD, and you need to get a <shift><clear> keystroke into the TYPEIN file, do the following: Make sure that KI/DVR is installed when doing the BUILD. When you get to the point at which you need to insert the <shift><clear> keystroke, actually put a <clear><shift><enter> character there. That character is produced by pressing the <clear> key, and while still holding is down, press the <shift> key, and while still holding both of them, press the <enter> key and then release them all. A Plus/Minus symbol (±) or square block on the Model 1 and MAX-8Ø will appear on the screen. Now continue with the remainder of the key data. Whenever you need to execute this kind of a TYPEIN procedure, use the following format: TYPEIN SELECT/KYS (X1=X'7F1F'). Each <clear><shift><enter> will now be translated to the required <shift><clear> when executing. This TYPEIN command can, of course, be placed in a user menu.

All prices mentioned above that pertain to LSI products are valid only through May 31st, 1984. Radio Shack pricing policies and product availability is subject to change. Please contact your local Radio Shack Computer Center for more information.

LSI Quick Hint #3

Don't forget about the LDOS SIG (Special Interest Group) on CompuServe. If you are already a CompuServe member, simply GO PCS-49. If are not a member, any Radio Shack Computer Center can set you up with a membership kit (note: RS232 and modem required). There are many useful public domain programs and utilities in the LDOS SIG databases, and questions about LDOS and TRSDOS 6 will be answered promptly, often both by the LDOS Support staff and experienced users. This is an excellent way to keep abreast of the latest in the LDOS world. For most people, CompuServe is only a local call away, and that local call plus CompuServe's modest $6 per hour fee is a lot less than a long distance call during the day to LSI.

LSI JOURNAL

View from the bottom floor

Bill Schroeder, Logical Systems, Inc.

Hello to all *LSI Journal* subscribers and readers of *Basic Computing*. From this point forward, the contents of the *LSI Journal* will be presented in *Basic Computing*. We feel that this will be of great benefit to both our tens of thousands of users, and to the existing fans of *Basic Computing* magazine. If you are a subscriber to the *LSI Journal*, your subscription will be fulfilled by *Basic Computing*, which will contain this *LSI Journal* section. Our subscribers gain the additional information, ads, and interesting articles from the rest of the magazine, while *Basic Computing* subscribers gain technical information and articles of special interest concerning LDOS 5.1.x, TRSDOS 6.x, LSI products, and last (and least), my ramblings.

Just so rumors don't start to fly . . . LSI was ***not*** bought out by *Basic Computing*, nor were they bought by LSI. There is NO connection between our companies. The name *LSI Journal* still belongs to LSI, but *Basic Computing* has been granted the right to use the name as needed for promotional purposes both in and outside of this publication. LSI provides all the editorial material in our corner of *Basic Computing* magazine. No money changed hands between the folks at *Basic Computing* and us here at LSI.

This arrangement is purely one of convenience. LSI is a software company, *Basic Computing* is a magazine. Both companies are very good at what they do best, and less efficient at "the other guy's specialty". So, *Basic Computing* will be our publisher.

Mike Schmidt (the owner of *Basic Computing* magazine) and I have been discussing the possibility of this arrangement for well over a year now. Being from a conservative German background, I tend to move very slowly on this type of major decision. To slow things down even more, Mike comes from the same conservative German background. Between the two of us, we were almost moving backward. The final result is, from now on, the *LSI Journal* will be incorporated in (about) every third issue of *Basic Computing* (read that as at least that often).

Should you have occasion to write or call regarding information presented in this area of the magazine, please contact LSI directly at: 8970 N. 55th St. PO Box 23956, Milwaukee, WI 53223 (414) 355-5454. Of course you can always contact *Basic Computing* also, but I would like to make it very clear that LSI provides all of the material contained in this section of *Basic Computing*.

Now down to what's happening at LSI. There have been many changes at LSI as we prepare to enter the MS-DOS world (IBM & compatibles). We will ***not*** be providing an operating system for this market. . . at this time. Our first product will be a truly "RELATIONAL" data base manager. This product is very dynamic in concept for a micro. It will be easier to use than any existing product of similar capabilities, faster, more versatile and will be very "friendly" to the average user. The name and price for this product remain undecided at the time of this writing, but should be known by the time you read this. Please call for additional information if you are interested. Note: This product will be for use with MS-DOS 2.x only.

Question: What is AT&T up to !?

Answer: Who knows?

One thing is certain: with IBM having sold about 850,000 PCs in 1983, and projecting about 2,500,000 (that's right-- 2.5 million!) to be sold in 1984, it will take a very heavy hitter to slow them down. If there is a company that can actually go head to head against IBM in the small business and professional market, it is most certainly AT$T (whoops-that $ must have been a slip). AT&T is one of the largest and richest companies in the world (much larger than even IBM).

AT&T could have kept their anti-trust case of the Seventies in court for another 20 or 30 years if they wanted to. But lo and behold, all of a sudden they offer a settlement to the U.S. government. What is most amazing is the fact that the air-heads in Washington, D.C. accepted it. AT&T drew their own draft for a break-up and re-organization of their own operations. They would agree to this "massive" break-up only if they would be allowed to enter "new markets", that they had (years earlier) agreed to stay out of. One of those markets is, of course, computers and other end-user DP equipment and services.

For many years, the fellows up at Bell Labs have been far ahead of private industry in computer technology. They have perfected practical bubble memory, 32-bit microprocessors and some of the best concepts (and implementations) in the software industry. Now, AT&T owns Bell Labs, and gets to take the cost of operating it as a TAX DEDUCTION (by a special Act of Congress). Boy, I wish LSI could write off our costs of software development that way.

AT&T has more ready cash, a larger manufacturing capability, a larger support system, and a larger marketing system than any company in the microcomputer field today. For that matter, make that ANY company at all, in any field.

And now AT&T cometh

Let us imagine an AT&T manufactured, marketed, and supported microcomputer system. The word "system" is very important here, and is going to encompass a lot more than what IBM, Apple or Tandy calls a "system" today. Think more of something like the United States telephone "SYSTEM". That, of course, was the last AT&T-controlled accomplishment. The U.S. phone system (and other types of communications associated with this "system") are without a doubt a major reason for the prosperity, industrial growth and power of the U.S.

I believe that the next AT&T system will begin to take shape in 1985, and be providing full services to the top 50 U.S. population centers by 1990, with full coverage of the North American continent by 2000. Let's take a look at what this next "system" might bring.

With the present cable, microwave, satellite, and radio network under the direct and indirect control of AT&T, it is possible today to communicate in "full duplex" with almost every residence and business in the United States, and many foreign countries. Much of the capacity of this network is already utilized for data transmission, collection and distribution. This usage is for many different reasons, and by many varied interests ranging from Government activities to medical education to airline pilot weather data. Note: at present, the list of data accumulation and distribution uses for the AT&T system is growing at an accelerating rate, faster than ever before, and this rate will continue to accelerate for a long time to come as the true potential of the system, and the hunger for "DATA" is exploited by our friends at AT&T.

I believe that AT&T is about to introduce a new computer intended for everyone, from the home user, to the president of a large company, to the bank around the corner. This computer will become part of a system of distributive processing the likes of which the world has never dreamed of. These machines will be entirely "solid state" in that they will contain *no moving parts* (other than the keyboard system until full voice control is perfected).

These machines will have a megabyte or more of bubble memory, and at least 256K of regular RAM (bubble memory is relatively slow). There will be a 9- or 12-inch monitor, with color and graphics capability, and TV interface options. A keyboard of 90-plus keys will be provided and initially a 1200 to 4800 baud communication system. This will all be available to the customer as components, CPU, keyboard, and monitor, with several options for each component.

The systems will initially rent for between $50 and $250 per month, with all maintenance provided by AT&T. By 1990, the price should fall to around $25 to $75 (or the equivalent at the time, adjusted for inflation). The initial thrust of this product will be aimed at the business market place. As production distribution and profits rise, they will decrease the costs to truly enter the "home" market at around the cost of present phone service.

Now for the "magic" — the SOFTWARE system

The most expensive and the most volatile part of any computer system is the software. Not the operating system itself, but the user applications. The mass market for "software" does not exist! But the "mass use" potential of software does exist. Think about it. Any particular piece of music is enjoyed by many more people than own the recording. Movies are watched by tens of thousands more than own a legitimate copy of the movie. Libraries are used by many more people than own complete libraries, or for that matter, those that even own a significant number of books.

For the want of a better term, let's call this concept "mass-use software". All the items mentioned above are "authored", "edited", "produced", "published", "stored on media", "reproduced" and then "marketed". Computer software is handled in a similar manner until it reaches the "marketed" stage. In most cases you must buy or license a physical copy of the reproduced media to use the product. That would be like having to buy a "print" of a movie so you could watch it, or marketing a book and NOT allowing it to be added to libraries.

This is not efficient, and would have greatly reduced the mass acceptance of products from the aforementioned industries. In the early days of each of these industries, the one to one marketing concept was all that existed, just like the handling of software today. I believe AT&T is about to change that, and bring computing in the U.S. to a new age of maturity.

AT&T could easily provide fifty or more supercomputers, properly placed around the country. When you turned on your own AT&T computer, it would automatically establish a link to the local supercomputer. You would then select the software you wished to use for the day (or as long as your machine stays on). The software and all support files would be sent to your machine. The host system would automatically know who you are and provide you with your personal data files at the same time. It would then disconnect from your computer. When you complete your work and sign off at the end of the day, your machine would automatically call the host and send back your updated data files, and then shut itself off.

Initially, data transmission speeds will be limited to 4800 bits per second, or less. This will change rapidly in the 1990s as fiber optic cables are laid, and fiber optic links enter most houses in the U.S. This "optical cable" will carry your T.V. as well as your phone and computer services, at up to hundreds of times faster than existing data transmission speeds.

There are several very strong points in favor of such a system. First, software piracy is eliminated (or made very difficult), as only the host has your data, and you will not have on-sight permanent storage. Handling of diskettes, making proper backups of data files and the like would not be the user's problem (most users don't handle this job correctly anyway).

Second, the customer would

always have the most recent version of the software to run. Then, if a bug is reported to a vendor, he simply corrects the bug and updates the product on any one of the host supercomputers. In the middle of the night, the host machines exchange updated files. The next morning, all software is corrected for EVERY user. The same concept applies to documentation and updates.

Third, you will be able to send data to anyone in the country in just hours or maybe minutes. The postal service would finally be dealt a long deserved death blow.

Fourth would be the reduced cost of such a system for the user. How much software do you own that you DO NOT USE? With this system, you could select between dozens (or hundreds) of word processors or spreadsheet programs, and only pay when you use them. The "Software Usage Fee, For End Reception" ("SUFFER" for short) would probably range from 25 cents to as much as $25 per day for "rental". AT&T would keep a percentage of this fee as a "distribution" charge (20 to 30% seems reasonable). The SUFFER charges would be levied on your regular phone bill, along with: your mail charges, T.V. usage, data file storage charges, and of course your charges for local equipment, options and services. Sounds like AT$T will be getting a lot more out of us than when they were "just the phone company".

One last point is that AT&T would probably have little or nothing to do with applications software (too much trouble). They would accept most professionally written packages for installation on the system. It would be up to the developer to promote and advertise the product to generate usage. Programs with very limited use would be removed from the system. A good software product will make millions for the authors under this system. Users will get much better quality software, as "JUNK MERCHANTS" will not be able to survive under this type of system. The computer owners would have to use a product repeatedly to make it a success.

One big negative is that this whole system will make 1984 (the "Big Brother" syndrome) a reality, and turn Orwell into a prophet. The government will be able to get their hands on "anything and everything", from payroll data to the letter you write to your mother, and how much you owe on your house, directly through the system. Tax collections should skyrocket. This is the main reason why I feel this overall concept will become reality. The government will want access to the tremendous database created by this type of system. Complete electronic banking will become reality (there will be no cash transactions to avoid taxes) and no one will have any truly private information.

The end of an era and the dawn of the information revolution is upon us. Whatever the outcome, our civilization is about to undergo radical change.

For all that the AT&T Mega-lith has done for the American people in the past, AT&T, we thank you. One can only wish that the results of your next accomplishment will be as benevolent to the American people. I hope so.

Locating high memory routines under LDOS/TRSDOS 6.X

Richard Schulman, MagiComp
2710 W. Country Club Rd., Philadelphia, PA 19131

A short while after my Mod 4 came rolling in, I got the tech manual and turned to the SVC section to find out how to convert my Mod 3 subroutines. One of the first SVCs I looked for was the one to get USTOR$ - the LDOS 8 byte storage area allocated to the user. I store the addresses of my subroutines there so that I can find them from LBASIC. Unfortunately, I never found the SVC - because there isn't one. So I plunged in to find out how to locate my routines.

LSI has adopted a convention for a header to precede high memory routines. Using this header allows an SVC to locate the routine for you. Listing 1 demonstrates the technique by fetching the address of INKY4.

Listing 1

```
START:  LD      DE,FSPEC        ;header to find
        LD      A,83            ;GTMOD
        RST     28H
        .
        .
        .
FSPEC:  DEFM    'INKY4'
        DEFB    0DH
```

DE is loaded with the address of the name of the routine you want to find, and A with the number of the SVC (in this case 83). The name must be in UPPER CASE characters and terminated with a character whose code is in the range 0-31. After executing the SVC with RST 28H, the starting address of INKY4 is in HL, and the Z flag is set (NZ if it wasn't found).

Of course, that doesn't entirely solve the problem of how to get this address into BASIC so that you can CALL (or USR) the routine.

The method I chose involves passing a parameter to the routine, and sending the address back in the parameter. The key to finding the routine is to know where to find the name of the routine (so it can be loaded into DE). The program is shown in Listing 2. The parameter passed to this program is the ADDRESS of the program itself.

That is the secret to finding the location of the name of the routine you wish to locate. The program is exactly 18 bytes long. Therefore adding 18 to the address of the program (the parameter we passed) gives us the location of the name of the routine we are searching for. That is the purpose of the LD HL,18 and ADD HL,DE instructions. The 18 instructions can be loaded into an integer array as shown in the BASIC program in Listing 3.

On line 16, X is set to the location of X(0) in memory. It is both the location of the routine CALLed in line 17 and the parameter passed to it. The integer INKY4 is the location of the high memory routine.

There is one more secret to success. You have to translate the name of the high memory routine into integers. The letters I,N,K,Y and 4 are represented in memory by the ASCII codes 73,78,75,89 and 52 respectively. To find out what integers to use, let's look at I and N. Those two letters make up one integer. Set I%=0. Then poke the codes for the letters into I%: POKE VARPTR(I%),73:POKE VARPTR(I%)+1,78. Then PRINT I% and you will find that I%=20041. The value for NK is 22859, and the value for 4 + carriage return (code 13) is 3380. Those are the last three values in the DATA statement at line 15.

If your routine name has an even number of characters you can terminate the name with an integer value of 0-31 (assuming an extra byte of value 0 following the terminating character). I just use 13 from habit.

Listing 2

```
PUSH    HL                      ;Save the parameter address
LD      E,(HL)                  ;Load the actual parameter
INC     HL                      ;itself into DE
LD      D,(HL)
LD      HL,18
ADD     HL,DE                   ;Add 18 to DE
EX      DE,HL                   ;Address of 'INKY4' to DE
LD      A,83                    ;SVC number
RST     28H
POP     DE                      ;Recover parameter location
EX      DE,HL                   ;Move routine location to DE
LD      (HL),E                  ;Move LSB to parameter
INC     HL
LD      (HL),D                  ;Move MSB to parameter
RET
DEFM    'INKY4'
DEFB    ØDH
```

Listing 3

```
14 DIM US(11):FOR X=Ø TO 11:US(Ø)=Ø:NEXT X
15 DATA 24293,22Ø51,4641,64ØØ,161Ø7,-4269,-5167,9Ø75,
   -13966,2ØØ41,22859,338Ø
16 FOR X=Ø TO 11:READ US(X):NEXT X:X=VARPTR(US(Ø))
17 CALL X(X):INKY4=X '** SAVE INKY4 ADDRESS **
2Ø1ØØ '** INKEY ROUTINE **
2Ø1Ø5 CALL INKY4(Z)
```

Listing 4

```
;************
;** HEADER **
;************
BEGIN:   JR      START
         DEFW    LAST-1   ;HIGHEST MEMORY BYTE
         DEFB    5
         DEFM    'INKY4'
MODDCB:  DEFW    $-$
         DEFW    Ø
START:   PUSH    HL       ;SAVE LOCATION OF PARAMETER
```

High Memory Headers

None of this is possible without the proper header. My header for INKY4 is shown in Listing 4. The first two bytes jump to the start of the actual routine. Next is a two byte integer with the address of the highest byte of memory occupied by the routine. Then one byte which gives the length of the name of the routine, 5 in this case - INKY4. There follows two byte reserved for the address of a Device Control Block if the routine is associated with a device and two bytes that are reserved for I don't know what.

Do it Again

You can reuse the routine in lines 14-17 in the same program. Line 18 would reset US(9) to (11) or however many elements after US(8) need to be changed. For example, let's find the routine FLASH. Line 18 would be US(9)=19526:US(10)=21313:US(11)=72 (I terminated 'FLASH' with a 00), followed by X=VARPTR(US(0)):

CALL X(X):FLASH=X. You can do it as many times as you need in the same program. And since you can wipe out arrays in BASIC under LDOS (TRSDOS) 6.x, you can ERASE US when you're done to reclaim its space and have the addresses of your routines in the integers INKY4 and FLASH (and whatever else).

Be sure to put X=VARPTR(US(0)) before each CALL X. BASIC moves things around dynamically and the address of US(0) might change between calls to X. And be absolutely certain that X is an integer. I can promise you from experience that you will not like the results if X in not an integer.

That's all there is to it. Simply MERGE lines 14-17 with your BASIC program, then remove the last three integers and replace them with the name of the high memory routine you are looking for.

Parity = Odd

© 1983, Tim Daneliuk, T&R Communications Associates

Welcome to PARITY = ODD! This may be the first time you've seen this column, so introductions are in order. PARITY = ODD originated in the early days of the LDOS Quarterly. At the time, I was reviewing TRS-80 related products for several magazines, as well as beta testing products for the LDOS operating system. In talking with Bill Schroeder of Logical Systems, it became clear that LDOS was soon to become a dominant force in the TRS-80 industry. And what a force it is! There are LDOS products for the TRS-80 Models 1 and 3, and the latest machines in that family (the Models 4 and 4P) use a descendant of LDOS 5.1 as their primary DOS. Yes folks, TRSDOS 6 is a version of LDOS. In fact, LDOS and TRSDOS 6 are so closely related in concept that most of the major commands work almost identically.

Originally, PARITY = ODD was created as a forum for examining topics of particular interest to the LDOS user. In this column you'll see product reviews, discussions of programming technique, solutions to user's problems, and almost anything else that strikes my fancy. If you have a particular question or want to see a specific product reviewed, let me know. Moreover, feel free to bring up topics unrelated to LDOS directly. In the coming installments you'll probably see items concerning CP/M and MS-DOS now that Tandy will be distributing these operating systems.

I can be contacted by mail and via CompuServe. If you choose to write, please send your letter to this magazine and my attention. If you want to "talk" by electronic mail, you'll find me skulking about on the LDOS Special Interest Group (SIG) of Compuserve. My PPN # is 70745,1520. Either way, there are some vital pieces of information you should always include. First, *please* include your name and address (a telephone number is helpful too). Second, *always* include a description of the system you are using. Please state explicitly which TRS-80 or TRS-80 "work alike" you are using as well as the types and sizes of disks drives, controller, etc., you have. Finally, if you are requesting help with a problem, please try to describe the problem in a logical, step-by-step manner. Be as explicit as possible. I can't much with a problem like "It doesn't work right".

Remember, this column reflects my opinion. I try to responsibly evaluate those products about which I write, but errors can and will happen. If you find such an error, let me know.

Chips

Tandy recently unveiled their MS-DOS machine, the Tandy 2000. The 2000 is notable because it is one of the first personal computers to use the Intel 80186 microcomputer chip. To understand the power of the '186 we need to take a quick look under the hood of the modern microprocessor.

The much-touted IBM-PC uses the Intel 8088 microprocessor. IBM will tell you that this makes the PC a "sixteen bit" computer. Horse-feathers! Internally, the 8088 contains data registers which are indeed 16 bits wide. BUT -- externally (i.e. those places where the 8088 "talks" to the rest of the computer and the outside world) data is transferred 8 bits at a time. This is a situation which is much like the 8 bit Z-80 microprocessor. The Z-80 also has 16 bit internal registers and it transfers data to the rest of the system a byte at a time. In fact, the only significant advantage that a 8088 has is that it can address up to 1 Megabyte of memory directly compared to the Z-80's 64K Byte maximum. This 16 Bit internal/8 Bit external architecture is a large part of the reason the IBM-PC is so wretchedly slow! That's just my opinion of course, but if you want an eye-opening experience, try benchmarking a program on a TRS-80 Model 4 or LOBO MAX-80 against an IBM-PC. The PC will generally be slightly faster, but not enough in my judgement to justify the 30-50 percent price difference between these machines.

So, what's an 80186? It is an enhanced version of the Intel 8086 microprocessor. The 8086 is a processor which has the same instruction set as the 8088. However, the '86 not only has internal 16-bit data paths, it also has EXTERNAL 16-bit data paths. For this reason alone, it generally runs quite a bit faster than the '88. The '186 enhances this further by including almost an entire computer on one chip, and offering a faster clock speed. In addition to the 8086 microprocessor, the 80186 includes such things as a DMA (Direct Memory Access) controller, and memory chip select logic all on one piece of silicon. This makes the '186 a very cost effective part when you're designing a complete computing system. The 80186 also has some new instructions which the '88 and '86 don't have. The '186 also has the advantage of being compatible with all the instructions of the '88 and '86. This means that programs written for the latter two will also run on the 80186!

The bottom line is that you can expect the Tandy 2000 to be quite a performer. Though it does not appear to be "compatible" with the IBM-PC, most software which uses the MS-DOS operating system for I/O (and doesn't try to "talk" to the hardware directly) ought to work with the Tandy 2000. The one potential weakness of the 2000 is that is uses "quad density" (double-sided, double-density eighty track) disk drives. While this gives an enormous amount of storage, it may not be easy to move programs and files to and from "normal" MS-DOS forty track drives. I'm expecting a review machine fairly soon - I'll let you know what I find.

Disk, Disk, Disk . . .

If you've followed the evolution of LDOS at all, you probably realize that Logical Systems also publishes many useful add-on products for their operating system. The latest such product is called diskDISK, and will be especially useful for those of you using LDOS on a hard disk drive. To understand how diskDISK can be used, we need to step back and look at a hard disk system.

By virtue of its tremendous storage capacity, a hard disk can contain literally hundreds of files. While all that storage is great, it becomes difficult to keep track of all the files on the disk after a while. One partial solution to this dilemma is to use hard disk "partitions". A partition is simply part of the hard disk treated as a separate logical drive. For instance, if you had a disk drive with four platters (storage surfaces) each capable of storing 2 Megabytes, you could organize the drive one of several ways. You could have one eight Meg, two four Meg, four two Meg or any combination of the above which totals 8 Megabytes of storage. By the way, partitioning is only possible if the hard disk driver program is written to do so. The point is that one large hard disk is made to "look" like several smaller disk drives. The advantage of this approach is that you break the mass storage into smaller, more easily managed "chunks" of storage.

Even with disk partitioning, there are still times when you will be limited by the large number of files you have to manage on a hard disk. It would be ideal to go even further and sub-divide a given PARTITION of a hard disk. The diskDISK utility does just that. With diskDISK, you can create a "logical" storage area on a hard disk which has the capacity of a 5 or 8 inch floppy. In other words, diskDISK creates a file which is large enough to store as much as say, a single 5" floppy. Then this FILE is installed in LDOS as a logical disk drive.

For instance, I generally use a LOBO 1850 hard disk as the principal drive of my system. This drive stores 8 Megabytes, and is partitioned into two 2 Megabyte partitions (Drives 0 and 1) and one 4 Megabyte partition (Drive 2). On Drive 1, I have a file called C/DD. This is a diskDISK file which ordinarily looks just like any other file to the system. If I issue the command "DD :3 C/DD" from LDOS Ready, this FILE is installed as logical Drive 3. From then on, any LDOS operation on Drive 3 (DIR, FREE, KILL, etc.) takes place physically in the file C/DD. This is entirely invisible to the user. The diskDISK drivers make C/DD "look" like a floppy disk drive. In the case of this particular file, I used diskDISK to "format" it to look like a double-density, double-sided, 40 track disk drive. This gives me about 360K of storage on this logical drive, more than enough to store a C compiler and my current C program source files.

You can create a different diskDISK file for each major type of file in your system to help organize your files. For instance, I can have one diskDISK set up for BASIC programs, another for PROFILE, and still another for utilities. There's a hidden benefit in all this too. The granule size for an LDOS hard disk system is always at least 4K. Even if your file only has 2 bytes of data in it it will still occupy at least 4K of disk space. When you save a file on a diskDISK, the granule size is that of the floppy disk you're emulating. For instance, files stored on a SSDD 5" diskDISK use 1.5 K granules. Your two byte file will occupy 1.5 K instead of 4 K, for a net savings of 2.5 K of disk space. If you have many small files, the savings are tremendous. A special diskDISK format (called type 1) uses 256 byte granules for the most efficient storage possible.

Another advantage is that a diskDISK only occupies one directory entry on the hard disk, no matter how many files actually exist inside. If you have a lot of small files on a hard disk or 8 inch floppy, you can easily run out of directory space before you run out of disk space. In a sense, diskDISK gives LDOS the capacity of having sub-directories similar to MD-DOS 2.x and UNIX.

diskDISKs can be created on any type of LDOS compatible media. This product is available for either the Model 4 running TRSDOS 6 or the LDOS 5.1.x family of DOS products. Either package costs $99 and is available from Logical Systems Inc., 8970 N. 55th Street, P.O. Box 23956, Milwaukee, WI 53223.

Random Items of Interest

Because of the volume of products sent to me for review, it isn't always possible to look at each one in depth.

From time to time, you'll see mini-reviews like these. First, if you're an LDOS 5.1.x user and need a great disk cataloging program, take a look at ZCAT from MicroConsultants. It is written entirely in assembly language and is the best of its type I've seen to date. It costs only $35 and is available from MicroConsultants, 7509 Wellesley Drive, College Park, MD 20740-3037 (301) 474-8486.

Model 4 owners may be interested in the new "PRO-CESS" utility from MISOSYS. This product is very similar to the CMDFILE program included with LDOS 5.1, except that PROC-CESS runs under TRSDOS 6. This product also has some nice new twists, like being able to sort load module records by address and conversion of X-type patches to D patches. PRO-CESS is $40 and can be obtained from MISOSYS, P.O. Box 4848, Alexandria, VA 22303-0848, (703) 960-2998.

Finally, if you use SuperScripsit, there are two sources of printer drivers you should know about. One is softERware, and the other is PowerSOFT. I've used the softERware product which seems to work fine, but I've not seen the PowerSOFT driver. Contact these companies for more details: PowerSOFT, 11500 Stemmons Freeway, Suite 125, Dallas, TX 75229 (214) 484-2976, or softERware, 300 Grenola St., Pacific Palisades, CA 90272 (213) 459-3414.

The Finishing Touches

That about wraps it up for this installment of PARITY = ODD. Hopefully the next time you read this column, I'll have something to report on the Tandy 2000, as well as the usual collection TRS-80 reviews and trivia! So long for now.

The "C" Language, part 6

Pointers, arrays, structures and common errors

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

Could you use some POINTERS on how to STRUCTURE better C programs? In this part, Part VI, structures will be introduced and the discussion on pointers and arrays will continue. Oh? You thought when you read the word POINTERS and the word STRUCTURE that this part would really be discussing techniques for improving your C code? Ha! Well, okay, not to disappoint you, included in this part is a discussion of the most common errors or "things" not to do in a C program. Will that help?

First, let's continue on from the last part with pointers and arrays.

In the last part, you saw an expression *ptr++. Were you puzzled? Remember back when the ++ and -- operators were introduced? It was stated that ++ added one to its operand and -- subtracted one from its operand. Be careful applying this to pointers. The "one" referred to which is added to or subtracted from a pointer is actually a scale factor. This scale factor is dependent on the type the pointer points to. That means it is scaled by a size equal to the data type length. This holds true for all "pointer arithmetic". (For example, in a Z80 based machine the scale factors are 1 for char, 2 for int.)

There are some rules to follow when doing arithmetic in C using pointers. It is legal to:

add an integer to or subtract an integer from a pointer
subtract a pointer from a pointer
compare a pointer to another pointer

All other conceivable arithmetic, including shifting or masking is illegal. Note: a pointer containing NULL or 0 is a special case. The C language guarantees that if a pointer points to valid data, it will not contain 0. The 0 value is usually used to indicate an error condition. An example of this would be when a storage allocate function is called. This function may have been designed to return a non zero pointer to the beginning of the allocated storage. If storage can not be allocated it could return a NULL (zero) value indicating an error of some type occurred. Consider the statements below for the discussion following:

```
char *ptr;
static char a[5] = "test";
ptr = a;
++ptr;
```

ptr is a pointer to type character. ptr is initially set to the address of the array a. This is written as &a[0] or simply a. Next, ptr is incremented to point to the next element of the array. This is written as ptr++. (Other possible ways to code it, in this example, could have been *ptr++, *++ptr, *(++ptr) or *(ptr++). Note that (*ptr)++ would create a different undesired result than *ptr++. The ++ and * operators are of equal precedence and associate right to left.) From the above statements, you can conclude that array subscripting can be done by incrementing a pointer. You can also conclude that the following two expressions are equivalent:

```
ptr = a;
ptr = &a[0];
```

(Note that, in effect, an array name is a pointer expression. Note also that using pointers rather than array subscripting usually results in more efficient code.)

As general rules:

a[n] is equivalent to *(a+n)

*(&a[n]) is equivalent to *(a+n)

&a[n] is equivalent to &a[0]+n is equivalent to a+n

Perhaps if I spelled out how to "pronounce" some of the expressions used in the general rules above, these rules might become more clear?

```
&         means the address

*         means the data at

a[n]      means element n of array a

&a[n]     means the address of element n of array a

*(&a[n])  means the data (element) at the address of

                  element n of array a

          This says the same thing as element n of

                  array a

*(a+n)    means element n of array a

a + n     means element n of array a
```

Did the above help?

If you have been looking at some sample C programs, you may have seen by now that sometimes an array name is written as a[] or *a when used as parameters in a function. Rather nice don't you think? The function, when passed an array name, can treat it as an array, as a pointer or both. If you have some doubt, look at the C code in Listing 1.

The arguments argc and argv are not new to you, they were described in Part II. As you noted, argv is treated as an array and as a pointer in the above program. Are you curious about the argv[0][0] expression used in the printf function? What will print is a single character, the first character of the command line argument after the program name. If the above program was called test, to invoke it and pass it an argument, you might type:

test -l myfile/dat

If you compile it and try it using the above invocation, you should see the - printed. (Try it with different argument values and different numbers of arguments.) Of what value is this? Well, actually this program might be used as part of a larger program and the argv[0][0] could be used to test for a "switch" such as + or - in front of a parameter. In the example invocation above, I included the myfile/dat parameter to suggest some possibilities for you to ponder!

Listing 1

```
main (argc, argv)

    int argc,

    char **argv;

{

    if (argc < 2) {

        printf("Error - no parameter was given!");

        exit(1);

    }

    ++argv;

    printf("%c\n",argv[0][0]);

}
```

The argv function parameter, as mentioned in Part II, is a pointer to an array of pointers. Here is a list of possible ways or forms in which you might see it used: argv, *argv, argv[n], *argv[0], (*argv)[0], argv[0][0]. Having some trouble "visualizing" what each represents? Look at a possible storage map (chart) of argv:

```
      |---------|

argv  | address | ---------|

      |---------|          |

                           |

                      |----V----|

   argv[0] or *argv   | address | ------------|

                      |---------|             |  *argv[0]

       |------------  | address |             |    or

       |              |---------|             |  argv[0][0]

       |              | address |        |----V----|

       |              |---------|        | string  |

  |----V----|         |         |        |  data   |

  | string  |

  | data    |
```

I hope the above map will be of some aid. Try to fit into the above map all of the ways of using the function argument argv. Enough of this for awhile! Let's switch topics and introduce structures.

A nice feature for a language is the ability to group variables of different types and treat them as one. This grouping of variables, called a structure in C, is called a record in other computer languages. Listing 2 is an example of the declaration of a sample C structure.

The struct keyword is used to declare a structure. An optional name or tag can follow the struct keyword. In this example, I used the tag of payroll. The tag names the structure and can be used as a shorthand method for the complete structure declaration. For example, to declare two more structures of type payroll, it might be done as follows:

struct payroll person1, person2;

The variables declared in the structure are referred to as members. Structure members or tags can have the same name as other simple variables. The C compiler can tell them apart due to the way they are used. Members of a structure are referenced as follows:

structure-name.member

The "." is called the structure operator. It connects the structure name to a member name. More will be said about structures in the next part!

Now, as promised, Table 1 is a list of many of the most common errors found in a C program. Keep these "mistakes" in mind as you code in C. Looking out for these pitfalls will help you design a more "bug free" program.

As you read some of the above most frequent C coding errors didn't you say to yourself, "Yes, I have done that before."? If you did, you are not alone! Some of these errors are caught by the C compiler, many are not. Another program for detecting possible errors in C code is called LINT. It typically better enforces the rules of C and reports more possible errors than does the C compiler.

In the next part, the C programming environment will be discussed along with many functions which are part of (or should be part of) the "standard" library. Structures will also be covered in more detail. Practice your C coding techniques until then!

Listing 2

```
struct payroll { char name[30];
                 int    age;
                 char   sex;
                 int    pay;
               };
```

Table 1 — C Language Pitfalls

```
Using = instead of == in an if statement
Thinking arrays start at index 1 instead of 0
Unclosed braces or brackets
Forgetting a ;
Using / instead of \
"Off by one" errors in looping or array indexing
declaring function arguments after the {
Forgetting the precedence of operators
Thinking C has built in string comparisons
Using ' instead of "
Using () instead of []
function arguments placed in the wrong order
Not reserving an array element for the terminal \0
Forgetting about "side effects"
```

LDOS: How it works — an introduction to COMM and LCOMM

Joseph J. Kyle-DiPietropaolo

LCOMM (on LDOS 5.1.x) and COMM (on TRSDOS 6.x) are both very powerful communications packages, but few people use them to their full advantage. Admittedly, the manual(s) are a bit terse in regards to these packages, but with a little help everyone should be able to use these utilities. If you need any help in regards to physically connecting your modem to your computer, contact your vendor for the proper cables and instructions.

Getting started– First, type the commands below at DOS ready:

```
Under TRSDOS 6.x:
  SET *CL COM/DVR<enter>
  SETCOM (DTR,W=8,P=N)<enter>
  COMM *CL<enter>

Under LDOS 5.1.x:
  SET *KI KI (T,J)<enter>
  SET *CL RS232T (DTR,W=8,P=N)<enter>
  LCOMM *CL<enter>
```

Note that <enter> means to press the "ENTER" key. If you have a Model I, on the second line type: "SET *CL RS232R<enter>" instead. Your system is now set up for RS232 communication at 300 baud, the most common mode when using a modem over the telephone line. This setup should work for most modems, including the Radio Shack modems and their DC Modem II.

When you enter COMM/LCOMM in this manner, you are immediately in the terminal mode. Anything you type will be sent to the modem, and whatever is received will be displayed on your screen. If you use a JCL (Job Control Language) procedure to set this up automatically for you, don't forget to add a line reading "//STOP" as the last line of the JCL file. If you don't, control will be returned to DOS if you attempt to execute certain COMM/LCOMM commands.

Now to actually communicate. Let's take CompuServe as an example. Dial the phone number for your local CompuServe node. If you have a "smart"-type modem, refer to your operators manual for dialing instructions. Once the phone number has been dialed, put your modem on-line. For most "dumb" modems, this will mean flipping the switch on, or placing the phone handset into the acoustic cups. Most "smart"-type modems automatically enter the on-line mode after dialing.

When the "CD" or carrier detect

light comes on, type a <control>-C. On the Models I and III, this is the combination <Shift><Down Arrow> (meaning "control") and (while still holding them down) then a <C>. On the Model 4, you have a <control> key. Depress this, then (while still holding it down) type a <C>. In either case, Compuserve should respond with "User Id:". This is your first prompt. Type in your user number (provided in your sign-up package - obtain from Radio Shack). For example, the user number for the people here at LSI is 76703,437. We would type in "76703,437;".

The semi-colon is very important. Until you tell CompuServe otherwise, they assume that you are using what is called a "Videotex-compatible terminal". When using LCOMM or COMM, this will cause all sorts of nasty things to appear on your video display. The semi-colon prevents this from happening. Once you are logged on, if you change your terminal type (tell CompuServe that you have an "other"-type terminal) this will not happen and you may omit the semi-colon.

CompuServe will now respond with "Password:". Type your password from the sign-up package and hit <enter>. After a few seconds, CompuServe will respond with its first menu. Congratulations! You have successfully communicated!

Now we know how to establish a communications channel from our computer, let's look at how we give instructions to COMM/LCOMM. All commands begin with a <clear><keystroke> or <clear><shift><keystroke> sequence. For instance, pressing <clear> and holding it while you press <8> will display the COMM/LCOMM command menu. The <clear> key is used as a second type of "control" key, one that has special meaning to COMM/LCOMM (and many other DOS utilities).

But- what does all this menu information represent? Let's take a look at an example. *PR stands for the "logical" printer device, and the word "ON" is relatively self-explanatory. If you hit <clear><3> and then <clear><:>, (the keystrokes that represent "*PR" and "ON" respectively) the *PR device will be "turned on". Typing <clear><8> will re-display the menu. Do you see the difference? There is now a "*" below the *PR device, indicating that it is "ON". Now, any characters received by your computer will be sent to the printer in addition to the video display.

So far, we have covered the initial "set-up" phase of COMM/LCOMM. In future installments, we will cover more advanced features of these utilities, such as file uploading and downloading.

Les information — faster file access

Les Mikesell

When accessing data files and devices in machine language, there are many different techniques that can be used. The operating system @GET and @PUT calls (these are called the "Byte I/O calls" because they move one byte at a time) are easy to use, and allow use of either devices or files. See past issues of the LDOS Quarterly (volume 2, numbers 2 and 3) for more information on this method.

@GET and @PUT are convenient, but there is a speed penalty as compared to full sector operations. The single byte operations can only access one sector per disk revolution, while full sector operations may be able to handle an entire track in two or three revolutions (depending on the disk type and the processing time needed between sectors). Thus, it may be possible to speed up disk operations considerably by buffering as much as possible in memory using full sector disk I/O.

One significant drawback to this approach is that the program then becomes responsible for observing or setting the proper end-of-file offset when the data file does not end on a sector boundary.

The sector interleave on floppy disks is designed to allow just enough time to move a sector of data in or out of the file buffer before the next consecutive sector passes under the read/write head. Any additional processing at this point will usually cause the next sector to be missed, and necessitate a wait until the next revolution of the disk. When using a hard drive or MemDISK, the interleave factor is not critical, but programs will still benefit from the reduced overhead of full sector operations.

After opening a file, its size can be determined from the contents of the open FCB (file control block). The ending record number (ERN) is stored at FCB+12 & 13, and the end of file offset (EOF) is at FCB+8. If the ERN is 00, the file is empty. Otherwise, ERN-1 is the number of full sectors in the file, and EOF is the number of bytes included in the ending sector (where 0 = 256). Thus EOF-1 is the offset of the last valid byte in the file buffer when the last sector is read. This may be easier to remember by keeping in mind that these three bytes are always maintained as a pointer to the next record to write to extend the file.

For many operations, a file may be read into memory until the DOS error 1CH (end of file) or 1DH (past end of file) occurs, then the buffer pointer adjusted back to the correct byte offset in the previous sector. However, it may sometimes be necessary to determine if the current sector contains any data past the end of file before reading the next sector (which would return the EOF error). In this case, the next record number (NRN) at FCB+10 & 11 may be compared to the ERN after the read. If the NRN is the same as the ERN, the sector just read contains the end-of-file.

When writing sequential data using the full sector operations, it is necessary to update the EOF byte before closing the file. If @POSN has been used, it is necessary to update the ERN, since the system will then consider the file to be "random-access" and update the length only if it has been extended. Moving the NRN into the ERN in the FCB will set the current position as the end of file even if the previous ERN was larger.

Listing 1, a program to either add line-feed characters after carriage returns, or remove line-feeds, demonstrates some of the techniques of handling byte data with the DOS sector operations. Note that the input routine simply moves the buffer pointer in the FCB for each sector read rather than moving the data from the file buffer.

The output routines are a little slower, and will miss the disk interleave on a Model I or III with the standard CPU speed. This could be avoided by processing the data into a larger buffer space, then writing several sectors at once. Pre-allocating the disk space before the write would increase the speed also, by reducing the number of times the system has to go to the disk directory. This may be done simply by using @POSN and @WRITE to write (anything) to the last sector of the output file, then re-position to record 0.

This program may also be assembled for use with the TRSDOS/LDOS 6.x system by deleting the beginning lines between the asterisks, and including the standard 6.x header. For TRSDOS/LDOS 6.x operation, delete lines 100 to 410 and insert the code from Listing 2.

Listing 1 LDOS 5.1 version

```
00100 ;**********************************
00110 ;This part is for LDOS 5.1
00120 ;Operating system entry points:
00130 @ABORT  EQU     4030H
00140 @CLOSE  EQU     4428H
00150 @DSPLY  EQU     4467H
00160 @ERROR  EQU     4409H
00170 @EXIT   EQU     402DH
00180 @FSPEC  EQU     441CH
00190 @INIT   EQU     4420H
00200 @KEYIN  EQU     0040H
00210 @OPEN   EQU     4424H
00220 @READ   EQU     4436H
00230 @WRITE  EQU     4439H
00240 HIGH$   EQU     4411H           ;Model 3
00250 HIGH1   EQU     4049H           ;Model 1
00260 ;
00270         ORG     5200H
00280 ;Put file buffer first to force location
00290 ;On memory page boundary
00300 BUFFR1  DS      256
00310 BUFFR2  DS      256
00320 ;Machine specific code:
00330 BEGIN:  LD      A,(125H)        ;Check modl/3 ROM
00340         CP      'I'
00350         LD      HL,(HIGH$)      ;Mod 3 location
00360         JR      Z,SETHI         ;Go if mod 3
00370         LD      HL,(HIGH1)      ;Mod 1 location
00380 SETHI:  LD      (MYMEM),HL      ;Store correct value
00390 ;
00400 ;End of LDOS 5.1 specific code
00410 ;********************************
00420 ;
00430 ;Special chars
00440 ETX     EQU     03H
00450 CR      EQU     0DH
00460 LF      EQU     0AH
00470 ;
00480 ;
00490 ;FCB offset definitions
00500 BUFRLO  EQU     3               ;Buffer address
00510 BUFRHI  EQU     4
00520 ERNHI   EQU     13              ;Ending record #
00530 ERNLO   EQU     12
00540 EOF     EQU     8               ;Offset of last byte
00550 NRNLO   EQU     10              ;Next record pointer
00560 NRNHI   EQU     11
00570 ;
00580 START:  LD      HL,LOGON        ;Log on
00590         CALL    @DSPLY
00600 DISKIN: LD      HL,MSG1         ;Prompt for input file
00610         CALL    INPFSP          ;Get answer
00620         LD      DE,FCB1         ;=>FCB
00630         CALL    @FSPEC          ;Move filename
00640         LD      B,0             ;LRL=0  (256)
00650         LD      HL,BUFFR1       ;=>disk buffer
00660         CALL    @OPEN           ;Open the file
00670         JR      Z,GOTINP        ;Go if successful
00680         CALL    SHOERR          ;Else report error
00690         JR      DISKIN          ;And ask again
00700 ;
00710 ;File is open, check if it contains any records
00720 GOTINP: LD      HL,(FCB1+ERNLO)
00730         LD      A,H             ;Is ending record 0?
00740         OR      L
00750         JP      NZ,ASK          ;Go if file has data
00760         LD      HL,NODAT        ;Else report empty file
00770         CALL    @DSPLY
00780         JP      @ABORT          ;And quit
00790 ;
00800 ASK:    LD      HL,MSG2         ;Prompt for output file
```

```
00810          CALL    INPFSP          ;Get answer
00820          LD      DE,FCB2
00830          CALL    @FSPEC          ;Move filename
00840          LD      B,0
00850          LD      HL,BUFFR2
00860          CALL    @INIT           ;Create file
00870          JR      Z,ASK1A         ;Continue if good init
00880          CALL    SHOERR          ;Else report error
00890          JR      ASK             ;And re-prompt
00900 ;
00910 ;Set up for add or remove LF's
00920 ASK1A:   LD      HL,MSG3         ;Remove linefeeds?
00930          LD      DE,RLF          ;=>flag
00940          CALL    SGETYN          ;Prompt, set flag
00950          JR      Z,AGAIN         ;If yes, skip 2nd prompt
00960 ;
00970 ASK1B:   LD      HL,MSG4         ;Add linefeeds?
00980          LD      DE,ALF          ;=>flag
00990          CALL    SGETYN          ;Prompt, set flag
01000          JR      NZ,ASK1A        ;Must do one or the other
01010 ;
01020 ;Read file into memory - set (MORE)=0FFH if it doesn't fit
01030 AGAIN:   LD      HL,(MYMEM)      ;=>end of free space
01040          LD      BC,BUFFER       ;=>working buffer start
01050          OR      A
01060          SBC     HL,BC
01070          LD      B,H             ;# of sectors that will fit
01080          LD      HL,BUFFER       ;Start of buffer
01090          LD      DE,FCB1         ;=>input file buffer
01100 RLOOP:   LD      (FCB1+BUFRLO),HL        ;Set load address
01110          CALL    @READ           ;Get a sector
01120          JP      NZ,CKEND        ;End or error
01130          INC     H               ;Bump ptr for next
01140          DJNZ    RLOOP           ;Stop if memory is full
01150 ;Now check to see if the last sector contained the EOF
01160 ;To be sure all data loaded belongs in file
01170          PUSH    HL              ;Save current posn
01180          LD      HL,(FCB1+NRNLO) ;Check if this is
01190          LD      DE,(FCB1+ERNLO) ;The last sector
01200          OR      A
01210          SBC     HL,DE           ;Is NRN=ERN?
01220          POP     HL              ;Get end ptr
01230          JR      Z,ISEOF         ;Go if this is EOF
01240          LD      A,0FFH          ;Set flag for more input
01250          LD      (MORE),A
01260          JR      FINIS           ;And start processing
01270 ;
01280 CKEND:   CP      1CH             ;EOF?
01290          JR      Z,ISEOF
01300          CP      1DH             ;Or past EOF?
01310          JP      NZ,DOSERR       ;Quit if other error
01320 ISEOF:   LD      A,(FCB1+EOF)    ;Get EOF offset byte
01330          DEC     A               ;Point to end byte
01340          LD      E,A
01350          LD      D,0             ;DE=offset of EOF
01360          DEC     H               ;Back up to last full sector
01370          ADD     HL,DE           ;Add in contents of last sector
01380          INC     HL              ;HL=>1 past buffer
01390 ;
01400 FINIS:   EX      DE,HL           ;Pointer to DE
01410          LD      BC,BUFFER       ; START OF DATA AREA
01420 ;
01430 ; Memory is loaded, check for changes
01440 ; BC=>current char, DE =>end+1
01450 ACHAR:   LD      A,D             ;Is this the end of the buffer?
01460          CP      B
01470          JP      NZ,NOTEND       ;Go if not
01480          LD      A,E
01490          CP      C
01500          JR      Z,CHECKM        ;End, check for more input
01510 NOTEND:  LD      A,(BC)          ;Get a character
01520          INC     BC              ; bump pointer
01530          CP      LF              ; a linefeed?
01540          JP      NZ,NTLF         ; go if not LF
01550          LD      A,(RLF)         ;Removing line feeds?
01560          OR      A
01570          JR      Z,DOLF          ;Not removing line feeds
01580          JR      ACHAR           ;Skip LF
01590 NTLF:    CP      CR              ; carriage RET?
01600          JR      NZ,WRBYT        ;Write if not CR
01610          CALL    PUTTER          ;Write CR
01620          LD      A,(ALF)         ;Adding line feeds?
01630          OR      A               ;Test
01640          JR      Z,ACHAR         ;Skip if not wanted
01650 DOLF:    LD      A,LF            ;Load the line feed
01660 WRBYT:   CALL    PUTTER          ;Write char to file
01670          JP      ACHAR           ;Loop through buffer
01680 ;
01690 ;End of buffer, is file done?
01700 CHECKM:  LD      HL,MORE         ;Is there more to read?
01710          LD      A,(HL)          ;Zero if done
01720          LD      (HL),0          ;Set for next time
01730          OR      A               ;Set Z if done
01740          JP      NZ,AGAIN        ;Loop till finished with files
01750 ;
01760 ;Finished, close the output file
01770          CALL    LSTSEC          ;Flush buffer
01780          LD      DE,FCB2         ;=>output FCB
01790          CALL    @CLOSE          ;Close output file
01800          JP      NZ,DOSERR       ;Go if error
01810          LD      HL,MSG5         ;Else report completion
01820          CALL    @DSPLY
01830          JP      @EXIT           ;And go back to DOS
01840 ;
01850 ;Set flag according to 'Y' response
01860 SGETYN:  PUSH    DE              ;Save flag address
01870          CALL    @DSPLY          ;Issue prompt
01880          LD      HL,YN$          ;Add Y/N?
01890          CALL    @DSPLY
01900          LD      B,1             ;Max input wanted
```

```
Ø191Ø         CALL    KEYIN            ;Get 1 char answer
Ø192Ø         AND     5FH              ;Force upper case
Ø193Ø         CP      'Y'              ;Is it Y?
Ø194Ø         POP     HL               ;Get flag address
Ø195Ø         RET     NZ               ;Return if not "Y"
Ø196Ø         LD      (HL),ØFFH        ;Set flag if "Y" response
Ø197Ø         RET
Ø198Ø ;Get a filename:
Ø199Ø INPFSP: CALL    @DSPLY           ;Display prompt
Ø2ØØØ         LD      B,1FH            ;Set max length
Ø2Ø1Ø ;Get user input
Ø2Ø2Ø KEYIN:  LD      HL,INBFR         ;=>buffer to receive input
Ø2Ø3Ø         CALL    @KEYIN           ;Get input
Ø2Ø4Ø         JP      C,@ABORT         ;Quit if BREAK pressed
Ø2Ø5Ø         LD      A,(HL)           ;Else pick up 1st char
Ø2Ø6Ø         RET
Ø2Ø7Ø ;
Ø2Ø8Ø ;Add a byte to disk buffer/write if full
Ø2Ø9Ø ;Note that disk buffer must end on XXFFH boundary
Ø21ØØ ;So the INC L will set the Z flag when sector
Ø211Ø ;Buffer is full.  (PUTPTR) is a pointer to the next char
Ø212Ø ;Position in the sector buffer
Ø213Ø PUTTER  LD      HL,(PUTPTR)      ;Point to buf pos
Ø214Ø         LD      (HL),A           ;Move to buffer
Ø215Ø         INC     L                ;Bump buffer ptr
Ø216Ø         LD      (PUTPTR),HL      ;Save for next
Ø217Ø         RET     NZ               ;If not full
Ø218Ø ;Write a physical sector to disk
Ø219Ø WSEC:   PUSH    DE
Ø22ØØ         LD      DE,FCB2          ;=>FCB
Ø221Ø         CALL    @WRITE           ;Write
Ø222Ø         POP     DE
Ø223Ø         RET     Z                ;That was easy..
Ø224Ø DOSERR: CALL    SHOERR           ;Report any error
Ø225Ø         JP      @ABORT           ;And quit
Ø226Ø ;
Ø227Ø SHOERR: OR      ØCØH             ;Mask for short msg.,ret
Ø228Ø         JP      @ERROR
Ø229Ø ;
Ø23ØØ ;Fill remainder of sector buffer with ØØ's
Ø231Ø ;Set the EOF offset, and flag DOS to reset the
Ø232Ø ;File's ERN to the current sector
Ø233Ø LSTSEC: LD      HL,(PUTPTR)      ;Get posn in buffer
Ø234Ø         LD      A,L              ;Did last write
Ø235Ø         OR      A                ;Hit sector end?
Ø236Ø         JR      Z,SETEOF         ;Finished on sec boundary
Ø237Ø         PUSH    AF               ;Save last char +1
Ø238Ø         XOR     A                ;Set A=Ø
Ø239Ø FLSEC:  LD      (HL),A           ;Zero remaining buffer
Ø24ØØ         INC     L
Ø241Ø         JR      NZ,FLSEC         ;Pad sec w/nulls
Ø242Ø         CALL    WSEC             ;Write the last sector
Ø243Ø         POP     AF               ;EOF byte to A
Ø244Ø ;
Ø245Ø SETEOF: LD      (FCB2+8),A       ;Set EOF offset
Ø246Ø ;Note: this step is actually not necessary unless @POSN called
Ø247Ø         LD      HL,(FCB2+NRNLO) ;Put NEXT record no.
Ø248Ø         LD      (FCB2+ERNLO),HL ;Into ENDING record no.
Ø249Ø         RET
Ø25ØØ ;
Ø251Ø ;
Ø252Ø ;END OF PROGRAM AREA
Ø253Ø ;ASCII DATA
Ø254Ø LOGON:  DB      LF,'TEXT FILE PROCESSOR  ',LF,CR
Ø255Ø MSG1:   DB      'Input Filespec? ',ETX
Ø256Ø MSG2:   DB      'Output Filespec? ',ETX
Ø257Ø MSG3:   DB      LF,'Remove line-feed characters',ETX
Ø258Ø MSG4:   DB      'Add line-feed characters',ETX
Ø259Ø MSG5:   DB      LF,'File output completed ',CR
Ø26ØØ NODAT:  DB      'Input file is empty!',CR
Ø261Ø YN$:    DB      ' (Y/N) ? ',ETX
Ø262Ø ;
Ø263Ø ;BUFFERS & POINTERS
Ø264Ø RLF:    DB      Ø                ;Remove LF flag
Ø265Ø ALF:    DB      Ø                ;Add LF flag
Ø266Ø MORE:   DB      Ø                ;More input flag
Ø267Ø EOFFLG: DB      Ø                ;Last sector flag
Ø268Ø MYMEM:  DW      Ø                ;HIGH$ pointer
Ø269Ø PUTPTR: DW      BUFFR2           ;Posn in output buffer
Ø27ØØ GETPTR: DW      BUFFR1+255       ;Posn (-1) in input buffer
Ø271Ø FCB1:   DS      32               ;File FCBs
Ø272Ø FCB2:   DS      32
Ø273Ø INBFR:  DS      4Ø               ;KB input buffer
Ø274Ø BUFFER  EQU     $                ; START OF DATA BUFFER
Ø275Ø         END     BEGIN
```

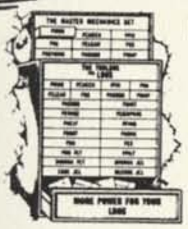
Listing 2 LDOS 6.x version

```
ØØ1ØØ ;*********************************
ØØ11Ø ; Header to convert programs with 5.1 CALLS to 6.x
ØØ12Ø ;
ØØ13Ø         ORG     26ØØH
ØØ14Ø BUFFR1: DS 256                 ;put on page boundary
ØØ15Ø BUFFR2: DS 256
ØØ16Ø ;
ØØ17Ø BEGIN:  DI
ØØ18Ø         LD      (STACK),SP       ;save SP at entry
ØØ19Ø         PUSH    HL               ;Save ptr to CMD buffer
ØØ2ØØ         LD      HL,Ø
ØØ21Ø         LD      A,1Ø3            ;Disable break vectoring
ØØ22Ø         RST     4Ø
ØØ23Ø         EI
ØØ24Ø         LD      HL,Ø             ;get HIGH$
ØØ25Ø         LD      B,L              ;B=Ø
ØØ26Ø         LD      A,1ØØ
ØØ27Ø         RST     4Ø
ØØ28Ø         LD      (MYMEM),HL       ;store for later
ØØ29Ø         LD      A,1Ø1            ;set up IY
ØØ3ØØ         RST     4Ø               ;pointing to flag table
ØØ31Ø         PUSH    IY               ;trans to DE
ØØ32Ø         POP     DE
```

  SYSTEM(SLOW) V2N1P58
Limited backup policy explained - Bill Schroeder V2N3P04 V2N4P06
"Linking to LDOS in Assembly" - Roy Soltoff V1N1P12
"LISP Implementations for the Z-80" - Lee Rice & Daniel Lofy V1N6P26
Listing utility for TBA V2N3P19
LScript made easy V2N2P22
"LScript Patches to Add Versatility" - Scott Loomer V1N6P45 V2N1P47
Load module structure V1N4P42
Lower case lock V1N4P11
LSI Journal submission and subscription policies V2N4P02
MAX-80 LDOS described V2N1P50
"MAX-80 Memory Map" - Chuck Jensen V2N4P58
Magazines V2N3P35
Manual story, with corrections to 1st edition V1N1P16
Memory map V2N1P34
Minimum configuration disk V1N2P29
"Mixing Newscript, Electric Webster, LDOS and Sole" - Jerry Latham V1N6P55
"My BASIC Answer" - E. Cheatham (TBA review) V2N3P17
"Newscript 7.0 and REFLEX" - Gordon Thompson V2N1P28
"Newscript and The BASIC Answer" - Jerry Latham V2N2P20
"New Version - EDAS IV, A" - Marc Leager V1N6P09
Number base conversion V2N4P40
Old DAMs (see also Data Address Marks) V1N5P19
Parity errors V1N1P04
"*** PARITY = ODD ***" - Tim Daneliuk:
  introduction, DATAENTR 200 & ISAM 200 V1N4P17
  tips for better programming, DISCATER, Filter Disk V1N5P11
  BASF drives, drive poll, TAS, MODEM80, HEXSPELL II, Utilities, HELP V1N6P59
  Tandy, MAX-80, AEROCOMP, Electric Webster, The BASIC Answer V2N1P42
  disk drive poll, gold plugs, LX-80, Proofreader, Scripsit Dict. V2N2P27
  magazines V2N3P35
  rumors, LX-80 software compatibility, TRSDOS 6.0 software V2N4P31
"Partitioned Data Sets, MISOSYS Announces" - Roy Soltoff V1N3P23
"Pascal 80, LDOS and" - D. Hill V2N1P22
Passwords for LDOS 5.1.x V1N1P03
Patch, how to V2N1P52
"PDS - Standard and Other Types of Uses" - Scott Loomer V2N1P24 V2N3P45
"Performing Date Conversions in BASIC" - Dick Konop V2N4P60
Printers V2N1P39
Profile III Plus with LDOS V2N4P55
Quarterly reader survey results V2N2P06
Radio Shack V2N1P42 V2N3P03
"Relocating Code for LBASIC USR Routines" - Chuck Jensen V1N3P47
Reviews:
  Alternate Source (TAS) - Tim Daneliuk V1N6P60
  APL*PLUS/80 (STSC) - Daniel Lofy & Lee Rice V2N1P09
  CHROMAtrs (South Shore Computer Concepts) - Scott Loomer V2N3P28
  DATAENTR200 & ISAM200 (Johnson Associates) - Tim Daneliuk V1N4P18
  EDAS IV (MISOSYS) - Marc Leager V1N6P09
  Electric Webster (Cornucopia) - Tim Daneliuk V2N1P45
  HELP (MISOSYS) - Tim Daneliuk V1N6P63
  HEXSPELL II (Hexagon) - Tim Daneliuk V1N6P61
  LDOS Utilities (Powersoft) - Tim Daneliuk V1N6P62
  LISP (Supersoft) - Lee Rice & Daniel Lofy V1N6P29
  MAX-80 (Lobo) - Tim Daneliuk V2N1P44
  Microcomputer Math book (SAMS) - Earle Robinson V1N5P10
  MODEM80 (LSI) - Tim Daneliuk V1N6P60
  Pascal (Alcor) - Scott Loomer V2N1P18
  Pascal-80 (New Classics) - D. Hill V2N1P22
  PDS (MISOSYS) - Scott Loomer V2N1P24
  Printers - Earle Robinson V2N1P39
  Profile III+ (Radio Shack) - Sam Goldberg V2N3P25
  Proofreader (Aspen Software) - Tim Daneliuk V2N2P30
  Scripsit Dictionary (Radio Shack) - Tim Daneliuk V2N2P30
  Structured BASIC Translator (Acorn) - Sue Ratkowski V1N4P21
  The BASIC Answer (LSI) - E. Cheatham V2N3P17
  The BASIC Answer (LSI) - Tim Daneliuk V2N1P46
  UOLISP (Far West) - Lee Rice & Daniel Lofy V1N6P30
Routing a device V1N3P33
"Roy's Technical Corner" - Roy Soltoff:
  load module structure V1N4P42
  task processor V1N5P20 V1N6P76
  error handling during byte I/O, @CKDRV, KFLAG$, @ICNFG, @KITSK V1N6P76
  LDOS 6.0 V2N2P46
RS-232 drivers V1N6P83
"Running 8-Inch Drives on the Model III Under LDOS" - Peter Simon V2N3P08
Sector I/O V2N2P55
"SOLEFIX - Fix that GAT Error" - Erik Ruf V2N2P14 V2N4P38
Speedup kits V1N1P13
Sysgen and type-ahead V1N1P04
System routines:
  @ADTSK, @RMTSK, @KLTSK, @RPTSK - task proc. V1N5P31 V1N6P76 V2N4P19
  @CKDRV - determining 5.1.2 vs. 5.1.3 for @CKDRV adjustment V1N6P78
  @CKDRV - moved V1N5P06
  @CMNDI - command interpreter V1N1P12
  @CTL, @GET, @PUT - byte I/O V1N3P38
  DAY$ - day of the week V1N4P11
  @GET and @PUT under LDOS 6.0 V2N3P56
  @ICNFG - configuration interfacing V1N6P81
  @KITSK - keyboard task V1N6P82
  KFLAG$ - keyboard scanner V1N6P78
  "@PARAM, Using" - Roy Soltoff V1N2P31
  @PARAM under 6.0 V2N4P61
  "@PARAM, @DSPLY, @EXIT and INBUF$ for Everyone" - David Vinzant V2N2P23
  @RAMDIR documentation correction V2N2P39
  SVC's and Fortran V2N4P19
Tandy V2N1P42 V2N3P03
Task processor V1N5P20 V1N6P76
"T-Timer, LDOS Supports the" - Roy Soltoff V1N4P12 V1N5P18 V1N6P66
TRSDOS (Mod I) to LDOS (Mod III) transfer without REPAIR V1N3P03
TRSDOS (LDOS) 6.0 (see also LDOS 6.x) V2N2P46
Update policy explained - Bill Schroeder V1N2P02 V2N4P04
Upper case lock V1N4P11
Users group directory V2N1P06
Utilities:
  CONV V2N2P44
  COPY238 V2N2P44
  PATCH V2N1P52
  REPAIR V2N2P44
Version number explanation V1N2P02
VisiCalc made easy V2N3P23
"VisiCalc with LDOS, Using" - Roy Soltoff V1N2P20

Much Needed Software from PowerSOFT!

For Users of LDOS™ and TRSDOS 6.x™



WANTED
TRSDOS 6.X™ or LDOS™ Users

Our ToolBox for LDOS™ was rated FOUR STARS in the Oct. '83 80-MICRO as a package to "perfect the use of LDOS". We now have it for the Mod 4 as well! These new disk utilities are intended for TRSDOS 6.x or LDOS 5.1.3/4 only! Unlike SU+, these will work on any combination of single/double sided drives and hard drives! They will work with Mod 4, Mod I or III, single or double density, ALL media formats, including 8 in. drives on a MAX-80. Also works with the Radio Shack, or any hard drive configured for use with LDOS or TRSDOS 6.x.

Utilities include disk repair, disk check, extensive zapper for disk, memory, or files, verifiers, comparers, filters, mapping, password manipulation, password removing, mass file moving, mass file killing, erasing, exercising, search/replace, etc. Each utility also contains a built-in "HELP" command, in case you get lost. All utilities are machine language, contain excellent documentation, and disk is unprotected.

MODEL 4 ToolBelt™ (for TRSDOS 6.x only)	**$49.95**
The ToolBox for LDOS™ (MOD I/III)	**$69.95**

WANTED
Your BASIC
Impakt!

DO YOU USE DISK BASIC? WHAT DOS? TRSDOS? LDOS?

How'd you like with a single command from BASIC to: search and/or replace a particular keyword or string, remove remarks, remove extraneous spaces, remove extraneous colons, remove GOTOs following THENs, compress a program, decompress a program, renumber, trace a program without destroying the screen display either by line number or by step, set breakpoints in trace mode, join lines, copy lines from one part of the program to another with automatic renumber, move blocks of lines, check lines for validity, display memory usage statistics for a given program, decode packed string graphics, rescue programs lost by an inadvertant reboot or NEW, automatically save and load programs from disk, automatically convert upper-case strings to lower-case, and more! Whew! Get all those fancy trick features in the other DOSes, plus a LOT more, and use your existing DOS!

What Super Utility Plus is to disk utilities, IMPAKT! is to BASIC! This new self relocating machine-language module provides numerous extended functions for BASIC programmers. Unlike other BASIC extensions, it does not require special formats, BASIC commands, or any modification to BASIC/CMD or LBASIC/CMD, therefore programs written under IMPAKT! will run under regular BASIC. If you program heavily in BASIC you will appreciate the power that IMPAKT! gives you, plus the tremendous time it will save.

IMPAKT! You'll wonder how you ever got along without it! Compatible with most current TRS-80 compatible DOSes. A definite BOOST to TRSDOS Mod I or III and LDOS in particular. Beef up those BASICS! (Not TRSDOS 2.7DD or 6.x compatible.)

IMPAKT! on disk
$39.95 with complete users manual.

US/Canada Please add $2.50 Shipping/Handling-Foreign $10. The above programs cannot be explained in this ad space. Please write for complete catalog with full details.

MOST WANTED LIST

SUPER UTILITY PLUS 3.2

"THE JAWS OF LIFE" rated FIVE STARS in the Oct. '83 80-MICRO! A MUST-HAVE program if you have disks, as reviewed by all the TRS-80 magazines. They all agree that SU+ is a standard in the industry. This new version is the ULTIMATE and features better documentation. Includes directions for use on the Mod 4.

$79.95

PowerMAIL PLUS

The most powerful mailing/information system yet for the TRS-80. Runs under most current DOSes as well as hard drives. 100% machine language for maximum speed! Features 24 definable "flags", 10 level sort, mounting new data disks, and machine compatible data files. It also "converts" many other mailing programs' data to PowerMAIL+ format. Eliminates re-typing!

Only $150 each

Please specify:
MOD I/III
MOD 4 TRUE Mod 4 Mode!
MOD II/12/16 (Comes on DP-II Kernel System)

POWERDOT II

A GRAPHICS BREAKTHROUGH! The ULTIMATE in graphics design. This version is MUCH MORE POWERFUL than previous versions and includes BETTER documentation as well. New features include AUTODRAW™ and CIRCLE commands. Now allows you to design your own character sets! Includes lots of examples on disk. Your screen is only a "picture window" to a much larger drawing area! You are only limited by disk storage...not memory! Works on EPSON Series (Graftrax or Graftrax+ required) or the C.Itoh 8510 (PROWRITER or NEC version). Coming soon for DMP-2100, DMP-400, and LP-8.

$59.95

POWERDRAW

A full screen graphics/text editor. Allows you to design your graphics and merge them with your BASIC or assembly program in SIX different formats. The files may be used alone or chained together for animation effects. MANY programs being produced commercially today incorporate POWERDRAW graphics! No royalties to pay, and they're easy to do! Has received EXCELLENT reviews in many magazines.

$39.95

POWERDRIVERs for use with SuperScripsit™

Allows you to use ALL of SuperScripsit's™ features with your EPSON, PROWRITER, or F-10 STARWRITER printers! Includes patches for running SuperScripsit on LDOS with floppy or hard drive. Now includes DISKFILE driver. Save formatted text to disk now!

Only $29.95 each

Please Specify:
PowerDRIVER/P - C.Itoh 8510 PROWRITER
PowerDRIVER/E - EPSON or GEMINI Series
PowerDRIVER/F - C.Itoh STARWRITER (Leading Edge)

POWERSOFT
PRODUCTS FROM BREEZE/QSD, INC.

11500 Stemmons Fwy.
Suite 125
Dallas, Texas 75229
(214) 484-2976

✓306

SuperScripsit is a Trademark of Tandy Corp.
LDOS is a trademark of LSI

Mention the LSI Quarterly in your order and deduct 15% from prices!

LOGICAL
SYSTEMS
INC.

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

BULK RATE
U.S. POSTAGE PAID
Permit No. 36
Mequon, WI 53092

In This issue:

- The 'C' Language — Part 6
- Fast Machine — Language
- File I/O
- An Introduction to COMM and LCOMM

POSTMASTER: DATED MATERIAL DO NOT DELAY